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

Length generalization in sequence modeling tasks is a fundamental challenge in deep learning, particularly in algorithmic problems such as integer addition. While humans can effortlessly generalize addition to numbers of arbitrary length, transformer models often struggle with out-of-distribution (OOD) generalization to sequences longer than those seen during training. Understanding the underlying reasons for this limitation is crucial for advancing the capabilities of neural networks in algorithmic reasoning and compositional generalization.

1.2 Problem Statement

Despite the success of transformers in various domains, their ability to perform multi-digit integer addition with length generalization remains limited. Prior work suggests that positional encoding schemes play a significant role in this failure. Standard absolute positional encodings do not provide the necessary alignment cues for the model to correctly match digits by their place value, leading to errors when extrapolating to longer sequences. Alternative methods, such as the Abacus positional encoding, have been proposed but often involve task-specific modifications that may not generalize to other problems.

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, we aim to address the following research questions:

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

- **How** do different positional encoding schemes affect the model's ability to perform addition correctly on longer sequences?
- **How** does the inclusion of sub-task data (e.g., carry detection, digit alignment) influence the model's compositional learning and generalization capabilities?
- **How** can mechanistic interpretability techniques be applied to understand the internal representations and failure modes of transformer models in the context of integer addition?

By exploring these questions, we aim to gain insights into the limitations of current transformer architectures and positional encoding methods, and to identify principles that could lead to improved generalization in algorithmic tasks.

Chapter 2

Background

TODO: *rewrite*

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

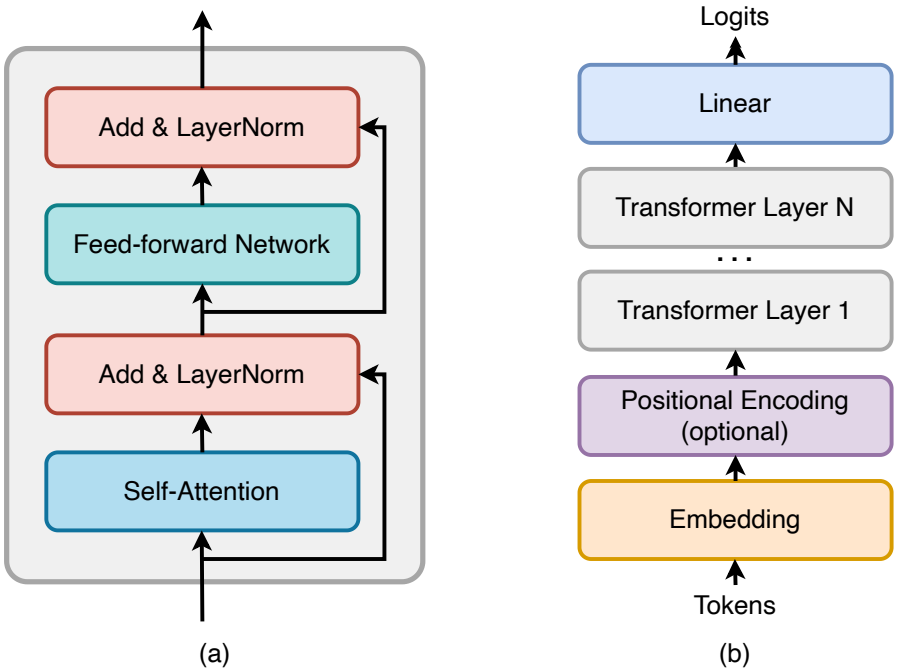

Figure 2.1: (a) A single Transformer layer, consisting of multi-head self-attention, feed-forward network (MLP), and layer normalization. (b) A decoder-only transformer model. In the decoder, the self-attention mechanism has a causal mask to prevent attending to future tokens.

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 sublayers: 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 sublayer: 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.

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.

