ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Artificial Intelligence in Industry

ON AUTOREGRESSIVITY IN GENERATIVE MODELS

CANDIDATE

Samuele Marro

SUPERVISOR

Prof. Michele Lombardi

CO-SUPERVISORS

Dott. Emanuele La Malfa

Dott. Davide Evangelista

Academic year 2023-2024

Session 1st

To my family, my friends, my mentors,

and all the people who made me the person I am today.

Abstract

We study diffusion models and causal transformers under the same lens by treating both architectures as discrete approximations of continuous stochastic processes. To do so, we introduce Continuous Causal Transformers (CCTs), a time- and space-continuous generalization of causal transformers, and provide qualitative evidence showing that vanilla causal transformers implicitly approximate CCTs. We then introduce Structured Autoregressivity, a collection of five properties that are shared by diffusion models and causal transformers, and show how they emerge naturally from our analysis. Finally, we describe the implications of our framework, identifying research directions for the design of both generative and non-generative models.

Contents

1 Introduction **1**

2 Background **3**

2.1 Diffusion Models 3

2.1.1 Denosing Diffusion Implicit Models 4

2.1.2 Diffusion Models as Discretizations of Score-Based SDEs . . . 5

2.2 Transformers and Causal Language Modeling 6

3 Transformers Are Implicitly Continuous **9**

3.1 Preliminaries 10

3.1.1 Notation 10

3.1.2 Continuous Softmax 11

3.2 Continuous Causal Transformers 11

3.2.1 Continuous Embedding 12

3.2.2 Continuous Positional Encoding 12

3.2.3 Continuous Self-Attention 12

3.2.4 Continuous Causal Self-Attention 13

3.2.5 Continuous Transformer Block 13

3.2.6 Continuous Causal Language Modeling Head 14

3.2.7 Continuous Causal Language Modeling 14

3.2.8 Duration . . . 15
3.3 Qualitative Evidence . . . 16
3.3.1 Experimental Setup . . . 16
3.3.2 Experiments . . . 18
3.4 Implications and Potential Extensions . . . 24
3.4.1 The Implicit Dynamic . . . 25
3.4.2 Beyond Decoder-Only Causal Modeling . . . 26
3.4.3 Integrating Over the Embedding Space . . . 27

4 Structured Autoregressivity **29**
4.1 Task Subdivision and Weight Sharing . . . 30
4.2 Teacher Forcing . . . 31
4.3 Frequent Supervision . . . 32
4.4 Complete Supervision . . . 33

5 Next Steps **35**
5.1 Ablation Study . . . 35
5.1.1 Task Subdivision . . . 36
5.1.2 Weight Sharing . . . 37
5.1.3 Teacher Forcing . . . 37
5.1.4 Frequent Supervision . . . 39
5.1.5 Complete Supervision . . . 40
5.2 Implications for Model Design . . . 40
5.2.1 Sampling Strategies . . . 40
5.2.2 Synthetic Autoregressivity . . . 41
5.2.3 Beyond Generation . . . 42

6 Conclusion **44**

Bibliography **46**

Acknowledgements **50**

A Additional Qualitative Results **51**

List of Figures

3.1 Impact of embedding interpolation on the predicted output probability . 20
3.2 Visualization of the duration reduction experiment 21
3.3 Impact of reducing the duration of selected tokens on the predicted output probability 22
3.4 Visualization of the translation invariance experiment 23
3.5 Visualization of the scale invariance experiment 24
3.6 Effect of translation on the predicted output probability 25
3.7 Effect of scaling on the predicted output probability 26
3.8 Visualization of the density invariance experiment 27
3.9 Impact of sampling density on the predicted output probability 28

A.1 Impact of embedding interpolation on the predicted output probability for eight pairs of numbers 53
A.2 Impact of reducing the duration of selected tokens on the predicted output probability for eight pairs of numbers 54
A.3 Impact of translation on the predicted output probability for eight pairs of numbers 55
A.4 Impact of scaling on the predicted output probability for eight pairs of numbers 56

A.5 Impact of sampling density on the predicted output probability for eight pairs of numbers 57

Chapter 1

Introduction

Diffusion models [14, 23] have quickly established themselves as the de facto standard for image generation. At the same time, the development of transformers [29] has enabled the rise of Large Language Models (LLMs), with several architectures achieving state-of-the-art performances in a wide range of language and cognitive tasks [1, 2, 4, 26, 27]. However, the specific reasons for why both of these architectures outperform older approaches in their respective tasks are still unclear. In this (mostly) theoretical thesis, we propose an unorthodox approach: we study both causal transformers and diffusion models under the lens of continuous stochastic processes. Specifically, we show that both diffusion models and causal transformers can be seen as discretizations of continuous stochastic processes, which are trained in settings with five key properties (namely Task Subdivision, Weight Sharing, Teacher Forcing, Frequent Supervision, and Complete Supervision). We name the collection of these properties *Structured Autoregressivity* and show through a preliminary analysis how they arise naturally from considerations on stochastic process approximation. While the relation between diffusion models and continuous stochastic processes is known in the literature [24], as part of our analysis we complement this result by introducing Continuous Causal Transformers (CCTs) a continuous-state and continuous-time extension of causal transformers for language

modelling. Our qualitative analyses show that not only pretrained causal transformers can be effortlessly treated as CCTs, but that doing so leads to behaviors consistent with human intuition about language. Overall, while our results are preliminary, they pave the way for several research directions, such as adapting diffusion sampling strategies to language modeling (and vice-versa), improving language model performance through synthetic autoregressivity, and even applying lessons learned from generative modeling to discriminative tasks.

Our work is structured as follows. First, we provide a brief background on diffusion models and causal transformers (Chapter 2) and introduce a continuous extension of the latter (Chapter 3). Then, we detail the properties that constitute Structured Autoregressivity (Chapter 4). Finally, we outline future research directions that emerge as the result of our analysis (Chapter 5) and offer our closing thoughts (Chapter 6).

Chapter 2

Background

2.1 Diffusion Models

Diffusion models are autoregressive image generation models that operate through a process of progressive denoising.

Specifically, consider a data distribution $q(\boldsymbol{x}_0)$. Training a diffusion model involves optimizing a parameter vector $\boldsymbol{\theta}$ such that the distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}_0)$ approximates $q(\boldsymbol{x}_0)$ as closely as possible. In Denoising Diffusion Probabilistic Models (DDPMs) [14], the generative distribution is modeled as

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}_0) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}_{0:T}) d\boldsymbol{x}_{1:T}, \tag{2.1}$$

where T is a hyperparameter and $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{0:T})$ is defined as

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}_{0:T}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}_T) \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t), \tag{2.2}$$

with $p_{\boldsymbol{\theta}}(\boldsymbol{x}_T) = \mathcal{N}(\boldsymbol{x}_T|\mathbf{0}; I)$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{x}_{t-1}|\mu_{\boldsymbol{\theta}}(\boldsymbol{x}_t, \alpha_t); \sigma_t^2 I)$.

We optimize $\boldsymbol{\theta}$ such that $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{0:T})$ approximates $q_{\boldsymbol{\theta}}(\boldsymbol{x}_{0:T})$, which is a Markov chain

of the form

$$q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = \mathcal{N}\left(\boldsymbol{x}_t \left| \sqrt{\frac{\alpha_t}{\alpha_{t-1}}}\boldsymbol{x}_{t-1}; \left(1 - \frac{\alpha_t}{\alpha_{t-1}}\right) I\right.\right), \tag{2.3}$$

where $\{\alpha_t\}_{t\in[0,T]}$ is a decreasing sequence in the interval $[0, 1]$ that denotes a *diffusion schedule*. We use as loss function the Evidence Lower BOund (ELBO) loss, i.e.

$$\mathcal{L}(\boldsymbol{\theta}) = -\mathbb{E}_{q(\boldsymbol{x}_{0:T})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_{0:T}) - \log q(\boldsymbol{x}_{1:T})\right], \tag{2.4}$$

which can be rewritten as

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{t=1}^{T} \gamma_t \mathbb{E}_{q(\boldsymbol{x}_t|\boldsymbol{x}_0)} \left\|\mu_{\boldsymbol{\theta}}(\boldsymbol{x}_t, \alpha_t) - \tilde{\mu}(\boldsymbol{x}_t, \boldsymbol{x}_0)\right\|_2^2. \tag{2.5}$$

In other words, the model needs to minimize the weighted mean square error between the reconstructed image at the t-th step and the ground truth image obtained by the process denoted by q.

In practice, this approximation is often learned using a U-Net [22], a Convolutional Neural Network that uses a series of downsampling and upsampling steps to learn mappings at various scales.

2.1.1 Denosing Diffusion Implicit Models

Denoising Diffusion Implicit Models (DDIMs) are a variant of DDPMs that uses a non-Markovian diffusion process defined as

$$q_\sigma(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0) = q_\sigma(\boldsymbol{x}_T|\boldsymbol{x}_0) \prod_{t=2}^{T} q_\sigma(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0), \tag{2.6}$$

where

$$q_\sigma(\boldsymbol{x}_T|\boldsymbol{x}_0) = \mathcal{N}\left(\boldsymbol{x}_T \left| \sqrt{\alpha_T}\boldsymbol{x}_0, (1 - \alpha_T)I\right.\right), \tag{2.7}$$

$$q_\sigma(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) = \mathcal{N}\left(\boldsymbol{x}_{t-1} \left| \mu_{\sigma_t}(\boldsymbol{x}_0, \alpha_{t-1}); \sigma_t^2 I\right.\right), \tag{2.8}$$

and with $\mu_{\sigma_t}(\boldsymbol{x}_0, \alpha_{t-1})$ defined as

$$\mu_{\sigma_t}(\boldsymbol{x}_0, \alpha_{t-1}) = \sqrt{\alpha_{t-1}}\boldsymbol{x}_0 + \sqrt{1 - \alpha_{t-1} - \sigma_t^2} \cdot \frac{\boldsymbol{x}_t - \sqrt{\alpha_t}\boldsymbol{x}_0}{\sqrt{1 - \alpha_t}}. \tag{2.9}$$

As a consequence of our choice of $q_\sigma(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)$, we have

$$\boldsymbol{x}_t = \sqrt{\alpha_t}\boldsymbol{x}_0 + \sqrt{1 - \alpha_t}\boldsymbol{\epsilon}_t, \tag{2.10}$$

with $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\boldsymbol{\epsilon}_t|\mathbf{0}; I)$. Note that by setting $\sigma_t = 0$ in Equation (2.8) the process becomes deterministic.

From a practical point of view, the parameters of $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$ can be optimized by training a neural network $\boldsymbol{\epsilon}_{\boldsymbol{\theta},t}(\boldsymbol{x}_t, \alpha_t)$ to estimate the noise $\boldsymbol{\epsilon}_t$ that was added to $\boldsymbol{x}_0$ to obtain $\boldsymbol{x}_t$.

2.1.2 Diffusion Models as Discretizations of Score-Based SDEs

A particularly relevant result for our analysis is that of [17, 24], who found that the stochastic mapping from $\boldsymbol{x}_T$ to $\boldsymbol{x}_0$ follows a score-based continuous Stochastic Differential Equation (SDE) whose dynamic is determined by terms related to the gradient of the ground-truth probability distribution of the data. The sampling procedure for DDIMs can thus be obtained by discretizing the deterministic probabilistic flow associated with this dynamic. As a consequence, the training process of a DDIM leads to an approximation of the score function of the ground-truth distribution.

We will complement this result by showing that causal transformers can be treated as discretizations of continuous stochastic processes as well (Chapter 3).

2.2 Transformers and Causal Language Modeling

Transformers are a class of generative models designed to model sequences. Unlike previous architectures, transformers take the entire sequence as input, which improves their capacity to learn long-term dependencies. In particular, causal transformers model sequences in an incremental fashion, predicting the $(t+1)$-th element $\boldsymbol{x}_{t+1}$ given all the previous elements $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t$. For the sake of simplicity, we focus exclusively on decoder-only transformers.

Self-Attention The core of a transformer is the self-attention mechanism, a module that takes as input the entire sequence at once and learns to identify its "important" parts. Given a sequence of vectors $X \in \mathbb{R}^{n \times d_{in}}$ and a hyperparameter d_{model}, the output $Y \in \mathbb{R}^{n \times d_{model}}$ of the transformer is defined as follows:

$$Y = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_{in}}}\right)V. \tag{2.11}$$

$Q, K, V \in \mathbb{R}^{n \times d_{model}}$ are defined as

$$\begin{aligned} Q &= XW^{Q^\top} \\ K &= XW^{K^\top} \\ V &= XW^{V^\top}, \end{aligned} \tag{2.12}$$

with $W^Q, W^K, W^V \in \mathbb{R}^{d_{model} \times d_{in}}$. A variant of self-attention, named *multihead self-attention*, involves using different sets of W^Q, W^K and W^V in order to extract different features from the input sequence.

Causal Attention Mask The causal attention mask is a mask applied to the input of the softmax such that, for a given element $\boldsymbol{x}_t$, the self-attention only considers the sequence $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t$. This prevents the transformer from "looking at the future" (i.e. considering

inputs with time steps greater than t), which would lead to incorrect behavior at inference time (since future time steps are not available when generating a new sequence). Note that there are two main ways to implement the causal mask: as a *multiplicative mask* (with values in $\{0, 1\}$) that is multiplied with the exponentiated logits in the softmax, or as an *additive mask* (with values in $\{-\infty, 0\}$) that is added directly to the logits before applying the softmax.

Transformer Block The transformer block is defined as the concatenation of the following modules:

- Multihead self-attention with an additive skip connection;
- Some form of normalization, usually Layer Normalization [4] or Root Mean Square Normalization [31];
- A feedforward layer with an additive skip connection;
- Normalization again.

Tokenization and Input Embedding When using transformers for language data, the text is subdivided into *tokens*, where each token is part of a set (named *vocabulary*). Since these tokens are categorical, they are mapped onto a continuous space $\mathbb{R}^{emb}$ by the *input embedding*, which is often learned as part of the training process.

