Doctoral theses at NTNU, 2025:6

Ruslan Khalitov

# Beyond Transformer: Scalable Neural Attention Networks for Long Sequential Data

Doctoral thesis

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of
Philosophiae Doctor
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Ruslan Khalitov

# Beyond Transformer: Scalable Neural Attention Networks for Long Sequential Data

Thesis for the Degree of Philosophiae Doctor

Trondheim, January 2025

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science



NTNU

Norwegian University of
Science and Technology

**NTNU**
Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Department of Computer Science



ISBN 978-82-326-8600-1 (printed ver.)
ISBN 978-82-326-8599-8 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

Doctoral theses at NTNU, 2025:6

Printed by NTNU Grafisk senter

# Abstract

Sequential data is ubiquitous across a wide range of application domains. To extract meaningful representations from long sequential data effectively and efficiently, neural networks need to demonstrate several key properties including a full receptive field and the ability to scale to very long sequences. Traditional neural networks, however, frequently struggle to handle long sequential data due to their inherent architectural limitations.

This research introduces novel scalable neural network architectures specifically designed for representation learning from long sequential data. At the core of our approach is a sparse factorization technique that approximates attention matrices using a product of sparse, full-rank matrices with circular and dilated patterns. This design enables efficient processing while maintaining a full receptive field. The complexity is reduced from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$ for sequences of length $N$. Our flagship architecture, ChordMixer, combines position-wise multi-scale rotation layers with element-wise MLPs, allowing it to process sequences with millions of tokens. Moreover, our method can handle variable-length sequences without padding or truncation.

We evaluate our approaches on diverse benchmarks, including the Long Range Arena where ChordMixer establishes new state-of-the-art performance, notably achieving 98.63% accuracy on the most challenging PathfinderX task. To assess performance in real-world scenarios with variable-length sequences, we introduce novel benchmark tasks spanning DNA sequence taxonomy classification (sequences up to 447K base pairs), long document classification (up to 131K characters), and arithmetic operations (up to 1.5M elements). On these benchmarks, ChordMixer significantly outperforms existing approaches, maintaining robust performance even at extreme sequence lengths where other methods either fail to process or show substantial degradation in performance.

# Preface

This thesis is submitted to the Norwegian University of Science and Technology (Norges teknisk-naturvitenskapelige universitet, NTNU) in partial fulfillment of the requirements for the degree of Philosophiae Doctor (Ph.D.).

This doctoral research has been conducted at the Department of Computer Science (IDI), the Faculty of Information Technology and Electrical Engineering (IE), NTNU, Trondheim, under the main supervision of Professor Zhirong Yang and co-supervision of Professor Jingyue Li.

The research presented here was financially supported by the Research Council of Norway with the grant number 287284.

The thesis takes the form of a paper collection, where the included papers have already been published at conferences or in journals. For consistency and readability, the publications have been reformatted and typeset anew for inclusion in this thesis. Therefore, they deviate visually and typographically from their published versions.

# Acknowledgments

I extend my heartfelt gratitude to all those who supported me throughout my PhD journey.

Foremost, I owe a deep debt of gratitude to my supervisor, Professor Zhirong Yang. His invaluable assistance, guidance, and inspirational life-changing ideas have profoundly shaped my research experience. I am truly grateful for his support.

I also wish to express my appreciation to my co-supervisor, Professor Jingyue Li, for his encouragement and insightful suggestions that have been crucial in my pursuit of this research degree.

A special thank you goes to my colleagues, Tong Yu and Lei Cheng. Collaborating on numerous projects, conducting experiments, co-authoring articles, and participating in academic conferences together has been a cornerstone of my academic growth. Your partnership has been indispensable.

I am thankful to the Department of Computer Science (IDI) for access to the high-performance computing cluster IDUN and a powerful server equipped with GPUs, which I have had the privilege of using extensively. My colleagues deserve recognition for creating a friendly and supportive working environment with excellent resources.

Heartfelt thanks to my friends from around the globe for their unwavering support and belief in my abilities.

My profound gratitude is reserved for my family, whose unconditional support and endless love have been my pillars of strength. Your encouragement has been the backbone of my journey. I could not have completed this without you.

# Contents

# List of Figures

# List of Tables

# Acronyms

**1D** one-dimensional
**2D** two-dimensional
**ANN** Artificial Neural Network
**AUROC** Area Under the Receiver Operating Characteristic curve
**BERT** Bidirectional Encoder Representations from Transformers
**bp** base pairs
**CDIL-CNN** Circular DILated Convolutional Neural Network
**CNN** Convolutional Neural Network
**DCNN** Dynamic Convolutional Neural Network
**DNA** Deoxyribonucleic acid
**DNN** Deep Neural Network
**GPT** Generative Pre-trained Transformer
**GRCh38** Genome Reference Consortium Human Build 38
**GRU** Gated Recurrent Units
**kbp** kilobase pairs
**LLM** Large Language Model
**LRA** Long Range Arena
**LSTM** Long Short-Term Memory
**MLM** Masked Language Modeling
**MLP** Multilayer Perceptron
**NLP** Natural Language Processing
**OCR** Open Chromatin Region

**RNN** Recurrent Neural Network

**RQ** Research Question

**SSL** Self-Supervised Learning

**TCN** Temporal Convolutional Network

# Part I
# Research Overview

# Chapter 1

# Introduction

This chapter provides an overview of the research conducted during my doctoral studentship. It begins with a brief motivation and description of the research topic, followed by the formulation of the research goal and the research questions that guide the study. Finally, it summarizes the conducted research.

## 1.1 Motivation

The analysis of sequential data is pivotal across various scientific disciplines and practical applications, including speech recognition [9], natural language processing (NLP) [10], signal processing [11], and genomic research [12]. Traditionally, extracting valuable information from such data has relied heavily on domain-specific knowledge and statistical methods [13, 14, 15, 16, 17].

In predictive tasks, a specific predictor is employed to transform pre-processed features into targets, such as classification or regression. Before the advent of Deep Learning, standard models like decision trees [18], various forms of regression, SVMs [19], and boosting machines [20] were commonly used to achieve the target goal. These classical ML models struggled with handling raw sequential data due to their length and variability, necessitating

dimensionality reduction and feature representation processes. Consequently, their performance was limited and heavily dependent on manual, labor-intensive feature extraction.

The advent of Artificial Neural Networks (ANNs) marked a transformative shift in feature extraction from raw sequential data. With the introduction of the back-propagation algorithm and subsequent advancements in optimization techniques, ANNs emerged as powerful, universal feature extractors applicable across diverse fields. These networks refine the weights (parameters) of connections between neurons by aligning the network's outputs with expected results through iterative training. This process enhances performance and automates the extraction of meaningful features directly from raw data, significantly boosting the model's adaptability for various data science tasks, including those involving complex sequences.

The evolution of sequence modeling in deep learning has led to significant advancements, particularly through Deep Neural Networks (DNNs) [21]. Notable among these are Recurrent Neural Networks (RNNs) such as Long Short-Term Memory (LSTM) [22] networks and Gated Recurrent Units (GRU) [23], which process sequences by maintaining a memory of prior inputs. However, RNNs as a network backbone struggle with long sequences due to the vanishing and exploding gradient problem, which impedes their ability to capture long-range dependencies. Convolutional Neural Networks (CNNs) have been applied to sequence processing to capture local temporal patterns, yet they often miss broader contextual information [24].

The introduction of neural attention and, notably the Transformer [2], revolutionized the field by using specific transformational mechanisms that weigh the relevance of different parts of the input data, enabling parallel processing and effectively addressing long-range dependencies. However, the dot-product self-attention mechanism in Transformer leads to quadratic increases in computational and memory costs with longer sequences. To address this scalability challenge, the community has proposed many Transformer variants. While these variants attempt to overcome the challenges, they often sacrifice certain properties, making none of them as universally applicable as the original Transformer [3].

Besides the scalability challenges, the real applications often involve sequences that vary widely in length, which poses an additional difficulty for incorporating the sequence

modeling. Despite advances in processing variable-length short sequences with RNN-based and several attention-based networks, existing methods struggle with extremely long and highly variable sequences, where dependencies may span thousands of sequence tokens, as seen in DNA/RNA sequences [25, 26]. These ongoing challenges underscore the need for continued innovation in neural architectures to efficiently process very long and variable-length sequences without sacrificing performance.

## 1.2 Research Goal and Research Questions

The goal of my PhD study is to develop a novel neural attention method that enables the modeling of long sequential data. At the end of my PhD study, we delivered a versatile learning framework based on artificial neural networks, as well as its associated algorithms, theoretical analysis, computer software, and datasets.

The research goal focuses on addressing two key research questions (RQs):

**Research Question 1:** How can we design effective and efficient attention neural networks to perform representation learning for long sequential data?

Research Question 1 (RQ1) aims to develop new neural network architectures that efficiently and effectively learn representations from long sequential data. In addressing this question, we evaluated existing models and introduced innovative methodologies and techniques. We explored four different neural network architectures, each demonstrating superior performance on a variety of new and established benchmarks involving long sequences. Our strongest model, ChordMixer, exhibited exceptional performance on benchmarks with extremely long sequences, proving effective in modeling sequences from various domains, including NLP, mathematical operations, image processing, and genomics.

A central design feature common to all proposed networks is the sparse, dilated, and circular mixing of input signals, which achieves scalable attention networks and offers enhanced performance on the longest sequences.

Another significant challenge in this field is processing sequences with highly variable lengths, which requires a neural attention backbone to manage both local and long-range interactions within the sequences. The ChordMixer neural network is particularly adept at addressing the second research question (RQ2), which involves handling sequential data with varying lengths.

**Research Question 2:** How can we implement efficient representation learning for sequences with variable lengths?

Apart from scalability, an essential property of a good neural network for sequential data is the ability to work with sequences of highly variable lengths within a single task. We developed a new neural network backbone designed to effectively handle sequences of highly variable lengths. Standard approaches, like chunking long sequences or padding shorter ones, can lead to the loss of long-range interactions or increased computational demands due to padding.

Recognizing the absence of benchmarks for testing neural network efficacy on such data, we introduced several challenging tasks across different domains—DNA, natural language, and arithmetic tasks—where sequences may differ in length by several orders of magnitude. This benchmark demonstrated the ChordMixer architecture's ability to manage modeling both long and highly variable sequences, in comparison to other sequential neural network models.

These research questions capture two critical elements of the formulated research goal: designing neural architectures for long sequences and modeling sequences that are long and dramatically vary in length. The ultimate aim is to learn representations from long sequential data efficiently and effectively, expanding this capability to handle variable-length sequences proficiently. The subsequent section summarizes how these questions are explored through five publications produced during this doctoral study.

## 1.3 Research Conducted

Our research includes five publications, denoted as A-E, which are organized in chronological order based on the publication date. The symbol † indicates the first co-/authorship. Full published versions of the research papers are included in Part II of this thesis.

**A.** Ruslan Khalitov†, Tong Yu†, Lei Cheng, and Zhirong Yang. Sparse factorization of square matrices with application to neural attention modeling. *Neural Networks*, 152:160–168, 2022.

**B.** Tong Yu†, Ruslan Khalitov†, Lei Cheng, and Zhirong Yang. Paramixer: Parameterizing mixing links in sparse factors works better than dot-product self-attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 691–700, 2022.

**C.** Lei Cheng†, Ruslan Khalitov, Tong Yu, Jing Zhang, and Zhirong Yang. Classification of long sequential data using circular dilated convolutional neural networks. *Neurocomputing*, 518:50–59, 2023.

**D.** Ruslan Khalitov†, Tong Yu, Lei Cheng, and Zhirong Yang. Chordmixer: A scalable neural attention model for sequences with different length. *In The Eleventh International Conference on Learning Representations*, 2023.

**E.** Lei Cheng†, Tong Yu†, Ruslan Khalitov, and Zhirong Yang. Self-supervised learning for dna sequences with circular dilated convolutional networks. *Neural Networks*, 171:466–473, 2024.

In this part, we provide a summary of the content within each publication, its relation to the formulated RQs, and the individual contributions of each author.

Publication A [27] addressed RQ1 by introducing a new approximation technique known as Sparse Factorization. This method breaks down an $N \times N$ square matrix into the product of $\log_2 N$ full-rank sparse matrices. By employing the Chord protocol [28], we specified the locations of the non-zero entries, ensuring that each factorizing matrix included $N \log_2 N$

non-zero entries. The full receptive field is guaranteed after mixing all sparse matrices through matrix multiplication. The method showed its effectiveness in both parametric (PSF-Attn) and non-parametric settings. The computational complexity of this method is $O(N(\log_2 N)^2)$, which is efficient for feature extraction from neural networks when dealing with large sequence lengths $N$. In this publication, Ruslan Khalitov and Tong Yu jointly designed and implemented the model, interpreted the model using visualizations, and collaboratively wrote and revised the manuscript; Ruslan Khalitov hosted and maintained the GitHub codebase and conducted experiments in sections 5.2 and 5.4; Tong Yu conducted experiments in sections 5.3 and 5.4; Lei Cheng assisted in model implementation, experiment design, and paper writing; Zhirong Yang formulated the research problem, designed the model framework, and conducted the experiments in section 5.1.

Publication B [29] addressed RQ1 by extending the parametric framework designed in Publication A (PSF-Attn). In addition to the Chord protocol, the Paramixer design incorporates an additional sparse protocol — the circular dilated (CDIL) protocol investigated in publication C [30]. The CDIL protocol exhibits a computational complexity of $O(KN \log_2 N)$, where $K$ represents the kernel size. Both protocols facilitate an efficient approximation, enhancing the model's capability to handle long sequential data. Further improvements were made by incorporating multiple mixing layers into the architecture, which enabled the extraction of higher-level representations and led to enhanced performance. In this publication, Ruslan Khalitov and Tong Yu jointly designed and implemented the model and collaboratively wrote and revised the manuscript; Tong Yu hosted and maintained the GitHub codebase and conducted experiments in sections 4.2, 4.3, and 4.4; Ruslan Khalitov implemented the mixing module, conducted experiments in section 4.1, assisted experiments in section 4.2, interpreted the model using visualizations; Lei Cheng assisted in model implementation, experiment design, and paper writing; Zhirong Yang formulated the research problem and drafted the model design.

Publication C [30] addressed Research Question 1 by introducing the CDIL-CNNs, a novel convolutional architecture that includes symmetric dilated convolutions, circular mixing, and residual connections. This configuration ensures that each element in the final output representation is equally exposed to all information from the entire input sequence

after only $\lceil \log_2 \frac{N}{2} \rceil$ layers of CDIL. The model effectively minimizes boundary effects and demonstrates scalability for handling long sequential data. In this publication, Lei Cheng designed and implemented the model, conducted all experiments and ablation studies, interpreted the model using visualizations, created and maintained the GitHub codebase, wrote and revised the manuscript; Ruslan Khalitov assisted in model fitting and experiments in section 4.2; Tong Yu assisted in experiments in section 4.2; Jing Zhang assisted in experiment analysis and paper writing; Zhirong Yang formulated the research problem and drafted the model design.

Publication D [31] is the most important publication within my thesis that addresses both RQ1 and RQ2 by proposing a simple neural attention network called ChordMixer. Each ChordMixer block consists of a position-wise multi-scale rotation layer, which operates without learnable parameters, and an element-wise MLP layer. The rotation follows the Chord protocol, necessitating $\log_2 N$ ChordMixer blocks to provably achieve a full receptive field, leading to a sub-quadratic complexity of the ChordMixer backbone. ChordMixer efficiently processes sequences with varying lengths by dynamically adjusting the number of attention blocks. To showcase this capability, the study developed several unique benchmarks with sequences having high length variability. Consequently, ChordMixer not only excels at learning representations from extremely long sequences (up to 1.5 million tokens) but also effectively manages varying sequence lengths across multiple domains. In this publication, Ruslan Khalitov designed and implemented the model, conducted experiments in sections 4.1, 4.3, 4.4, and LRA experiments, interpreted the model using visualizations, created and maintained the GitHub codebase, wrote and revised the manuscript; Tong Yu conducted experiments in section 4.2 and genetic variant experiments; Lei Cheng assisted in model implementation, experiment design, and paper writing; Zhirong Yang drafted the model design.

Publication E [32] addressed RQ1 by demonstrating the ability of Circular Dilated convolutions (CDIL-CNN) to extract powerful features in a self-supervised learning framework. During the pre-training phase, a masking strategy was employed, which relies on unmasked tokens to predict masked ones [33]. This methodology substantially enhances the effectiveness of representation learning on DNA sequences, proving usefulness in scenarios

with limited labeled data. In this publication, Lei Cheng and Tong Yu jointly designed and implemented the model, created and maintained the GitHub codebase, and collaboratively wrote and revised the manuscript; Lei Cheng conducted experiments in sections 4.2 and 4.3, interpreted the model using visualizations; Tong Yu conducted experiments in section 4.1; Ruslan Khalitov assisted in model implementation and experiment design; Zhirong Yang formulated the research problem.

# Chapter 2

# Background and Related Work

This chapter provides an overview of the theoretical background relevant to this research project. First, we define the objective and desired properties of representation learning on long sequential data. Second, we review classical neural network architectures for sequence modeling and discuss their properties. Third, we dig deeper into the attention mechanism and discuss its existing variants. The methods described in this section provide an extensive overview of the research on long-sequence modeling conducted over the last five years.

## 2.1 Representation Learning on Long Sequences

The success of machine learning models largely depends on the quality of data representation they use. Effective data representation, or feature extraction, transforms raw data into a form that facilitates the model's ability to identify and utilize patterns. Traditionally, this transformation has required significant manual effort in feature engineering, limiting the scalability and flexibility of model deployment and highlighting a major limitation of many learning algorithms — their reliance on human-crafted features [34].

Feature engineering is a process that relies heavily on domain expertise and human insight to identify the most important aspects of the data [35, 36]. However, this dependence on human input reveals a significant gap in the autonomy of learning algorithms.

Ideally, learning algorithms would automatically discover and utilize underlying patterns in the data, reducing the need for manual feature engineering. This would not only speed up the deployment of machine learning systems in new applications but also improve their adaptability and ease of use [34].

Deriving meaningful features from one-dimensional structured data, such as sequences, poses significant challenges. Consider an input sequence $X$ consisting of $N$ elements: $(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N)$, where each $\vec{x}_t \in \mathbb{R}^D$ (for $1 \leq t \leq N$) denotes a $D$-dimensional vector positioned at $t$. The goal of feature extraction is to transform this sequence from its raw form into a feature-enriched space that better encapsulates the inherent structure and relationships within the data. This transformation employs a mapping function $f$, defined as $f : \mathbb{R}^{N \times D} \rightarrow \mathbb{R}^{N' \times D'}$, where $N'$ and $D'$ represent the dimensions of the resultant feature space. The new feature set, $E$, is derived by applying $f$ to $X$, thus $E = f(X)$.

The challenge of learning a transformation function $f$ increases with the length $N$ of the sequence, especially for sequences of highly variable lengths. Effective representations must reveal complex, non-linear dependencies that can span large portions of the sequence, often covering thousands of elements. This requires techniques that can handle both short-range and long-range dependencies within the sequence, which is crucial in fields like natural language processing, bioinformatics, and time series analysis [21, 37].

Adding to the complexity, representation learning models need to be effective across different domains. A model that works well for text analysis might struggle with genomic sequences or time series data due to differences in data characteristics and underlying patterns [38, 39]. This challenge requires the development of flexible and adaptable representation learning methods that can adjust to the unique properties of various data types without needing extensive reconfiguration for each new application.

Additionally, sequences often contain temporal or spatial relationships that are crucial for understanding the overall data structure. For example, the order of amino acids in a protein or the progression of stock prices over time carries important information that is contextually rich. Effective machine learning models must not only capture these relationships but also adapt to their dynamic nature, potentially changing their processing strategies based on the context provided by new data inputs [22, 40].

As we explore traditional methods for representation learning on sequences, we will examine their effectiveness and limitations. Starting with classical statistical techniques and moving through modern advancements in machine learning such as Recurrent Neural Networks (RNNs) and Attention Mechanisms, each approach represents a step toward automating the feature extraction process. These methods aim to reduce the dependence on manual feature engineering and improve the ability to capture the rich, complex dependencies that characterize sequential data [1, 2]. Our discussion will conclude with a detailed look at how these technologies have evolved to address the unique challenges posed by long sequences, setting the stage for the exploration of innovative attention models in the following sections.

## 2.2 Classical Methods of Representation Learning for Sequences

Before the widespread adoption of deep learning techniques, various classical methods were employed to extract features from sequence data in multiple domains, each tailored to specific types of data and analysis requirements. These methods heavily relied on domain expertise and were not easily transferable across different fields, highlighting a significant limitation in their applicability.

In speech recognition, techniques like Mel-Frequency Cepstral Coefficients (MFCCs) and Perceptual Linear Prediction (PLP) were foundational, converting raw audio signals into a representation that could capture the essential phonetic characteristics [41]. For a very long time, the ASR systems were dominated by a combination of Hidden Markov Models (HMMs) and Gaussian Mixture Models (GMMs), where GMMs modeled the association between acoustic features and phonemes, while HMMs modeled the sequence of phonemes [37, 42]. Similarly, in natural language processing (NLP), a sophisticated array of feature extraction algorithms has evolved from simple methods such as Bag-of-Words [43] and Term Frequency-Inverse Document Frequency (TF-IDF) [44] to more advanced models like Word2Vec [45] and GloVe [46] that capture intricate word relationships and

semantic structures.

In realm of signal processing, Wavelet and Fourier Transforms are commonly used to identify frequency variations over time, helping to analyze temporal signals effectively [16, 47]. In the field of genomics, the analysis of DNA, RNA, or protein sequences often involves methods that draw parallels with NLP techniques. For instance, approaches like $k$-mers (DNA subsequences of length $k$) and amino acid composition (AAC) are employed to analyze the frequency and arrangement of nucleotides or amino acids [17, 48, 49]. More advanced techniques, such as Position-Specific Scoring Matrix (PSSM) [50] and Pseudo Amino Acid Composition (PseAAC) [51], extend these analyses to capture more detailed features of the sequence data, which are crucial for tasks like protein function prediction [52]. These methods transform raw genomic sequences into a form that can be more effectively used for further analytical tasks.

Despite their effectiveness within their respective domains, these classical methods of feature extraction came with notable drawbacks. They often lacked the ability to automatically capture and model the complex dependencies within long sequence data, such as long-range interactions and temporal dependencies. For example, traditional methods like Hidden Markov Models (HMMs) or Fourier Transforms provide a limited context window, which can miss long-term dependencies [53]. Moreover, the necessity for manual feature engineering made these methods labor-intensive and less adaptable to new, unexplored types of data. Each domain-specific method required substantial domain knowledge, limiting the ability to leverage advances across different fields.

Additionally, once features were extracted using these classical methods, the data typically entered a second phase of processing where it was fed into traditional machine learning models like decision trees, boosting algorithms, and various forms of regression. These models were chosen for their proficiency in handling structured datasets and were fundamental in numerous applications due to their interpretability and the ability to model nonlinear relationships [54, 55]. However, the performance of these models heavily depended on the quality and appropriateness of the initial feature representation, underscoring the pivotal role of feature engineering in the traditional machine learning pipeline.

In summary, while classical methods for sequence representation were effective for

specific applications, their reliance on extensive domain knowledge, the laborious nature of feature engineering, and their limited transferability across domains highlighted the need for more flexible, automated methods of feature extraction, setting the stage for the adoption of neural networks in representation learning tasks.

## 2.3 Neural Networks for Sequential Data

Neural networks revolutionized the field of machine learning by enabling automatic feature extraction, significantly reducing the dependency on manual feature engineering that is prevalent in classical methods. Unlike traditional approaches, neural networks learn to identify patterns directly from raw data, providing a more generalized and flexible approach to handling various types of sequential data. This capability is especially transformative for domains like speech recognition, natural language processing, and time-series forecasting, where capturing intricate temporal dependencies is crucial.

### 2.3.1 From MLP to RNN for Sequences

We begin by examining the use of a simple Feed-Forward Network, or Multi-Layer Perceptron (MLP), for modeling sequences. MLPs, consisting of layers where information flows unidirectionally, face significant challenges when applied to sequential data. The primary limitations stem from their fixed input size and inability to capture temporal dependencies among sequential inputs [37]. In an MLP, each input is processed independently, without regard for the time-dependent nature of the sequence. This inherent limitation becomes particularly problematic in domains like time series analysis or natural language processing, where the order and context of inputs crucially impact their interpretation and the resultant outputs [56, 57].

To address these challenges, Recurrent Neural Networks (RNNs) were introduced, which are specifically designed to handle sequence prediction problems by maintaining a hidden state vector that acts as a memory of the information seen so far. The core idea behind RNNs is the use of shared weights in a looped network structure, allowing them to

extend their memory over arbitrary lengths of sequences and to generalize across different positions in time, which is not feasible with traditional feedforward networks.

$$h_t = \sigma(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \tag{2.1}$$

$$y_t = W_{yh}h_t + b_y \tag{2.2}$$

Here $h_t$ is the hidden state at time $t$, $x_t$ is the input at time $t$, $W_{hx}$, $W_{hh}$, and $W_{yh}$ are weights, $b_h$ and $b_y$ are biases, and $\sigma$ is a non-linear activation function, typically a tanh or sigmoid.

Despite their theoretical advantage of handling sequences of variable lengths, RNNs often suffer from practical difficulties such as the vanishing gradient problem, where gradients of the loss function decay exponentially with time, making it challenging to learn long-range dependencies. This issue is particularly problematic in tasks requiring understanding of context over long sequence spans, such as in document classification or conversational models.

### 2.3.2 LSTM and GRU for Longer Sequences

To overcome the limitations of traditional RNNs, particularly in handling long sequences, gated variants such as Long Short-Term Memory (LSTM) [22] units and Gated Recurrent Units (GRU) [23] have been developed. These models incorporate mechanisms that allow them to regulate the flow of information, which helps in maintaining stability in the gradient across long sequences and thereby effectively addressing the vanishing gradient problem.

LSTMs introduce a sophisticated gating mechanism that controls the memory cell, which can store information over long periods. The gates in an LSTM — input, output, and forget gates — determine whether to retain or discard information, thus enabling the model to make selective decisions about maintaining relevant information through time:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad \text{(forget gate)} \tag{2.3}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad \text{(input gate)} \tag{2.4}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad \text{(candidate cell state)} \tag{2.5}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad \text{(cell state update)} \tag{2.6}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad \text{(output gate)} \tag{2.7}$$

$$h_t = o_t * \tanh(C_t) \quad \text{(hidden state update)} \tag{2.8}$$

These equations show how LSTM manages its cell state by filtering information that is either added or removed, providing a control mechanism that mitigates the vanishing gradient issue by preserving gradients through time via the cell state.

A gated recurrent unit (GRU) makes each recurrent unit adaptively capture dependencies of different time scales [58]. Similarly to the LSTM unit, the GRU has gating units that modulate the flow of information inside the unit, however, without having a separate memory cells. The activation $h_t$ of the GRU at time $t$ is a linear interpolation between the previous activation $h_{t-1}$ and the candidate activation $\tilde{h}_t$:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \quad \text{(update gate)} \tag{2.9}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \quad \text{(reset gate)} \tag{2.10}$$

$$\tilde{h}_t = \tanh(W_h \cdot [r_t * h_{t-1}, x_t] + b_h) \quad \text{(candidate hidden state)} \tag{2.11}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \quad \text{(hidden state update)} \tag{2.12}$$

Despite these advancements, LSTM and GRU architectures still face challenges with sequences extending over thousands of steps. They often require additional mechanisms like gradient clipping to control exploding gradients or adding skip connections. Moreover, while they significantly improve the handling of temporal dependencies compared to traditional RNNs, their capacity to model extremely long dependencies is still limited, necessitating the use of enhancements such as attention mechanisms for complex tasks like machine translation. The subsequent section will explore these enhancements in greater detail.

### 2.3.3 The Rise of Neural Attention

An intriguing aspect of the human visual system is its use of attention [59]. Due to our limited capacity to process multiple sources simultaneously, attention mechanisms help us select, focus on, and prioritize information relevant to our behavior. For decades, the concept of attention has been explored in fields like philosophy, psychology, neuroscience, and computing. Inspired by biological attention mechanisms, researchers have developed similar methods to train artificial neural models for improved visual intelligence [60, 61, 62, 63]. Early neural architectures, such as RNNs and Restricted Boltzmann Machines (RBMs) combined with attention mechanisms, showed promising results in image modeling [64].

In natural language processing, the attention mechanism has been applied to Neural Machine Translation (NMT) using a concept called soft attention. Here, attention weights are dynamically recalculated and used while processing inputs with a bi-directional RNN [1]. The key advantage of this method is that it doesn't compress an entire input sentence into a single fixed-length vector. Instead, it encodes the sentence into a sequence of vectors, adaptively selecting subsets of these vectors during translation. This attention mechanism allows additional information to flow through the network, overcoming the information bottleneck of a fixed-size context vector. For our research, the design of attention and the flow of information between the encoder and decoder parts are of particular interest.

Given an input sequence $(x_1, x_2, \ldots, x_T)$, the encoder calculates an annotation $a_j$ for each word $x_j$ in the sequence by concatenating the forward and backward hidden states obtained by a BiRNN [65]. Thus, the encoder maps the input sentence to a sequence of annotations $(a_1, a_2, \ldots, a_T)$. Next, in the decoder part, a weight $\alpha_{ij}$ of each annotation $a_j$ is obtained by using its associated energy $e_{ij}$ that is computed by a feed-forward neural network $f$ as in Eq. 2.13. This neural network $f$ serves as an alignment model that can be jointly trained with the proposed architecture. This alignment model tells us about the relation between the inputs around position $j$ and the output at position $i$. In this way, the decoder applies an attention mechanism. Next, the probability $\alpha_{ij}$ as an output of softmax function (Eq. 2.14), determines the importance of annotation $a_j$ with respect to the previous hidden state $h_{i-1}$. Next, the context vector $c_i$ is computed as a weighted sum of these annotations 2.15. Finally, based on $s_t$ the RNN hidden state from the decoder state,

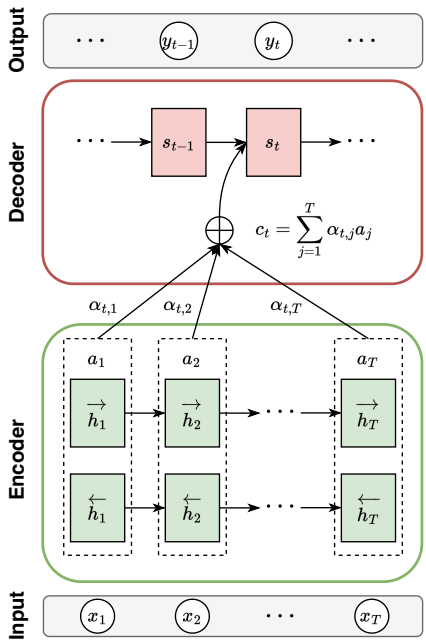



Figure 2.1: The attention-based encoder-decoder architecture for Neural Machine Translation proposed by [1]. The model generates the $t$-th target word $y_t$ given a source sentence $(x_1, x_2, \ldots, x_T)$. The figure is a modified version of the Fig. 1 from [1].

$c_t$ the context, and $y_{t-1}$ the previous generated word, the target word $y_t$ is predicted (Fig. 2.1).

$$e_{ij} = f(h_{i-1}, a_j) \tag{2.13}$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \tag{2.14}$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} a_j \tag{2.15}$$

The described architecture has exceptional importance in popularizing the attention-based architectures [64, 66]. Many more adaptations and modifications of the proposed attention variant were introduced across several applications [67, 68, 69, 70]. The most important works were to extend the attention-based models family by proposing global (always attending to all source words) and local (looks at a subset of source words at a time) attention [71], as well as self-attention [72], a successful extension of the Long Short-Term Memory architecture with a Memory Network [73] in place of a single memory cell.

The important drawback of these architectures comes from the RNNs core property (sequential processing from token to token), each annotation $a_i$ contains information about the input sequence with a strong focus on the parts surrounding the $i$-th word of the input sequence. Thus, the locality is inherently presented in the architecture, restricting its ability to learn long-term dependencies in sequential data.

### 2.3.4 Transformer

After the proposed attention mechanisms, researchers published studies that mostly modified or implemented them to different tasks [66]. However, in 2017, a novel neural network architecture, namely the Transformer, based entirely on self-attention was presented [2]. The Transformer achieved great results on two machine translation tasks in addition to English constituency parsing. The most impressive point about this architecture is that it contains neither recurrence nor convolution. The Transformer performs well by replacing the conventional recurrent layers in the encoder-decoder architecture used for NMT with self-attention.

The original Transformer architecture is composed of encoder-decoder stacks of multiple identical layers, each combining a proposed multi-head self-attention mechanism with a multi-layer perceptron or Feed-Forward block, incorporating residual connections and layer normalization to facilitate training of deep architectures [2]. Transformers can primarily be used in three ways, namely: (1) encoder-only (e.g., for classification: BERT [74], RoBERTA [75], ALBERT [76]), (2) decoder-only (e.g., for language modeling: GPT

[77, 78], LLaMA [79], MISTRAL [80]), and (3) encoder-decoder (e.g., for machine translation: T5 [81], M2M-100 [82]). In representation learning, the encoder part plays a key role in extracting powerful features from raw input. For that reason, we focus primarily on the encoder-only setting, analyzing the structural elements of the Transformer blocks. Moreover, the main bottleneck of the model lies in the self-attention part, which similarly appears in both encoder and decoder blocks.

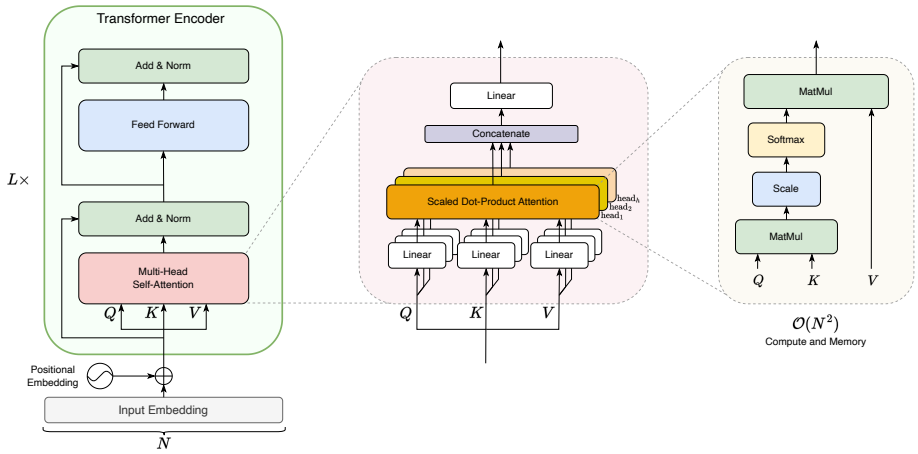



Figure 2.2: The original Transformer encoder architecture proposed by [2]. We do not include the decoder part in the plot. The main bottleneck appears in the Scaled Dot-Product Attention part, where the dot-product operation $QK^T$ results in quadratic memory and computing cost. The figure is a modified version of the Fig. 1 from [3].

The calculations of self-attention are slightly different from the mechanisms described so far in this work. It uses three vectors namely query, key, and value for each word. These vectors are computed by multiplying the input with weight matrices $W^Q$, $W^K$, and $W^V$ which are learned during training. In general, each value is weighted by a function of the query with the corresponding key. The output is computed as a weighted sum of the values. The proposed variant of the attention mechanism called scaled dot-product attention, calculates the dot products of the query with all keys, and then the result is divided into the square root of the dimension of the keys. They pass into the softmax function, thus the weights for the values are obtained, ready to be multiplied with the corresponding values $V$ (Eq. 2.16).

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.16}$$

where $Q = XW^Q, K = XW^K, V = XW^V$ are the query, key, and value representations of the input $X$, and $d_k$ is the scaling factor for the dot products.

The second proposed component is called multi-head attention. Instead of performing a single attention function with $d_{\text{model}}$-dimensional representations, it is beneficial to linearly project the queries, keys, and values $h$ times with different, learned linear projections to $d_q$, $d_k$, and $d_v$ dimensions, respectively (See Fig. 2.2). This modification allows the model to simultaneously learn various types of relationships while keeping the same computational costs. The attention function is performed in parallel on these projected versions of queries, keys, and values, i.e., heads. In this way, $d_v$-dimensional output values are obtained. To get the final values, they are concatenated and projected linearly through matrix $W^O$.

The ability to scale to long sequences is the main bottleneck of the Transformer. The memory and computational complexity required to compute the attention matrix is quadratic in the input sequence length, i.e., $N \times N$. In particular, the $QK^T$ matrix multiplication operation alone consumes $N^2$ time and memory. Together with the limitations of the current hardware, this restricts the overall utility of self-attentive models in applications that demand the processing of long sequences. This problem has been among the main research directions within the architecture design and drove advances in ways how to make the Transformer work cheaper and increase its capabilities to model long-range interactions within long sequences.

### 2.3.5 More Efficient Transformers

This subsection outlines a general taxonomy of efficient Transformer models, characterized by their core techniques and primary use case. We will cover several main classes of Transformer modifications found in literature and describe in detail two main streams of research, that are closely related to our research: methods that sparsify the attention matrix (either local windows or fixed strides) and methods that apply low-rank matrix factorization

techniques (both low-rank decomposition and kernel methods).

### Sparse and local window attention

Fixed patterns in attention provide a simple modification to the conventional self-attention mechanism in Transformers, aiming to reduce computational overhead while preserving the model's ability to process sequential data effectively. This method focuses on implementing pre-defined patterns that limit the scope of attention, thereby reducing the complexity and memory requirements.

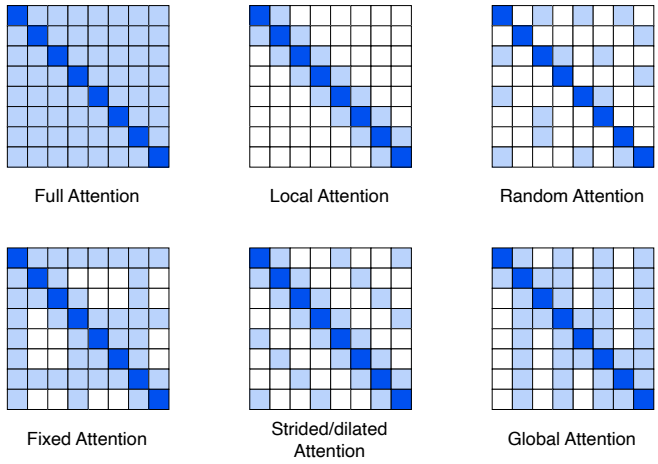



Figure 2.3: Conventional approaches for sparsification of the attention patterns used by several Transformer alternatives. Despite a specialized usage of local (blockwise) attention, several architectures combine different sparsity strategies to reduce costs. For example, Sparse Transformer [4] applies strided and fixed attention, while Longformer [5] uses a combination of local, dilated, and global attention. BigBird [6] utilizes global, random, and local attention.

- **Blockwise Attention**. Blockwise attention is a primary example of fixed patterns in attention. By dividing the input sequence into blocks of a fixed size, the attention mechanism is constrained to these smaller subsets of the sequence. This localized attention cuts down the computational cost from $O(N^2)$ to $O(B^2)$, where $B$ is the block

size and is much smaller than $N$, the length of the input sequence. In practical terms, this can be visualized as transforming a large matrix into a diagonal matrix of smaller blocks, each block representing a localized window within which the self-attention is computed. This method is illustrated in models like the Blockwise self-attention [83] and Local Attention [84]. The application span of such methods is limited to the tasks where local context is sufficient for making predictions, such as certain types of image processing or in language tasks where long-range dependencies are less critical. Moreover, at the boundaries of blocks, the model may fail to properly integrate information due to the abrupt cutoff. This boundary effect can degrade the quality of the model's output, particularly for elements near block boundaries.

- **Strided Attention**. Strided patterns offer a method to sparsify the attention mechanism. The core idea is that by selectively focusing attention at fixed intervals, these models can cover a broader range without the computational expense of a full pairwise comparison across the entire sequence. Notable implementations include the Sparse Transformer [4], which integrates dilated windows to manage longer sequences, and the Longformer [5], designed to handle documents of thousands of words. Strided patterns can lead to uneven coverage of the input sequence, with some areas receiving less attention than others. This can result in missing important information that falls between strides. Another disadvantage of such methods lies in the sensitivity to the choice of hyperparameters, such as stride length, which can be difficult to tune optimally for varied content lengths within data domains.

- **Learnable Patterns Attention**. Dynamic and adaptive, learnable patterns such as those employed by Reformer [85] and Routing Transformer [86], use data-driven methods to optimize the efficiency of attention. These models learn to cluster or sort tokens based on their relevance, which allows for more efficient attention mechanisms. Their core algorithms, namely locality-sensitive hashing (Reformer) and k-means token clustering (Routing Transformer) explicitly add bias toward local attention that limits the applications where the long-range dependencies are important. They are complex to implement, often require longer training times and more data, and are at risk of overfitting to training-specific patterns. Effective use of learnable

patterns also depends heavily on careful initialization and regularization to ensure robustness and generalization across different datasets and tasks.

**Low-rank decomposition and kernel methods**

Other popular techniques for enhancing the efficiency of neural attention models involve leveraging low-rank approximations of the self-attention matrix or decomposing the main computations into more manageable forms through kernelization or other approximation techniques. These methods primarily aim to circumvent the computationally expensive full attention matrix calculation, which combines a quadratic complexity with the softmax operation.

- **Low-Rank Attention Approximation**: The low-rank decomposition approach reduces the dimensionality of the attention matrix by assuming a low-rank structure within the $N \times N$ attention matrix. A prominent implementation of this principle is found in Linformer [87]. Linformer's approach suggests that the attention matrix, particularly in some text processing tasks, can be effectively approximated using its largest singular values, implying a fundamentally low-rank structure. It achieves this by projecting the length dimension of the keys and values to a lower-dimensional space, reducing $N$ to $k$. This transformation significantly alleviates the memory complexity associated with self-attention as the $N \times N$ matrix is effectively reduced to $N \times k$. However, challenges arise as not all attention patterns exhibit low-rank characteristics [87], and projecting onto the length dimension can cause unintended mixing of sequence information within a single transformation. This mixing complicates maintaining causal masking and preventing the intermingling of past and future information when computing attention scores.

- **Kernel Methods**: Recent innovations have also explored kernel methods to enhance the efficiency of Transformers, conceptualizing the attention mechanism through the lens of kernelization. This approach, by reformulating the self-attention calculation, avoids directly computing of the $N \times N$ matrix. Kernel methods can also be viewed as a form of low-rank approximation. For instance, the Performer [88] utilizes a

technique known as Fast Attention Via positive Orthogonal Random features (FAVOR+), which approximates the softmax function using kernel methods. The Performer specifically manipulates the query and key matrices to facilitate more efficient computation, sidestepping the direct calculation of the attention matrix by computing transformed versions of queries and keys that allow for quicker multiplication with values. Despite its efficiency, kernel methods are not without drawbacks; they struggle with parallelization across the time dimension during training, especially in autoregressive setups. This limitation can lead to inefficiencies in learning sequence dependencies effectively.

**Other modifications and common problems**

Beyond sparse and low-rank approximations, other enhancements have been explored to address the computational demands of Transformer architectures. Memory augmentation methods such as Set Transformers [89] and ETC [90] introduce external or augmented memory units to effectively increase the model's capacity without proportionally increasing computational complexity. These methods aim to enhance the model's ability to handle longer sequences by providing additional storage for information from distant sequence parts. However, integrating external memory can introduce complexities in training dynamics and increase the overall architectural complexity.

Recurrent variants of Transformers, such as Transformer-XL [91] and Universal Transformer [92], incorporate recurrent processing within the Transformer architecture to better handle sequential data through repeated application of the same set of weights. These models aim to merge the benefits of RNNs and Transformers, potentially reducing the number of parameters by reusing weights across different positions in the sequence. While this approach can mitigate some of the Transformer's computational burdens, it often leads to increased training times and difficulties in capturing long-range dependencies as effectively as the standard Transformer model. Additionally, methods like chunking, where the input sequence is divided into smaller segments or chunks before being processed, can help manage memory usage but may disrupt the model's ability to understand context across chunk boundaries.

Despite these various adaptations and modifications, common problems persist across all methods aimed at optimizing Transformers. One primary issue is that none of these methods has achieved universal applicability or success across all types of tasks. While many papers present promising results on specific datasets or under certain conditions, these methods often fail to generalize across diverse tasks and data sets. This lack of universality is partly due to each method's inherent trade-offs, such as reduced context awareness in sparse attention or potential information loss in low-rank approximations. Moreover, these adaptations can complicate the model architecture, making it harder to tune and scale across different applications. The challenge remains to develop a method that balances efficiency and effectiveness without compromising the model's general applicability and robustness across a wide range of tasks.

# Chapter 3

# Methodology

This chapter delves into the core conceptual and technical contributions of my PhD research, focusing on the development of novel neural attention architectures tailored to effective representation learning from long sequential data. Among the models introduced are PSF-Attn, Paramixer, CDIL-CNN, and, most notably, ChordMixer. These models are designed for effective learning from long sequences of data and are conceptualized as variants of the attention mechanism, albeit with significant deviations from conventional attention architectures, described in the previous chapter. Lastly, we introduce a novel set of tasks specifically designed to test modeling tools in modeling long sequences in very realistic and complex settings: sequences are very long and their length differs a lot.

## 3.1 Rethinking Neural Attention

Although the Transformer architecture has significantly advanced sequential data processing, it is critical to recognize that its softmax-based attention mechanism is not the only method for modeling attention. The Transformer's approach, while powerful, encounters substantial scalability challenges, particularly when dealing with long sequences. These limitations stem from its reliance on a softmax function that scales quadratically with the length of the sequence, leading to prohibitive computational and memory costs [2, 3].

In response to these challenges, the research community has actively explored various modifications to the traditional Transformer architecture. Innovations aimed at approximating or even bypassing the softmax function have yielded several promising models. However, no alternative has universally superseded the performance of the original architecture across all tasks [3, 85].

In our work, we propose to redefine the self-attention mechanism in a more generalized and flexible form:

$$X^{\text{out}} = AV \tag{3.1}$$

where $V = f(X^{\text{in}})$ represents a token-wise transformation of the input sequence $X^{\text{in}}$. The function $f$ can be any neural network layer that conditions the inputs for subsequent attention processing, such as a feed-forward layer or an embedding layer.

Distinct from conventional attention mechanisms, $A$ is not restricted to being a stochastic matrix; its row elements are not required to sum to one. This relaxation grants the model the capacity to adopt a more varied and potentially more powerful set of combinations than the convex combinations typically used in standard attention frameworks. The essence of this generalized approach lies in the mixing matrix $A$, which dictates the interaction between elements in $V$ to produce the output $X^{\text{out}}$.

The limitation caused by softmax has also been observed in several recent studies. Even though a popular way is to approximate the original softmax attention by low-rank decomposition or kernelization, many studies studied the necessity of the softmax itself. One of the workaround solutions proposed in cosFormer, where the probability constraint of $A$ was relaxed to a non-negative valued matrix, shows that competitive attention can be achieved without strict softmax normalization [93]. F-Net uses the Fourier transform to encode positional information, providing a method to compute attention through convolutions in the frequency domain, thereby sidestepping the need for softmax normalization [94]. Further, a body of research has demonstrated the efficacy of replacing softmax dot-product attention with unrestricted mixing mechanisms, such as those composing MLP-Mixer [95], ConvMixer [96], Poolformer [97] or even utilizing non-learnable shifting operations as a mixer engine [98], while maintaining competitive performance.

By decoupling from the traditional softmax-based framework, our approach seeks not only to mitigate the scalability issues of the Transformer but also to broaden the horizon for sequence modeling. This potentially paves the way for novel neural network architectures that can process extremely long and complex data sequences more efficiently.

## 3.2 Parameterizing Mixing Links

This section introduces our initial effort to achieve a scalable and efficient approximation of the mixing matrix $A$. PSF-Attn as a core mixer engine of Paramixer avoids traditional dot-product mixing and softmax restrictions. Our method innovatively parameterizes the mixing links in the attention matrix $A$ through several sparse factors, ensuring that each factorizing matrix is full-rank. We explore two strategies for defining the non-zero positions in these sparse factors, both yielding an economical approximation of the full attention matrix with computational costs reduced to $O(N \log N)$. Consequently, this framework is adept at handling long sequences.

### 3.2.1 Sparse Factorization Approach

Matrix factorization is a widely used approach that approximately factorizes a square matrix into cheaper matrices. In conventional matrix factorization, the factorizing matrices must be low-rank, as seen in Truncated Singular Value Decomposition (TSVD) and Nyström approximation [99, 100]. However, these conventional approaches do not work well if the square matrix in the approximation is not low-rank.

