Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Compositional Generalization in Transformers

Masterthesis
at Research Group Knowledge Technology, WTM
Prof. Dr. Stefan Wermter

Department of Informatics
MIN-Faculty
Universität Hamburg

submitted by
Imran Ibrahimli
Course of study: Intelligent Adaptive Systems
Matrikelnr.: 7486484
on
24.10.2024

Examiners:	Prof. Dr. Stefan Wermter
	Dr. Jae Hee Lee

Abstract

Your English abstract here (mandatory if written in English and recommended otherwise).

Zusammenfassung

Hier die deutsche Zusammenfassung einfügen (notwendig).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Robust length generalization in sequence modeling tasks remains a significant challenge in deep learning, particularly for algorithmic problems that require precise manipulation of sequential data. Integer addition serves as a canonical example of such a task, where the ability to add numbers of arbitrary length is trivial for humans but non-trivial for neural networks. The transformer architecture Vaswani et al. 2017, known for its success in natural language processing, computer vision, and other domains, often fails to learn generalizable algorithms that also correctly process sequences longer than those encountered during training. This limitation highlights a fundamental gap in our understanding of how neural networks learn and represent algorithmic processes. Understanding the mechanisms underlying this failure is essential and has broader implications for developing neural networks capable of robust algorithmic reasoning in various domains.

Large language models (LLMs) exhibit impressive emergent capabilites and state-of-the-art performance on many benchmarks, but nonetheless struggle with algorithmic tasks that require compositional reasoning and precise manipulation of structured data. Even small transformer models trained from scratch on a specific task share this failure mode. It is hypothesized that the limitation stems from models struggling to effectively use *position-based addressing* to focus on structure of the sequence, in contrast to their strength in *content-based addressing* as seen in natural language processing tasks (Ebrahimi, Panchal, and Memisevic 2024). In line with this, recent studies have pointed towards positional encoding as a key factor influencing length generalization in transformers McLeish et al. 2024; Y. Zhou et al. 2024. Alternative positional encoding methods, such as the Abacus positional encoding McLeish et al. 2024, have shown improved generalization but often involve task-specific modifications that may not extend to other algorithmic problems. This raises questions about the universality and flexibility of currently used positional encoding schemes and especially their impact on algorithmic task performance. Furthermore, despite many research efforts directed towards novel architectures and positional encoding methods, the underlying reasons for the failure of standard transformers in algorithmic tasks remain underexplored in the literature.

1.2 Problem Statement

Despite the widespread success of transformer models in various domains, their ability to perform multi-digit integer addition with length generalization remains limited. Standard absolute positional encodings, integral to the transformer architecture, fail to provide the necessary alignment cues for correctly matching digits by their place value.

The core issue lies in the transformer's difficulty in aligning digits of different operands based on their positional significance across different sequence lengths. In multi-digit addition, each digit must be correctly matched with its corresponding digit in the other operand, and the positional encoding must facilitate this alignment. Without correct position-based retrieval of the correct digit from the sequence, the model cannot learn the necessary compositional structure to generalize addition to longer sequences.

This thesis seeks to investigate whether transformers can achieve length generalization in integer addition without resorting to task-specific positional encodings or architectural modifications. The goal is to understand the limitations of standard transformer models and identify principles that enable better generalization by exploring various positional encoding schemes and training datasets, while minimizing the architectural changes to standard unsupervised language modeling with transformers.

1.3 Research Questions

The primary goal of this thesis is to investigate the reasons behind the failure of transformer models to generalize integer addition to longer sequences and to understand how different positional encoding schemes and data formatting strategies impact this ability. Specifically, the following research questions are addressed:

- **RQ1: Why do transformer models with standard absolute positional encodings fail to generalize integer addition to sequences longer than those seen during training?**

  Focus is to analyze the limitations of absolute positional encodings in facilitating digit alignment and carry propagation over longer sequences. Understanding the fundamental reasons for this failure might inform the development of more flexible positional encoding schemes.

- **RQ2: How does the inclusion of sub-task data (carry detection, digit-wise modular addition, reversing, digit alignment) influence the model's compositionality and length generalization capabilities?**

Incorporating sub-task data may help the model learn the underlying algorithmic components of addition. This work explores whether training on sub-tasks enables the model to compose these functions and generalize to longer sequences.

- **RQ3: How can mechanistic interpretability techniques be applied to understand the internal representations and failure modes of transformer models in the context of integer addition?**

  Mechanistic interpretability methods allow to analyze the learned representations and identify mechanisms that lead to success or failure in length generalization.

Through exploring these questions, the goal is to gain insights into the limitations of current transformer architecture and positional encoding methods, and to identify principles that could boost length generalization in algorithmic tasks. This thesis focuses on maintaining the standard transformer architecture without introducing task-specific modifications, seeking solutions that are generalizable and applicable to a broader range of problems.

1.4 Thesis Structure

The remainder of this thesis is organized as follows:

- **Chapter 2** provides background information on transformer model, positional encoding schemes, and mechanistic interpretability.
- **Chapter 3** reviews related work domains of integer addition, length generalization, and reasoning in transformers.
- **Chapter 4** describes the experimental approach, including data formatting strategies and training setups used in this research.
- **Chapter 5** summarizes the findings of the thesis, discusses the implications of the results, and outlines directions for future research.
- **Appendices** include additional experimental results, technical details, and supplementary material relevant to the thesis.

Chapter 2

Background

This chapter provides an overview of the Transformer architecture, focusing on its core components, different variants, positional encoding schemes, and the processes of training and inference. It also discusses mechanistic interpretability and the deep double descent phenomenon.

Notation

$\mathcal{V}$	Vocabulary (set) of input tokens.
n	Length of the input sequence.
b	Batch size for batched input sequences.
$\mathbf{x}$	Input sequence of tokens, $\mathbf{x} \in \mathcal{V}^n$
x_i	i-th token in the input sequence, $x_i \in \mathcal{V}$.
d	Dimensionality of the token embedding space.
E_i	d-dimensional embedding vector of token x_i, $E_i \in \mathbb{R}^d$.
E	Embedding matrix, $E \in \mathbb{R}^{\lvert\mathcal{V}\rvert \times d}$.
H	Matrix of a sequence of embedding vectors, $H \in \mathbb{R}^{n \times d}$.
H_i	i-th vector in the sequence of embeddings, $H_i \in \mathbb{R}^d$.
O	Output matrix from a Transformer layer, $O \in \mathbb{R}^{n \times d}$.
Q	Queries matrix, $Q \in \mathbb{R}^{n \times d}$.
K	Keys matrix, $K \in \mathbb{R}^{n \times d}$.
V	Values matrix, $V \in \mathbb{R}^{n \times d}$.
θ	Model parameters (weights and biases).

2.1 Transformer

The Transformer architecture, introduced by Vaswani et al. (2017), performs sequence modeling by relying entirely on self-attention mechanism, instead of using convolution or recurrence.

Let $\mathcal{V}$ denote the vocabulary of input tokens. While the tokens can represent anything, in language modeling tasks they are usually learned subword units. In this work, however, a simple character tokenization scheme is used that is suitable for algorithmic tasks, so each token corresponds to a single character (letter, digit or symbol) in the sequence. An input sequence is represented as $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, where $x_i \in \mathcal{V}$ and n is the sequence length. Each token x_i is mapped to a d-dimensional embedding vector $E_i \in \mathbb{R}^d$ using an embedding matrix $E \in \mathbb{R}^{|\mathcal{V}| \times d}$. Thus, each row of E corresponds to the embedding of a token in the vocabulary.

The input matrix to a Transformer layer is a sequence of vector embeddings (also called the *latent* or *hidden representation* for intermediate layer inputs), denoted $H \in \mathbb{R}^{n \times d}$, where $H = [H_1^\top, H_2^\top, \ldots, H_n^\top]^\top$. The output of a Transformer layer is also a sequence of vectors with the same sequence length, denoted $O \in \mathbb{R}^{n \times d}$.

In practice, the inputs to the Transformer are batched, so the input has an additional dimension for the batch size, denoted b, with $H \in \mathbb{R}^{b \times n \times d}$. This results in the first (batch) dimension being added throughout the intermediate representation and the output, but has no bearing on the description of the Transformer model.

2.1.1 Elements

The Transformer is composed of several key components to model dependencies in sequential data: multi-head attention, feed-forward networks, layer normalization, and residual connections. Informally, the attention mechanism transfers information *between* tokens, while the feed-forward networks process information *within* tokens. These components are stacked in each layer of the Transformer as shown in Figure 2.1 (a).

Token Embeddings The discrete input tokens are first converted into continuous embeddings using an embedding matrix $E \in \mathbb{R}^{|\mathcal{V}| \times d}$, where $|\mathcal{V}|$ is the size of the vocabulary and d is the embedding dimension. The embedding matrix is learned during training, and the embeddings are used as input to the Transformer. After applying the Transformer layers, the output embeddings are passed through a linear *unembedding* layer to get the *logits* (unnormalized log-probabilities) for the next token.

Attention Mechanism The attention mechanism (Bahdanau, Cho, and Bengio 2014) allows the model to weigh the relevance of different positions in the input sequence. For this purpose, it computes *queries*, *keys*, and *values* from the input embeddings and uses them to calculate attention scores. The output is a weighted sum of the values, where the weights are determined by the attention scores. The names “queries”, “keys”, and “values” are derived from the context of information retrieval, where the queries are the elements being searched for, the keys are the elements being searched, and the values are the elements being retrieved. In the context of the Transformer, intuitively, the query represents what the current token is “looking for” in the sequence, the keys represent what the token at a given position “offers” to the current token, and the values are the actual information that the current token “receives” from the other tokens.

First, the queries, keys, and values are computed as:

$$Q = HW^Q,$$
$$K = HW^K,$$
$$V = HW^V,$$

where $W^Q, W^K, W^V \in \mathbb{R}^{d \times d_k}$ are the weight matrices, and d_k is the dimension of the queries, keys, and values. In practice, the convention is to set $d_k = d$, which is the case for the models in this work. Thus, $Q, K, V \in \mathbb{R}^{n \times d}$.

Given queries Q, keys K, and values V, the attention output is computed as:

$$O_{att} = \text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V.$$

where the softmax function is applied along the last dimension, and is defined as:

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}}.$$

Note that the output of the attention mechanism has the same shape as the input, $O_{att} \in \mathbb{R}^{n \times d}$.

Multi-Head Attention Multi-head attention extends the attention mechanism with multiple independent *heads* to allow the model to focus on information from different representation subspaces. So, instead of applying attention to the d-dimensional queries, keys, and values directly, they are projected into h different d_{head}-dimensional subspaces, where h is the number of heads. In this work, the head dimension d_{head} is set to d/h, so that the total dimensionality remains d. The outputs the heads are concatenated and linearly transformed to the original dimensionality:

$$\text{MultiHeadAttention}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O,$$

where $W^O \in \mathbb{R}^{d \times d}$ is the learned output weight matrix, and each head is computed as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V),$$

and $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d \times d_{head}}$ are the weight matrices for the i-th head.

Feed-Forward Networks Position-wise feed-forward networks (FFN), also called the Multi-layer Perceptron (MLP), are applied independently to each position in the sequence:

$$\text{FFN}(H_i) = \sigma(H_i W_1 + \mathbf{b}_1)W_2 + \mathbf{b}_2,$$

where $H_i \in \mathbb{R}^d$ is the input vector, $W_1 \in \mathbb{R}^{d \times d_{\text{ff}}}$, $W_2 \in \mathbb{R}^{d_{\text{ff}} \times d}$, and σ is an activation function such as Rectified Linear Unit (ReLU). The intermediate dimensionality d_{ff} is usually set to $4d$ in the original Transformer model and in all experiments presented in this work.

Layer Normalization Layer normalization (Ba, Kiros, and Hinton 2016) is applied after each sub-layer over the last (feature) dimension. The LayerNorm function for a vector $v \in \mathbb{R}^d$ is defined as:

$$\text{LayerNorm}(v) = \frac{v - \mu}{\sigma}\gamma + \beta,$$

where the scale γ and bias vector β are learned scaling and shifting parameters, and μ and σ are the mean and standard deviation of v,computed as follows:

$$\mu = \frac{1}{d}\sum_{i=1}^{d} v_i,$$
$$\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(v_i - \mu)^2}.$$

Residual Connections Residual connections (He et al. 2016) are usually applied to ease gradient flow and enable the training of deeper networks. In Transformer models, the residual connections are applied after each sub-layer (self-attention and MLP), followed by a layer normalization. Thus, the output of each sub-layer is:

$$\text{SubLayerOutput} = \text{LayerNorm}(\mathbf{x} + \text{SubLayer}(\mathbf{x})).$$

The residual connections-based view of the model also enables the concept of a *residual stream*, which is important in mechanistic interpretability. In this alternative view of the model, the residual connections are the main backbone of information flow through the model, with the sub-layers processing the hidden representation tensor H and adding it back to the residual stream.

Block Structure The original Transformer introduced in Vaswani et al. 2017 consists of stacked encoder and decoder blocks, each containing multi-head attention and feed-forward networks, along with residual connections and layer normalization. However, different modified architectures have been proposed, such as the encoder-only BERT model (Devlin et al. 2019) and the decoder-only GPT model (Radford and Narasimhan 2018).

2.1.2 Encoder and Decoder Architectures

In this section, different Transformer architectures are summarized: encoder-decoder, encoder-only, and decoder-only models. The original Transformer (Vaswani et al. 2017) employs an encoder-decoder structure, where the encoder transforms input sequences into continuous representations, and the decoder generates output sequences based on these representations and previously generated tokens. Encoder-only models like BERT (Devlin et al. 2019) focus solely on encoding input sequences into contextual embeddings, making them well-suited for understanding tasks such

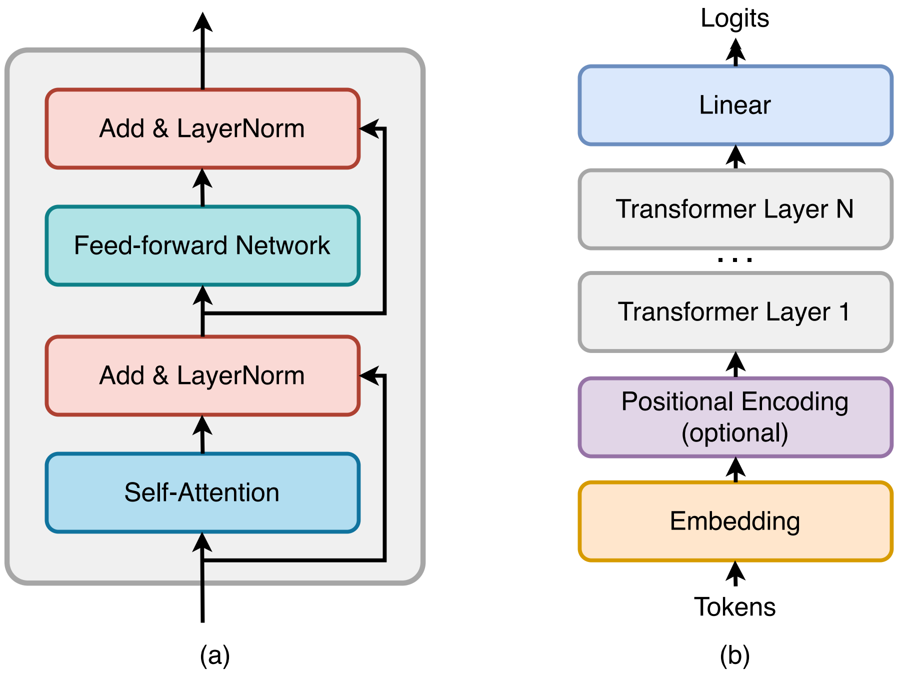


Figure 2.1: (a) A single Transformer layer, consisting of multi-head self-attention, feed-forward network (MLP), and layer normalization. (b) A decoder-only transformer model. In the decoder, the self-attention mechanism has a causal mask to prevent attending to future tokens.

as text classification and question answering. Decoder-only models, like GPT (Radford and Narasimhan 2018), generate sequences by predicting the next token based on prior tokens, primarily used for text generation. While both encoder-decoder and decoder-only architectures can perform autoregressive sequence generation, decoder-only models are the focus of this work.