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 re-

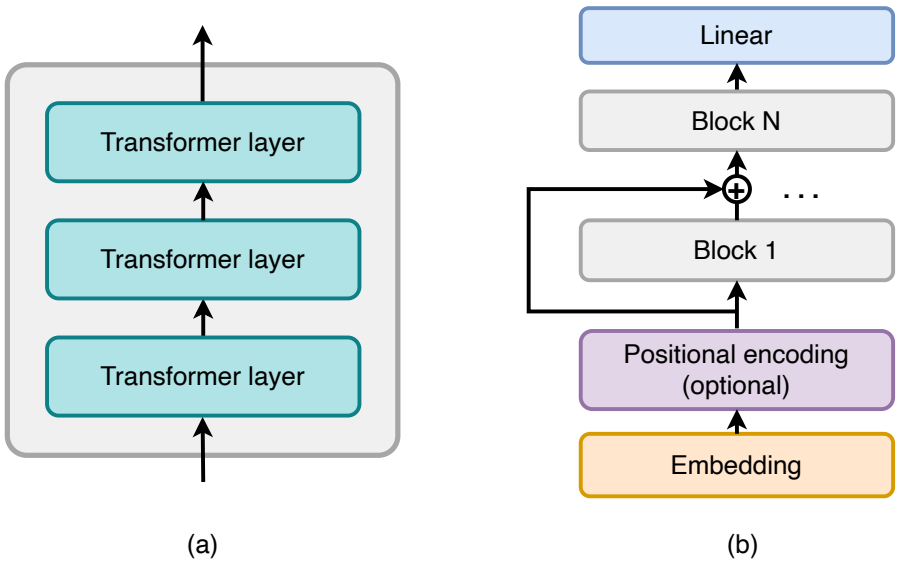

Figure 2.2: (a) A block of multiple transformer layers used in the looped transformer. (b) Looped decoder-only transformer architecture. The input injection mechanism adds skip connections (arrow around blocks) from the original input sequence to the input of each block.

cent 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:

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

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).