Unlike the conventional low-rank approaches, we address the quadratic computational challenge of traditional attention mechanisms by applying a sparse factorization of the square mixing matrix $A$:

$$A = \prod_{m=1}^{M} W^{(m)}, \tag{3.2}$$

where each $W^{(m)}$ is a full-rank sparse matrix (See Fig.3.1, Lower). We parameterize the

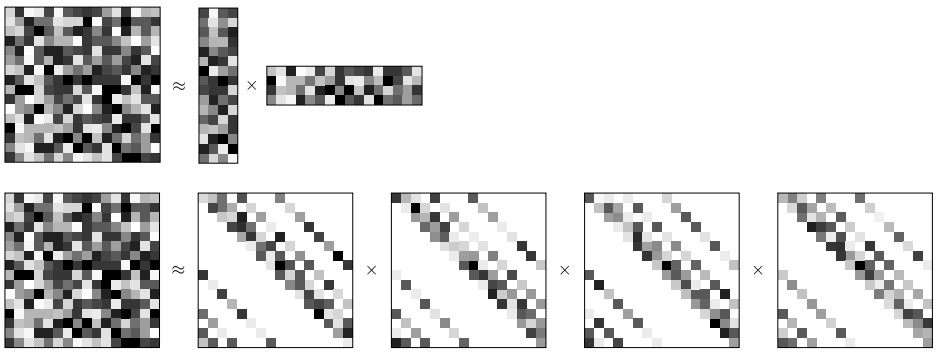


Figure 3.1: (Upper) Two-factor low-rank factorization of the square matrix $A$ for $N = 16$. (Lower) Our proposed Sparse Factorization approach with a Chord protocol. Grayscale squares in the factorizing matrices represent stored entries (non-zeros and explicit zeros), and white squares represent non-stored entries (implicit zeros). Each factorizing matrix is full-rank and sparse. (Modified version of Fig.1 and Fig.2 of Publication A.)

non-zero entries of each row in $W^{(m)}$ as follows:

$$W^{(m)}_{i:} = f^{(m)}(X_i; \theta_m), \tag{3.3}$$

where $f^{(m)} : \mathbb{R}^d \to \mathbb{R}^K$ is a Multilayer Perceptron (MLP) that outputs the $K$ non-zero entries of each row. Here, $i = 1, \ldots, N$, and $K$ is significantly smaller than $N$, reducing the overall non-zero entries and thus the computational load. We explored two different ways (protocols) to define the sparsity pattern of each factorizing matrix, described in the next section.

### 3.2.2 Protocols for Sparse Structure

Each factor $W^{(m)}$ can be treated as an adjacency matrix of a directed graph, where the $(i, j)$-th non-zero entry can be seen as a link from $i$ to $j$. We detail two protocols for specifying the sparse structure in $W^{(m)}$, both aiming to maintain effective connectivity while being computationally efficient. Both protocols share common principles: a high level of sparsity

and circular dilated mixing. The graph visualization and the parameterization are illustrated in Figure 3.2.

**CHORD Protocol:** Inspired by the CHORD peer-to-peer lookup protocol [101], the non-zero pattern of each row in $W^{(m)}$ is structured as follows:

$$W_{ij}^{(m)} = \begin{cases} f_{i1}^{(m)} & \text{if } j = i \\ f_{ik}^{(m)} & \text{if } j = i + 2^{k-2} \mod N \\ 0 & \text{otherwise,} \end{cases} \tag{3.4}$$

where $j = 1, \ldots, N$ and $k = 1, \ldots, K$. Each row has $K$ non-zeros, and in total, all $W^{(m)}$'s have $MNK$ non-zero values. We use $M = \log N$ and $K = M$, storing $N \log^2 N$ entries in total. As a result of using the Chord protocol, each factorizing matrix has an identical sparsity structure (See Figure 3.2, a & b).

**CDIL Protocol (Circular Dilated):** Drawing from Temporal Convolution Networks (TCN), the CDIL protocol modifies dilated connections symmetrically along a circular arrangement to ensure complete and balanced receptive fields. Similarly, each row in $W^{(m)}$ is parameterizing as following:

$$W_{ij}^{(m)} = \begin{cases} f_{i1}^{(m)} & \text{if } j = i \\ f_{ik}^{(m)} & \text{if } j = i + p_{k-1} 2^m \mod N \\ 0 & \text{otherwise,} \end{cases} \tag{3.5}$$

where $p = [1, 2, \ldots, \frac{K-1}{2}, -1, -2, \ldots, -\frac{K-1}{2}]$ and $k = 2, \ldots, K$. Each factor contains $KN$ non-zero entries, and in total, there are $KN \log N$ non-zeros if $M = \log N$, which is more economical than CHORD if $K < \log N$. Differently from the CHORD protocol, the dilation $2^m$ varies at different $m$'s resulting in different sparsity within the factorizing factors (See Figure 3.2, c & d).

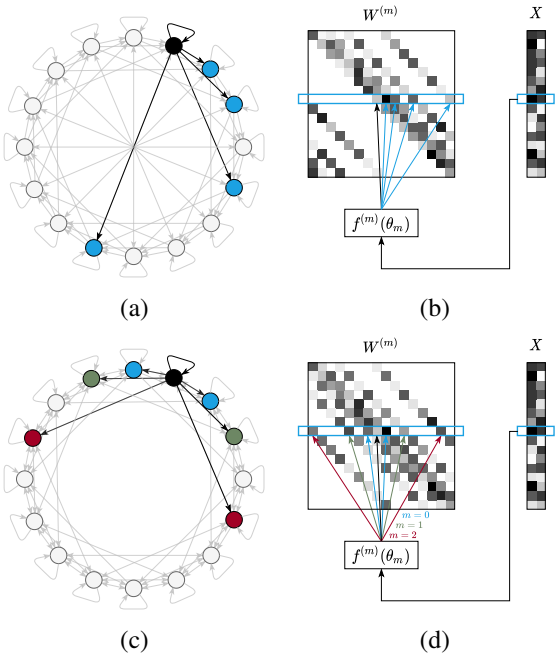



Figure 3.2: Illustration of (a & b) the CHORD and (c & d) the CDIL protocols for $N = 16$. Each node in the circular graph represents a sequence element. The links between nodes correspond to the non-zero entries in $W^{(m)}$ (here $m = 1$) output from $f^{(m)}$. (Modified version of Fig. 1 of Publication B.)

### 3.2.3 Paramixer Network

The Paramixer architecture utilizes the described protocols to define the connectivity patterns within the sequence, illustrated in Figure 3.2. After applying multiple sparse factors, the graph provably becomes fully connected, ensuring that each element in the sequence can influence every other element, facilitating the comprehensive integration of long-range contextual information across the entire sequence.

A Paramixer block transforms an $N \times d$ tensor $X$ + PE (input sequence with added positional encoding) into another tensor of the same size (See Figure 3.3). Here we use a

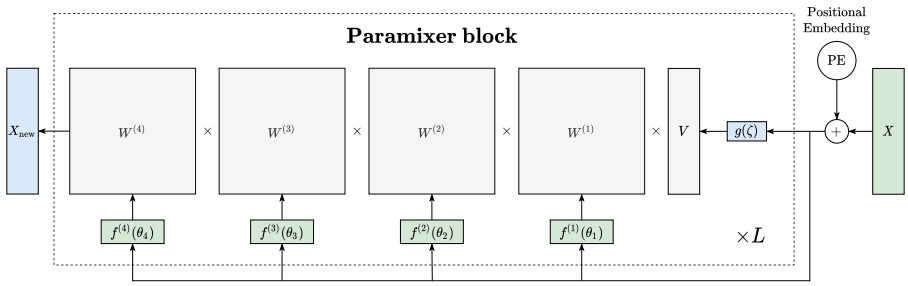



Figure 3.3: Illustration of an example Paramixer neural network ($M = 4$). After adding the positional embedding, it applies $L$ identical Paramixer blocks and obtains the transformed tensor $X_{\text{new}}$. (Fig. 2 of Publication B.)

neural network $g : \mathbb{R}^d \mapsto \mathbb{R}^d$ with weights $\zeta$ to mix the columns of the input tensor. As mixers for both $f^{(m)}$'s and $g$, we use simple two-layer MLPs, each with GELU activation. Stacking $L$ such blocks forms a deep neural network backbone for representation learning on long sequential data.

## 3.3 Circular Dilated Convolutions

As we explore innovative approaches to model the attention matrix for long sequences, circular dilated convolutions (CDIL-CNN) emerge as a compelling candidate. Drawing inspiration from Temporal Convolutional Networks (TCN), CDIL-CNN integrates symmetric dilations that increase exponentially and circular padding to efficiently handle the complexities of long sequence classification tasks.

Employing a sparsity pattern in the attention matrix approximation, similar to the Chord protocol, has demonstrated substantial efficacy in scaling to long sequences. A fundamental distinction between the Chord and CDIL protocols lies in their respective sparsity patterns. The Chord protocol achieves multiscale dilations within a single layer, facilitating diverse contextual integrations simultaneously. In contrast, the CDIL protocol incrementally increases the scales of sparsity across different layers, enhancing the model's ability to adapt

to the increasing data complexity as it moves deeper into the network. This kind of sparsity can be implemented using dilated convolutions. Dilated convolutions, which are pivotal in TCN [102], adapt this concept by using dilated, causal 1D convolutional layers that maintain the same input and output lengths, traditionally employed to prevent future information leakage in sequence regression and time-series forecasting. In contrast, CDIL-CNN employs symmetric convolutions that permit each output at timestep $t$ to gather information from both preceding and succeeding sequence elements, enhancing the model's capacity for tasks where comprehensive sequence context is vital for accurate predictions.

The mathematical formulation for the convolutional output at each timestep $t$ in a layer $l$ is given by the following equation:

$$b_t = \sum_{k=0}^{K-1} w_k^{(l)} \cdot a_{t+\left(k-\frac{K-1}{2}\right)\cdot d_l} \tag{3.6}$$

where $K$ represents the kernel size, typically an odd number, and $w^{(l)}$ are the convolution coefficients for that layer. The increasing dilation factor $d_l = 2^{l-1}$ at each layer enables the network to rapidly expand its receptive field, thereby accommodating exceptionally long sequences efficiently. This strategy leverages the proven effectiveness of dilated convolutions across various domains, including image and sequence processing [103, 104, 105].

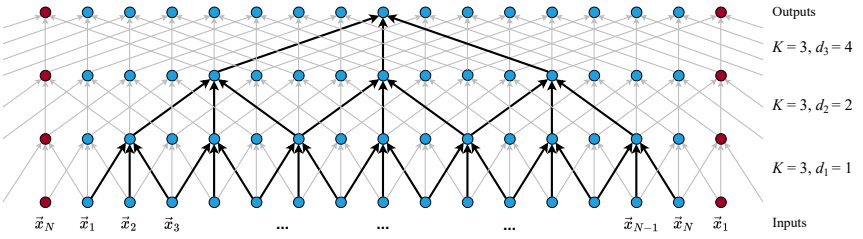



Figure 3.4: Illustration of a CDIL-CNN network with four layers, showcasing symmetric dilated convolutions and circular padding. Blue circles represent original or transformed sequence elements; red nodes highlight the circular padding where elements from the sequence's opposite end are used. (Modified version of Fig. 1 of Publication C.)

Circular padding, another crucial feature of CDIL-CNN, addresses the common issues

associated with zero-padding at sequence boundaries, which can distort model performance and affect generalization. By ensuring that convolution operations at sequence edges incorporate data from the sequence's opposite ends, circular padding provides a consistent and holistic sequence representation. This technique not only enhances the model's robustness against input shifts but also ensures uniform treatment across all sequence positions, reinforcing the concept of the sequence as a continuous loop.

The Circular Dilated Convolutions offer a straightforward implementation—the main mixing engine utilizes a 1D-Convolutional layer, a standard feature in the Pytorch library. However, this network and the CDIL protocol come with several limitations. First, the use of varying dilations across different layers mandates the use of exactly $\log_K N$ meta-blocks for sequences of length $N$, constraining the potential for deeper network architectures. Another issue, which also affects the Paramixer methodology, is the inability to process sequences of highly different lengths without resorting to padding. This drawback is inherited from the inflexibility of convolutional operations, which require input and output tensors of consistent sizes. In the upcoming section, we will introduce a more adaptable approach that overcomes these drawbacks, offering a robust framework for sequence modeling that improves performance and speeds up data processing.

## 3.4 ChordMixer

This section proposes the attention architecture that is the main focus of our methodology called ChordMixer. This design outperforms the alternative attention mechanisms, including the Transformer and its more efficient variants. Moreover, compared to the previously proposed methods, Paramixer and CDIL-CNN, it opens a new core ability — to model attention for sequences with variable length. We start with discussing the properties of a good attention mechanism. Then we describe the ChordMixer design, its complexity analysis, and its properties in detail. Finally, we delve into the design of the newly proposed benchmarks that involve extremely long sequences with highly variable lengths.

### 3.4.1 Properties of good attention mechanisms

Our development of ChordMixer targets the advancement of attention mechanisms beyond traditional scaled dot-product methods. We formulate and focus on the following essential properties:

- *Full Receptive Field*: Every output element should be influenced, directly or indirectly, by all input elements in the sequence. This property is critical for tasks where every element can impact the target output. While most models achieve a full receptive field, those employing uncontrolled sparsity, such as random attention, do not.
- *Scalability*: The method should accurately handle very long sequences. This is a significant limitation of current methods, including the original Transformer, which suffers from computational and memory overheads, and RNN-based approaches, which struggle in these scenarios.
- *Decentrality*: The design should be decentralized or symmetric, ensuring no sequence element or position is more central or closer to the output. This property is essential for tasks requiring long interactions between elements. Convolutional networks, RNNs, and SSMs face challenges here due to their strong locality biases. Many Transformer approximations, such as those using local windows or asymmetric sparsity patterns, also fail to achieve decentrality.
- *Length Flexibility*: The model should handle sequences with varying lengths without the need for preprocessing steps like chunking or resampling. While standard Transformer and RNN-based methods show great flexibility, convolutional methods, including CDIL-CNN, require heavy padding or chunking. Transformer approximations using sequence chunking face similar issues.

To our knowledge, no existing method achieves all four properties. The original Transformer comes closest but lacks scalability for long sequences. Methods approximating dot-product self-attention and RNN-based approaches largely sacrifice decentrality, while

convolutional methods struggle with length flexibility. In the next sections, we demonstrate how ChordMixer achieves all four properties, enabling efficient processing of very long sequences with high length variability.

### 3.4.2 Chordmixer Architecture Design

#### Integrating a Solution from P2P Networks

To achieve the desired properties, we draw inspiration from the Peer-to-Peer (P2P) communication network [101]. In a naive full-connection protocol, every peer communicates with the other $N-1$ peers, resulting in huge routing tables and inefficient communication. This is analogous to the original Transformer's attention mechanism, which faces scalability issues due to its all-to-all communication.

A more efficient solution is the Chord protocol, where peers are organized in a circle. At each lookup hop, the $i$-th peer communicates with neighbors at multiple scales: $i+2^0, i+2^1, \ldots, i+2^m \mod N$, where $m = \lceil \log_2 N \rceil$. This setup ensures that each peer can collect information from all other peers after $O(\log_2 N)$ hops [101]. This approach satisfies the first three desired properties: (1) the receptive field becomes full, (2) exponentially increasing dilations enable great scalability on long sequences, and (3) identical connections ensure decentralized communication. The proof of full connectivity of the Chord protocol is provided in Paper D, Proposition 3.1.

#### ChordMixer Block

We translate the P2P Chord protocol into an artificial neural network by treating sequence tokens as peers, network blocks as hops, and transformations between blocks as communications. This forms the ChordMixer block.

We divide the sequence rows or channels into $M = \lceil \log_2 N \rceil + 1$ tracks, each with approximately $d/M$ channels. Each ChordMixer block for an input sequence $x^{\text{in}} \in \mathbb{R}^{d \times N}$ comprises two layers:

1. *Rotate*: This layer rotates the tracks at different scales ($z \in \mathbb{R}^{d \times N}$):

$$z_{ij} = x^{\text{in}}_{ik} \tag{3.7}$$

for $i = 1, \ldots, d$ and $j = 1, \ldots, N$, where

$$k = \begin{cases} j & \text{if } t_i = 1 \\ (j + 2^{t_i - 2}) \mod N & \text{if } t_i > 1 \end{cases} \tag{3.8}$$

Here, $t_i$ is the track index of the $i$-th channel, starting from 1. We include a self-loop (non-rotated track) to incorporate the same token in mixing. Figure 3.5 (b)-(d) illustrates the Rotate forward pass.

The Rotate layer is efficient, with parallel copy operations over $i, j, k$. It can be implemented using CUDA or PyTorch's `roll` function. The backward pass involves reverse rotation: $\partial \mathcal{J} / \partial x^{\text{in}}_{ik} = \partial \mathcal{J} / \partial z_{ij}$ for a learning objective $\mathcal{J}$.

2. *Mix*: This layer transforms each token (column) using a neural network $f$:

$$x^{\text{out}}_{:j} = f(z_{:j}; w) \tag{3.9}$$

for $j = 1, \ldots, N$, where $w$ represents the network weights of $f$. In our experiments, we used a two-layer MLP with a hidden dimension $h$ and GELU nonlinearity, applying dropout between the Rotate and Mix steps.

### 3.4.3 Network Architecture and Variable Sequences Processing

A ChordMixer network comprises $\lceil \log_2 N_{\text{max}} \rceil$ ChordMixer blocks, where $N_{\text{max}}$ represents the length of the longest sequence or the maximum range of possible interactions. Each block maintains an identical architecture with $\lceil \log_2 N_{\text{max}} \rceil + 1$ tracks but features a distinct $f$ network. Residual connections are incorporated around each ChordMixer block to facilitate learning.

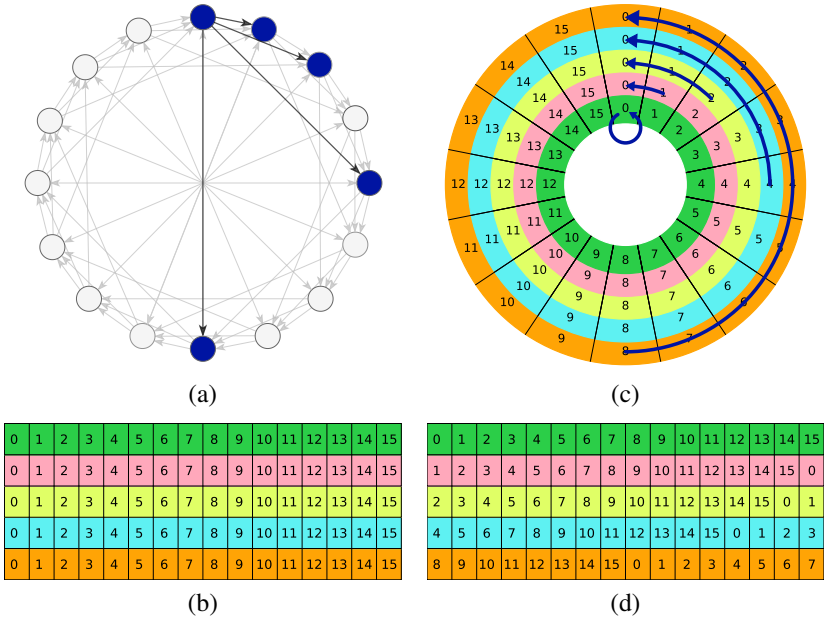



Figure 3.5: Illustration of the Rotate step in ChordMixer for $N = 16$: a) the original Chord protocol, where blue dots are peers, and arrows indicate communications at each hop, b) the sequence before rotation, where the numbers indicate the original position in each track, c) the rotations in different bands, where the arcs indicate the rotation sources of new Position 0, and d) the rotated tracks. Afterward, each column will be fed to the Mix step of ChordMixer. (Fig. 1 of Publication D.)

The ChordMixer network serves as the backbone of a neural learner. After the mixing process, elements at every position encapsulate essential information from all input positions. We can subsequently apply a lightweight predictor (e.g., classifier or regressor) to the individual output tokens and integrate their predictions using ensemble learning. In this study, we utilize linear predictors and ensemble averaging. Due to the exchangeable nature of linear operations and averaging, we can first compute the mean of the output tokens and then apply the linear predictor.

A notable feature of the ChordMixer network is its dynamic block arrangement. The number of ChordMixer blocks required for complete mixing depends on the sequence

length. For instance, a sequence of length $1k$ necessitates only 10 ChordMixer blocks, whereas a sequence of length $1M$ requires 20 identical blocks. To ensure efficient batch processing, sequences within the same $\log_2 N$ group should be processed in a batch. To achieve this, we dynamically organize batches where sequences have the same $\lceil \log_2 N \rceil$ value, ensuring that they pass through the same number of blocks. We construct batches dynamically, grouping sequences within the same length range, so the instances in each batch vary from epoch to epoch.

Within each block, we rotate sequences in a batch separately, then concatenate the rotated sequences for MLP mixing only once. Finally, the mixed tensor is de-concatenated into sequences for block output. This method eliminates the need for padding or chunking, thereby fulfilling the last property of effective attention mechanisms and achieving all four desired properties.

**Analysis**

Here, we analyze the complexity of the ChordMixer:

- The Rotate layer is parameter-free, so all learnable parameters in a ChordMixer block are contained in $w$. For the two-layer MLP implementation of $f$, there are $O(dh)$ parameters per block. Thus, a ChordMixer network has $O(dh \log_2 N_{\text{max}})$ learnable parameters, where $N_{\text{max}}$ is the maximum sequence length observed during training.

- The Mix step is the primary contributor to running time. The MLP mixing in each block incurs a cost of $O(dhN)$, resulting in an overall cost of $O(dhN \log_2 N)$ for all blocks.

- The memory cost for processing a sequence of length $N$ through a ChordMixer network is $O(dN \log_2 N)$.

- A ChordMixer block is full-rank on flattened $x_{\text{in}}$ when using two-layer MLPs as $f$. The proof of this is provided in Paper D, Appendix A.

This architecture circumvents the low-rank bottleneck, ensuring unrestricted information flow and broad information exchange between sequence elements. The complexity analysis indicates that the ChordMixer network is economical, with sub-quadratic memory and computational complexity relative to sequence length.

### 3.4.4 Variable and Long Sequences Benchmark Design

A key contribution of this work is the introduction of new benchmarks specifically designed for long sequences with high length variability. One of the core issues in comparing different models is the inconsistency of benchmarks used across studies. Recent efforts like the Long Range Arena (LRA) [7] have attempted to standardize the evaluation of Transformer variants, benchmarking 10 different variants on long-range modeling tasks. However, these benchmarks fall short as the sequences are not long enough (up to 16k) and primarily consist of sequences with uniform lengths within each task.

To address this, we have developed and benchmarked different models on a new dataset comprising three tasks that involve extremely long sequences with significant length variability. Below, we detail the data collection and dataset design processes.

It is crucial to establish common design principles to achieve the desired research outcomes. Firstly, the benchmark should include very long sequences, at least 100,000 tokens in length, which distinguishes it from previous experiments focusing on relatively short sequences. Secondly, the sequences should exhibit highly variable lengths within the benchmark to test the neural attention mechanisms' ability to extract both local and long-range dependencies. Thirdly, sequences from different classes should have identical length distributions to prevent models from exploiting shortcuts based on sequence length alone. Finally, the dataset should be sufficiently large to ensure robust training and testing, allowing for reliable conclusions about generalization capabilities. Only problems that meet these criteria are included in the benchmark, and we evaluate various attention backbones on these tasks.

**Long Document Classification**

This task evaluates the performance of ChordMixer in modeling complex long-term dependencies in lengthy texts. The dataset consists of academic papers sourced from GitHub, categorized into four classes: cs.AI, cs.NE, math.AC, and math.GR, resulting in a dataset of 11,956 documents. We split the dataset into 70% training, 20% validation, and 10% test sets. Unlike previous research [106], we encoded each document at the character level to create a more challenging task. The document lengths range from 4.5k to 131k, with a few documents exceeding this length (up to 2,483,331). For comparison, we truncated documents at 131k tokens, discarding only the tail beyond this point, affecting a negligible 0.47% of documents.

**Synthetic Experiments with Long Sequences**

The Adding problem is a synthetic regression task designed to test the network's ability to detect relevant tokens and perform summation, inspired by [22]. Each input sequence consists of pairs $(a_i, b_i)$, where $a_i \sim U(-1, 1)$ and $b_i \in \{0, 1\}$ for $i = 1, \ldots, N$. Signals are placed at two random positions $t_1$ and $t_2$ such that $b_{t_1} = b_{t_2} = 1$ and $b_i = 0$ elsewhere. The target is $y = 0.5 + \frac{a_{t_1} + a_{t_2}}{4}$. Unlike previous studies [22, 107], we generated sequences with lengths $N = \text{round}(\lambda \cdot \zeta)$, where $\zeta \sim \text{LogNormal}(\mu, \sigma^2)$ with $\mu = 0.5$ and $\sigma = 0.7$. This introduces a left-skewed length distribution with a wide spread, adding complexity to the task. A prediction $\hat{y}$ is considered correct if $|y - \hat{y}| < 0.04$. Figure 3.6 illustrates an example sequence and its length distribution for $\lambda = 16$k.

We used four different $\lambda$ values: 200, 1k, 16k, and 128k, generating 60,000 instances for each $\lambda$. Larger $\lambda$ values correspond to more difficult tasks. For example, $\lambda = 200$ results in sequences up to 6.7k long, while $\lambda = 128k$ results in sequences up to 1.5 million elements. Figure 3.6 (right) shows the sequence length distribution for $\lambda = 16$k.

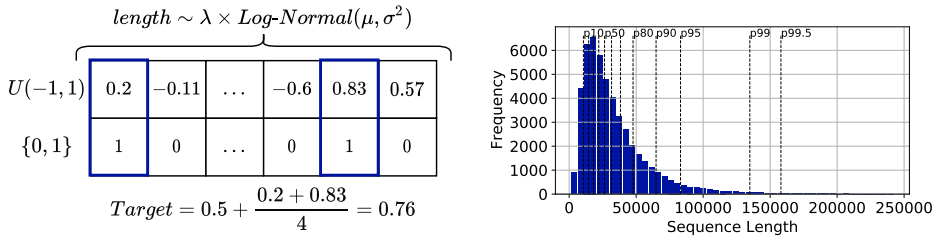



Figure 3.6: (Left) One instance of the Adding problem with variable lengths. Lengths are sampled from a Log-Normal distribution scaled by a base length ($\lambda$). (Right) Example of sequence lengths distribution for $\lambda = 16$k. The dashed vertical lines indicate percentiles of the distribution. (Fig. 2 of Publication D.)

**DNA Sequence-Based Taxonomy Classification**

DNA sequences, naturally exhibiting high length variability, were sourced from the GenBank database, an annotated collection of publicly available DNA sequences. The DNA sequences are paired with metadata describing their taxonomic classification. We created binary classification tasks ensuring meaningful comparisons while satisfying the criteria for a robust benchmark.

The detailed procedure for forming the classes and sequences includes:

- **Data Downloading**: We downloaded DNA sequences in FASTA format for eight organism categories: Bacterial, Other Mammalian, Plant, Primate, Rodent, Viral, and Other Vertebrate, along with their taxonomic labels and structure.
- **Phylogenetic Tree Construction**: Each sequence ID was classified as a node in a phylogenetic tree, showing evolutionary relationships based on genetic similarities and differences.
- **Sequence Classification**: Each sequence was labeled according to its position in the phylogenetic tree.
- **Initial Filtering**: To avoid classification based on simple sequence differences, we

selected binary classes (genera) within the same family, ensuring close evolutionary relationships and increased task complexity.

- **Final Selection**: We selected genera with similar sequence length distributions and sufficient, comparable class sizes. This ensures a balanced representation of sequence lengths, reducing the likelihood of skewed decision boundaries based on length alone.

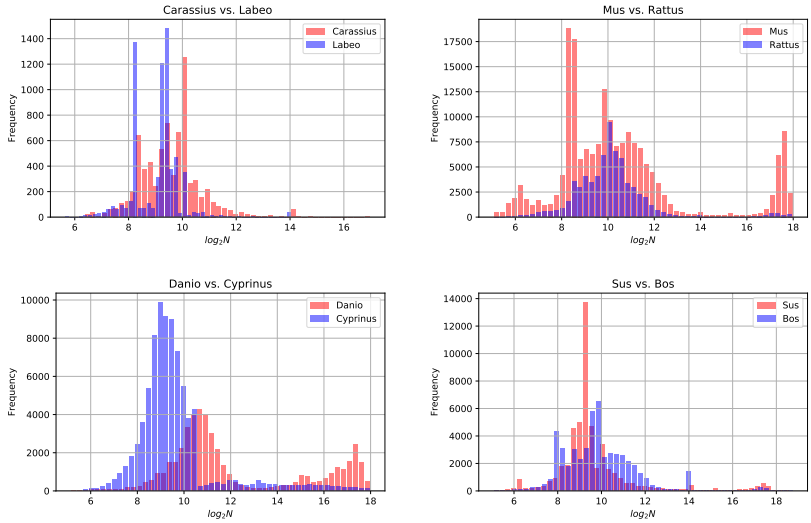



Figure 3.7: Sequence lengths histograms for the selected DNA-based taxonomy classification tasks. (Fig. 7 of Publication D.)

As a result, we identified four binary classification tasks that meet all criteria for a robust DNA sequence-based classification benchmark:

1. **Carassius vs. Labeo**: This task includes 7,263 genes from Carassius (a type of fish) and 6,718 from Labeo (another fish species). Sequence lengths range from 79 to 100,101 base pairs, making it the easiest problem within this section.

2. **Mus vs. Rattus**: This dataset, the largest at the genus level, classifies genes as either

rat (Rattus) or mouse (Mus). It includes 275,636 Mus sequences and 133,310 Rattus sequences, with lengths ranging from 32 to 261,093 base pairs.

3. **Danio vs. Cyprinus**: Both genera belong to the cyprinidae family, with Danio being small freshwater fish and Cyprinus being a genus of carps. The dataset includes 47,135 Danio and 83,321 Cyprinus sequences, with lengths from 34 to 261,943 base pairs.

4. **Sus vs. Bos**: This task classifies genes from domestic pigs (Sus) and cattle (Bos), both vertebrates. The dataset is nearly balanced with 50,027 Sus and 51,973 Bos sequences, but has the most varied lengths, ranging from 63 to 447,010 base pairs.

| **Task Name** | **Sequence Length Statistics** |
|---|---|
| Carassius vs. Labeo | min 79, median 651, max 100K |
| Mus vs. Rattus | min 32, median 979, max 261K |
| Danio vs. Cyprinus | min 34, median 868, max 261K |
| Sus vs. Bos | min 63, median 664, max 447K |

Table 3.1: Sequence length statistics of the datasets in the DNA taxonomy classification experiments.

Based on the sequence length statistics (Table 3.1) and the similarity of the sequence length distributions (Figure 3.7), these tasks are sufficiently challenging to evaluate existing sequence models' ability to handle very long sequences with high length variability.

# Chapter 4

# Research Results

This chapter presents the experimental results that support our methodology and address the research questions outlined in Section 1.2. Each section corresponds to a research question, structured into two main parts: an experimental data description and the modeling results obtained from applying our methods to this data.

## 4.1 Modeling Long Sequences

This section describes the experiments designed to demonstrate our ability to model long sequential data. The experiments encompass a variety of domains, including images, synthetic data, text, DNA sequences, and the Long Range Arena public benchmark. We begin with experiments showcasing our Sparse Factorization technique's ability to approximate large square matrices, a foundational step in modeling the attention matrix. We then move on to specifically designed supervised tasks involving long sequential data and conclude with results from broadly accepted benchmarks on long sequences, comparing our approach to other methods.

### 4.1.1 Non-parametric Experiment

In our research, we explore whether our Sparse Factorization (SF) method can compete with conventional low-rank matrix decomposition techniques, such as those used in Linformer and Performer, which are discussed in detail in Section 2.3.5. These standard methods have been benchmarks for approximating square attention matrices; therefore, our initial step is to compare our approach against these to establish a baseline performance.

This experiment is designed to assess whether our Chord Sparse Factorization approach can not only match but potentially exceed the performance of truncated singular value decomposition (TSVD), a well-regarded method for matrix factorization. Our focus is on demonstrating two key capabilities of our approach: firstly, achieving superior error metrics with an equivalent number of non-zero matrix entries compared to TSVD, and secondly, identifying which types of matrices are best suited for our method.

To test our SF method, we selected $256 \times 256$ grayscale images as our test matrices, allowing us to visually and quantitatively assess the effectiveness of our approach. The choice of images as matrices helps illustrate the practical implications of our method in real-world applications where image data is prevalent.

For each image, we performed a side-by-side comparison of our method against TSVD, focusing on the F-norm approximation errors, which measure the accuracy of the matrix approximations. Below each image in Figure 4.1, we display these errors, highlighting which method performs better.

The experiment reveals interesting insights about the performance of TSVD and SF under different conditions:

- Images like the `chess` board, which are inherently low-rank due to their simple, repetitive patterns, are better handled by TSVD.
- Images such as `Lena` and `apple`, which primarily contain low-frequency details, also see better or comparable performance from TSVD.
- Conversely, images rich in high-frequency details, such as `skyscraper`, `lostdoor`, and `ship`, demonstrate where SF excels, outperforming TSVD. This indicates that

SF is particularly advantageous for matrices that capture complex, detailed patterns, which are often challenging for traditional low-rank approximations.

To further analyze this, we processed the images to highlight their high-frequency components by computing their gradient magnitudes. This transformation accentuates edges and fine details, providing a starker contrast between areas of uniform color and those with detailed features. Our results confirm that SF provides lower approximation errors in these high-detail areas, underscoring its suitability for complex matrices that defy simple low-rank approximations.

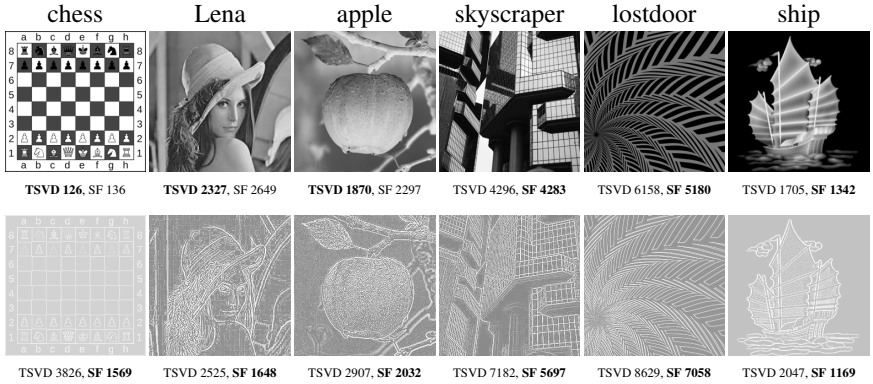



Figure 4.1: Example square matrices: (top) original images and (bottom) the gradient magnitude images (displayed after histogram equalization for better visibility). Image names are approximation errors by using TSVD and SF with the same number of non-zeros shown below the images. Boldface font indicates the winner for each case. (Modified version of Fig.4 of Publication A.)

In summary, our results show that TSVD does not outperform SF when using the same number of non-zeros for approximating square matrices. The cases where SF outperforms TSVD suggest that SF is particularly effective for matrices that are: 1) sparse, 2) intrinsically high-rank, or 3) rich in high-frequency details. These findings indicate that parametric sparse factorization (PSF-Attn, Paramixer) could be an effective alternative to dot-product attention and its low-rank approximation variants. In the next section, we will demonstrate its effectiveness using synthetic experiments with long sequences.

### 4.1.2 Synthetic experiments with fixed-length sequences

Here we test whether a model based on the Chord Protocol, Paramixer is scalable to approximate large attention matrices. We have used two synthetic data sets composed of long sequences for supervised learning tasks. A similar experimental setup appeared in [22] for scalability tests. The details of the data sets and tasks are given below:

- **Adding Problem**: A sequence regression task where each sequence element is a pair of numbers $(a_i, b_i)$, with $a_i$ uniformly distributed between -1 and 1, and $b_i$ being a binary indicator. Two randomly selected positions, $t_1$ and $t_2$, are marked by setting $b_{t_1} = b_{t_2} = 1$ (with all other $b_i = 0$). The target output $y$ is defined as $0.5 + \frac{a_{t_1}+a_{t_2}}{4}$. The task challenges the model to accurately identify and sum the values at positions $t_1$ and $t_2$, regardless of their distance within the sequence.

- **Temporal Order**: A sequence classification task involving sequences composed of symbols from the set $\{a, b, c, d, X, Y\}$. Symbols $X$ and $Y$ are signals, while the rest serve as noise. Each sequence contains exactly two signal symbols at arbitrary positions, determining one of four possible ordered classes based on their arrangement $(X, X)$, $(X, Y)$, $(Y, X)$, and $(Y, Y)$.

For each task, sequences with varying lengths were generated, ranging from $N = 128$ to $N = 2^{15}$, doubling in length at each increment. Each sub-task comprised 100,000 training sequences and 5,000 testing instances.

We benchmarked the performance of PSF-Attn against several prominent scaled dot-product attention models known as X-formers—specifically, Linformer [108], Performer [109], Reformer [85], and Nyströmformer [8], as well as the original Transformer [2]. Models were implemented using open-source PyTorch frameworks [1].

Results depicted in Figure 4.2 show that while all models perform comparably well with short sequences ($N \leq 256$), disparities emerge as sequence length increases. In the Adding problem, traditional Transformer models exhibit declining performance with increasing

[1] available at `https://github.com/lucidrains`

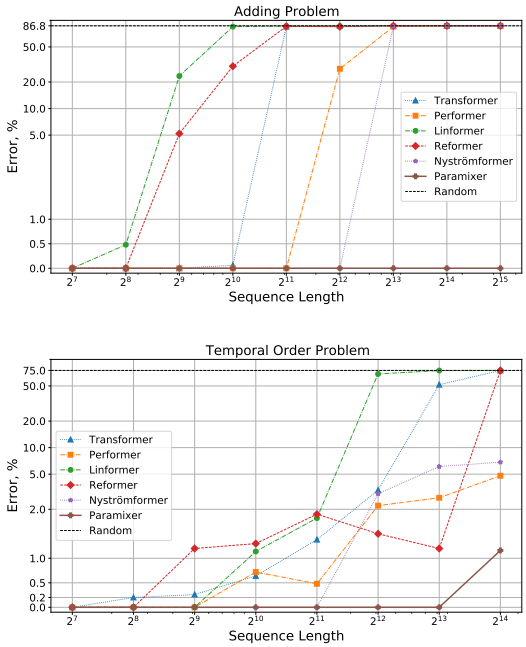



Figure 4.2: Error percentage of Paramixer and the X-formers for (top) the Adding problem and (bottom) the Temporal Order problem with increasing sequence lengths. (Fig.3 of Publication B.)

sequence length, where error rates surge notably for $N$ beyond 1024. In contrast, Paramixer maintains robust accuracy even at much longer lengths, showcasing superior scalability.

In the Temporal Order task, although most models maintain high accuracy up to $N = 256$, performance begins to taper off beyond $N = 512$. Notably, Paramixer continues to demonstrate high accuracy, achieving nearly perfect scores up to $N = 8192$ and significantly outperforming other models at $N = 16384$.

Overall, Paramixer's scalability and performance on tasks with extended sequence lengths suggest the potential applicability of the Sparse Factorization approach with the Chord protocol to real-world scenarios involving extensive sequential data. The next sections will extend this evaluation to text and DNA sequence analysis tasks.

### 4.1.3 Text problem with fixed-length sequences

This subsection explores the effectiveness of Paramixer in handling extended text sequences for natural language processing (NLP) tasks. Character-level text classification tasks provide a robust platform to evaluate model performance over long sequences, offering a substantial increase in complexity compared to typical word-level NLP benchmarks.

We employed the Long-document dataset, a publicly accessible compilation of academic papers sourced through the arXiv sanity preserver program and available on GitHub [106]. This dataset spans eleven distinct academic fields. For our experiments, we focused on four categories: cs.AI, cs.NE, math.AC, and math.GR, comprising a total of 11,956 documents. We converted each document into a character sequence, resulting in an average sequence length of 52,112 characters, and partitioned the data into 70% training, 20% validation, and 10% testing sets.

Table 4.1: Test accuracies on long document classification.

| Model | $N = 16k$ | $N = 32k$ |
|---|---|---|
| Transformer | 25.62 | 25.62 |
| Linformer | 64.69 | 65.36 |
| Performer | 25.62 | 25.20 |
| Reformer | 25.04 | 25.04 |
| Transformer-LS | 70.25 | 60.93 |
| Nyströmformer | 71.70 | 67.69 |
| Paramixer | **83.89** | **84.55** |

To investigate the impact of sequence length on model performance, we standardized the sequence lengths to 16,384 and 32,768 characters. This approach tests the model's capacity to leverage longer textual contexts, a known advantage in NLP.

Table 4.1 demonstrates that Paramixer substantially outperforms other transformer-based models by margins of 12.19% and 16.86% at the two sequence lengths, respectively.

Notably, Paramixer exhibits enhanced accuracy with the longer sequence, indicating superior generalization capabilities. This performance suggests that Paramixer effectively utilizes extended contexts, in contrast to other models like Linformer, Transformer-LS, and Nyströmformer, which struggle with longer sequences, revealing potential deficiencies in their handling of high-rank NLP tasks due to low-rank factorization approaches.

### 4.1.4 DNA experiments with fixed-length sequences

With the rising application of deep learning in genomic studies, such as chromatin-profile prediction and genome analysis, challenges persist in processing extensive genomic and protein sequences directly due to their length. Traditional preprocessing methods typically involve segmenting these long sequences into smaller sections, a process that potentially omits significant long-range interaction information.

Table 4.2: Performance of Paramixer compared to other Transformers on genomic classification tasks, measured by ROC AUC (%).

| Model | HFDNA | MTcDNA |
|---|---|---|
| Transformer | 94.32 | 79.38 |
| Linformer | 91.65 | 77.06 |
| Performer | 93.46 | 78.92 |
| Reformer | 92.32 | 74.94 |
| Transformer-LS | 93.19 | 75.81 |
| Nyströmformer | 92.26 | 71.98 |
| Paramixer | **100.00** | **84.86** |

To address these challenges, we utilized Paramixer's design, which accommodates entire sequences without preliminary segmentation, thereby preserving the integrity of long-distance interactions. We created two datasets for this purpose: one from NONCODEv6 [110], consisting of lengthy human and fruitfly DNA sequences, and another comprising mouse and turtle cDNA sequences from the Ensembl genome browser [111, 112].

Both datasets, named HFDNA and MTcDNA, were designed to test binary classification tasks with sequences uniformly adjusted to a length of 16,384. This standardization facilitates direct comparisons across models.

As shown in Table 4.2, Paramixer achieves outstanding performance, notably a 100% ROC AUC score on the HFDNA dataset, significantly outperforming all other models. This superior performance is also evident in the MTcDNA dataset, where Paramixer surpasses the next best model by a substantial margin. These results not only validate the effectiveness of Paramixer in handling very long sequences but also underscore its potential in facilitating advanced biological research that requires detailed modeling of complex genetic interactions.

### 4.1.5 Long Range Arena Public Benchmark

This section discusses the performance of our proposed methods on the Long Range Arena (LRA), a publicly available benchmark designed for assessing the capability of models to handle long sequential data [7]. The LRA benchmark includes six diverse datasets from different modalities including vision recognition, synthetic tasks, mathematical operations, and natural language processing. We present results for our methods alongside comparisons with both traditional transformer-based models and other novel mixing mechanisms.

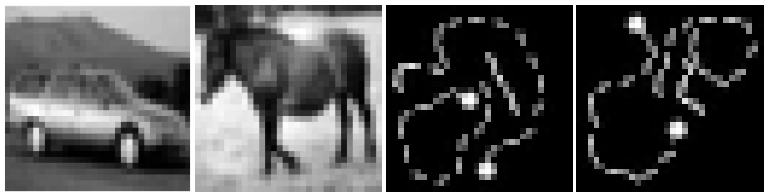



Figure 4.3: Sample data from the Image Classification and Pathfinder tasks within the LRA benchmark, illustrating the diversity of challenges presented by the benchmark (Fig 2. of Publication B).

- *ListOps*. This task evaluates a model's ability to parse and evaluate nested operations within sequences [113]. In our extended ListOps, sequences can reach up to 2 000 elements with operations including maximum, minimum, median, and modular summation, challenging the models to manage complex hierarchical structures efficiently.

- *Text Classification*. Based on IMDb reviews [114], this task involves classifying movie reviews as positive or negative at a character level, extending sequence lengths to 4 000 characters, which intensifies the processing challenge.

- *Image Classification*. This involves classifying flattened grayscale CIFAR-10 images [115], treating pixel intensities as token values, thus transforming image recognition into a sequence prediction task with sequences of 1 024 elements.

- *Retrieval*. Focused on identifying shared citations between pairs of documents from the ACL Anthology Network, this task doubles the sequence length to 8 000 when considering document pairs, intensifying the demand on memory and processing power.

- *Pathfinder*. A task inspired by cognitive psychology, where models must determine if a path exists between two points in a synthetic image, treating the image as a sequence of 1 024 pixel values.

- *PathfinderX*. An extension of the Pathfinder task with images scaled up to 256x256, resulting in sequences of 16 384 elements, making this the most challenging task within the benchmark.

The early proposed mixing techniques, namely PSF-Attn, Paramixer, and CDIL-CNN, have consistently outperformed traditional Transformer models and their efficient variants across all tasks in the benchmark. Particularly notable is the CDIL-CNN, a convolutional network approach that excelled in the Pathfinder and Image tasks, demonstrating superior handling of complex patterns. Uniquely, CDIL-CNN was the only model capable of effectively tackling the Retrieval task. PSF-Attn and Paramixer, meanwhile, were especially

Table 4.3: Classification accuracy of Paramixer, CDIL_CNN, and ChordMixer compared with Transformer variants on the LRA tasks. The performance of the competing models was taken from the original LRA paper [7] and Nyströmformer is taken from the corresponding publication [8]. Boldface numbers are winners in each task, while the underlined are runner-ups.

| Model | ListOps $N = 2000$ | Text $N = 4000$ | Image $N = 1024$ | Retrieval $N = 4096$ | Pathfinder $N = 1024$ | PathfinderX $N = 16384$ |
|---|---|---|---|---|---|---|
| Transformer | 36.37 | 64.27 | 42.44 | 57.46 | 71.40 | ✗ |
| Longformer | 35.63 | 62.58 | 42.22 | 56.89 | 69.71 | ✗ |
| Linformer | 37.70 | 53.94 | 38.56 | 52.27 | 76.34 | ✗ |
| Reformer | 37.27 | 56.10 | 38.07 | 53.40 | 68.50 | ✗ |
| Performer | 18.01 | 65.40 | 42.77 | 53.82 | 77.05 | ✗ |
| Nyströmformer | 37.15 | 65.52 | 41.58 | 79.56 | 70.94 | ✗ |
| PSF-Attn | 38.85 | 77.32 | 45.01 | - | 80.49 | ✗ |
| Paramixer | 37.78 | 83.32 | 46.58 | - | 67.13 | ✗ |
| CDIL-CNN | - | 87.61 | 64.49 | 84.27 | 91.00 | ✗ |
| **ChordMixer** | **60.12** | **88.82** | **89.98** | **90.17** | **96.69** | **98.63** |

proficient in the ListOps arithmetic task, showcasing their ability to handle complex hierarchical data structures.

Our most sophisticated model, ChordMixer, outperforms all tested Transformer variants and our earlier methods in every task, establishing itself as the most capable model in this series. It set a new state-of-the-art, especially in the challenging PathfinderX task, surpassing other models by significant margins. This model's ability to accurately classify sparse images in synthetic tasks validated our hypothesis that our approaches are particularly effective when dealing with high-rank and sparse data, while still maintaining superior performance across a variety of sequence types.

To confirm this, we visualize the attention maps of one of our Chord-based mixers on the Pathfinder task and compare them to those from Transformer and its approximation variants.

These visualizations clearly demonstrate that our methods generate more distinct and relevant attention patterns, closely aligning with key elements of the image, such as paths

and endpoints. In contrast, traditional Transformer attention tends to cluster around endpoints, while approximate methods like Linformer and Performer distribute attention more diffusely, often missing critical paths.

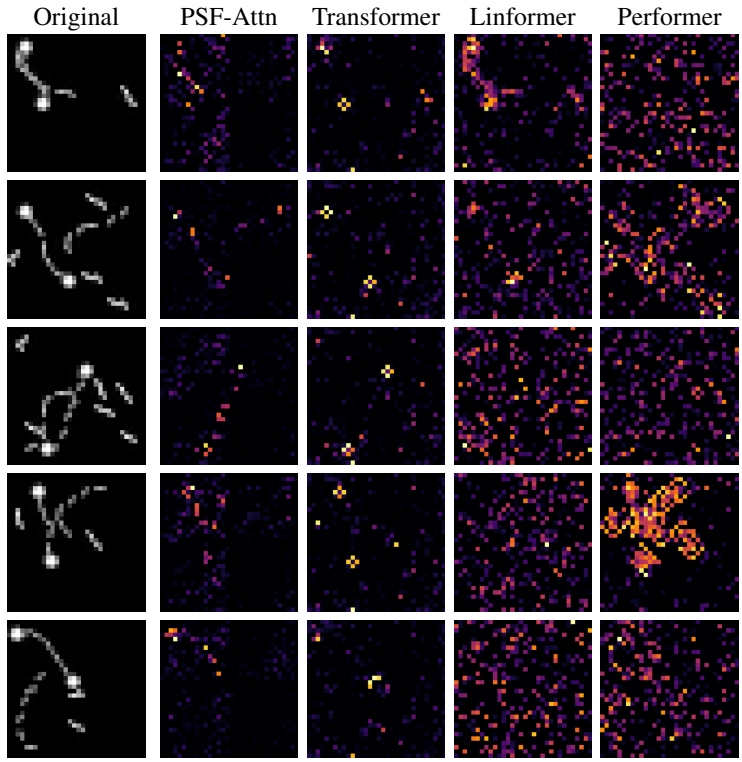



Figure 4.4: Comparison of the attention maps in the Pathfinder task by using PSF-Attn, Transformer, Linformer, and Performer. The first column of images shows the original instances, and the other columns are heat maps of the attention values of the corresponding models. The procedure for visualization is the same for all the methods. (Fig. 3 of Publication A.)

In addition to accuracy comparisons, we analyze the computational efficiency of different models on the Long Range Arena benchmark. Figure 4.5 presents the relationship between accuracy, inference speed, and memory consumption across various attention mechanisms. Linear attention approximations like Linformer and Performer demonstrate

higher inference speeds but at the cost of reduced accuracy. ChordMixer achieves a balanced profile, showing moderate inference speed (approximately 250 instances per second) and memory requirements comparable to linear attention models, while significantly outperforming all methods in accuracy.