Encoder-Decoder The original Transformer model introduced by Vaswani et al. Vaswani et al. 2017 employs an encoder-decoder architecture. In this architecture, the encoder processes an input sequence $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ into a sequence of continuous representations $\mathbf{z} = (z_1, z_2, \ldots, z_n)$. The decoder then generates an output sequence $\mathbf{y} = (y_1, y_2, \ldots, y_m)$ by predicting the next token y_t based on the encoder's output and the previously generated tokens.

The encoder consists of a stack of N identical layers, each containing two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. The decoder has a similar structure but includes a third sub-layer: the *cross-attention*, also called *encoder-decoder attention*, where the queries come from the previous decoder layer, and the keys and values come from the output of the encoder. Hence, the difference between the self-attention and cross-attention mechanisms is that self-attention is usually applied to the same sequence, while cross-attention is applied between two different sequences (e.g. one from the encoder, and one from the decoder). Moreover, the self-attention mechanism in the decoder has a causal mask to prevent attending to "future tokens", ensuring output is generated autoregressively. The T5 model (Raffel et al. 2020) is an example of a large encoder-decoder Transformer model capable of performing many NLP tasks such as translation, summarization, and question answering.

Encoder-only Encoder-only models focus exclusively on encoding the input sequence into a contextual representation without a decoder component. BERT (Bidirectional Encoder Representations from Transformers) Devlin et al. 2019 is a prominent example of this architecture. BERT utilizes a stack of Transformer encoder layers to produce deep bidirectional representations by jointly conditioning on both left and right context. This makes encoder-only models particularly well-suited for tasks that require a comprehensive understanding of the input, such as text classification, named entity recognition, and question answering.

These models are typically pre-trained on large unlabeled text corpora using self-supervised objectives like masked language modeling (MLM) and next sentence prediction (NSP). The pre-trained models can then be fine-tuned on specific downstream tasks.

Decoder-only Decoder-only models generate sequences based on prior tokens and are designed primarily for autoregressive language modeling and text generation tasks. GPT (Generative Pre-trained Transformer) Radford and Narasimhan 2018 is a canonical example of a decoder-only architecture. In these models, the Transformer decoder predicts the next token in a sequence by attending to the previous tokens without an encoder component. The encoder is not needed since in a decoder-only Transformer, the input sequence $\mathbf{x}$ is prepended to the decoder input sequence $\mathbf{y}$, and only passed through the decoder layers to autoregressively generate the output sequence. Recent research on language modeling has mostly focused decoder-only models, since they can also be used on other language tasks through prompting, few-shot learning, and fine-tuning. In particular, majority of the latest state-of-the-art large language models (LLMs) are decoder-only transformers pre-trained on large text corpora.

Encoder-Decoder vs. Decoder-Only Both model types are capable of autoregressive sequence generation and can be used for a wide range of tasks. The decoder-only models are favored in recent works due to them being simpler and having less inductive bias. However, encoder-decoder models are still widely used in machine translation, robotics, and multi-modal learning tasks. The additional structure in encoder-decoder models, as compared to using a decoder-only model with would-be encoder input sequence prepended to the decoder's input can be summarized as:

- The input to the encoder passes through more layers (encoder layers) before reaching the decoder.

- It is assumed that input and output sequences are sufficiently different to justify using separate parameters for them (encoder and decoder).

With large language models and massive datasets, the difference between the two architecture becomes less relevant. In summary, the choice of Transformer architecture depends on the specific requirements of the task, though decoder-only models have been more widely used in recent research and are the focus of this work.

2.1.3 Recurrent and Looping Architectures

Multiple Transformer extensions have been proposed that incorporate iterative application of weight-sharing layers, particularly suitable for algorithmic tasks that require reasoning over sequences. The Universal Transformer and Looped Transformer are two such examples that introduce recurrence into the Transformer architecture. There are other modifications the Transformer architecture, a few examples of which include the Memory Transformer (M. S. Burtsev et al. 2021), Recurrent Memory Transformer (Bulatov, Kuratov, and M. Burtsev 2022), and Neural Data Router (Csordás, Irie, and Schmidhuber 2021), but these architectures are not widely adopted in pretrained language models, and are not tested in this work.

Universal Transformer The Universal Transformer (Dehghani et al. 2018) introduces recurrence into the Transformer architecture by repeatedly applying the same transformer layers multiple times, in both encoder and decoder parts:

$$H^{(t+1)} = \text{TransformerLayer}(H^{(t)}),$$

where t denotes the iteration step. An adaptive computation time (ACT) mechanism is used to dynamically adjust the number of iterations based on the input sequence. The Universal Transformer has been shown to achieve better performance on algorithmic tasks compared to the original Transformer. It is also an interesting extension of the Transformer architecture from theoretical point of view, since it makes the model Turing-complete under certain conditions (Dehghani et al. 2018).

Looped Transformer Similar to the Universal Transformer, the Looped Transformer (Yang et al. 2023) extends the Transformer by incorporating iterative application of a block of transformer layers. This modification has been shown to perform better than the original, non-recurrent Transformer on algorithmic tasks (Csordás 2023; Yang et al. 2023). In particular, looped transformer achieves better length generalization on the binary addition task as shown by Fan et al. 2024. The looped decoder-only transformer architecture is illustrated in Figure 2.2. However, unlike the Universal Transformer, the Looped Transformer is not necessarily encoder-decoder, and might not use the adaptive computation time mechanism.

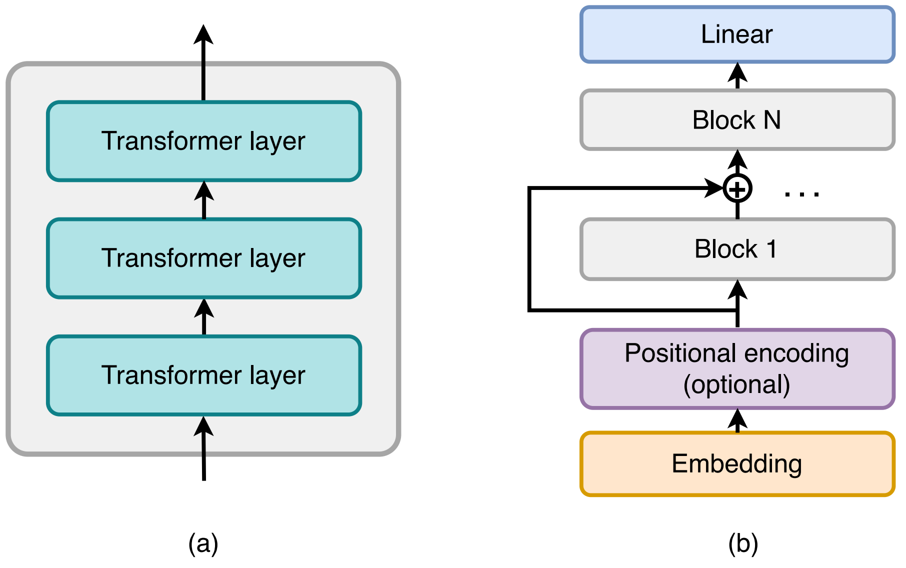


Figure 2.2: (a) A block of multiple transformer layers used in the looped transformer. (b) Looped decoder-only transformer architecture. The input injection mechanism adds skip connections (arrow around blocks) from the original input sequence to the input of each block.

2.1.4 Positional Encoding Schemes

Since the Transformer lacks inherent sequential order, positional encodings are added to input embeddings to provide position information. Though, several recent works have shown that the causal attention mechanism in a decoder-only transformer can also implicitly learn to encode positional information in the absence of explicit positional encodings (Haviv et al. 2022; Zuo and Guerzhoy 2024; Y. Zhou et al. 2024). Positional encoding methods are also crucial for algorithmic tasks, especially multi-digit integer addition (Shen et al. 2023; Kazemnejad et al. 2023; Ruoss et al. 2023)

Absolute Positional Encoding The original Transformer (Vaswani et al. 2017) uses additive vectors of same dimensionality as the embeddings to encode the *absolute* positions of the tokens. These vectors could be learned (from a random initialization), or *sinusoidal*, where the latter are defined as:

$$\begin{aligned}\mathbf{p}_{i,2k} &= \sin\left(\frac{i}{10000^{2k/d}}\right),\\ \mathbf{p}_{i,2k+1} &= \cos\left(\frac{i}{10000^{2k/d}}\right),\end{aligned}$$

for position in the sequence i and dimension k. In Vaswani et al. 2017, the performance difference between learned and sinusoidal positional encodings was found to be insignificant, and the use of sinusoidal encodings is justified by possibility of generalization to longer sequences. In practice, however, absolute positional encodings do not generalize well to sequences longer than the ones seen during training (Press, Smith, and Lewis 2021).

Randomized Positional Encodings Randomized positional encodings Ruoss et al. 2023 aim to improve length generalization by simulating longer sequences during training. Similarly to absolute positional encodings, they are added to the input embeddings, and have separate vectors for each position in the sequence. However, instead of using sequential positions $1, 2, \ldots, n$, the positions are randomly sampled (keeping order) from a range $[1, n_{\max}]$, where $n_{\max}$ is the maximum sequence length. Thus, the model is exposed to a wider range of positional encodings during training, which helps improve generalization to longer sequences. A related method is to randomly insert spaces between tokens in the input sequence (e.g. `123 + 456` might become `12 3 + 45 6`), which disrupts the model's dependence on absolute position information and encourages it to learn more robust representations (Shen et al. 2023). It is important to distinguish this method from the one introduced by Shen et al. 2023 named Random Embedding, where a random Gaussian "tag" is added to a subset of embedding dimensions.

Relative Position Encoding Relative position representations (Shaw, Uszkoreit, and Vaswani 2018) encode the relative distances between sequence elements directly into the attention mechanism. In RPE, a vector $\mathbf{a}_{i,j} \in \mathbb{R}^d$ is learned for each pair of positions (i, j), and added to the keys before computing the attention scores:

$$A_{ij} = \frac{\mathbf{q}_i(\mathbf{k}_i + \mathbf{a}_{ij}^K)^\top}{\sqrt{d}}$$

where $\mathbf{q}_i$ and $\mathbf{k}_j$ are the query and key vectors for positions i and j, respectively. Relative position encodings have been shown to improve performance on tasks requiring long-range dependencies (Shaw, Uszkoreit, and Vaswani 2018).

Attention with Linear Biases (ALiBi) ALiBi (Press, Smith, and Lewis 2021) introduces a linear bias to the attention scores based on the relative positions. For this additive method, the computation of the (pre-softmax) attention logits is modified as:

$$A_{\text{ALiBi}}(X) = QK^\top + B,$$

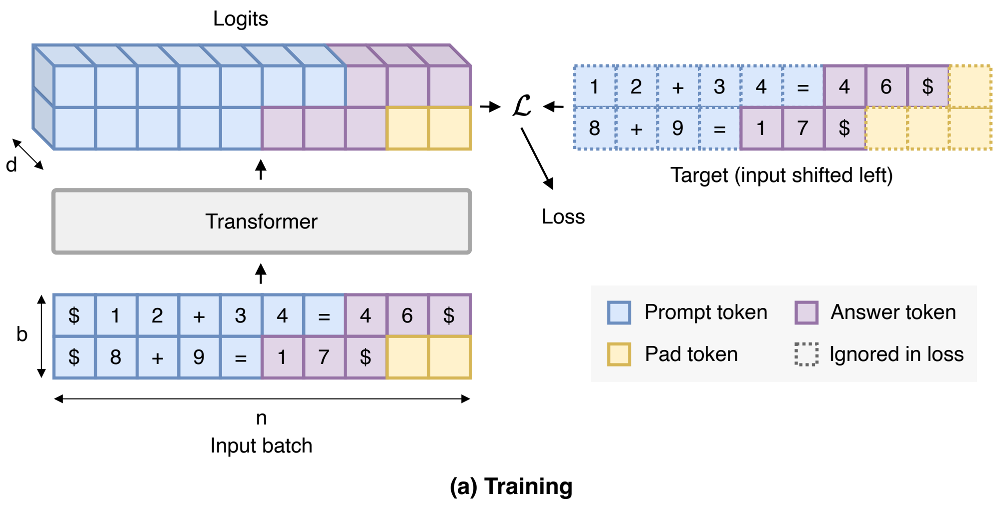
where the bias matrix $B \in \mathbb{R}^{n \times n}$ is computed by the position encoding function $b : \mathbb{N}^2 \to \mathbb{R}$, such that the (i, j)-th entry of B is $b(i, j)$. The bias function for the relative position encoding is defined as:

$$b(i, j) = -r|i - j|$$

where the r is a fixed slope pre-computed for each head.

Rotary Position Encoding (RoPE) RoPE Su et al. 2024 encodes positions using rotations of the query and key vectors with an angle proportional to their absolute positions before the dot product attention, which results in attention being a function of the relative distance between the tokens, capturing the relative positional information. It is a non-additive relative positional encoding. Works such as Press, Smith, and Lewis 2021; Kazemnejad et al. 2023 show that RoPE also has poor length generalization in addition tasks.

2.1.5 Training and Inference



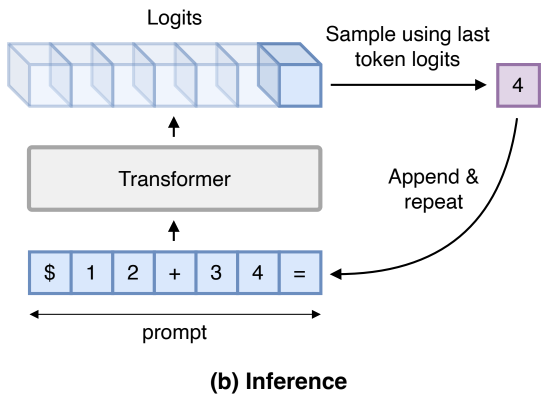


Figure 2.3: Training and inference setup for Transformer models on the arithmetic task. During training (subfigure a), the padded input batch is passed through the model, and the loss is computed by comparing the predicted logits with the target sequence (*teacher forcing*). Unlike regular unsupervised learning, the loss from non-answer tokens is masked out in experiments. During inference (subfigure b), the model generates outputs by greedily sampling tokens one by one until maximum output length is reached, or the end-of-sequence token $ is generated.