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

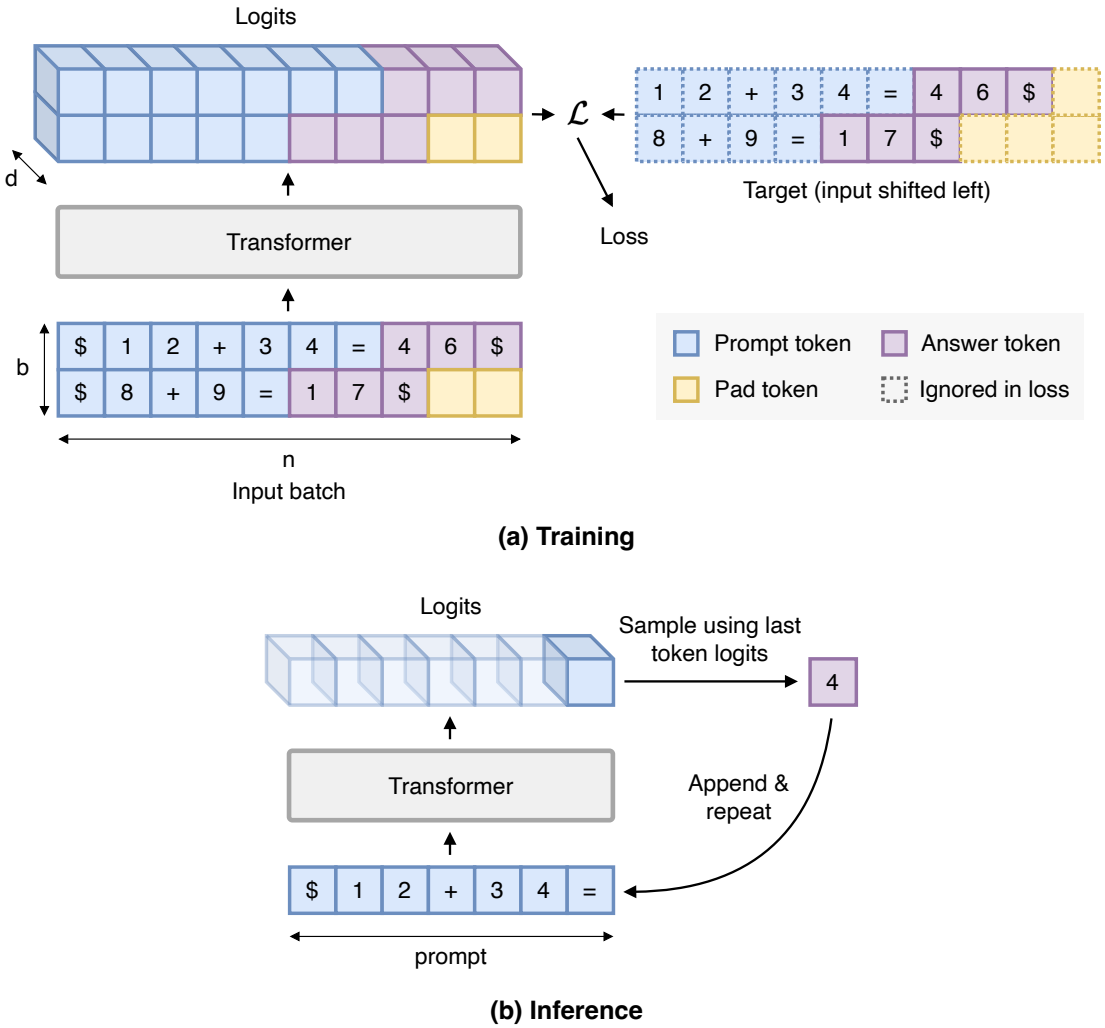

Figure 2.3: Training and inference setup for Transformer models on the arithmetic task. During training (subfigure a), the padded input batch is passed through the model, and the loss is computed by comparing the predicted logits with the target sequence. Unlike regular unsupervised learning, the loss from non-answer tokens is masked out. 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.

Activation Patching Activation patching (F. Zhang and Nanda 2023) is a causal analysis technique where activations from a run of the model are replaced by activations from a different context, allowing to observe how this change affects the model's predictions. By selectively patching different layers or tokens, it becomes possible to identify which parts of the model are responsible for specific behaviors, such as copying repeated sequences or in-context learning. For instance, patching activations in the residual stream can reveal the contribution of circuits like *induction heads*, which are attention heads that predict repeated sequences (Olsson et al. 2022).

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.

2.3 Realizability

In the context of Transformers, realizability 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.

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.

TODO: *rewri summary*

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 al-

ready 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.

TODO: *cite Hu kes et al. 2020 a expand definition*

The work of Hupkes et al. 2020 provides a theoretical framework for understanding compositional generalization in neural networks. They propose tests to investigate whether models systematically recombine known parts and rules, generalize to longer sequences, and favor rules over exceptions during training. This framework is relevant to our analysis of how transformers learn and generalize in arithmetic tasks.

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, M. 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 a domain-specific language describing the transformer architecture) 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.

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, 1.124x (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.

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 Data Formatting and Preprocessing

To investigate the impact of data formatting on length generalization, we experimented with various input representations:

- **Standard Format**: Direct representation of numbers without any modifications.
- **Reversed Operands**: Reversing the digits of the operands to test positional alignment (e.g., transforming "123 + 456" to "321 + 654").
- **Random Spaces**: Introducing random spaces between digits to disrupt positional patterns and assess the model's robustness.
- **Scratchpad**: Incorporating a chain-of-thought by including intermediate computational steps before the final answer.
- **Sub-task Annotations**: Augmenting the data with labels for sub-tasks such as carry detection and digit alignment to encourage compositional learning.

4.2 Positional Encoding Schemes

We evaluated multiple positional encoding methods to understand their effect on digit alignment and generalization:

- **Absolute Positional Encoding**: Both learned and sinusoidal encodings assigning unique positions to each token.
- **Randomized Positional Encoding**: Randomly assigning positions to examine the model's reliance on position information.
- **Relative Positional Encoding**: Encoding positions relative to each other to capture distance information.

- **No Positional Encoding (NoPE)**: Omitting positional information to test the model's ability to infer positions implicitly.
- **ALiBi Encoding**: Using attention linear biases to encode position without additional embeddings.
- **Contextual Positional Encoding**: Incorporating context-dependent position information.
- **FIRE Encoding**: A frequency-based encoding aiming to capture longer-range dependencies.
- **Abacus Encoding**: Assigning positions based on digit indices within operands to enhance digit alignment.