Positional Encoding Since transformers do not preserve positional information, it is necessary to add it in the form of positional encoding. The positional encoding, as defined in [29], is a matrix $PE \in \mathbb{R}^{T \times d_{emb}}$ such that

$$\begin{aligned} PE_{pos,2i} &= \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \\ PE_{pos,2i+1} &= \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right). \end{aligned} \tag{2.13}$$

which is then summed with the input embedding. However, there exist other forms of positional encoding as well, such as Rotary Positional Embedding (RoPE) [25].

Casual Language Modeling Head The causal language modeling head is a module that takes the tth output of a sequence and applies a feedforward layer having as output a vector of logits (one for each token in the vocabulary).

Causal Transformer A causal transformer is thus defined as the concatenation of the following modules:

- Input embedding;
- Positional encoding;
- One or more transformer blocks with a causal attention mask;
- Causal language modeling head.

Given the sequence $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t$, the model is thus trained to predict $\boldsymbol{x}_{t+1}$. At inference time, the sequence can be generated by fixing the first element (which is usually a special token, named beginning-of-string or BOS token) and generating for each step t the element $\boldsymbol{x}_{t+1}$ given the sequence $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t$.

Chapter 3

Transformers Are Implicitly Continuous

Before studying causal transformers and diffusion models under the same lens, we first need to address an evident problem.

Diffusion models are defined as approximations of continuous-time processes over a continuous space: for each time step $t \in [0, T] \subsetneq \mathbb{R}$, the state of the process is known, and in particular it is defined as the linear combination of an image and Gaussian noise, making it inherently continuous. Therefore, while in practice diffusion models are trained by employing a discretization of this process (using a finite number of time steps and discretizing the state as a vector of floating-point values), they can be naturally studied as continuous processes.

On the other hand, language models are discrete-time processes over a discrete space: whichever fundamental unit of language is chosen (word, subword or character), there are only a finite number of them, and each unit follows the other in a discrete fashion (e.g. with character-level models, in the word *hello* there are no intermediate characters between *h* and *e*).

However, we will show that causal transformers can be treated as a discretization of

a continuous-time continuous-space model, which we non-creatively name Continuous Causal Transformer (CCT). CCTs are a very simple extension of vanilla causal transformers: the biggest change involves replacing the sums across the sequence length with integrals. They are also not the first example of continuous transformers (see for instance [9, 7, 19]), and in fact they can be seen as a variant of ContiFormers [7].

Instead, CCTs are designed to study the implicit dynamic of existing language models through a continuous lens. In fact, as we will show empirically, when treating pretrained causal transformers as CCTs they follow a behavior that is consistent with human intuition of language (Section 3.3).

3.1 Preliminaries

We provide in this section a quick overview of some definitions that will be useful for our analysis.

3.1.1 Notation

Real-Valued Function We use $[\boldsymbol{s}^{(t)}]_{t\in[T_{start},T_{end}]}$ to denote a real-valued function $s(t)$ with domain $[T_{start}, T_{end}]$. We say that $T_{end} - T_{start}$ is the *length* of $[\boldsymbol{s}^{(t)}]_{t\in[T_{start},T_{end}]}$.

Ordered Finite Set We use $\{\boldsymbol{s}^{(t)}\}_{t\in X}$, with $X \subseteq \mathbb{R}$ finite, to denote an ordered finite set where each element has a corresponding time step t.

Composition We use $[f(\boldsymbol{s}^{(t)}, t)]_{t\in[T_{start},T_{end}]}$ to denote the function such that

$$\left[f\left(\boldsymbol{s}^{(t)}\right)\right]^{(\tau)}_{t\in[T_{start},T_{end}]} = f\left(\boldsymbol{s}^{(\tau)}, \tau\right). \tag{3.1}$$

Similarly, we use $\{f(\boldsymbol{s}^{(t)}, t)\}_{t\in X}$ to denote the ordered finite set such that

$$\left\{f\left(\boldsymbol{s}^{(t)}\right)\right\}_{t\in X}^{(\tau)} = f\left(\boldsymbol{s}^{(\tau)}, \tau\right). \tag{3.2}$$

3.1.2 Continuous Softmax

For a vector $\boldsymbol{x} \in \mathbb{R}^n$, the vanilla softmax function is defined as

$$\text{softmax}(\boldsymbol{x})_i = e^{x_i} / \sum_{j=1}^{n} e^{x_j}. \tag{3.3}$$

The softmax function can also be rewritten as a function that accepts an ordered finite set

$$\text{softmax}\left(\left\{\boldsymbol{x}^{(t)}\right\}_{t\in X}\right)^{(t)} = e^{\boldsymbol{x}^{(t)}} / \sum_{j\in X} e^{\boldsymbol{x}^{(j)}}. \tag{3.4}$$

Given a real-valued function $[\boldsymbol{x}^{(t)}]_{t\in[0,T]}$ with $\boldsymbol{x}^{(t)} \in \mathbb{R}$, we provide a continuous extension of the softmax function, defined as

$$\text{csoftmax}\left(\left[\boldsymbol{x}^{(t)}\right]_{t\in[0,T]}\right)^{(t)} = e^{\boldsymbol{x}^{(t)}} / \int_0^T e^{\boldsymbol{x}^{(\tau)}}\, d\tau. \tag{3.5}$$

Note that discretizing the integral using the rectangle method with $\Delta t = 1$ allows us to recover the original softmax function for $X = \{1, \ldots, T\}$.

3.2 Continuous Causal Transformers

We now provide an overview on how Continuous Causal Transformers meaningfully generalize vanilla transformers to continuous time and space.

We begin by defining a *sentence* as a real-valued function $[\boldsymbol{s}^{(t)}]_{t\in[0,T]}$, where T is the *sentence length* and $\boldsymbol{s}^{(t)} \in \mathbb{R}^{d_{emb}}$ is the *sentence unit* at time t, represented as a point in the embedding space.

We then detail, for each component of a transformer, the differences between the discrete and continuous variants.

3.2.1 Continuous Embedding

Transformers are by default capable of accepting continuous inputs in the form of embeddings (which are real-valued vectors). The only discretization that concerns the input space is the fact that vanilla transformers restrict the range of possible inputs to a discrete, potentially learnable subset of the embedding space (i.e. the vocabulary): relaxing this constraint undoes this discretization step.

3.2.2 Continuous Positional Encoding

Similarly to embeddings, the majority of commonly used positional encodings (e.g. sinusoidal encoding [29] and rotary embeddings [25]) are by default capable of accepting non-integer inputs. For this reason, the positional encoding of CCTs is the same as that of vanilla transformers.

3.2.3 Continuous Self-Attention

Let $W^Q, W^K, W^V \in \mathbb{R}^{d_{model} \times d_{in}}$ be respectively the query, key and value projections.

Let $\boldsymbol{x}^{(t)}, t \in [0, T]$ (with $\boldsymbol{x}^{(t)} \in \mathbb{R}^{d_{in}}$ be an input sentence. We define the output $[\boldsymbol{y}^{(t)}]_{t \in [0,T]}$ of the continuous self-attention as

$$\boldsymbol{y}^{(t)} = \int_0^T \text{csoftmax}\left(\left[\frac{\boldsymbol{q}^{(\varphi)} \cdot \boldsymbol{k}^{(\tau)}}{\sqrt{d_{in}}}\right]_{\varphi \in [0,T]}\right)^{(\tau)} \boldsymbol{v}^{(\tau)}\, d\tau, \tag{3.6}$$

where $\boldsymbol{q}^{(t)} = W^Q \boldsymbol{x}^{(t)}, \boldsymbol{k}^{(\tau)} = W^K \boldsymbol{x}^{(\tau)}, \boldsymbol{v}^{(\tau)} = W^V \boldsymbol{x}^{(\tau)}$.

Note that replacing the integral in Equation (3.6) with its approximation using the rectangle method with $\Delta t = 1$ leads to a form that is equivalent to the self-attention used

in vanilla transformers:

$$\boldsymbol{y}^{(t)} = \sum_{j=1}^{T} \text{softmax}\left(\left\{\frac{\boldsymbol{q}^{(t)} \cdot \boldsymbol{k}^{(i)}}{\sqrt{d_{in}}}\right\}_{i\in\{1,\dots,T\}}\right)^{(j)} \boldsymbol{v}^{(j)}. \tag{3.7}$$

3.2.4 Continuous Causal Self-Attention

The continuous causal self-attention is a variant of the continuous self-attention where, instead of integrating from 0 to T, we integrate from 0 to t. Specifically,

$$\boldsymbol{y}^{(t)} = \int_0^t \text{csoftmax}\left(\left[\frac{\boldsymbol{q}^{(\varphi)} \cdot \boldsymbol{k}^{(\tau)}}{\sqrt{d_{model}}}\right]_{\varphi\in[0,t]}\right)^{(\tau)} \boldsymbol{v}^{(\tau)}\, d\tau. \tag{3.8}$$

Again, the numerical integration of the integral in Equation (3.8) using the rectangle method with $\Delta t = 1$ is equivalent to the causal self-attention used in vanilla transformers, i.e.

$$\boldsymbol{y}^{(t)} = \sum_{j=1}^{t} \text{softmax}\left(\left\{\frac{\boldsymbol{q}^{(t)} \cdot \boldsymbol{k}^{(i)}}{\sqrt{d_{model}}}\right\}_{i\in\{1,\dots,t\}}\right)^{(j)} \boldsymbol{v}^{(j)}. \tag{3.9}$$

3.2.5 Continuous Transformer Block

The continuous transformer block is defined in the same way as the vanilla one, but with continuous self-attention instead of the vanilla one. In particular, let $[\boldsymbol{h}^{(t)}]_{t\in[0,T]}$ be the output of the continuous self-attention. Then the output of the first step of the continuous transformer block (i.e. add + normalization) is $[\boldsymbol{l}^{(t)}]_{t\in[0,T]}$, defined as

$$\boldsymbol{l}^{(t)} = \text{norm}(\boldsymbol{s}^{(t)} + \boldsymbol{h}^{(t)}), \tag{3.10}$$

where norm is the normalization function (e.g. Layer Normalization). The overall output of the continuous transformer block (i.e. after applying the second step, which involves a feedforward layer, an add step and a normalization) is the sentence $[\boldsymbol{y}^{(t)}]_{t\in[0,T]}$, defined as

$$\boldsymbol{y}^{(t)} = \text{norm}\left(\text{ReLU}(W\boldsymbol{l}^{(t)} + b) + \boldsymbol{l}^{(t)}\right), \tag{3.11}$$

where W and b are respectively the weight and bias of the feedforward layer and ReLU is the Rectified Linear Unit function.

3.2.6 Continuous Causal Language Modeling Head

We define a continuous causal language modeling head as a function that takes a processed sentence unit $\boldsymbol{r}^{(t)}$ and outputs a continuous distribution over $\mathbb{R}^{d_{emb}}$. The vanilla causal language modeling head can thus be seen as a discretization of the continuous causal language modeling head. Specifically, let $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subsetneq \mathbb{R}^{d_{emb}}$ be a finite set of embeddings (corresponding to the vocabulary). Then the output $\boldsymbol{y}$ of the vanilla causal language modeling head is a vector (with n elements) that denotes a discrete distribution over X, where $p(\boldsymbol{x}_i) = y_i$.

3.2.7 Continuous Causal Language Modeling

Putting it all together, we say that the CCT for a given t approximates the probability distribution of $\boldsymbol{s}^{(t)}$.

The training process of vanilla transformers can be seen as a discrete version of this process where the model, given the elements of the sequence with time steps less than or equal to t, is trained to predict $\boldsymbol{s}^{(t+1)}$. This is similar to how both Neural Ordinary Differential Equations (ODEs) [6] and Universal Differential Equations (UDEs) [20] are in principle capable of approximating time-continuous dynamical systems, but are trained in practice with respect to a specific time discretization. As a result, both Neural ODEs and UDEs are only capable of making predictions with respect to the discretization level used at training time.

Analogously, while we will show in Section 3.3 that it is possible to use a pretrained

vanilla causal transformer to meaningfully handle continuous input (by treating the transformer as a CCT), how such models might be coaxed into making predictions for time steps that are not one time-unit away from the last state is unclear. This matter, while not fundamental for the purposes of our analysis, nevertheless represents an important research direction, one that is deeply intertwined with determining the nature of the implicit dynamic learned by vanilla causal transformers (Section 3.4.1).

3.2.8 Duration

Reasoning about CCTs requires significant shifts in intuition, the most evident of which concerns the role of what we call *duration*. From the point of view of CCTs, a vanilla transformer (which is defined as a rectangle-based approximation of the CCT) is equivalent to integrating assuming that, for each time step $\tau \in (t-1, t]$, $\boldsymbol{s}^{(\tau)} = \text{token}_t$, where token_t is the t-th token of the original, integer-time sequence (1-indexed). In other words, vanilla transformers assume that $\boldsymbol{s}^{(t)}$ is constant (and equal to token_t) for an interval $(t-1, t]$, then that it is constant (and equal to token_{t+1}) for an interval $(t, t+1]$, and so on. Intuitively speaking, vanilla transformers thus assign to each token a *duration* of 1.

When we feed non-integer-time sequences to CCTs, all practical implementations require some degree of discretization of the integral: in practice, this involves treating the input sequence as a set of samples $X = \{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(k)}\}$ of tokens[1], each with its own corresponding time step $\{t_1, \ldots, t_k\}$, and using them as samples for the rectangle method. In other words, we are still using a finite set of samples, but their "durations" are not necessarily 1.

[1] Throughout the rest of the document, we use $\boldsymbol{z}$ to denote finite samples used to approximate the input of a CCT.

Formally speaking, we define the duration d_i of a sample $\boldsymbol{x}^{(i)}$ as

$$d_i = \begin{cases} t_i - \alpha & i = 1 \\ t_i - t_{i-1} & \text{otherwise,} \end{cases} \tag{3.12}$$

where α is the lower limit of the integral (usually 0).

As we will show in the next section, duration is not a mere quirk of rectangle-based approximations, but actually carries a semantic meaning.