Notably, when averaging across all LRA tasks (excluding PathfinderX), ChordMixer achieves more than 20 percentage points improvement in accuracy compared to other approaches. The exclusion of PathfinderX from this analysis is warranted as all other models perform at random chance level on this task, while ChordMixer maintains 98.63% accuracy. These results demonstrate that ChordMixer strongly outperforms previously proposed attention architectures while achieving comparable computational efficiency.

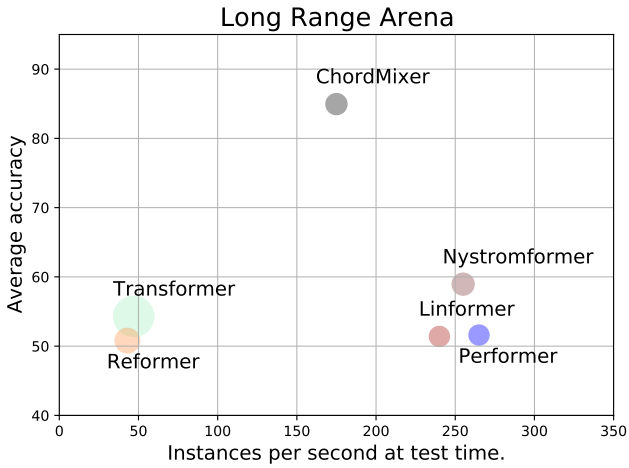



Figure 4.5: Accuracy (y-axis), speed (x-axis), and memory footprint (size of the circles) of different models measured on the LRA benchmark at test time. The reference results are taken from [7] and [8]. Note that PathfinderX was not included in the computation process since only ChordMixer achieved a non-random performance. (Figure 11 of Publication D.)

While the Long Range Arena benchmark is a good attempt to standardize the process of comparison models performance on relatively short sequences, it does not fully capture the complexity of real-world data, which can involve sequences stretching into the thousands or even millions of tokens. To bridge this gap, the subsequent sections will introduce

tasks that feature extremely long and variable-length sequences, showcasing our models' superior ability to handle these more demanding real-life scenarios.

## 4.2 Modeling Long Sequences with Variable Length

In this section, we expand our research to include experiments that demonstrate our model's proficiency in handling long sequential data with highly variable lengths. Our benchmark comprises challenging tasks across three domains: arithmetic operations with long-range dependencies, classification of lengthy text documents, and taxonomy classification based on DNA sequences. Furthermore, we compare our ChordMixer model against a broad spectrum of existing neural attention models, including both conventional architectures like the Transformer and newer innovations such as Reformer, Linformer, Performer, and Nyströmformer, among others. Additionally, we evaluate against models from other neural network families such as State Space Models, Recurrent and Recursive Neural Networks, as well as various convolutional network configurations.

The effectiveness of these models is assessed through their performance on test datasets segmented by sequence length, ensuring that models are evaluated only on sequence lengths they were trained on. This approach helps in understanding each model's ability to capture both short-term and long-term dependencies within the data and to utilize these features effectively for prediction. Multiple models from the existing literature on long sequence modeling were either used directly from open-source implementations or re-implemented where necessary.

### 4.2.1 Long document classification

This task evaluates ChordMixer's capability to manage intricate long-term dependencies within extensive text documents. Leveraging a publicly available dataset of academic papers segmented into four research areas (cs.AI, cs.NE, math.AC, math.GR), this setup mirrors the conditions used to assess Paramixer's performance in earlier experiments 4.1.3. Here, however, we retain the original lengths of the documents rather than truncating them,

presenting a more challenging scenario by utilizing complete texts to test the model's effectiveness over extended sequences.

The dataset comprises 11,956 documents sourced from Github, processed using the arXiv sanity preserver program. Unlike in previous benchmarks where padding was used to standardize input lengths, ChordMixer processes each document in its full length, avoiding potential biases introduced by padding and aligning more closely with realistic usage scenarios.

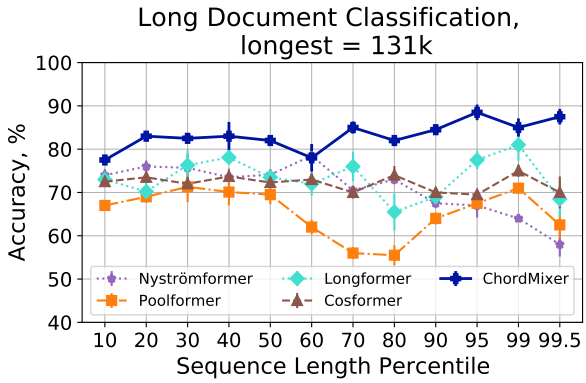



Figure 4.6: Accuracies in the Long Document Classification task at different length percentiles. (Fig. 4 of Publication D.)

From Figure 4.6 it is seen that ChordMixer consistently outperforms other methods, maintaining high accuracy across all document lengths. It is notably robust, achieving or exceeding 80% accuracy across varying text lengths. In contrast, Nyströmformer performs comparably at median lengths but struggles with longer texts. Cosformer shows moderate performance across all lengths, while other models like PoolFormer generally underperform, particularly at higher percentiles. Notably, several advanced models such as Transformer and Linformer encountered memory limitations and were excluded from this part of the analysis due to their inability to process longer documents.

### 4.2.2 Adding problem with variable lengths

Figure 4.7 contains the experimental results on the synthetic adding problem with variable lengths. Interestingly, all models perform well at the lowest complexity level ($\lambda = 200$). However, as $\lambda$ increases, indicating longer sequences, several models begin to struggle significantly. Notably, the Reformer model performs no better than random guessing at $\lambda = 1000$. Other models like Transformer, Nyströmformer, Linformer, and S4 show progressively worsening performance as the sequence length increases.

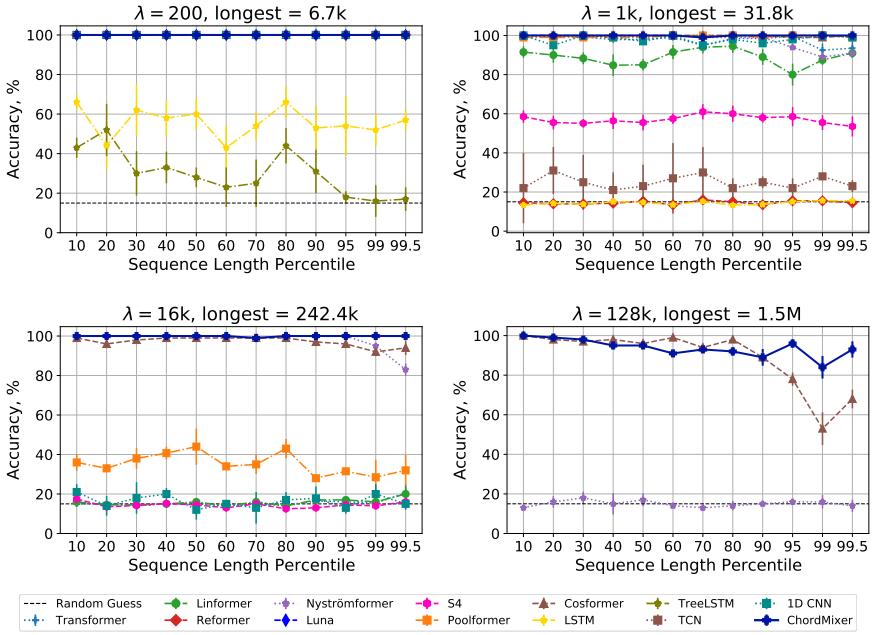



Figure 4.7: Prediction accuracies of the Adding problem with sequences of variable size. $\lambda$ controls the mean length of the sequences. The mean scores and error bars across 3 runs are plotted. For $\lambda = 16k$ and $\lambda = 128k$, We drop some architectures in the plots because they ran out of memory. (Fig. 3 of Publication D.)

A critical observation is that when $\lambda = 16k$, resulting in sequences as long as 242K, models such as Transformer, Reformer, and Luna exhaust available memory resources and fail to complete training. In contrast, ChordMixer, alongside Cosformer and Nyströmformer,

not only remains operational but achieves accuracies above 80%. ChordMixer notably excels by nearly reaching 100% accuracy at this high complexity level.

At the extreme test case of $\lambda = 128k$, only ChordMixer and Cosformer manage to complete the training phase. Here, ChordMixer continues to demonstrate robust performance, maintaining close to 90% accuracy across all tested sequences, while Cosformer's performance drops significantly for the longest sequences.

Table 4.4: Comparison of peak memory (GB) and average running time (milliseconds) per sequence for the Adding problem. For all models, except for Linformer, we applied padding to the maximum sequence length within a batch. We have used a Linux machine with one Tesla-V100 GPU (32GB memory). (Table 4 of Publication D.)

| Model | $\lambda = 200$ (longest=6.7k) | | $\lambda = 128k$ (longest=1.5M) | |
|---|---|---|---|---|
| | Peak Memory | Running Time | Peak Memory | Running Time |
| LSTM | 1.5 | 460 | - | - |
| Tree-LSTM | 25.3 | 9831 | - | - |
| 1D-CNN | 1.4 | 3.3 | - | - |
| TCN | 1.4 | 5.1 | - | - |
| Transformer | 1.9 | 9.8 | - | - |
| Linformer | 1.6 | 12.5 | - | - |
| Reformer | 2.0 | 23.3 | - | - |
| Longformer | 1.7 | 11.1 | - | - |
| Luna | 1.7 | 16.6 | - | - |
| S4 | 1.4 | 3.2 | - | - |
| Poolformer | 1.4 | 3.9 | 13.6 | 378 |
| Cosformer | 1.5 | 8.7 | 22.6 | 994 |
| ChordMixer | 1.5 | 4.9 | 23.1 | 604 |

Away from accuracy comparisons, we analyze the computational efficiency of the tested methods in terms of memory usage and execution time. Table 4.4 presents these metrics for both the smallest ($\lambda = 200$) and largest ($\lambda = 128k$) sequence length configurations of the Adding problem.

For shorter sequences (longest = 6.7k), most methods demonstrate reasonable computational requirements, with peak memory usage ranging from 1.4GB to 2.0GB, except for Tree-LSTM which requires 25.3GB. In terms of running time per sequence, lightweight architectures like 1D-CNN and S4 show the best performance, while ChordMixer maintains

competitive efficiency. Traditional approaches like LSTM and Transformer are significantly slower.

The computational demands become more pronounced for extreme sequence lengths (longest = 1.5M), where only three methods — Poolformer, Cosformer, and ChordMixer — did not result in OOM error during the training process. Among these, Poolformer demonstrates the highest efficiency with 13.6GB peak memory usage and 378ms running time, showing a 41% reduction in memory consumption and 37% faster execution compared to ChordMixer (23.1GB, 604ms). While Cosformer achieves comparable accuracy to ChordMixer, it requires similar memory (22.6GB) but significantly more running time (994ms).

These results should be interpreted in the context of our definition of scalability (Section 3.4.1), which requires both efficient processing and high accuracy for long sequences. While Poolformer and Cosformer demonstrate some computational advantages, ChordMixer is the only method that truly achieves scalability by maintaining high accuracy (around $90\%$) even at extreme sequence lengths. This distinguishing characteristic is particularly important given that computational efficiency alone is insufficient if model performance degrades with sequence length. This analysis suggests that ChordMixer provides a better solution for tasks requiring both high accuracy and the ability to process very long sequences.

### 4.2.3 DNA sequence-based taxonomy classification

The DNA sequence-based taxonomy classification task represents the most formidable challenge within our benchmark. We utilize the Area Under the Receiver Operating Characteristic Curve (ROC-AUC) as our primary evaluation metric, which effectively addresses class imbalance in the dataset. To ensure a balanced comparison across models, we employed cross-entropy loss with class weights derived empirically from the training dataset. Stratified sampling was used to maintain consistent class distribution across training, validation, and test splits. Unlike other models that required zero padding to the maximum sequence length, ChordMixer processed sequences in their original lengths. For models

such as Transformer, Reformer, and Linformer, sequences were truncated to manage memory constraints effectively.

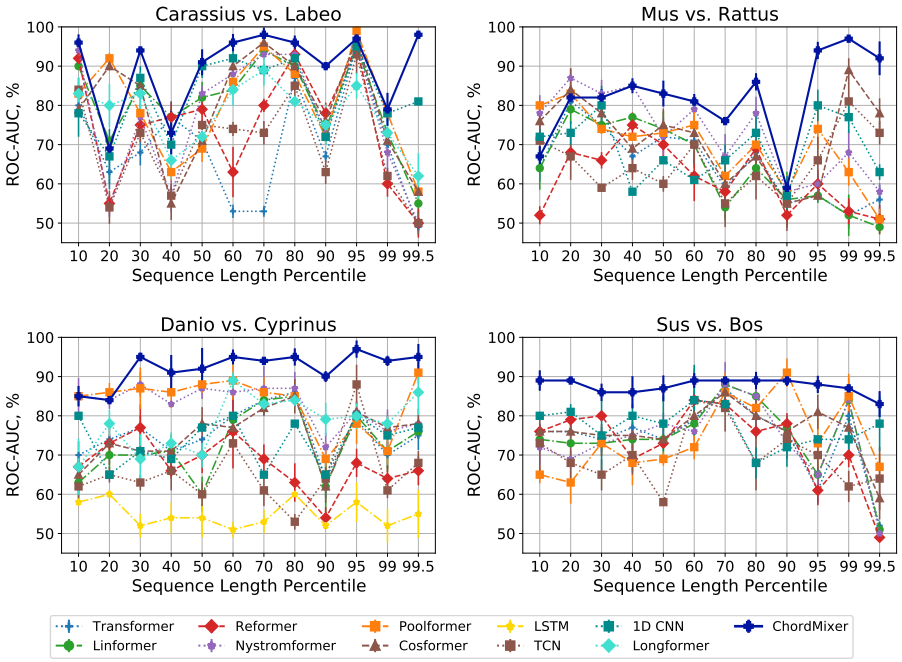



Figure 4.8: Test ROC-AUCs on the DNA-based taxonomy classification tasks. The score is the average of three runs with different random seeds. Plots are better read in colors. (Fig. 5 of Publication D.)

As depicted in Figure 4.8, ChordMixer consistently outperforms or matches the best-performing models across nearly all sequence length percentiles in all datasets. Notably, ChordMixer exhibits superior performance particularly at longer sequence percentiles, demonstrating its robustness in handling extensive sequence lengths without the need for padding. In the Carassius vs. Labeo dataset, while other models perform comparably at shorter sequence lengths, they significantly falter at the 99th percentile and beyond, where sequences exceed 5,630 elements.

For the most extended sequences in the Sus vs. Bos dataset, at the 99.5 percentile,

ChordMixer achieves a ROC-AUC of 89%, starkly outperforming other models that fall below 70%. This pattern is consistent, with ChordMixer maintaining top performance at a majority of the length percentiles, particularly in the Danio vs. Cyprinus and Sus vs. Bos datasets, where it leads in nine out of ten evaluated percentiles. Remarkably, ChordMixer sustains high ROC-AUC scores ranging from 84% to 89% across various lengths for the Sus vs. Bos dataset, confirming its efficacy and stability in managing DNA sequences of variable and extended lengths.

**Attention Visualization**

To gain insights into how ChordMixer processes and prioritizes information across varying sequence lengths, we visualize the output matrices from ChordMixer blocks. This visualization helps in understanding the attention mechanism's adaptability and effectiveness across different sequence contexts.

For this purpose, we trained the ChordMixer model on the Adding problem with variable sequence lengths controlled by $\lambda = 200$, where sequences range from a minimum of 32 to a maximum of 6655 elements. We selected three sequences of lengths 40, 240, and 1006 to showcase the model's attention patterns.

These visualizations (Figure 4.9) provide a clear depiction of how ChordMixer adjusts its focus across the sequence. Initially, in the output from the first block ($\text{output}_1$), the model's attention is primarily guided by the inherent scales of rotation, evidenced by distinct high values at certain intervals. As the sequence progresses through the model, by the last output block ($\text{output}_{\text{last}}$), the attention distribution becomes broader and more uniform across different segments of the sequence.

This broadening in attention suggests a transition from focusing on specific key elements to a more holistic processing approach, enabling ChordMixer to integrate and synthesize information across the entire sequence length effectively. Particularly, the last output block shows a notable shift towards multi-scale attention—local attention remains strong in upper tracks (minimal rotation), while more globally distributed attention features in the lower tracks (substantial rotation), demonstrating ChordMixer's capability to adapt its focus dynamically based on the sequence's demands.

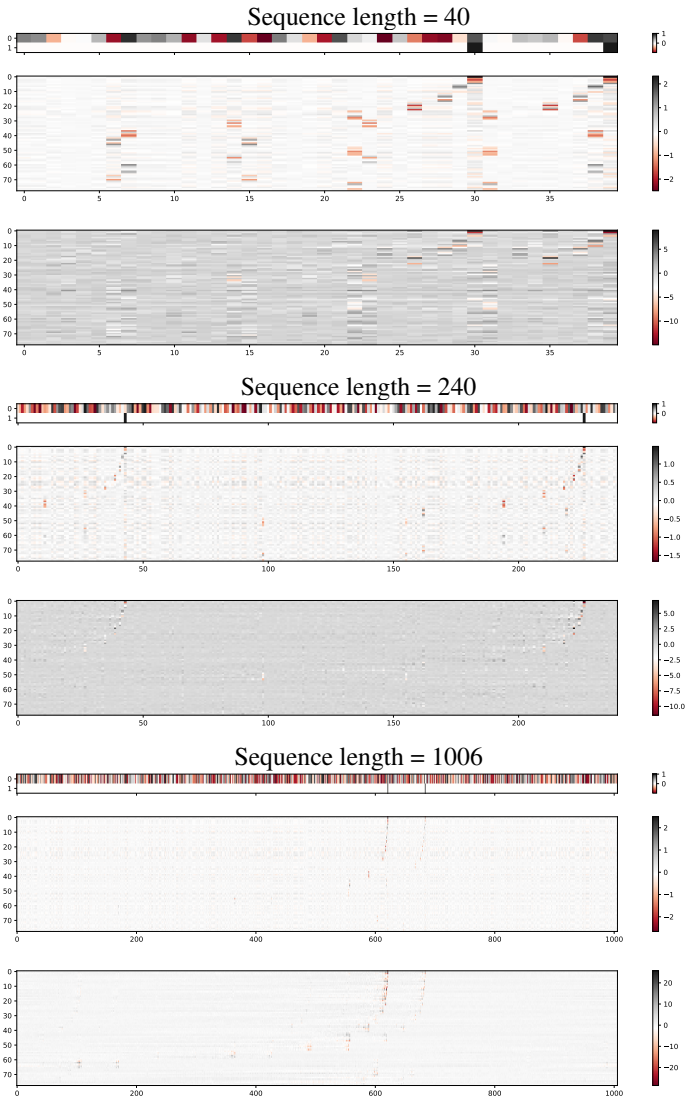



Figure 4.9: Visualizations of attention patterns in the Adding problem at three different sequence lengths using the ChordMixer model. For each example, the first image shows the initial sequence data, the second the output after the first ChordMixer block, and the third after the final block. (Fig. 8 of Publication D.)

# Chapter 5

# Conclusion and Discussion

This chapter summarizes the research undertaken during my PhD study, outlining how we addressed the research goals through the research questions posed. It emphasizes the contributions of three key research publications (Publication A, B, and D) and clarifies the relevance of other publications to the research questions outlined in Section 1. Additionally, this chapter contextualizes our research within recent developments in commercial Large Language Models (LLMs), which have demonstrated impressive capabilities in processing long sequences. We discuss how our architectural innovations complement and differ from these commercial advances, followed by an analysis of current limitations and promising directions for future work.

## 5.1 Summary of Research Contributions

In recent years, the importance of sequential data has grown across various fields, prompting a focus on the automatic extraction of meaningful features from this data. Motivated by these developments, we have designed scalable neural networks capable of effective representation learning from long sequential data. ChordMixer, our flagship network architecture, exemplifies the desired properties of robust neural attention as detailed in Section 3.4.1. These include a comprehensive receptive field, scalability, symmetry, and flexibility in handling various sequence lengths. We have conducted thorough theoretical analyses

and experimental validations of these models across several benchmarks, both existing and newly introduced, involving exceptionally long sequences.

The answers to the two primary research questions are as follows:

**Research Question 1:** How can we design effective and efficient attention neural networks to perform representation learning for long sequential data?

Neural attention, particularly in the form of Transformers, has recently attracted significant interest due to its impressive sequence modeling capabilities. Long sequences are particularly challenging for these methods due to scalability issues related to longer sequences. To address this, we first rethought the neural attention approach by modifying the softmax normalization, which is typically responsible for quadratic computational costs and memory usage. By reformulating this to a matrix approximation approach, we were able to bypass these constraints.

Our initial solution, Sparse Factorization, parameterizes the entries of sparse matrices through neural networks. This concept was implemented in the PSF-Attn (Publication A) and Paramixer (Publication B) networks, employing the Chord protocol to define the sparse structure of the factorizing matrices. Our methods demonstrated superior performance compared to Transformers and their efficient variants, particularly on the Long Range Arena benchmark and in scalability tests involving NLP and DNA sequences up to 32,000 elements in length.

The most robust and efficient model, ChordMixer (Publication D), provides a powerful backbone for handling extremely long sequences. Like earlier models, ChordMixer incorporates the Chord protocol through circular shifts along the time dimension of the input tensor. This innovative neural attention network achieves comprehensive coverage with logarithmic scaling, resulting in sub-quadratic time and memory complexity. It delivered outstanding results on the Long Range Arena, particularly on the PathfinderX task with sequences as long as 16,000 elements, and demonstrated its ability to learn from sequences up to 1.5 million elements in length. Its exceptional performance underscores its scalability and versatility across various domains and complex settings, where sequence lengths vary significantly.

**Research Question 2:** How can we implement efficient representation learning for sequences with variable lengths?

Sequences in fields like NLP, time series analysis, and genomics naturally vary in length. Neural attention's capacity to model both short-term and long-term dependencies is critical for these applications. Existing benchmarks typically focus on shorter sequences, up to 16,000 elements. We addressed this gap by introducing three new datasets involving both synthetic and real-world sequences that are not only exceptionally long but also vary significantly in length.

The first dataset, synthetic in nature, challenges scalability with sequences up to 1.5 million tokens, aiming to test the limits of current methodologies. The second dataset comprises byte-level document classification tasks using real scientific papers, with the longest document containing 131,000 tokens. The third dataset includes DNA sequences ranging from 32 to 500,000 elements. To our knowledge, no prior studies have developed such complex benchmarks incorporating both extreme length and high variability.

Standard batching techniques for handling variable-length sequences typically involve padding or chunking, which can introduce computational inefficiencies or lose long-term dependencies. Our proposed neural attention training method allows for dynamic adjustment based on sequence length, thereby avoiding these issues. By integrating the Chord protocol into the ChordMixer's mixing engine, we ensure efficient batch processing and eliminate the need for padding or chunking.

ChordMixer's performance has been benchmarked against various existing sequence modeling backbones, including CNNs, RNNs, and Transformers, along with their efficient variants. Our approach demonstrated superior performance across all tasks, particularly with very long sequences, affirming its exceptional capability in managing the most challenging sequence lengths.

## 5.2 Relationship to Long-Context Commercial LLMs

Recent developments in commercial language models have demonstrated remarkable capabilities in processing long sequences, with models like Gemini-1.5-Pro supporting context lengths of up to 2 million tokens[1], Claude-3.5-Sonnet handling 200,000 tokens[2], and GPT-4 processing up to 128,000 tokens[3]. While these achievements are impressive, it is important to contextualize them within the broader landscape of sequence modeling that this dissertation addresses.

The methods developed in this dissertation, particularly ChordMixer, share the foundational concept of self-attention with commercial LLMs but differ in several crucial aspects:

First, our approach provides a general-purpose architecture for sequence modeling that extends beyond natural language processing. While commercial LLMs focus primarily on text, our methods demonstrate versatility across diverse domains including DNA sequences, time series, and mathematical operations, as evidenced by our comprehensive benchmarks.

Second, our research prioritizes architectural innovation over hardware optimization. The recent release of LLama3.1 [116] revealed that training state-of-the-art language models requires thousands of NVIDIA H100 GPUs, with associated costs beyond the reach of most academic institutions. In contrast, ChordMixer achieves comparable sequence length capabilities (up to 1.5 million tokens) through architectural innovation alone. Our approach maintains sub-quadratic computational complexity and operates effectively under standard hardware constraints, making it accessible for both research and practical applications. While hardware-aware implementations remain an important direction for future research (see Section 5.3), our focus on architectural efficiency provides immediate value.

Third, commercial language models like OpenAI's ChatGPT, Anthropic's Claude, and Google's Gemini operate as closed systems, making it difficult to understand or adapt their methods for research purposes. Our work, conversely, provides a transparent, theoretically

[1] https://cloud.google.com/vertex-ai/generative-ai/docs/long-context
[2] https://www.anthropic.com/news/claude-3-5-sonnet
[3] https://platform.openai.com/docs/models

grounded approach to long sequence modeling with clear mathematical foundations and publicly available implementation.

The insights from our research could benefit future iterations of open-source language models. Our efficient attention mechanisms and rotation-based mixing strategies could be integrated into large language models to improve their efficiency in processing long sequences, potentially reducing the computational resources required for training and inference.

This context demonstrates that while commercial language models have achieved impressive capabilities in processing long sequences, the contributions of this dissertation remain highly relevant for advancing our understanding of efficient sequence modeling and providing practical solutions across a broader range of applications.

## 5.3 Limitations and Future Work

The primary goal of this research has been achieved—we have developed a scalable neural attention architecture suitable for extremely long sequences. We have made the datasets, experimental results, and model implementations publicly available, aiming to advance the current toolkit for sequence modeling and assist other researchers in tackling challenges associated with very long sequences.

However, this represents just the beginning of what is possible, and there is still much to explore in this field.

**Self-Supervised Learning:** To date, our experiments have primarily focused on supervised learning scenarios. Recent advances in language and vision have been driven by self-supervised pre-training, which has yielded remarkable results, such as those seen with ChatGPT [117], Gemini [118], MAE [119], and DINO [120]. Increasing the context size in Large Language Models has become a prominent topic within NLP research, closely aligning with our research goals [121, 122]. In future work, we plan to extend our methodology to massive pre-training efforts on language and DNA data using our architectures.

We hypothesize that pre-training on variable-length sequences might improve generalization across both short and long sequences, potentially leading to stronger models in both language and vision domains. Given the exceptional performance of ChordMixer across multiple domains, we also plan to explore contrastive learning methods to introduce multimodal capabilities into our model [123]

**Autoregressive Decoding:** One of the key strengths of the Transformer model is its robust decoder, which supports autoregressive pre-training and inference. Our research has primarily focused on the encoding process, with little exploration of the generative capabilities of our architectures. Our circular mixing technique currently does not restrict access to future tokens during sequence processing, which could be addressed by designing a decoder that supports masking and allows for fast, cache-efficient inference, potentially surpassing the efficiency of the Transformer's key-value cache. Our model could enable the generation of multiple words in a single inference step, enhancing the coherence of generated outputs over long sequences

**Memory and Speed Optimization:** In our initial implementation of ChordMixer, we used standard PyTorch functions for sequence manipulation, which may not be optimized for accessing distant tokens due to GPU memory architecture constraints. Custom CUDA kernels could be developed to accelerate the mixing module. Additionally, employing the Reversible Residual Network [124] technique could reduce memory requirements significantly, from $O(dN \log_2 N)$ to $O(dN)$. Optimizing GPU utilization during training by adjusting batch sizes based on token count rather than sequence count could also enhance efficiency.

Due to the unique processing requirements of variable-size sequences, standard normalization techniques such as BatchNorm [125] and LayerNorm [126] have not enhanced stability or performance as effectively as seen with Transformers and convolutional methods. Developing new stabilization techniques could help control gradient flow and further enhance the performance of our attention-based mixers.

# Appendix: code and data

**Publication A**

Sparse factorization of square matrices with application to neural attention modeling

`https://github.com/RuslanKhalitov/SparseFactorization`

**Publication B**

Paramixer: Parameterizing mixing links in sparse factors works better than dot-product self-attention

`https://github.com/wiedersehne/Paramixer`

**Publication C**

Classification of long sequential data using circular dilated convolutional neural networks

`https://github.com/LeiCheng-no/CDIL-CNN`

**Publication D**

ChordMixer: A scalable neural attention model for sequences with different lengths

`https://github.com/RuslanKhalitov/ChordMixer`

**Publication E**

Self-supervised learning for DNA sequences with circular dilated convolutional networks

`https://github.com/LeiCheng-no/cdilDNA`

`https://github.com/wiedersehne/cdilDNA`

# Bibliography


[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[3] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*, 2020.

[4] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[5] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[6] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

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

[8] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nystr\" omformer: A nystr\" om-based algorithm for approximating self-attention. *arXiv preprint arXiv:2102.03902*, 2021.

[9] Ali Bou Nassif, Ismail Shahin, Imtinan Attili, Mohammad Azzeh, and Khaled Shaalan. Speech recognition using deep neural networks: A systematic review. *IEEE access*, 7:19143–19165, 2019.

[10] Ming Zhou, Nan Duan, Shujie Liu, and Heung-Yeung Shum. Progress in neural nlp: modeling, learning, and reasoning. *Engineering*, 6(3):275–290, 2020.

[11] Yu Hen Hu and Jenq-Neng Hwang. *Handbook of neural network signal processing*. CRC press, 2018.

[12] Tianwei Yue, Yuanxin Wang, Longxiang Zhang, Chunming Gu, Haoru Xue, Wenping Wang, Qi Lyu, and Yujie Dun. Deep learning for genomics: A concise overview. *arXiv preprint arXiv:1802.00810*, 2018.

[13] Ron Bracewell and Peter B Kahn. The fourier transform and its applications. *American Journal of Physics*, 34(8):712–712, 1966.

[14] Warren John Ewens, Gregory R Grant, et al. *Statistical methods in bioinformatics: an introduction*, volume 15. Springer, 2005.

[15] Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

[16] Pierre Brémaud. *Mathematical principles of signal processing: Fourier and wavelet analysis*. Springer, 2002.

[17] Kuo-Chen Chou. Prediction of protein cellular attributes using pseudo-amino acid composition. *Proteins: Structure, Function, and Bioinformatics*, 43(3):246–255, 2001.

[18] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1:81–106, 1986.

[19] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.

[20] Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7:21, 2013.

[21] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

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

[23] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[24] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo-Yiin Chang, and Tara Sainath. Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing*, 13(2):206–219, 2019.

[25] Žiga Avsec, Vikram Agarwal, Daniel Visentin, Joseph R Ledsam, Agnieszka Grabska-Barwinska, Kyle R Taylor, Yannis Assael, John Jumper, Pushmeet Kohli, and David R Kelley. Effective gene expression prediction from sequence by integrating long-range interactions. *Nature methods*, 18(10):1196–1203, 2021.

[26] Wei Chen, Hao Lin, and Kuo-Chen Chou. Pseudo nucleotide composition or pseknc: an effective formulation for analyzing genomic sequences. *Molecular BioSystems*, 11(10):2620–2634, 2015.

[27] Ruslan Khalitov, Tong Yu, Lei Cheng, and Zhirong Yang. Sparse factorization of square matrices with application to neural attention modeling. *Neural Networks*, 152:160–168, 2022.

[28] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM computer communication review*, 31(4):149–160, 2001.

[29] Tong Yu, Ruslan Khalitov, Lei Cheng, and Zhirong Yang. Paramixer: Parameterizing mixing links in sparse factors works better than dot-product self-attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 691–700, 2022.

[30] Lei Cheng, Ruslan Khalitov, Tong Yu, Jing Zhang, and Zhirong Yang. Classification of long sequential data using circular dilated convolutional neural networks. *Neurocomputing*, 518:50–59, 2023.

[31] Ruslan Khalitov, Tong Yu, Lei Cheng, and Zhirong Yang. Chordmixer: A scalable neural attention model for sequences with different length. In *The Eleventh International Conference on Learning Representations*, 2023.

[32] Lei Cheng, Tong Yu, Ruslan Khalitov, and Zhirong Yang. Self-supervised learning for dna sequences with circular dilated convolutional networks. *Neural Networks*, 171:466–473, 2024.

[33] Wilson L Taylor. "cloze procedure": A new tool for measuring readability. *Journalism quarterly*, 30(4):415–433, 1953.

[34] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[35] Alice Zheng and Amanda Casari. *Feature engineering for machine learning: principles and techniques for data scientists*. " O'Reilly Media, Inc.", 2018.

[36] Samina Khalid, Tehmina Khalil, and Shamila Nasreen. A survey of feature selection and feature extraction techniques in machine learning. In *2014 science and information conference*, pages 372–378. IEEE, 2014.

[37] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[38] Marília Barandas, Duarte Folgado, Letícia Fernandes, Sara Santos, Mariana Abreu, Patrícia Bota, Hui Liu, Tanja Schultz, and Hugo Gamboa. Tsfel: Time series feature extraction library. *SoftwareX*, 11:100456, 2020.

[39] Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. *Nature biotechnology*, 33(8):831–838, 2015.

[40] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[41] Beth Logan et al. Mel frequency cepstral coefficients for music modeling. In *Ismir*, volume 270, page 11. Plymouth, MA, 2000.

[42] Lalit R Bahl, Frederick Jelinek, and Robert L Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE transactions on pattern analysis and machine intelligence*, 2:179–190, 1983.

[43] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[44] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[45] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[46] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[47] Xian-Da Zhang. *Modern signal processing*. Walter de Gruyter GmbH & Co KG, 2022.

[48] Benny Chor, David Horn, Nick Goldman, Yaron Levy, and Tim Massingham. Genomic dna k-mer spectra: models and modalities. *Genome biology*, 10:1–10, 2009.

[49] Benjamin Buchfink, Chao Xie, and Daniel H Huson. Fast and sensitive protein alignment using diamond. *Nature methods*, 12(1):59–60, 2015.

[50] Michael Gribskov, Andrew D McLachlan, and David Eisenberg. Profile analysis: detection of distantly related proteins. *Proceedings of the National Academy of Sciences*, 84(13):4355–4358, 1987.

[51] Kuo-Chen Chou. Pseudo amino acid composition and its applications in bioinformatics, proteomics and system biology. *Current Proteomics*, 6(4):262–274, 2009.

[52] Predrag Radivojac, Wyatt T Clark, Tal Ronnen Oron, Alexandra M Schnoes, Tobias Wittkop, Artem Sokolov, Kiley Graim, Christopher Funk, Karin Verspoor, Asa Ben-Hur, et al. A large-scale evaluation of computational protein function prediction. *Nature methods*, 10(3):221–227, 2013.

[53] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[54] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[55] Andreas Mayr, Harald Binder, Olaf Gefeller, and Matthias Schmid. The evolution of boosting algorithms. *Methods of information in medicine*, 53(06):419–427, 2014.

[56] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Number 4 in 4. Springer, 2006.

[57] Dan Jurafsky. *Speech & language processing*. Pearson Education India, 2000.

[58] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[59] Ronald A Rensink. The dynamic representation of scenes. *Visual cognition*, 7(1-3):17–42, 2000.

[60] Hugo Larochelle and Geoffrey E Hinton. Learning to combine foveal glimpses with a third-order boltzmann machine. *Advances in neural information processing systems*, 23, 2010.

[61] Misha Denil, Loris Bazzani, Hugo Larochelle, and Nando de Freitas. Learning where to attend with deep architectures for image tracking. *Neural computation*, 24(8):2151–2184, 2012.

[62] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. *Advances in neural information processing systems*, 27, 2014.

[63] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015.

[64] Alana de Santana Correia and Esther Luna Colombini. Attention, please! a survey of neural attention models in deep learning. *Artificial Intelligence Review*, 55(8):6037–6124, 2022.

[65] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[66] Derya Soydaner. Attention mechanism in neural networks: where it comes and where it goes. *Neural Computing and Applications*, 34(16):13371–13385, 2022.

[67] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. *Advances in neural information processing systems*, 28, 2015.

[68] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

[69] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.

[70] Hyeonseob Nam, Jung-Woo Ha, and Jeonghee Kim. Dual attention networks for multimodal reasoning and matching. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 299–307, 2017.

[71] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[72] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

[73] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[74] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[75] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[76] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[77] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[78] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell,

et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[79] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[80] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[81] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[82] Angela Fan, Shruti Bhosale, Holger Schwenk, Zhiyi Ma, Ahmed El-Kishky, Siddharth Goyal, Mandeep Baines, Onur Celebi, Guillaume Wenzek, Vishrav Chaudhary, et al. Beyond english-centric multilingual machine translation. *Journal of Machine Learning Research*, 22(107):1–48, 2021.

[83] Jiezhong Qiu, Hao Ma, Omer Levy, Scott Wen-tau Yih, Sinong Wang, and Jie Tang. Blockwise self-attention for long document understanding. *arXiv preprint arXiv:1911.02972*, 2019.

[84] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International conference on machine learning*, pages 4055–4064. PMLR, 2018.

[85] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[86] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.

[87] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[88] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[89] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pages 3744–3753. PMLR, 2019.

[90] Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. Etc: Encoding long and structured inputs in transformers. *arXiv preprint arXiv:2004.08483*, 2020.

[91] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[92] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

[93] Zhen Qin, Weixuan Sun, Hui Deng, Dongxu Li, Yunshen Wei, Baohong Lv, Junjie Yan, Lingpeng Kong, and Yiran Zhong. cosformer: Rethinking softmax in attention. *arXiv preprint arXiv:2202.08791*, 2022.

[94] James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. Fnet: Mixing tokens with fourier transforms. *arXiv preprint arXiv:2105.03824*, 2021.

[95] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34:24261–24272, 2021.

[96] Asher Trockman and J Zico Kolter. Patches are all you need? *arXiv preprint arXiv:2201.09792*, 2022.

[97] Weihao Yu, Mi Luo, Pan Zhou, Chenyang Si, Yichen Zhou, Xinchao Wang, Jiashi Feng, and Shuicheng Yan. Metaformer is actually what you need for vision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10819–10829, 2022.

[98] Ruiyang Liu, Yinghui Li, Linmi Tao, Dun Liang, and Hai-Tao Zheng. Are we ready for a new paradigm shift? a survey on visual deep mlp. *Patterns*, 3(7), 2022.

[99] Christopher Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, volume 13, 2001.

[100] Petros Drineas and Michael W. Mahoney. On the Nyström method for approximating a gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6:2153–2175, 2005.

[101] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[102] Colin Lea, René Vidal, Austin Reiter, and Gregory D. Hager. Temporal convolutional networks: A unified approach to action segmentation. In *ECCV 2016 Workshops*, pages 47–54, 2016.

[103] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[104] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[105] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[106] Jun He, Liqun Wang, Liu Liu, Jiao Feng, and Hao Wu. Long document classification from local word glimpses via recurrent attention learning. *IEEE Access*, 7:40707–40718, 2019.

[107] Ruslan Khalitov, Tong Yu, Lei Cheng, and Zhirong Yang. Sparse factorization of square matrices with application to neural attention modeling. *Neural Networks*, 152:160–168, 2022.

[108] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity, 2020.

[109] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers, 2020.

[110] Lianhe Zhao, Jiajia Wang, Yanyan Li, Tingrui Song, Yang Wu, Shuangsang Fang, Dechao Bu, Hui Li, Liang Sun, Dong Pei, et al. Noncodev6: an updated database dedicated to long non-coding rna annotation in both animals and plants. *Nucleic Acids Research*, 49(D1):D165–D171, 2021.

[111] Kevin L Howe, Premanand Achuthan, James Allen, Jamie Allen, Jorge Alvarez-Jarreta, M Ridwan Amode, Irina M Armean, Andrey G Azov, Ruth Bennett, Jyothish Bhai, et al. Ensembl 2021. *Nucleic acids research*, 49(D1):D884–D891, 2021.

[112] Ensembl Release 104 (May 2021). `http://www.ensembl.org/index.html`. Accessed: 2021-5.

[113] Nikita Nangia and Samuel R Bowman. Listops: A diagnostic dataset for latent tree learning. *arXiv preprint arXiv:1804.06028*, 2018.

[114] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.

[115] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[116] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[117] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[118] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[119] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009, 2022.

[120] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[121] Tian Wang and Kyunghyun Cho. Larger-context language modelling. *arXiv preprint arXiv:1511.03729*, 2015.

[122] Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. Longlora: Efficient fine-tuning of long-context large language models. *arXiv preprint arXiv:2309.12307*, 2023.

[123] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[124] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

[125] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[126] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

# Part II
# Publications

# Publication A - Sparse factorization of square matrices with application to neural attention modeling

Neural Networks 152 (2022) 160–168

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

# Sparse factorization of square matrices with application to neural attention modeling

Ruslan Khalitov[1], Tong Yu[1], Lei Cheng, Zhirong Yang[*]

*Norwegian University of Science and Technology, Norway*

ARTICLE INFO

*Article history:*
Received 26 October 2021
Received in revised form 17 March 2022
Accepted 14 April 2022
Available online 22 April 2022

*Keywords:*
Matrix factorization
Sparse
Neural networks
Attention modeling

ABSTRACT

Square matrices appear in many machine learning problems and models. Optimization over a large square matrix is expensive in memory and in time. Therefore an economic approximation is needed. Conventional approximation approaches factorize the square matrix into a number matrices of much lower ranks. However, the low-rank constraint is a performance bottleneck if the approximated matrix is intrinsically high-rank or close to full rank. In this paper, we propose to approximate a large square matrix with a product of sparse full-rank matrices. In the approximation, our method needs only $N(\log N)^2$ non-zero numbers for an $N \times N$ full matrix. Our new method is especially useful for scalable neural attention modeling. Different from the conventional scaled dot-product attention methods, we train neural networks to map input data to the non-zero entries of the factorizing matrices. The sparse factorization method is tested for various square matrices, and the experimental results demonstrate that our method gives a better approximation when the approximated matrix is sparse and high-rank. As an attention module, our new method defeats Transformer and its several variants for long sequences in synthetic data sets and in the Long Range Arena benchmarks. Our code is publicly available[2].



## 1. Introduction

Many machine learning models include one or more square matrices, such as the kernel matrix in Support Vector Machines (Cortes & Vapnik, 1995) or Gaussian Process, the affinity matrix in graph or network representation, and the attention matrix in Transformers (Vaswani et al., 2017b). In many machine learning problems, the solution space is also over square matrices, for example, graph matching and network architecture optimization.

Obtaining a full $N \times N$ matrix can cause infeasible storage and computational cost if $N$ is large. For example, a double-precision kernel matrix of the typical `MNIST` handwritten digit data set ($N = 70{,}000$) requires about 36.5G memory. In another example, we need to calculate eight quintillion float numbers to fill a full attention matrix of a human DNA sequence with about 3.2 billion base pairs.

Therefore we need economical surrogates to approximate the full square matrices at a large scale. Matrix factorization is a widely used approach that approximately factorizes the square matrix to some cheaper matrices. In conventional matrix factorization, the factorizing matrices must be low-rank, for example, in Truncated Singular Value Decomposition (TSVD) and Nyström approximation (Drineas & Mahoney, 2005; Williams & Seeger, 2001). However, these conventional approaches do not work well if the square matrix in the approximation is not low-rank.

This paper proposes a novel approximation method called Sparse Factorization (SF), where the approximated square matrix is factorized into some full-rank sparse matrices. Using the Chord protocol (Stoica, Morris, Karger, Kaashoek, & Balakrishnan, 2001) to specify the non-zero entry positions,[3] we can match an $N \times N$ full matrix with $\log N$ factorizing matrices, where each contains $N \log N$ non-zero entries and thus in total $N(\log N)^2$ non-zeros. Therefore the approximation is economical when $N$ is large.

We tested the new approximation method on a variety of synthetic and real-world square matrices, with comparison to the most accurate low-rank approximation method, TSVD. We find that given the number of non-zero entries, SF wins over TSVD for approximating sparse square matrices with unknown non-zero positions.

We exploit the above advantage of SF to design a novel neural attention model. We parameterize the mapping from input data to the non-zero entries in each factorizing matrix with neural networks. The parametric SF thus replaces the conventional scaled

* Corresponding author.
*E-mail address:* zhirong.yang@ntnu.no (Z. Yang).

[1] Equal contribution.

[2] https://github.com/RuslanKhalitov/SparseFactorization.

[3] Non-zero entries are those stored, including both non-zeros and explicit zeros.

https://doi.org/10.1016/j.neunet.2022.04.014

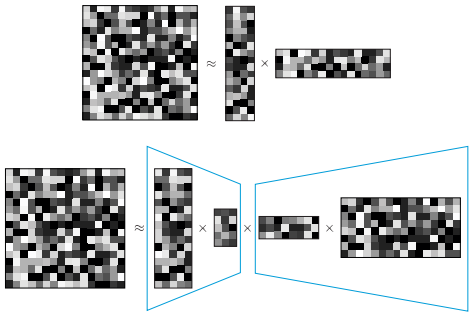


**Fig. 1.** Examples of low-rank matrix dense matrix factorization. Each small square represents a matrix entry. Top is a two-factor factorization and the bottom is a four-factor factorization. The trapezoids illustrate the "bottleneck", i.e. grouping the factorizing matrices according the place with the smallest rank.

dot-product attention. We find that only one such block of SF attention already defeats Transformer and its several state-of-the-art variants on very long synthetic sequence classification and on the public Long Range Arena benchmark tasks.

The remaining of the paper is organized as follows. We first briefly review the matrix approximation based on low-rank factorization in Section 2. In Section 3, we present the machine learning formulation of sparse factorization of square matrices. The parametric formulation using neural networks is proposed in Section 4. In Section 5, we present the experimental settings and results. Conclusion and future work are given in Section 6.

## 2. Low-rank matrix factorization

The conventional matrix approximation is to factorize the large square matrix $X$ into a few low-rank matrices, for example $X \approx \widehat{X} = WH$ or $X \approx WSH$ with $W \in \mathbb{R}^{N\times r}$, $S \in \mathbb{R}^{r\times r}$, $H \in \mathbb{R}^{r\times N}$ and $r \ll N$. There are different tri-factor kind decomposition, for example, Nyström decomposition (Williams & Seeger, 2001), where $W$ and $H$ are calculated using normal kernel function, and $S$ is learned, and CUR decomposition (Mahoney & Drineas, 2009), where $W$ contains $r$ columns of $X$, $H$ contains $r$ rows of $X$, and $S$ is learned.

The approximation error can be measured by a certain divergence, for example the squared Frobenius norm: $D(X \parallel \widehat{X}) = \|X-\widehat{X}\|_F^2 = \sum_{ij}\left(X_{ij}-\widehat{X}_{ij}\right)^2$. According to Eckart and Young (1936), minimization of $\|X - WH\|_F^2$ over $W$ and $H$ has a closed-form solution using Truncated Singular Value Decomposition (TSVD). Denote $\Lambda$ the diagonal matrix with the largest $r$ singular values. The corresponding left and right singular vectors as columns form matrices $U$ and $V$, respectively. Then the minimum appears by setting $W = U$ and $H = \Lambda V^T$, where we can move any scaling from $W$ to $H$ or vice versa.

Matrix factorization with more than two factors cannot achieve a lower approximation error than TSVD. In general, we write $\widehat{X} = \prod_{m=1}^{M} W^{(m)}$, where $W^{(m)} \in \mathbb{R}^{r_m, r_{m+1}}$ with $r_1 = r_{M+1} = N$. We can always reduce a multi-factor case to the two-factor form by grouping $L = \prod_{m=1}^{m'-1} W^{(m)}$ and $R = \prod_{m=m'}^{M} W^{(m)}$, where $m' = \arg\min_m \left(\{r_m\}_{m=2}^{M}\right)$ and $X \approx LR$. It is required that $r_{m'} \ll N$ when approximating large square matrices. Otherwise, the factorizing matrices are still too large. The grouping at the bottleneck is illustrated in Fig. 1 (bottom).

There are low-rank matrix factorization methods with certain constraints on the factorizing matrices, for example, nonnegative matrix factorization (Lee & Seung, 1999) and binary matrix factorization (Slawski, Hein, & Lutsik, 2013; Zhang, Li, Ding, & Zhang, 2007). However, the constraints limit the solution space and thus usually lead to a higher approximation error. Some other methods employ some penalty terms on the factorizing matrices, for instance, the Sparse Coding (Donoho, 2006) or Funk matrix factorization (Funk, 2006). Despite their particular applications, these methods generally give a higher approximation error than TSVD (in terms of Frobenius norm) due to compromise to the extra penalties.

## 3. Sparse factorization

As we have seen, the performance of LRMF is capped due to the low-rank constraint. Its performance is poor if the approximated square matrix is far from low-rank, i.e., the sum of the few largest eigenvalues does not dominate the matrix trace.

Here we propose a new approximation method that is free of the low-rank constraint. Our method implements the approximation with a number ($M$) of *square* and *sparse* factorizing matrices. The approximation is still economical if the total number of non-zero entries is much smaller than $N^2$.

There are many ways to specify the sparse structure (i.e. the non-zero positions). A good specification should guarantee that the product of the factorizing matrices, $\widehat{X}$, should be a full matrix. The condition prevents that the approximating matrix contains some always-zero entries. In addition, we consider a secondary requirement that each factorizing matrix has the same sparse structure, which provides better symmetry in the approximation.

We thus adopt a modified Chord protocol (Stoica et al., 2001) which originates for peer-to-peer distributed hash tables. In the protocol, the indices from 1 to $N$ are organized in a circular graph, where the $i$th node connects to itself and the $((i+2^k) \mod N)$th nodes with $k = 0, \ldots, K-2$. We set $K = \log_2 N$ in our work. The Chord protocol is illustrated in Fig. 2 (top).