Training Training involves minimizing a loss function, typically the *cross-entropy loss* for language tasks, using an optimization algorithm like Stochastic Gradient Descent (SGD) or AdamW (Loshchilov and Hutter 2018). The model learns the parameters θ by backpropagating the loss through the network layers. The cross-entropy loss is defined as:

$$\mathcal{L} = -\frac{1}{b}\sum_{i=1}^{b}\sum_{j=1}^{n}\log p(x_{i,j}|x_{i,<j}),$$

where b is the batch size, n is the sequence length, and $p(x_{i,j}|x_{i,<j})$ is the probability of token $x_{i,j}$ given the previous tokens $x_{i,<j}$.

Answer Loss Masking For tasks like integer addition, the loss is modified to focus only on the answer tokens, where a binary mask $m_{i,j} \in \{0, 1\}$ is applied to denote whether a token at position j in sequence i corresponds to an answer. The loss becomes:

$$\mathcal{L}_{\text{answer-only}} = -\frac{1}{b}\sum_{i=1}^{b}\sum_{j=1}^{n} m_{i,j}\log p(x_{i,j}|x_{i,<j}),$$

where b is the batch size, n is the sequence length, $m_{i,j} = 1$ if token $x_{i,j}$ is an answer token, otherwise $m_{i,j} = 0$, and $p(x_{i,j}|x_{i,<j})$ is the predicted probability of token $x_{i,j}$ given the preceding tokens $x_{i,<j}$.

This formulation ensures that the loss is computed only over the answer tokens, with non-answer tokens effectively ignored (since $m_{i,j} = 0$ for those positions). Without answer loss masking, the cross-entropy loss would penalize the model for incorrectly predicting randomly generated operands as well (which are not possible to predict in the first place), thus adding noise to the training process never reaching 0 loss.

Inference During inference, the trained model generates outputs by iteratively computing the forward pass through the network and adding the sampled tokens to the prompt sequence each time. In autoregressive models, tokens are generated one by one, conditioned on previous outputs. The token at each step is selected using *greedy sampling*, where:

$$x_j = \arg\max_{x\in\mathcal{V}} p_\theta(x|x_{<j}),$$

where $\mathcal{V}$ is the vocabulary, $p_\theta(x|x_{<j})$ are the logits computed by the model, and softmax is applied to obtain the probabilities from non-normalized logits. The input sequence for the next step is updated by appending the predicted token to the previous sequence:

$$x^{t+1} = [x^t, x_j].$$

The sampling process continues until a predefined stopping criterion is met, such as reaching a maximum sequence length or generating a special end-of-sequence token. Greedy sampling is computationally efficient but does not explore alternative sequences. However, it is sufficient for tasks like integer addition, where the output is deterministic.

2.2 Mechanistic Interpretability

Understanding how Transformers make decisions is crucial for interpretability and debugging. Relevant concepts include the *residual stream*, *activation patching*, and the *logit lens*, which provide insights into the model's internal representations and decision-making processes.

Residual Stream The residual stream in a Transformer is enabled by the residual connections around operations (as described in Subsection 2.1.1) and serves as a shared memory, accumulating information across layers via residual connections. Both attention heads and feedforward layers read from and write to this stream, which ensures the propagation of information throughout the model. Since each layer can only read from earlier layers, analyzing this stream is essential for understanding how information is transformed and stored across layers (Elhage et al. 2021). The residual stream can be studied by techniques such as activation patching to trace information flow.

Circuits Circuits in Transformers are a set of elements (i.e. attention heads, feed-forward layers) that are responsible for specific behaviors. Currently, while multiple methods exist that attempt to discover circuits, there is no structured way to identify all circuits of interest, nor is it certain that human-interpretable circuits exist in the first place in any given model (Ferrando et al. 2024). Some works succeeded in identifying specific high-level circuits such as ones performing modular addition (Nanda et al. 2022; Zhong et al. 2023) and indirect object identification (K. R. Wang et al. 2022). Discovering and explaining circuits is helpful for learning algorithmic tasks and compositionality, like multi-digit integer addition. For instance, a number of circuits might emerge in the model that perform the sub-tasks for integer addition (such as digit-wise modular addition, and carry propagation), and the answer might be generated by composing their outputs.

2.3 Expressivity

In the context of Transformers, expressivity refers to whether a given task or behavior can be reliably implemented using the model's learned representations. One framework for analyzing this is *RASP*, a restricted-access sequence processing language (Weiss, Goldberg, and Yahav 2021) that mimics the operations of transformers in a more interpretable manner. Moreover, Fan et al. 2024 show using RASP that looped transformers can generalize to longer sequence lengths for binary addition task, along with other algorithmic tasks.

A key aspect of Transformers is their ability to generalize *compositionally* in some tasks, as highlighted by Hupkes et al. 2020. This suggests that, under the right conditions, transformers are capable of learning systematic combinations of components, enabling them to generalize beyond seen examples in a structured manner. This compositional behavior is crucial for algorithmic tasks, where the model must apply learned rules consistently across different sequence lengths. Despite the theoretical results, there is no clear applied research about how it can be proven that a Transformer model of specific architecture and size can learn a particular task, like multi-digit integer addition.

H. Zhou et al. 2023 investigate the expressivity of Transformers on algorithmic tasks using RASP-L, and demonstrate the implementation of the addition algorithm within RASP-L but note certain limitations in their approach. Specifically, their result relies on the inclusion of index hints embedded within the input sequences to allow induction heads to perform the digit alignment and padding operands to the same digit length, sidestepping the issue of position-based digit alignment studied in this work.

2.4 Deep Double Descent

The phenomenon of *deep double descent* describes how increasing model capacity or training duration can initially lead to worse generalization before ultimately improving it. This was initially believed to challenge the traditional bias-variance tradeoff, where increasing model complexity was believed to always result in overfitting (Belkin et al. 2019; Nakkiran et al. 2021). Deep double descent arises when models, especially overparameterized ones, first memorize the training data, but later discover more generalizable features, leading to improved performance on the test set and zero training loss.

This phenomenon is still relevant in integer addition tasks, since the interpolation threshold — the point where the model transitions from memorization to generalization might change with respect to model and dataset size.

Chapter 3

Related Work

In this chapter, a literature review is presented focusing on the topics of transformers, learning arithmetic tasks, reasoning capabilities, compositional learning, and generalization to longer sequence lengths.

3.1 Learning Arithmetics with Transformers

The ability of transformer models (Vaswani et al. 2017) to learn arithmetic operations such as integer addition has been a subject of significant research interest. Evaluations demonstrated that large pre-trained language models such as GPT-3 (Brown et al. 2020) can exhibit emergent capabilities across general-purpose tasks, including basic few-digit arithmetic, despite these tasks not being explicitly encoded by the unsupervised next-token prediction objective. However, even largest state-of-the-art models like GPT-4 (Achiam et al. 2023) struggle to robustly solve multi-digit addition and multiplication, especially when a larger number of digits are involved.

N. Lee et al. 2023 investigate how even small transformers, trained from random initialization, can efficiently learn arithmetic operations such as addition and multiplication. They show that training on chain-of-thought style data that includes intermediate step results significantly improves accuracy, sample complexity, and convergence speed, even in the absence of pretraining. This approach aligns with the experiments presented in this thesis on chain-of-thought training, where models are trained to output the steps involved in solving addition problems. One limitation of their work is that it limits each task to a fixed number of digits (e.g. 7 and 7 digit operands), using padding to ensure uniform input length. In contrast, this thesis extends the task to variable-length addition problems which is more difficult due to the need for models to learn to position-wise align digits as discussed in Chapter 1.

Understanding how transformers learn arithmetic tasks is further explored by Quirke and Barez 2023, who present an in-depth mechanistic analysis of a one-layer transformer model trained for integer addition. They reveal that the model processes the problem in a non-intuitive way: it divides the task into parallel, digit-specific streams and employs distinct algorithms for different digit positions, merging the results in the MLP layer. This work also restricts the operands to a fixed length of 5 digits and employs padding, apart from restricting the model to a single layer.

Length generalization is a critical challenge in training transformers for arithmetic tasks that comes up in numerous research works. Jelassi et al. 2023 examine how transformers cope with learning basic integer arithmetic and generalizing to longer sequences than seen during training. They find that relative position embeddings enable length generalization for simple tasks such as addition, allowing models trained on 5-digit numbers to perform 15-digit sums. However, this method fails for multiplication, leading them to propose “train set priming” by adding a few (10 to 50) longer sequences to the training set. Despite showing interesting capabilites of learning from few examples, this defeats the purpose of *zero-shot* generalization to unseen lengths.

Similarly, Duan and Shi 2023 investigate the capabilities of transformers in learning arithmetic algorithms and introduce Attention Bias Calibration (ABC), a calibration stage that enables the model to automatically learn proper attention biases linked to relative position encoding mechanisms. Using ABC, they achieve robust length generalization on certain arithmetic tasks. Despite promising results, this work is limited due to the attention bias intervention being task-specific and the need for a modified training with 2 stages (first training to perfect interpolation accuracy to learn the attention biases, then training another model with extracted attention biases). Conversely, the main research interest in this domain lies in architectural modifications that apart from boosting algorithmic capabilities, also preserve or improve performance on other language tasks.

Recent work by McLeish et al. 2024 addresses the poor performance of transformers on arithmetic tasks by adding an embedding to each digit that encodes its position relative to the start of the number. This modification, along with architectural changes like input injection and recurrent layers, significantly improves performance, achieving up to 99% accuracy on 100-digit addition problems. As of now, this work represents the state-of-the-art in length generalization on integer addition, with test sequence lengths up to 6 times longer training sequence lengths (compared to previous SOTA of 2.5 times by Y. Zhou et al. 2024).

Investigations into the symbolic capabilities of large language models by Dave et al. 2024 and the arithmetic properties in the space of language model prompts by Krubiński 2023 further contribute to understanding how transformers process arithmetic operations and the challenges involved in symbolic reasoning tasks. Unlike this work and other studies that use smaller transformer models trained from scratch, these works focus on large pre-trained models which exhibit emergent capabilities in arithmetic tasks through large-scale unsupervised training.

Mechanistic interpretability of transformers on arithmetic tasks is further explored by Nanda et al. 2022, who study the phenomenon of grokking in small transformer models trained on modular addition tasks. They fully reverse-engineer the learned algorithm and discover that the model implements a discrete Fourier transform and uses trigonometric identities to convert addition into rotations on a circle. Similarly, Zhong et al. 2023 investigate the mechanistic explanations of neural networks on modular addition tasks. They find that neural networks can discover multiple qualitatively different algorithms when trained on the same task, including known algorithms like the “Clock” algorithm (same as Nanda et al. 2022) and a novel “Pizza” algorithm. Unlike the focus of this thesis, which deals with integer addition where each digit is treated as a separate token, these works are limited to modular addition tasks where each number is a single token, i.e. *123* token instead of tokens for *1*, *2*, and *3*. This simplification allows for a detiled mechanistic understanding of the model but does not address the challenges associated with variable-length inputs and position-wise alignment of digits in full integer addition. Moreover, their primary concern is interpretability and understanding training dynamics, rather than improving length generalization or exploring failure modes in more complex arithmetic tasks.

In summary, the literature demonstrates that transformers can learn arithmetic tasks like integer addition, but challenges remain in achieving robust length generalization and understanding the underlying mechanisms by which these models perform arithmetic operations. These findings are directly relevant to the focus of this thesis, i. e. length generalization on integer addition and failure modes of transformers, since works either simplify the task to be modular addition (Nanda et al. 2022, Zhong et al. 2023), fixed digit length (N. Lee et al. 2023, Quirke and Barez 2023), or propose task-specific architectural modifications to improve performance (McLeish et al. 2024) whose performance on other language tasks is not explored.

3.2 Reasoning in Transformers

Transformers have shown remarkable abilities in reasoning tasks, particularly when employing techniques such as including a chain-of-thought (CoT) in the sequence instead of directly outputting an answer. Z. Li et al. 2024 provide a theoretical understanding of the power of chain-of-thought for decoder-only transformers, demonstrating that CoT empowers the model with the ability to perform inherently serial computation, which is otherwise lacking in transformers, especially for fewer layers. Intuitively, this is due to the fact that a CoT part in generated sequence allows the model to write out intermediate results and perform more computation before arriving at the final answer. This theoretical perspective also supports the experiments described in Chapter 4 involving chain-of-thought training to enhance the reasoning capabilities of models on arithmetic tasks.

X. Wang and D. Zhou 2024 explore the idea that chain-of-thought reasoning paths can be elicited from pre-trained language models by simply altering the decoding process, rather than relying on specific prompting techniques. Instead of decoding by taking the tokens with most activation (greedy decoding), they propose evaluating multiple possible first tokens, and then continue with greedy decoding for each of them. This results in multiple decoded sequences instead of a single answer sequence. They find that paths including a chain-of-thought part already frequently exist among these alternative sequences and that the presence of a CoT in the decoding path correlates with higher confidence in the model's decoded answer. This work strengthens the argument that chain-of-thought reasoning is a powerful mechanism for transformers to improve reasoning capabilities.

Another research direction is training the models to output a chain-of-thought sequence using bootstrapping from existing data and pre-trained models, without the need for curated CoT data. Zelikman, Wu, et al. 2022 propose the Self-Taught Reasoner (STaR) method to bootstrap reasoning capabilities using existing models and answer-only datasets. In STaR, a pre-trained LLM is encouraged to generate intermediate steps before answer using few-shot prompting, and it is assumed that if the resulting answer is correct, the generated CoT steps are also correct. Then, the model is fine-tuned on the generated CoT data. This method is shown to improve reasoning capabilities on various tasks, including arithmetic. Further extending the concept of self-generated reasoning, Zelikman, Harik, et al. 2024 introduce Quiet-STaR, where language models learn to generate "rationales" at each token to explain future text using reinforcement learning, thereby improving their predictions. This approach generalizes previous work on self-taught reasoning by Zelikman, Wu, et al. 2022, training the LLM to implicitly reason without explicit generation of a CoT trace, and can leverage unsupervised text datasets since the objective is not task-specific.

Goyal et al. 2024 propose training language models with "pause tokens", allowing the model to perform more computation before outputting the next tokens. This method improves performance on reasoning tasks, including arithmetic. However, while it is tempting to think that the pause tokens implicitly perform a form of chain-of-thought reasoning, the authors do not explicitly analyze the internal reasoning processes of the model. Moreover, the pause tokens are not directly interpretable as distinct, structured intermediate steps, which is a key feature of chain-of-thought reasoning.

The limitations of transformers in performing counting tasks are highlighted by Yehudai et al. 2024, who focus on simple counting tasks involving counting the number of times a token appears in a string. They show that transformers can solve this task under certain conditions, such as when the dimension of the transformer state is linear in the context length. But in general this ability does not scale beyond this limit, based on the authors' theoretical arguments for the observed limitations. While this work does not directly address arithmetic tasks, it provides insights into the limitations of transformers in handling a related counting task.