Note that while we defined duration through the lens of the rectangle method, we believe that it is possible to find equivalent formulations for other approximations such as the trapezoidal rule. We leave a formalization of duration in these contexts and the implications of these alternate formulations to future work.

3.3 Qualitative Evidence

We now provide evidence for the fact that CCTs represent a meaningful generalization of vanilla transformers. To do so, we take a pretrained transformer (specifically Llama v1 7B [26]) and conduct four experiments meant to answer different facets of two questions:

- When treated as CCTs, do pretrained vanilla transformers behave in a continuous manner?
- Does the behavior of CCTs for non-discrete inputs (i.e. inputs with either continuous embeddings or continuous time) match human intuition?

3.3.1 Experimental Setup

We consider sentences of the form:

```
The sum of 24 and 13 is
```

Since 24 + 13 = 37, the causal language modeling head will predict with a high probability that the next token is *3*. As detailed in Section 3.2.8, from the point of view of CCTs we can see a sequence of tokens[2] for a vanilla transformer as a list of samples $\left\{\boldsymbol{z}^{(t)}\right\}_{t\in\{1,\dots,T\}}$ where $\boldsymbol{z}^{(t)} = \text{token}_t$. By modifying this sequence so that it corresponds to inputs that cannot be represented by vanilla transformers (e.g. using intermediate embeddings or assigning non-integer time steps to tokens), we can study the behavior of the transformer outside of its intended domain. If the training process of vanilla transformers made such models inherently discrete (which would imply that pretrained vanilla transformers cannot be meaningfully extended to CCTs), then we would expect the output of the transformer on nonstandard inputs to be completely nonsensical. Instead, we will show that these outputs are both reasonable and compatible with human intuition.

For all experiments, we conduct some form of interpolation between two extremes, mediated by an interpolation factor φ. We then treat the result of the interpolation (which is represented by a sequence of embeddings, each having its own corresponding time step) as a list of samples and use it to compute an approximation of the output of the CCT having the same weights as the vanilla causal transformer. For the sake of comparison, we always use variations of the same sum (i.e. 24 + 13). We report the results of using other pairs of numbers in Appendix A.

CCTs in Practice From an implementation perspective, extending a vanilla transformer to a CCT requires only three modifications:

- Accepting directly input embeddings, instead of taking input tokens and then computing their embedding;
- Accepting a vector of time steps, instead of computing them using `torch.arange(num_tokens)` or equivalent formulations;

[2] Throughout this section, we use the words "tokens" and "embeddings" interchangeably.

- Replacing the multiplicative causal mask with the multiplicative causal duration-based mask (as described in Section 3.2.8). If the model uses an additive mask for logits, then instead of using the multiplicative causal duration-based mask we use its elementwise natural logarithm.

Therefore, adapting an existing transformer to accept continuous inputs requires surprisingly little effort. The source code of our experiments, adapted from pyllama[3] is available at https://github.com/samuelemarro/cct-early-experiments.

3.3.2 Experiments

We now describe the specific experiments we conducted. We stress that these experiments do not provide information on the actual values of $\boldsymbol{s}^{(t)}$, but rather represent a way to qualitatively understand how transformers reason about inputs.

Transformers Reason Continuously About Embeddings We begin by studying how, assuming integer-valued tokens, transformers respond to interpolations of discrete embeddings. In particular, we interpolate linearly between the provided sentence and the following alternate sentence:

```
The sum of 24 and 31 is
```

Since this alternate sentence has the same tokenized length as the original sentence, it is possible to define straightforwardly the interpolation as

$$\boldsymbol{z}_{int}^{(i)} = interpolate\left(\boldsymbol{z}_{orig}^{(i)}, \boldsymbol{z}_{alt}^{(i)}, \varphi\right) \tag{3.13}$$

$$t_{i,int} = t_{i,orig} = t_{i,alt}, \tag{3.14}$$

[3] https://github.com/juncongmoo/pyllama

where $interpolate$ is an interpolation function and $\varphi \in [0, 1]$. For the sake of simplicity, we use linear interpolation, fully aware that it does not match semantic interpolation (i.e. it is known that the linear interpolation of two embeddings is not the embedding of the semantic interpolation):

$$interpolate\left(\boldsymbol{z}_{orig}^{(i)}, \boldsymbol{z}_{alt}^{(i)}, \varphi\right) = (1-\varphi)\boldsymbol{z}_{orig}^{(i)} + \varphi \boldsymbol{z}_{alt}^{(i)}. \tag{3.15}$$

The reason behind this choice is that introducing further complexity (e.g. by training a model to identify potentially meaningful interpolations) would risk undermining the validity of our results, since we aim to study the behavior of pretrained vanilla transformers with the fewest possible confounding factors. Nevertheless, as shown in Figure 3.1, even a simple linear interpolation carries semantic meaning for transformers: interpolating between the original and the alternate sentence leads to a smooth transition in the top probability from *3* (which is the corresponding output of the original sentence) to *5* (which is the corresponding output of the alternate sentence). Even when the probabilities of both *3* and *5* are low, the cumulative probability of all other digits greatly overshadows that of other tokens, suggesting that even if the model is uncertain about the correct output (since no token has a very high probability), it still knows that the output should be a digit. In other words, even simple interpolations in the embedding space carry semantic meaning, supporting the hypothesis that transformers inherently assign semantic meaning to the discrete embedding space outside of the subset corresponding to the vocabulary.

Transformers Reason Continuously About Duration We then study how transformers interpret the duration of a token. To do so, we progressively reduce the duration of the tokens corresponding to *1* and *3* with the following interpolation:

$$\boldsymbol{z}_{int}^{(i)} = \boldsymbol{z}_{orig}^{(i)} \tag{3.16}$$

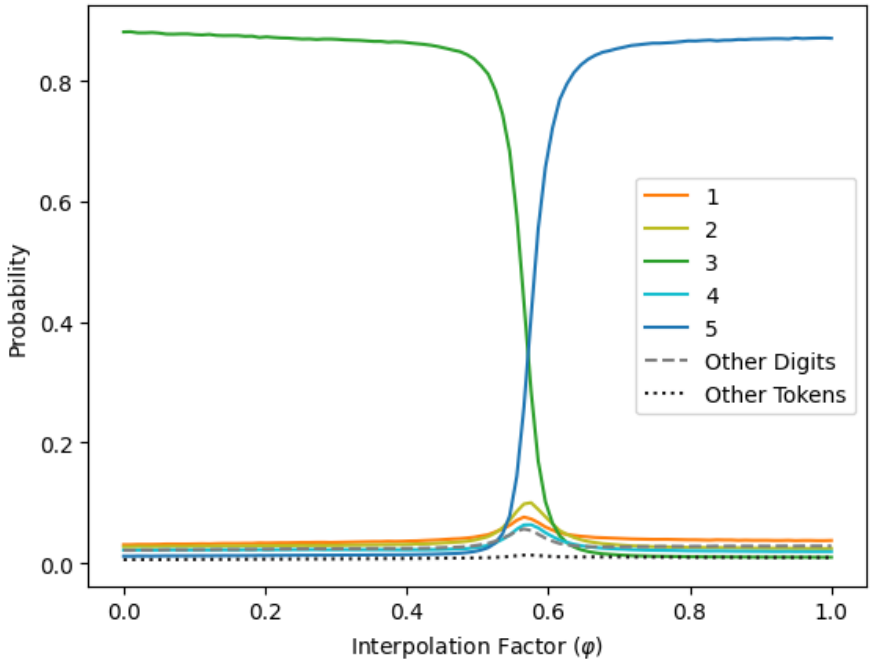


Figure 3.1: Impact of embedding interpolation on the predicted output probability. Digits that never reach a probability of at least 0.05 are merged into the category "Other Digits". All non-digit tokens are merged into "Other Tokens".

$$t_{i,int} = \begin{cases} t_{i,orig} & i < i_m \\ t_{i,orig} - \frac{1}{2}\varphi & i = i_m \\ t_{i,orig} - \varphi & i > i_m. \end{cases} \tag{3.17}$$

where i_m is the position of the token corresponding to *1*. As a consequence, the duration of each token will be:

- 1 for each token before *1*;
- $1 - \frac{1}{2}\varphi$ for *1* and *3*;
- 1 for each token after *3*.

In other words, the length of both *1* and *3* decreases (by up to 50%) as φ increases, which means that the overall duration of the number *13* goes from 2 when $\varphi = 0$ to 1

when $\varphi = 1$ (Figure 3.2). As we can see in Figure 3.3, as φ increases, the transformer indeed begins to treat *13* as a single-digit number: in fact, the sum of 24 and a single-digit number between 1 and 3 begins with 2, which is consistent with the fact that the probability of 2 increases. This result shows that transformers assign a semantic meaning to duration: a *13* with duration 2 is fundamentally different from a *13* with duration 1.

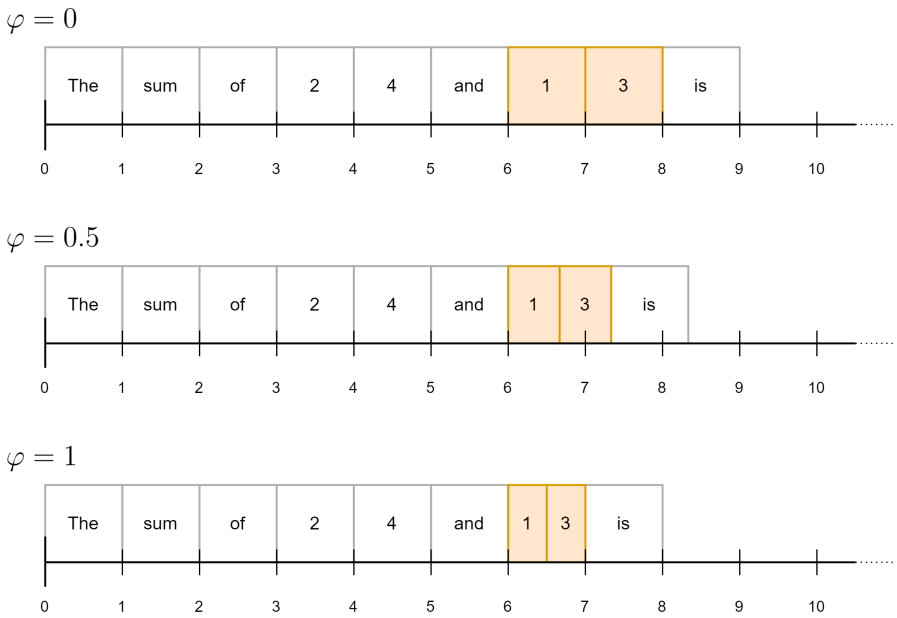


Figure 3.2: Visualization of the duration reduction experiment. Tokenization is simplified for illustrative purposes.

Transformers Are Translation-Invariant, But Not Scale-Invariant For this experiment we consider two tests (depicted in Figures 3.4 and 3.5). In the first one, we shift the time steps to the right by an amount equal to the interpolation factor, i.e. we set

$$\boldsymbol{z}_{int}^{(i)} = \boldsymbol{z}_{orig}^{(i)} \tag{3.18}$$

$$t_{i,int} = t_{i,orig} + \varphi, \tag{3.19}$$

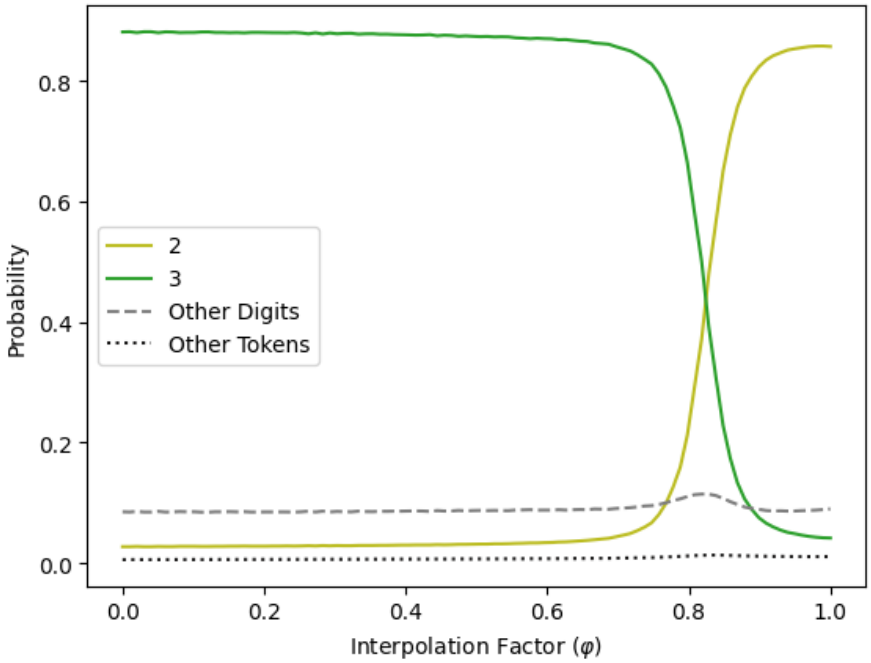


Figure 3.3: Impact of reducing the duration of selected tokens on the predicted output probability. Digits that never reach a probability of at least 0.05 are merged into the category "Other Digits". All non-digit tokens are merged into "Other Tokens".

with $\varphi \in [0, 1]$; we also integrate starting from φ, instead of 0, to avoid assigning an oversized duration to the first token. In the second one, we scale the time steps by a factor equal to the interpolation factor:

$$\boldsymbol{z}_{int}^{(i)} = \boldsymbol{z}_{orig}^{(i)} \tag{3.20}$$

$$t_{i,int} = \varphi t_{i,orig}, \tag{3.21}$$

with $\varphi \in [0.05, 1]$. While the impact of translation is minimal (Figure 3.6), scaling causes wild swings in predicted probabilities: with a small φ, all tokens are "smushed together", which leads to significant uncertainty over the correct output (Figure 3.7). As the duration of each token approaches that of a regular token seen during training (i.e. 1), the predictions grow closer to the correct one. This suggests that transformers inherently

learn the "duration" of a token, which when modified leads to changes in meaning.