4.3 Model Architecture and Training Setup

We utilized decoder-only transformer models with varying configurations:

- **Model Dimensions**: Tested dimensions from 64 to 1536 in increments of 64 to assess the effect of model capacity.
- **Number of Layers and Heads**: Fixed at six layers with standard attention head configurations.
- **Training Data**: Generated datasets with sizes of 10K, 100K, 1M, and 10M examples, training on numbers with 1–7 and 9 digits to evaluate OOD generalization to unseen lengths.
- **Sub-task Training**: Compared models trained on pure addition tasks versus those trained on mixed datasets including sub-tasks.

4.4 Mechanistic Interpretability Analysis

To understand *why* certain positional encodings like Abacus lead to better generalization, we performed mechanistic interpretability studies:

- **Activation Pattern Analysis**: Examined neuron activations to identify patterns corresponding to digit alignment and carry operations.
- **Attention Weight Visualization**: Analyzed attention distributions to see how models focus on relevant digits during computation.
- **Comparative Studies**: Compared models trained with and without sub-task data to observe differences in internal representations.

Chapter 5

Conclusion

5.1 Summary of Findings

Our investigation revealed that the failure of transformer models with standard absolute positional encodings to generalize integer addition to longer sequences is primarily due to their inability to align digits correctly across varying lengths. Positional encodings that do not emphasize digit-wise alignment hinder the model's capacity to extrapolate learned addition mechanisms to longer numbers.

Positional encoding schemes like Abacus significantly improve length generalization by providing explicit alignment cues, effectively informing the model of the positional correspondence between digits in the operands and solving the digit-wise alignment sub-task for the model. Incorporating sub-task data further enhances the model's compositional learning, allowing it to build circuits that internalize intermediate computational steps such as carry detection and digit addition.

5.2 Insights and Contributions

Through mechanistic interpretability, we uncovered that models utilizing effective positional encodings develop internal circuits that mirror the step-by-step process of human addition. Our findings contribute to a deeper understanding of how positional information and data formatting influence the algorithmic reasoning abilities of transformer models.

5.3 Future Work

Future research could explore the generalization of these findings to other algorithmic tasks requiring length extrapolation. Investigating alternative architectures or hybrid models that inherently capture positional hierarchies may also yield improved generalization without task-specific positional encodings. Additionally, refining mechanistic interpretability techniques could provide more granular insights into the learning dynamics of transformer models.

Appendix A

Nomenclature

Symbol	Description
SOTA	State-of-the-art
MLP	Multi-layer perceptron
SGD	Stochastic gradient descent
FFNN	Feed-forward neural 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

Table A.1: Table of nomenclature

Bibliography

Achiam, OpenAI Josh et al. (Mar. 15, 2023). "GPT-4 Technical Report". In: URL: https://www.semanticscholar.org/paper/GPT-4-Technical-Report-Achiam-Adler/163b4d6a79a5b19af88b8585456363340d9efd04 (visited on 09/22/2024).

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]. URL: http://arxiv.org/abs/1607.06450 (visited on 10/04/2024).

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (Sept. 1, 2014). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *CoRR*. URL: https://www.semanticscholar.org/paper/Neural-Machine-Translation-by-Jointly-Learning-to-Bahdanau-Cho/fa72afa9b2cbc8f0d7b05d52548906610ffbb9c5 (visited on 10/03/2024).

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. URL: https://www.pnas.org/doi/10.1073/pnas.1903070116 (visited on 09/29/2024).

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. URL: https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html (visited on 09/22/2024).

Bulatov, Aydar, Yury Kuratov, and Mikhail Burtsev (Dec. 2022). "Recurrent Memory Transformer". en. In: *Advances in Neural Information Processing Systems* 35, pp. 11079–11091. (Visited on 05/06/2024).

Burtsev, Mikhail S. et al. (Feb. 2021). *Memory Transformer*. arXiv:2006.11527 [cs]. DOI: 10.48550/arXiv.2006.11527. URL: http://arxiv.org/abs/2006.11527 (visited on 05/06/2024).

Csordás, Róbert (2023). "Systematic generalization in connectionist models". In: URL: https://sonar.ch/global/documents/326205 (visited on 05/13/2024).