Understanding how transformers learn causal structures is investigated by Nichani, Damian, and J. D. Lee 2024, who introduce an in-context learning task requiring the learning of latent causal structures. They prove that gradient descent on a simplified two-layer transformer learns to solve this task by encoding the latent causal graph in the first attention layer. However, this work is highly theoretical and does not directly address arithmetic tasks, nor other algorithmic or language tasks.

Overall, despite some theoretical results proving possibility of certain forms of reasoning, the literature does not show a robust solution that allows transformers to learn the generalizable algorithm for performing integer addition on an unbounded number of digits, which is the main focus of this thesis. Moreover, even if theoretical possibility was to be established, it would still be of interest to interpret the learned algorithm in a higher-level, human-understandable form akin to Nanda et al. 2022.

3.3 Compositional Learning

Compositionality refers to the ability of models to understand and generate new combinations of known components. This is central to generalization, particularly in tasks that require combining simpler operations to solve more complex problems, such as multi-digit integer addition. A key aspect of compositionality is *productivity* as defined by Hupkes et al. 2020, the ability to generalize beyond training examples to sequences of unbounded length, which is directly relevant to the problem of length generalization in addition tasks. In integer addition, this means models must handle operations on numbers larger than those seen during training, an open problem in literature. Alongside productivity, other aspects of compositionality are also introduced in the aforementioned work: *systematicity*, *substitutivity*, *localism*, and *overgeneralisation*. Systematicity refers to the model's ability to recombine familiar parts and rules to form novel outputs, which is relevant in integer addition, since digits and their positions must be combined correctly in the learned algorithm to generalize to new lengths. Substitutivity tests whether models are robust when parts of an input expression are replaced by semantically similar elements. Localism examines whether models process smaller constituents before handling larger structures, which in addition tasks translates to correctly managing digit-wise operations before generating the final answer using their results. Finally, overgeneralisation assesses whether models apply learned rules even in cases where exceptions exist. Thus, multiple aspects of compositionality are directly relevant to the problem explored in this thesis, in particular productivity and systematicity.

Saxton et al. 2019 introduce a symbolic mathematical dataset to benchmark models' ability to correctly compose functions to produce a final answer, and evaluate the Transformer models on this task. The results show that Transformer outperforms other recurrent models, showing promise for compositional learning.

Dziri et al. 2023 investigate the limits of transformers on compositional tasks, such as multi-digit multiplication and logic grid puzzles. They find that transformers tend to solve compositional tasks by reducing multi-step reasoning into *linearized subgraph matching rather* than developing systematic problem-solving skills, suggesting limitations in compositional generalization. Apart from suggesting an interesting description for how multi-step reasoning is realized on transformers, this work does not explore integer addition problems, instead focusing on ¡5 digit multiplication and logic puzzles.

Press, Zhang, et al. 2023 measure the compositionality gap in language models by evaluating how often models can correctly answer all sub-problems but fail to generate the overall solution. They find that as model size increases, single-hop question answering performance improves faster than multi-hop performance, indicating that while larger models have better factual recall, they do not necessarily improve in their ability to perform compositional reasoning. They propose methods like chain-of-thought and self-ask prompting to narrow the compositionality gap.

H. Zhou et al. 2023 propose the RASP-Generalization Conjecture, suggesting that transformers tend to length generalize on a task if it can be solved by a short program (in RASP, which is a domain-specific language describing the transformer architecture introduced by Weiss, Goldberg, and Yahav 2021) that works for all input lengths. They use this framework to understand when and how transformers exhibit strong length generalization on algorithmic tasks, which is closely related to compositional learning.

In summary, while transformers have demonstrated some ability to perform compositional tasks, significant challenges remain in achieving systematic compositional generalization. These findings inform this thesis' focus on multi-task learning and understanding how models can be trained to perform compositional operations like digit alignment and modular sum, before correctly combining results from these sub-tasks into a final answer.

3.4 Length Generalization

Generalization to longer sequence lengths is a critical challenge in training transformers for tasks like integer addition. Positional encoding plays a significant role in length generalization. Kazemnejad et al. 2023 conduct a systematic empirical study comparing different positional encoding approaches, including Absolute Position Embedding (APE) (Vaswani et al. 2017), Relative position encoding (Shaw, Uszkoreit, and Vaswani 2018), ALiBi (Press, Smith, and Lewis 2021), Rotary osition encoding (RoPE) (Su et al. 2024), and Transformers without positional encoding (NoPE). Interestingly, they find that explicit position embeddings are not essential for decoder-only transformers to generalize to longer sequences and that models without positional encoding outperform others in length generalization. The results of experiments conducted in this thesis are not consistent with these findings, observing much steeper performance drops on longer sequences when using NoPE, RoPE or APE.

Multiple works explore the impact of position-based addressing (and therefore positional encodings) in algorithmic tasks. Multiple recent studies discover that the models struggle to precisely select relevant tokens based on position in longer sequence lengths than during training (Ebrahimi, Panchal, and Memisevic 2024; Shen et al. 2023; Zhao et al. 2024; H. Zhou et al. 2023; Yehudai et al. 2024). The experiments in this thesis also confirm that the source of failure in length generalization is the inability of the model to align digits correctly across varying lengths without explicit digit-wise positional cues.

Ruoss et al. 2023 introduce randomized positional encodings to boost length generalization of transformers. They demonstrate that their method allows transformers to generalize to sequences of unseen length by simulating the positions of longer sequences and randomly selecting an ordered subset from this longer sequence. Experiments in this thesis also confirm that randomized positional encodings can improve length generalization on integer addition tasks. Moreover, experiments show that even without architectural changes, simply randomly adding spaces to the input sequence can simulate randomized positions and boost out-of-distribution accuracy for task lengths marginally longer than those encountered during training. The latter claim is also supported by work of Shen et al. 2023.

S. Li et al. 2024 propose a novel functional relative position encoding with progressive interpolation (FIRE) to improve transformer generalization to longer contexts. They theoretically show that FIRE can learn to represent other relative position encodings and demonstrate empirically that FIRE positional encoding enables models to better generalize to longer contexts on long text benchmarks.

The success of length generalization is also linked to data format and position encoding type. Y. Zhou et al. 2024 test the transformer's ability to generalize using the two integer addition task. They show that transformers can achieve limited length generalization, but find that the performance depends on random weight initialization as well. Their approach uses FIRE positional encoding and index hints (e.g. "123+456=579" is encoded as "a1b2c3+a4b5c6=a5b7c9"), achieving 2.5x length generalization ratio (maximum length of successfully predicted test sequences to training sequences). This work is related to the experiments conducted in this thesis, which also explore the impact of different positional encodings and data formatting on length generalization in integer addition tasks. Before this work, the state-of-the-art length generalization ratio was 1.5x (H. Zhou et al. 2023), and before that, 1.125x (Kazemnejad et al. 2023) and 1.1x (Shen et al. 2023).

As mentioned in Section 3.1, McLeish et al. 2024 address length generalization by adding an embedding to each digit that encodes its position relative to the start of the number. This fix, along with architectural modifications such as input injection and recurrent application of layers, allows transformers to generalize to larger and more complex arithmetic problems. Abacus embeddings generalize to 120-digit problems when trained on up to 20-digit problems, achieving state-of-the-art performance of 6x length generalization ratio on integer addition tasks.

Nogueira, Jiang, and Lin 2021 empirically show that transformer models struggle to learn addition rules that are independent of training sequence lengths. Moreover, they find that instead of using character tokenization or digit grouping as done in other works, adding position tokens (such as "3 10e1 2" instead of "32") allows models to learn the addition operation. Their work uses the much larger pre-trained T5 (Raffel et al. 2020) model or smaller encoder-decoder models trained from scratch. Interestingly, evaluation of failure modes shows a similar one to the one observed in this thesis when testing on longer sequences: instead of terminating the output earlier, the model shortens the sequence by omitting tokens from the middle.

In the experiments presented in this thesis, different positional encodings and multi-task training are explored to improve length generalization in integer addition tasks. The literature suggests that carefully designed positional encodings and data formatting can significantly impact the ability of transformers to generalize to longer sequences. Moreover, this work isolates the causes of failure in length generalization on integer addition, and proposes minimally invasive modifications to improve performance on longer sequences.

Chapter 4

Approach

4.1 Overview

TODO: overvie

4.2 Experimental Setup

This section details the experimental setup used to investigate the hypotheses outlined in Section 1.3, including data formatting techniques, model configuration, training procedures, and evaluation metrics employed in the experiments. The experiments cover analysis of the effects of positional encodings, data formatting, and inclusion of sub-task data on the length generalization capabilities of transformer models in the multi-digit integer addition task.

4.2.1 Data Formatting

Various data formatting techniques are employed and their effects on model generalization is exampined. All experiments use character-level tokenization; every character (digit, letter, symbol such as "+", "=", or space) is treated as an individual token. The vocabulary consists of 100 printable ASCII characters, ensuring that each token is represented uniquely. While character tokenization as such is a simple and flexible method, it is worth noting that current state-of-the-art LLMs use other subword tokenization schemes such as Byte-Pair Encoding (BPE) (Sennrich, Haddow, and Birch 2016; Brown et al. 2020). Table 4.1 summarizes the different data formatting techniques with examples.

Standard Format In the standard format, input sequences are represented as "`$a+b=c$`", where a and b are the operands, and c is the sum. The dollar signs "`$`" denote the start and end of the sequence; the final "`$`" also serves as the end-of-sequence token during autoregressive generation, stopping the process once it is generated. The plus "`+`" and equals "`=`" symbols separate the operands and the answer, respectively. For example, adding 123 and 456 is formatted as:

```
$123+456=579$
```

This format serves as the baseline, with other formatting methods building upon it.

Zero Padding Zero padding involves aligning the digits by prepending operands and answers with leading zeros to match a fixed length N_{pad}. For instance, if $N_{\text{pad}} = 5$, the addition of 123 and 456 becomes:

```
$00123+00456=00579$
```

This method ensures that corresponding digits in different numbers always occupy the same absolute positions in the sequence, simplifying the learning of positional relationships. However, it does not actually solve the problem, since it requires prior knowledge of the maximum sequence length and can't be applied to sequences longer than N_{pad}.

Reversing Reversing the digits of operands and/or the answer switches the digit ordering to start with least significant digits, which follows the flow of operations like carry propogation. For example, reversing the operands yields:

```
$321+654=579$
```

Reversing both operands and the answer gives:

```
$321+654=975$
```

Reversing the operands alone, in principle, should not significantly impact performance, as the model can learn appropriate attention patterns to handle reversed sequences. However, reversing the answer theoretically simplifies the task by localizing carry propagation - since the model generates the output from left to right and can compute each digit independently instead of being forced to implicitly perform the carry propagation through the complete answer before generating the first answer digit.

Random Spaces Random spaces are inserted between symbols in the input sequence to disrupt fixed positional patterns and encourage the model to learn position-invariant representations. The number of spaces inserted is controlled by a parameter ρ, representing the maximum ratio of random spaces to non-space tokens in the operands. The maximum number of random spaces is calculated as $n_{\max} = \rho \times L$, where L is the length of the sequence (excluding the start token \$). The actual number of spaces n is sampled uniformly from the set $\{0, \ldots, n_{\max}\}$. For example, if $\rho = 0.5$ and $L = 10$, then $n_{\max} = 5$, and n is randomly chosen from $\{0, 1, 2, 3, 4, 5\}$. The spaces are inserted at random positions within the operands. In all presented experiments, $\rho = 0.5$ when random spaces are enabled. An example of an input sequence with random spaces is:

```
$1 23 +4  5 6=579$
```

Scratchpad A scratchpad, or chain-of-thought, includes intermediate computational steps before the final answer, promoting step-by-step reasoning. This format also allows for the evaluation of intermediate results and aids in understanding the model's reasoning process. However, it requires task-specific data and therefore is not a general method. The scratchpad consists of reversed operands, modular addition with carry notation separated by semicolons, and the final answer separated by a vertical bar. An example with comments describing the parts is given below (the line breaks are included for convenience and not part of the actual sequence):

`$567+789=7 6 5 + 9 8 7;`	*Input equation and reversed operands*
`c=0,7+0+0=7,c=0;`	*Sum of units digit, no carry initially*
`6+9+0=5,c=1;`	*Sum of tens digit, carry generated*
`5+8+1=4,c=1;`	*Sum of hundreds digit and carry, carry generated*
`0+7+1=8,c=0`	*Sum of thousands digit and previous carry*
`|8457$`	*Final result*

This sequence represents the addition of 567 and 789 with detailed computation steps, where `c` denotes the carry variable.

Table 4.1: Examples of Data Formatting Techniques

Format	**Example**
Plain	`$123+456=579$`
Zero Padding (to length 5)	`$00123+00456=00579$`
Reversed Operands	`$321+654=579$`
Reversed Answer	`$123+456=975$`
Reversed Operands and Answer	`$321+654=975$`
Random Spaces	`$12  3 +45 6=579$`
Scratchpad	`$567+789=7 6 5 + 9 8 7;`
	`c=0,7+0+0=7,c=0;`
	`6+9+0=5,c=1;`
	`5+8+1=4,c=1;`
	`0+7+1=8,c=0`
	`|8457$`
Subtask Prefix (placeholder `xxx`)[1]	`xxx$123+456=321+654$`

4.2.2 Data Generation

The training and test datasets for the experiments are systematically generated to evaluate the length generalization capabilities of transformer models on integer

[1]See Section 4.2.2 for details on subtasks and corresponding prefixes.

addition tasks. Each dataset consists of addition problems where both operands a and b are positive integers of equal number of digits, denoted as the *digit length.* An addition problem involving operands of n digits is referred to as an n-digit or $n \times n$ addition problem. For instance, a “4-digit” or “4x4” addition refers to both operands having exactly four digits.

Multiple datasets are created, each encompassing a specific range of in-distribution (ID) digit lengths for training and corresponding out-of-distribution (OOD) digit lengths for testing. The datasets are designed to assess the models’ ability to generalize to sequence lengths beyond those seen during training. In each case, the datasets are split into training, validation, and test sets. The training set contains a specified number of ID digit length samples, while separate validation and test datasets are generated for ID and OOD digit lengths. To prevent data contamination, all samples in the validation and test sets are unique and not present in the training set.

The operand values are randomly sampled to ensure uniform coverage of possible combinations within the specified digit lengths. Numbers are generated such that they have exactly the specified number of digits (i.e., they do not start with zero). Unless specified otherwise, no attempt is made to balance the digit distribution nor the number of carries required in the addition operations across the samples.

Following the methodology of N. Lee et al. 2023, for 1-digit operands (1x1 addition), all possible 100 combinations (operands ranging from 0 to 9) are included in the training dataset and therefore excluded from the test sets to avoid overlap. For 2-digit operands, 900 samples are randomly selected, and for 3-digit operands, 9000 samples are used. For digit lengths of 4 and above, an equal number of samples per digit length are randomly generated to fill the remaining training set size.

Out-of-distribution test sets are constructed by including digit lengths not present in the training data. Each OOD test set contains 1000 unique samples for each OOD digit length to evaluate the model’s length generalization performance.