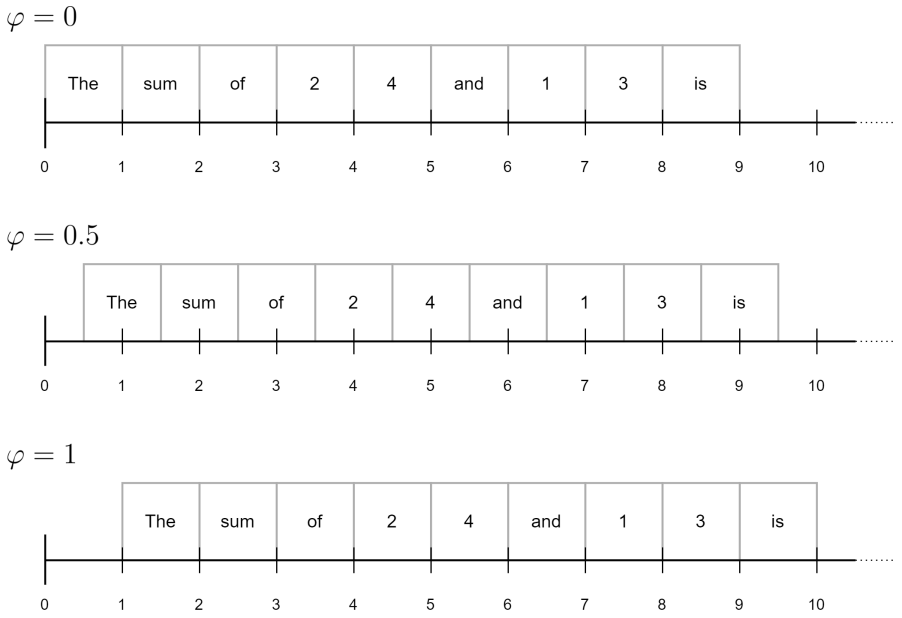


Figure 3.4: Visualization of the translation invariance experiment. Tokenization is simplified for illustrative purposes.

Transformers Are (Approximately) Density-Invariant For our final experiment, we simply augment the original list of samples by replacing each sample $\boldsymbol{z}^{(i,orig)}$ with φ new samples $\boldsymbol{z}_{new}^{(i,1)}, \boldsymbol{z}_{new}^{(i,2)}, \ldots, \boldsymbol{z}_{new}^{(i,\varphi)}$, with $\varphi \in \{1, \ldots, 10\}$. These samples are defined as follows:

$$\boldsymbol{z}_{new}^{(i,j)} = \boldsymbol{z}_{orig}^{(i)} \tag{3.22}$$

$$t_{i,j,new} = t_{i,orig} + \frac{j}{\varphi}. \tag{3.23}$$

In other words, we are sampling with higher density from the implicit dynamic denoted by assuming that each original token has a duration of 1. If our intuition about duration is correct, φ samples with duration $1/\varphi$ should be treated the same as 1 sample with duration 1, provided that the new samples all have the same value as the original one

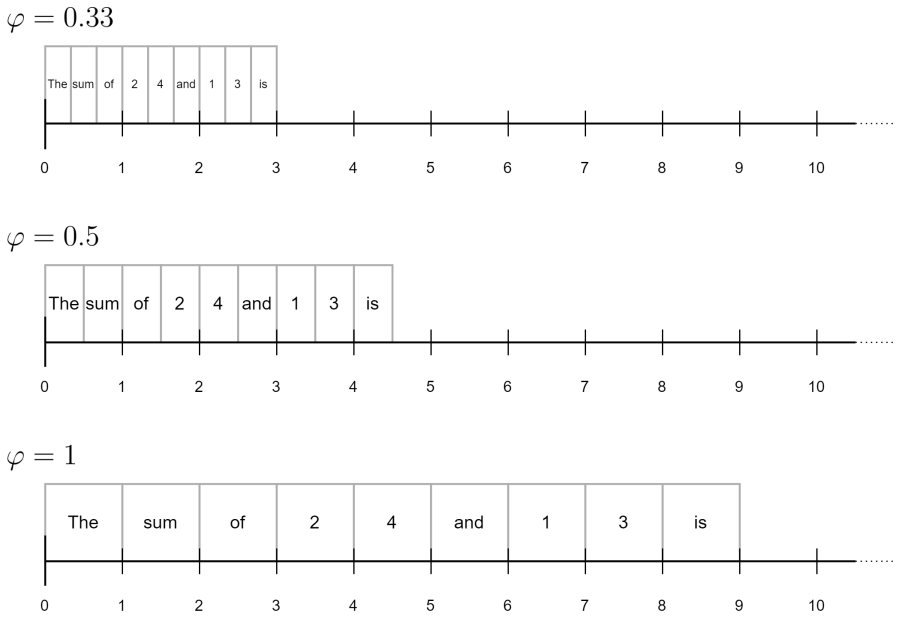


Figure 3.5: Visualization of the scale invariance experiment. Tokenization is simplified for illustrative purposes.

(Figure 3.8). Indeed, as shown in Figure 3.9 the effect of increasing the sampling density is present, but moderate: this suggests that duration is a reasonable framework to interpret the behavior of a CCT and, by extension, a vanilla transformer.

3.4 Implications and Potential Extensions

While we introduced CCTs as a tool to draw more accurate comparisons between causal transformers and diffusion models, our extension can both shed insight into the nature of transformers and suggest potential new architectures. We therefore provide a brief overview of future research directions and how they connect to CCTs.

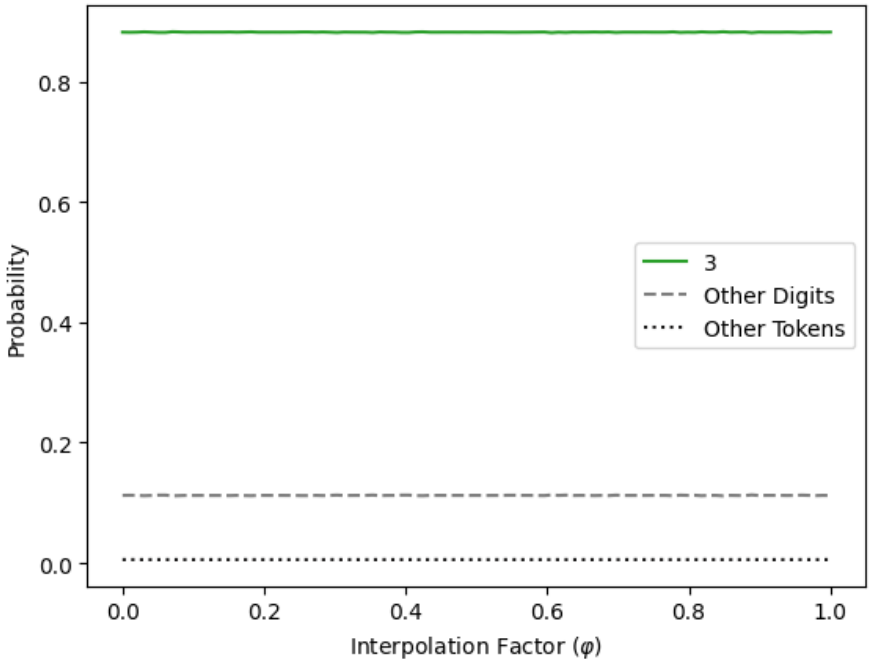


Figure 3.6: Effect of translation on the predicted output probability. Digits that never reach a probability of at least 0.05 are merged into the category "Other Digits". All non-digit tokens are merged into "Other Tokens".

3.4.1 The Implicit Dynamic

Our analysis does not, for a given vanilla transformer, determine the distribution of $\boldsymbol{s}^{(t)}$ for $t \notin \{1, \ldots, T\}$. However, the fact that feeding sequences with non-integer time steps gives results that are consistent with human intuition rules out the possibility that continuous extensions of vanilla transformers are meaningless. In other words, we know that transformers reason about time in a continuous manner (and we have a testable way to determine if a process matches empirical findings), but we do not know the exact nature of $\boldsymbol{s}^{(t)}$. This opens up an interesting research direction: *what are the properties of the continuous process learned implicitly by transformers?* We stress that this process is not necessarily equivalent to that of natural language (assuming that natural language can even be modeled as a continuous process): instead, it can be seen as the process induced by the structural priors of transformers. If such properties were determined, it

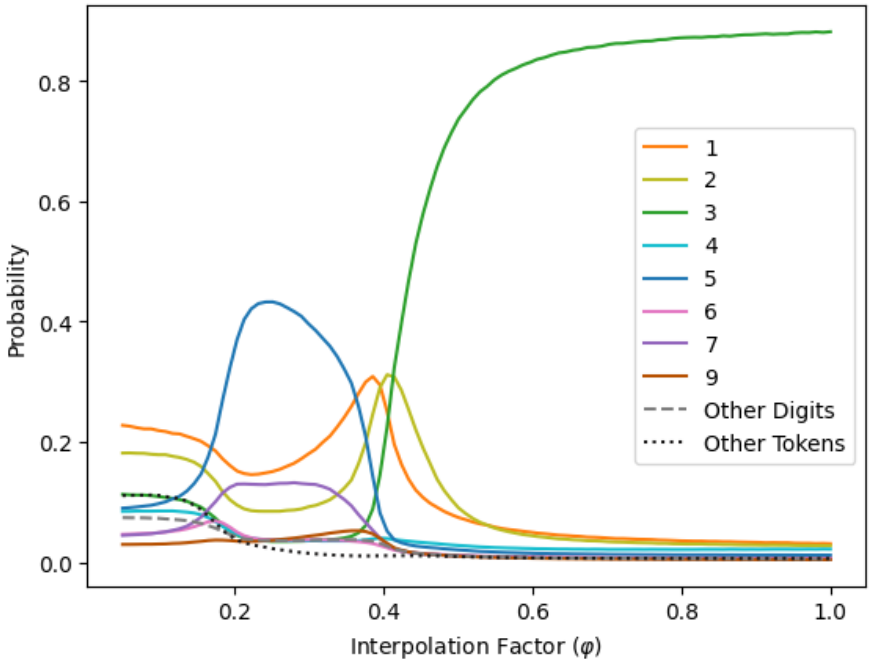


Figure 3.7: Effect of scaling on the predicted output probability. Digits that never reach a probability of at least 0.05 are merged into the category "Other Digits". All non-digit tokens are merged into "Other Tokens".

might be possible to study alternative training setups based on these findings, with the goal of learning a better approximation of the target dynamic.

3.4.2 Beyond Decoder-Only Causal Modeling

CCTs are, by definition, decoder-only causal models, but they can be easily adapted to other types of transformers. For instance, it might be possible to adapt CCTs to encoder-decoder or masked language modeling tasks by making minor changes to the architecture. This opens up new directions for the study of continuity in sequence-to-sequence tasks, with two main goals. First, a unified theoretical framework for continuous transformers might improve our understanding of the advantages and disadvantages of masked language modeling compared to causal language modeling. Second, extending CCTs beyond decoder-only modeling might both shed insight into their relation with

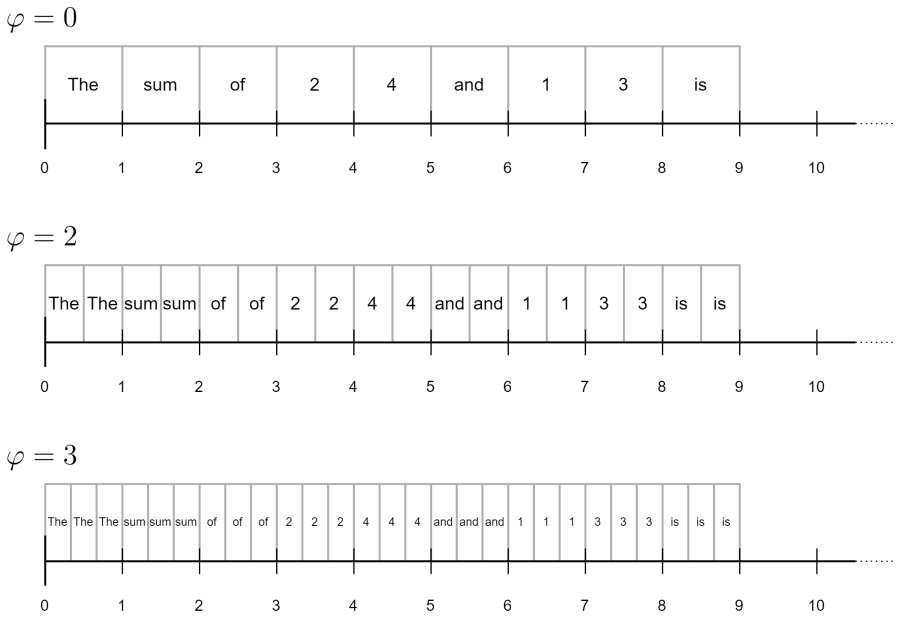


Figure 3.8: Visualization of the density invariance experiment. Tokenization is simplified for illustrative purposes.

encoder-decoder models and suggest new language model architectures.

3.4.3 Integrating Over the Embedding Space

CCTs rely on the assumption that a sentence is a sequence (either finite or continuous) of embeddings: in other words, for each time step t, there is exactly one corresponding embedding element $\boldsymbol{s}^{(t)} \in \mathbb{R}^{emb}$, even if that embedding does not necessarily correspond to a discrete token. However, it might be interesting to study how transformers respond to sentences with multiple embeddings at the same time: intuitively, the presence of multiple embeddings in the same time step is fundamentally different from the presence of a single embedding obtained by interpolating multiple embeddings. This study would require extending CCTs so that they integrate both across time and across the embedding space. We leave the details of this extension, as well as the potential interpretations of experimental results obtained from this model, to future work.