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. URL: https://openreview.net/forum?id=KBQP4A_J1K (visited on 10/09/2024).

Dave, Neisarg et al. (May 2024). "Investigating Symbolic Capabilities of Large Language Models". In: URL: https://www.semanticscholar.org/paper/Investigating-Symbolic-Capabilities-of-Large-Models-Dave-Kifer/cb1addf9cefe4e96763d28437f72a3d3cbfa7225 (visited on 06/15/2024).

Dehghani, Mostafa et al. (Sept. 2018). "Universal Transformers". en. In: URL: https://openreview.net/forum?id=HyzdRiR9Y7 (visited on 02/17/2024).

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. URL: https://aclanthology.org/N19-1423 (visited on 09/29/2024).

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. URL: http://arxiv.org/abs/2310.11984 (visited on 02/17/2024).

Dziri, Nouha et al. (Nov. 2023). "Faith and Fate: Limits of Transformers on Compositionality". en. In: URL: https://openreview.net/forum?id=Fkckkr3ya8 (visited on 02/17/2024).

Elhage, Nelson et al. (2021). "A Mathematical Framework for Transformer Circuits". In: *Transformer Circuits Thread*. https://transformer-circuits.pub/2021/framework/inde

Fan, Ying et al. (Sept. 25, 2024). *Looped Transformers for Length Generalization*. DOI: 10.48550/arXiv.2409.15647. arXiv: 2409.15647[cs]. URL: http://arxiv.org/abs/2409.15647 (visited on 09/27/2024).

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. URL: http://arxiv.org/abs/2405.00208 (visited on 05/06/2024).

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. URL: http://arxiv.org/abs/2310.02226 (visited on 07/15/2024).

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. URL: https://aclanthology.org/2022.findings-emnlp.99 (visited on 10/09/2024).

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. URL: http://ieeexplore.ieee.org/document/7780459/ (visited on 10/05/2024).

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. URL: https://jair.org/index.php/jair/article/view/11674 (visited on 09/10/2024).

Jelassi, Samy et al. (June 2023). *Length Generalization in Arithmetic Transformers*. arXiv:2306.15400 [cs]. DOI: 10.48550/arXiv.2306.15400. URL: http://arxiv.org/abs/2306.15400 (visited on 02/17/2024).

Kazemnejad, Amirhossein et al. (Nov. 2023). "The Impact of Positional Encoding on Length Generalization in Transformers". en. In: URL: https://openreview.net/forum?id=Drrl2gcjzl (visited on 02/17/2024).

Krubiński, Mateusz (Oct. 2023). "Basic Arithmetic Properties in the Space of Language Model Prompts". en. In: URL: https://openreview.net/forum?id=RCiRtdERCW (visited on 09/09/2024).

Lee, Nayoung et al. (Oct. 2023). "Teaching Arithmetic to Small Transformers". en. In: URL: https://openreview.net/forum?id=YfhuG7xHQ8&referrer=%5Bthe%20profile%20of%20Jason%20D.%20Lee%5D(%2Fprofile%3Fid%3D~Jason_D._Lee1) (visited on 02/17/2024).

Li, Shanda et al. (Mar. 2024). *Functional Interpolation for Relative Positions Improves Long Context Transformers*. en. arXiv:2310.04418 [cs]. URL: http://arxiv.org/abs/2310.04418 (visited on 06/14/2024).

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. URL: http://arxiv.org/abs/2402.12875 (visited on 09/16/2024).

Loshchilov, Ilya and Frank Hutter (Sept. 27, 2018). "Decoupled Weight Decay Regularization". In: International Conference on Learning Representations. URL: https://openreview.net/forum?id=Bkg6RiCqY7 (visited on 10/04/2024).

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. URL: http://arxiv.org/abs/2405.17399 (visited on 06/02/2024).

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. URL: https://dx.doi.org/10.1088/1742-5468/ac3a74 (visited on 09/24/2024).

Nanda, Neel et al. (Sept. 2022). "Progress measures for grokking via mechanistic interpretability". en. In: URL: https://openreview.net/forum?id=9XFSbDPmdW (visited on 04/09/2024).