Sub-Task Data To enhance the compositional learning abilities of the models and investigate their impact on length generalization, various sub-task data was incorporated into some datasets. The sub-tasks include *reversing*, *carry detection*, *digit-wise modular addition*, and *digit alignment.* Each sub-task focuses on a specific aspect of the addition process, aiming to help the model learn underlying algorithmic components that could facilitate better generalization.

In contrast to the scratchpad approach, where intermediate computations are appended sequentially (leading to potential compounding errors), the sub-task training treats each sub-task as an independent auxiliary task. This method allows the model to learn each sub-task simultaneously without relying on the outputs of other tasks. Conversely, sub-task training implicitly involves composing multiple algorithmic parts due to pressure from the complete addition problem included alongside sub-tasks, instead of enforcing composition in each sequence.

To allow the model to differentiate between the sub-tasks each example includes a 3-letter task prefix, resulting in the format `xxx$a+b=c$`, where `xxx` denotes the sub-task identifier, and `c` represents the sub-task-specific output.

The sub-tasks, their prefixes, and their formats are as follows:

- **Digit Alignment** (`ali`): Focuses on aligning the digits of the two operands for position-wise operations. The model learns to output the corresponding digit pairs from each operand.

  Example: `ali$1234+4567=1+4,2+5,3+6,4+7$`

- **Reversing** (`rev`): Involves reversing the digits of each operand. This sub-task helps the model understand the reversal operation, which inverts the propagation order from least significant digit to most.

  Example: `rev$1234+4567=4321+7654$`

- **Carry Detection** (`car`): Requires the model to identify positions where a carry operation would occur during addition. This sub-task is essentially a lookup operation from 2 digits to a binary value. The output is a string of "c"s and dashes, indicating positions with and without carries respectively.

  Example: `car$1234+4567=---c$`

- **Digit-wise Modular Addition** (`mad`): The model performs addition modulo 10 on each pair of corresponding digits without considering carries. This sub-task is also in principle a lookup operation, from 2 digits to another digit.

  Example: `mad$1234+4567=5791$`

- **Addition** (`add`): The standard addition task, where the model computes the sum of the two operands. This serves as the main task and is included alongside sub-tasks in the dataset to facilitate composition of their output.

  Example: `add$1234+4567=5801$`

Datasets Several datasets were generated to support different experiments, each tailored to investigate specific aspects of length generalization and compositional learning. Below, we detail each dataset and its composition:

- `1-3_digit`:
  - **Training Set**: Contains 10,000 samples of addition problems where the operands have 1 and 3 digits. The dataset follows the methodology of N. Lee et al. 2023 and is used to replicate their baseline results.
  - **Test Sets**: Separate test sets are created for each digit length from 1 to 4 digits, including OOD lengths of 2- and 4-digit addition problems not seen during training.
- `1-7_digit`:

  - **Training Set**: Includes addition problems with operands ranging from 1 to 7 digits.
  - **Test Sets**: Comprises test samples for digit lengths 1 to 8, with 8-digit addition serving as the OOD evaluation.
  - **Purpose**: Replicates the extended baseline from N. Lee et al. 2023, examining generalization to slightly longer sequences.

- `generalize_to_longer`:
  - **Training Set**: Consists of 1 million samples with digit lengths from 1 to 17 and 19 digits, intentionally excluding 18-digit problems to create an interpolation gap.
  - **Test Sets**: Includes OOD test sets for 18-digit (interpolation) and 20-digit (extrapolation) addition problems.
  - **Purpose**: Evaluates the model's ability to generalize to unseen lengths within (interpolation) and beyond (extrapolation) the training range.

- `generalize_to_longer_mini`:
  - **Training Set**: Contains addition problems with digit lengths of 1 to 7 digits and 9 digits, deliberately omitting 8-digit problems.
  - **Test Sets**: OOD test sets for 8-, 10-, and 11-digit addition problems.
  - **Scales**: Training data is generated at multiple scales, with datasets of 10K, 100K, 1M, and 10M examples to study the impact of dataset size, where "K" denotes thousands and "M" denotes millions.
  - **Purpose**: Investigates length generalization across different data scales and the effect of missing intermediate digit lengths.

- `generalize_to_longer_mini_multitask`:
  - **Composition**: Similar to `generalize_to_longer_mini`, but includes all five sub-tasks (addition, reversing, carry detection, digit-wise modular addition, digit alignment) in the training data. Contains 2 variants for each scale: addition-only and multi-task training (including sub-tasks).
  - **Scales**: Generated at the same data scales as above.
  - **Purpose**: Examines the effect of sub-task training on compositionality and length generalization as compared to addition-only training.

- `generalize_to_longer_mini_gap`:
  - **Training Set**: Comprises 100K addition problems with digit lengths of 1 to 7 digits and 11 digits, creating a larger "gap" in the training digit lengths.

- **Test Sets**: OOD test sets for digit lengths 8, 9, 10 (interpolation), and 12, 13 (extrapolation).
- **Purpose**: Designed to evaluate whether the model can generalize across a larger gap in sequence lengths.

4.2.3 Model Configuration

Transformer decoder models are used as described in the Background chapter (Section 2.1.2). Unless specified otherwise, a standard transformer decoder architecture is employed. The models are varied along several dimensions to assess their impact on performance:

- **Number of layers (depth)**: The number of decoder layers.
- **Model dimension (width)**: The dimensionality of the model embeddings and hidden representations.
- **Number of attention heads**: The number of attention heads h is chosen such that d is divisible by h, commonly set to powers of 2.
- **Feed-forward layer dimension**: The hidden dimension of the feed-forward layers is set to $d_{\text{ff}} = 4 \times d$.
- **Context length**: The maximum input sequence length, denoted as L_{max}, is set based on the task requirements and is only relevant for models with absolute positional encodings.

Unless otherwise noted, models use absolute positional encodings. For experiments involving different positional encoding schemes, the specific configurations are detailed in the respective sections.

4.2.4 Training and Evaluation

Training Setup Overall, the training parameters from the NanoGPT (Karpathy 2022) are used with modified number of steps, learning rates, and batch sizes. All models are trained using the AdamW optimizer with hyperparameters as shown in Table 4.2.

A learning rate scheduler with linear warm-up and cosine decay is used; for the first $T_{\text{w}} = 100$ iterations, the learning rate increases linearly from 0 to the chosen learning rate. After warm-up, the learning rate at the current iteration $\eta(t)$ is cosine-annealed to a minimum learning rate of $\eta_{\text{min}} = 0.1 \times \eta$ over the course of training until maximum number of iteartions T_{max} according to the formula:

$$\eta(t) = \begin{cases} \eta_{\text{max}} \times \dfrac{t}{T_{\text{w}}}, & \text{if } t < T_{\text{w}} \\ \eta_{\text{min}} + \dfrac{1}{2}(\eta_{\text{max}} - \eta_{\text{min}}) \left[1 + \cos\left(\pi \dfrac{t - T_{\text{w}}}{T_{\text{max}} - T_{\text{w}}}\right)\right], & \text{if } T_{\text{w}} \leq t \leq T_{\text{max}} \\ \eta_{\text{min}}, & \text{if } t > T_{\text{max}} \end{cases}$$

where t is the current iteration.

The batch size is selected based on the task, the model size and the sequence length to maximize GPU utilization while avoiding memory constraints. For large models or long sequences, gradient accumulation over multiple steps is employed to achieve an effective batch size. The number of training epochs is not set explicitly, rather the number of optimizer steps (iterations) is used. Answer loss masking (as described in Section 2.1.5) is applied during training unless specified otherwise. This means that the loss is computed only over the tokens corresponding to the answer part of the sequence, excluding any start tokens or padding tokens.

Table 4.2: Optimizer Hyperparameters

Hyperparameter	Value
Optimizer	AdamW
Learning rate	3×10^{-4}
Betas	$(0.9, 0.999)$
Epsilon	1×10^{-8}
Weight decay	0.1

Inference Procedure During inference, top-k sampling with $k = 1$ is used, which corresponds to greedy decoding by selecting the token with the highest probability at each timestep. Although beam search was implemented and tested, it did not yield significant improvements due to the sharpness of the output distribution; the model's predictions are typically highly confident, with the softmax probabilities concentrated on a single token.

Evaluation Metrics For all experiments, the models are evaluated using two primary metrics:

- **Cross-Entropy Loss**: Calculated over the answer tokens only, providing a measure of the model's average log-likelihood of the correct answer tokens. Lower loss values indicate better performance. The loss is also related to the *perplexity* of the model, which is the exponential of the loss and is often used as a metric in language modeling tasks. The perplexity represents the average number of choices the model has for the next token, with lower values indicating better performance.

- **Accuracy**: Defined as the proportion of samples where all answer tokens are predicted correctly, also known as full match accuracy. A single incorrect digit in the answer results in the sample being marked as incorrect. To compute the accuracy, autoregressive inference (as described in Section 2.1.5) is used to generate the full answer sequence one token at a time until the end-of-sequence token is predicted or the maximum sequence length is reached.

Higher accuracy values indicate better performance. The cross-entropy loss can be used to compare model performance when the accuracy is the same, e.g. when models have been trained short of full convergence and task accuracy is 0.

During training, both training and validation losses are recorded to monitor the learning dynamics. Models are evaluated on validation sets corresponding to both in-distribution (ID) and out-of-distribution (OOD) digit lengths to assess their generalization capabilities.

4.3 Limitations of Absolute Positional Encoding

The limitations of absolute positional encoding become evident when evaluating length generalization in integer addition tasks. Models using absolute positional encodings are unable to generalize even to slightly longer sequences than those seen during training, failing to interpolate or extrapolate to unseen lengths. This failure appears to stem from the inability to extend learned attention patterns beyond the training lengths. Evidence from the analysis of the attention scores presented in Section 4.3.2 indicates that models with absolute positional encodings lack the sharp and well-formed attention patterns observed in models using digit alignment-focused positional encodings like the Abacus encoding. The absolute positional encoding struggles to precisely select the correct digits based on their positions, leading to misalignment in longer sequences. Introducing random spaces into the input sequences slightly smoothens the attention maps, which helps the model generalize in a limited way.

Furthermore, analysis of the next-token prediction uncertainty in Section 4.3.1 reveals that models exhibit high confidence in their predictions for in-distribution lengths but become increasingly uncertain for out-of-distribution lengths, especially for longer sequences. While for in-distribution digit positions the distribution over possible next tokens is usually collapsed, with all probability mass concentrated on the correct token. With OOD positions, on the other hand, next token distribution becomes more uniform and the probability of predicting the correct next token decreases towards zero. This suggests that extending the position addressing patterns to OOD lengths is the main issue, consistent with the literature indicating that vanilla Transformers struggle with index-based addressing operations.

By addressing the digit alignment problem, for example through the use of specialized positional encodings like the Abacus encoding (McLeish et al. 2024), models can generalize well to longer sequences without any other modifications. However, the challenge remains to enhance the generalization capabilities of models using standard absolute positional encodings without task-specific modifications. Introducing random spaces into the sequences is a potential strategy to encourage generalization, allowing models to interpolate and extrapolate to slightly longer sequences. The effects of different data formatting strategies, including the addition of random spaces, on length generalization are explored in the subsequent section. First, the baselines for length generalization are established demonstrating complete failure to generalize to any unseen lengths. Then, experiments are described that investigate the hypotheses that digit alignment is the root cause of failure and adding random spaces is a solution that partially address it while requiring no changes to the architecture.

4.3.1 Length Generalization Baseline

To establish a baseline for length generalization, models with absolute positional encodings were trained on integer addition tasks with operands of certain digit lengths and tested on both seen and longer unseen digit lengths. Specifically, models were trained on sequences with operands of 1 and 3 digits and evaluated on sequences with operands of 2 and 4 digits. The models failed to generalize to the unseen lengths, demonstrating poor performance on both interpolation (lengths between those seen during training) and extrapolation (lengths beyond those seen during training). This inability to generalize supports the hypothesis that absolute positional encodings limit the model's capacity to handle sequences longer than those encountered during training.

In another experiment, models were trained on sequences of 1–7 digits and tested on sequences of 8 digits. The models again failed to generalize to the unseen lengths, further confirming the limitations of absolute positional encodings in facilitating length generalization for integer addition tasks.

Longer Training Sequences To investigate whether training on longer sequences could alleviate this issue, models were trained on sequences with operands of 1–17 and 19 digits and tested on sequences with operands of 18 and 20 digits. Despite the increased training sequence lengths, the models still failed to generalize to the unseen lengths in the vanilla setup with absolute positional encodings. This suggests that merely extending the training sequence lengths does not enable models with absolute positional encodings to generalize to longer sequences.

Figure 4.1 illustrates the performance of models trained under these configurations, showing that absolute positional encodings inherently limit the model's ability to generalize to unseen sequence lengths.

Next-Token Uncertainty Analysis of the next-token prediction uncertainties provides further insights into the limitations of absolute positional encodings. For in-distribution lengths, models exhibit high confidence in their predictions, with low entropy in the next-token distributions. However, for OOD lengths, the models become increasingly uncertain, entropy of the next-token distributions increases, and the probability of the correct token decreases towards zero. Moreover, the analysis suggests that models often omit digits from different positions in the answer and thus attempt to end the sequence prematurely. This is in line with findings from Newman et al. 2020 that the EOS token is output prematurely. Even

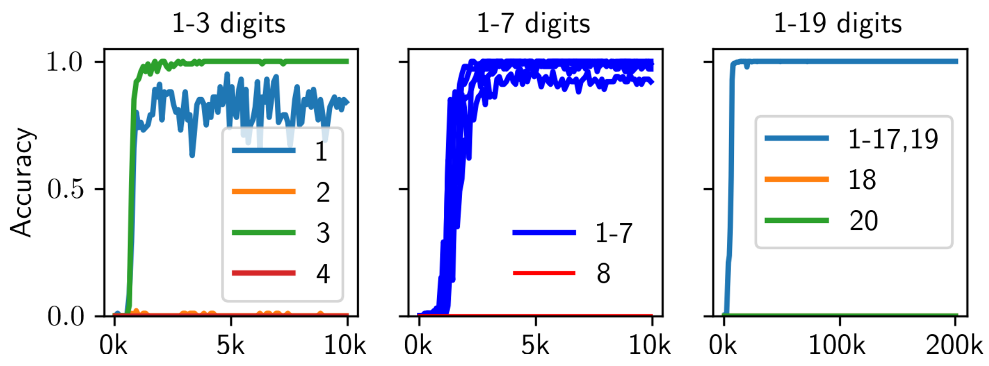


Figure 4.1: Baseline performance of Transformer decoder with absolute positional encodings over training. The model is a NanoGPT with 6 layers, 768 embedding dimension, and 4 attention heads. (Left) Model trained on 1 and 3-digit sequences fail to generalize to 2 and 4-digit sequences. (Middle) Same failure to generalize happens when trained on 1-7 digits and additionally tested on 8 digits. (Right) Model trained on 1–17 and 19-digit sequences fail to generalize to 18 and 20-digit sequences.