We use the Chord protocol to specify the non-zero positions in each factorizing matrix, where each matrix row has $\log_2 N$ non-zero entries. That is, for $m = 0, \ldots, M$ and $k = 0, \ldots, K-2$

$$W_{ij}^{(m)} \begin{cases} \neq 0 & \text{if } j = i \text{ or } j = (i + 2^{k-2}) \mod N \\ = 0 & \text{otherwise.} \end{cases} \tag{1}$$

Thus every factorizing matrix has $N \log_2 N$ stored non-zero entries.

The product of the factorizing matrices corresponds to the connections in the circular graph after multiple hops. We can set the number of factorizing matrices to $M = \log_2 N$, which corresponds to the number of hops, and the resulting matrix product becomes a full matrix with high probability (see Stoica et al., 2001, Theorem 2). In total, there are $N(\log N)^2$ non-zero entries, still much smaller than $N^2$ for a large $N$.

The (Chord) Sparse Factorization (SF) can thus be formulated as the following optimization problem:

$$\underset{W^{(1)},\ldots,W^{(M)}}{\text{minimize}} \left\| X - \prod_{m=1}^{M} W^{(m)} \right\|_F^2 \tag{2}$$

where $W^{(m)}$'s are sparse square matrices with non-zero positions specified by the Chord protocol. The approximation scheme is illustrated in Fig. 2 (bottom).

The minimization in Eq. (2) can be implemented by any existing solvers for unconstrained smooth optimization, where the cost of computing the gradient is $O(NMK)$. Once the approximation error is minimized, we obtain the factorizing matrices $W^{(1)}, \ldots, W^{(M)}$. We call the approach non-parametric SF as we directly optimize over the factorizing matrices.

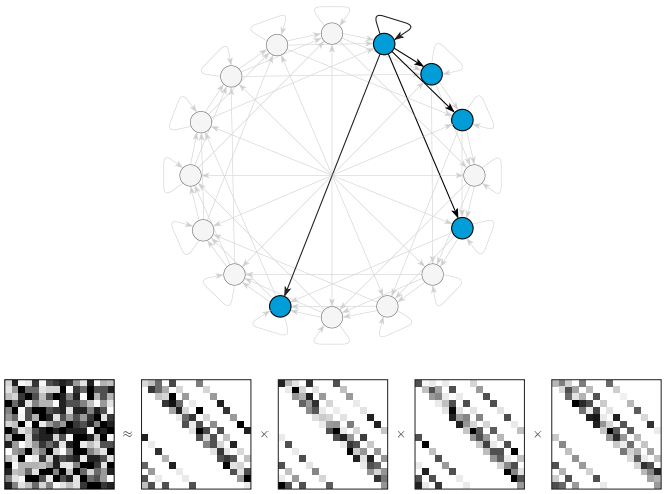


**Fig. 2.** Illustration of (top) the Chord protocol and (bottom) sparse factorization of a square matrix for $N = 16$. Grayscale squares in the factorizing matrices represent stored entries (non-zeros and explicit zeros), and completely white squares represent non-stored entries (implicit zeros).

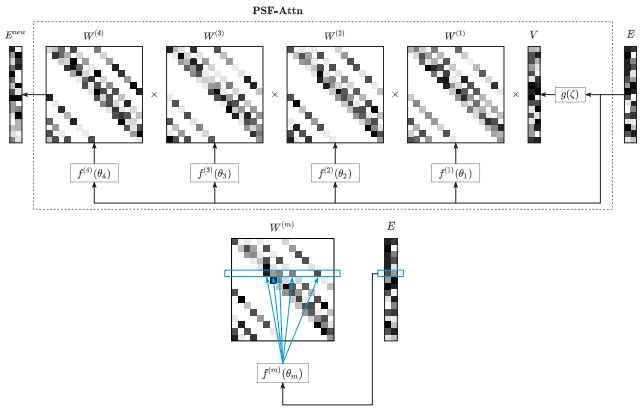



**Fig. 3.** Illustration of (top) the parametric transformation from current embedding $E$ to new embedding $E^{\text{new}}$ based on Sparse Factorization and (bottom) setting of MLP output to non-zero entries in the corresponding sparse factorizing matrix.

## 4. Parametric sparse factorization

Besides direct optimization over the factorizing matrices, we can consider the mapping from vectorial input data to the factorizing matrix entries because each node in the Chord protocol has the same out-degree. The mapping as a component can endow the model (1) generalization to newly coming data and (2) representation learning by transforming the current embedding representation to a new embedding.

We present a transformation model as a concrete example to illustrate the idea. It is a transformer-like model (see Fig. 3 top), where we replace the scaled dot-product attention with a product of sparse square matrices. The matrix product provides an approximation to a full non-normalized attention matrix.

One block of such Parametric Sparse Factorization Attention (PSF-Attn) transforms data in the current embedding $E$ to a new embedding $E^{\text{new}} = \text{PSF-Attn}(E)$. There is a number of MLPs in the block, where the MLP $f^{(m)}$ with parameters $\theta_m$ takes $E_{i:}$ (the $i$th row of $E$) as an input and returns the non-zeros in the $i$th row of $W^{(m)}$, for $i = 1, \ldots, N$ and $m = 1, \ldots, M$. That is, for $k = 2, \ldots, K$,

$$W^{(m)}_{ij} = \begin{cases} \left[f^{(m)}(E_{i:}; \theta_m)\right]_1 & \text{if } j = i \\ \left[f^{(m)}(E_{i:}; \theta_m)\right]_k & \text{if } j = (i + 2^{k-2}) \mod N \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Similar to transformers, we use another MLP $g$ with parameters $\zeta$ to convert an $E$ row to the corresponding $V$ row.

The new model can work as a building block in representation learning frameworks. For example, if we define an objective function $\mathcal{J}$ over $E^{\text{new}}$, the learning can be formulated as the following optimization problem:

$$\underset{\theta_1,\ldots,\theta_M,\zeta}{\text{minimize}} \quad \mathcal{J}(\text{PSF-Attn}(E;\theta_1,\ldots,\theta_M,\zeta)), \tag{4}$$

where the optimization can be implemented with back-propagation and a gradient based algorithm such as Adam (Kingma & Ba, 2014). It is also straightforward to stack multiple PSF-Attn blocks to implement deeper learning.

The PSF-Attn method has two advantages. First, the original transformer and many of its variants (e.g. Choromanski et al., 2020; Katharopoulos, Vyas, Pappas, & Fleuret, 2020; Tay et al., 2021; Wang, Li, Khabsa, Fang, & Ma, 2020) are built upon scaled dot-product, which essentially employ low-rank matrix factorization. They may not work well if the attention is intrinsically high rank. In contrast, all factorizing matrices in our method are full-rank, and so is their product. Therefore PSF-Attn does not suffer from the low-rank bottleneck constraint. Second, our model does not require softmax over a large square matrix, avoiding many computational difficulties. Removing the softmax also endows more freedom in mixing the $V$ rows because the mixture can go beyond the convex hull.

Our method also differs from the previous multi-layer sparse attention approaches (e.g. Child, Gray, Radford, & Sutskever, 2019; Correia, Niculae, & Martins, 2019; Li et al., 2019) because they still use scaled dot-products. PSF-Attn can give full attention in one block. For the $i$th element in the sequence, its attention to other elements can directly be obtained by vector-matrix product $W^{(M)}_{i:}\prod_{m=1}^{M-1} W^{(m)}$.

## 5. Experiments

We have performed four groups of experiments. In the first group, we studied which types of square matrices are more suitable for our method. In the second group, we verified the scalability of the new attention model PSF-Attn on synthetic sequences up to tens of thousands positions. Next, we compare our method with several Transformer variants on a DNA sequences classification task. Finally, we tested PSF-Attn for the Long Range Arena benchmark data sets to see its performance in real-world practice. The first group of experiments was run on a standard PC with an Intel Core i9 CPU. The other groups were run on a Linux server with one NVIDIA-Tesla V100 GPU with 32 GB of memory.

### 5.1. Exploring real-world matrices for SF

There are many types of square matrices. TSVD is the best for approximating matrices close to low rank in terms of F-norm. For non-parametric SF, we performed an empirical study (1) to show that SF can supersede TSVD in F-norm by using the same number of non-zeros and (2) to identify the types of square matrices particularly suitable for SF.

Given a square matrix, we used the Matlab `fminunc` optimizer to solve the problem in Eq. (2), where the non-zero entries in the factorizing matrices are initialized to random numbers between $[K^{-1}, K^{-1}+10^{-2}]$. The total number of non-zero entries for SF and TSVD are $N(\log_2 N)^2$ and $2Nr+r$. For a fair comparison, we set the $r = \lceil(\log_2 N)^2/2\rceil$ in TSVD such that it has nearly the same number and no fewer non-zeros than SF. We first used $256\times 256$ grayscale images as approximated matrices because we can directly see them. Fig. 4 (left) shows six typical square images, where we provide the resulting TSVD and SF approximation errors in F-norm below each image (more examples in the supplemental document).

**Table 1**
Approximation errors in F-norm by using TSVD and SF for different types of square matrices.

| Data type | Data name | TSVD | SF |
|---|---|---|---|
| Dense graph | AuraSonar | **8.54e+00** | 8.68e+00 |
| Dense graph | Protein | 1.17e+01 | **1.09e+01** |
| Dense graph | Voting | **8.07e−04** | 1.71e+01 |
| Dense graph | Yeast | 3.72e+01 | **3.61e+01** |
| Network | Sawmill | 3.24e+00 | **1.03e+00** |
| Network | Scotland | 5.90e+00 | **3.76e+00** |
| Network | A99 m | 1.47e+01 | **1.01e+01** |
| Network | Mexican Power | 3.85e+00 | **1.71e+00** |
| Network | Strike | 2.73e+00 | **1.04e+00** |
| Network | Webkb Cornell | 6.98e+00 | **4.80e+00** |
| Network | Worldtrade | 8.65e+04 | **4.47e+04** |
| Surface mesh | Mesh1e1 | 1.87e+01 | **9.82e+00** |
| Surface mesh | Mesh2e1 | **2.48e+02** | 3.47e+02 |
| Surface mesh | OrbitRaising | 9.37e+01 | **8.35e+01** |
| Surface mesh | Shuttle Entry | 2.73e+03 | **1.86e+03** |
| Surface mesh | AntiAngiogenesis | 5.85e+01 | **3.29e+01** |
| Covariance | Phoneme | **2.80e+01** | 5.27e+01 |
| Covariance | MiniBooNE | **1.04e+00** | 6.36e+03 |
| Covariance | Covertype | 8.22e−02 | **1.90e−02** |
| Covariance | Mfeat | **1.11e+03** | 4.01e+05 |
| Covariance | OptDigits | **3.28e+01** | 7.01e+01 |
| Covariance | PenDigits | 4.00e+02 | **1.87e+02** |
| Covariance | Acoustic | 1.36e−02 | **1.11e−02** |
| Covariance | IJCNN | 5.24e−02 | **3.03e−02** |
| Covariance | Spam Ham | 1.07e−01 | **4.97e−02** |
| Covariance | TIMIT | **9.64e+01** | 1.56e+02 |
| Covariance | Votes | 4.00e−01 | **1.70e−01** |

The `chess` image (matrix) is close to low-rank because the black-and-white chessboard is two-rank. Therefore TSVD performs better as expected. TSVD also works well for the `Lena` and `apple` images because TSVD tends to preserve low-frequency information in images. In contrast, TSVD is not as good as SF for the other three images that contain rich high-frequency details such as lines and corners, which indicates a matrix type where SF can defeat LRMF.

To further verify this, we computed the gradient magnitudes of the images (shown in Fig. 4 right). In this way, the intensities in constant areas become zero, and the remaining non-zeros are mainly high-frequency details. We can see that SF gives a lower approximation error than TSVD for all such matrices, which confirms that SF is more advantageous for approximating matrices with rich high-frequency details.

In summary, the results show that the TSVD performance does not cap SF by using the same number of non-zeros for approximating square matrices. The winning cases indicate that SF is often better than LRMF when the approximated matrix is (1) sparse, (2) intrinsically high-rank, or (3) containing rich high-frequency details.

Besides square images, we have also compared TSVD and SF on several other types of square matrices. Table 1 shows the comparison results. The data types include affinity matrices of dense graphs (dense graph), affinity matrix of sparse networks (network), affinity matrix of surface mesh over 3D objects (surface mesh), and covariance matrix of vectorial data (covariance). We can see that for dense graph and covariance types, sometimes TSVD is better while sometimes SF can win. SF wins for most cases for surface mesh because the mesh networks probably do not have a low-rank structure. SF wins all data sets in the network type, which indicates that SF is more effective for approximating sparse matrices. We give the details of the data sets in the supplemental document.

### 5.2. Approximation to large attention matrices

Here we test whether PSF-Attn is scalable to approximate large attention matrices. We have used two synthetic data sets

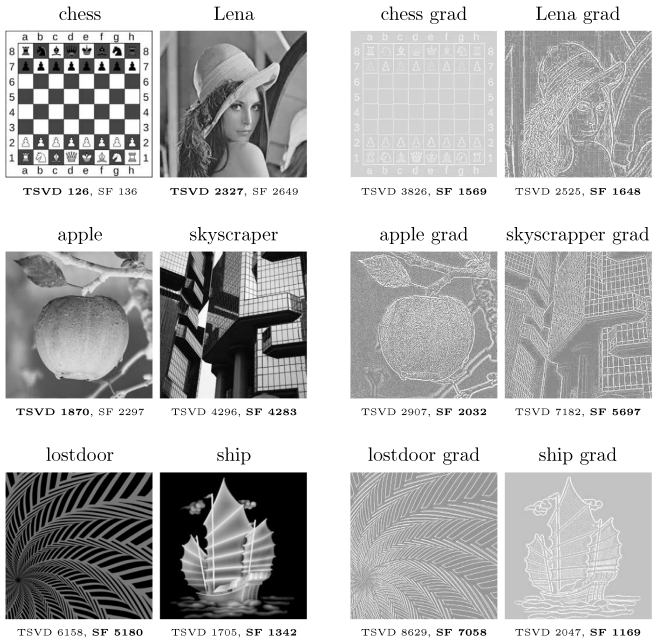



**Fig. 4.** Example square matrices: (left) original images and (right) the gradient magnitude images (displayed after histogram equalization for better visibility). Approximation errors by using TSVD and SF with the same number of non-zeros are shown below the images. Boldface font indicates the winner for each case.

composed of long sequences for supervised learning tasks. A similar experimental setup was used by Hochreiter and Schmidhuber (1997) for scalability tests. The details of the data sets and tasks are given below:

- *Adding Problem*. This is a sequence regression task. Each element of an input sequence is a pair of numbers $(a_i, b_i)$, where $a_i \sim U(-1, 1)$, $b_i \in \{0, 1\}$, $i = 1, \ldots, N$. We generated signals at two randomly selected positions $t_1$ and $t_2$ such that $b_{t_1} = b_{t_2} = 1$ and $b_i = 0$ elsewhere. The learning target is $y = 0.5 + \frac{a_{t_1} + a_{t_2}}{4}$. For example, an input sequence $[(0.1, 0), (-0.4, 1), (0.3, 0), (-0.2, 0), (0.7, 1)]$ will have the learning target $y = 0.575$. Unlike Hochreiter and Schmidhuber (1997), we do not restrict the $t_1$ and $t_2$ choice to make the task more challenging. That is, the relevant signals can appear either locally or at a great distance from each other. In evaluation, a network prediction $\hat{y}$ is considered correct if $|y - \hat{y}| < 0.04$.
- *Temporal Order*. This is a sequence classification task. A sequence consists of randomly chosen symbols from the alphabet $\{a, b, c, d, X, Y\}$, where the first four are noise symbols. Each sequence has two signal symbols, either $X$ or $Y$, which appear at two arbitrary positions. The four target classes correspond to the ordered combinations of the signal symbols $(X, X)$, $(X, Y)$, $(Y, X)$, and $(Y, Y)$. For example, an input sequence $[b, a, c, b, X, a, a, Y, b]$ should be classified as Class 2.

We prepared data of different sequence lengths for each problem. We started with $N = 128$ and gradually increased the length by the factor of two, up to $N = 2^{14}$. For every sequence length, we generated 200,000 training and 5000 testing instances.

We compared PSF-Attn with several popular methods based on scaled dot-product attention (referred to as X-former architectures). The first two, Linformer (Wang et al., 2020) and Performer (Choromanski et al., 2020) have also claimed to be scalable attention-based architectures. For completeness, we also included the original Transformer (Vaswani et al., 2017a). We have used their open-source PyTorch implementations.[4]

We followed standard cross-validation techniques to tune the main hyperparameters, such as the number of layers and heads, dimensionality of the token embedding, and query/key/value dimensions. For the Temporal Order problem, we directly fed the data instances to the embedding layers. For the Adding problem, the input data was only two-dimensional, and one of them was real-valued. Directly using such a low-dimensional embedding space would lead to poor attention. We augmented the dimensionality with a linear layer to assure sufficient freedom for the scaled dot-products in the X-former architectures. All the models were optimized using the Adam optimizer (Kingma & Ba, 2014) with the learning rate of 0.001 using batch size of 40.

The results are shown in Fig. 5. For the Adding problem, we see that all models work fine for short sequences (100% for $N \leq 256$). The X-former methods, however, turn worse or even useless when the sequences are longer. Linformer has an error rate of 0.48% for $N = 512$ and 86.18% for $N = 1024$, which is nearly as bad as random guessing (92%). Transformer becomes problematic (84.48% error) when $N \geq 2048$. Performer starts to get wrong when $N = 4096$, giving only 71.76% accuracy, and when $N =$

[4] Available at https://github.com/lucidrains/performer-pytorch and https://github.com/lucidrains/linformer.

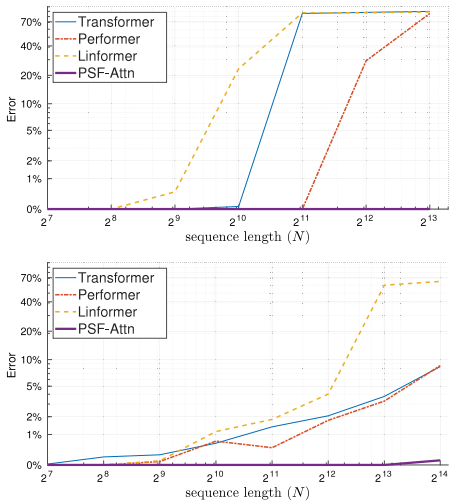



**Fig. 5.** Error percentage of PSF-Attn and the X-formers for (top) the Adding problem and (bottom) the Temporal Order problem with increasing sequence lengths.

**Table 2**
Area under the ROC curve (ROC-AUC) for the compared methods on the genome classification task.

| Model | ROC-AUC |
|---|---|
| Transformer | 85.42% |
| Linformer | 86.30% |
| Performer | 85.40% |
| PSF | **90.04%** |

8192, its prediction becomes almost random guessing. In contrast, PSF-Attn achieves 100% accuracy for all the tested lengths.

A similar pattern holds for the Temporal Order problem. When $N \leq 512$, all compared models give perfect or nearly perfect predictions. With longer sequences, for example when $N = 4096$, the error rates of Transformer, Performer, and Linformer become 2.06%, 1.76%, and 4.02%, respectively. When $N = 16{,}324$, their error rates become 8.38%, 8.60%, and 64.22%, respectively. In contrast, PSF-Attn achieves 100% accuracy for $N \leq 8192$, and 99.89% accuracy for $N = 16{,}324$.

In summary, our method has better scalability than the three attention models based on scaled dot-products in terms of lower learning errors. PSF-Attn can still achieve nearly 100% prediction accuracy for an attention matrix size up to tens of thousands.

### 5.3. Genome classification

It is known that long-range interaction commonly exists in genome sequences (Avsec et al., 2021). Here we used a genome data set DDcDNA containing Complementary DNA (cDNA) sequences of dogs and donkeys[5] Each sample is composed of a set of four symbols: A, C, G, T. The task is to classify each cDNA sequence to its organism (dog or donkey). We removed the sequences shorter than 5k and thus got 6251 sequences for dogs and 2335 for donkeys. The mean length for all sequences is 6985. All the sequences are padded or truncated to a fixed length of 16,384. We used ROC-AUC as the comparison metric due to the class imbalance.

We compare the performance of PSF-Attn with Transformer, Linformer, and Performer on this task. The model hyperparameters were chosen according to the same procedure in Section 5.2. The results are shown in Table 2. We can see that Transformer and its variants show a similar mediocre performance. In contrast, PSF-Attn scores 90.04% ROC-AUC on DDcDNA, much better than the runner-up (Linformer) by 3.7 percentage points, which suggests that PSF-Attn can classify long DNA sequences more accurately.

### 5.4. Long range arena public benchmark

Next we provide the experimental results on Long Range Arena (LRA), a publicly available benchmark for modeling long sequential data (Tay et al., 2020). We select four tasks from LRA that cover various data types and demand flexible reasoning abilities of tested models.

- *ListOps*. This is a sequence classification task for measuring the ability of models to identify and parse hierarchically constructed data (Nangia & Bowman, 2018). We used an enlarged version of the original ListOps, with a max sequence length up to 2000 and tree depth up to 10 (Tay et al., 2020). Each element in a sequence can be an operator, a digit, and a left or right bracket. The brackets define lists of items. Each operator, MAX, MIN, MED, and SUM_MOD, takes the items in a list as input and returns a digit, where MED means median and SUM_MOD means summation followed by modulo 10. An example sequence [MAX 6 [MED 3 2 2] 8 5 [MIN 8 6 2]] has ground truth answer 8. A prediction is correct if an output value of a neural network matches the ground truth label. For good predictions, a model should access all sequence elements and identify the proper parsing structure.
- *Text Classification*. This is a binary sentiment classification task constructed from the IMDb reviews (Maas et al., 2011). Given a review as a sequence of characters, the goal is to classify it as positive or negative. Due to the character-level representation, the sequences are much longer than the word-level version used in conventional language modeling. We truncated or padded every sequence to a fixed length ($N = 4000$).
- *Image Classification*. This task is to classify images into one of ten classes. The images and class labels come from the grayscale version of CIFAR10. Each image is flattened to form a sequence of length 1024. Unlike conventional computer vision, the task requires the predictors to treat the grayscale levels (0–255) as categorical values. That is, each image becomes a sequence of symbols with an alphabet size of 256. Two example images and their class labels are shown in Fig. 6.
- *Pathfinder*. This is a binary classification task on synthetic images, which is motivated by cognitive psychology (Linsley, Kim, Veerabadran, Windolf, & Serre, 2018). Each image (size $32 \times 32$) contains two highlighted endpoints and some path-like patterns. Similar to the Image Classification task, the predictors must treat the pixels as categorical values and flatten the image to a sequence of length 1024. The task is to classify whether there is a path consisting of dashes between two highlighted points. Two example images and their classes are shown in Fig. 6.

[5] Downloaded from the Ensembl genome browser https://www.ensembl.org/.

**Table 3**
Classification accuracies by the compared methods for the four LRA tasks. A dash ("-") means the result is absent in the corresponding paper. For PSF-Attn, we present the mean ($\mu$) and standard deviation ($\sigma$) across multiple runs in the $\mu \pm \sigma$ format.

| Model | ListOps $N = 2000$ | Text $N = 4000$ | Image $N = 1024$ | Pathfinder $N = 1024$ |
|---|---|---|---|---|
| Transformer (Tay et al., 2020) | 36.37 | 64.27 | 42.44 | 71.40 |
| Transformer (Zhu et al., 2021) | 37.13 | 65.35 | – | – |
| Transformer (Xiong et al., 2021) | 37.10 | 65.02 | 38.20 | 74.16 |
| Sparse transformer (Tay et al., 2020) | 17.07 | 63.58 | 44.24 | 71.71 |
| Longformer (Tay et al., 2020) | 35.63 | 62.58 | 42.22 | 69.71 |
| Linformer (Tay et al., 2020) | 37.70 | 53.94 | 38.56 | 76.34 |
| Linformer (Zhu et al., 2021) | 37.38 | 56.12 | – | – |
| Linformer (Xiong et al., 2021) | 37.25 | 55.91 | 37.84 | 67.60 |
| Reformer (Tay et al., 2020) | 37.27 | 56.10 | 38.07 | 68.50 |
| Reformer (Zhu et al., 2021) | 36.44 | 64.88 | – | – |
| Reformer (Xiong et al., 2021) | 19.05 | 64.88 | 43.29 | 69.36 |
| Performer (Tay et al., 2020) | 18.01 | 65.40 | 42.77 | 77.05 |
| Performer (Zhu et al., 2021) | 32.78 | 65.21 | – | – |
| Performer (Xiong et al., 2021) | 18.80 | 63.81 | 37.07 | 69.87 |
| BigBird (Tay et al., 2020) | 36.06 | 64.02 | 40.83 | 74.87 |
| Linear transformer (Tay et al., 2020) | 16.13 | 65.90 | 42.34 | 75.30 |
| Transformer-LS (Zhu et al., 2021) | 38.36 | 68.40 | – | – |
| RFA-Gaussian (Peng et al., 2021) | 36.80 | 66.00 | – | – |
| Nyströmformer (Zhu et al., 2021) | 37.34 | 65.75 | – | – |
| Nyströmformer (Xiong et al., 2021) | 37.15 | 65.52 | 41.58 | 70.94 |
| PSF-Attn | **38.85**±0.1 | **77.32**±0.3 | **45.01**±0.2 | **80.49**±0.1 |

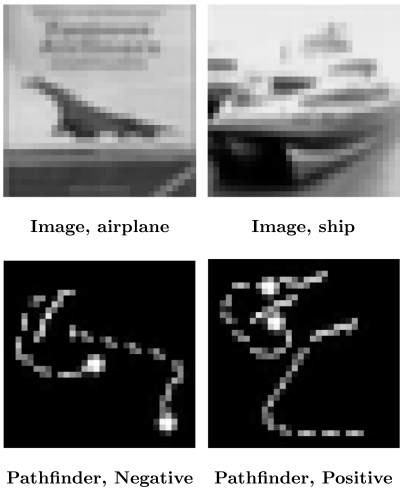



**Fig. 6.** Example matrices: (top two) from the Image Classification and (bottom two) from the Pathfinder tasks.

We have tested PSF-Attn in the above LRA learning tasks, where the hyperparameters in PSF-Attn were again tuned by cross-validation. No external data was used for pre-training. We ran PSF-Attn four times with a different random seed for each task. The mean and standard deviation across the multiple runs are reported in Table 3.

For comparison, we quote the prediction accuracies reported for many X-former methods in the literature, including Transformer (Vaswani et al., 2017b), Sparse Transformer (Child et al., 2019), Longformer (Beltagy, Peters, & Cohan, 2020), Linformer (Wang et al., 2020), Reformer (Kitaev, Kaiser, & Levskaya, 2020), Performer (Choromanski et al., 2020), Linear transformer (Katharopoulos et al., 2020), BigBird (Zaheer et al., 2020), Transformer-LS (Zhu et al., 2021), RFA-Gaussian (Peng et al., 2021), and Nyströmformer (Xiong et al., 2021). If a method has different implementations, we quote all variants and their results. We exclude results that rely on external data for pre-training.

We see that PSF-Attn wins all tasks by giving the best classification accuracy among all compared methods. Such strong cross-task wins suggest that our method usually provides better attention approximation than those based on scaled dot-products.

Remarkably, our method has substantially improved the state-of-the-art in the Text and Pathfinder tasks. For Text, PSF-Attn achieves 77.32% accuracy, compared to the runner-up 68.4% by Transformer-LS. Our method wins by 80.49% accuracy for Pathfinder, which gains about 5% higher than the best X-former (Linear Transformer 75.3%). The significant improvement brought by PSF-Attn is probably because the two tasks involve sparse attention matrices.

We also investigated whether the approximating attention $\widehat{X} = \prod_m W^{(m)}$ is meaningful by visualization. Fig. 7 shows an attention vector (absolute values) of token ["CLS"]. Tokens in the review having more weight are highlighted. We see that PSF-Attn has a good performance in capturing sparse attention and identify the relevant words.

Next, we visualize the attention values for four instances from the Pathfinder task. We extracted the attention matrix $\widehat{X}$ at test time and applied the following visualization procedure. We define the attention vector of the $i$th point as the $i$th row in $\widehat{X}$. We averaged the attention vectors of the endpoints and their direct neighbors and then reshaped the average vector to $32 \times 32$ for visualization.

The resulting visualizations are shown in Fig. 8. Good attention towards correct classification should trace the path between the endpoints and neglect non-relevant high intensities in the original image. It is clear PSF-Attn performs pretty well in this respect, where its attention values highlight mostly the relevant parts of the connecting path. In contrast, Transformer mainly highlights

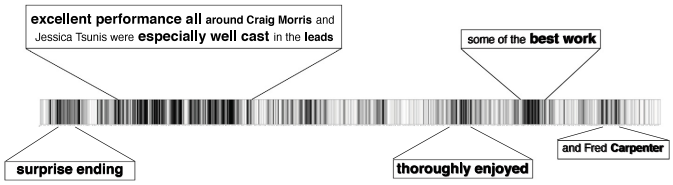



**Fig. 7.** Attention vector visualization of a positive review in Text Classification. Darker cells have larger absolute values.

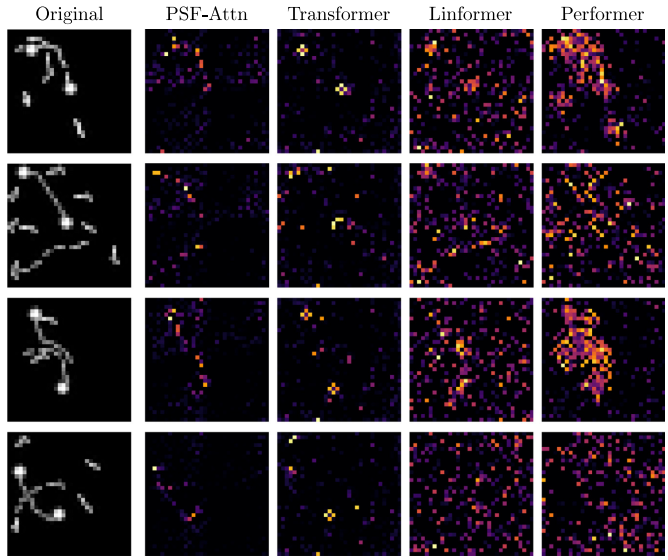



**Fig. 8.** Comparison of the attention maps in the Pathfinder task by using PSF-Attn, Transformer, Linformer, and Performer. The first column of images shows the original instances, and the other columns are heat maps of the attention values of the corresponding models. The procedure for visualization is the same for all the methods.

the area around the endpoints without the path connecting them. For Linformer and Performer, sometimes their attention maps basically smooth the original image intensities, including the irrelevant parts, and sometimes there is simply no pattern. More instances are provided in the supplemental material.

## 6. Conclusion

We have proposed a new approximation method called Sparse Factorization for square matrices by using a product of sparse matrices. Given the same budget of non-zero numbers, our method has shown superior performance over conventional low-rank matrix factorization approaches, especially in cases where the approximated matrix is sparse, high-rank, or contains high-frequency details. We have also given parametric design and performed an empirical study on the classification of long sequential data. As the critical attention component, our method has demonstrated clear wins over the conventional scaled dot-product transformer and its several variants in terms of scalability and accuracy.

We have employed the Chord protocol to fix the non-zero positions in the sparse factorizing matrices. Later we could consider the other predefined protocols or even adaptively learned protocols for the sparse structure. In this work, we have considered approximating unconstrained square matrices. In the future, we could investigate the approximation to more specific matrices such as non-negative, stochastic, symmetric, or semi-definite matrices. As a result, we could extend the method for further applications such as large-scale graph matching, Gaussian Process, or network structure identification.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgment

This work was financially supported by The Research Council of Norway, Grant Number 287284. We acknowledge for using the IDUN computing cluster (Själander et al., 2019).

## Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.neunet.2022.04.014.

## References


Avsec, Ž., Agarwal, V., Visentin, D., Ledsam, J. R., Grabska-Barwinska, A., Taylor, K. R., et al. (2021). Effective gene expression prediction from sequence by integrating long-range interactions. *Nature Methods*, *18*(10), 1196–1203.

Beltagy, I., Peters, M. E., & Cohan, A. (2020). Longformer: The long-document transformer. arXiv preprint arXiv:2004.05150.

Child, R., Gray, S., Radford, A., & Sutskever, I. (2019). Generating long sequences with sparse transformers. arXiv preprint arXiv:1904.10509.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., et al. (2020). Rethinking attention with performers. arXiv:2009.14794.

Correia, G. M., Niculae, V., & Martins, A. F. (2019). Adaptively sparse transformers. arXiv preprint arXiv:1909.00015.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297.

Donoho, D. L. (2006). For most large underdetermined systems of linear equations, the minimal $l_1$-norm solution is also the sparsest solution. *Communications on Pure and Applied Mathematics*, *59*(6), 797–829.

Drineas, P., & Mahoney, M. W. (2005). On the nyström method for approximating a gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, *6*, 2153–2175.

Eckart, C., & Young, G. (1936). The approximation of one matrix by another of lower rank. *Psychometrika*, *1*(3), 211–218.

Funk, S. (2006). Funk matrix factorization. Winner solution of the Netflix Prize, https://sifter.org/~simon/journal/20061211.html.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Katharopoulos, A., Vyas, A., Pappas, N., & Fleuret, F. (2020). Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning* (pp. 5156–5165). PMLR.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.

Kitaev, N., Kaiser, Ł., & Levskaya, A. (2020). Reformer: The efficient transformer. arXiv preprint arXiv:2001.04451.

Lee, D. D., & Seung, H. S. (1999). Learning the parts of objects by nonnegative matrix factorization. *Nature*, *401*, 788–791.

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X., et al. (2019). Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *Advances in neural information processing systems, vol. 32*.

Linsley, D., Kim, J., Veerabadran, V., Windolf, C., & Serre, T. (2018). Learning long-range spatial dependencies with horizontal gated recurrent units. In *Proceedings of the 32nd international conference on neural information processing systems* (pp. 152–164).

Maas, A., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies* (pp. 142–150).

Mahoney, M. W., & Drineas, P. (2009). CUR matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, *106*(3), 697–702.

Nangia, N., & Bowman, S. R. (2018). Listops: A diagnostic dataset for latent tree learning. arXiv preprint arXiv:1804.06028.

Peng, H., Pappas, N., Yogatama, D., Schwartz, R., Smith, N. A., & Kong, L. (2021). Random feature attention. arXiv preprint arXiv:2103.02143.

Själander, M., Jahre, M., Tufte, G., & Reissmann, N. (2019). EPIC: an energy-efficient, high-performance GPGPU computing research infrastructure. arXiv preprint arXiv:1912.05848.

Slawski, M., Hein, M., & Lutsik, P. (2013). Matrix factorization with binary components. In *Advances in neural information processing systems, vol. 26*.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., & Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, *31*(4), 149–160.

Tay, Y., Bahri, D., Metzler, D., Juan, D.-C., Zhao, Z., & Zheng, C. (2021). Synthesizer: Rethinking self-attention for transformer models. In *International conference on machine learning* (pp. 10183–10192). PMLR.

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., et al. (2020). Long range arena: A benchmark for efficient transformers. arXiv preprint arXiv:2011.04006.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017a). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998–6008).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017b). Attention is all you need. In *Advances in neural information processing systems, vol. 30*.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., & Ma, H. (2020). Linformer: Self-attention with linear complexity. arXiv:2006.04768.

Williams, C., & Seeger, M. (2001). Using the nyström method to speed up kernel machines. In *Advances in neural information processing systems, vol. 13*.

Xiong, Y., Zeng, Z., Chakraborty, R., Tan, M., Fung, G., Li, Y., et al. (2021). Nyströmformer: A nyström-based algorithm for approximating self-attention. arXiv preprint arXiv:2102.03902.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., et al. (2020). Big bird: Transformers for longer sequences. In *NeurIPS*.

Zhang, Z., Li, T., Ding, C., & Zhang, X. (2007). Binary matrix factorization with applications. In *Proceedings of international conference on data mining* (pp. 391–400).

Zhu, C., Ping, W., Xiao, C., Shoeybi, M., Goldstein, T., Anandkumar, A., et al. (2021). Long-short transformer: Efficient transformers for language and vision. arXiv preprint arXiv:2107.02192.

# Sparse Factorization of Square Matrices with Application to Neural Attention Modeling (Supplemental Document)

Ruslan Khalitov*, Tong Yu*, Lei Cheng, Zhirong Yang**
*Norwegian University of Science and Technology*

**Abstract**

This document gives supplemental information to the main paper.

## 1. Non-parametric Experiments

### *1.1. Results on other square images*

In addition to six square images from the main paper, we have studied another six square images using the previously described approach. The results are shown in Figure 1. The conclusions are aligned with those in the main paper

- There exist cases where SF performs better than TSVD, which means the SF approximation quality is not capped by TSVD by using the same number of non-zeros.
- SF is more likely to win for images where there are more high-frequency details, especially for the gradient-magnitude images.

*Equal contribution
**Corresponding author
*Email address:* zhirong.yang@ntnu.no (Zhirong Yang)

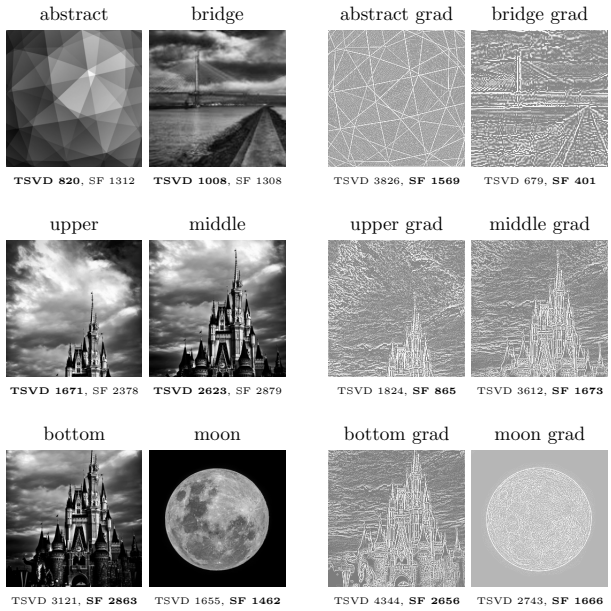



Figure 1: Other example square matrices: (left) original images and (right) the gradient magnitude images (displayed after histogram equalization for better visibility). Image names are approximation errors by using TSVD and SF with the same number of non-zeros are shown below the images. Boldface font indicates the winner for each case.

### *1.2. Details of Other Square Matrices*

In the main paper (Table 1), we have present comparison between TSVD and SF for approximating different types of square matrices using the same number of non-zeros. Here we provide the sources and statistics of the square matrices.

- `AuralSonar` ($N = 100$). It is the *Aural Sonar* data set from (Chen et al., 2009). The original research investigated the human ability to distinguish different types of sonar signals by ear (Philips et al., 2006).

- `Protein` ($N = 213$). It is the *Protein* data set from (Chen et al., 2009), which contains the radial basis function (RBF) kernel between 213 proteins.

- `Voting` ($N = 435$). It is the *Voting* data set from (Chen et al., 2009), which contains dissimilarities between 435 voting records with 16 scaled voting attributes.

- `Yeast` ($N = 200$). It is the *Yeast-SW-7-12* data set from the same repository in (Chen et al., 2009). The data set converts the pairwise Smith-Waterman similarities $s_{ij}$(Lanckriet et al., 2004; Xu et al., 2014) to dissimilarities by $d_{ij} = \sqrt{s_{ii} + s_{jj} - s_{ij} - s_{ji}}$.

- `Sawmill` ($N = 36$). It is the *Sawmill* communication network from the Pajek data sets[1]. This is a sparse matrix with 124 non-zero entries.

- `Scotland` ($N = 108$). It is the *Scotland* network from the Pajek data sets, which is about Corporate interlocks in Scotland (1904-5). The matrix is sparse with 644 non-zero entries.

- `A99m` ($N = 234$). It is the *GD'99 - Linden strasse* network from the Pajek data sets. The network is about the characters and their relations in the long-running German soap opera called 'Lindenstrasse'. The matrix is sparse with 510 non-zero entries.

- `Mexican power` ($N = 35$). It is the *Mexican* network from the Pajek data sets. The network contains the core of this political elite: the presidents and their closest collaborators. The matrix is sparse with 117 non-zero entries.

- `Strike` ($N = 24$). It is the *Strike* network from the Pajek data sets. This is a social network about informal communication within a sawmill on strike. The matrix is sparse with 38 non-zero entries.

[1] available at `http://vlado.fmf.uni-lj.si/pub/networks/data/`

- `Webkb Cornell` ($N = 195$). It is the Cornell subset in the LINQS WebKB data set[2]. The network is about citations among the 195 publication from Cornell. The matrix is sparse with 304 non-zero entries.

- `WorldTrade` ($N = 80$). It is the *World_trade* network in the Pajek data sets. The network is about world trade in miscellaneous manufactures of metal, 1994. The matrix is sparse with 998 non-zero entries.

- `Mesh1e1` ($N = 48$). It is the *mesh1e1* data set in SuiteSparse Matrix Collection[3]. The matrix was originally from NASA, collected by Alex Pothen. It is a sparse matrix with 306 non-zero entries.

- `Mesh2e1` ($N = 306$). It is the *mesh2e1* data set in SuiteSparse Matrix Collection. The matrix was originally from NASA, collected by Alex Pothen. It is a sparse matrix with 2018 non-zero entries.

- `OrbitRaising` ($N = 442$). It is the *orbitRaising_1* data set in SuiteSparse Matrix Collection. The matrix was from an optimal control problem. It is a sparse matrix with 2906 non-zero entries.

- `Shuttle Entry` ($N = 560$). It is the *spaceShuttleEntry_1* data set in SuiteSparse Matrix Collection. The matrix was from an optimal control problem. It is a sparse matrix with 6891 non-zero entries.

- `AntiAngiogenesis` ($N = 205$). It is the *tumorAntiAngiogenesi_1* data set in SuiteSparse Matrix Collection. The matrix was from an optimal control problem. It is a sparse matrix with 1783 non-zero entries.

- `Phoneme` ($N = 256$). It is the covariance matrix of the *Phoneme* data set accompanied with the Elements of Machine Learning book (Hastie et al., 2001). The original data[4] has 4508 instances of 256 dimensions.

[2]available at `https://linqs.soe.ucsc.edu/data`
[3]available at `https://sparse.tamu.edu/`
[4]available at `https://web.stanford.edu/~hastie/ElemStatLearn/data.html`

- `MiniBooNE` ($N = 50$). It is the covariance matrix of the *MiniBooNE particle identification* data set in the UCI Repository[5]. The original data has 130064 instances of 50 dimensions.

- `Covertype` ($N = 54$). It is the covariance matrix of the *Covertype* data set in the UCI Repository. The original data has 581012 instances of 54 dimensions.

- `Mfeat` ($N = 649$). It is the covariance matrix of the *Multiple Features* data set in the UCI Repository. The original data has 2000 instances of 649 dimensions.

- `OptDigits` ($N = 64$). It is the covariance matrix of the *Optical Recognition of Handwritten Digits* data set in the UCI Repository. The original data has 5620 instances of 64 dimensions.

- `PenDigits` ($N = 16$). It is the covariance matrix of the *Pen-Based Recognition of Handwritten Digits* data set in the UCI Repository. The original data has 10992 instances of 16 dimensions.

- `Acoustic` ($N = 50$). It is the covariance matrix of the *SensIT Vehicle (acoustic)* data set in the LIBSVM Classification (Multi-class) data collection[6]. The original data has 98528 instances of 50 dimensions.

- `IJCNN` ($N = 22$). It is the covariance matrix of the *ijcnn1* data set in the LIBSVM Classification (Binary class) data collection[7]. The original data has 126701 instances of 22 dimensions.

- `Spam Ham` ($N = 448$). It is the covariance matrix of a data set for email classification practice. The original data has 10000 instances of 448 features.

[5] available at `https://archive.ics.uci.edu/ml/`

[6] available at `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html`

[7] available at `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`

**Easy** **Medium** **Hard**

Figure 2: Example images form the Pathfinder task.

- `TIMIT` ($N = 390$). It is the covariance matrix of the TIMIT speech recognition data[8]. There are 151290 instances, each contains 390 features (concatenating MFCC features in 10 consecutive 30ms windows).

- `Votes` ($N = 16$). It is the covariance matrix of the *Congressional Voting Records* data set in the UCI Repository. The original data has 435 instances of 16 dimensions.

## 2. Long Range Arena Data Description

In this section, we provide a detailed explanation of the LRA experiments (Tay et al., 2020). The technical information, such as memory consumption and time per epoch, is provided in Table 1. In addition, in Figure 3 we visualized attention maps for five additional instances from the Pathfinder task.

The data set for the LRA benchmark is publicly available. The information about data and the download link can be found in the official GitHub repository: `https://github.com/google-research/long-range-arena`.

- **ListOps** The raw data for this problem is organized as three separate files `basic_train.tsv`, `basic_test.tsv`, `basic_val.tsv` for training, testing, and validation data, respectively. The split is fixed. In addition to the tokens described in the main paper, each sequence has "(" and ")" symbols, which should be removed. To equalize the lengths of the sequences,

[8] available at `https://catalog.ldc.upenn.edu/LDC93S1`

we used the built-in PyTorch padding functional. After the sequences are prepared, the embedding layer processes each unique value, thus mapping elements to the embedding space. The rest of the training process is straightforward.

- **Text Classification** The IMDB data set is downloaded as a built-in object from the `tensorflow data set` library. It includes 25 000 instances for training and another 25 000 for the test set. To transform the raw reviews into sequences, we first went through the whole corpus and extracted the character vocabulary. Then we mapped each sequence to a vector of indices using this vocabulary. Finally, we truncated or padded each sequence to a fixed length of $4k$ and started training.

- **Image Classification** CIFAR10 is a well-known data set, which can be downloaded from the `torchvision` package. The train/test splitting is fixed. To make images gray-scaled, we used standard transformation `transforms.Grayscale` from the same package. An image is flattened to a sequence of length 1024. Then each element is mapped to a dictionary of size 256 (all possible intensity values) and given to the embedding layer.

- **Pathfinder** The problem data consists of two types of files: images and metafiles. Metafiles store information about all the images and their corresponding labels (positive or negative). There are three classes of images: `curv_contour_length_14` (hard), `curv_contour_length_9` (medium), and `curv_baseline` (easy). An image class corresponds to the distance between its endpoints (curve length), thus positively correlates with the difficulty level. Three images from the different catalogs are present in Figure 2. The exact data split is not provided. To separate the data into three parts, we iterated over all metafiles from the catalogs and constructed the training/val/test (90%/5%/5%) sets such that all three types of images are present equally. The rest of the processing is similar to the Image Classification task.

**References**


Chen, Y., Garcia, E.K., Gupta, M.R., Rahimi, A., Cazzanti, L., 2009. Similarity-based classification: Concepts and algorithms. Journal of Machine Learning Research 10, 747–776.

Hastie, T., Tibshirani, R., Friedman, J., 2001. The Elements of Statistical Learning. Springer New York Inc.

Lanckriet, G., Deng, M., Cristianini, N., Jordan, M., Noble, W., 2004. Kernel-based data fusion and its application to protein function prediction in yeast. Biocomputing 2004, Proceedings of the Pacific Symposium, Hawaii, USA , 300–311.

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., Metzler, D., 2020. Long range arena: A benchmark for efficient transformers. arXiv preprint arXiv:2011.04006 .

Xu, W., Hancock, E.R., Wilson, R.C., 2014. Ricci flow embedding for rectifying non-euclidean dissimilarity data. Pattern Recognition 47, 3709–3725.

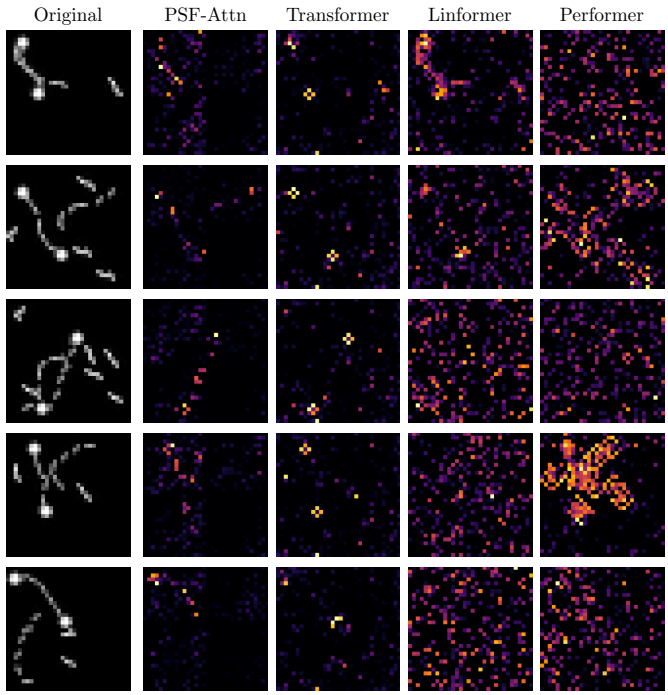



Figure 3: Comparison of the attention maps in the Pathfinder task by using PSF-Attn, Transformer, Linformer, and Performer. The first column of images shows the original instances, and the other columns are heat maps of the attention values of the corresponding models. The procedure for visualization is the same for all the methods.