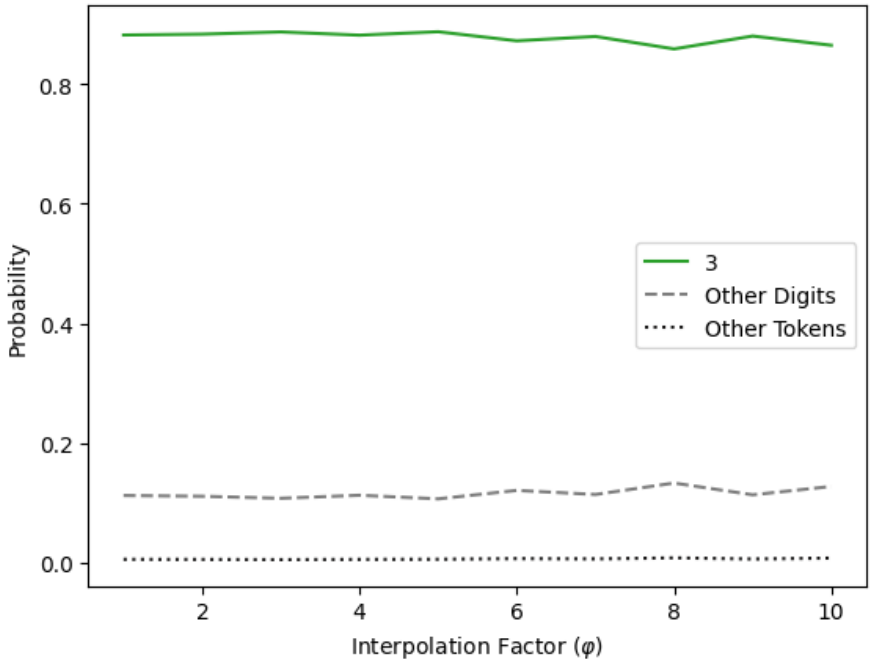


Figure 3.9: Impact of sampling density on the predicted output probability. Digits that never reach a probability of at least 0.05 are merged into the category “Other Digits”. All non-digit tokens are merged into “Other Tokens”.

Chapter 4

Structured Autoregressivity

In the previous section, we showed that vanilla causal transformers can be seen as discrete approximations of Continuous Causal Transformers, similarly to how diffusion models can be seen as discrete approximations of their corresponding score-based continuous process. However, the similarities between causal transformers and diffusion models, as trained in practice, extend beyond this fact alone. In particular, we identify five properties that diffusion models and causal transformers share (even in their discrete form):

- Task Subdivision;
- Weight Sharing;
- Teacher Forcing;
- Frequent Supervision;
- Total Supervision.

We name the collection of these properties *Structured Autoregressivity*. These properties might seem purely coincidental, but as we will show, they emerge naturally when

studying diffusion models and transformers under the lens of approximating continuous stochastic processes.

4.1 Task Subdivision and Weight Sharing

Autoregressive generative models are characterized by the fact that they a) split the generation process into a certain number of steps, which are b) all computed by the same model. Both of these properties are usually treated as facets of being autoregressive, but they are not necessarily co-occurring: for instance, a task can be solved using a pipeline, where the task is divided into sub-tasks, but each sub-task is solved by a different model. Additionally, the quantitative metrics related to these properties can be tuned independently: a model may have a high number of steps and a low degree of weight sharing across steps, and vice-versa. We therefore treat them as separate properties, naming them respectively Task Subdivision and Weight Sharing.

Intuition From the point of view of approximating a continuous process, Task Subdivision reduces the step size of a discrete approximation, which improves the first-order approximation error: in other words, when using small step sizes the model's prediction of the next intermediate step is less likely to diverge from the actual intermediate state, which is why a high degree of task subdivision tends to lead to performance improvements.

Weight sharing, on the other hand, introduces an architectural prior such that the nature of the process is assumed to be consistent throughout its entire length. While using different models for various parts of the process would lead to a better overall approximation quality, weight sharing improves data efficiency, since the data on the process at a given time step also provides information on the behavior in other time steps. Therefore, as long as there are no major differences in behavior throughout the

process, weight sharing brings significant benefits.

Overall, both of these phenomena provide an intuition for why continuous processes are often approximated using autoregressive models, and why they can be potentially useful for diffusion and language models.

Comparison with the Literature In the context of image generation, autoregressivity is a major distinguishing factor for diffusion models, since most of the other state-of-art models, such as Variational AutoEncoders (VAEs) [18], Generative Adversarial Networks [10] and Normalizing Flows [21], are non-autoregressive, with the notable exception of PixelCNN [28].

Instead, language models have long used autoregressivity to better model the target sequence, such as in Long-Short Term Networks (LSTMs) [15] and Gated Recurrent Units (GRUs) [8]. However, as we will show in the next few sections, there are also other properties that affect performance.

4.2 Teacher Forcing

Another highlight of both causal transformers and diffusion models is that they are trained following teacher forcing, as opposed to free running. The former trains a model by feeding a state $\boldsymbol{x}^{(0)}$ and asking it to predict the next state $\boldsymbol{x}^{(1)}$; regardless of how accurate the prediction is, the prediction is replaced with the ground truth $\boldsymbol{x}^{(1)}$, and the model is trained to predict $\boldsymbol{x}^{(2)}$ given $\boldsymbol{x}^{(1)}$. This process is repeated for the entirety of the sequence. With free running, on the other hand, the predicted output is used as input: the model is trained to predict $\boldsymbol{x}^{(1)}$ given $\boldsymbol{x}^{(0)}$, $\boldsymbol{x}^{(2)}$ given $f(\boldsymbol{x}^{(0)})$, $\boldsymbol{x}^{(3)}$ given $f(f(\boldsymbol{x}^{(0)}))$, and so on.

While free running has the advantage of training the model in an end-to-end fashion, with the model learning to account for its own approximation inaccuracies, it comes with

a significant downside: training with free running is significantly less stable. This is due to the fact that the model rarely receives as input the correct intermediate states: with each step, the approximation errors accumulate, which means that while the model is trained to approximate the ground truth, it needs to do so taking as input an intermediate state that is very far from the ground truth, especially after several free running steps.

On the other hand, while teacher forcing can lead to reductions in performance (since at inference time the model will receive as inputs its own imperfect predictions instead of the ground truth states, leading to a training-testing domain shift), at least in the context of language modeling there are results showing that the benefits tend to outweigh the drawbacks [13].

4.3 Frequent Supervision

Another major aspect of diffusion models and casual transformers is that there is *sufficiently frequent* information on what the intermediate states should be: in other words, when training a diffusion model or a causal transformer, not only there are enough intermediate states such that predicting the next state given the current one is feasible, but we also have ground truths for each intermediate state. Since the dynamics of both transformers and diffusion models are inherently *synthetic* (i.e. they have been constructed such that it is by design possible to know what each intermediate state should be), we always have Frequent Supervision. Intuitively, frequent supervision is equivalent to teaching a student both the solution to a problem and the steps required to solve the problem, which leads to better performance compared to providing the solution alone.

Compare this with Neural Turing Machines (NTMs) [11] and Differentiable Neural Computers (DNCs) [12], which have several intermediate states, but no supervision on what these intermediate states should look like. It is thus no surprise that both NTMs and DNCs struggle with learning complex tasks [11, 12].

From the point of view of approximating dynamical systems, without sufficiently frequent supervision there are only two options for the model designer: either the model is trained to predict larger steps, skipping entirely the unknown intermediate states (which is only feasible if the larger step size does not come with an excessive increase in complexity), or the model needs to be trained using free running without intermediate supervision, which leads to the problems described in Section 4.2, but with the additional drawback of not having guidance on the unsupervised states.

Overall, we hypothesize that the synthetic nature of the intermediate states of diffusion models and causal transformers, which enables frequent supervision even with long sequences, is a major reason for the better performance of these models. This observation suggests some potential research direction for model design, which we outline in Sections 5.2.2 and 5.2.3.

4.4 Complete Supervision

Finally, a potentially undervalued aspect of diffusion models and causal transformers is the fact that, when training these models, the state of a sequence at any given step is completely known, without any uncertainty. In the context of approximating a dynamical system, partial observability has long been known to be a complicating factor: consider for instance the difficulties of learning a Partially Observable Markov Decision Process (POMDP) [3] compared to its fully observable counterpart. It is thus reasonable to assume that having complete information on the target output improves the performance of both diffusion models and causal transformers.

This observation might seem trivial, but several classes of models do not share this property, the most noteworthy of which are models with memory. For instance, the state of an LSTM is described by its input vector, the previous output vector (also known as the hidden vector), and the cell vector (which can be seen as the memory storage of the

LSTM). While the first two are known, in typical training setups there is no information on what a “good” cell vector should look like. Therefore, the model needs to figure out how to properly store information on its own, slowing down training and introducing complexity. Diffusion models and transformers, on the other hand, have complete information on the intermediate states, which might lead to faster and more stable training.

Chapter 5

Next Steps

We conclude this work by outlining some future research directions inspired by our findings. Specifically, we provide an overview of our ongoing ablation study (Section 5.1) and describe some specific examples of how our results can inform model design (Section 5.2).

5.1 Ablation Study

While both diffusion models and transformers share the five Structured Autoregressivity properties, how relevant they are for model performance is unclear. We therefore discuss in this section some potential ablations of these properties in order to determine their actual impact. Since we are still in the early stages of data collection, we do not have conclusive evidence, which is why we limit ourselves to reporting the experimental design, leaving the actual results to future work. However, we have found that even studying how to properly ablate a given property provides significant intuition into their nature: as a matter of fact, several theoretical insights emerged from trying to approach ablations in a rigorous manner.

Additionally, note that no matter the results of the experiments, we have a *win-win*

situation: either a certain property has a significant impact on model performance, which means that future architectures should take into account how to maximize it, or it does not, which means that future architectures can trade these properties for potentially more useful ones.

5.1.1 Task Subdivision

In order to ablate Task Subdivision, we need to subdivide the task into fewer steps. For vanilla language models, the number of steps is the number of tokens in the sentence. This means that we have three main ablation options:

- Reducing the length of the sentence (e.g. through summarization): while this approach requires the fewest changes to the architecture, summarization can significantly alter the meaning of the sentence, which might have an unexpected impact on the performance (e.g. a transformer learning only to model short sentences with few details). Additionally, there are limits to how much a sentence can be summarized before it loses its meaning;

- Using longer tokens (e.g. switching from character tokens to subword tokens or from subword tokens to word tokens): this approach can have unexpected side effects too, since the nature of the tokens affects how transformers learn the relationship between their embeddings. For instance, the relationship between words (e.g. how "house" and "housing" are more semantically similar than "house" and "cloud") is more informative than that between characters (e.g. "c" and "d" do not necessarily have more in common than "c" and "m"), which might have an impact on performance;

- Predicting $k > 1$ tokens at once: this approach allows us to maintain both the content of the original sentence and the relationship between the tokens, while still reducing the number of steps in the sequence by a factor of k.

We thus follow the third approach, specifically using an implementation inspired by Semi-Autoregressive Transformers [30]. These architectures have k output heads, each corresponding to a different subsequent token: given n input tokens, the first head predicts the $(n+1)$-th token, the second head predicts the $(n+2)$-th, and so on.

For diffusion models, on the other hand, reducing the sequence length simply involves using fewer diffusion steps, which leads to a straightforward experimental setup.

5.1.2 Weight Sharing

Ablating Weight Sharing involves reducing the level of parameter sharing of the model throughout different steps of the sequence. A simple way to do so is to use k different models for different steps of the sequence: doing so splits the training task into multiple ones, where each model needs to accurately approximate its part of the sequence. Two potential splitting strategies are interval-based splitting (i.e. the first model is trained on $[0, T/k]$, the second on $(T/k, 2T/k]$, the third on $(2T/k, 3T/k]$, and so on) and module-based splitting (i.e. the first model is trained on the steps $\{km \mid m \in \{0, \dots, T/k\}\}$, the second on $\{km+1 \mid m \in \{0, \dots, T/k\}\}$, the third on $\{km+2 \mid m \in \{0, \dots, T/k\}\}$, and so on). However, a notable advantage of the former approach is that it is independent of the discretization precision (i.e. the 1 over the number of steps) of the continuous process, while the latter requires considering such a value carefully. Nevertheless, we plan to study and compare the effect of both techniques.

5.1.3 Teacher Forcing

In order to ablate Teacher Forcing, we apply free running for k subsequent steps, before applying a teacher forcing step. Specifically. let f be the model to be trained and let $\boldsymbol{x}^{(t)}$ be the sequence to be learned. We compute the predicted output $\tilde{\boldsymbol{x}}^{(t)}$ (which will be

compared with the ground truth) as follows:

$$\tilde{\boldsymbol{x}}^{(t+1)} = \begin{cases} f(\boldsymbol{x}^{(t)}) & t \equiv 0 \mod k \\ f(\tilde{\boldsymbol{x}}^{(t)}) & \text{otherwise.} \end{cases} \tag{5.1}$$

In other words, throughout the sequence we use free running, but every k steps we replace the predicted state with the ground truth. Note it is possible to trivially achieve complete free running by setting k to a value greater than or equal to the sequence length.

While this approach can be implemented without major obstacles for diffusion models, vanilla causal transformers have an additional complication: the output of a transformer is a vector of logits where each element corresponds to an embedding, but transformers accept as input only one embedding per time step. We therefore outline some possible strategies:

- Discretizing the output, either by taking the highest-probability embedding or by sampling from the output distribution. This approach, while straightforward, involves modifying the output, since all the information on the other potential embeddings is discarded;
- Branching: by sampling more embeddings and expanding the sequence based on the various possible choices (thus creating a sort of tree), it is possible to avoid discarding part of the information, but this benefit comes at the cost of a higher computational requirement;
- Training an extension of a transformer that accepts multiple embeddings as input (e.g. similar to those described in Section 3.4.3): aside from the implementation obstacles that one such transformer would need to overcome, it is unclear how relevant the results of training such extensions would be for the purposes of understanding vanilla transformers.

Since all three approaches present some significant downsides, we leave further investigation over which course of action is preferable to future work.