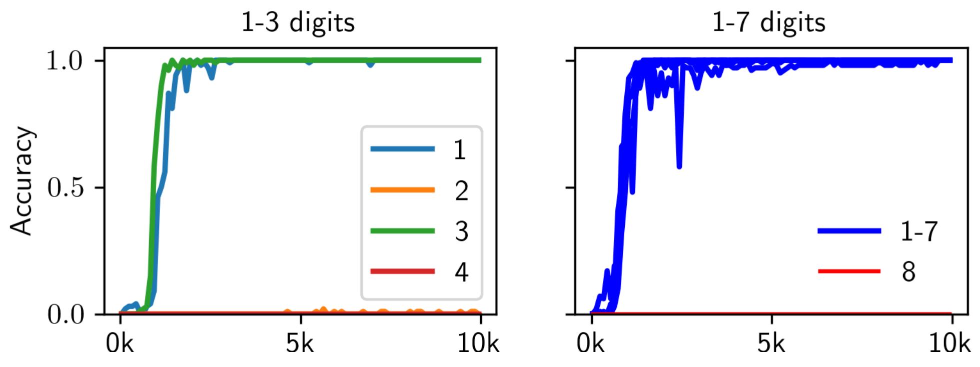


Figure 4.2: Baseline performance of Universal Transformer with absolute positional encodings over training. The model has embedding dimension of 384, 6 attention heads, and 3 recurrent steps (in encoder and decoder each). Similar failure to generalize is observed as in decoder-only model in Figure 4.1.

if EOS token probability is suppressed to let the model output more digits in the answer, the digits are also often incorrect. Figure 4.3 shows the average entropy and probabilities of the correct and end-of-sequence (EOS) tokens over the generated answer sequences for models trained on 1–7 and 9-digit lengths, demonstrating the stark contrast between in-distribution and OOD length performances.

4.3.2 Digit Alignment-Focused Positional Encodings Improve Generalization

The inability of models with absolute positional encodings to generalize to longer sequences suggests that digit alignment is the root issue. To investigate this hypothesis, models were trained using both absolute positional encodings and digit

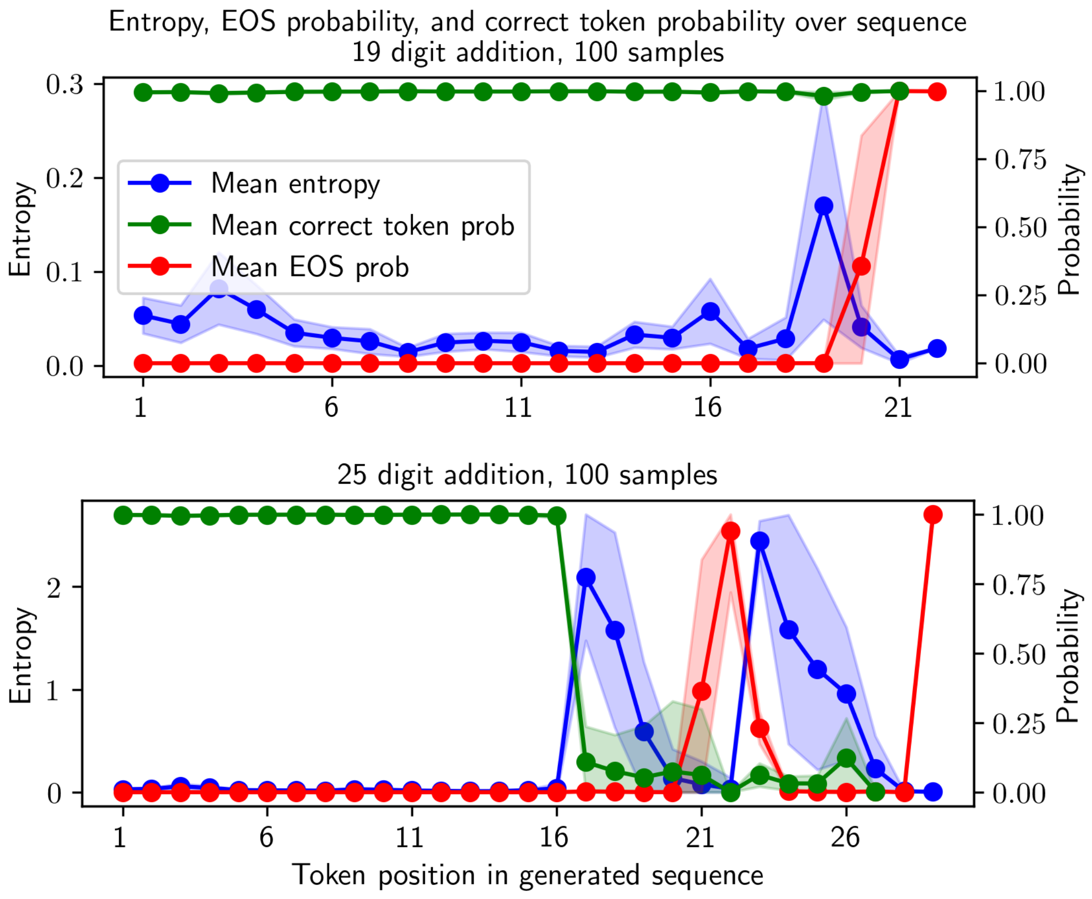


Figure 4.3: Next-token entropy analysis for a model trained on 1-17 and 19 digit lengths with random spaces. The blue, red, and green correspond to the mean entropy, mean probability of EOS token, and mean probability of the ground-truth answer token respectively, with shaded area showing the standard deviation over 100 different prompts. Top graph shows that for in-distribution lengths of up to 19, the entropy (uncertainty) of the next token is very low, and correct tokens are predicted. In the bottom graph, for 25 digit operands the absolute positions of the answer tokens after position 16 are already OOD, with more uncertainty and incorrect EOS prediction showing reliance on absolute positions.

alignment-focused positional encodings, such as the Abacus encoding, under the same setup. Specifically, models were trained on 100,000 samples with operand lengths of 1–7 digits and tested on OOD lengths of 8 and 10 digits.

Attention maps from these models reveal significant differences in how they process input sequences. Models with absolute positional encodings exhibit unclear and diffuse attention patterns that do not extend coherently to longer sequences. In contrast, models using the Abacus positional encoding display crisp attention maps, precisely selecting the correct digits based on their positions, even for sequences longer than those seen during training.

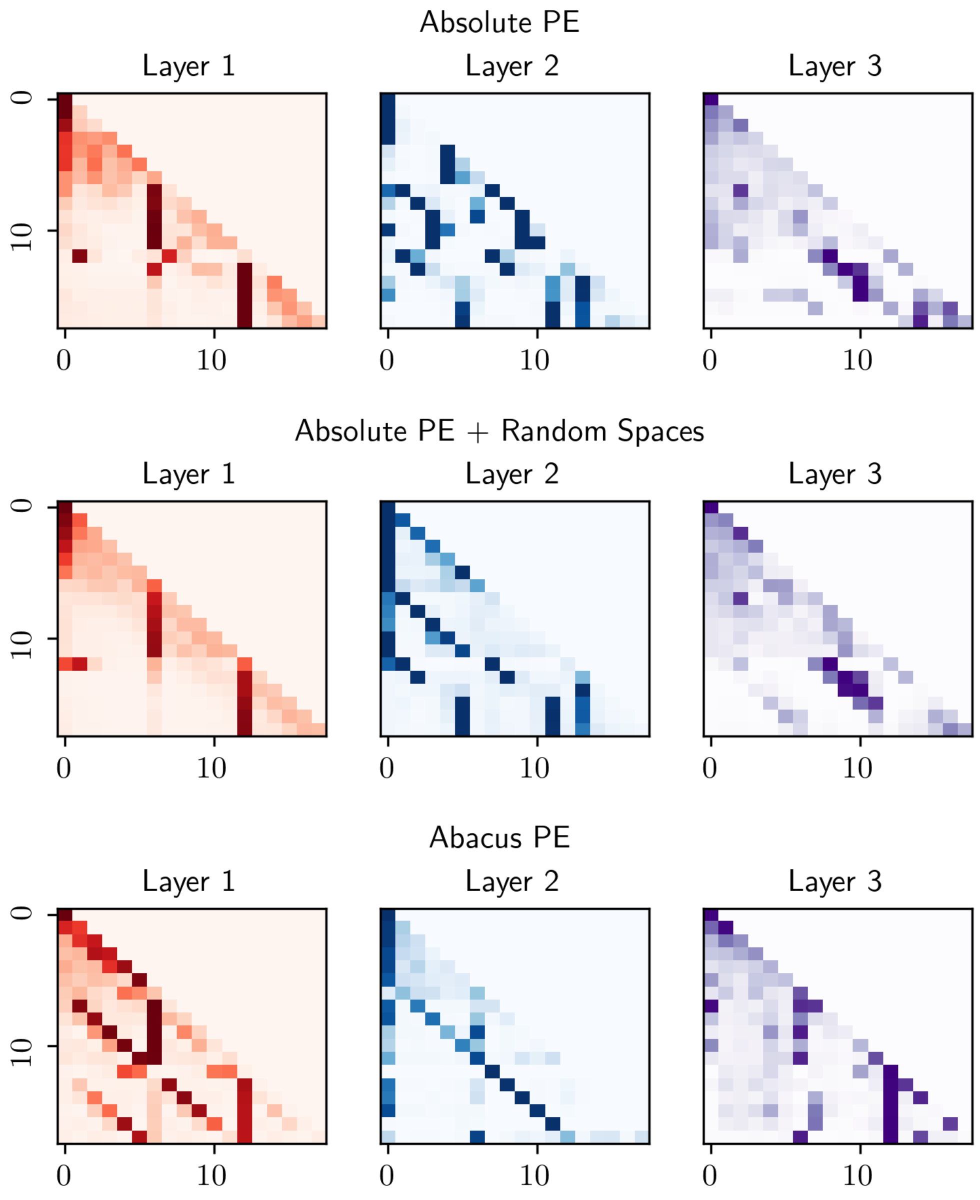


Figure 4.4: Comparison of attention maps (maximum over heads) for models with absolute positional encoding (top), absolute + random spaces, and digit alignment-focused Abacus (McLeish et al. 2024) positional encoding (bottom) on the prompt `$12345+45678=`. It is expected to see orderly lines if the addition is performed according to the correct algorithm. The model with absolute positional encodings shows unclear attention patterns with some randomness, while the Abacus model is significantly less noisy. While random spaces does not fix the issue completely, the attention patterns are significantly more aligned. The differences are espetially evident in Layer 2.

Figure 4.4 illustrates the attention maps for both models. The attention patterns in the Abacus model suggest that aligning digits effectively is crucial for generalizing integer addition to longer sequences. This supports the hypothesis that digit alignment-focused positional encodings enhance the model's capacity to generalize by facilitating correct digit alignment.

Furthermore, evaluation results, as shown in Figure 4.5, demonstrate that models with digit alignment-focused positional encodings significantly outperform those with absolute positional encodings on OOD lengths. This underscores the importance of digit alignment in achieving length generalization in integer addition tasks.

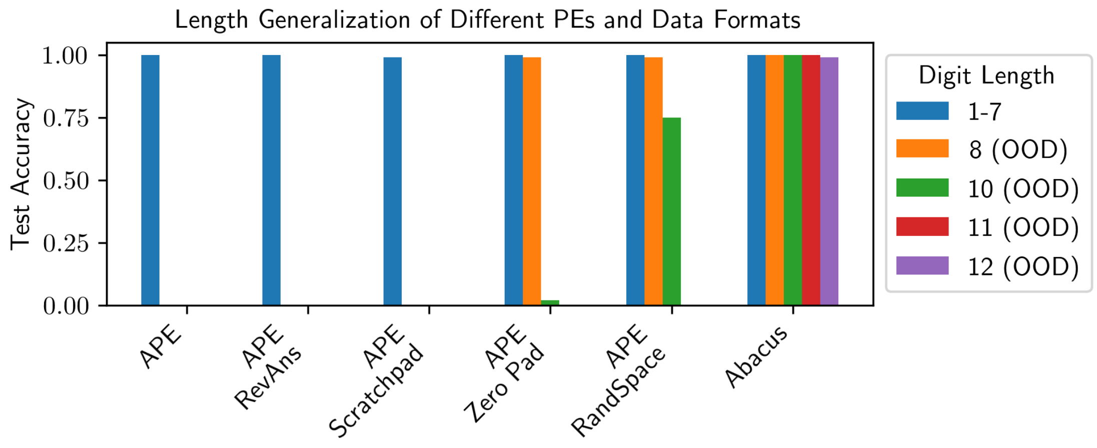


Figure 4.5: Performance comparison of models with different positional encodings and data formatting methods on in-distribution (1–7 digits) and out-of-distribution (8, 10, 11, and 12 digits) lengths. Regular absolute positional encoding (APE) fails on all OOD lengths. Reversing the answer and scratchpad do not improve OOD accuracy. Zero-padding numbers to 13 digits improves interpolating (8 digit) OOD length but does not help extrapolation, while adding random spaces helps to partially extrapolate by 1 digit. Abacus encoding solves the underlying position alignment issue and generalizes to OOD lengths.

4.3.3 Breaking Positional Patterns in Absolute Positional Encodings Allows Weak Generalization

To attempt improve length generalization without altering the model architecture or introducing task-specific modifications, the impact of randomly adding spaces into input sequences was investigated. The hypothesis is that random spaces disrupt fixed positional patterns that the model might overfit to, encouraging it to learn more robust representations that are less dependent on absolute positions.

Experiments were conducted where random spaces were added to the input sequences during training according to the method described in Section 4.2.1. The models trained with this data formatting strategy exhibited marginal improvements in length generalization, successfully interpolating to lengths within the training distribution and extrapolating to sequences that are one digit longer than those seen during training.

Attention maps of these models, as shown in Figure 4.4, indicate that introducing random spaces smoothens the attention patterns, helping the model to generalize slightly better. The models become less reliant on fixed positional patterns and more capable of handling variations in the input sequences.

Evaluation results in Figure 4.5 compare the performance of models with absolute positional encodings, with and without random spaces, highlighting the modest gains in generalization achieved through this data formatting strategy.

4.4 Impact of Other Data Formats on Length Generalization

Different data formatting strategies can influence the model's ability to generalize to longer sequences. This section investigates the effects of various data formatting techniques, including zero padding, reversing the answer or operands, introducing random spaces, and using a scratchpad approach, on length generalization in integer addition tasks.

4.4.1 Zero Padding

Zero padding involves padding the operands to a fixed maximum length by adding leading zeros, effectively aligning the digits of the operands. This technique directly addresses the digit alignment issue by ensuring that digits in the same positional place are aligned across all sequences.

Experiments were conducted where models were trained on zero-padded sequences up to a fixed maximum length. The results showed that zero padding significantly improved performance on sequences up to the maximum padded length, as the model could effectively learn to align and process the digits.

However, zero padding has inherent limitations. It requires prior knowledge of the maximum sequence length, which may not be practical in scenarios where the sequence lengths are unbounded or variable. Additionally, zero padding does not enable the model to generalize to sequences longer than the maximum padded length, as the model has not learned to handle sequences of greater length.

TODO: *refer 4.5*

4.4.2 Reversing the Answer

Reversing the answer involves reversing the order of the digits in the output sequence. Theoretically, reversing the answer can simplify the addition task for the model by reducing the need for carry propagation through the entire answer before outputting the first digit. This is because the least significant digit, which is computed first in the addition process, becomes the first digit in the output sequence.