Table 1: Technical details of the PSF-Attn training on all experiments: time per epoch (sec.), number of training parameters, and peak memory usage (GB). Benchmarks are run on a Linux machine with one NVIDIA Tesla V100 32GB, Intel Xeon Gold 6240 CPU @ 2.60GHz processors, with 754GB of system memory.

| Problem | $N$ | Train Size (k) | Time | Param. (M) | Memory |
|---|---|---|---|---|---|
| ListOps | 2000 | 96 | 431 | 1.86 | 7.90 |
| Text | 4096 | 25 | 74 | 0.21 | 6.54 |
| Image | 1024 | 50 | 21 | 0.29 | 1.73 |
| Pathfinder | 1024 | 540 | 374 | 0.17 | 3.58 |
| Adding | $2^7$ | 200 | 34 | 0.01 | 1.40 |
| | $2^8$ | 200 | 36 | 0.02 | 1.43 |
| | $2^9$ | 200 | 39 | 0.03 | 1.52 |
| | $2^{10}$ | 200 | 45 | 0.06 | 1.73 |
| | $2^{11}$ | 200 | 87 | 0.10 | 2.12 |
| | $2^{12}$ | 200 | 175 | 0.18 | 3.08 |
| | $2^{13}$ | 200 | 377 | 0.35 | 5.12 |
| | $2^{14}$ | 200 | 845 | 0.68 | 9.97 |
| Order | $2^7$ | 200 | 34 | 0.02 | 1.40 |
| | $2^8$ | 200 | 38 | 0.03 | 1.43 |
| | $2^9$ | 200 | 45 | 0.05 | 1.52 |
| | $2^{10}$ | 200 | 61 | 0.08 | 1.74 |
| | $2^{11}$ | 200 | 104 | 0.15 | 2.14 |
| | $2^{12}$ | 200 | 193 | 0.28 | 3.10 |
| | $2^{13}$ | 200 | 397 | 0.55 | 5.17 |
| | $2^{14}$ | 200 | 871 | 1.07 | 10.47 |

# Publication B - Paramixer: Parameterizing mixing links in sparse factors works better than dot-product self-attention

# Paramixer: Parameterizing Mixing Links in Sparse Factors Works Better than Dot-Product Self-Attention

Tong Yu * Ruslan Khalitov * Lei Cheng Zhirong Yang †
Norwegian University of Science and Technology

## Abstract

*Self-Attention is a widely used building block in neural modeling to mix long-range data elements. Most self-attention neural networks employ pairwise dot-products to specify the attention coefficients. However, these methods require $O(N^2)$ computing cost for sequence length $N$. Even though some approximation methods have been introduced to relieve the quadratic cost, the performance of the dot-product approach is still bottlenecked by the low-rank constraint in the attention matrix factorization. In this paper, we propose a novel scalable and effective mixing building block called Paramixer. Our method factorizes the interaction matrix into several sparse matrices, where we parameterize the non-zero entries by MLPs with the data elements as input. The overall computing cost of the new building block is as low as $O(N \log N)$. Moreover, all factorizing matrices in Paramixer are full-rank, so it does not suffer from the low-rank bottleneck. We have tested the new method on both synthetic and various real-world long sequential data sets and compared it with several state-of-the-art attention networks. The experimental results show that Paramixer has better performance in most learning tasks.*

## 1. Introduction

Transformer models have been widely used on many tasks such as text classification [28], text summarization, promoter region prediction [33], and image classification [10]. The main engine in Transformer is the self-attention mechanism, which can work in parallel to mix long-range tokens in a long sequence. This fundamental innovation eliminated the sequential dependency in recurrent neural networks and was used as a building block for many powerful models, such as Bert [9], GPT [6] and Ernie [25].

However, the original self-attention is not scalable because it requires computing and storing all pairwise dot-products, which incurs $O(N^2)$ cost for sequence length $N$. The scalability issue significantly restricted the application of neural models based on self-attention.

Various methods have been introduced to alleviate the quadratic cost of full attention. Some of them attempt to shorten the sequence length [2, 29], even though much information is lost. Others try to break up the softmax by a certain kernel factorization. Another family of methods sparsify the attention matrix with predefined attention [3, 4, 7, 27, 33]. However, most Transformer variants stick to the dot-product self-attention, of which the expressive power is restricted by the low-rank bottleneck [5] because the dimensionality of the dot-product space is much smaller than the sequence length. Therefore, they cannot accurately model the transformation if the attention is intrinsically high-rank.

This paper proposes a scalable and effective attention building block called Paramixer without dot-product and softmax. Our method directly parameterizes the mixing links in several sparse factors to form an attention matrix, where all factorizing matrices are full-rank. Therefore Paramixer does not suffer from the low-rank bottleneck. We present two ways to specify the non-zero positions in each sparse factor. Both lead to an economical approximation of the full attention matrix, with the computing cost as low as $O(N \log N)$. As a result, our method can easily model very long sequential data.

We have tested Paramixer on various sequence data sets and compared it with many popular self-attention neural networks based on dot-products. The experimental results show that Paramixer gets the best performance on very long sequence tasks, including synthetic data inference, Genome classification, and character-level long document classification. Paramixer also achieves state-of-art accuracy on the public Long Range Arena benchmark tasks.

We organize the rest of the paper as follows. Section 2 investigates dot-product self-attention and its related work. Section 3 introduces the development clue and model architecture of Paramixer. The experimental settings and results are presented in Section 4, and we conclude the paper in Section 5.

*Equal contribution.
†Corresponding author. zhirong.yang@ntnu.no

## 2. Related Work

Self-attention (SA) is a building block in neural networks which enables long range interaction between elements in a sequence. The most widely used SA architecture is called Transformer [28], where the self-attention matrix is constructed using scaled dot-product followed by softmax. Given an input sequence of $N$ elements encoded in $X \in \mathbb{R}^{N\times d}$, a self-attention building block calculates a weighted average of feature representations $V \in \mathbb{R}^{N\times d_v}$, where the weights are the result of scaled dot-product of $Q \in \mathbb{R}^{N\times D}$ and $K \in \mathbb{R}^{N\times D}$: $Q = XW_q$, $K = XW_k$, and $V = XW_v$. Then the self-attention in Transformer is

$$\text{Self-Attention}(Q, K, V) = AV, \tag{1}$$

where

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right) \tag{2}$$

and the softmax applies row-wise on the scaled dot-products.

The attention matrix in Eq. 2 is not scalable because it requires computing and storing $N^2$ attention values, which is infeasible for a large $N$. The quadratic cost becomes a significant bottleneck when applying self-attention applications for long sequences. Many research teams have proposed Transformer variants to relieve this problem.

The first branch of methods attempts to reduce $N$. The Linformer approximation [30] uses random projection to reduce the rows of $K$ and $V$ from $N$ to $r$ with $r < N$. However, because $r \propto \epsilon^{-2}$ with $\epsilon$ the approximation error bound, Linformer has to use a large $r$ to achieve satisfactory approximation quality. Another method called Enformer [2] shortens the sequence length by convolutional network pooling.

The second branch of methods tries to break up the softmax by a certain kernel factorization $A \approx \phi(Q)\phi(K)^T$, where $\phi_Q$ and $\phi_K \in \mathbb{R}^{N\times r'}$ with $r' < N$. Then by the association rule of multiplication, we can calculate $\phi(Q)\left[\phi(K)^T V\right]$ and avoid the quadratic cost. For example, Nyströmformer [31] uses a few landmarks as surrogates to construct $\phi(Q)$ and $\phi(K)$. Linear Transformer [15] directly chooses $\phi(x) = \text{elu}(x) + 1$. Performer [8] uses $r'$ orthogonal random features to obtain $\phi(Q)$ and $\phi(K)$. Random features were also used in another work [22], with gating mechanism combined.

The third branch of methods chooses only a subset of $(i, j)$ pairs in each attention layer. These include Sparse Transformers [3, 7] that use a set of neighboring tokens, Sinkhorn Transformer [27] that uses blockwise sparsity, Longformer [4] that uses dilated sparse connections, and BigBird [33] that uses both blockwise and dilated sparse connections.

## 3. Paramixer

Despite many variants of Transformers, there are still several drawbacks because they stick to dot-product + softmax or their approximation. Below we discuss the disadvantages of these two components and propose our solution without them.

### 3.1. Drawbacks of dot-products and softmax

The first drawback is the low-rank bottleneck: dot-product self-attention requires $D \ll N$; otherwise, the $Q$ and $K$ matrices are unaffordable for a large $N$. Bhojanapalli et al. have shown that the low-rank bottleneck restricts the representation power and attention performance [5]. However, their workaround that uses $D = N$ does not apply for long sequences due to the quadratic cost.

Another limitation comes from the pairwise definition of dot-product. Although dot-products have a theoretical connection to Reproducing Kernel Hilbert Space (RKHS), self-attention cannot benefit from more than two factors in the matrix product because the kernels are defined in pairs.

Another key component, softmax, in Transformer is also problematic. First, such nonlinearity over dot-products leads to quadratic cost. Comprehensive approximation methods are required to approximate softmax for more economical matrix products.

Second, softmax limits the mixing capability. The softmax layer output probabilities and the mixing result are thus constrained in the convex hull of the existing elements. The constraint is more severe in sparse attention methods such as [3, 4, 7, 27, 33], where the mixing result is in a rather constrained convex hull of a few involved elements.

Moreover, softmax is incompatible with sparse attention. The latter was introduced to relieve the quadratic computing cost caused by softmax. However, after each sparse attention layer with softmax, the network can output probabilities for only a few sequence elements.

### 3.2. Parameterizing mixing links

Seeing the drawbacks of dot-products and softmax, we rethink self-attention as a transformation block and redesign the neural model. First, we drop the softmax because it is not compulsory for the mixing function but limits the mixing capability. Then the transformation becomes $V^{\text{new}} = AV$ for an unconstrained mixing matrix $A$.

Next, we consider parameterizing the mixing links or coefficients for each matrix row $A_{i:} = f(X_i; \theta)$, where $f : \mathbb{R}^d \mapsto \mathbb{R}^N$ is a neural network with weights $\theta$. However, such simple parameterization does not solve the quadratic cost. We thus consider a sparse factorization of the mixing

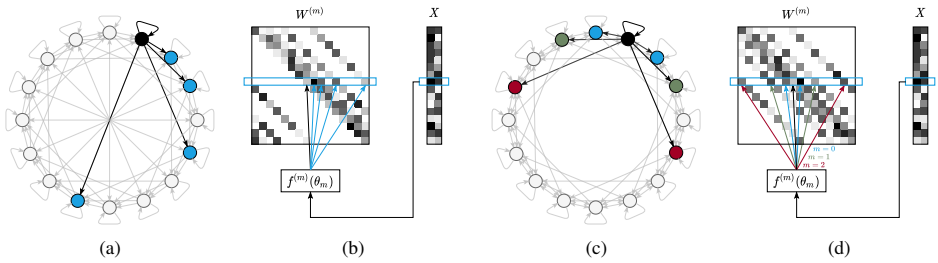



Figure 1. Illustration of (a & b) the CHORD and (c & d) the CDIL protocols for $N = 16$. Each node in the circular graph represents a sequence element. The links between nodes correspond to the non-zero entries in $W^{(m)}$ (here $m = 1$) output from $f^{(m)}$. Note that the sparse structure of all factors in CHORD is the same, while it varies at different $m$'s in CDIL.

matrix $A$:

$$A = \prod_{m=1}^{M} W^{(m)}, \tag{3}$$

where each sparse factor $W^{(m)}$ is a full-rank sparse square matrix. Then we parameterize each sparse factor $W^{(m)}$ as

$$W^{(m)}_{i:} = f^{(m)}(X_i; \theta_m), \tag{4}$$

where $i = 1, \ldots, N$ and $f : \mathbb{R}^d \mapsto \mathbb{R}^K$ is a Multilayer Perceptron (MLP) that outputs the $K$ non-zero entries[1] of each row in $W^{(m)}$. If the total number of non-zeros in the factors is much smaller than $N^2$, we obtain a new economical self-attention method without dot-product and softmax. We call the new method Paramixer.

There are different ways or protocols to specify the sparse structure or the non-zero entries. We have studied two protocols and present them below. For short, we abbreviate $f^{(m)}_{ik} \stackrel{\text{def}}{=} \left[f^{(m)}(X_i; \theta_m)\right]_k$.

The first protocol CHORD modifies from an algorithm with the same name in peer-to-peer lookup service [24], where the $i$-th row of each sparse factor $W^{(m)}$ is parameterized as

$$W^{(m)}_{ij} = \begin{cases} f^{(m)}_{i1} & \text{if } j = i \\ f^{(m)}_{ik} & \text{if } j = i + 2^{k-2} \mod N \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

where $j = 1, \ldots, N$ and $k = 1, ..., K$. That is, each row has $K$ non-zeros, and in total all $W^{(m)}$'s have $MNK$ non-zeros. In this work, we follow the original CHORD algorithm to use $K = M = \log N$. Therefore the number of stored entries is $N \log^2 N$.

[1]Non-zero entries are those stored, including both non-zeros and explicit zeros.

Each factor $W^{(m)}$ can be treated as the adjacency matrix of a directed graph, where the $(i, j)$-th non-zero entry can be seen as a link from $i$ to $j$. The graph visualization and the parameterization are illustrated in Figure 1.

The product of the factorizing matrices corresponds to the connections in the circular graph after multiple sparse factors. Theorem 2 in [24] guarantees that the graph will become complete with high probability after $M = \log N$ sparse factors. That is, the resulting $A$ matrix will become full after the matrix product.

The second protocol CDIL (Circular DILated) originates from Temporal Convolution Networks (TCN) [18], where we modify the dilated connections at both sides and along a circular graph to ensure the receptive fields are symmetric. The $i$-th row of $W^{(m)}$ is parameterized as

$$W^{(m)}_{ij} = \begin{cases} f^{(m)}_{i1} & \text{if } j = i \\ f^{(m)}_{ik} & \text{if } j = i + p_{k-1} 2^m \mod N \\ 0 & \text{otherwise,} \end{cases} \tag{6}$$

where $p = \left[1, 2, ..., \frac{K-1}{2}, -1, -2, -\frac{K-1}{2}\right]$ and $k = 2, \ldots, K$. Similar to CHORD, the CDIL protocol includes self links but the other links appear on both sides, and the dilation $2^m$ varies at different $m$'s. The CDIL protocol and parameterization are illustrated in Figure 1.

In CDIL, each factor contains $KN$ non-zero entries, and in total, there are $KN \log N$ non-zeros if $M = \log N$, which is more economical than CHORD if $K < \log N$. It can be proven that the product $A = \prod_{m=1}^{M} W^{(m)}$ is a full matrix with the CDIL protocol.

The following proposition states that the factorizing matrices constructed by the two protocols are full-rank. The proof in given the supplemental document.

**Proposition 3.1.** *$W^{(1)}, \ldots, W^{(M)}$ constructed in Eq. 5 or Eq. 6 are in general full-rank.*

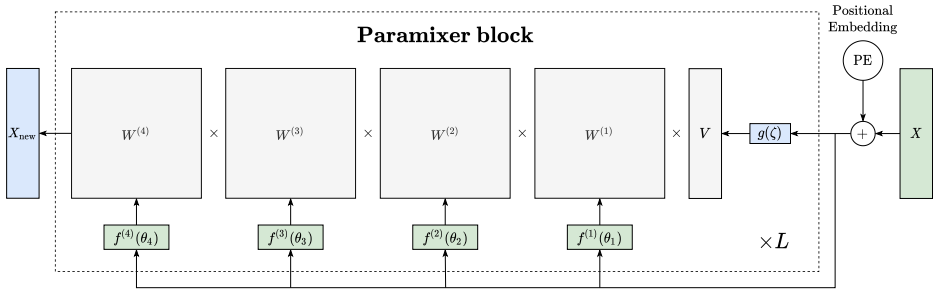



Figure 2. Illustration of an example Paramixer neural network ($M = 4$). After adding the positional embedding, it applies $L$ Paramixer blocks and obtains the transformed tensor $X_{\text{new}}$.

### 3.3. Paramixer neural networks

A Paramixer block transforms an $N \times d$ tensor $X$ to another tensor of the same size. See Figure 2. Here we use an MLP $g : \mathbb{R}^d \mapsto \mathbb{R}^d$ with weights $\zeta$ to mix the columns of the input tensor, which is similar to the product with $W_v$ in Transformer. In this work, we used simple three-layer MLPs (Linear-GELU-Linear) for both $f^{(m)}$'s and $g$. Stacking $L$ such blocks forms a neural network backbone, denoted by ParamixerNet, for representation learning.

We have tried two options for parameterization in multi-block setting: $W^{(l,m)} = f^{(l,m)}\left(X^{(l-1)}; \theta_m^{(l)}\right)$ and $W^{(l,m)} = f^{(l,m)}\left(X^{(0)}; \theta_m^{(l)}\right)$, where the superscript $l$ indexes the $l$-th block and $X^{(l-1)}$ is the input to the $l$-th block, with $X^{(0)} = X + \text{PE}$. We find that the latter option makes the network easier to train and works better.

The ParamixerNet output can then be fed to a loss function $\mathcal{J}$, and overall, the learning task can be formulated as the following optimization problem:

$$\underset{\Theta}{\text{minimize}} \quad \mathcal{J}(\text{ParamixerNet}(X + \text{PE}; \Theta)), \tag{7}$$

where $\Theta = \left\{\theta_1^{(l)}, \ldots, \theta_M^{(l)}, \zeta^{(l)}\right\}_{l=1}^{L}$. The optimization can be implemented with back-propagation, and a gradient-based algorithm such as Adam [16].

## 4. Experiments

We conducted four groups of experiments. In the first group, we demonstrate the scalability of Paramixer on long synthetic sequences with lengths up to tens of thousands of positions. Then, we tested the performance of Paramixer on the pubic Long Range Arena benchmark data sets. In the third group, we built a character-level document classification task to evaluate if Paramixer can handle real-world long text sequences with tens of thousands of tokens on average. Finally, we showcase if Paramixer performs well in modeling long genome sequences. We ran all experiments on a Linux machine with 3×NVIDIA Tesla V100 32GB, Intel Xeon Gold 6240 CPU @ 2.60GHz processors, with 754GB of system memory.

### 4.1. Synthetic Scalability Test

In this section, we examine the scalability of Paramixer and compare its performance with several competitors. We used two synthetic data sets composed of long sequences for supervised learning tasks. An experimental setup was inspired by [13], where similar synthetic sequences appeared for scalability tests. The details of both tasks are given below:

- *Adding Problem*. This is a sequence regression task. Each element of an input sequence is a pair of numbers $(a_i, b_i)$, where $a_i \sim U(-1, 1)$, $b_i \in \{0, 1\}$, $i = 1, \ldots, N$. We generated signals at two randomly selected positions $t_1$ and $t_2$ such that $b_{t_1} = b_{t_2} = 1$ and $b_i = 0$ elsewhere. The learning target is $y = 0.5 + \frac{a_{t_1} + a_{t_2}}{4}$. For example, an input sequence $[(0.5, 1), (-0.2, 0), (0.2, 1), (-0.8, 0), (0.6, 1)]$ will have the learning target $y = 0.825$. Unlike [13], we did not restrict the $t_1$ and $t_2$ choice and made the task more challenging. That is, the relevant signals can appear either locally or at a great distance from each other. In evaluation, a network prediction $\hat{y}$ is considered correct if $|y - \hat{y}| < 0.04$.

- *Temporal Order*. This is a sequence classification task. Each sequence consists of randomly chosen symbols from the alphabet $\{a, b, c, d, X, Y\}$, where the first four are noise symbols. Each sequence has two signal

symbols, either $X$ or $Y$, which appear at two arbitrary positions. The four target classes correspond to the ordered combinations of the signal symbols $(X, X)$, $(X, Y)$, $(Y, X)$, and $(Y, Y)$. For example, an input sequence $[a, d, Y, c, b, a, Y, c, d]$ should be classified as Class 3.

We generated data of different sequence lengths for each problem: from $N = 128$ to $N = 2^{15}$, progressively increasing the length by the factor of two. For each sequence length, a model can access 100 000 training sequences and 5 000 testing instances for evaluation.

We compared Paramixer with a group of popular methods based on scaled dot-product attention (referred as X-formers), including Linformer [29], Performer [8], Reformer [17] and Nyströmformer, which all [31] have claimed to be scalable. For completeness, we also included the original Transformer [28]. We used the open-source PyTorch implementations[2] of these models.

We fine-tuned the main hyperparameters in a standard cross-validation manner for Paramixer and X-formers, including the number of layers and heads, dimensionality of the token embedding, and query/key/value dimensions. For the Temporal Order problem, we directly fed the data instances to the embedding layers. For the Adding problem, the input data was only two-dimensional, and one of them was real-valued. Directly using such a low-dimensional embedding space would limit the expressive power. So we added a linear layer to augment the dimensionality to allow sufficient freedom for the scaled dot-products in the X-former architectures. All the models were optimized using the Adam optimizer [16] with the learning rate of 0.001 using a batch size of 40.

The results are shown in Figure 3. For the Adding problem, we see that all models obtain 100% accuracy for $N \leq 256$. The X-former models become weak or ineffective when scaling to longer sequences, while Paramixer still performs well. Linformer and Reformer get an error rate of 33.38% and 30.20% for $N = 1024$, and become invalid on $N = 2048$. Transformer starts to lose some performance (99.94%) on $N = 1024$, and becomes not working when $N = 2048$. Performer survives when $N \leq 2048$, however, gets an error rate of 29.24% for $N = 4096$ and becomes as bad as random guessing for $N \geq 8192$. Nyströmformer achieves 100% accuracy for $N = 4096$ and fails on $N \geq 8192$. Instead, Paramixer reaches 100% accuracy for all the tested lengths.

For the Temporal Order problem, the performance of all the models is 100% or nearly 100% when $N \leq 256$, and starts to drop for $N \geq 512$. When $N = 2048$, the results of Performer (99.52%) and Nyströmformer (100%) are still solid while Transformer, Linformer, and Reformer have an

[2]available at https://github.com/lucidrains

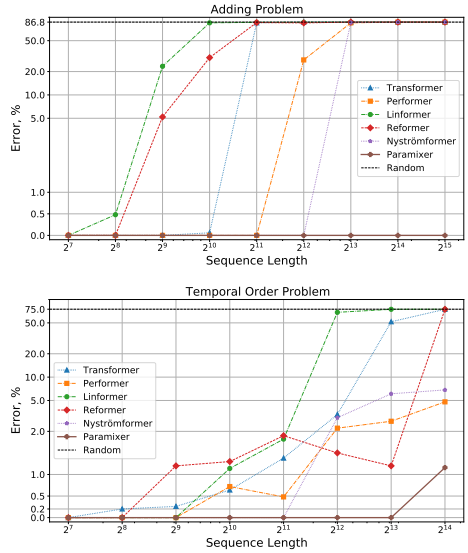



Figure 3. Error percentage of Paramixer and the X-formers for both the Adding problem and the Temporal Order problem with increasing sequence lengths.

error rate around 1.4%. When increasing $N$ to 4,096 and 8,192, Linformer and Transformer fail in succession. When $N = 16384$, Reformer does not work while Performer and Nyströmformer give a weaker performance with error rates, 7.6% and 7.9%, respectively. On the contrary, Paramixer achieves 100% accuracy for $N \leq 8192$, and 98.84% accuracy for $N = 16384$.

In summary, Paramixer has better scalability than the attention neural networks based on scaled dot-products. When scaled to tens of thousands, Paramixer still gives very high prediction accuracy. The results suggest we can evaluate our model on more real-world tasks with very long sequences.

### 4.2. Long Range Arena Public Bechmark

Next, we evaluate the performance of Paramixer on Long Range Arena (LRA), a publicly available benchmark for modeling long sequential data [26]. The details of the tasks are the following:

- *ListOps*. ListOps is a classification task designed for measuring the ability of models to parse hierarchically constructed data [21]. Each sequence is composed of operators, digits, and left or right brackets. The brack-

Table 1. Classification accuracy by Paramixer and X-formers on the four LRA tasks. Methods that are absent in the corresponding paper are signed by a dash ("-"). For Paramixer, we present the mean ($\mu$) and standard deviation ($\sigma$) across multiple runs in the $\mu \pm \sigma$ format.

| Model | ListOps $N = 2000$ | Text $N = 4000$ | Image $N = 1024$ | Pathfinder $N = 1024$ |
|---|---|---|---|---|
| Transformer [26] | 36.37 | 64.27 | 42.44 | 71.40 |
| Transformer [35] | 37.13 | 65.35 | - | - |
| Transformer [31] | 37.10 | 65.02 | 38.20 | 74.16 |
| Sparse Transformer [26] | 17.07 | 63.58 | 44.24 | 71.71 |
| Longformer [26] | 35.63 | 62.58 | 42.22 | 69.71 |
| Linformer [26] | 37.70 | 53.94 | 38.56 | 76.34 |
| Linformer [35] | 37.38 | 56.12 | - | - |
| Linformer [31] | 37.25 | 55.91 | 37.84 | 67.60 |
| Reformer [26] | 37.27 | 56.10 | 38.07 | 68.50 |
| Reformer [35] | 36.44 | 64.88 | - | - |
| Reformer [31] | 19.05 | 64.88 | 43.29 | 69.36 |
| Performer [26] | 18.01 | 65.40 | 42.77 | 77.05 |
| Performer [35] | 32.78 | 65.21 | - | - |
| Performer [31] | 18.80 | 63.81 | 37.07 | 69.87 |
| BigBird [26] | 36.06 | 64.02 | 40.83 | 74.87 |
| Linear Transformer [26] | 16.13 | 65.90 | 42.34 | 75.30 |
| Transformer-LS [35] | 38.36 | 68.40 | - | - |
| RFA-Gaussian [22] | 36.80 | 66.00 | - | - |
| Nyströmformer [35] | 37.34 | 65.75 | - | - |
| Nyströmformer [31] | 37.15 | 65.52 | 41.58 | 70.94 |
| Paramixer (CHORD) | **39.57**$\pm$0.32 | 83.12$\pm$0.33 | 45.01$\pm$0.21 | **80.49**$\pm$0.13 |
| Paramixer (CDIL) | 37.78$\pm$0.28 | **83.32**$\pm$0.19 | **46.58**$\pm$0.05 | 67.13$\pm$0.42 |

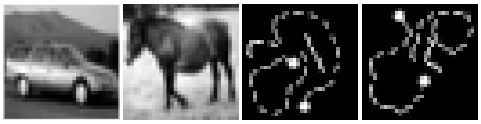



Figure 4. Example data from the Image Classification (left two) and Pathfinder tasks (right two).

ets define lists of items. Each operator in a sequence takes the items in a list as input and returns a digit.

- *Text Classification*. We use the IMDb Review dataset [20] which requires the model to classify each review as positive or negative. The task uses a character-level representation for each sequence, which makes the tasks more challenging than the word-level version. We truncated or padded every sequence to a fixed length ($N = 4k$).

- *Image Classification*. This task is to classify images into one of ten classes. Each image is flattened to form a sequence of length 1024. Unlike conventional computer vision, the task requires the predictors to treat the grayscale levels (0-255) as categorical values. That is, each image becomes a sequence of symbols with an alphabet size of 256. Two example matrices are shown in Figure 4.

- *Pathfinder*. This task is motivated by cognitive psychology [19], and constructed using synthetic images. Each image (size $32 \times 32$) contains two highlighted endpoints and some path-like patterns. The models need to classify whether there is a path consisting of dashes between two highlighted points. Similar to the Image Classification task, the predictors must flatten the image to a sequence of symbols with length 1024. Two example matrices are shown in Figure 4.

For a fair comparison, we followed the experiment settings in the original paper [26] and evaluated multi-block ParamixerNet in the above tasks. We constructed Paramixer in variable blocks and used cross-validation to report the model with the best hyperparameters. We ran the model four times with a different random seed for each task.

Table 2. Test accuracies on long document classification.

| Model | $N = 16k$ | $N = 32k$ |
|---|---|---|
| Transformer | 25.62 | 25.62 |
| Linformer | 64.69 | 65.36 |
| Performer | 25.62 | 25.20 |
| Reformer | 25.04 | 25.04 |
| Transformer-LS | 70.25 | 60.93 |
| Nyströmformer | 71.70 | 67.69 |
| Paramixer | **83.89** | **84.55** |

We compared Paramixer with many X-former architectures in prediction accuracies. If a method has different implementations, we quote all the alternatives and their results. The results show that Paramixer beats all the other self-attention-based transformers on all the tasks, having the best classification accuracy. Such stable cross-task wins suggest that forming an attention matrix with parameterizing mixing links works better than those based on scaled dot-products.

Remarkably, Paramixer has achieved state-of-the-art performance on Text and Pathfinder tasks. For Text Classification, ParamixerNet achieves $83.32\%$, which is $14.92$ p.p. higher than the best Transformer variant, Transformer-LS ($68.4\%$). Our method also wins with the accuracy of $80.49\%$ on Pathfinder, where it gains about $5$ percentage points higher than the runner-up model. The compelling improvement brought by ParamixerNet is a probable cause of the high-rank nature of Natural Language [32].

For Paramixer, we reported two variants for this task, using the CHORD and CDIL protocols independently. As shown in Table 1, CHORD wins on ListOps and Pathfinder, while CDIL gains the best results on Text and Image. However, CHORD still has a competitive accuracy on Text (83.12%) and Image (45.01%), which demonstrates that Paramixer has more stable performance using the CHORD protocol. Consequently, we used CHORD as a default protocol for the following experiments.

### 4.3. Long Document Classification

The goal of this subsection is to study the benefits of modeling long sequences in NLP tasks. The character-level text classification task from the LRA benchmark is not long enough to conclude this pattern, so we seek for Longer Document Data with tens of thousands of tokens.

Long-document-dataset is a publicly available dataset. It contains a collection of academic papers, which are parsed using the arXiv sanity preserver program and published on Github [12]. There are eleven diverse research areas a paper may belong to. Similar to the source article, we used only four classes of documents: cs.AI, cs.NE, math.AC, math.GR, having a final set of 11 956 documents in total. We used 70% of the data for training, 20% for validation, and 10% for the test. Unlike the original study [12], we transformed each article into a sequence of characters and finally got a challenging task with an average training sequence length of 52 112.

We truncate every sequence to a fixed length. Since NLP task is known to benefit from using longer sequence length, we tested Paramixer on sequence lengths of 16 384 and 32 768 to see if Paramixer can be in favor of using longer context. We studied other transformer-like variants as well.

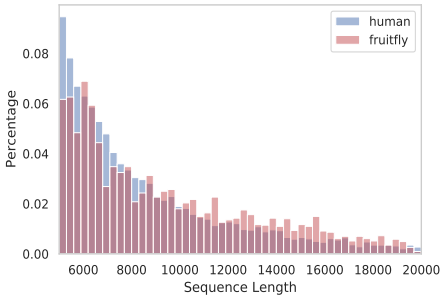
As shown in Table 2, Paramixer outperforms the follow-up transformer-based competitors by 12.19 and 16.86 percentage points on two sequence lengths, respectively. Paramixer achieves higher accuracy when using 32k tokens, which signals a better generalization from using more extended context. Among all the competitors, only Linformer, Transformer-LS, and Nyströmformer can handle the sequences of this length to a certain degree. Nevertheless, the gap in performance between sequence lengths 32k and 16k is severe, suggesting that the low-rank factorization weakens the prediction power on high-rank NLP tasks.

### 4.4. Genome Classification

Inspired by a recent rise in using deep learning models in biological applications, such as Chromatin-profile prediction [33] and genome analysis [2]. However, directly processing genome and protein sequences with tens of thousands of positions are problematic. The standard preprocessing pipeline includes chunking a long sequence into many fragments, which are modeled independently. This approach is associated with inevitable information loss caused by the long-distant nature of interaction processes in the DNA sequences [11]. We tested our design on this type of data. By design, Paramixer can treat a full sequence as an input without the need for its preliminary segmentation, allowing it to capture highly non-local effects.

We built two data sets for this task. The first group of DNA sequences were downloaded from NONCODEv6 [34][3]. We used human and fruitfly DNA sequences to construct a binary-classification task. We filtered out the sequences shorter than 5k and thus got 9 536 human DNA sequences and 4401 fruitfly DNA sequences with mean sequence lengths 10 586 and 9 793, respectively. We name the data set HFDNA. As shown in Figure 5, the two species in HFDNA have similar sequence length distributions, which means they can not be directly distinguished based on a sequence length threshold. We split the data set to 60% training, 20% validation, and 20% test. Each sequence is either padded or truncated to a fixed length of 16 384.

[3] http://www.noncode.org/download.php



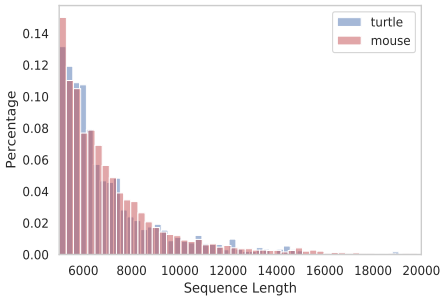




Figure 5. Sequence length distributions for HFDNA (top) and MTcDNA (bottom). X-axis is the range of genome sequence length, and y-axis is the percentage of total instance numbers for each bin.

The other data set, namely MTcDNA (mouse and turtles' cDNA), was built following the same preprocessing/split strategy. The original cDNA sequences were downloaded from Ensembl genome browser[4] [1, 14]. The task is to predict a binary class, given either mouse or turtles cDNA sequence. The mouse class presented in 12 300 cDNA sequences with a mean sequence length of 7 235, including two sub-species: *Mus musculus* and *Mus spretus*. The turtle class contains 4 193 cDNA sequences of *Chelonoidis abingdonii* and *Gopherus agassizii* with a mean sequence length of 7 072. The percentage histograms of sequence lengths are shown in Figure 5. It is clearly seen that the two species have a big overlap in terms of sequence length, which makes it hard to discriminate between them only by length.

We compared ParamixerNet with several transformer-based models on this task. Because both data sets are imbalanced, so we report ROC AUC values for this experiment.

[4]http://www.ensembl.org/info/data/ftp/index.html

Table 3. The performance of Paramixer in comparison to Xformers on the Genomic classification task. The metric is Area Under the Receiver Operating Characteristic Curve (ROC AUC) $\times 100\%$

| Model | HFDNA | MTcDNA |
|---|---|---|
| Transformer | 94.32 | 79.38 |
| Linformer | 91.65 | 77.06 |
| Performer | 93.46 | 78.92 |
| Reformer | 92.32 | 74.94 |
| Transformer-LS | 93.19 | 75.81 |
| Nyströmformer | 92.26 | 71.98 |
| Paramixer | **100.00** | **84.86** |

As shown in Table 3, Paramixer achieves 100% ROC AUC on HFDNA, outperforming the runner-up result from Transformer (94.32%). Notably, our design holds the best score at 84.86% ROC AUC on MTcDNA, which is higher than Transformer by 5.48 percentage points. The experimental results show that Paramixer can successfully handle long DNA sequences and has the potential to assist in the biological applications that require modeling long-distant gene interactions.

## 5. Conclusion

We have proposed a scalable and effective building block called Paramixer for attention neural networks. Our method replaced the dot-products and softmax with parameterization of the mixing links in full-rank sparse factors of the attention matrix and thus got rid of the low-rank bottleneck in most existing attention models. Our method has complexity as low as $O(N \log N)$ and can efficiently deal with sequences up to tens of thousands. Besides scalability, Paramixer has also demonstrated strong performance in terms of accuracy. Neural networks, by stacking the proposed Paramixer blocks, have defeated Transformer and many of its variants in a variety of tasks, including synthetic data inference, the public Long Rang Arena benchmark, and classification of very long text documents and genome sequences.

In the future, we could study other applications beyond classification, for example, gene expression prediction from sequence and pretraining with unsupervised data. The basic Paramixer block could be extended using other existing attention techniques such as multiple heads and relative positional encoding. Later, in addition to the CHORD and CDIL protocols, we could consider the other predefined protocols or even adaptively learned protocols for the sparse structure.

**Acknowledgement:** This work was financed by The Research Council of Norway, grant number 287284. We acknowledge for using the IDUN computing cluster [23].

## References


[1] Ensembl Release 104 (May 2021). http://www.ensembl.org/index.html. Accessed: 2021-5. 8

[2] Effective gene expression prediction from sequence by integrating long-range interactions. *Nature Methods*, 18:1196–1203, 2021. 1, 2, 7

[3] Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. Etc: Encoding long and structured inputs in transformers. *arXiv preprint arXiv:2004.08483*, 2020. 1, 2

[4] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020. 1, 2

[5] Srinadh Bhojanapalli, Chulhee Yun, Ankit Singh Rawat, Sashank J. Reddi, and Sanjiv Kumar. Low-rank bottleneck in multi-head attention models. In *International Conference on Machine Learning*, 2020. 1, 2

[6] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. 1

[7] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019. 1, 2

[8] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers, 2020. 2, 5

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 1

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 1

[11] Molly Gasperini, Jacob M Tome, and Jay Shendure. Towards a comprehensive catalogue of validated and target-linked human enhancers. *Nature Reviews Genetics*, 21(5):292–310, 2020. 7

[12] Jun He, Liqun Wang, Liu Liu, Jiao Feng, and Hao Wu. Long document classification from local word glimpses via recurrent attention learning. *IEEE Access*, 7:40707–40718, 2019. 7

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 4

[14] Kevin L Howe, Premanand Achuthan, James Allen, Jamie Allen, Jorge Alvarez-Jarreta, M Ridwan Amode, Irina M Armean, Andrey G Azov, Ruth Bennett, Jyothish Bhai, et al. Ensembl 2021. *Nucleic acids research*, 49(D1):D884–D891, 2021. 8

[15] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020. 2

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. 4, 5

[17] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020. 5

[18] Colin Lea, René Vidal, Austin Reiter, and Gregory D. Hager. Temporal convolutional networks: A unified approach to action segmentation. In *ECCV 2016 Workshops*, pages 47–54, 2016. 3

[19] Drew Linsley, Junkyung Kim, Vijay Veerabadran, Charlie Windolf, and Thomas Serre. Learning long-range spatial dependencies with horizontal gated recurrent units. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 152–164, 2018. 6

[20] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011. 6

[21] Nikita Nangia and Samuel R Bowman. Listops: A diagnostic dataset for latent tree learning. *arXiv preprint arXiv:1804.06028*, 2018. 5

[22] Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A Smith, and Lingpeng Kong. Random feature attention. *arXiv preprint arXiv:2103.02143*, 2021. 2, 6

[23] Magnus Själander, Magnus Jahre, Gunnar Tufte, and Nico Reissmann. EPIC: An energy-efficient, high-performance GPGPU computing research infrastructure, 2019. 8

[24] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001. 3

[25] Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Xuyi Chen, Han Zhang, Xin Tian, Danxiang Zhu, Hao Tian, and Hua Wu. Ernie: Enhanced representation through knowledge integration. *arXiv preprint arXiv:1904.09223*, 2019. 1

[26] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020. 5, 6

[27] Yi Tay, Aston Zhang, Luu Anh Tuan, Jinfeng Rao, Shuai Zhang, Shuohang Wang, Jie Fu, and Siu Cheung Hui. Lightweight and efficient neural natural language processing with quaternion networks. *arXiv preprint arXiv:1906.04393*, 2019. 1, 2

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. 1, 2, 5

[29] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity, 2020. 1, 5

[30] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020. 2

[31] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nystr\" omformer: A nystr\" om-based algorithm for approximating self-attention. *arXiv preprint arXiv:2102.03902*, 2021. 2, 5, 6

[32] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. Breaking the softmax bottleneck: A high-rank rnn language model. *arXiv preprint arXiv:1711.03953*, 2017. 7

[33] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. In *NeurIPS*, 2020. 1, 2, 7

[34] Lianhe Zhao, Jiajia Wang, Yanyan Li, Tingrui Song, Yang Wu, Shuangsang Fang, Dechao Bu, Hui Li, Liang Sun, Dong Pei, et al. Noncodev6: an updated database dedicated to long non-coding rna annotation in both animals and plants. *Nucleic Acids Research*, 49(D1):D165–D171, 2021. 7

[35] Chen Zhu, Wei Ping, Chaowei Xiao, Mohammad Shoeybi, Tom Goldstein, Anima Anandkumar, and Bryan Catanzaro. Long-short transformer: Efficient transformers for language and vision. *arXiv preprint arXiv:2107.02192*, 2021. 6

# Paramixer: Parameterizing Mixing Links in Sparse Factors Works Better than Dot-Product Self-Attention (Supplemental Document)

Tong Yu * Ruslan Khalitov * Lei Cheng Zhirong Yang †
Norwegian University of Science and Technology

## 1. Synthetic Data Experiments

For both problems in the scalability test, we generated sequences using the setup described in the main paper. We ran the experiments on sequences with variable lengths: from 128 to 32k. The longer sequences, the more complex the retrieval process. There is a slight difference in the pre-processing part. For the Adding problem, the input data was only two-dimensional. To avoid using such a low-dimensional embedding space, we augmented the dimensionality with an additional linear layer to assure sufficient freedom for dot-product attention architectures. The training configuration and hyperparameters are the same for both the Adding problem and the Temporal Order problem. Their summary is in Table 1

## 2. Long Range Arena

The data set for the LRA benchmark is publicly available. The information about data and the download link can be found in the official GitHub repository: https://github.com/google-research/long-range-arena.

- **ListOps** The raw data for this problem is organized as three separate files `basic_train.tsv`, `basic_test.tsv`, `basic_val.tsv` for training, testing, and validation data, respectively. The split is fixed. In addition to the tokens described in the main paper, each sequence has "(" and ")" symbols, which should be removed. To equalize the lengths of the sequences, we used the built-in PyTorch padding functional. After the sequences are prepared, the embedding layer processes each unique value, thus mapping elements to the embedding space. The rest of the training process is straightforward.

- **Text Classification** We downloaded IMDB data set using the `tensorflow-dataset` package, and got 25000 instances for training and another 25000 for testing. We went through the whole corpus and extracted the character vocabulary. Then we mapped each sequence to a vector of indices using this vocabulary. Finally, we truncated or padded each sequence to a fixed length of 4096. For every review, we add ["CLS"] token to each sequence and use the embedding of ["CLS"] token for final classification. We used three blocks Paramixer for this task.

- **Image Classification** CIFAR10 is a well-known dataset, which can be downloaded from the `torchvision` package. The train/test splitting is fixed. To make images grayscaled, we used standard transformation `transforms|grayscale` from the same package. An image is flattened to a sequence of length 1024. Then each element is mapped to a dictionary of size 256 (all possible intensity values) and given to the embedding layer.

- **Pathfinder** The problem data consists of two types of files: images and metafiles. Metafiles store information about all the images and their corresponding labels (positive or negative). There are three classes of images: `curv_baseline` (easy), `curv_contour_length_9` (medium), `curv_contour_length_14` (hard). An image class corresponds to the distance between its endpoints (curve length), thus positively correlates with the difficulty level. The exact data split is not provided. To separate the data into three parts, we iterated over all metafiles from the catalogs and constructed the training/val/test (90%/5%/5%) sets such that all three types of images are present equally. The rest of the processing is similar to the Image Classification task.

## 3. Long Document Classification

The task is a four-class classification problem. The class of a paper is defined by its arxiv categorization, namely, cs.AI, cs.NE, math.AC, and math.GR. Each class in the data set is almost equally presented, with a slight class imbalance: 2995, 3012, 2885, and 3065 documents, respectively.

*Equal contribution.
†Corresponding author. zhirong.yang@ntnu.no

Table 1. Hyperparameters details for every task. $N$, $B$, $V$, $E$, $H$, lr refer to max sequence length, batch size, vocabulary size, embedding size, hidden states size, and learning rate, respectively. The vocabulary size includes padding index and ["CLS"].

| Task | $N$ | Protocol | n_links | lr | $B$ | $V$ | $E$ | $H$ | pos_embed | Pooling Type |
|---|---|---|---|---|---|---|---|---|---|---|
| Adding | 32768 | CHORD | 15 | 0.001 | 40 | - | 32 | 32 | True | FLAT |
| Temporal Order | 16384 | CHORD | 14 | 0.001 | 40 | 6 | 32 | 32 | True | FLAT |
| ListOps | 2000 | CHORD | 12 | 0.001 | 48 | 16 | 32 | 32 | True | FLAT |
| CIFAR10 | 1024 | CDIL | 3 | 0.001 | 64 | 256 | 32 | 32 | True | FLAT |
| Text | 4096 | CDIL | 9 | 0.0001 | 32 | 97 | 32 | 128 | False | CLS |
| Pathfinder | 1024 | CHORD | 11 | 0.001 | 64 | 256 | 32 | 32 | True | FLAT |
| Long Document | 16384 | CHORD | 15 | 0.0001 | 16 | 4290 | 100 | 128 | False | CLS |
| Long Document | 32768 | CHORD | 16 | 0.0001 | 16 | 4290 | 100 | 128 | False | CLS |
| Genome Classification | 16384 | CHORD | 15 | 0.0001 | 16 | 5 | 32 | 128 | True | FLAT |

To transform the raw articles into sequences, we first went through the whole corpus and extracted the character vocabulary. Then we mapped each character sequence to a vector of indices using this vocabulary. We fine-tuned Paramixer and X-formers to get the best results. The hyperparameters of Paramixer were selected using a similar process. Final configurations are shown in Table 1. For every document, we add ["CLS"] token to each sequence and use the result embedding of ["CLS"] token for the final classification.

## 4. Genome Classification

When building MTcDNA we downloaded cDNA sequences of Chelonoidis abingdonii and Gopherus agassizii, and merged them as a turtle data set. Following the same strategy, we built the mouse data set using Mus musculus and Mus spretus. More details can be found in the main paper. For the HFDNA classification task, one Paramixer block is enough to get 100% accuracy. However, ParamixerNn with two blocks result in the best test accuracy for MTcDNA. The selected hyperparameters are listed in Table 1.

## 5. Proof of Proposition 3.1 in the main paper

**Definition 1.** *An $N \times N$ circulant matrix $C$ takes the form*

$$C = \begin{bmatrix} c_0 & c_{N-1} & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{N-1} & \cdots & c_2 \\ \vdots & c_1 & c_0 & \ddots & \vdots \\ c_{N-2} & \cdots & \ddots & \ddots & c_{N-1} \\ c_{N-1} & c_{N-2} & \ddots & c_1 & c_0 \end{bmatrix}$$

**Definition 2.** *The polynomial*

$$f(x) = c_0 + c_1 x + \cdots + c_{N-1} x^{N-1}$$

*is called the associated polynomial of circulant matrix $C$.*

We have the following theorem in the literature [1]:

**Theorem 1.** *The rank of a circulant matrix $C$ is equal to $N - d$, where $d$ is the degree of the polynomial GCD$(f(x), x^{N-1})$.*

Now we can prove Preposition 3.1 for the CHORD protocol. The proof for CDIL follows similarly.

*Proof.* The associated polynomial of $W^{(m)}$ is

$$f(x) = \sum_{k=0}^{\log_2 N - 1} x^k$$

Because GCD$(f(x), x^N - 1) = 1 = x^0$, the rank of $W^{(m)}$ is $N$. □

## References

[1] A. W. Ingleton. The rank of circulant matrices. *Journal of the London Mathematical Society*, s1-31(4):445–460, 1956. 2

# Publication C - Classification of long sequential data using circular dilated convolutional neural networks

Neurocomputing 518 (2023) 50–59

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

# Classification of long sequential data using circular dilated convolutional neural networks

Lei Cheng [a], Ruslan Khalitov [a], Tong Yu [a], Jing Zhang [b], Zhirong Yang [a,*]

[a] Norwegian University of Science and Technology, Norway
[b] Beijing Jiaotong University, China

ARTICLE INFO

*Article history:*
Received 21 December 2021
Revised 6 September 2022
Accepted 24 October 2022
Available online 31 October 2022
Communicated by Zidong Wang

*Keywords:*
Classification
Sequential data
Convolutional neural networks

ABSTRACT

Classification of long sequential data is an important Machine Learning task and appears in many application scenarios. Recurrent Neural Networks, Transformers, and Convolutional Neural Networks are three major techniques for learning from sequential data. Among these methods, Temporal Convolutional Networks (TCNs) which are scalable to very long sequences have achieved remarkable progress in time series regression. However, the performance of TCNs for sequence classification is not satisfactory because they use a skewed connection protocol and output classes at the last position. Such asymmetry restricts their performance for classification which depends on the whole sequence. In this work, we propose a symmetric multi-scale architecture called Circular Dilated Convolutional Neural Network (CDIL-CNN), where every position has an equal chance to receive information from other positions at the previous layers. Our model gives classification logits in all positions, and we can apply a simple ensemble learning to achieve a better decision. We have tested CDIL-CNN on various long sequential datasets. The experimental results show that our method has superior performance over many state-of-the-art approaches. The model and experiments are available at (https://github.com/LeiCheng-no/CDIL-CNN).



## 1. Introduction

Sequence classification is the task of predicting class labels for sequences. It is of central importance in many applications, such as document classification, genomic analysis, and health informatics. For example, classifying documents into different topic categories is a challenge for library science, especially for modern digital libraries [1]. Genomic classification help researchers to further understand some diseases [2]. Classifying ECG time series tells if someone is a healthy person or a patient with heart disease [3].

Machine Learning, especially Deep Learning, becomes widely used in end-to-end sequence classification, where a single model learns all steps between the initial inputs and the final outputs. Recurrent Neural Networks (RNNs), Transformers, and Convolutional Neural Networks (CNNs) are three primary techniques for analyzing sequential data.

RNNs use their internal states to process the sequence step by step. Despite success for short sequences, traditional RNNs cannot scale to very long sequences [4]. One reason is that they are challenging to train due to exploding or vanishing gradient problems [5]. In addition, the prediction of each timestep must wait for all its predecessors to complete, which makes RNNs difficult to parallelize. Transformers are a family of models relying on self-attention mechanism [6,7]. They have quadratic time and memory complexities to the input sequence length because they compute pairwise dot-products. Comprehensive approximations are required to reduce the cost [8].