5.1.4 Frequent Supervision

Ablating Frequent Supervision involves removing target information from a dataset, for example by taking a dataset sample and removing the target information of a given sequence element with uniform probability ρ. When dealing with missing targets, there are three possible strategies:

- Avoiding computing the loss for outputs that do not have a corresponding target, essentially ignoring part of the sequence for each sample from the dataset;
- Using free running to compute the unknown intermediate states. Note that this experiment is different from the one described in Section 5.1.3 since the latter computes the loss for each intermediate state (which inherently requires knowing the target for each step);
- Using larger step sizes to skip over the missing targets entirely: since this experiment is equivalent to the one described in Section 5.1.1, we do not re-implement it.

Therefore, the two main experimental approaches for ablating Frequent Supervision are ignoring unknown parts of the sequence and using free running, although we are yet to determine if one solution is clearly superior to the other.

Note that the removed information needs to be consistent throughout the training in order for the ablation to be successful. In other words, removing a different subset of the target information every epoch is ineffective, since the model would still be seeing the data point ρ% of the time and would eventually learn from it (although it would happen over the course of more epochs).

5.1.5 Complete Supervision

Finally, the most straightforward way to ablate Complete Supervision is to only provide partial information on the target output.

For diffusion models, we propose to mask a certain proportion of the image's pixels (with each image in the dataset having a different mask).

Transformers, on the other hand, require a slightly different approach, since the target of a vanilla causal transformer is a single, discrete output. Therefore, we proceed as follows. First, we pick for a given state a subset of the output vector that does not contain the logit for the target label. We name this subset the *masked* subset. We then take the complementary of this subset and use it to compute the softmax, completely ignoring the non-masked logits. Finally, we compute the loss (usually the cross-entropy loss) between the softmax output and the target label. Overall, this approach sidesteps the problem of the single-target output, but it still achieves the intended result: from the point of view of the loss, any value for a logit contained in the masked subset is equivalent, which means that the model lacks complete information on the "full" output distribution.

5.2 Implications for Model Design

Finally, we provide an overview of how our unified analysis of causal transformers and diffusion models can inform the development of new architectures.

5.2.1 Sampling Strategies

Both diffusion models and causal transformers, being stochastic models, require some form of sampling at inference time. For diffusion models, these are commonly DDPM and DDIM sampling (with various diffusion schedules), while for causal transformers the most common ones include greedy sampling, beam search, and top-p sampling [16].

These sampling strategies have been developed independently, but if we treat diffusion models and causal transformers as fundamentally similar, the insight into one model might suggest improvements for the other.

For instance, it might be possible to apply beam search to diffusion models by sampling multiple predictions in a DDPM fashion and picking the most promising beams. Conversely, the diffusion schedules of diffusion models could be applied to causal language modeling: instead of generating a sentence with linear time steps, we could use another schedule, such as cosine scheduling. Doing so might potentially improve the generation quality of causal transformers by strategically making the model "talk slower" in certain parts of the generation process. Another possibility would be to use a dynamic step size based on how "complex" a certain part of the process is.

5.2.2 Synthetic Autoregressivity

An interesting aspect of diffusion models is that the denoising-based generative process is, all things considered, pretty arbitrary: progressively removing Gaussian noise until we obtain an image is a very different way to approach image generation compared to how a human would create a piece of art. And yet, since by design determining the target intermediate states is simple, it is possible to train a model in a manner that follows the principles of Structured Autoregressivity.

On the other hand, for language models, the subdivision of the task in word tokens (or subword tokens) is more consistent with human intuition, but it suffers from a significant limitation: the sentence generation process can only be split into as many intermediate states as the number of tokens in the sentence. This puts a constraint on how simple predicting the next state can be: a model must be powerful enough to predict the next token in a single pass.

If, however, we were to introduce new intermediate states (which we name *synthetic states*) between existing ones, we might be able to remove this constraint as long as three

conditions are satisfied:

- The synthetic states are reasonable intermediate steps towards predicting a real state;
- The synthetic states do not hamper the model's ability to be trained following Structured Autoregressivity;
- The synthetic states can be computed easily.

For instance, we could add synthetic tokens between each "real" one, thus increasing the length of the sequence and reducing the effort required to predict each one. These synthetic tokens do not necessarily need to have a semantic meaning: even something as simple as an interpolation of the surrounding real tokens could be sufficient.

This extension could not only improve the performance of large language models, but also enable the creation of smaller language models: since each individual step would be smaller, computing the next step would require a smaller model, although of course the smaller size would come at the cost of having to perform more steps to generate the sequence. In other words, we would be trading speed for a reduced memory footprint.

5.2.3 Beyond Generation

While the purpose of this work is to study the properties of diffusion models and causal transformers, our insights into Structured Autoregressivity can be applied to a large class of models, including non-generative ones. For this reason, applying our findings to other ML tasks represents a potentially beneficial research direction.

For the sake of example, consider an image classifier $f : X \to Y$, where $X \subseteq \mathbb{R}^{in}$ is the input space and $Y = \Delta^{N-1}$ is the space of distributions over N possible classes. Usually, a classifier directly predicts the class of the image in a non-autoregressive fashion. However, we could extend this process by introducing synthetic intermediate states

(as described in Section 5.2.2) and thus splitting the classification task into multiple sub-steps, which are then modeled by an autoregressive architecture. Compared to just building a deeper model, this approach has the advantage of Weight Sharing, leading to better parameter efficiency.

An example of intermediate states $\boldsymbol{h}^{(t)}$ might be to use ever more accurate estimates of the correct label, i.e.

$$\boldsymbol{h}^{(t)} = \left(1 - \frac{t}{T}\right) \boldsymbol{u} + \frac{t}{T} \mathbf{1}_c, \tag{5.2}$$

where $\boldsymbol{u}$ is a uniform vector, $\mathbf{1}_c$ is the one-hot encoding of the target class c and T is a hyperparameter representing the number of steps. The model would then take as input $\boldsymbol{h}^{(t)}$ and $\boldsymbol{x}$ and compute an approximation of the next intermediate state conditioned on $\boldsymbol{x}$:

$$f(\boldsymbol{h}^{(t)}, \boldsymbol{x}) \approx \boldsymbol{h}^{(t+1)}. \tag{5.3}$$

This approach, while certainly unorthodox, could potentially provide performance gains, since having an initial estimate of the correct class might influence how low-level feature extractors operate (similarly to top-down attention in biological brains [5]).

Chapter 6

Conclusion

In this thesis, we studied causal transformers and diffusion models under the lens of continuous stochastic processes. By doing so, we showed not only that diffusion models and causal transformers can be treated as discretizations of continuous stochastic processes, but also that they feature several common properties which, when viewing the models as approximations of such processes, emerge as natural design decisions. As part of this analysis, we introduced Continuous Causal Transformers, a generalization of causal transformers which, even on its own, has the potential to shed insight into the structural priors of transformers, as evidenced by our qualitative results. We also discussed how both Continuous Causal Transformers and our common framework can suggest novel directions for model design and training.

We believe that the strength of this work lies, rather than in specific results, in how it provides an alternative view of state-of-the-art generative models. Treating diffusion models and causal transformers as different variants of the same mechanism can both provide a more holistic insight into generative modeling and inform future developments in the field. While we do not claim to have provided conclusive empirical evidence of the practical relevance of this intuition, the directions we outlined in this work certainly warrant further exploration.

Overall, there are still many open questions in the field of generative modeling, both in the context of understanding why some architectures perform better than others and in determining the fundamental limits of neural network-based approximations. We hope that our findings will contribute to this ongoing exploration and help develop a more complete view of generative modeling.

Bibliography

[1] M. Abdin, S. A. Jacobs, A. A. Awan, J. Aneja, A. Awadallah, H. Awadalla, N. Bach, A. Bahree, A. Bakhtiari, H. Behl, et al. Phi-3 technical report: a highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

[2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[3] K. J. Åström. Optimal control of markov processes with incomplete state information i. *Journal of mathematical analysis and applications*, 10:174–205, 1965.

[4] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[5] F. Baluch and L. Itti. Mechanisms of top-down attention. *Trends in neurosciences*, 34(4):210–224, 2011.

[6] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[7] Y. Chen, K. Ren, Y. Wang, Y. Fang, W. Sun, and D. Li. Contiformer: continuous-time transformer for irregular time series modeling. *Advances in Neural Information Processing Systems*, 36, 2024.

[8] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[9] A. H. d. O. Fonseca, E. Zappala, J. O. Caro, and D. Van Dijk. Continuous spatiotemporal transformers. *arXiv preprint arXiv:2301.13338*, 2023.

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[11] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[12] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[13] T. He, J. Zhang, Z. Zhou, and J. Glass. Exposure bias versus self-recovery: are distortions really incremental for autoregressive text generation? *arXiv preprint arXiv:1905.10617*, 2019.

[14] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[15] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[16] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

[17] V. Khrulkov, G. Ryzhakov, A. Chertkov, and I. Oseledets. Understanding ddpm latent codes through optimal transport. *arXiv preprint arXiv:2202.07477*, 2022.

[18] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[19] F. Moreno-Pino, Á. Arroyo, H. Waldon, X. Dong, and Á. Cartea. Rough transformers: lightweight continuous-time sequence modelling with path signatures. *arXiv preprint arXiv:2405.20799*, 2024.

[20] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, and A. Edelman. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

[21] D. Rezende and S. Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

[22] O. Ronneberger, P. Fischer, and T. Brox. U-net: convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

[23] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[24] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[25] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu. Roformer: enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[26] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[27] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[28] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al. Conditional image generation with pixelcnn decoders. *Advances in neural information processing systems*, 29, 2016.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[30] C. Wang, J. Zhang, and H. Chen. Semi-autoregressive neural machine translation. *arXiv preprint arXiv:1808.08583*, 2018.

[31] B. Zhang and R. Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

Acknowledgements

We thank the Alan Turing Institute for providing the computational power required for our ongoing empirical studies, and Angelo Huang for his invaluable help in their implementation. The author would also like to thank the Collegio Superiore di Bologna for funding his visiting period at Oxford University. Finally, the author would like to thank Andrea Iacco, Elena Bresolin, Francesco Prem Solidoro, and all the residents of the second floor of the Collegio Superiore di Bologna for graciously listening to his constant ramblings on a long series of topics, the most recent of which is the content of this thesis.

Appendix A

Additional Qualitative Results

We report the qualitative results for eight pairs of numbers (including the pair studied in Section 3.3.2) in Figures A.1 to A.5. The pairs of numbers were chosen randomly from the set of pairs that satisfy the following conditions:

- The numbers are two-digit numbers;
- Their sum is less than 100;
- The sum of the second digit of the first number and the second digit of the second number must be less than 10 (i.e. there is no carry step when adding the numbers);
- All digits are distinct;
- 0 cannot appear in any position;
- In the standard setting (i.e. without any modifications), the predicted probability of the correct output is at least 80%.

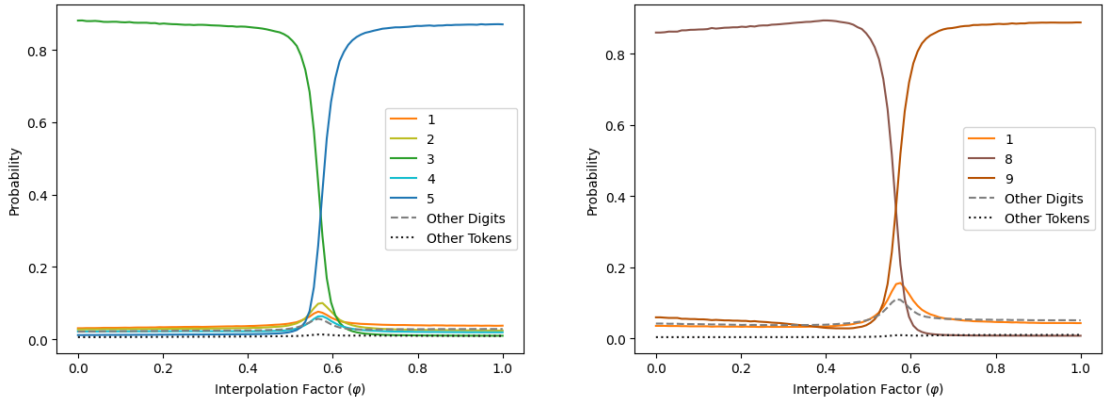
The chosen pairs are:

- 24 + 13;

- 32 + 56;
- 13 + 74;
- 52 + 31;
- 14 + 23;
- 16 + 42;
- 82 + 14;
- 38 + 51.

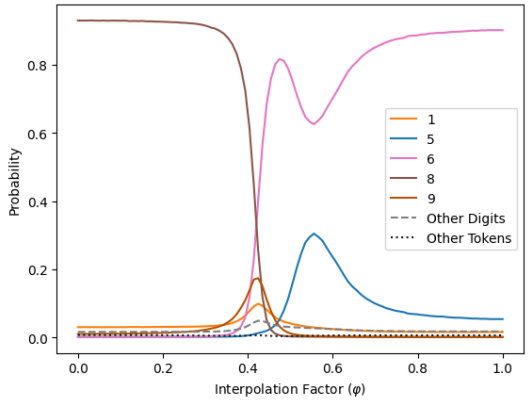
Some highlights include:

- For 38 + 51, reducing the duration of *51* does not lead to an increase in the probability of *5*, but rather an increase in the probability of *4* (which is compatible with the model treating the sum as 38 + 5);
- For 16 + 42, interpolating the embeddings with $\varphi = 1$ (i.e. computing 16 + 24) leads to an uncertain output, even if in such a situation all embeddings have a corresponding token. In other words, the vanilla transformer did not learn properly how to compute 16 + 24. This phenomenon may explain the unusual results in other experiments, particularly duration interpolation and density interpolation.