Experiments were conducted to assess the impact of reversing the answer on model generalization. Models were trained with and without reversed answers using absolute positional encodings. Contrary to expectations, reversing the answer did not lead to significant improvements in generalization performance. As shown in Figure 4.5, the performance of models with reversed answers was comparable to those without reversal.

In the case of digit alignment-focused positional encodings like the Abacus encoding, reversing the operands is also necessary to maintain proper digit alignment. Reversing the operands aligns the least significant digits, facilitating accurate addition.

Interestingly, during training, models with both reversed and non-reversed answers exhibited similar learning patterns. Initially, the models correctly predicted the most significant digits, gradually improving their predictions for the less significant digits as training progressed, up to the maximum in-distribution length. This suggests that reversing the answer does not significantly alter the model's learning dynamics or its ability to generalize to longer sequences.

A detailed analysis of the digit-wise prediction accuracy over the course of training is provided in Appendix **??**, illustrating how the models improve their predictions for each digit position.

4.4.3 Scratchpad

The scratchpad approach involves including intermediate computation steps in the training data, similar to a chain-of-thought, where the model outputs not only the final answer but also the intermediate results leading to it. This method aims to make the underlying computation process explicit, potentially aiding the model in learning the algorithmic steps required for addition.

Experiments were conducted to evaluate the effectiveness of the scratchpad approach in improving length generalization. The results indicated that the scratchpad approach did not significantly enhance generalization to longer sequences. Models trained with scratchpad data exhibited similar performance on OOD lengths compared to models trained without it.

However, the scratchpad approach proved useful for mistake analysis. By examining the model's intermediate outputs, it was possible to identify specific points of failure in the computation process. This provided insights into which parts of the addition algorithm the model struggled with, such as carry propagation or digit-wise addition.

The limited effectiveness of the scratchpad approach in improving generalization contrasts with findings in the literature, where scratchpad has been shown to aid in generalization (N. Lee et al. 2023). One possible explanation is that smaller models, as used in these experiments, may not have sufficient capacity to leverage the additional information provided by the scratchpad.

Figure 4.6 presents the evaluation results for models trained with the scratchpad approach, including violin plots of intermediate task accuracies.

Overall, while the scratchpad approach did not enhance length generalization, it provided valuable insights into the model's internal computations and error patterns. The practicality of the scratchpad method is limited for tasks where intermediate computations are not readily available or known in practice.

Figure 4.6: Evaluation results for models trained with the scratchpad approach. The violin plots depict the distribution of accuracies for intermediate tasks, highlighting the model's performance on different components of the addition process.

4.5 Sub-task Learning for Compositionality

Incorporating sub-task data into the training process aims to enhance the model's compositionality and length generalization capabilities by explicitly teaching the underlying algorithmic components of integer addition. This section explores the hypothesis that training on sub-tasks, such as carry detection, digit-wise modular addition, reversing, and digit alignment, can improve the model's ability to compose these functions and generalize to sequences longer than those seen during training.

A comprehensive analysis was conducted across various model dimensions, ranging from 64 to 1536, and dataset sizes of 10,000, 100,000, 1 million, and 10 million examples. Models were trained under two settings: addition-only and multi-task training, where the latter involved mixing addition tasks with sub-task data. The main finding is that mixed-task training sometimes improves OOD loss, with the effect being particularly pronounced for smaller models and smaller dataset sizes.

4.5.1 Sub-Tasks Marginally Improve Length Generalization

To evaluate the impact of sub-task learning on length generalization, models were trained under two settings: addition-only and mixed-task training. The mixed-task training involved combining the main addition task with various sub-tasks, such as carry detection, digit-wise modular addition, reversing, and digit alignment. The experiments spanned a range of model dimensions (from 64 to 1536) and dataset sizes (10,000 to 10 million examples).

The results indicate that incorporating sub-task data marginally improves length generalization, particularly for smaller models and smaller dataset sizes. As shown in Figure 4.7, smaller models (e.g., with a dimension of 64 and 8 attention heads) benefit more from mixed-task training compared to larger models. The smaller models exhibit lower OOD validation losses when trained with sub-tasks, suggesting that they are better able to leverage the compositional learning provided by the sub-tasks.

One possible explanation for this phenomenon is that larger models have sufficient capacity to memorize the training data, including the sub-tasks, without necessarily learning the underlying algorithms that facilitate generalization. In contrast, smaller models are constrained by their limited capacity and are thus compelled to learn more generalizable representations and algorithms, with the sub-tasks aiding in this process.

Table 4.3 summarizes the performance differences between addition-only and mixed-task training across different model dimensions and dataset sizes.

Table 4.3: Comparison of OOD accuracy for models trained with addition-only and sub-task training across dataset sizes. Smaller datasets show greater improvement with mixed-task training. Despite weak absolute performance, sub-task training can marginally improve length generalization.

	8 Digits		10 Digits	
Dataset	Acc (Add)	Acc (Mix)	Acc (Add)	Acc (Mix)
10K	7.0 ± 22.7	35.4 ± 39.6	0 ± 0	2.4 ± 8.4
100K	21.5 ± 29.7	41.8 ± 36.5	0 ± 0	1.0 ± 5.2
1M	16.1 ± 24.8	39.5 ± 36.3	0 ± 0	1.5 ± 7.6
10M	18.6 ± 33.1	37.3 ± 37.2	0 ± 0	3.1 ± 10.1
Improvement		**+22.7**		**+2.0**

Figure 4.7 illustrates the validation loss versus model dimension for both in-distribution and OOD lengths, highlighting how mixed-task training impacts models of different sizes.

Additionally, analyzing the in-distribution versus out-of-distribution losses reveals that in some cases, sub-task learning helps mitigate overfitting. As depicted in Figure 4.8, certain models achieve a Pareto improvement in OOD loss without an increase in in-distribution loss when trained with sub-tasks.

4.5.2 Smaller Models Benefit More from Sub-Tasks

The experiments demonstrate that smaller models benefit more significantly from sub-task learning compared to larger models. Specifically, models with smaller dimensions exhibit greater reductions in OOD validation loss when trained with mixed tasks.

For the smallest model configuration (64 dimensions), the incorporation of sub-task data leads to a notable improvement in length generalization. In contrast, larger models (e.g., with 1536 dimensions) show minimal differences in performance between addition-only and mixed-task training. This suggests that larger models may already have sufficient capacity to memorize the training data, including the addition task, without relying on compositional learning provided by the sub-tasks.

The observed trend indicates that sub-task learning is particularly advantageous for models with limited capacity, as it encourages the learning of generalizable algorithms rather than memorization.

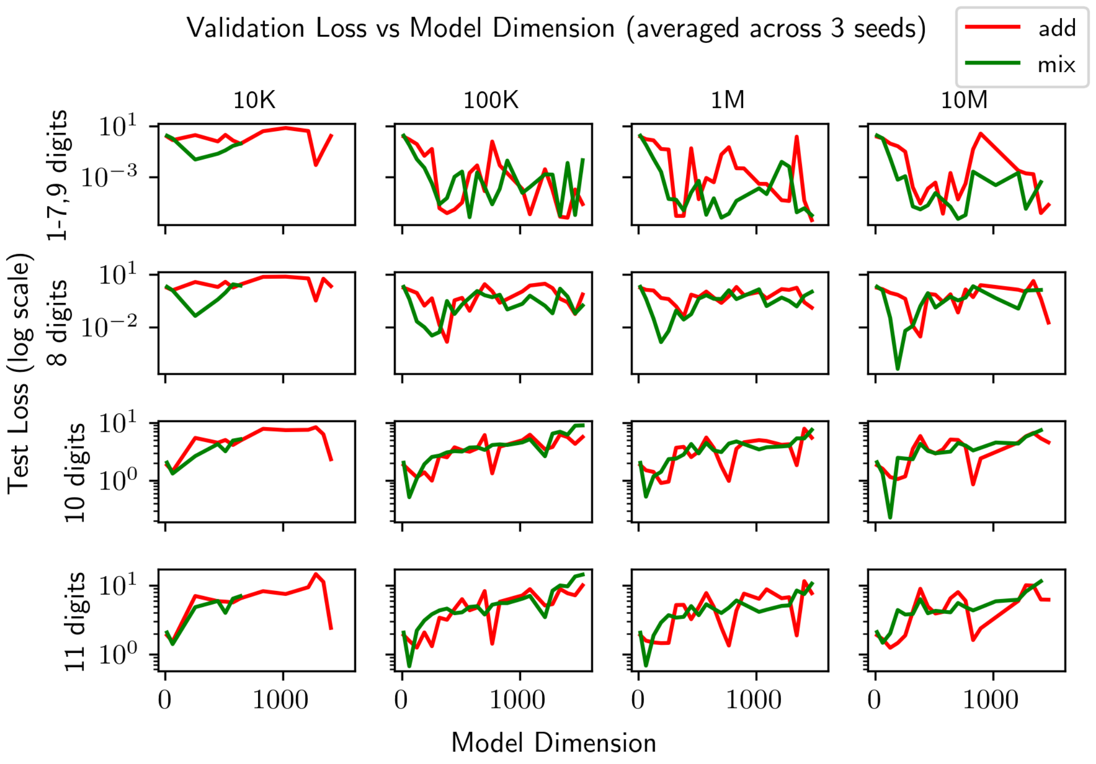


Figure 4.7: Validation loss versus model dimension for addition-only (blue) and mixed-task training (orange). The effect of mixed-task training is more pronounced in smaller models and diminishes as model size increases.

4.5.3 Sub-Task Difficulty and Learning Order

An analysis of the sub-tasks reveals variations in their relative difficulty and the order in which they are learned during training. Some sub-tasks, such as digit-wise modular addition, are learned more quickly and consistently across different training runs. Other sub-tasks, like carry detection, exhibit instability, with models achieving either high or low accuracy after the same number of training iterations, depending on the random seed.

Figure 4.9 presents violin plots of sub-task accuracies across different model sizes, highlighting the variability in sub-task difficulty. Smaller models may find certain sub-tasks more challenging, while larger models may learn them more easily.

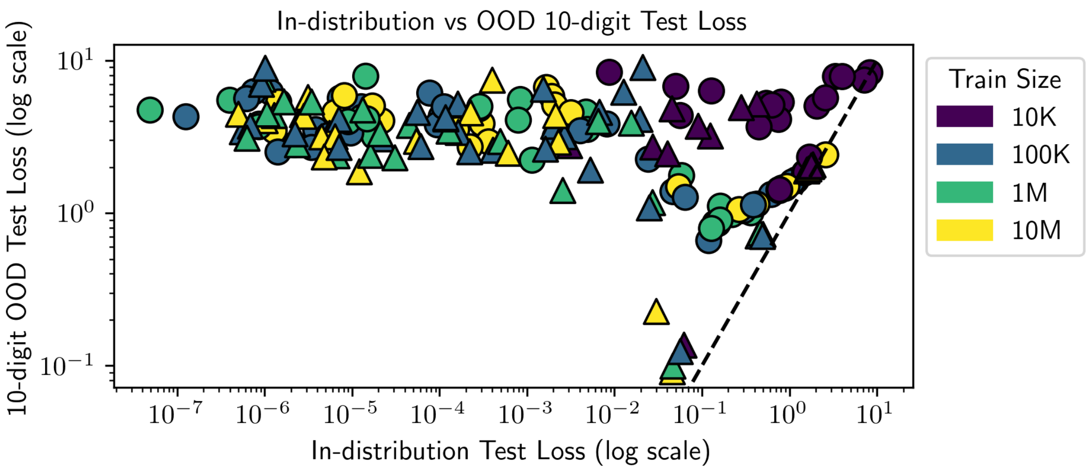


Figure 4.8: Comparison of in-distribution test loss and 10-digit out-of-distribution (OOD) test loss across different training sizes for models trained on addition only (circles) and sub-tasks (triangles). Colors represent the training dataset size (10K, 100K, 1M, and 10M), and the dashed line indicates the diagonal where in-distribution and OOD losses match. Models trained with sub-tasks can achieve a lower OOD loss by an order of magnitude, compared to addition-only training.

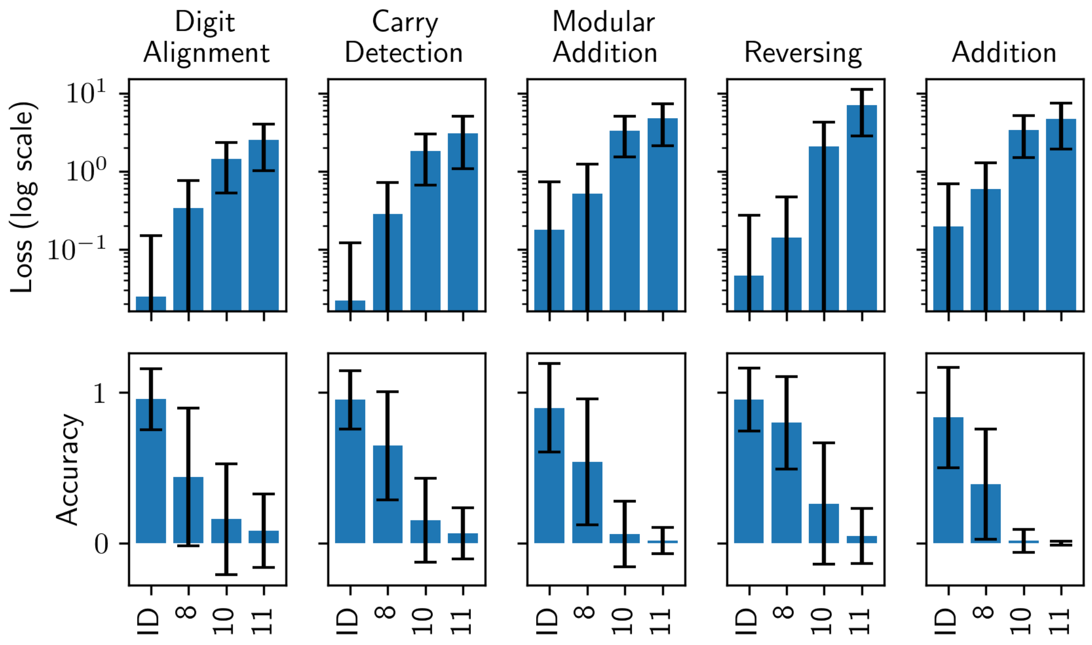


Figure 4.9: Sub-task difficulty analysis. The bar plots show the mean and standard deviation of test loss (top row) and accuracy (bottom row) across in-distribution (ID) and out-of-distribution (OOD) datasets (8, 10, and 11 digits) and subtasks. Loss values across tasks are not directly comparable, but can show ID-vs-OOD lengths deterioration in performance within a task. Based on the accuracy, modular adddition and digit alignment are the most difficult sub-tasks, and their performance has the most variability across model sizes as well.

Chapter 5

Conclusion

TODO: *conclud*

5.1 Contributions

5.2 Limitations

The work focuses on small, specialist models trained from scratch, and connection to LLMs and pre-trained models is not explored, but very interesting and relevant (not clear how findings translate to LLMs and what practical suggestions can be made to improve their performance). Also exploring other popular positional encodings is interesting and also missing.