In contrast, CNNs are able to handle very long sequences. A convolutional layer uses sparse connections and no recurrent nodes. Therefore, CNNs are easier to train and parallelize. In addition, dilated convolutions can exponentially enlarge the receptive fields, allowing CNNs to use fewer layers to capture long-term dependencies. For example, Temporal Convolutional Networks (TCNs) recently provide remarkable performance on sequence regression tasks [9]. However, the performance of TCNs for classification tasks is not satisfactory. TCNs use causal convolutions which implement a skewed connection protocol. The asymmetric design causes a tendency to focus on the latter part of a sequence.

In this paper, we propose a novel convolutional architecture named Circular Dilated Convolutional Neural Network (CDIL-CNN), which can scale to very long sequences and have superior performance on various classification tasks. Unlike TCNs, we use symmetric convolutions to mix information, and thus every posi-

* Corresponding author.
*E-mail address:* zhirong.yang@ntnu.no (Z. Yang).

https://doi.org/10.1016/j.neucom.2022.10.054
0925-2312/

tion can receive both earlier and later information from previous layers in a circular manner. Unlike traditional pyramid-like CNN architecture, every position of the last convolutional layer in our design has an equal chance to receive all information from the whole sequence and gives its classification logits. Then a simple average ensemble learning helps our model achieve better accuracy.

We have tested our model on extensive sequence classification tasks, including synthetic data, images, texts, and audio series. Experimental results show that CDIL-CNN outperforms several state-of-the-art models. Our method can accurately and robustly classify across tasks with both short-term and long-term dependencies for very long sequences.

The remaining of the paper is organized as follows. We review some popular models for sequential data and their limitations in Section 2. In Section 3, we present our model, CDIL-CNN, including its connection protocol and network architecture. Experimental tasks and results are provided in Section 4, and we discover that the simple convolutional network has superior performance over other models in various scenarios. Finally, we conclude the paper in Section 5.

## 2. Related Works

A sequence $x$ of length $N$ is a list of elements $[x_1, x_2, \cdots, x_N]$, where $x_t \in \mathbb{R}^D (1 \leqslant t \leqslant N)$ is the $D$-dimensional element at the $t$-th position. Given a training set $\{x^{(i)}, y^{(i)}\}_{i=1}^{I}$ with $I$ sequences and their class labels, sequence classification uses the training set to fit a model $f : \mathbb{R}^{N\times D} \mapsto \mathbb{C}$, where $\mathbb{C}$ is the space of class labels. The fitted model can then be used to classify newly coming sequences.

Many deep neural networks have been proposed for various sequence classification tasks. RNNs, Transformers, and CNNs are three significant branches for learning from sequential data.

RNNs read and process inputs sequentially. At each timestep, an RNN takes the current sequence element and the hidden state as the input and outputs the next hidden state. The hidden state at a timestep is expected to act as the representation of all its earlier inputs. Because the prediction of each timestep must wait for all its predecessors to complete, the sequential process is difficult to parallelize, which makes RNNs hard to handle very long sequences. Moreover, basic RNNs suffer from vanishing and exploding gradient problems, making model training very difficult for long sequences [5]. Gated RNNs, such as Long Short-Term Memory (LSTM) [10] and Gated Recurrent Unit (GRU) [11], have been proposed to relieve the gradient problems. They have many additional gates to regulate the flow of information. The gated RNNs are used in many sequence classification tasks, such as ECG arrhythmia [12] and text [13,14]. However, they can process only short sequences (about 500–1000 timesteps) [4].

Transformers, a family of models based on attention mechanism, quantify the interdependence within the sequence elements (self-attention). Originally, attention was used in conjunction with recurrent networks and convolutional networks [15,16]. Later, Transformer, an architecture based solely on attention mechanism, was proposed. The vanilla Transformer computes pairwise dot-products between all sequence elements, which leads to a quadratic complexity w.r.t. the sequence length and makes it infeasible to process very long sequences. Approximated attention methods have been proposed to tackle this problem. Sparse Transformer [17], LogSparse Transformer [18], Longformer [19], and Big Bird [20] use sparse attention mechanism. Linformer [21] and Synthesizer [22] apply low-rank projection attention. Performer [23], Linear Transformer [24], and Random Feature Attention [25] rely on kernel approximation. Reformer [26], Routing Transformer [27], and Sinkhorn Transformer [28] follow the paradigm of rearranging sequences. However, their approximation quality is questionable. Later in Section 4, we will show that their performance is inferior for long sequence classification.

CNNs are good at processing data that has a grid-like topology. Two-dimensional CNNs achieve great success in computer vision [29–32], while one-dimensional CNNs are commonly used for sequential data [33–35]. Among these models, TCNs which use causal convolutions with skewed connections attempt to capture the temporal interactions and have been applied to various regression tasks, such as action segmentation and detection [36,37], lip-reading [38,39], and ENSO prediction [40]. The comparison of the convolutional and recurrent architectures shows that a simple TCN outperforms canonical RNNs across a wide range of sequence modeling tasks [9].

## 3. Circular Dilated CNN

Although TCN is suitable for long sequence regression, their performance for classification is not satisfactory. In this paper, we propose a new convolutional model, named CDIL-CNN, to overcome the TCN drawbacks in long sequence classification. More details are described as follows.

### 3.1. Symmetric Dilated Convolutions

Our model uses symmetric convolutions that can receive both earlier and later information from previous layers. Because no information is allowed to be leaked from future to past in regression tasks, TCN uses causal convolutions that implement a skewed connection protocol, meaning that the output at timestep $t$ can only receive information of $t$ and earlier from previous layers. However, classification tasks do not have the restriction because the classification result depends on the whole sequence. Therefore, symmetric convolutions help our model better capture interactions.

Our model also uses increasing dilation sizes with the depth of the network. Dilated convolutions (or atrous convolutions) were originally introduced for dense image prediction, where they helped the model to capture multi-scale information [41–45]. For 1D CNNs, dilated convolutions are generally used to enlarge the receptive fields [33,34,37,9]. Following these works, we increase the dilation sizes exponentially, i.e., $d_l = 2^{l-1}$ where $d_l$ is the dilation size at the $l$-th convolutional layer. The combination of deep networks and exponentially dilated convolutions enables the receptive fields to expand quickly, which makes our model scalable to very long sequences. Our model needs $\lceil \log_2 \frac{N}{2} \rceil$ or $O(\log_2 N)$ layers to achieve full receptive fields for sequence length $N$.

To avoid notional clutter, we start from the $D = 1$ case. Let $[a_1, a_2, \cdots, a_N]$ denotes an 1-dimensional input sequence of the $l$-th convolutional layer. The convolutional output $b_t$ at the $t$-th $(1 \leqslant t \leqslant N)$ position is computed by $b_t = \sum_{k=0}^{K-1} w_k^{(l)} \cdot a_{t+(k-\frac{K-1}{2})\cdot d_l}$, where the kernel size $K$ is usually an odd number[1] and $w^{(l)}$ are the convolution coefficients of the $l$-th layer. See Fig. 1 for an illustration of a 3-layer symmetric dilated convolutions with $K = 3$. It is straightforward to extend the convolution with the bias term and for the $D > 1$ cases.

### 3.2. Circular Mixing

In traditional CNNs, zero-padding is often used for the boundary positions where the subscripts of their convoluted input positions $\left[t + (k - \frac{K_l-1}{2}) \cdot d_l\right]$ are smaller than 1 or larger than $N$. However, this

[1] We used $K = 3$ in all our experiments.

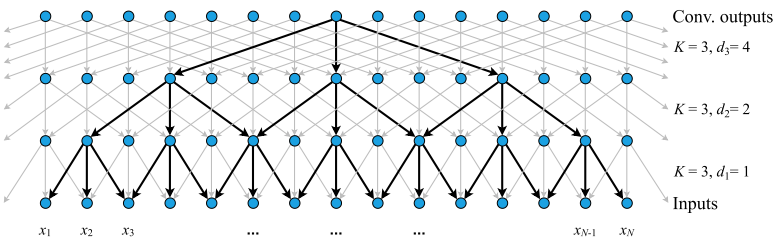



**Fig. 1.** Illustration of symmetric dilated convolutions. Blue nodes represent the sequence elements. Each layer keeps the same length as the input sequence. Symmetric convolutions of kernel size 3 are used in all layers. Dilation sizes are increased exponentially.

can cause boundary effect because signals near the boundaries have to be mixed up with zeros and thus have less chance to be forwarded. The boundary effect creates blind spots and makes CNNs sensitive to absolute positions [46,47]. For example, CNNs with zero-padding can fail to capture the useful patterns if translation exists in the test data but not in the training data (see Section 4.4).

We use a circular protocol because its corresponding circular padding can relieve the boundary effect [46,47]. In our model, a signal on one end is no longer convoluted with zeros but with signals from the other end. Circular padding makes our model more robust to data shift and less sensitive to absolute position information. The circular dilated convolutions are shown in Fig. 2. The convolutional output $b_t$ becomes

$$b_t = \sum_{k=0}^{K-1} w_k^{(l)} \cdot a_{\left[t+\left(k-\frac{K-1}{2}\right)d_l\right] \bmod N} \tag{1}$$

Using circular dilated convolutions, our model can connect boundary positions and learn long-term dependencies even in the first layer, unlike lower layers of traditional CNNs which only focus on local information. In our design, every position of the last convolutional layer has an equal chance to receive all information of the whole input sequence. Therefore, our model can apply a simple average ensemble learning as below.

### 3.3. Ensemble Learning

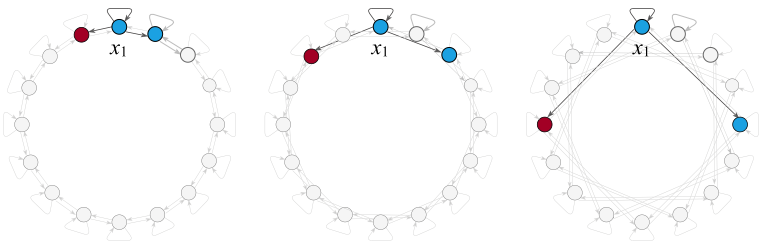
We use a simple average ensemble learning to achieve better performance. RNNs and TCN assume that the last position contains all information of the whole sequence and the class decision depends only on the last position. In our model, every position of the last convolutional layer can receive all information of the whole sequence. A linear module $\mathbb{R}^C \mapsto \mathbb{C}$, where $C$ is the number of convolution channels, is applied on each convolutional output position, and each position gives its preliminary class logits. Then a simple average pooling as ensemble learning aggregates the individual logits. In the implementation, we can perform the average first to speed up the network because the linear module and the average pooling are exchangeable.

Our model also uses residual connections to facilitate the training and to improve the accuracy [48,49]. A residual block contains a skip connection where the inputs are added before the block outputs. A schematic view of our model is depicted in Fig. 3.

## 4. Experiments

We have compared our model with many popular models: Transformer [7], Linformer [21], Performer [23], LSTM [10], GRU [11], TCN [9]. We have also included deformable convolutional networks (Deformable) that learn the adaptive receptive field using additional offsets [50] and convolutional neural networks (CNN) with dilation 1. We used stride 1 in CNN and left out the pooling layers to ablate the effect of dilations in CDIL-CNN. In the supplemental document (Section 4), we also compared CDIL-CNN with ResNet18, a popular convolutional architecture with striding and pooling.

The compared models are tested on various long sequential datasets in three groups of experiments. First, we used a synthetic dataset with increasing sequence lengths to show the scalability of our model. Then, we tested our model on the Long Range Arena (LRA) benchmark suite which contains different dependencies. Finally, we tried three time series classification datasets that con-




(a) $K = 3, d_1 = 1$ (b) $K = 3, d_2 = 2$ (c) $K = 3, d_3 = 4$

**Fig. 2.** Illustrations of circular mixing. (a), (b), and (c) are the first, second, and third convolutional layer, respectively. Red nodes represent convoluted positions from the other end.

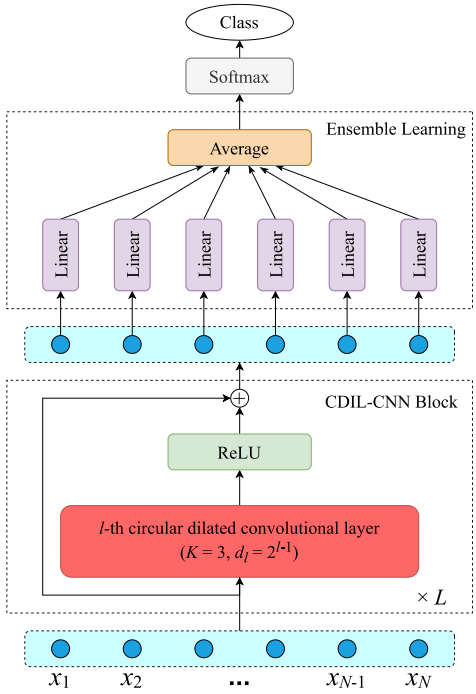



**Fig. 3.** Our neural network architecture for sequence classification. The model comprises $L$ blocks of CDIL-CNN and an average ensemble learning. Each block outputs the same length as the input sequence. After the convolutions and the linear transformation, each position gives its prediction logits, and the average ensemble learning aggregates the logits.

tain important local information and much noise. We used PyTorch[2] to implement our method. All experiments were run on a Linux server with one NVIDIA-Tesla V100 GPU with 32 GB of memory. More details are given in the supplemental document.

### 4.1. Synthetic Task: XOR Problem

The XOR problem is a classical classification problem in artificial neural network research which cannot be solved by a single perceptron [51,52]. We created more challenging XOR tasks with increasing sequence lengths. For each length $N$, a sequence consists of $N$ pairs of numbers, where the first number, called value, is randomly chosen from the interval $[0, 1)$, and the second number is used as a marker. Most markers are 0 except two 1's at randomly selected positions. Let $X_1$ and $X_2$ denote the two values at the 1-marked positions. A sequence belongs to Class 0 if the values belong to the same half interval, i.e., ($X_1 < 0.5$ and $X_2 < 0.5$) or ($X_1 \geqslant 0.5$ and $X_2 \geqslant 0.5$). Otherwise, the sequence is labeled as Class 1. Fig. 4 shows four examples of the XOR problem. We have used $N = 2^n$, where $n = 4, \ldots, 11$. A larger $N$ corresponds to a more challenging task. For each $N$, training, validation, and testing sets respectively have 10000 labeled sequences.

[2] https://pytorch.org/

We have compared our model with several popular approaches (including RNNs, Transformers, and CNNs). All convolutional networks use the $n-1$ layers and 32 channels for a fair comparison.

The results are shown in Fig. 5. Our model performs accurately for all sequence lengths, where CDIL-CNN achieves less than 1% error rate even when $N = 2^{11}$. Transformer and its variants, RNNs, and Deformable achieve comparable error rates for short sequences. However, they turn inaccurate ($\sim 50\%$ accuracy) when the sequences become longer than $N = 128$. TCN and CNN perform even worse, where they respectively have 50% and 20% errors when $N = 32$. The results indicate that CDIL-CNN is more scalable than the other compared methods.

### 4.2. Long Range Arena Benchmark

Long Range Arena is a public benchmark suite for evaluating model quality in long-context scenarios [53]. The suite consists of different data types, such as images and texts. Many Transformers have been evaluated on the suite [25,53–55]. We compared our CDIL-CNN with other models on the following datasets:

- **Image**. This is a 10-class image classification task. The images come from the gray-scale version of CIFAR-10 [56], where pixel intensities (0–255) are treated as categorical values. Two example images and their labels are shown in Fig. 6. Every image is flattened to a sequence of length $N = 1024$. The task requires the model to learn the 2D spatial relations while using the 1D sequences.
- **Pathfinder**. This is a synthetic image task motivated by cognitive psychology [57,58]. The task requires the model to make a binary decision whether two highlighted points are connected by a dashed path. Two example images and their labels are shown in Fig. 6. Similar to the Image task, every pathfinder image is flattened to a sequence of length $N = 1024$ with an alphabet size of 256.
- **Text**. This is a binary sentiment classification task of predicting whether an IMDb movie review is positive or negative [59]. The task considers the character-level sequences which generate longer inputs and make the task more challenging. We use a fixed length $N = 4000$ for every sequence, which is truncated or padded when necessary.
- **Retrieval**. This is a character-level task with the ACL Anthology Network dataset [60]. The task requires the model to process a pair of documents and determine whether they have a common citation. Like the Text task, every document is truncated or padded to the sequence length of 4000, making the total length $N = 8000$ for the pair.

For a fair comparison, we followed the same data preprocessing and training/validation/testing splitting in [53]. We quoted the results of Transformer and its variants from the literature and ran RNNs and CNNs for completeness. We used one layer with a hidden size of 128 for RNNs and 64 channels for CNNs. All experiments were run five times with different random seeds, where means and standard deviations are reported in Table 1. We have used paired $t$-test at the significance level of 0.05 to verify whether CDIL-CNN is significantly different from RNNs or other CNNs.

Our model achieves the best mean accuracies in all tasks and is significantly better than RNNs and other CNNs in 17 out of 20 comparisons. The significant wins over all other methods hold for the Image and Pathfinder tasks. Especially for the Image task, CDIL-CNN achieves substantially higher mean accuracies (20.25% better than the best transformer variant, 20.09% better than the best RNN, 25.87% better than other CNNs). Deformable and CNN get compa-

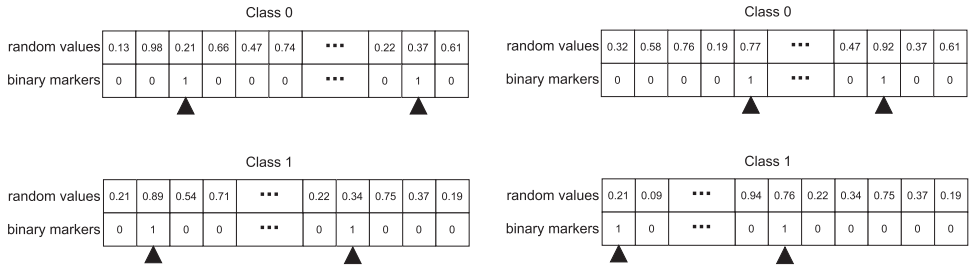



**Fig. 4.** Examples of the XOR problem.

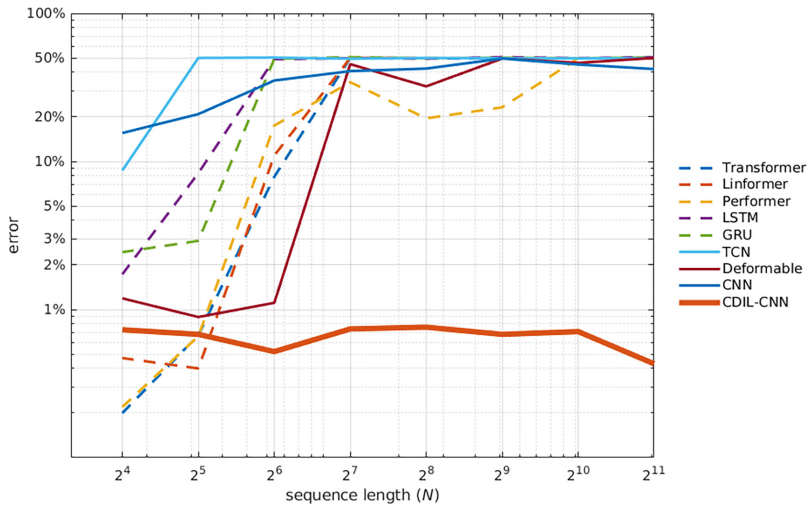



**Fig. 5.** Error rate for the XOR problem with increasing sequence lengths.

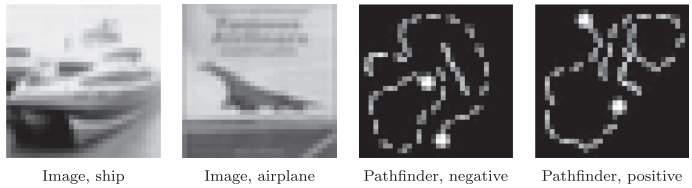



**Fig. 6.** Examples of Image (left two) and Pathfinder (right two).

rable accuracies with CDIL-CNN for Text and Retrieval, probably because the two tasks mainly rely on local patterns.

### 4.3. Time Series

The UEA & UCR Repository[3] consists of various time series classification datasets [61]. Many time series classification problems can be solved by detecting local patterns [62–64]. These tasks require the model to pick out important local information from long sequences which contain much noise. We compared our CDIL-CNN with other popular models on three audio datasets:

- **FruitFlies**. The dataset comes from the same optical sensor which recorded the change in amplitude of an infra-red light as it was occluded by the wings of fruit flies during flight. The dataset contains 17259 training and 17259 testing sequences

[3] http://www.timeseriesclassification.com/

**Table 1**
Classification accuracy (%) of different models on LRA tasks. $N$ is the sequence length. The dash means the result is absent in the reference paper. Means and standard deviations are computed across 5 runs. • denotes significant difference, and ◦ denotes insignificance.

| Model | Image $N$=1024 | Pathfinder $N$=1024 | Text $N = 4000$ | Retrieval $N = 8000$ |
|---|---|---|---|---|
| Transformer [53] | 42.44 | 71.40 | 64.27 | 57.46 |
| Transformer [55] | 38.20 | 74.16 | 65.02 | 79.35 |
| Transformer [54] | - | - | 65.35 | 82.30 |
| Local Attention [53] | 41.46 | 66.63 | 52.98 | 53.39 |
| Sparse Transformer [53] | 44.24 | 71.71 | 63.58 | 59.59 |
| Longformer [53] | 42.22 | 69.71 | 62.85 | 56.89 |
| Linformer [53] | 38.56 | 76.34 | 53.94 | 52.27 |
| Linformer [55] | 37.84 | 67.60 | 55.91 | 79.37 |
| Linformer [54] | - | - | 56.12 | 79.37 |
| Reformer [53] | 38.07 | 68.50 | 56.10 | 53.40 |
| Reformer [55] | 43.29 | 69.36 | 64.88 | 78.64 |
| Reformer [54] | - | - | 64.88 | 78.64 |
| Sinkhorn Transformer [53] | 41.23 | 67.45 | 61.20 | 53.83 |
| Synthesizer [53] | 41.61 | 69.45 | 61.68 | 54.67 |
| BigBird [53] | 40.83 | 74.87 | 64.02 | 59.29 |
| Linear Transformer [53] | 42.34 | 75.30 | 65.90 | 53.09 |
| Performer [53] | 42.77 | 77.05 | 65.40 | 53.82 |
| Performer [55] | 37.07 | 69.87 | 63.81 | 78.62 |
| Performer [54] | - | - | 65.21 | 81.70 |
| Nyströmformer [55] | 41.58 | 70.94 | 65.52 | 79.56 |
| Nyströmformer [54] | - | - | 65.75 | 81.29 |
| RFA-Gaussian [25] | - | - | 66.0 | 56.1 |
| Transformer-LS [54] | - | - | 68.40 | 81.95 |
| LSTM | $32.99 \pm 5.46$• | $61.26 \pm 12.14$• | $85.80 \pm 0.31$• | $77.18 \pm 0.23$• |
| GRU | $44.40 \pm 1.12$• | $85.45 \pm 0.16$• | $86.70 \pm 0.21$• | $77.08 \pm 0.26$• |
| TCN | $38.62 \pm 0.41$• | $85.48 \pm 0.46$• | $60.54 \pm 0.44$• | $76.85 \pm 0.08$• |
| Deformable | $36.57 \pm 3.03$• | $56.14 \pm 0.48$• | $86.91 \pm 0.22$• | $83.69 \pm 0.97$◦ |
| CNN | $35.85 \pm 0.62$• | $55.95 \pm 0.06$• | $87.29 \pm 0.13$◦ | $83.33 \pm 1.59$◦ |
| CDIL-CNN | **64.49** $\pm 0.61$ | **91.00** $\pm 0.37$ | **87.61** $\pm 0.33$ | **84.27** $\pm 0.76$ |

of length $N = 5000$. The task requires the model to classify a sequence as one of three species of the fruit fly.

- **RightWhaleCalls**. Right whale calls are difficult to hear due to some low-frequency anthropogenic sounds. Up-calls are the most commonly documented right whale vocalization. The task requires the model to decide whether a sequence contains a set of right whale up-calls or not. The training and testing sizes of this dataset are 10934 and 1962, respectively. All sequences have a fixed length $N = 4000$.
- **MosquitoSound**. The dataset represents the wing beat of the flying mosquito. Both training and testing sets have 139883 instances with sequence length $N = 3750$. The task requires the model to classify each sequence into one of six species.

We split every original training set into training (70%) and validation (30%) parts, and used the original testing set for testing.

We have compared our model with Transfomer, its two popular variants, RNNs, and CNNs. We also included dynamic convolutional neural networks (DCNNs) [65,66], because it combines CNN and dynamic time warping, a widely used component in many time series classifiers. We used 32 channels for every convolutional layer. The classification results are shown in Table 2.

Our model significantly wins all three tasks with mean accuracies of 97.09%, 91.99%, and 91.54%, respectively. Transformers and RNNs struggle in the time series classification tasks. We found that convolutional networks perform better, probably because local signals are more important in these tasks. However, other CNNs are still inferior to our model.

### 4.4. Ablation Study: dilated convolutions and circular mixing

Compared with conventional CNN, the proposed CDIL-CNN has two major contributed components: dilated convolutions and circular mixing (padding). In this section we performed an ablation study to verify that both components are conducive to accurate and robust classifications. For this goal, we include a middle method called DIL that contains only the dilated convolution component but zero-padding. We then compare DIL with conventional CNN and CDIL-CNN.

**Table 2**
Classification accuracy (%) of different models on time series datasets. $N$ is the sequence length. A DCNN run cannot finish in two days for the MosquitoSound dataset. Means and standard deviations are computed across 5 runs. All observed differences are statistically significant according to paired $t$-test at the significance level ($p$-value) of 0.05.

| Model | FruitFlies $N = 5000$ | RightWhaleCalls $N = 4000$ | MosquitoSound $N = 3750$ |
|---|---|---|---|
| Transformer | $55.26 \pm 1.47$ | $71.84 \pm 0.65$ | $32.92 \pm 0.69$ |
| Linformer | $81.80 \pm 1.61$ | $71.17 \pm 0.84$ | $60.44 \pm 0.70$ |
| Performer | $86.57 \pm 0.98$ | $73.57 \pm 0.44$ | $68.34 \pm 0.88$ |
| LSTM | $56.61 \pm 2.50$ | $61.39 \pm 6.61$ | $32.40 \pm 1.10$ |
| GRU | $61.47 \pm 12.35$ | $63.18 \pm 8.54$ | $42.44 \pm 5.66$ |
| TCN | $91.65 \pm 0.74$ | $86.92 \pm 0.38$ | $85.99 \pm 0.28$ |
| Deformable | $92.68 \pm 1.70$ | $82.70 \pm 1.24$ | $88.92 \pm 0.43$ |
| DCNN | $86.15 \pm 4.27$ | $69.98 \pm 1.58$ | - |
| CNN | $95.30 \pm 0.27$ | $78.34 \pm 1.05$ | $89.72 \pm 0.12$ |
| CDIL-CNN | **97.09** $\pm 0.08$ | **91.99** $\pm 0.16$ | **91.54** $\pm 0.22$ |

For comparison, we first designed a more challenging XOR problem, where $N = 2^{11}$ and the test data can have the same position distribution as the training/validation data (Similar Test) or a different distribution (Dissimilar Test). See Fig. 7 for illustration. In training/validation datasets, the two marked values appear in the first half for Class 0 and in the second half for Class 1. The test data follows the same pattern in the Similar Test, while the halves flip in the Dissimilar Test. The data shift brings an extra challenge, where a non-robust model can wrongly classify the sequences by the absolute positions of the markers instead of the required XOR pattern from marked values.

The results are shown in Table 3. The CNN predictions are as bad as random guessing on both test sets, probably because it can-

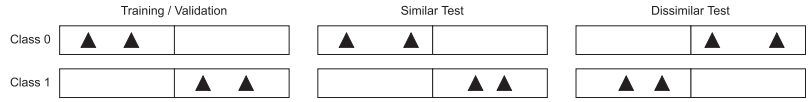



**Fig. 7.** Position distribution of Similar Test and Dissimilar Test. Similar Test has the same position distribution as the training and validation datasets. Dissimilar Test flips the position distribution.

**Table 3**
Classification accuracy (%) of Similar Test and Dissimilar Test. Means and standard deviations are computed across 5 runs.

| Model | Similar Test | Dissimilar Test |
|---|---|---|
| CNN | $50.79 \pm 0.57$ | $50.46 \pm 0.54$ |
| DIL | $99.99 \pm 0.01$ | $0.81 \pm 0.42$ |
| CDIL-CNN | $99.18 \pm 0.22$ | $98.91 \pm 0.37$ |

**Table 4**
Classification accuracy (%) of noisy RightWhaleCalls. Means and standard deviations are computed across 5 runs.

| Model | Similar Test | Dissimilar Test |
|---|---|---|
| CNN | $78.20 \pm 0.65$ | $78.12 \pm 0.97$ |
| DIL | $91.20 \pm 0.26$ | $50.33 \pm 3.87$ |
| CDIL-CNN | $91.33 \pm 0.33$ | $91.39 \pm 0.37$ |

not capture the long-range interaction between the marked positions. DIL, equipped with dilated convolutions, clearly improves the performance in Similar Test. However, DIL performs poorly on Dissimilar Test, which indicates that DIL overfits to training data and does not classify sequences by the required XOR pattern. CDIL-CNN differs from DIL by using circular padding instead of zero-padding. This change doesn't affect prediction performance in Similar Test, while achieves nearly perfect predictions in Dissimilar Test. The winning of CDIL-CNN shows that both dilated convolutions and circular padding are needed for robust classification.

We also created a noisy time series classification task using RightWhaleCalls, where the test data can have the same data shift as the training/validation data (Similar Test) or different shift (Dissimilar Test). We added the Gaussian noise of length 2000 at the end of every sequence in the training/validation set. The test set in Similar Test follows the same preprocessing, while in Dissimilar Test, the Gaussian noise part is inserted in front of each original test sequence. The mean and standard deviation of Gaussian noise equal those of the original sequence. Fig. 8 shows examples of noisy RightWhaleCalls.

The results are reported in Table 4, which leads to similar conclusions in the XOR problem. CNN gives mediocre accuracies in both Similar Test and Dissimilar Test. Dilated convolutions endow DIL better performance in Similar Test than CNN. However, zero-padding makes DIL sensitive to the data shift and degrades its performance close to random guessing in Dissimilar Test. Equipped with both dilated convolutions and circular padding, CDIL-CNN can robustly and accurately classify (higher than 91% accuracy) the time series in both cases.

Next, we visualized an input sequence of the XOR problem and its output features of CNN, DIL, and CDIL-CNN in Fig. 9. The visualization helps us understand the difference among the methods in terms of receptive field and boundary effect. In conventional CNN, the important information is present locally even in the last layer, which can lead to a wrong prediction if the two markers are distant. In contrast, the receptive field in DIL and CDIL-CNN is much wider because they use dilated convolutions. It is known that zero-padding can cause boundary artifacts [46,47]. As we can see, here DIL has such artifacts in the left-most part of its visualization. Consequently, DIL probably misses the left marked value and thus gives wrong classification. In comparison, CDIL-CNN with circular padding leads to more even output features across columns and does not suffer from boundary artifacts.

In summary, both dilated convolutions and circular padding are useful for robust and accurate classification. With the two components, CDIL-CNN well mixes the signals from the input sequence to every output position. As a result, the subsequent averaging and linear classifier provide a good ensemble and do not lessen the contributions of the important information.

### 4.5. Ablation Study: receptive fields and ensemble learning

Here we perform extra analysis on the relation between two contributed components: 1) dilated convolutions that lead to full receptive fields and 2) ensemble learning that aggregates logits from all positions. We chose the Image task to verify their contributions to the performance improvement.

First, we study the effect of different ensemble sizes. We picked $M$ positions, where $M = 1, 21, 41, \ldots, 1021, 1024$ and averaged their outputs to make the classification decision. The mean accuracy for each $M$ over five runs is reported in Fig. 10. We can see that a larger ensemble size usually leads to a better accuracy.

Second, we find that both full receptive fields and ensemble learning are needed for better performance. We recapitulate the comparison among CNN, TCN, and CDIL-CNN. Similar to CDIL-CNN, the CNN method also averages the outputs from all positions, but it has only local receptive fields because it uses dilation size 1. TCN applies dilated convolutions to achieve full receptive fields,

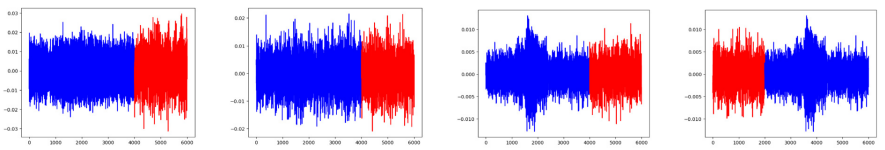
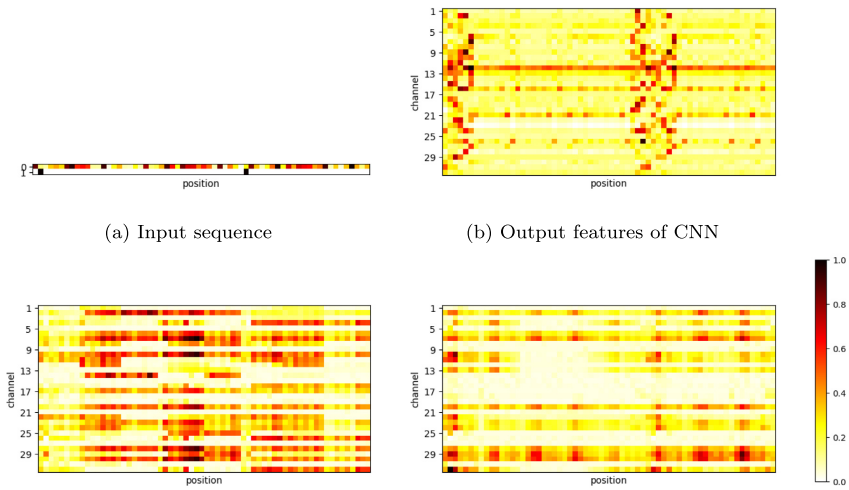



**Fig. 8.** Examples of noisy RightWhaleCalls. Blue is the original sequence, and red is the additional noise.




(a) Input sequence (b) Output features of CNN

(c) Output features of DIL (d) Output features of CDIL-CNN

**Fig. 9.** Normalized matrix plots of an example input sequence and its outputs by the last layer of CNN, DIL, and CDIL-CNN. Darker colors indicate larger values.

while the classification result comes only from the last position without ensemble learning. From Table 5, we can see that using only full receptive fields or ensemble learning leads to mediocre accuracies. In contrast, CDIL-CNN, equipped with both components, has a substantially better accuracy.

Finally, we compare the computational cost of CDIL-CNN with previous convolutional networks CNN and TCN in the inference stage. All models have the same size (128.78 K parameters) for a fair comparison. We recorded their inference time for the test dataset in terms of FLOPs and seconds. For space cost, we measured their GPU memory consumption in MB. The results are shown in Table 6. We can see that CDIL-CNN has the same FLOPs as CNN and is just slightly slower than CNN due to the circular padding. TCN has much higher FLOPs and requires more computational time, probably because PyTorch does not provide causal padding. Consequently, TCN has to use longer padding and cut off the redundant right side, which is more expensive. For space cost, all compared models used the same amount of GPU memory because they used the same kernel size, the number of layers and convolution channels. In conclusion, using full receptive fields and ensemble learning does not cause much extra computational cost compared to conventional convolution networks.

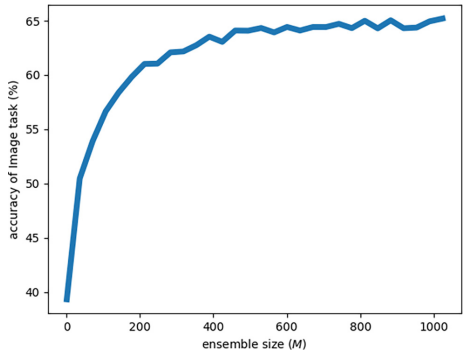



**Fig. 10.** Accuracy for the Image task with increasing ensemble size.

**Table 5**
Ablation study on full receptive fields and ensemble learning for the Image task.

| Model | CNN | TCN | CDIL-CNN |
|---|---|---|---|
| Full receptive fields | × | ✓ | ✓ |
| Ensemble learning | ✓ | × | ✓ |
| Accuracy | 35.85% | 38.62% | 64.49% |

**Table 6**
Computational cost of three convolutional neural networks.

| Model | CNN | TCN | CDIL-CNN |
|---|---|---|---|
| FLOPs (M) | 113.247 | 503.219 | 113.247 |
| Time (s) | 2.49 | 3.61 | 2.80 |
| Memory (MB) | 0.496 | 0.496 | 0.496 |

## 5. Conclusions

We have proposed a novel convolutional model named Circular Dilated Convolutional Neural Network (CDIL-CNN) for sequence classification. Based on the characteristic of very long sequential data, we have used a design that consists of multiple symmetric and circular convolutions with exponential dilation sizes. Therefore, our model can remove boundary effect and enlarge the receptive fields quickly. In this way, every position of the last convolutional layer has an equal chance to receive all information of the whole input sequence. Finally, a simple average ensemble

learning is applied to improve the accuracy. Experimental results show that our model has superior performance over all other models on various long sequential datasets.

In the future, we could apply our model to other types of long sequences, for example genome data that are known to have many long-range interactions among the sequence elements. Our model could also be used as the backbone for large-scale self-supervised pretraining (e.g., infer the masked elements from the non-masked) and then applied to few-shot or zero-shot learning, where only a few supervised labels are required in training.

## Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgements

This work was supported by The Research Council of Norway, Grant No. 287284, and the China Scholarship Council. We acknowledge for using the IDUN computing cluster [67].

## References

[1] A. Khan, B. Baharudin, L.H. Lee, K. Khan, A review of machine learning algorithms for text-documents classification, Journal of advances in information technology 1 (1) (2010) 4–20.
[2] R. Akbani, K.C. Akdemir, B.A. Aksoy, M. Albert, A. Ally, S.B. Amin, H. Arachchi, A. Arora, J.T. Auman, B. Ayala, et al., Genomic classification of cutaneous melanoma, Cell 161 (7) (2015) 1681–1696.
[3] S. Led, J. Fernández, L. Serrano, Design of a wearable device for ecg continuous monitoring using wireless technology, The 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Vol. 2, IEEE, 2004, pp. 3318–3321.
[4] S. Li, W. Li, C. Cook, C. Zhu, Y. Gao, Independently recurrent neural network (indrnn): Building a longer and deeper rnn, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 5457–5466.
[5] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE transactions on neural networks 5 (2) (1994) 157–166.
[6] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, arXiv preprint arXiv:1409.0473.
[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in neural information processing systems, 2017, pp. 5998–6008.
[8] Y. Tay, M. Dehghani, D. Bahri, D. Metzler, Efficient transformers: A survey, arXiv preprint arXiv:2009.06732.
[9] S. Bai, J.Z. Kolter, V. Koltun, An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, arXiv preprint arXiv:1803.01271.
[10] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780.
[11] K. Cho, B. Van Merriënboer, D. Bahdanau, Y. Bengio, On the properties of neural machine translation: Encoder-decoder approaches, arXiv preprint arXiv:1409.1259.
[12] S. Singh, S.K. Pandey, U. Pawar, R.R. Janghel, Classification of ecg arrhythmia using recurrent neural networks, Procedia computer science 132 (2018) 1290–1297.
[13] A.A. Sharfuddin, M.N. Tihami, M.S. Islam, A deep recurrent neural network with bilstm model for sentiment classification, in: in: 2018 International Conference on Bangla Speech and Language Processing (ICBSLP), IEEE, 2018, pp. 1–4.
[14] J. Du, C.-M. Vong, C.P. Chen, Novel efficient rnn and lstm-like architectures: Recurrent and gated broad learning systems and their applications for text classification, IEEE transactions on cybernetics 51 (3) (2020) 1586–1597.
[15] V. Mnih, N. Heess, A. Graves, et al., Recurrent models of visual attention, in: Advances in neural information processing systems, 2014, pp. 2204–2212.
[16] Y. Kim, C. Denton, L. Hoang, A.M. Rush, Structured attention networks, arXiv preprint arXiv:1702.00887.
[17] R. Child, S. Gray, A. Radford, I. Sutskever, Generating long sequences with sparse transformers, arXiv preprint arXiv:1904.10509.
[18] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, X. Yan, Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting, Advances in Neural Information Processing Systems 32 (2019) 5243–5253.
[19] I. Beltagy, M.E. Peters, A. Cohan, Longformer: The long-document transformer, arXiv preprint arXiv:2004.05150.
[20] M. Zaheer, G. Guruganesh, K.A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, et al., Big bird: Transformers for longer sequences., in: NeurIPS, 2020.
[21] S. Wang, B.Z. Li, M. Khabsa, H. Fang, H. Ma, Linformer: Self-attention with linear complexity, arXiv preprint arXiv:2006.04768.
[22] Y. Tay, D. Bahri, D. Metzler, D.-C. Juan, Z. Zhao, C. Zheng, Synthesizer: Rethinking self-attention for transformer models, in: International Conference on Machine Learning, PMLR, 2021, pp. 10183–10192.
[23] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, et al., Rethinking attention with performers, arXiv preprint arXiv:2009.14794.
[24] A. Katharopoulos, A. Vyas, N. Pappas, F. Fleuret, Transformers are rnns: Fast autoregressive transformers with linear attention, in: International Conference on Machine Learning, PMLR, 2020, pp. 5156–5165.
[25] H. Peng, N. Pappas, D. Yogatama, R. Schwartz, N.A. Smith, L. Kong, Random feature attention, arXiv preprint arXiv:2103.02143.
[26] N. Kitaev, Ł. Kaiser, A. Levskaya, Reformer: The efficient transformer, arXiv preprint arXiv:2001.04451.
[27] A. Roy, M. Saffar, A. Vaswani, D. Grangier, Efficient content-based sparse attention with routing transformers, Transactions of the Association for, Computational Linguistics 9 (2021) 53–68.
[28] Y. Tay, D. Bahri, L. Yang, D. Metzler, D.-C. Juan, Sparse sinkhorn attention, in: International Conference on Machine Learning, PMLR, 2020, pp. 9438–9447.
[29] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.
[30] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Advances in neural information processing systems 25 (2012) 1097–1105.
[31] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556.
[32] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.
[33] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu, Wavenet: A generative model for raw audio, arXiv preprint arXiv:1609.03499.
[34] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. v. d. Oord, A. Graves, K. Kavukcuoglu, Neural machine translation in linear time, arXiv preprint arXiv:1610.10099.
[35] J. Gehring, M. Auli, D. Grangier, Y.N. Dauphin, A convolutional encoder model for neural machine translation, arXiv preprint arXiv:1611.02344.
[36] C. Lea, R. Vidal, A. Reiter, G.D. Hager, Temporal convolutional networks: A unified approach to action segmentation, in: European Conference on Computer Vision, Springer, 2016, pp. 47–54.
[37] C. Lea, M.D. Flynn, R. Vidal, A. Reiter, G.D. Hager, Temporal convolutional networks for action segmentation and detection, in: proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 156–165.
[38] B. Martinez, P. Ma, S. Petridis, M. Pantic, Lipreading using temporal convolutional networks, in: ICASSP 2020–2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2020, pp. 6319–6323.
[39] P. Ma, Y. Wang, J. Shen, S. Petridis, M. Pantic, Lip-reading with densely connected temporal convolutional networks, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2021, pp. 2857–2866.
[40] J. Yan, L. Mu, L. Wang, R. Ranjan, A.Y. Zomaya, Temporal convolutional networks for the advance prediction of enso, Scientific reports 10 (1) (2020) 1–15.
[41] F. Yu, V. Koltun, Multi-scale context aggregation by dilated convolutions, arXiv preprint arXiv:1511.07122.
[42] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, A.L. Yuille, Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs, IEEE transactions on pattern analysis and machine intelligence 40 (4) (2017) 834–848.
[43] L.-C. Chen, G. Papandreou, F. Schroff, H. Adam, Rethinking atrous convolution for semantic image segmentation, arXiv preprint arXiv:1706.05587.
[44] L.-C. Chen, M.D. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, J. Shlens, Searching for efficient multi-scale architectures for dense image prediction, arXiv preprint arXiv:1809.04184.
[45] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, G. Cottrell, Understanding convolution for semantic segmentation, in: 2018 IEEE winter conference on applications of computer vision (WACV), IEEE, 2018, pp. 1451–1460.
[46] O.S. Kayhan, J.C. v. Gemert, On translation invariance in cnns: Convolutional layers can exploit absolute spatial location, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 14274–14285.
[47] B. Alsallakh, N. Kokhlikyan, V. Miglani, J. Yuan, O. Reblitz-Richardson, Mind the pad–cnns can develop blind spots, arXiv preprint arXiv:2010.02178.
[48] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
[49] K. He, X. Zhang, S. Ren, J. Sun, Identity mappings in deep residual networks, in: European conference on computer vision, Springer, 2016, pp. 630–645.
[50] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, Y. Wei, Deformable convolutional networks, in: Proceedings of the IEEE international conference on computer vision, 2017, pp. 764–773.

[51] M. Minsky, S. Papert, Perceptrons: An Introduction to Computational Geometry, MIT Press, Cambridge, MA, USA, 1969.
[52] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning internal representations by error propagation, in: D.E. Rumelhart, J.L. Mcclelland (Eds.), Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations, MIT Press, Cambridge, MA, 1986, pp. 318–362.
[53] Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, D. Metzler, Long range arena: A benchmark for efficient transformers, arXiv preprint arXiv:2011.04006.
[54] C. Zhu, W. Ping, C. Xiao, M. Shoeybi, T. Goldstein, A. Anandkumar, B. Catanzaro, Long-short transformer: Efficient transformers for language and vision, arXiv preprint arXiv:2107.02192.
[55] Y. Xiong, Z. Zeng, R. Chakraborty, M. Tan, G. Fung, Y. Li, V. Singh, Nyströmformer: A nyström-based algorithm for approximating self-attention, arXiv preprint arXiv:2102.03902.
[56] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images.
[57] R. Houtkamp, P.R. Roelfsema, Parallel and serial grouping of image elements in visual perception, Journal of Experimental Psychology: Human Perception and Performance 36 (6) (2010) 1443.
[58] D. Linsley, J. Kim, V. Veerabadran, C. Windolf, T. Serre, Learning long-range spatial dependencies with horizontal gated recurrent units, in: Proceedings of the 32nd International Conference on Neural Information Processing Systems, 2018, pp. 152–164.
[59] A. Maas, R.E. Daly, P.T. Pham, D. Huang, A.Y. Ng, C. Potts, Learning word vectors for sentiment analysis, in: Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies, 2011, pp. 142–150.
[60] D.R. Radev, P. Muthukrishnan, V. Qazvinian, A. Abu-Jbara, The acl anthology network corpus, Language Resources and Evaluation 47 (4) (2013) 919–944.
[61] H.A. Dau, A. Bagnall, K. Kamgar, C.-C.M. Yeh, Y. Zhu, S. Gharghabi, C.A. Ratanamahatana, E. Keogh, The ucr time series archive, IEEE/CAA Journal of Automatica Sinica 6 (6) (2019) 1293–1305.
[62] P. Geurts, Pattern extraction for time series classification, in: European conference on principles of data mining and knowledge discovery, Springer, 2001, pp. 115–127.
[63] L. Ye, E. Keogh, Time series shapelets: a new primitive for data mining, in: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, 2009, pp. 947–956.
[64] B.K. Iwana, S. Uchida, Time series classification using local distance-based features in multi-modal fusion networks, Pattern Recognition 97 (2020) .
[65] K. Buza, Time series classification and its applications, in: Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics, WIMS '18, Association for Computing Machinery, New York, NY, USA, 2018. doi:10.1145/3227609.3227690. URL:https://doi.org/10.1145/3227609.3227690.
[66] K. Buza, M. Antal, Convolutional neural networks with dynamic convolution for time series classification, in: K. Wojtkiewicz, J. Treur, E. Pimenidis, M. Maleszka (Eds.), Advances in Computational Collective Intelligence, Springer International Publishing, Cham, 2021, pp. 304–312.
[67] M. Själander, M. Jahre, G. Tufte, N. Reissmann, EPIC: An energy-efficient, high-performance GPGPU computing research infrastructure (2019). arXiv:1912.05848.

**Lei Cheng** received her Bachelor and Master degrees from Beijing Jiaotong University in 2016 and 2019, respectively. She is working toward the Ph.D. degree at Norwegian University of Science and Technology (NTNU). Her research interests include neural modeling for long sequential data and representation learning.

**Ruslan Khalitov** received his Bachelor and Master degrees from Higher School of Economics in 2016 and 2018, respectively. Currently he is a Ph.D. candidate at Norwegian University of Science and Technology (NTNU). He is interested in deep learning and graph theory.

**Tong Yu** received his Bachelor degree from Xidian University in 2016, and Master degree from Xi'an Jiaotong University in 2019, respectively. He is doing a Ph.D. at Norwegian University of Science and Technology (NTNU). His research focuses on neural modelling for long sequences and representation learning.

**Jing Zhang** received his Bachelor degree in computer science and technology from Beijing Jiaotong University in 2017. He is currently pursuing the Ph.D. degree at the same school. His research interests include machine learning, deep learning and zero-shot learning.

**Zhirong Yang** received his Bachelor and Master degrees from Sun Yat-Sen University, China in 1999 and 2002, and his Doctoral degree from Helsinki University of Technology, Finland in 2008, respectively. Presently he is a professor with the Norwegian Open AI Lab and Department of Computer Science at Norwegian University of Science and Technology (NTNU). He leads a research group with a focus on neural modeling for very long sequential data and learning representations for structured data.