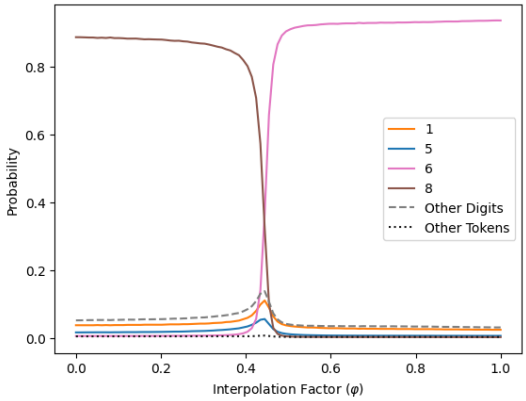


(a) 24 + 13

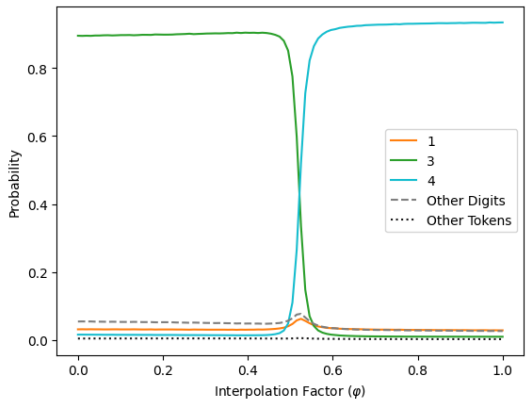
(b) 32 + 56



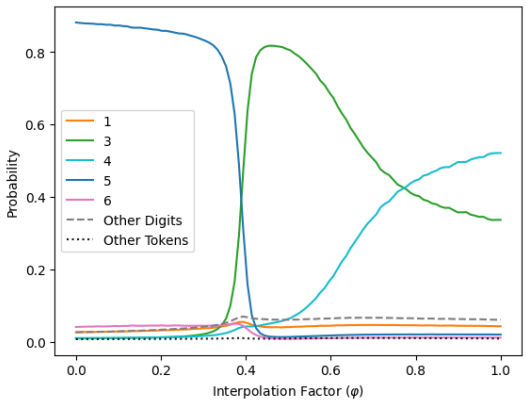
(c) 13 + 74



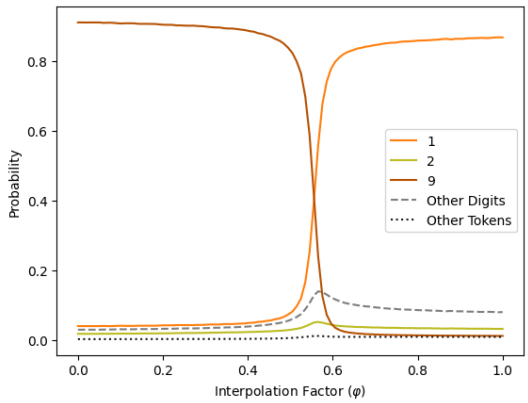
(d) 52 + 31



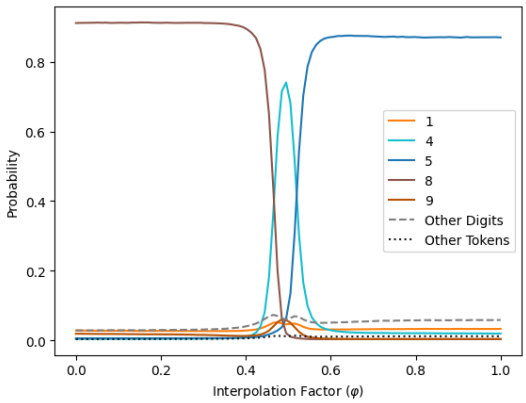
(e) 14 + 23



(f) 16 + 42

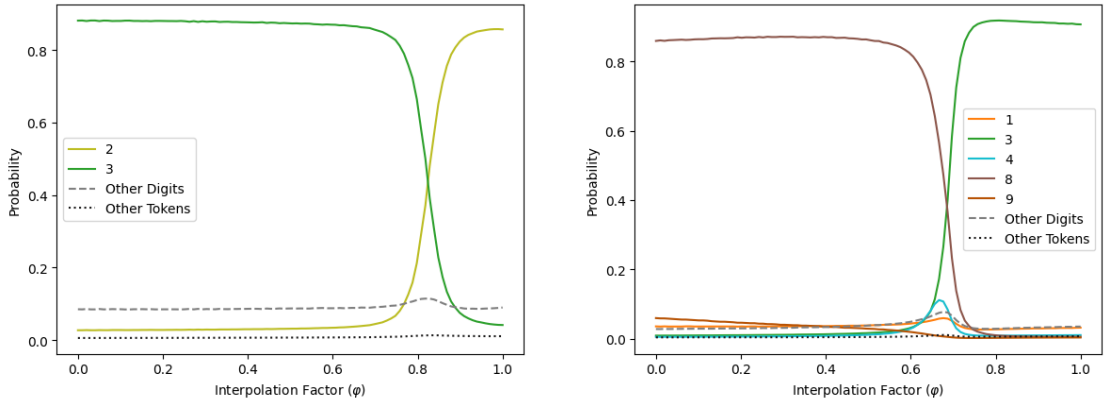


(g) 82 + 14



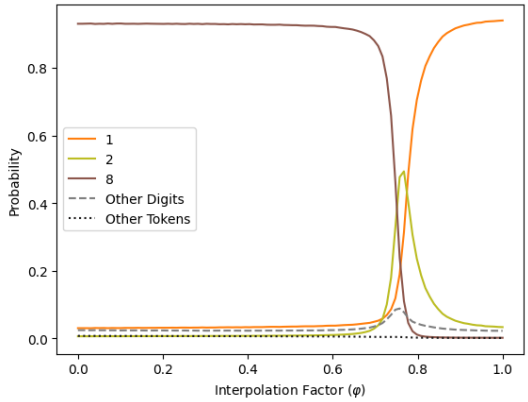
(h) 38 + 51

Figure A.1: Impact of embedding interpolation on the predicted output probability for eight pairs of numbers. Digits that never reach a probability of at least 0.05 are merged into the category “Other Digits”. All non-digit tokens are merged into “Other Tokens”.

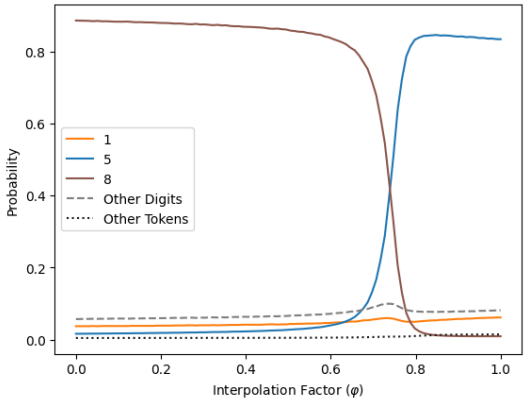


(a) 24 + 13

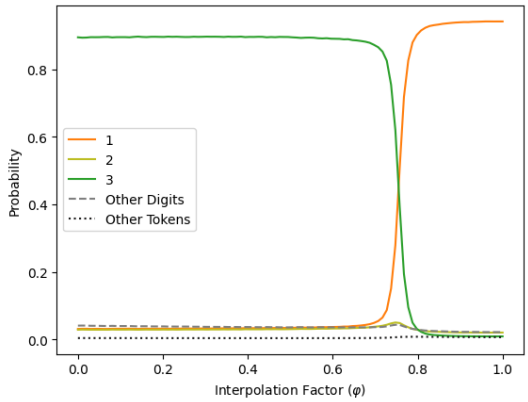
(b) 32 + 56



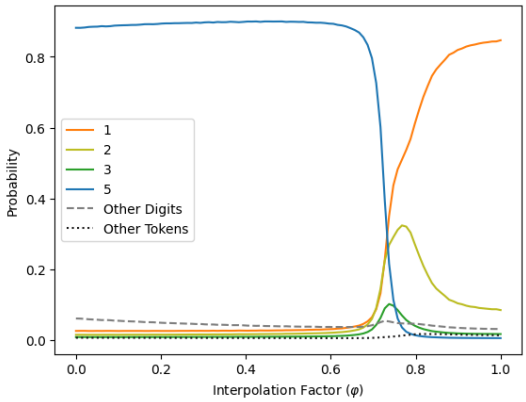
(c) 13 + 74



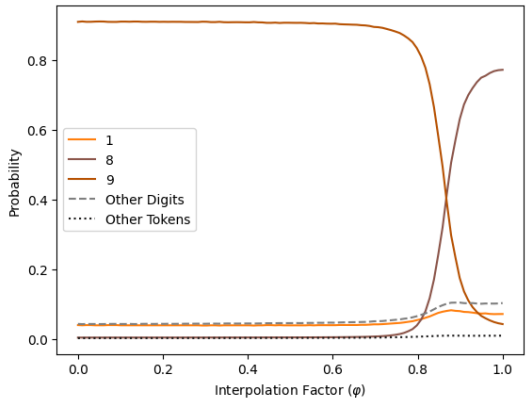
(d) 52 + 31



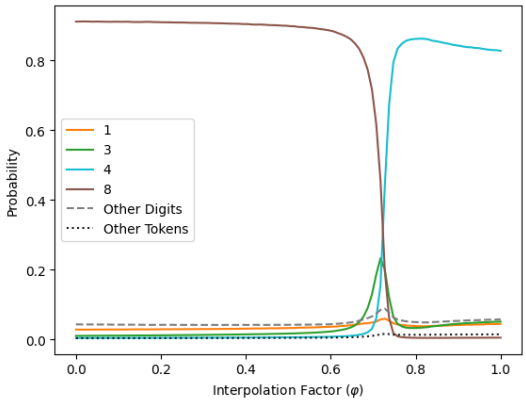
(e) 14 + 23



(f) 16 + 42

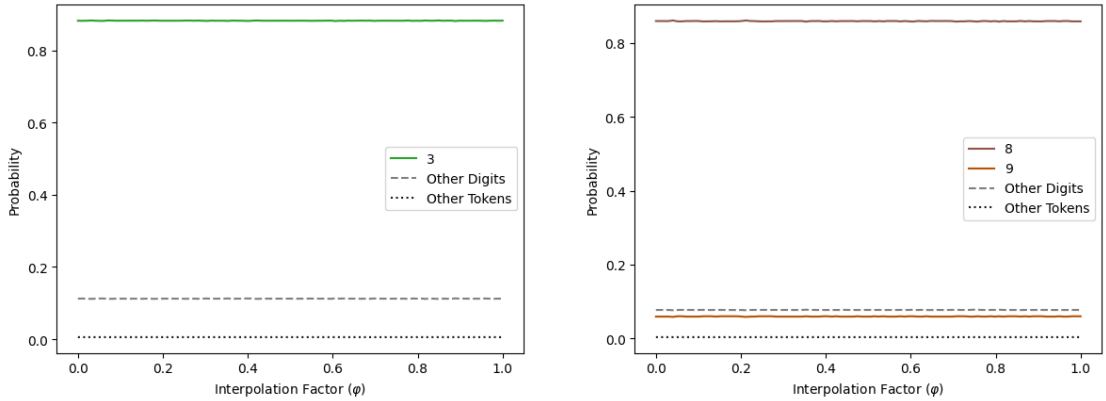


(g) 82 + 14



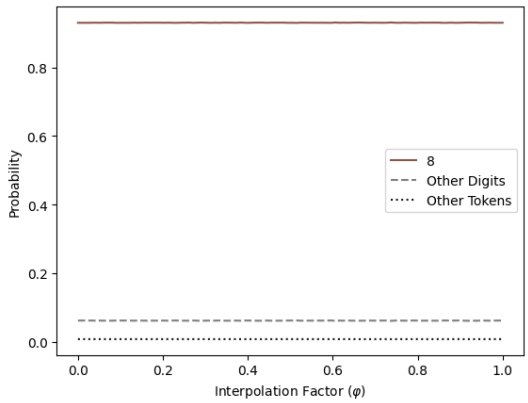
(h) 38 + 51

Figure A.2: Impact of reducing the duration of selected tokens on the predicted output probability for eight pairs of numbers. Digits that never reach a probability of at least 0.05 are merged into the category "Other Digits". All non-digit tokens are merged into "Other Tokens".