The sub-task experiment hints at interesting stuff, but has the confounder of number of training steps (all models are trained for the same number of steps and are under-trained due to compute constraints). Thus, maybe the results change when training is done for much more steps.

Mechanistic interpretability methods to causally trace the origins of failure at circuit-level are not attempted, though from literature it is unclear whether the usual activation patching etc. can be useful in algotihmic tasks, and what other methods can be used.

5.3 Future Work

More mechanistic interpretability methods for algorithmic tasks can be conceived and explored, to understand the failure modes of transformers in these tasks.

Applying insights to data curation for finetuning on sub-tasks or other architectural aspects of pre-trained LLMs is interesting, and can be explored.

To investigate sub-task learning more might try more data scales, and try keeping all addition data and adding subtasks on top instead of decreasing the amount of addition data to keep the total constant, see if models can still overfit to that. Furthermore, can try to figure out the “interpolation threshold” (max number of samples it can overfit to) for a given model size, and then start with that amount of addition data, then see if subtasks help push that threshold further or not: idea is that if the model is already at capacity (can barely memorize more samples), the only way to keep loss on addition and improve on subtasks is to learn a generalizing algorithm and then collapse the circuits -¿ subtasks help. If the model is not at capacity, it can just memorize more samples and not learn the generalizing algorithm, so subtasks don't help.

Also, discuss connection to active learning, where the model can choose which samples to learn from, and how that can be applied here to select subtasks or samples. Different methods can be used to choose that, e.g. uncertainty, diversity, etc. Continual learning is also interesting here, can benefit from the many methods in literature to apply curriculum learning with varying subtask difficulty and digit length, avoid catastrophic forgetting, etc.

ODO: *cite parisi ntinual 2019 ontinual lifelong arning with neural etworks A review*

Appendix A

Nomenclature

Symbol	**Description**
SOTA	State-of-the-art
MLP	Multi-layer perceptron
SGD	Stochastic gradient descent
FFN	Feed-forward network
RNN	Recurrent neural network
GPT	Generative pre-trained transformer
BERT	Bidirectional encoder representations from transformers
T5	Text-to-text transfer transformer
LLM	Large language model
CoT	Chain-of-thought
OOD	Out-of-distribution
MHA	Multi-head attention
PE	Positional encoding
APE	Absolute positional encoding

Table A.1: Table of nomenclature

Appendix B

Additional Results

B.1 Intermediate Representation Analysis with Logit Lens

The logit lens method (nostalgebraist 2020) involves inspecting the logits (pre-softmax outputs) of the model at various layers to interpret intermediate representations. By projecting the activations of each layer onto the output logits, it allows the predictions to be tracked as they are refined through the layers of the network.

B.2 Algorithmic String Tasks

To show that index-based addressing is the problem, simplified string tasks such as determining string length, or retrieving a character at given index, which vanilla transformers with absolute positional encodings also fail on OOD lengths.

B.3 Models Learn Most Significant Digits First

Interestingly, over the duration of training, first model gets the digit in most significant position right, and then gets more and more digits right towards the least significant. This is also seen in loss, which drops in steps. This is not because in sampling, the most significant digit is output first (left-to-right), since the effect also happens (in reverse) when the answer is reversed!

TODO: *PLO nanogpt trainii plot stepwise lo and number digits correct*

Bibliography

Achiam, OpenAI Josh et al. (Mar. 15, 2023). "GPT-4 Technical Report". In.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton (July 21, 2016). *Layer Normalization*. DOI: `10.48550/arXiv.1607.06450`. arXiv: `1607.06450[cs, stat]`.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (Sept. 1, 2014). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *CoRR*.

Belkin, Mikhail et al. (Aug. 6, 2019). "Reconciling modern machine-learning practice and the classical bias–variance trade-off". In: *Proceedings of the National Academy of Sciences* 116.32. Publisher: Proceedings of the National Academy of Sciences, pp. 15849–15854. DOI: `10.1073/pnas.1903070116`.

Brown, Tom et al. (2020). "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., pp. 1877–1901.

Bulatov, Aydar, Yury Kuratov, and Mikhail Burtsev (Dec. 2022). "Recurrent Memory Transformer". en. In: *Advances in Neural Information Processing Systems* 35, pp. 11079–11091.

Burtsev, Mikhail S. et al. (Feb. 2021). *Memory Transformer*. arXiv:2006.11527 [cs]. DOI: `10.48550/arXiv.2006.11527`.

Csordás, Róbert (2023). "Systematic generalization in connectionist models". In.

Csordás, Róbert, Kazuki Irie, and Jürgen Schmidhuber (Oct. 6, 2021). "The Neural Data Router: Adaptive Control Flow in Transformers Improves Systematic Generalization". In: International Conference on Learning Representations.

Dave, Neisarg et al. (May 2024). "Investigating Symbolic Capabilities of Large Language Models". In.

Dehghani, Mostafa et al. (Sept. 2018). "Universal Transformers". en. In.

Devlin, Jacob et al. (June 2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. NAACL-HLT 2019. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186. DOI: `10.18653/v1/N19-1423`.

Duan, Shaoxiong and Yining Shi (Oct. 2023). *From Interpolation to Extrapolation: Complete Length Generalization for Arithmetic Transformers*. arXiv:2310.11984 [cs]. DOI: `10.48550/arXiv.2310.11984`.

Dziri, Nouha et al. (Nov. 2023). “Faith and Fate: Limits of Transformers on Compositionality”. en. In.

Ebrahimi, MohammadReza, Sunny Panchal, and Roland Memisevic (Aug. 26, 2024). “Your Context Is Not an Array: Unveiling Random Access Limitations in Transformers”. In: First Conference on Language Modeling.

Elhage, Nelson et al. (2021). “A Mathematical Framework for Transformer Circuits”. In: *Transformer Circuits Thread.*

Fan, Ying et al. (Sept. 25, 2024). *Looped Transformers for Length Generalization.* DOI: `10.48550/arXiv.2409.15647`. arXiv: `2409.15647[cs]`.

Ferrando, Javier et al. (May 2024). *A Primer on the Inner Workings of Transformer-based Language Models.* arXiv:2405.00208 [cs]. DOI: `10.48550/arXiv.2405.00208`.

Goyal, Sachin et al. (Apr. 2024). *Think before you speak: Training Language Models With Pause Tokens.* arXiv:2310.02226 [cs]. DOI: `10.48550/arXiv.2310.02226`.

Haviv, Adi et al. (Dec. 2022). “Transformer Language Models without Positional Encodings Still Learn Positional Information”. In: *Findings of the Association for Computational Linguistics: EMNLP 2022.* Findings 2022. Ed. by Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, pp. 1382–1390. DOI: `10.18653/v1/2022.findings-emnlp.99`.

He, Kaiming et al. (June 2016). “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* Conference Name: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) ISBN: 9781467388511 Place: Las Vegas, NV, USA Publisher: IEEE, pp. 770–778. DOI: `10.1109/CVPR.2016.90`.

Hupkes, Dieuwke et al. (Apr. 2020). “Compositionality Decomposed: How do Neural Networks Generalise?” en. In: *Journal of Artificial Intelligence Research* 67, pp. 757–795. ISSN: 1076-9757. DOI: `10.1613/jair.1.11674`.

Jelassi, Samy et al. (June 2023). *Length Generalization in Arithmetic Transformers.* arXiv:2306.15400 [cs]. DOI: `10.48550/arXiv.2306.15400`.

Karpathy, Andrej (2022). *NanoGPT.* `https://github.com/karpathy/nanoGPT`.

Kazemnejad, Amirhossein et al. (Nov. 2023). “The Impact of Positional Encoding on Length Generalization in Transformers”. en. In.

Krubiński, Mateusz (Oct. 2023). “Basic Arithmetic Properties in the Space of Language Model Prompts”. en. In.

Lee, Nayoung et al. (Oct. 2023). “Teaching Arithmetic to Small Transformers”. en. In.

Li, Shanda et al. (Mar. 2024). *Functional Interpolation for Relative Positions Improves Long Context Transformers.* en. arXiv:2310.04418 [cs].

Li, Zhiyuan et al. (May 2024). *Chain of Thought Empowers Transformers to Solve Inherently Serial Problems.* arXiv:2402.12875 [cs, stat]. DOI: `10.48550/arXiv.2402.12875`.

Loshchilov, Ilya and Frank Hutter (Sept. 27, 2018). “Decoupled Weight Decay Regularization”. In: International Conference on Learning Representations.

McLeish, Sean et al. (May 2024). *Transformers Can Do Arithmetic with the Right Embeddings*. arXiv:2405.17399 [cs] version: 1. DOI: `10.48550/arXiv.2405.17399`.

Nakkiran, Preetum et al. (Dec. 2021). "Deep double descent: where bigger models and more data hurt*". In: *Journal of Statistical Mechanics: Theory and Experiment* 2021.12. Publisher: IOP Publishing and SISSA, p. 124003. ISSN: 1742-5468. DOI: `10.1088/1742-5468/ac3a74`.

Nanda, Neel et al. (Sept. 2022). "Progress measures for grokking via mechanistic interpretability". en. In.

Newman, Benjamin et al. (Nov. 2020). "The EOS Decision and Length Extrapolation". In: *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*. BlackboxNLP 2020. Ed. by Afra Alishahi et al. Online: Association for Computational Linguistics, pp. 276–291. DOI: `10.18653/v1/2020.blackboxnlp-1.26`.

Nichani, Eshaan, Alex Damian, and Jason D. Lee (Feb. 2024). *How Transformers Learn Causal Structure with Gradient Descent*. arXiv:2402.14735 [cs, math, stat]. DOI: `10.48550/arXiv.2402.14735`.

Nogueira, Rodrigo, Zhiying Jiang, and Jimmy Lin (Apr. 12, 2021). *Investigating the Limitations of Transformers with Simple Arithmetic Tasks*. DOI: `10.48550/arXiv.2102.13019`. arXiv: `2102.13019`.

nostalgebraist (Aug. 31, 2020). "interpreting GPT: the logit lens". In.

Press, Ofir, Noah Smith, and Mike Lewis (Oct. 6, 2021). "Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation". In: International Conference on Learning Representations.

Press, Ofir, Muru Zhang, et al. (Dec. 2023). "Measuring and Narrowing the Compositionality Gap in Language Models". In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, pp. 5687–5711. DOI: `10.18653/v1/2023.findings-emnlp.378`.

Quirke, Philip and Fazl Barez (Oct. 2023). "Understanding Addition in Transformers". en. In.

Radford, Alec and Karthik Narasimhan (2018). "Improving Language Understanding by Generative Pre-Training". In.

Raffel, Colin et al. (Jan. 1, 2020). "Exploring the limits of transfer learning with a unified text-to-text transformer". In: *J. Mach. Learn. Res.* 21.1, 140:5485–140:5551. ISSN: 1532-4435.

Ruoss, Anian et al. (May 2023). *Randomized Positional Encodings Boost Length Generalization of Transformers*. arXiv:2305.16843 [cs, stat]. DOI: `10.48550/arXiv.2305.16843`.

Saxton, D. et al. (Apr. 2, 2019). "Analysing Mathematical Reasoning Abilities of Neural Models". In: *ArXiv*.

Sennrich, Rico, Barry Haddow, and Alexandra Birch (Aug. 2016). "Neural Machine Translation of Rare Words with Subword Units". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. ACL 2016. Ed. by Katrin Erk and Noah A. Smith.

Berlin, Germany: Association for Computational Linguistics, pp. 1715–1725. DOI: 10.18653/v1/P16-1162.

Shaw, Peter, Jakob Uszkoreit, and Ashish Vaswani (June 2018). "Self-Attention with Relative Position Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. Ed. by Marilyn Walker, Heng Ji, and Amanda Stent. New Orleans, Louisiana: Association for Computational Linguistics, pp. 464–468. DOI: 10.18653/v1/N18-2074.

Shen, Ruoqi et al. (Nov. 22, 2023). *Positional Description Matters for Transformers Arithmetic*. DOI: 10.48550/arXiv.2311.14737. arXiv: 2311.14737.

Su, Jianlin et al. (Feb. 2024). "RoFormer: Enhanced transformer with Rotary Position Embedding". In: *Neurocomputing* 568, p. 127063. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2023.127063.

Vaswani, Ashish et al. (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc.

Wang, Kevin Ro et al. (Sept. 29, 2022). "Interpretability in the Wild: a Circuit for Indirect Object Identification in GPT-2 Small". In: The Eleventh International Conference on Learning Representations.

Wang, Xuezhi and Denny Zhou (May 23, 2024). *Chain-of-Thought Reasoning Without Prompting*. DOI: 10.48550/arXiv.2402.10200. arXiv: 2402.10200[cs].

Weiss, Gail, Yoav Goldberg, and Eran Yahav (July 1, 2021). "Thinking Like Transformers". In: *Proceedings of the 38th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, pp. 11080–11090.

Yang, Liu et al. (Oct. 13, 2023). "Looped Transformers are Better at Learning Learning Algorithms". In: The Twelfth International Conference on Learning Representations.

Yehudai, Gilad et al. (July 2024). *When Can Transformers Count to n?* DOI: 10.48550/arXiv.2407.15160.

Zelikman, Eric, Georges Harik, et al. (Mar. 2024). *Quiet-STaR: Language Models Can Teach Themselves to Think Before Speaking*. arXiv:2403.09629 [cs]. DOI: 10.48550/arXiv.2403.09629.

Zelikman, Eric, Yuhuai Wu, et al. (Dec. 2022). "STaR: Bootstrapping Reasoning With Reasoning". en. In: *Advances in Neural Information Processing Systems* 35, pp. 15476–15488.

Zhao, Liang et al. (Apr. 2, 2024). *Length Extrapolation of Transformers: A Survey from the Perspective of Positional Encoding*. DOI: 10.48550/arXiv.2312.17044. arXiv: 2312.17044[cs].

Zhong, Ziqian et al. (Nov. 2, 2023). "The Clock and the Pizza: Two Stories in Mechanistic Explanation of Neural Networks". In: Thirty-seventh Conference on Neural Information Processing Systems.

Zhou, Hattie et al. (Oct. 2023). *What Algorithms can Transformers Learn? A Study in Length Generalization*. arXiv:2310.16028 [cs, stat]. DOI: 10.48550/arXiv.2310.16028.

Zhou, Yongchao et al. (Feb. 2024). *Transformers Can Achieve Length Generalization But Not Robustly.* arXiv:2402.09371 [cs].

Zuo, Chunsheng and Michael Guerzhoy (June 16, 2024). *Breaking Symmetry When Training Transformers.* DOI: `10.48550/arXiv.2402.05969`. arXiv: `2402.05969[cs]`.

Erklärung der Urheberschaft

Hiermit versichere ich an Eides statt, dass ich die vorliegende Masterthesis im Studiengang Intelligent Adaptive Systems selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel - insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ort, Datum	Unterschrift

Erklärung zur Veröffentlichung

Ich stimme der Einstellung der Masterthesis in die Bibliothek des Fachbereichs Informatik zu.

Ort, Datum	Unterschrift