# Classification of Long Sequential Data using Circular Dilated Convolutional Neural Networks (Supplemental Document)

Lei Cheng[1], Ruslan Khalitov[1], Tong Yu[1], Jing Zhang[2], and Zhirong Yang[1]

[1]*Norwegian University of Science and Technology*
[2]*Beijing Jiaotong University*

## 1. XOR Problem

In this group of experiments, we used the categorical cross-entropy loss function and the Adam optimizer [1] with the learning rate of 0.001. We trained every model for 100 epochs using the batch size of 40. For RNNs, namely LSTM and GRU, we used 1 layer with a hidden size of 128. For Transformer, Linformer, and Performer, we used 32 dimensions, 4 layers, and 4 heads. In CNNs, we used the kernel size of 3 and 32 channels for every convolutional layer. We adopted the varying depth of CNNs so that the last position of TCN and each position of CDIL-CNN can cover the whole sequence, i.e., the depth $L = n - 1$ for the sequence length $N = 2^n$. Other convolutional networks used the same layers for a fair comparison. Table 1 gives the model sizes.

Table 1: The number of parameters ($\approx K$) for every model on the XOR Problem. $N$ is the sequence length. CNNs include TCN, CNN, and CDIL-CNN.

| Model | $N = 2^4$ | $N = 2^5$ | $N = 2^6$ | $N = 2^7$ | $N = 2^8$ | $N = 2^9$ | $N = 2^{10}$ | $N = 2^{11}$ |
|---|---|---|---|---|---|---|---|---|
| Transformer | 26.02 | 26.02 | 26.02 | 26.02 | 26.02 | 26.02 | 26.02 | 26.02 |
| Linformer | 45.47 | 47.52 | 51.62 | 59.81 | 76.19 | 108.96 | 174.50 | 305.57 |
| Performer | 101.28 | 101.28 | 101.28 | 101.28 | 101.28 | 101.28 | 101.28 | 101.28 |
| LSTM | 67.84 | 67.84 | 67.84 | 67.84 | 67.84 | 67.84 | 67.84 | 67.84 |
| GRU | 50.95 | 50.95 | 50.95 | 50.95 | 50.95 | 50.95 | 50.95 | 50.95 |
| CNNs | 6.69 | 9.83 | 12.96 | 16.10 | 19.23 | 22.37 | 25.51 | 28.64 |
| Deformable | 8.30 | 12.25 | 15.39 | 18.53 | 21.66 | 24.80 | 27.93 | 31.07 |

**2. Long Range Arena**

For this group of experiments, we quoted Transformers' results from reference papers [2, 3, 4, 5] and ran LSTM, GRU, TCN, CNN, Deformable, and CDIL-CNN for comparison. During training, we used the categorical cross-entropy loss function and the Adam optimizer with the learning rate of 0.001. For LSTM and GRU, we used 1 layer with a hidden size of 128. We used the kernel size of 3 and 64 channels for every convolutional layer. The depth was decided by the sequence length. All tasks had a vocabulary size of 256 and an embedding dimension of 64. Every model was trained for 100 epochs. More details of RNNs and CNNs are given in Table 2.

Table 2: Hyperparameters details of RNNs and CNNs for every LRA task. $N$, $C$, $B$, $L$, $P_M$ refer to the sequence length, classes, batch size, CNN depth, and parameter size ($\approx K$) of the model $M$, respectively. CNNs include TCN, CNN, and CDIL-CNN.

| Task | $N$ | $C$ | $B$ | $P_{LSTM}$ | $P_{GRU}$ | $L$ | $P_{CNNs}$ | $P_{Deformable}$ |
|---|---|---|---|---|---|---|---|---|
| Image | 1024 | 10 | 32 | 117.00 | 92.17 | 9 | 128.78 | 133.61 |
| Pathfinder | 1024 | 2 | 256 | 115.97 | 91.14 | 9 | 128.26 | 133.09 |
| Text | 4000 | 2 | 32 | 115.97 | 91.14 | 11 | 153.09 | 157.92 |
| Retrieval | 8000 | 2 | 256 | 116.74 | 91.91 | 11 | 153.47 | 158.30 |

**3. Time Series**

In this group of experiments, we used the categorical cross-entropy loss function and the Adam optimizer with the learning rate of 0.001. We trained every model for 100 epochs using the batch size of 64. For LSTM and GRU, we used 1 layer with a hidden size of 128. For Transformer, Linformer, and Performer, we used 32 dimensions, 4 layers, and 4 heads. For CNNs, we used kernel size of 3 and 32 channels for every convolutional layer. The depth was decided by the sequence length. More details are given in Table 3.

Table 3: Hyperparameters details for every time series task. $L$ and $P_M$ refer to the CNN depth, and parameter size ($\approx K$) of the model $M$, respectively. CNNs include TCN, CNN, and CDIL-CNN.

| Task | $P_{Trans.}$ | $P_{Lin.}$ | $P_{Per.}$ | $P_{LSTM}$ | $P_{GRU}$ | $L$ | $P_{CNNs}$ | $P_{DCNN}$ | $P_{Deformable}$ |
|---|---|---|---|---|---|---|---|---|---|
| FruitFlies | 26.02 | 683.43 | 101.28 | 67.46 | 50.69 | 12 | 34.82 | 34.60 | 37.25 |
| RightWhaleCalls | 25.99 | 555.39 | 101.25 | 67.33 | 50.56 | 11 | 31.65 | 31.43 | 34.08 |
| MosquitoSound | 26.12 | 523.53 | 101.38 | 67.85 | 51.08 | 11 | 31.78 | - | 34.21 |

## 4. CDIL-CNN vs. ResNet18

Conventional CNNs often include striding and pooling. We add a comparison to a popular CNN architecture ResNet18 with the two components. ResNet18 was originally designed for images. Here we replaced the 2D operations in ResNet18 with their 1D counterparts to suit our experiments for sequences. The comparison results for the Image and the RightWhaleCalls tasks are shown in Tables 4 and 5, respectively. We recorded the inference time for the test datasets in terms of FLOPs and seconds. For space cost, we measured their GPU memory consumption in MB and the number of parameters.

CDIL-CNN achieves better accuracies than ResNet18 in both tasks, with 6.36% and 7.62% improvement. Moreover, our model requires less running time and computational cost because ResNet18 increases the number of channels when stacking convolutional layers.

Table 4: Classification accuracy and computational cost of CDIL-CNN and ResNet18 for the Image task. Means and standard deviations are computed across 5 runs.

| Model | Accuracy (%) | FLOPs (M) | Time (s) | Memory (MB) | # params |
|---|---|---|---|---|---|
| CDIL-CNN | $64.49 \pm 0.61$ | 113.247 | 2.80 | 0.496 | 128.78K |
| ResNet18 | $58.13 \pm 0.88$ | 190.142 | 3.07 | 14.905 | 3.89M |

## References

[1] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.

Table 5: Classification accuracy and computational cost of CDIL-CNN and ResNet18 for the RightWhaleCalls task. Means and standard deviations are computed across 5 runs.

| Model | Accuracy (%) | FLOPs (M) | Time (s) | Memory (MB) | # params |
|---|---|---|---|---|---|
| CDIL-CNN | $91.99 \pm 0.16$ | 123.392 | 1.05 | 0.130 | 31.65K |
| ResNet18 | $84.37 \pm 0.65$ | 686.273 | 1.57 | 14.719 | 3.84M |


[2] H. Peng, N. Pappas, D. Yogatama, R. Schwartz, N. A. Smith, L. Kong, Random feature attention, arXiv preprint arXiv:2103.02143.

[3] Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, D. Metzler, Long range arena: A benchmark for efficient transformers, arXiv preprint arXiv:2011.04006.

[4] C. Zhu, W. Ping, C. Xiao, M. Shoeybi, T. Goldstein, A. Anandkumar, B. Catanzaro, Long-short transformer: Efficient transformers for language and vision, arXiv preprint arXiv:2107.02192.

[5] Y. Xiong, Z. Zeng, R. Chakraborty, M. Tan, G. Fung, Y. Li, V. Singh, Nyströmformer: A nyström-based algorithm for approximating self-attention, arXiv preprint arXiv:2102.03902.

# Publication D - ChordMixer: A scalable neural attention model for sequences with different lengths

# CHORDMIXER: A SCALABLE NEURAL ATTENTION MODEL FOR SEQUENCES WITH DIFFERENT LENGTHS

**Ruslan Khalitov, Tong Yu, Lei Cheng & Zhirong Yang**
Department of Computer Science
Norwegian University of Science and Technology
{ruslan.khalitov,tong.yu,lei.cheng,zhirong.yang}@ntnu.no

## ABSTRACT

Sequential data naturally have different lengths in many domains, with some very long sequences. As an important modeling tool, neural attention should capture long-range interaction in such sequences. However, most existing neural attention models admit only short sequences, or they have to employ chunking or padding to enforce a constant input length. Here we propose a simple neural network building block called ChordMixer which can model the attention for long sequences with variable lengths. Each ChordMixer block consists of a position-wise rotation layer without learnable parameters and an element-wise MLP layer. Repeatedly applying such blocks forms an effective network backbone that mixes the input signals towards the learning targets. We have tested ChordMixer on the synthetic adding problem, long document classification, and DNA sequence-based taxonomy classification. The experiment results show that our method substantially outperforms other neural attention models.[1]

## 1 INTRODUCTION

Sequential data appear widely in data science. In many domains, the sequences have a diverse distribution of lengths. For example, text information can be as short as an SMS limited to 160 characters or as long as a novel with over 500,000 words[2]. In biology, the median human gene length is about 24,000 base pairs (Fuchs et al., 2014), while the shortest is 76 (Sharp et al., 1985) and the longest is at least 2,300,000 (Tennyson et al., 1995). Meanwhile, long-range interactions between DNA elements are common and can be up to 20,000 bases away (Gasperini et al., 2020). Modeling interactions in such sequences is a fundamental problem in machine learning and brings great challenges to attention approaches based on deep neural networks.

Most existing neural attention methods cannot handle long sequences with different lengths. For efficient batch processing, architectures such as Transformer and its variants have been proposed, they usually assume constant input length. Otherwise, they have to use chunking, resampling, or padding to enforce the same input length. However, these enforcing approaches either lose much information or cause substantial waste in storage and computation. Even though some architectures, such as scaled dot-product (Vaswani et al., 2017), can deal with short sequences with variable lengths, they are not scalable to very long sequences.

In this paper, we propose a novel neural attention model called ChordMixer to overcome the above drawbacks. The new neural network takes a sequence of any length as input and outputs a tensor of the same size. Moreover, ChordMixer is scalable to very long sequences (we demonstrate lengths up to 1.5M in our experiments).

ChordMixer comprises several modules or blocks with a simple and identical architecture. Each block has a Multi-Layer Perceptron (MLP) layer over the sequence channels and a multi-scale rotation layer over the element positions. The rotation has no learnable parameters, and therefore the model size of each block is independent of sequence length. After $\log_2 N$ ChordMixer blocks, every number in the output has a full receptive field of all input numbers for length $N$.

[1] Code is publicly available at https://github.com/RuslanKhalitov/ChordMixer
[2] https://en.wikipedia.org/wiki/List_of_longest_novels

We compared ChordMixer with Transformer and many of its variants in three tasks over sequential data: synthetic adding problem, long document classification, and DNA sequence-based taxonomy classification. Our method wins in nearly all tasks, which indicates that ChordMixer mixes well the signals in long sequences with variable lengths and can serve as a transformation backbone in place of conventional neural attention models.

The next section will briefly review the definitions and related work. We present the ChordMixer, including its design, properties, and implementation details, in Section 3. Settings and results of three groups of experiments are provided in Section 4. In Section 5, we conclude the paper and discuss future work.

## 2 Background and related work

A sequential data instance, or a sequence, is a one-dimensional array of sequence elements or tokens. In this work, tokens are represented by vectors of the same dimensionality. Therefore a sequence can be treated as a matrix $x \in \mathbb{R}^{d \times N}$, where $N$ is the sequence length and $d$ is the token dimensionality (or the number of channels). Each sequence may have a different length in a data set, and the distribution of $N$ can have a high range.

Neural attention or mixing is a basic function of a neural network, which transforms a data tensor into another tensor toward the learning targets. In the self-attention setting, the input and output tensors usually have the same size. Without losing information, neural attention should have a full receptive field; that is, each output number can receive information from all input numbers. However, naively connecting each pair of input and output numbers is infeasible. Self-attention of a sequence $x$ with the naive implementation requires $N^2d^2$ connections, which is too expensive when $Nd$ is large.

Most existing neural attention methods employ two-stage mixing to relieve the expense due to the full connections. For example, the widely used Transformer model (Vaswani et al., 2017) alternates the token-wise and position-wise mixing steps. The connections are limited in each step, but the receptive field becomes full after one alternation. However, Transformer has a quadratic cost to sequence length because it fully connects every token pair.

Numerous approximation methods of Transformer have been proposed to reduce the quadratic cost. For example, Longformer (Beltagy et al., 2020) and ETC (Ainslie et al., 2020) use a learnable side memory module that can access multiple tokens at once; Nyströmformer (Xiong et al., 2021) uses a few landmarks as surrogates to the massive tokens; downsampling methods also include Perceiver (Jaegle et al., 2021) and Swin Transformer (Liu et al., 2021; 2022); Performer (Choromanski et al., 2020) and Random Feature Attention (Peng et al., 2021) approximate the softmax kernel by low-rank matrix products; Switch Transformer (Fedus et al., 2021) and Big Bird (Zaheer et al., 2020) use sparse dot-products at multiple layers. A more thorough survey can be found in Tay et al. (2022). However, the approximation methods still follow the scaled dot-product approach in the original Transformer, and thus their performance remains mediocre or inferior (Gu et al., 2022; Khalitov et al., 2022; Yu et al., 2022a).

## 3 ChordMixer

In this work, we go beyond the scaled dot-product approach and aim to develop a neural attention model with the following properties:

- *full-receptive field*: every output number is mixed directly or indirectly from all input numbers of a sequence;
- *scalability*: the new method can give accurate predictions for very long sequences;
- *decentrality*: the new design is decentralized, where no sequence element or position is more central or closer to the output;
- *length flexibility*: the model can handle sequences of diverse lengths without extra preprocessing such as chunking, resampling, or padding.

Moreover, we do not assume that useful patterns appear only locally, and we try to learn all possible interactions from data. Therefore we avoid operations such as pooling over local windows of the sequences.

### 3.1 INSPIRATION FROM P2P NETWORK

We find an analogous solution in the Peer-to-Peer (P2P) communication network to achieve the above properties (Stoica et al., 2001). Full-connection is a naive communication protocol when every peer tries to communicate with the other $N-1$ peers. However, the naive protocol leads to huge routing tables, and message flooding often causes wasteful communication.

As a solution, one of the best-known protocols in P2P is Chord, where all peers are organized in a circle. At each hop of lookup, the $i$-th peer will communicate with the neighbors $i+2^0, \ldots, i+2^m$ at multiple scales and with modulus $N$, where $m = \lceil \log_2 N \rceil$. See Figure 1 (a) for illustration. The information collected from the communication will be stored and communicated at the next hop. In the distributed setting, each peer is able to collect information (directly or indirectly) from all other peers after $O(\log_2 N)$ hops (Stoica et al., 2001).

### 3.2 CHORDMIXER BUILDING BLOCK AND NETWORK ARCHITECTURE

We can mimic the P2P Chord protocol with an artificial neural network by treating sequence tokens as peers, network blocks as hops, and transformations between blocks as communications. We called the resulting building block ChordMixer.

We divide the sequence rows or channels into $M = \lceil \log_2 N \rceil + 1$ tracks, where each track has about $d/M$ channels. Then each ChordMixer block contains two layers for an input sequence $x^{\text{in}} \in \mathbb{R}^{d \times N}$:

1. *(Rotate)*: the layer rotates the tracks at different scales ($z \in \mathbb{R}^{d \times N}$):

$$z_{ij} = x^{\text{in}}_{ik} \tag{1}$$

   for $i = 1, \ldots, d$ and $j = 1, \ldots, N$, where

$$k = \begin{cases} j & \text{if } t_i = 1 \\ \left(j + 2^{t_i - 2}\right) \mod N & \text{if } t_i > 1 \end{cases} \tag{2}$$

   with $t_i$ the track index of the $i$-th channel (starting from 1). Here we add the self-loop (a non-rotated track) to the Chord protocol to include the same token in mixing. The Rotate forward pass is illustrated in Figure 1 (b)-(d).

   The Rotate layer is cheap to implement. The copy operations from $x$ to $z$ can run in parallel over $i, j, k$, and thus can efficiently be implemented by e.g., CUDA. It can also be done in PyTorch with the `roll` function. The backward pass is simply a reverse rotation: $\partial \mathcal{J} / \partial x^{\text{in}}_{ik} = \partial \mathcal{J} / \partial z_{ij}$ for a learning objective $\mathcal{J}$.

2. *(Mix)*: transforms every token (column) with a neural network $f$

$$x^{\text{out}}_{:j} = f(z_{:j}; w) \tag{3}$$

   for $j = 1, \ldots, N$, where $w$ is the network weights of $f$. In the experiments, we simply used a two-layer MLP to implement $f$ with a hidden dimension $h$ and a GELU nonlinearity. We also apply dropout between the Rotate and Mix steps.

A ChordMixer network consists of $\lceil \log_2 N_{\text{max}} \rceil$ ChordMixer blocks, where $N_{\text{max}}$ is the length of the longest sequence or the longest range of possible interactions. Each block has an identical architecture (with $\lceil \log_2 N_{\text{max}} \rceil + 1$ tracks) but a different $f$ network. Residual connection is applied around each ChordMixer block.

A ChordMixer network forms a backbone of a neural learner. We believe after the mixing, elements at every position summarize good information from all input positions. We can then use a lightweight predictor (e.g., classifier or regressor) on the individual output tokens and integrate their predictions by ensemble learning. In this work, we use linear predictors and ensemble averaging. Because the linear operation and averaging are exchangeable, we can first take the mean of output tokens and then apply the linear predictor.

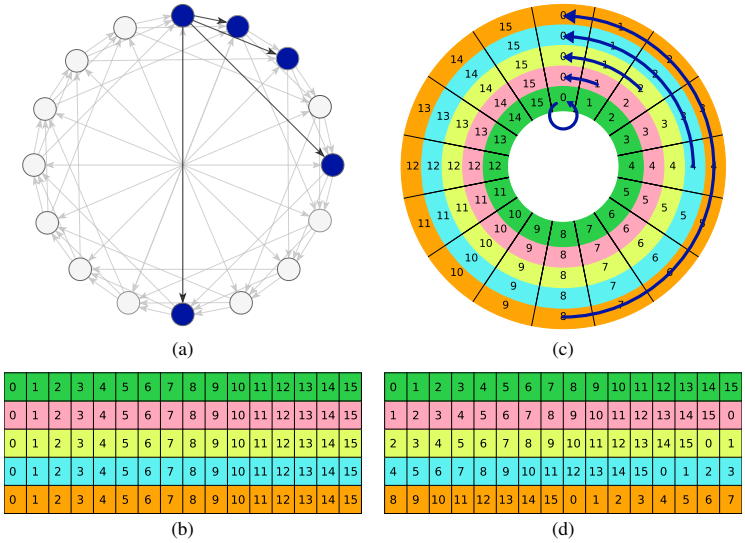



Figure 1: Illustration of the Rotate step in ChordMixer for $N = 16$: a) the original Chord protocol, where blue dots are peers, and arrows indicate communications at each hop, b) the sequence before rotation, where the numbers indicate the original position in each track, c) the rotations in different bands, where the arcs indicate the rotation sources of new Position 0, and d) the rotated tracks. Afterward, each column will be fed to the Mix step of ChordMixer.

We arrange batch learning and inference where the sequences in a batch have the same $\lceil \log_2 N \rceil$ value such that they pass through the same number of blocks. We rotate sequences in a batch in each block separately, then concatenate the rotated for MLP mixing only once. Finally, the mixed tensor is de-concatenated to sequences for block output. There is no padding or chunking in the procedure. Here the key point is that the Rotate layer is cheap, while the Mix layer benefits from the batch-mode acceleration. The same trick does not apply to Transformer-like approaches where dot products and other attention types are expensive.

### 3.3 Analysis

**Proposition 3.1.** *The receptive field becomes full for every output number after $O(\log_2 N)$ blocks.*

The proposition is a straightforward corollary of Theorem 2 in the Chord paper (Stoica et al., 2001), and we skip the proof here. By this result, not all blocks are needed for every sequence. A length-$N$ sequence passes through only the first $\lceil \log_2 N \rceil$ blocks, and therethrough the gradient back-propagation.

**Definition 3.2.** *A neural network for $D$-dimensional inputs is full-rank if the network comprises linear layers and elementwise nonlinearities, and all the linear layers have at least rank $D$.*

**Proposition 3.3.** *A ChordMixer block is full-rank on flattened $x^{in}$ if $f$ is full-rank.*

The proof is given in Appendix A. By this result, our method differs from typical bottleneck designs in conventional neural networks that contain low-rank projections. Instead, ChordMixer is a wide neural network that directs the input signal towards the learning targets in a broad way.

Next, we analyze the complexity of ChordMixer:

- Because the Rotate layer is parameter-free, all learnable parameters in a ChordMixer block are $w$. For the two-layer MLP implementation of $f$, there are $O(dh)$ parameters in one block. Overall, a ChordMixer network has $O(dh \log_2 N_{\max})$ learnable parameters.

- The Mix step dominates the running time. The MLP mixing in each block costs $O(dhN)$ and thus $O(dhN \log_2 N)$ for all blocks.

- The memory cost for a length-$N$ sequence passing a ChordMixer network is $O(dN \log_2 N)$. If we employ the Reversible Residual Network trick (Gomez et al., 2017) to avoid storing intermediate activations, the memory cost can be reduced to $O(dN)$.

### 3.4 Comparison to related work

A recent neural attention model called Paramixer (Yu et al., 2022a) also uses the Chord protocol for token mixing. It employs similar mixing MLPs over channels. Differently, Paramixer decomposes an attention matrix into several sparse factors. The non-zero structure of the sparse factors is specified by the Chord protocol, and their values are parameterized by multilayer perceptrons. Paramixer assumes that the input lengths are the same. Otherwise, truncation or padding is required to ensure the same length.

PoolFormer (Yu et al., 2022b) is another neural attention model which employs a non-parametric operator over positions. Different from the track rotations in ChordMixer, PoolFormer applies pooling over a local window around every position. The pooling connects all elements within the window and thus has a similar role for information propagation. However, defining such windows requires extra assumptions on locality. Despite success in vision and speech, the locality assumptions may not apply to other types of sequential data, as we shall see in the experiments.

Some other methods organize the sparse connections in a hierarchical manner (e.g., Goller & Küchler, 1996; Choi et al., 2018; Lea et al., 2016). Although they can achieve a full-receptive field with even fewer connections, the hierarchy breaks the symmetry: some elements (or positions) become closer to the output than others. The artifact is undesired for neural attention because a good attention model should allow each sequence element to have a (nearly) equal chance to interact with the others. In contrast, the design of ChordMixer is *decentralized*, where no element or position is below or above another. When passing through a ChordMixer network, every output position can receive information from all input positions with nearly the same probability (see Appendix G).

There is a theoretical equivalence between ChordMixer and a comprehensive convolution: each output track equals the sum of $\log_2 N$ convolutions, where each convolution applies to the corresponding input track. The first of the $\log_2 N$ convolution is $1 \times 1$, while each of the rest uses a different dilation, kernel size 1, circular padding, and causal connection (self-excluding). However, such a convolution is impractical because 1) no current software supports such a complicated convolution, and 2) the convolutions have to be run track by track, which is much slower than the proposed rotate-mix steps.

## 4 Experiments

In this section, we test the scalability and performance of ChordMixer in three different domains: 1) an arithmetic task with long-range dependence between sequence elements, 2) long text document classification, and 3) taxonomy classification based on DNA sequences. In the tasks, ChordMixer is compared with many other existing neural attention approaches, including Transformer (Vaswani et al., 2017), Reformer (Kitaev et al., 2020), Linformer (Wang et al., 2020), Nyströmformer (Xiong et al., 2021), Luna (Ma et al., 2021), Longformer (Beltagy et al., 2020), Cosformer (Qin et al., 2022), Poolformer (Yu et al., 2022b), and Structured State-Space sequence model (S4) (Gu et al., 2022). For all methods, we used a validation set to tune their hyperparameters (details in Appendix C). For completeness, we also include Recurrent Neural Network (LSTM; Hochreiter & Schmidhuber, 1997), Recursive Neural Network (TreeLSTM; Choi et al., 2018), conventional convolution network (1D CNN), hierarchical convolutional network (TCN; Lea et al., 2016) in the comparison. All experiments were conducted on a Linux machine with 8xNVIDIA Tesla V100-32GB GPUs.

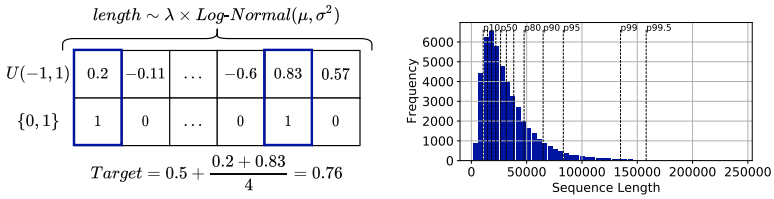



Figure 2: (Left) one instance of the Adding problem with variable lengths. Lengths are sampled from a Log-Normal distribution and times with a base length ($\lambda$). (Right) example of sequence lengths distribution for $\lambda = 16$k. The dashed vertical lines correspond to the percentiles of the distribution.

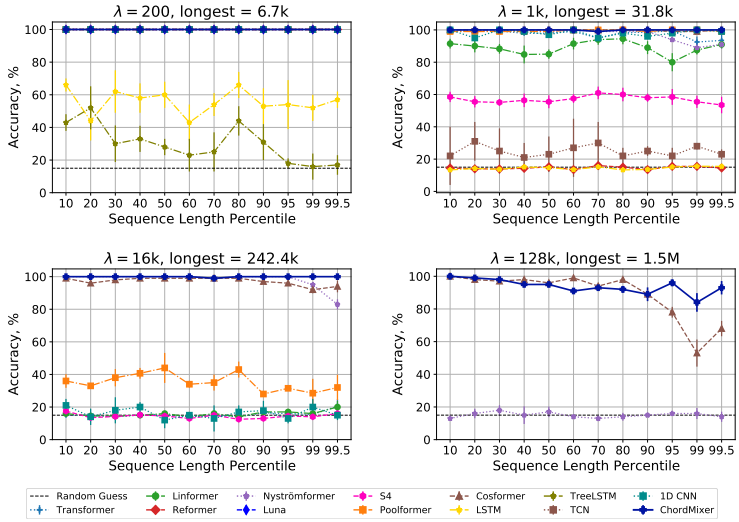



Figure 3: Prediction accuracies of the Adding problem with sequences of variable size. $\lambda$ controls the mean length of the sequences. The mean scores and error bars across 3 runs are plotted. For $\lambda = 16k$ and $\lambda = 128k$, We drop some architectures in the plots because they ran out of memory.

### 4.1 Adding problem with variable lengths

The Adding problem is a synthetic task inspired by Hochreiter & Schmidhuber (1997). This is a regression task used to test networks ability to detect relevant tokens and perform the sum operation on them. Each element of an input sequence is a pair of numbers $(a_i, b_i)$, where $a_i \sim U(-1, 1)$, $b_i \in \{0, 1\}$, $i = 1, \ldots, N$. We generated signals at two randomly selected positions $t_1$ and $t_2$ such that $b_{t_1} = b_{t_2} = 1$ and $b_i = 0$ elsewhere. The learning target is $y = 0.5 + \frac{a_{t_1} + a_{t_2}}{4}$. Unlike Hochreiter & Schmidhuber (1997); Khalitov et al. (2022), we generated sequences with different length $N = \text{round}(\lambda \cdot \zeta)$ and $\zeta \sim \text{LogNormal}(\mu, \sigma^2)$, where $\mu = 0.5$, $\sigma = 0.7$, and $\lambda$ is a base length. The resulting length distribution is left-skewed and has a wide spread (see Figure 2 right). The

variable lengths bring extra challenge to the task. In evaluation, a network prediction $\hat{y}$ is considered correct if $|y - \hat{y}| < 0.04$. A sequence example is illustrated on Figure 2 (left).

In the experiment setup, we used four different $\lambda$'s: 200, 1k, 16k, and 128k. For each $\lambda$, we generated 60,000 learning instances (sequences and their targets). A larger $\lambda$ corresponds to a more difficult task. For example, when $\lambda = 200$, the longest sequence has a length of 6.7k, while $\lambda = 128k$ leads to the longest sequence up to 1.5 million elements. Figure 2 (right) shows the histogram of sequence lengths with $\lambda = 16$k.

Figure 3 shows the comparison results. The tested models worked perfectly in the easiest setup ($\lambda = 200$). With increasing $\lambda$'s, some methods started to fail. For example, Reformer performed as poorly as random guessing for $\lambda = 1000$. Transformer, Nyströmformer, Linformer, and S4 also had more or less degraded accuracies when $\lambda = 1000$. When $\lambda = 16k$ and the longest sequence becomes 242K, three methods (Transformer, Reformer, and Luna) ran out of memory and could not finish training. Only ChordMixer, Cosformer, and Nyströmformer achieved more than 80% accuracy, while the others got close to random guess. ChordMixer was the only method that reached nearly 100% for $\lambda = 16k$. When $\lambda = 128k$, only ChordMixer and Cosformer finished training, while the methods had out-of-memory errors. ChordMixer still achieved almost 90% accuracy for all sequences, while Cosformer became inaccurate for the 10% longest sequences.

### 4.2 Long document classification

This task measures the performance of ChordMixer in modeling complex long-term dependencies for very long texts. The data set consists of a collection of academic papers and is publicly available on Github. Following He et al. (2019), we used four classes of the documents: cs.AI, cs.NE, math.AC, math.GR, which results in a data set of 11956 documents. We split the data set into 70% training, 20% validation, and 10% test sets. Unlike the original research (He et al., 2019), we encoded each document on the character level and formed a more challenging task. The minimum, median, and maximum lengths of the documents are 4.5k, 39k, and 131k, respectively. All compared methods except ChordMixer used zero padding to align with the maximum length.

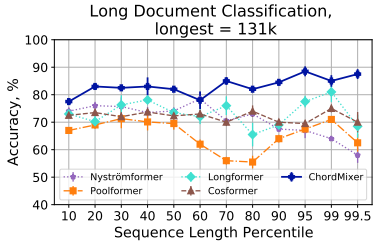



Figure 4: Accuracies in the Long Document Classification task at different length percentiles.

Figure 4 shows the classification accuracies. ChordMixer performs the best, where it is the only method that achieves nearly or above 80% accuracy at all percentiles. Nyströmformer is as good as ChordMixer at 60% sequence length percentile, while it becomes worse and worse for longer documents. Cosformer achieves around 70% accuracy at all percentiles. The accuracies of PoolFormer are lower than 70% almost everywhere and as low as 55%. The accuracies of Longformer fluctuate around 70%. Five other compared methods (Transformer, Linformer, Reformer, S4, and Luna) ran out of memory, and we have to drop them from the comparison figure.

### 4.3 DNA sequence-based taxonomy classification

Next, we test ChordMixer and other compared methods in biology. We have downloaded DNA sequences from the Genbank database[3] as well as their taxonomic labels. Then we organized four binary classification tasks (ordered by the maximum sequence length):

1. **Carassius vs. Labeo.** This is a small data set with 7263 genes for Carassius (a kind of fish) and 6718 sequences for Labeo (another kind of fish). The shortest sequence has 79 base pairs, while the longest one has 100101. In terms of the required scalability of the tested models, this task is the easiest problem within this section.

[3] `https://www.ncbi.nlm.nih.gov/genbank/`

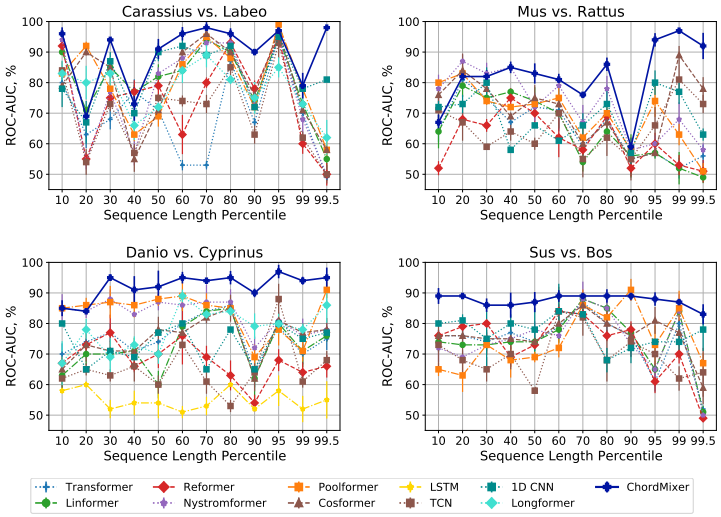



Figure 5: Test ROC-AUCs on the DNA-based taxonomy classification tasks. The score is the average of three runs with different random seeds. Plots are better read in colors.

2. **Mus vs. Rattus.** This data set is the heaviest among used within this problem setup at the genus level. The classes are strongly imbalanced, differing in size by almost a factor of two. The task is to classify a gene as a rat (rattus) or a mouse (mus). There are 275636 Mus and 133310 Rattus sequences, with minimum and maximum sequence lengths of 32 and 261093, respectively.
3. **Danio vs. Cyprinus.** Both genera belong to the cyprinidae family, where danio is small freshwater fish, and cyprinus is a genus of carps. The shortest gene consists of 34 bases, while the longest has 261943. Classes have imbalanced: 47135 Danio and 83321 Cyprinus sequences.
4. **Sus vs. Bos.** Both genera belong to the Vertebrate organisms group, where sus and bos roughly correspond to domestic pigs and cattle, respectively. Although the genera are nearly balanced (51973 sequences for bos and 50027 for sus), the data set is the most challenging because the lengths are highly different (shortest 63 and longest 447010).

We chose these four classification tasks because the taxa have a similar length distribution, and it is difficult to classify them only by sequence length (statistics in Appendix B).

We have used Area under the receiver operator curve (ROC-AUC) as the evaluation metric to overcome the class imbalance effect. All models were trained using cross-entropy loss, with empirical class weights calculated on the training set. The train/validation/test split was stratified to ensure equal class distribution within these data sets. We have applied zero padding to the maximum sequence length for all compared models except ChordMixer. For Transformer, Reformer, and Linformer, the sequences were truncated to the length such that they do not throw an out-of-memory error.

Figure 5 shows the ROC-AUC results of the four taxonomy classification tasks. ChordMixer stands at the top or near the top at almost all sequence length percentiles for all data sets. Especially at the right end of every plot, our method achieves substantially higher ROC-AUC than all other approaches, which shows that ChordMixer has a clear win over those methods which require zero padding. For the smallest data set (Carassius vs. Labeo), several methods are comparable to ChordMixer for short

sequences. However, starting from the 99 percentile (corresponds to sequence length 5630), all other compared methods experience a significant performance drop. For the longest part (99.5 percentile in Sus vs. Bos), ChordMixer achieves 89% ROC-AUC, while all other methods are below 70%. With increasing maximum sequence length, ChordMixer wins at more length percentiles. For Danio vs. Cyprinus and Sus vs. Bos, ChordMixer is clearly the best at nine out of ten listed percentiles. Interestingly, ChordMixer is the only method that achieves stable ROC-AUC (around 84-89%) across various lengths for the Sus vs. Bos data set.

### 4.4 Ablation Study

In the above experiments, every learning system comprises a neural attention part and a predictor (regressor or classifier). Here we perform an ablation study to show that the main improvement comes from the ChordMixer as the neural attention part and not from the predictor.

We chose the Carassius vs. Labeo taxonomy classification task for the ablation study. Besides ChordMixer, we included two other compared methods, Poolformer and Cosformer, because their overall accuracy or ROC-AUC seem to be more competitive than the rest (others' results in Appendix E to avoid clutter). Each neural model was combined with two different predictors: AVG) averaging followed by a linear classifier as we used in the above ChordMixer results, and CLS) using a classification token in the neural attention and then applying a linear classifier. For Poolformer and Cosformer, we also tried a linear classifier on the flattened output from neural attention (FLAT).

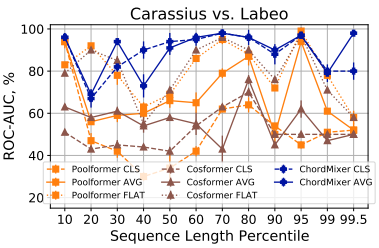



Figure 6: Ablation study on different combinations of neural attention models and classifiers.

In Figure 6, ChordMixer, either with CLS or AVG, wins for all sequence lengths except the 20 and 95 percentiles. Even with the same classifier, both Poolformer and Cosformer perform worse than ChordMixer. The FLAT strategy sometimes improves ROC-AUC for the competing models but still cannot surpass ChordMixer at most percentiles.

## 5 Conclusion and Future Work

We have proposed an effective neural attention model ChordMixer for long sequences with variable lengths. The building block of our model contains two simple layers: non-parametric multi-scale rotation and channel mixing. ChordMixer does not require chunking or padding at all and easily scales up to a maximum sequence length of 1.5 million in our work. We have provided experiments from different domains, where our method has outperformed many other recent approaches.

In our method, sequences pass through a variable number of ChordMixer blocks, depending on their lengths. We have not found practical issues so far for such a non-conventional setting. However, it challenges theoretical analysis because the mapping function space becomes more complicated, which demands more advanced mathematical tools.

In our design, we do not assume any local patterns in the data to endow nearly equal chance for all tokens to interact with each other. If specific local patterns prevail in the data (e.g., Markovian property in images), our method might require more training data than approaches that use locality as an inductive bias. In the future, we could study how to learn such patterns using our model with massive self-supervised masked training. Relative position information could be incorporated (in an economical way) to facilitate pattern learning.

## 6 ETHICS STATEMENT

It is implausible that our work would bring potential negative societal impacts. We have proposed a basic methodology for machine learning. Our research has no connection to weapons, surveillance systems, adverse experiments, or privacy and security issues. Therefore, there is little risk of harmful applications for living beings and human rights. Moreover, our findings benefit the environment by substantially reducing the computational complexity in neural attention models.

Our work also fulfills the requirements from the ICLR Code of Ethics. The research does not use human-derived data. For the synthetic data set generated by us, we publish the data generation code (under the MIT license). For the public document corpus and genome (non-human) data set, we have followed the usage requirement from the data provider and cited their contributions. There are no domain-specific considerations when working with high-risk groups because our work does not involve research on human beings. Our data does not contain offensive content. Moreover, our work complies with GDPR because we do not reveal sensitive personal information.

## REFERENCES

Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. ETC: Encoding long and structured inputs in transformers. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 268–284, 2020.

Žiga Avsec, Vikram Agarwal, Daniel Visentin, Joseph R Ledsam, Agnieszka Grabska-Barwinska, Kyle R Taylor, Yannis Assael, John Jumper, Pushmeet Kohli, and David R Kelley. Effective gene expression prediction from sequence by integrating long-range interactions. *Nature methods*, 18 (10):1196–1203, 2021.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv*, 2004.05150, 2020.

Jihun Choi, Kang Min Yoo, and Sang goo Lee. Learning to compose task-specific tree structures. In *The Thirty-Second AAAI Conference on Artificial Intelligence*, pp. 5094–5101, 2018.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers. *arXiv*, 2009.14794, 2020.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv*, 2101.03961, 2021.

Gilad Fuchs, Yoav Voichek, Sima Benjamin, Shlomit Gilad, Ido Amit, and Moshe Oren. 4sUDRB-seq: measuring genomewide transcriptional elongation rates and initiation frequencies within cells. *Genome Biology*, 15(5):1–10, 2014.

M. Gasperini, J. M. Tome, and J. Shendure. Towards a comprehensive catalogue of validated and target-linked human enhancers. *Nature Reviews Genetics*, 21:292–310, 2020.

C. Goller and A. Küchler. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of International Conference on Neural Networks*, pp. 347–352, 1996.

Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations (ICLR)*, 2022.

Jun He, Liqun Wang, Liu Liu, Jiao Feng, and Hao Wu. Long document classification from local word glimpses via recurrent attention learning. *IEEE Access*, 7:40707–40718, 2019.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Andrew Jaegle, Felix Axel Gimeno Gil, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning (ICML)*, 2021.

Ruslan Khalitov, Tong Yu, Lei Cheng, and Zhirong Yang. Sparse factorization of square matrices with application to neural attention modeling. *Neural Networks*, 152:160–168, 2022.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv*, 2001.04451, 2020.

Colin Lea, Rene Vidal, Austin Reiter, and Gregory D Hager. Temporal convolutional networks: A unified approach to action segmentation. In *European Conference on Computer Vision*, pp. 47–54, 2016.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *International Conference on Computer Vision (ICCV)*, 2021.

Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin transformer v2: Scaling up capacity and resolution. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Xuezhe Ma, Xiang Kong, Sinong Wang, Chunting Zhou, Jonathan May, Hao Ma, and Luke Zettlemoyer. Luna: Linear unified nested attention. *Advances in Neural Information Processing Systems*, 34, 2021.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A Smith, and Lingpeng Kong. Random feature attention. *arXiv*, 2103.02143, 2021.

Zhen Qin, Weixuan Sun, Hui Deng, Dongxu Li, Yunshen Wei, Baohong Lv, Junjie Yan, Lingpeng Kong, and Yiran Zhong. cosformer: Rethinking softmax in attention. *arXiv preprint arXiv:2202.08791*, 2022.

S. J. Sharp, J. Schaack, L. Cooley, D. J. Burke, and D. Söll. Structure and transcription of eukaryotic tRNA genes. *CRC Critical Reviews in Biochemistry*, 19(2):107–144, 1985.

Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

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

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 2022.

C. N. Tennyson, H. J. Klamut, and R. G. Worton. The human dystrophin gene requires 16 hours to be transcribed and is cotranscriptionally spliced. *Nature Genetics*, 9:184–190, 1995.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv*, 2006.04768, 2020.

Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A Nyström-based algorithm for approximating self-attention. *arXiv*, 2102.03902, 2021.

Tong Yu, Ruslan Khalitov, Lei Cheng, and Zhirong Yang. Paramixer: Parameterizing mixing links in sparse factors works better than dot-product self-attention. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022a.

Weihao Yu, Mi Luo, Pan Zhou, Chenyang Si, Yichen Zhou, Xinchao Wang, Jiashi Feng, and Shuicheng Yan. Metaformer is actually what you need for vision. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022b.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big Bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

## A PROOF OF PROPOSITION 3.3

*Proof.* A ChordMixer block contains a Rotate layer and a Mix layer. Below we show that both layers are full-rank.

The Rotate layer re-orders the numbers in the matrix. Therefore, it is equivalent to left-multiplying the flattened $x^{\text{in}}$ with the corresponding permutation matrix. Because permutation is linear and full-rank, the Rotate layer is full-rank.

The Mix layer applies $f$ on the columns (tokens). By the assumption, $f$ is a full-rank network that contains element-wise non-linearities and linear operators. Each linear operator is equivalent to left-multiplying a block-wise matrix with the column-major flattened input $x$, where each block in the multiplying matrix is the linear weights $W$. The following illustrates the first linear operator, where its flattened output equals

$$\begin{bmatrix} W & 0 & 0 & \cdots & 0 \\ 0 & W & 0 & \cdots & 0 \\ 0 & 0 & W & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & W \end{bmatrix} \begin{bmatrix} x_{:1} \\ x_{:2} \\ x_{:3} \\ \vdots \\ x_{:N} \end{bmatrix} \tag{4}$$

Because $W$ has at least rank $d$, the whole multiplying matrix has at least rank $dN$. Therefore, the Mix layer is full rank. □

## B DATA PREPARATION

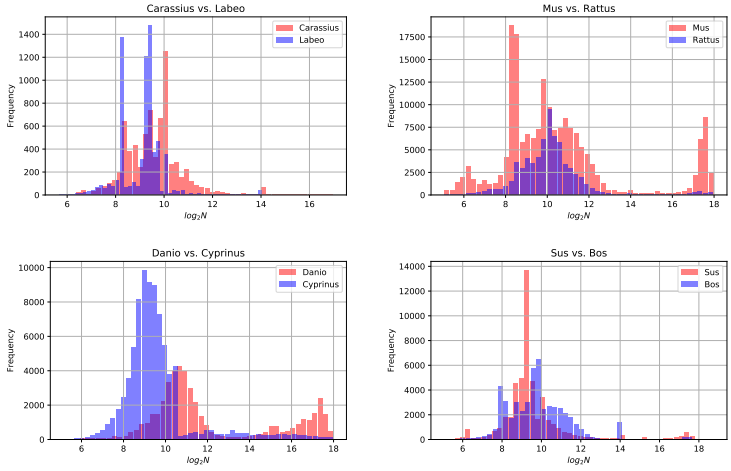



Figure 7: Sequence lengths distribution for the DNA-based taxonomy classification task.

Here we provide extra information about data preparation in the three groups of experiments

**Adding problem.** For each setup, we generated $60,000$ data instances (including training, validation, and test sets), except for the hardest task. The reason for the exception is that all sequences are padded to the maximum length, and storing so many long tensors requires enormous hard disk space and heavy RAM for reading. With $\lambda$ =128k, the longest among $60,000$ generated sequences has over three million elements, which can cause all compared methods except ChordMixer to run out of

memory. Therefore, we reduced the number of generated sequences to $12,000$ for $\lambda =$128k for a more reasonable comparison.

Because there are only two input channels in the raw sequences, we used a linear module to augment the embedding space to higher dimensions. For ChordMixer, the embedding layer outputs $d$ channels, where $d$ equals TrackSize times $\log_2 N_{\text{max}}$. The number of channels for the other compared methods, ranging from 32 to 128, are tuned by hyperparameter search (see Section C for details).

**Long document classification.** There are a few documents are extremely long (longest=$2,483,331$). Because all compared methods except ChordMixer padded zeros to align with the longest, they are easily prone to an out-of-memory error. Therefore, we used the maximum length of 131k for a more reasonable comparison. There are 56 among 11956 documents ($\leq 0.47\%$) longer than 131k, and we discarded their tailing part beyond 131k (no compared method has access to the discarded tailing part).

**DNA-based taxonomy classification.** The GenBank collection contains massive genes from many taxa. However, not all branches in the taxonomy tree are suitable for comparing neural attention models. Quite often, the taxa can easily be separated by the gene lengths. Here we particularly chose four classification tasks where the involved taxa have overlapped length distributions. See Figure 7 for sequence length histograms in the tasks, where we can see that it is hard to classify the taxa using their lengths solely.

## C MODEL CONFIGURATIONS AND TRAINING DETAILS

In this section, we provide the full set of hyperparameters used to train models and report their results. The final hyperparameters were chosen based on the Bayesian optimization algorithm with a 32GB GPU memory constraint. After the best hyperparameters selected by the minimum validation loss, we applied the corresponding model to the test set and report its performance.

The adding problem is a regression task, where we used the MSE loss. The other two tasks are classification, where we used the cross-entropy loss, with class weights, calculated on the training set, to overcome the class imbalance effect. Adam optimizer with task- and model-specific learning rates was used for all problems.

The training configuration and hyperparameters for each neural network and problem are summarized in Table 1 and Table 2. For the adding problem, we used the same hyperparameters across all setups.