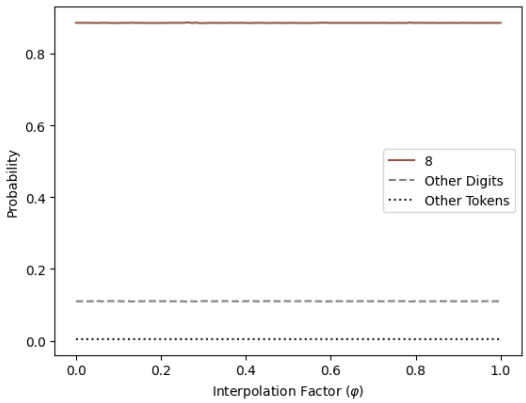


(a) 24 + 13

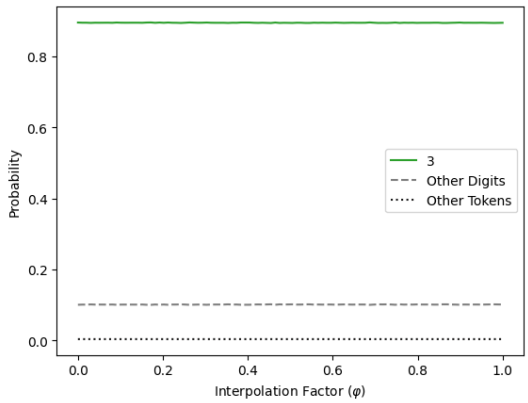
(b) 32 + 56



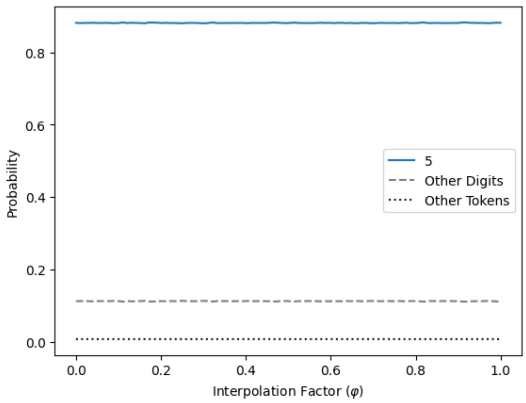
(c) 13 + 74



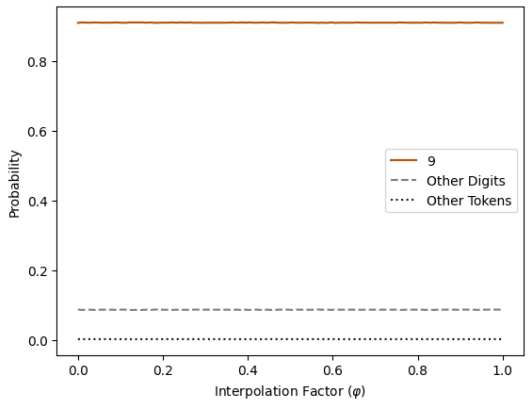
(d) 52 + 31



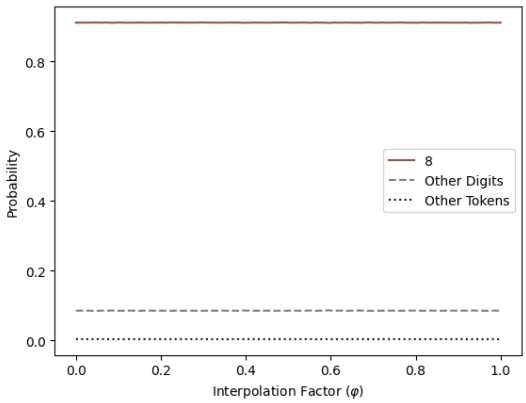
(e) 14 + 23



(f) 16 + 42

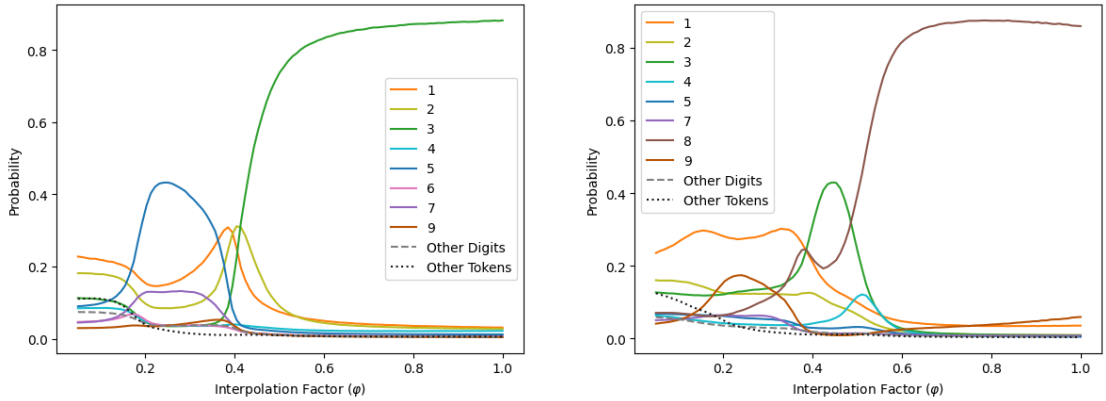


(g) 82 + 14



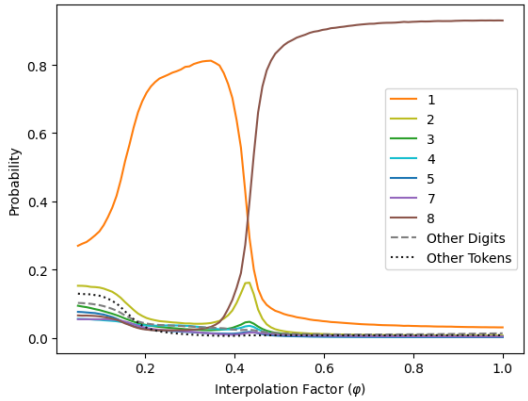
(h) 38 + 51

Figure A.3: Impact of translation on the predicted output probability for eight pairs of numbers. Digits that never reach a probability of at least 0.05 are merged into the category “Other Digits”. All non-digit tokens are merged into “Other Tokens”.

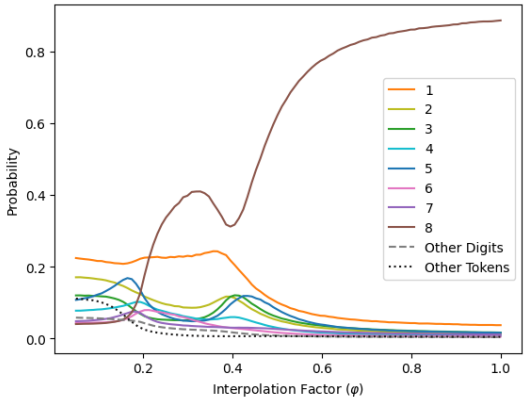


(a) 24 + 13

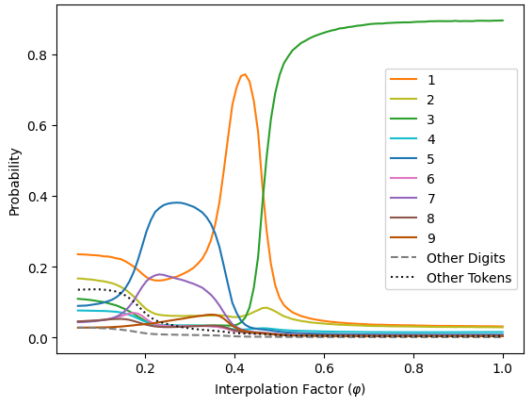
(b) 32 + 56



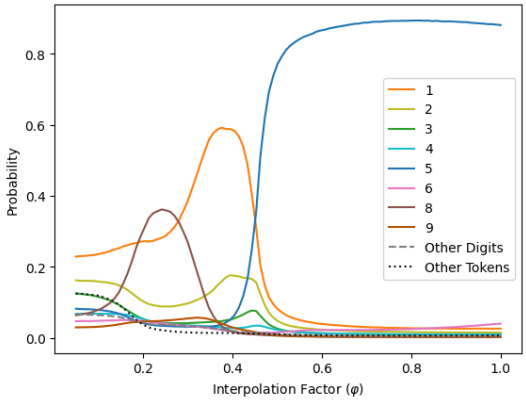
(c) 13 + 74



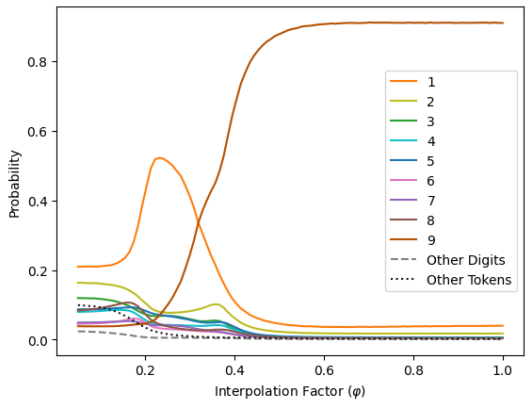
(d) 52 + 31



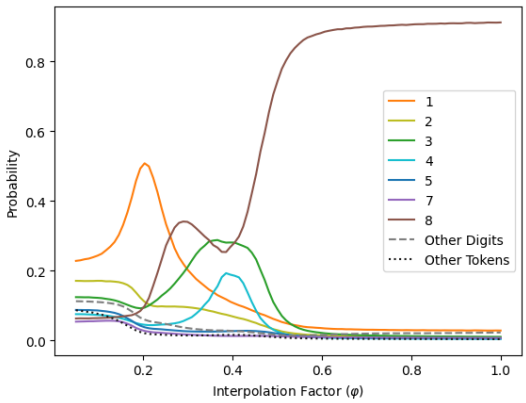
(e) 14 + 23



(f) 16 + 42

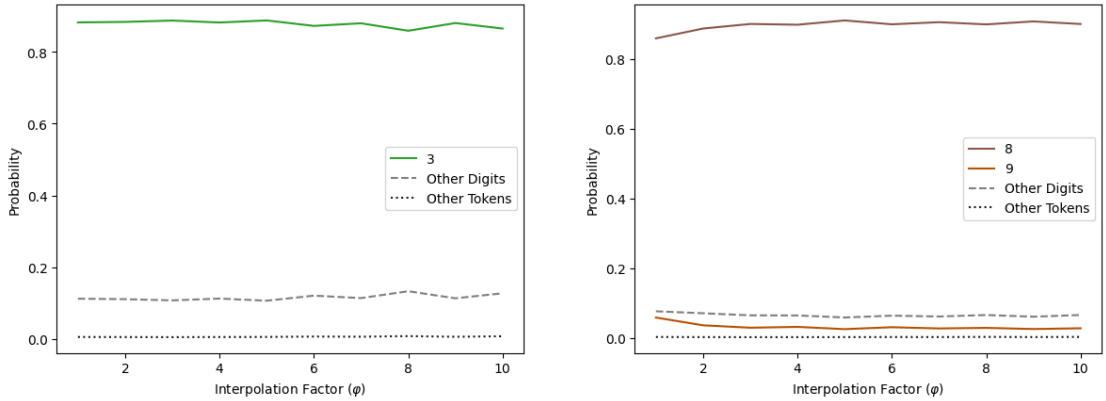


(g) 82 + 14



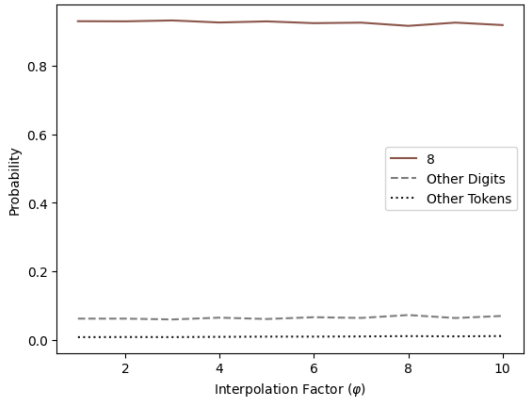
(h) 38 + 51

Figure A.4: Impact of scaling on the predicted output probability for eight pairs of numbers. Digits that never reach a probability of at least 0.05 are merged into the category "Other Digits". All non-digit tokens are merged into "Other Tokens".

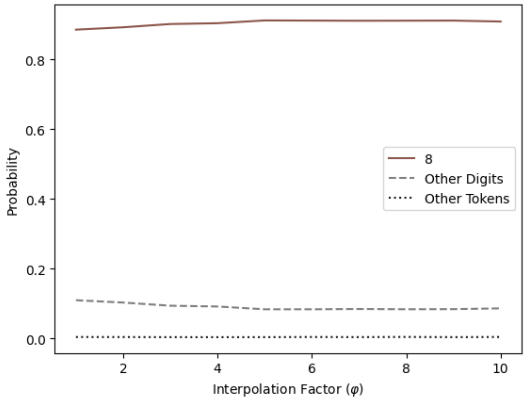


(a) 24 + 13

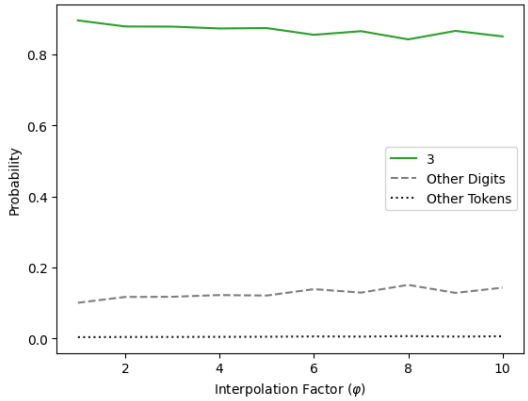
(b) 32 + 56



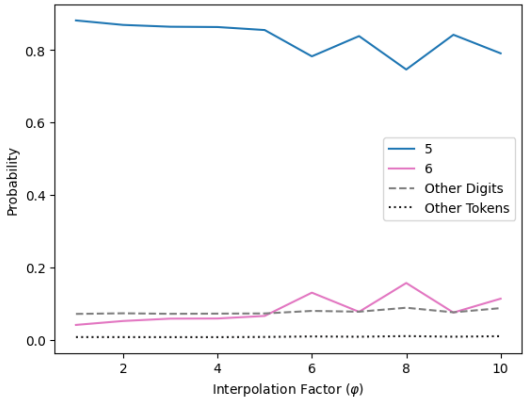
(c) 13 + 74



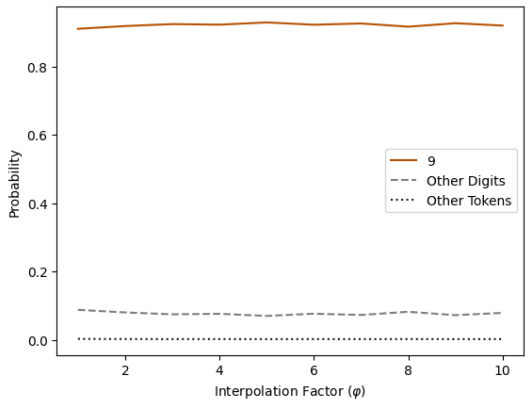
(d) 52 + 31



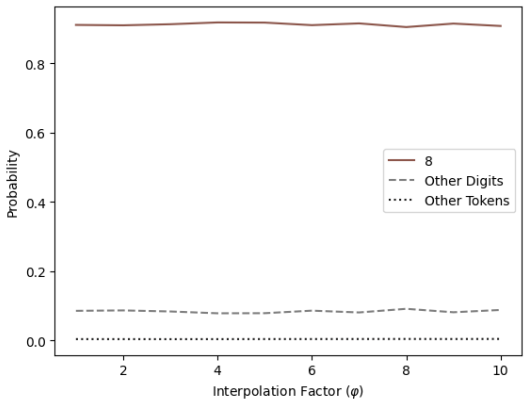
(e) 14 + 23



(f) 16 + 42



(g) 82 + 14



(h) 38 + 51

Figure A.5: Impact of sampling density on the predicted output probability for eight pairs of numbers. Digits that never reach a probability of at least 0.05 are merged into the category “Other Digits”. All non-digit tokens are merged into “Other Tokens”.