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. URL: http://arxiv.org/abs/2402.14735 (visited on 02/26/2024).

nostalgebraist (Aug. 31, 2020). "interpreting GPT: the logit lens". In: URL: https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens (visited on 10/11/2024).

Olsson, Catherine et al. (Sept. 2022). *In-context Learning and Induction Heads*. arXiv:2209.11895 [cs]. DOI: 10.48550/arXiv.2209.11895. URL: http://arxiv.org/abs/2209.11895 (visited on 04/07/2024).

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. URL: https://openreview.net/forum?id=R8sQPpGCv0 (visited on 09/27/2024).

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. URL: https://aclanthology.org/2023.findings-emnlp.378 (visited on 09/09/2024).

Quirke, Philip and Fazl Barez (Oct. 2023). "Understanding Addition in Transformers". en. In: URL: https://openreview.net/forum?id=rIx1YXVWZb (visited on 03/11/2024).

Radford, Alec and Karthik Narasimhan (2018). "Improving Language Understanding by Generative Pre-Training". In: URL: https://www.semanticscholar.org/paper/Improving-Language-Understanding-by-Generative-Radford-Narasimhan/cd18800a0fe0b668a1cc19f2ec95b5003d0a5035 (visited on 10/05/2024).

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. URL: http://arxiv.org/abs/2305.16843 (visited on 05/13/2024).

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. URL: https://aclanthology.org/N18-2074 (visited on 04/03/2024).

Shen, Ruoqi et al. (Nov. 22, 2023). *Positional Description Matters for Transformers Arithmetic*. DOI: 10.48550/arXiv.2311.14737. arXiv: 2311.14737. URL: http://arxiv.org/abs/2311.14737 (visited on 10/11/2024).

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. URL: https://www.sciencedirect.com/science/article/pii/S0925231223011864 (visited on 06/17/2024).

Vaswani, Ashish et al. (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc. (Visited on 02/17/2024).

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. URL: https://openreview.net/forum?id=NpsVSN6o4ul (visited on 10/11/2024).

Wang, Xuezhi and Denny Zhou (May 23, 2024). *Chain-of-Thought Reasoning Without Prompting*. DOI: 10.48550/arXiv.2402.10200. arXiv: 2402.10200[cs]. URL: http://arxiv.org/abs/2402.10200 (visited on 09/22/2024).

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. URL: https://proceedings.mlr.press/v139/weiss21a.html (visited on 10/11/2024).

Yang, Liu et al. (Oct. 13, 2023). "Looped Transformers are Better at Learning Learning Algorithms". In: The Twelfth International Conference on Learning Representations. URL: https://openreview.net/forum?id=HHbRxoDTxE (visited on 10/03/2024).

Yehudai, Gilad et al. (July 2024). *When Can Transformers Count to n?* arXiv:2407.15160 [cs] version: 1. DOI: 10.48550/arXiv.2407.15160. URL: http://arxiv.org/abs/2407.15160 (visited on 09/06/2024).

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. URL: http://arxiv.org/abs/2403.09629 (visited on 07/15/2024).

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. (Visited on 07/31/2024).

Zhang, Fred and Neel Nanda (Oct. 2023). "Towards Best Practices of Activation Patching in Language Models: Metrics and Methods". en. In: URL: https://openreview.net/forum?id=Hf17y6u9BC (visited on 02/17/2024).

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. URL: https://openreview.net/forum?id=S5wmbQc1We (visited on 09/29/2024).

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. URL: http://arxiv.org/abs/2310.16028 (visited on 02/17/2024).

Zhou, Yongchao et al. (Feb. 2024). *Transformers Can Achieve Length Generalization But Not Robustly*. arXiv:2402.09371 [cs]. URL: http://arxiv.org/abs/2402.09371 (visited on 02/17/2024).

Zuo, Chunsheng and Michael Guerzhoy (June 16, 2024). *Breaking Symmetry When Training Transformers*. DOI: 10.48550/arXiv.2402.05969. arXiv: 2402.05969[cs]. URL: http://arxiv.org/abs/2402.05969 (visited on 10/06/2024).

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