Table 1: The final competitors' hyperparameters used in experiments. $L$, $H$, $E$, and $P$ refer to number of layers, number of heads, embedding size, pooling strategy, respectively. Dash ("-") indicates that the corresponding parameter is not present in the architecture.

| Task. | Model | $L$ | $H$ | $E$ | $P$ | LearningRate | BatchSize |
|---|---|---|---|---|---|---|---|
| Adding | Transformer | 2 | 4 | 64 | FLAT | 0.0005 | 5 |
| | Linformer | 1 | 2 | 128 | FLAT | 0.0001 | 10 |
| | Reformer | 2 | 2 | 64 | FLAT | 0.0003 | 5 |
| | Cosformer | 2 | 2 | 128 | FLAT | 0.0005 | 5 |
| | Poolformer | 2 | 2 | 128 | FLAT | 0.0005 | 10 |
| | Luna | 4 | - | 32 | FLAT | 0.0005 | 5 |
| | S4 | 4 | - | 32 | FLAT | 0.0005 | 5 |
| | Nyströmformer | 2 | 2 | 128 | FLAT | 0.0002 | 5 |
| Long Document Classification | Cosformer | 1 | 2 | 64 | FLAT | 0.0005 | 10 |
| | Poolformer | 1 | 2 | 64 | FLAT | 0.0005 | 10 |
| | Nyströmformer | 2 | 2 | 128 | FLAT | 0.0005 | 10 |
| Genbank C/L | Transformer | 1 | 2 | 128 | TRUNC | 0.0005 | 4 |
| | Linformer | 1 | 2 | 128 | TRUNC | 0.0005 | 4 |
| | Reformer | 1 | 2 | 64 | TRUNC | 0.0005 | 4 |
| | Cosformer | 2 | 2 | 128 | FLAT | 0.0003 | 4 |
| | Poolformer | 2 | 2 | 128 | FLAT | 0.0003 | 4 |
| | Nyströmformer | 2 | 2 | 128 | FLAT | 0.0002 | 4 |
| Genbank M/R | Transformer | 1 | 2 | 128 | TRUNC | 0.0005 | 4 |
| | Linformer | 1 | 2 | 128 | TRUNC | 0.0005 | 4 |
| | Reformer | 1 | 2 | 64 | TRUNC | 0.0005 | 4 |
| | Cosformer | 2 | 2 | 128 | FLAT | 0.0005 | 4 |
| | Poolformer | 2 | 2 | 128 | FLAT | 0.0005 | 4 |
| | Nyströmformer | 2 | 2 | 128 | FLAT | 0.0005 | 4 |
| Genbank D/C | Transformer | 1 | 2 | 128 | TRUNC | 0.0005 | 4 |
| | Linformer | 1 | 2 | 128 | TRUNC | 0.0005 | 4 |
| | Reformer | 1 | 2 | 64 | TRUNC | 0.0005 | 4 |
| | Cosformer | 2 | 2 | 64 | FLAT | 0.0003 | 4 |
| | Poolformer | 2 | 2 | 64 | FLAT | 0.0003 | 4 |
| | Nyströmformer | 2 | 2 | 64 | FLAT | 0.0002 | 4 |
| Genbank S/B | Transformer | 1 | 2 | 128 | TRUNC | 0.0005 | 4 |
| | Linformer | 1 | 2 | 128 | TRUNC | 0.0005 | 4 |
| | Reformer | 1 | 2 | 64 | TRUNC | 0.0005 | 4 |
| | Cosformer | 2 | 2 | 64 | FLAT | 0.0003 | 4 |
| | Poolformer | 2 | 2 | 64 | FLAT | 0.0003 | 4 |
| | Nyströmformer | 2 | 2 | 64 | FLAT | 0.0002 | 4 |

Table 2: Hyperparameters used to train ChordMixer. $h$ and $P$ refer to hidden layer size and pooling strategy, respectively. The embedding size is TrackSize$\times \log_2 N_{\text{max}}$.

| Task. | $h$ | TrackSize | $P$ | LearingRate | BatchSize |
|---|---|---|---|---|---|
| Adding | 128 | 16 | AVG | 0.0001 | 2 |
| Long Document Classification | 196 | 16 | AVG | 0.0002 | 2 |
| Genbank C/L | 128 | 16 | AVG | 0.0001 | 2 |
| Genbank M/R | 128 | 16 | AVG | 0.0001 | 2 |
| Genbank D/C | 128 | 16 | AVG | 0.0001 | 2 |
| Genbank S/B | 128 | 16 | AVG | 0.0001 | 2 |
| LRA Image | 420 | 28 | AVG | 0.01 | 100 |
| LRA Text | 96 | 16 | AVG | 0.007 | 90 |
| LRA Listops | 88 | 8 | AVG | 0.007 | 150 |
| LRA Retrieval | 128 | 12 | AVG | 0.007 | 90 |
| LRA Pathfinder | 320 | 24 | AVG | 0.005 | 150 |
| LRA PathfinderX | 128 | 10 | AVG | 0.001 | 120 |

## D CHORDMIXER ATTENTION EXAMPLE

Every ChordMixer block outputs a $d \times N$ matrix. We can visualize the matrices to study how the model attends to different positions. We trained a neural network for the Adding problem with $\lambda = 200$, where the shortest and longest sequences have lengths 32 and 6655, respectively. Figure 8 shows source sequences and the resulting matrices after applying the first and the last ChordMixer block (denoted by output$_1$ and output$_{\text{last}}$). We investigated three sequences with different lengths: 40, 240, and 1006.

As we can see in the top track (the non-rotated track), ChordMixer correctly identifies the relevant positions around the markers. The pattern is clearly visible for all sequences and in both output$_1$ and output$_{\text{last}}$). In output$_1$, high values basically reflect the rotation scales, while in output$_{\text{last}}$ the pattern spreads wider. Especially, output$_{\text{last}}$ demonstrates multi-scale attentions, where the upper tracks (i.e., those with no or small-scale rotations) show local attentions, while the lower tracks (i.e., with larger-scale rotations) show more even values.

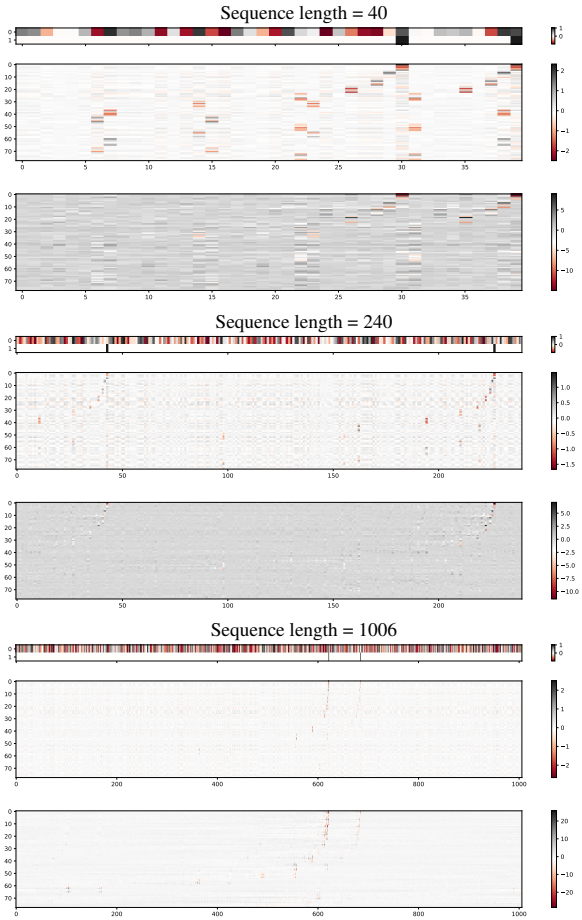



Figure 8: Visualizations of three examples in the Adding problem ($\lambda = 200$) with the ChordMixer model. The examples have different lengths. For each example, the upper chart corresponds to the input sequence, where the color map ranges from red (-1) to black (+1), with white correspond to 0; the middle and bottom charts are matrices after applying the first and the last ChordMixer blocks, respectively.

## E MORE RESULTS FROM ABLATION STUDY

In the main paper Section 4.4, we have included ChordMixer, Poolformer, and Cosformer, with three pooling strategies CLS, AVG, and FLAT. Here we provide more extensive results, including four other models (Transformer, Reformer, Linformer, and Nyströmformer) and two other strategies (TRUNC and SEGM).

TRUNC means truncation. That is, if a sequence is longer than the truncation threshold $\psi$ (we used $\psi = 16$k), the part longer than $\psi$ is discarded. For sequences shorter than $\psi$, we padded them to length $\psi$. In TRUNC, the attention model output is flattened and followed by a linear predictor.

SEGM means segmentation. If a sequence is longer than the segmentation threshold $\beta$ (we used $\beta = 2000$), it is chunked into segments of length $\beta$. For sequences shorter than $\beta$, we padded them to length $\beta$. The compared methods using SEGM are first applied to all segments. Then the predictor outputs on individual segments are averaged for the final decision. Different from TRUNC, SEGM allows the compared method to access all elements in a sequence.

TRUNC and SEGM were used for those models that are not scalable for the whole longest sequence, where only pieces of the sequences are fed to the models. These strategies assume that the characterizing patterns appear only locally (within $\psi$ or $\beta$). However, they might lose information because they drop long interactions in the truncation or segmentation.

The more extensive ablation study results are shown in Figure 9. The conclusion in the main paper remains: our method is better than the other compared methods even for all tested strategies. ChordMixer achieves the best ROC-AUC in most sequence length percentile, especially for the long sequences. The additional compared methods and strategies do not lead to better results.

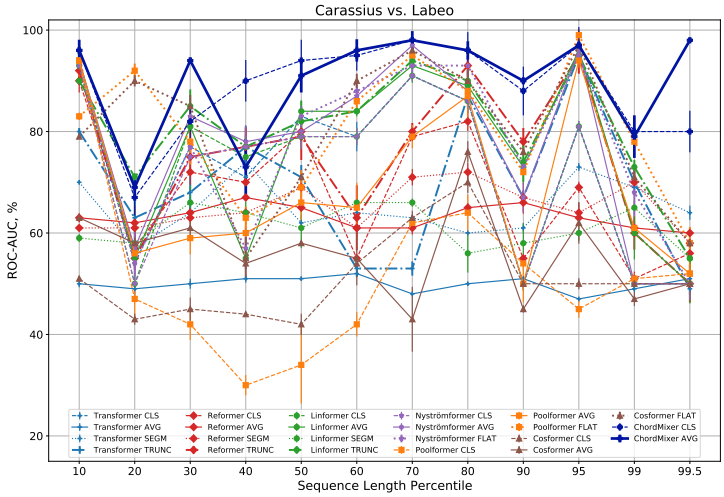



Figure 9: Ablation study on seven neural attention models with different pooling methods. Figure is better read in colors.

## F PADDING STRATEGY

For the results reported in the main paper we used zero padding to the maximum sequence length across the whole data set (Pad to Overall Max). However, some of used methods allow using different sequence lengths during training. For such methods we zero-pad sequences to the maximum sequence

length in a batch (Pad to Batch Max) to further improve training and inference speed. We also empirically verified that there is no significant change in resulting performance when switching padding strategies (see 3).

Table 3: Accuracy comparison of different models with two pooling strategies on the Long Document Classification task. Pad to Overall Max and Pad to Batch Max denote padding to the maximum sequence length in the whole data set and in a batch, respectively. Reformer gets an Out-of-Memory error in both setups even when processing sequence by sequence.

| Model | Pad to Overall Max | Pad to Batch Max |
|---|---|---|
| LongFormer | 75.3% | 75.7% |
| Nyströmformer | 73.1% | 73.0% |
| Poolformer | 67.0% | 68.1% |
| Cosformer | 72.3% | 72.6% |
| Reformer | Out-of-Memory | Out-of-Memory |

## G EMPIRICAL STUDY ON REACHING PROBABILITIES IN A CHORD NETWORK

At first glance, the Chord connection protocol looks asymmetric: in each hop, a node is connected to a few specific nodes in one direction. There is a question: if we perform a random walk from one node, how are its reaching probabilities to all nodes after several hops? It has been shown in the original Chord paper that any node can use $O(\log_2 N)$ hops to reach all $N$ nodes. Here we perform an empirical study on the distribution of the reaching probabilities.

We consider $N = 5000$ for example. We first normalize the adjacency matrix of the Chord graph $A$ to be right stochastic (i.e., rows sum to 1). Denote $\widetilde{A}$ the normalized matrix. Then $\widetilde{A}^{\lceil \log_2 N+1 \rceil} = \widetilde{A}^{14}$ gives the reaching probabilities from one node to another (including itself) after 14 hops. Because the circle is symmetric, we can pick any source node, e.g., the first, and plot the histogram of its reaching probabilities to all nodes as targets.

The result is shown in Figure 10. We can see that the reaching probabilities concentrate around the uniform probability $2 \times 10^{-4}$ with a small deviation (std=$3 \times 10^{-5}$), which justifies that in a ChordMixer network, each output position can receive information from all input positions with a nearly equal chance.

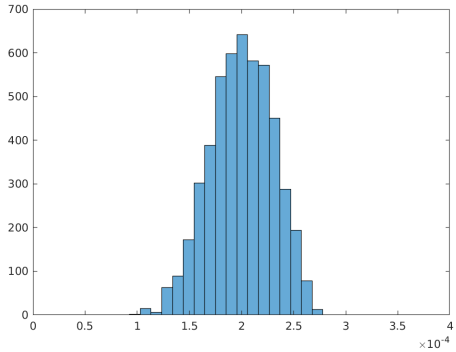



Figure 10: Histogram of reaching probabilities from a node to all nodes in a Chord network for $N = 5000$.

## H PEAK MEMORY AND RUNNING TIME COMPARISON

Table 4: Comparison of peak memory (GB) and average running time (milliseconds) per sequence for the Adding problem. For all models, except for Linformer, we applied padding to the maximum sequence length within a batch. We have used a Linux machine with one Tesla-V100 GPU (32GB memory).

| Model | $\lambda = 200$ (longest=6.7k) | | $\lambda = 128k$ (longest=1.5M) | |
|---|---|---|---|---|
| | Peak Memory | Running Time | Peak Memory | Running Time |
| LSTM | 1.5 | 460 | - | - |
| Tree-LSTM | 25.3 | 9831 | - | - |
| 1D-CNN | 1.4 | 3.3 | - | - |
| TCN | 1.4 | 5.1 | - | - |
| Transformer | 1.9 | 9.8 | - | - |
| Linformer | 1.6 | 12.5 | - | - |
| Reformer | 2.0 | 23.3 | - | - |
| Longformer | 1.7 | 11.1 | - | - |
| Luna | 1.7 | 16.6 | - | - |
| S4 | 1.4 | 3.2 | - | - |
| Poolformer | 1.4 | 3.9 | 13.6 | 378 |
| Cosformer | 1.5 | 8.7 | 22.6 | 994 |
| ChordMixer | 1.5 | 4.9 | 23.1 | 604 |

## I LONG RANGE ARENA PUBLIC BENCHMARK

We compared ChordMixer with Transformer and several of its variants on the Long Range Arena benchmark (Tay et al., 2020). Table 5 shows the accuracies of the compared methods on five LRA tasks. We can see that ChordMixer is substantially more accurate than the X-formers in all tasks. Similar to (Tay et al., 2020, Figure 3), we add ChordMixer to the plot of the relationship between accuracy and inference speed/memory footprint (see Figure 11). As we can see, ChordMixer consumes a similar amount of memory as the approximated variants of Transformer. Although Nyströmformer, Linformer, and Performer are faster, they sacrifice much prediction accuracy (more than 20% below ChordMixer).

Table 5: Classification accuracy of ChordMixer and X-formers on the LRA tasks. The performance of the competing models was taken from Tay et al. (2020) and Xiong et al. (2021). Boldface numbers are winners in each task, while the underlined are runner-ups.

| Model | ListOps $N = 2000$ | Text $N = 4000$ | Image $N = 1024$ | Retrieval $N = 4096$ | Pathfinder $N = 1024$ | PathfinderX $N = 16384$ |
|---|---|---|---|---|---|---|
| Transformer | 36.37 | 64.27 | 42.44 | 57.46 | 71.40 | ✗ |
| Longformer | 35.63 | 62.58 | 42.22 | 56.89 | 69.71 | ✗ |
| Linformer | <u>37.70</u> | 53.94 | 38.56 | 52.27 | 76.34 | ✗ |
| Reformer | 37.27 | 56.10 | 38.07 | 53.40 | 68.50 | ✗ |
| Performer | 18.01 | 65.40 | <u>42.77</u> | 53.82 | <u>77.05</u> | ✗ |
| Nyströmformer | 37.15 | <u>65.52</u> | 41.58 | <u>79.56</u> | 70.94 | ✗ |
| ChordMixer | **60.12** | **88.82** | **89.98** | **90.17** | **96.69** | **98.63** |

We do not include LRA in our main paper because it is not suitable for our problem setting: the sequences in each LRA task have the same length, and they are rather short compared to those in our experiments. See the statistics in Table 6.

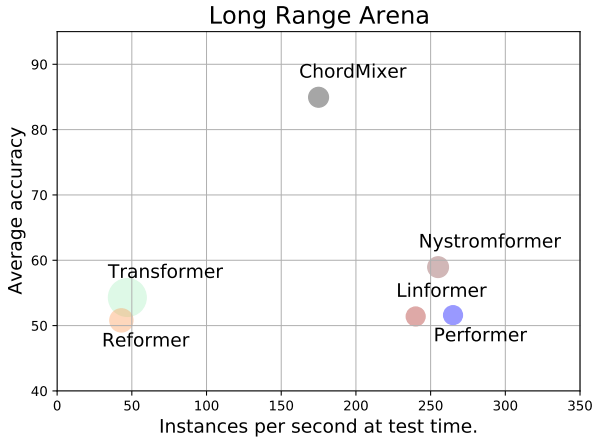



Figure 11: Accuracy (y-axis), speed (x-axis), and memory footprint (size of the circles) of different models measured on the LRA benchmark at test time.

| Task source | Task name | Sequence length |
|---|---|---|
| LRA | ListOps | 2K |
| LRA | Text | 4K |
| LRA | Retrieval | 4K |
| LRA | Image | 1K |
| LRA | PathFinder | 1K |
| LRA | PathFinder-X | 16K |
| Our experiments | Adding problem (base_length=128k) | min 32, median 134k, max 1.5M |
| Our experiments | Long document classification | min 4.5K, median 39K, max 131K |
| Our experiments | Carassius vs. Labeo | min 79, median 651, max 100K |
| Our experiments | Mus vs. Rattus | min 32, median 979, max 261K |
| Our experiments | Danio vs. Cyprinus | min 34, median 868, max 261K |
| Our experiments | Sus vs. Bos | min 63, median 664, max 447K |

Table 6: Length statistics of the data sets in our experiments and the Long Range Arena (LRA) data sets

## J SELF-SUPERVISED PRE-TRAINING WITH CHORDMIXER FOR GENETIC VARIANT CLASSIFICATION

In the main paper, we focused on ChordMixer for supervised learning tasks. We have been developing a framework where ChordMixer is used for self-supervised pre-training as well. Here we present some preliminary results for genetic variant importance prediction using DNA sequences.

An essential task in biology is to predict the influence of genetic variants on cell-type-specific gene expression, in order to inform fine-mapping of the many thousands of noncoding associations with phenotypes of interest from genome-wide association studies (GWAS). A genetic variant comprises a segment of gene sequence, a varied position, and the varied base at the position. The Genetic Variant Classification task is to predict whether a given variant effect is significant or not.

We have used the data set collected by Avsec et al. (2021). For self-supervised pre-training, we followed the Masked Language Modeling approach by using the non-masked bases (70%) to infer the masked bases (30%). In the pre-training, we have used over 100,000 randomly sampled human gene segments. The pre-training network employs the autoencoder architecture, where both encoder and decoder are ChordMixer networks. For supervised learning (fine-tuning), we extracted DNA sequences with up to 20k sequence length to construct the data set, which includes 97 992 instances over 49 tissues. In classification, we employ a light-weight classifier (average pooling + linear classifier) with the features output from the pre-trained encoder.

Table 7 shows the preliminary results, where we compared ChordMixer with the SOTA method in the field Enformer (Avsec et al., 2021). We can see that using ChordMixer as the classifier (without pre-training) is already substantially better than Enformer. The ROC-AUC is further improved by 3.7% by using the pre-trained encoder as a feature extractor.

Table 7: Preliminary results of comparing Enformer and ChordMixer on Genetic Variant Classification. The performance metric is the mean area under the receiver operating characteristic curve (ROC-AUC).

| Model | ROC-AUC$\times 100\%$ |
|---|---|
| Enformer (Avsec et al., 2021) | 70.5 |
| ChordMixer (without pre-training) | 84.9 |
| ChordMixer (with pre-training) | 88.1 |

# Publication E - Self-supervised learning for DNA sequences with circular dilated convolutional networks

Neural Networks 171 (2024) 466–473

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Full Length Article

# Self-supervised Learning for DNA sequences with circular dilated convolutional networks

Lei Cheng [a,1], Tong Yu [a,1], Ruslan Khalitov [a], Zhirong Yang [a,b,*]

[a] Department of Computer Science, Norwegian University of Science and Technology, Norway
[b] Jinhua Institute of Zhejiang Univerisity, China

ARTICLE INFO

*Keywords:*
Self-supervised learning
Masked learning
Circular dilated convolution

ABSTRACT

DNA molecules commonly exhibit wide interactions between the nucleobases. Modeling the interactions is important for obtaining accurate sequence-based inference. Although many deep learning methods have recently been developed for modeling DNA sequences, they still suffer from two major issues: 1) most existing methods can handle only short DNA fragments and fail to capture long-range information; 2) current methods always require massive supervised labels, which are hard to obtain in practice. We propose a new method to address both issues. Our neural network employs circular dilated convolutions as building blocks in the backbone. As a result, our network can take long DNA sequences as input without any condensation. We also incorporate the neural network into a self-supervised learning framework to capture inherent information in DNA without expensive supervised labeling. We have tested our model in two DNA inference tasks, the human variant effect and the open chromatin region of plants, where the experimental results show that our method outperforms five other deep learning models. Our code is available at https://github.com/wiedersehne/cdilDNA.

## 1. Introduction

With the decreasing cost of DNA sequencing, sequence-based inference has rapidly become popular in many applications, such as taxonomic classification (Khalitov, Yu, Cheng, & Yang, 2022; Rizzo, Fiannaca, La Rosa, & Urso, 2015; Yu, Khalitov, Cheng, & Yang, 2022), enhancer prediction (Sethi et al., 2020; Yang et al., 2017), variant effect prediction (Avsec et al., 2021; Lee et al., 2015), and gene expression prediction (Avsec et al., 2021; Kelley, 2020; Kelley et al., 2018; Zhou & Troyanskaya, 2015). Since the great success of deep learning methods in natural language processing (NLP), researchers have started to develop computational tools based on deep neural networks for genome sequences, as they are similar to natural language in many aspects (An et al., 2022; Ji, Zhou, Liu, & Davuluri, 2021).

Despite the many efforts to apply deep learning to sequence-based inference, the current methods still suffer from one or both of the following issues. First, they cannot scale to very long sequences, but long-range interactions are very common within DNA sequences. For example, the cis-regulatory elements (CREs) cooperating to make use of alternative promoters for biological functions can be several hundred kilobases away. Thus, the long interactions within DNA sequences must be learned in order to achieve a high-fidelity representation of the underlying biology. Second, the methods without self-supervised learning require many supervised labels to achieve accurate results. However, it is usually too expensive and time-consuming to acquire enough labeled data for genomic analysis. For example, to identify one enhancer, biologists need to place a fragment of presumed regulatory DNA near a promoter and evaluate whether or not the CRE element increases transcription (Cho, 2012). The task is very expensive, with nearly 1 million enhancers present in the mammalian genome. Therefore, we need a model that can generalize well even when labeled data is limited.

To address the problem, we propose a deep learning architecture called the Circular DILated convolutional network (CDIL) and apply it to self-supervised learning for long DNA sequences. The circular dilated design of CDIL allows it to capture the long-range interactions in DNA sequences, while the pretraining benefits CDIL with only a few supervised labels. We demonstrate that CDIL can handle long sequences and accurately conduct DNA-sequence-based inference.

We tested CDIL on two tasks, human variant effect prediction and open chromatin region (OCR) prediction, and compared it with several popular methods for sequence modeling. The experimental results show that CDIL can scale up to 10 kbp for pretraining on human genome

* Corresponding author.
*E-mail address:* zhirong.yang@ntnu.no (Z. Yang).
[1] Equal contribution.

https://doi.org/10.1016/j.neunet.2023.12.002
Received 26 May 2023; Received in revised form 11 October 2023; Accepted 1 December 2023
Available online 2 December 2023
0893-6080/

reference 38 (GRCh38) and a plant genome dataset. CDIL also achieves the best performances on both tasks.

## 2. Related work

There are three main research streams for using deep learning techniques for sequence-based inference. Convolutional neural networks (CNNs) are the most popular methods, used in many sequence-based inference tasks, such as gene expression prediction (Kelley et al., 2018), regulatory activity prediction (Zhao et al., 2021), and metagenomics gene prediction (Al-Ajlan & El Allali, 2019). For example, DeepMind (Alipanahi, Delong, Weirauch, & Frey, 2015) trained a CNN to identify the binding preference of DNA-binding and RNA-binding proteins, and it outperformed previous state-of-the-art computational methods. DeepSEA (Chen, Cofer, Zhou, & Troyanskaya, 2019; Zhao et al., 2021; Zhou & Troyanskaya, 2015) also applied a CNN to predict the non-coding variant effects and saturated mutagenesis from DNA sequences. Basenji (Kelley et al., 2018) developed a CNN framework to predict cell-type-specific epigenetic and transcriptional profiles in large mammalian genomes from DNA sequences. The second group of methods is recurrent neural networks (RNNs), such as long short-term memory (LSTM; Hochreiter & Schmidhuber, 1997) and gated recurrent unit (GRU; Bahdanau, Cho, & Bengio, 2014). These methods are also used to capture the sequential characteristics of DNA sequences. For example, Liu et al. (2019) used LSTM to detect DNA base modifications, while Yang et al. (2017) applied GRU for enhancer prediction. Recently, there has been increasing interest in incorporating Transformer or Transformer-like models for sequence-based inference. For example, Khalitov et al. (2022) studied the performances of different X-formers on taxonomy classifications. Wang et al. (2021) presented a transferable Transformer-based method, BindTransNet, for cross-cell type DNA-protein binding prediction.

More popularly, researchers tend to combine several deep learning methods to catch the long interactions within the DNA sequences. For example, Enformer (Avsec et al., 2021) first used a CNN tower to condense the sequence to a short piece and then stacked Transformer encoders to integrate long-range information. In addition, there are methods that combine CNN and LSTM or GRU for predicting the interaction between enhancer and promoter (Min, Ye, Liu, & Zeng, 2021; Zhuang, Shen, & Pan, 2019), DNA binding site prediction (Zhang, Qiao, Ji, & Li, 2020), and DNA classification (Gunasekaran et al., 2021). Nevertheless, because of insufficient labeled genomes, models based on a supervised learning paradigm have mediocre performance and limited generalization to other inference tasks (Mo et al., 2021).

Some recent work on pretraining DNA sequences followed BERT's success in NLP (Devlin, Chang, Lee, & Toutanova, 2018). DNABERT (Ji et al., 2021) adapted the idea of the BERT model to the genomic DNA setting and developed a Transformer-based architecture to achieve superior performance across various downstream DNA prediction tasks. GeneBERT (Mo et al., 2021) was proposed for multimodal pretraining to improve the model's generalizability across different cell types. Specifically, it leverages sequence pretraining, region pretraining, and sequence-region matching together for genome modeling. MoDNA (An et al., 2022) incorporated the motif pattern into the pretraining stage, which introduced external domain knowledge and contributed to learning rich semantic information from DNA sequences. However, these self-supervised methods take short DNA sequences (maximum length is 512 bp) as input due to the notorious quadratic problem.

Existing deep learning methods for DNA-based inference face two main challenges. Firstly, they struggle with scaling to very long sequences, despite the prevalence of long-range interactions in DNA sequences. Secondly, approaches without self-supervised learning require a substantial number of supervised labels, which is costly and time-consuming for genomic analysis.

## 3. Method

We propose a novel framework for modeling long DNA sequences using self-supervised learning. We begin by presenting the CDIL network architecture, followed by an overview of the self-supervised pretraining and supervised fine-tuning processes.

### 3.1. Network architecture

The key component in our network architecture is convolution with exponentially increasing dilations and circular padding. Dilated convolutions (or atrous convolutions) can achieve large receptive fields in a few layers (Bai, Kolter, & Koltun, 2018; Kalchbrenner, Espeholt, Simonyan, Oord, Graves, & Kavukcuoglu, 2016; Lea, Flynn, Vidal, Reiter, & Hager, 2017; Oord et al., 2016). As shown in Fig. 1A, we increase the dilation sizes exponentially, i.e., $d_l = 2^{l-1}$, where $d_l$ is the dilation size at the $l$th layer. Every output element can receive information from all input elements in $O(\log_2 N)$ layers for a length-$N$ sequence, which makes our model scalable to long sequences.

The circular padding convolves signals at one end with those at the other end instead of zeros. In Fig. 1A, the red nodes present signals from the other end. Conventional zero padding can cause boundary effects because signals near the boundaries have to be mixed up with zeros and thus have less chance to be forwarded. The boundary effect creates blind spots and makes CNNs sensitive to data shifting (Alsallakh, Kokhlikyan, Miglani, Yuan, & Reblitz-Richardson, 2020; Kayhan & Gemert, 2020). Circular padding relieves the boundary effect and provides robust capacity to our architecture.

A *circular dilated convolution* is defined below. Let $[\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N]$ denote an input sequence of the $l$th layer. The output sequence $[\vec{y}_1, \vec{y}_2, \ldots, \vec{y}_N]$ of circular dilated convolution is given by

$$\vec{y}_t = \sum_{k=0}^{K-1} \vec{W}_k^{(l)} \vec{x}_p + \vec{b} \tag{1}$$

where $t = 1, \ldots, N$, $p = \left[t + \left(k - \frac{K-1}{2}\right) \cdot d_l\right] \bmod N$, $K$ is the kernel size, $\vec{W}_k^{(l)}$ is the $k$th convolution weights of $l$th layer, and $\vec{b}$ is the bias. We have used the same output channel size as the input channel size, which is denoted as $D$, i.e., $\vec{y}_t$, $\vec{x}_p$, and $\vec{b} \in \mathbb{R}^D$, $\vec{W}_k^{(l)} \in \mathbb{R}^{D \times D}$.

We define a *CDIL layer*, as depicted in Fig. 1C, which contains two circular dilated convolutions, each being followed by a batch normalization (NORM) and a nonlinear activation (ReLU). A residual skip connection is incorporated around the operations. A *CDIL network* consists of $\lceil \log_2 \frac{N}{2} \rceil$ CDIL layers, where the $l$th layer uses dilation $d_l$.

The input DNA sequence of our model is represented in one-hot encoding, with the four DNA nucleobases A, T, G, and C represented as [1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], and [0, 0, 0, 1], respectively. Following this step, we apply an element-wise linear projector to map the bases to the sequence embedding $[\vec{e}_1, \ldots, \vec{e}_N]$, where the dimensionality of the embedding matches the channel size of the encoder (a CDIL network). Afterward, the sequence embedding is fed to the encoder to obtain its representation in a latent space.

It is also important to notice that we do not shorten the sequence length after each CDIL layer, which is different from bottleneck designs in conventional neural networks that contain pooling or low-rank projections. Therefore, CDIL is a wide neural network that directs the input signal toward the learning targets in a broad way. Keeping the same length as the input sequence enables our model to implement masked learning as pretraining. More details are described below.

Fig. 1B illustrates our network architecture. The processing chain of our method as a flowchart is depicted in Fig. 2.

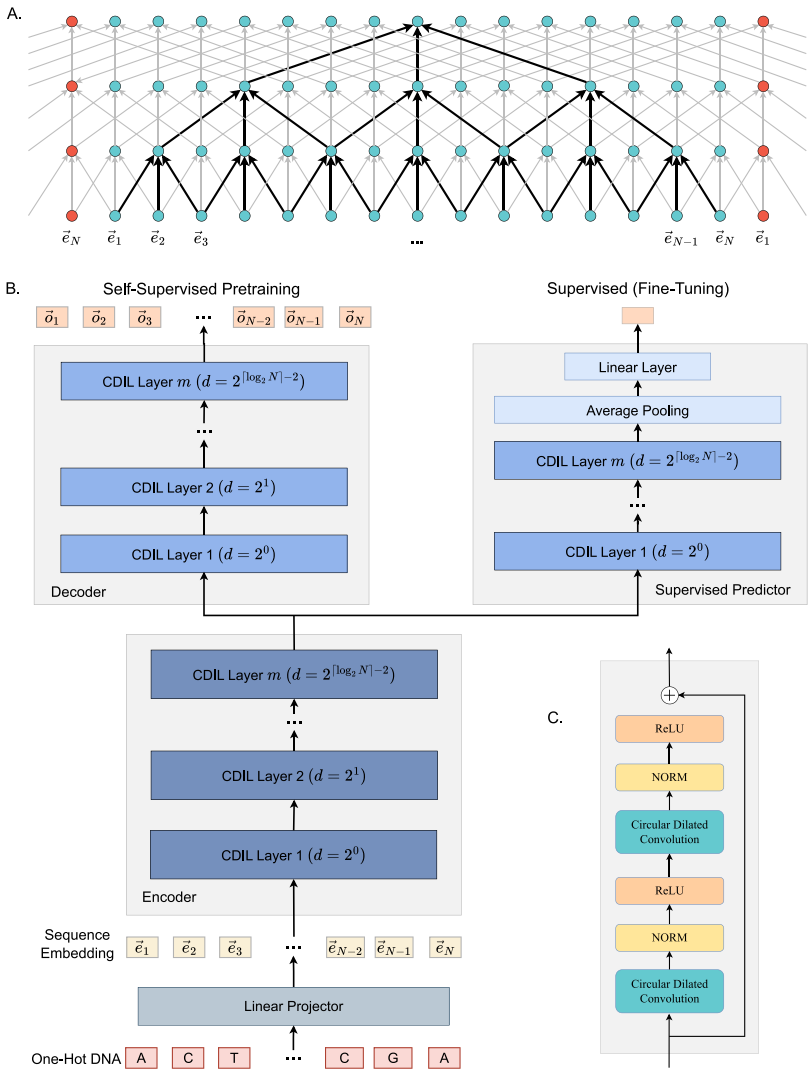



**Fig. 1.** Illustration of the CDIL framework. Panel A shows a CDIL network with three layers (dilation = 1, 2, 4, with kernel size $K = 3$). The red nodes show how circular padding is applied. Panel B illustrates the self-supervised learning architecture with CDIL networks. The self-supervised pretraining branch includes one encoder and one decoder, both being CDIL networks. The pretrained encoder is then combined with a supervised predictor and fine-tuned with supervised information. After fine-tuning, the right branch (encoder + supervised predictor) is used in sequence-based inference. Panel C presents a single CDIL layer.

### 3.2. Pretraining

The CDIL network has been applied in our previous work for supervised classification of long sequences (Cheng, Khalitov, Yu, Zhang, & Yang, 2023). This paper shows that CDIL also performs well for self-supervised learning, where all the key components in our framework include three CDIL networks.

Our pretraining follows masked language modeling (MLM; Taylor, 1953), a successful pretraining method in NLP, except that we replace Transformer-like attention modules with CDIL networks. Our method first maps the original information to a latent representation with a linear projector and an encoder and then reconstructs the masked tokens using a decoder. Because the CDIL network is good at modeling long-range interactions, we apply it to the constructions of both the encoder and decoder.

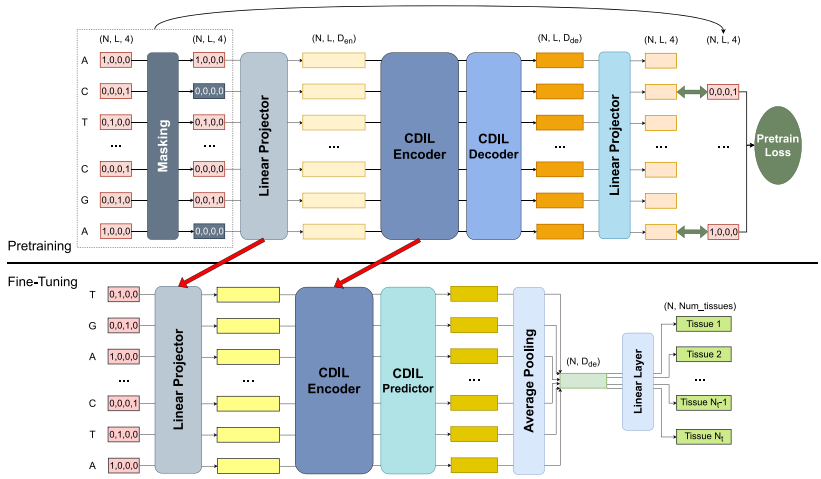



**Fig. 2.** Overview of the processing chain of CDIL with self-supervised learning. We demonstrate the tensor shape (batch size, sequence length, dimensionality) or (batch size, dimensionality) at every step. The DNA bases are represented in one-hot encoding. In the pretraining stage, we randomly mask some positions. Then, we apply an element-wise linear projector to augment the dimensionality. The latent features are fed to a CDIL encoder and a lightweight CDIL decoder. The pretraining loss is only calculated on masked positions. After pretraining, the embedding linear projector and the encoder are used as the feature extractor in downstream tasks. In the fine-tuning stage, we stack a CDIL predictor and an average pooling layer to learn a high-level feature for each input sequence. Finally, we use a linear layer to make predictions about multiple tissues.

In masked learning, we randomly sample a subset of sequence positions as a mask and use the non-masked sequence elements to infer those that are masked. We replace 80% of the selected elements with [0, 0, 0, 0] and 10% with the one-hot encoding of a random nucleobase, A, T, G, or C; the remaining 10% are unchanged. Methods using $\kappa$-mer encoding, such as GeneBERT and MoDNA, have to use more comprehensive contiguous masking to avoid inference leakage from neighboring tokens. In contrast, our method can use simple uniform random masking, which is more straightforward and without data leakage.

The sequence in latent space is fed to another CDIL network as a decoder, which tries to reconstruct the masked bases. Because the reconstruction part appears only in pretraining and is not used in the sequence-based inference, we follow He et al. (2022) and use a lightweight decoder with fewer channels. We use CrossEntropyLoss to compare the decoder output $[\vec{o}_1, \ldots, \vec{o}_N]$ and the original sequence elements in the masked positions.

### 3.3. Fine-tuning

After pretraining, the encoder can be used as a feature extractor in downstream sequence-based supervised inference tasks. The process is shown in the right branch of Fig. 1B.

The embedding and encoding steps are the same as in pretraining, except that the sequence is not masked. Afterward, the sequence in latent space is fed to a predictor, which comprises another CDIL network followed by a position-wise average pooling and a linear layer. The CDIL networks enable each final output position to capture information from the entire input sequence; thus, the average pooling and linear layer act as average ensemble learning to achieve better performance. The predictor is trained by supervised learning and possible fine-tuning of the encoder (see Section 4.3 for details).

## 4. Experiments

We have tested CDIL on two tasks, including DNA sequences of humans and plants. Our model is compared with the conventional CNN using dilation 1 and zero-padding, and with Transformer and its two variants, Nyströmformer and Performer. For the variant effect prediction, we downloaded the trained Enformer as the feature extractor and fine-tuned a random forest classifier. For the plant task, we also included PlantDeepSEA (Chen et al., 2019; Zhao et al., 2021), which employs a pyramid convolutional architecture.

### 4.1. Variant effect prediction in human tissues

This binary classification task is to determine whether a particular gene variant has an impact on a specific tissue. A gene variant is a specific version of a gene that differs from the reference gene in one position. For example, given the reference sequence ATTGCATGTTC and the alternative sequence ATTGCGTGTTC of the same tissue, the sixth nitrogenous base becomes G in the alternative sequence, which can lead to either a positive or a negative effect on the tissue itself.

The human genome dataset prepared by Avsec et al. (2021) includes 97,992 instances across 49 human tissues in total. In this dataset, each instance includes one reference sequence (from GRCh38), one alternative sequence, the tissue, and the variant effect label. We randomly split all the instances into training (80%), validation (10%), and test (10%) sets.

In the pretraining stage, we extracted a total of 400,000 DNA sequences of 10 kbp length from GRCh38 as follows: (1) we randomly selected one of the 24 chromosomes; (2) we randomly selected a starting position on the chosen chromosome; (3) we extracted the DNA sequence of 10 kbp from left to right. Each extracted DNA sequence was then one-hot encoded. We used 80% of the sequences for training, 10% for validation, and 10% for testing.

To accommodate the input of two sequences, we make slight modifications to the supervised pipeline. As illustrated in Fig. 3, the initial step involves utilizing a pretrained model to obtain the latent representation of both sequences. Subsequently, a subtraction operation is performed between the two representations. The resulting subtraction values are then passed into the predictor.

Table 1 summarizes the computational and memory efficiency of the compared models. During training, we input each model with a

**Table 1**
Number of parameters, running time, memory consumption, and AUROC values of different models for variant effect prediction. The configuration of all the models is for a 1 kbp sequence length (details in the supplemental document). Running time and memory were calculated for one batch. Here, running time means the time used for both forward and backpropagation. We report the average running time for 10 random batches. All the experiments were conducted on the same Nvidia A100-80GB GPU. The use of a "/ " signifies that the model cannot be assessed using the same evaluation methodology as other models. Means and standard deviations of AUROC are computed across five runs.

| Model | Parameters (k) | Running time (ms) | Memory (MB) | AUROC (×100%) |
|---|---|---|---|---|
| CDIL w/ pretrain | 39.8 | 14.51 | 65.22 | **88.08 ± 0.09** |
| CDIL w/o pretrain | 39.8 | 14.52 | 65.22 | 87.50 ± 0.01 |
| CNN | 39.8 | 11.60 | 49.03 | 85.42 ± 0.04 |
| Transformer | 33.3 | 66.65 | 190.05 | 66.49 ± 0.25 |
| Nyströmformer w/ pretrain | 42 | 35.08 | 133.19 | 87.71 ± 0.16 |
| Nyströmformer w/o pretrain | 42 | 35.08 | 133.19 | 87.50 ± 0.01 |
| Performer | 41.3 | 24.25 | 76.44 | 87.32 ± 0.06 |
| Enformer | / | / | / | 84.53 ± 0.04 |

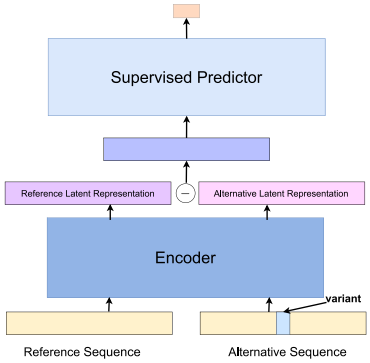



**Fig. 3.** Pipeline for variant effect prediction. The first CDIL network extracts the latent representations for reference and alternative sequences by using the pretrained encoder. The second CDIL network, an average pooling, and a linear layer work together as a predictor.

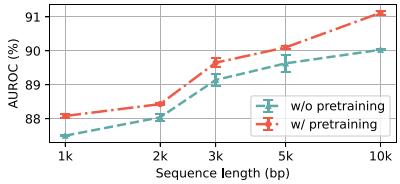



**Fig. 4.** The AUROC ×100% of CDIL on variant effect prediction with various sequence lengths. Means and standard deviations are computed across five runs.

single batch of 1 kbp sequences as input to evaluate their efficiency. The evaluation was conducted on the same hardware and software setup. Notably, despite having a similar model capacity to the two Transformer variants, Nyströmformer and Performer, CDIL demonstrates pronounced enhancements in terms of both running time and memory efficiency. Enformer cannot be properly assessed in the same way as the other models in terms of efficiency. This is because Enformer is a heavily pretrained model which is only used as a feature extractor for a random forest classifier. Comparing neural network models directly to the random forest would be unjust.

The last column in Table 1 shows the comparison between CDIL and other competitors in AUROC (area under the receiver operating characteristic curve) for the sequence length of 1 kbp. We followed the original inference process of Enformer (Avsec et al., 2021): we downloaded the trained Enformer model and used it to get latent representations of both reference and alternative sequences; then, we applied a random forest for classification output. The result shows that CDIL with self-supervised pretraining performs the best, with a mean AUROC of 88.08%. Our method is significantly better than the runner-up, Nyströmformer, with a $p$-value of 0.00021.

Furthermore, we find that pretraining on CDIL helps as the sequence length scales up. We present the AUROC of CDIL (average over all 49 tissues) in Fig. 4 for five different sequence lengths. CDIL's performance improves as the DNA sequence length increases, both with and without pretraining. This result suggests that CDIL can effectively incorporate long-range interactions within DNA sequences. We also observe that CDIL with pretraining performs better than without, indicating that pretraining can benefit CDIL for variant effect prediction. In particular, when the sequence length is 10 kbp, pretraining improves the performance by 1.09%, indicating that CDIL can benefit from pretrained information on longer sequences. The best result achieved by CDIL with self-supervised learning is $91.12 \pm 0.06\%$ using a sequence length of 10 kbp, which is substantially better than all the other methods.

### 4.2. OCR prediction in plants

This is a set of binary classification tasks to determine whether a given DNA sequence belongs to an OCR for multiple specific features. OCRs are special gene regions that are closely associated with important downstream tasks, such as predicting gene expression and discovering high-impact gene sites. OCRs were identified by the assay for transposase-accessible chromatin using sequencing (ATAC-seq; Buenrostro, Giresi, Zaba, Chang, & Greenleaf, 2013). A DNA sequence was labeled as positive if its center 200 bp had more than 50% overlap with the OCR, and as negative otherwise. The dataset contains multiple features for each plant (ranging from 9 to 19; see the supplemental document's Table 2 for details).

The plant genome dataset from Zhao et al. (2021) includes six representative species: *Arabidopsis thaliana*, *Brachypodium distachyon*, *Oryza sativa* (rice), *Setaria italica*, *Sorghum bicolor*, and *Zea mays* (maize). We used seven reference genomes, including two rice varieties (*O.sativa-MH* and *O.sativa-ZS*) and one genome for each other species.

In the pretraining stage, we used one-hot encodings for nucleobases A, T, G, and C. We followed the setting of Zhao et al. (2021) and encoded all miscellaneous bases (e.g., N, M, K) to [0.25, 0.25, 0.25, 0.25]. We used two data size configurations for pretraining, 1% (64,000 sequences from maize and 51,200 sequences from other species) and 100% (6,400,000 sequences from maize and 5,120,000 sequences from other species). In self-supervised pretraining, we used DNA sequences of 1 kbp length and a separate model for each genome variety. In total, we obtained 14 pretrained models.

First, we compare CDIL with other models on sequences of length 1 kbp in Table 2. There are also two data size configurations for

**Table 2**
The AUROC ×100% of different models on OCR prediction of length 1 kbp. The top part uses the 1% data size configuration, i.e., 64,000 DNA sequences from maize and 51,200 DNA sequences from other species. The bottom part uses the 100% data size configuration, i.e., 6,400,000 sequences from maize and 5,120,000 sequences from other species. Means and standard deviations are computed across five runs. CDIL with pretraining is significantly better than the runner-up for each task, with $p$-values $< 0.05$ from the $t$-test. Nys denotes Nyströmformer.

| Plants | *A.thaliana* | *B.distachyon* | *O.sativa-MH* | *O.sativa-ZS* | *S.italica* | *S.bicolor* | *Z.mays* |
|---|---|---|---|---|---|---|---|
| CDIL w/ pretrain | **92.90 ± 0.08** | **93.47 ± 0.11** | **93.53 ± 0.24** | **93.09 ± 0.12** | **94.43 ± 0.07** | **96.42 ± 0.11** | **97.21 ± 0.08** |
| CDIL w/o pretrain | 90.39 ± 0.25 | 91.95 ± 0.25 | 91.45 ± 0.51 | 91.66 ± 0.28 | 92.42 ± 0.84 | 95.80 ± 0.17 | 95.87 ± 0.18 |
| CNN | 82.02 ± 0.21 | 87.09 ± 0.21 | 86.05 ± 0.34 | 84.94 ± 0.35 | 88.30 ± 0.21 | 91.52 ± 0.25 | 89.79 ± 0.23 |
| PlantDeepSEA | 90.24 ± 0.39 | 90.59 ± 0.15 | 91.08 ± 0.44 | 90.18 ± 0.14 | 92.39 ± 0.13 | 94.86 ± 0.23 | 95.19 ± 0.21 |
| Transformer | 76.74 ± 0.54 | 85.63 ± 0.20 | 83.17 ± 0.37 | 83.56 ± 0.17 | 86.27 ± 0.15 | 88.90 ± 0.45 | 84.78 ± 0.78 |
| Nys w/ pretrain | 89.80 ± 0.32 | 92.24 ± 0.27 | 90.75 ± 0.23 | 90.50 ± 0.17 | 92.75 ± 0.02 | 95.45 ± 0.05 | 95.20 ± 0.16 |
| Nys w/o pretrain | 86.02 ± 0.40 | 90.19 ± 0.40 | 86.92 ± 0.67 | 86.31 ± 0.62 | 90.94 ± 0.09 | 93.67 ± 0.41 | 89.77 ± 0.37 |
| Performer | 75.95 ± 0.18 | 85.25 ± 0.14 | 82.81 ± 0.11 | 83.02 ± 0.11 | 86.60 ± 0.12 | 88.09 ± 0.27 | 85.13 ± 0.55 |
| CDIL w/ pretrain | **95.02 ± 0.04** | **95.39 ± 0.02** | **95.97 ± 0.06** | **95.72 ± 0.08** | **96.06 ± 0.03** | **97.78 ± 0.02** | **98.66 ± 0.02** |
| CDIL w/o pretrain | 94.50 ± 0.06 | 95.25 ± 0.04 | 95.81 ± 0.07 | 95.61 ± 0.08 | 95.90 ± 0.07 | 97.67 ± 0.02 | 98.30 ± 0.05 |
| PlantDeepSEA | 94.51 ± 0.05 | 95.03 ± 0.09 | 95.83 ± 0.13 | 95.52 ± 0.06 | 95.76 ± 0.10 | 97.66 ± 0.02 | 98.27 ± 0.04 |

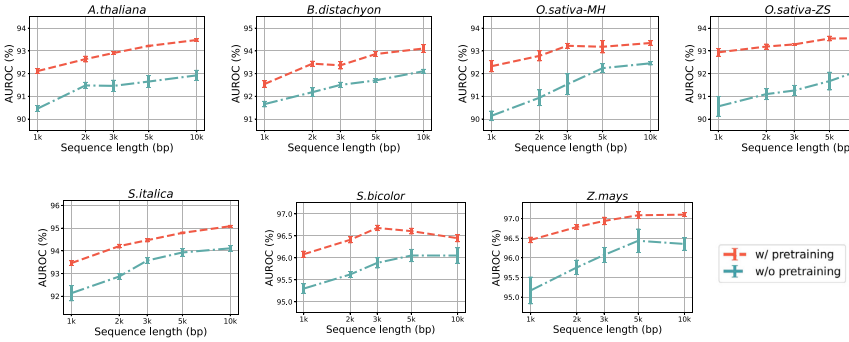



**Fig. 5.** The AUROC ×100% of CDIL on OCR prediction with increasing sequence lengths. We compared w/o pretraining and w/ pretraining for each plant. Means and standard deviations are computed across five runs.

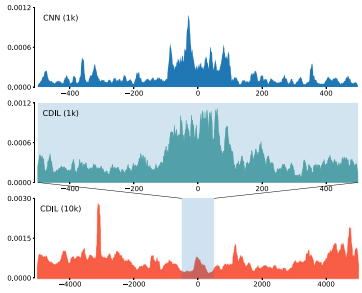



**Fig. 6.** Contribution scores of different methods. The top and middle come from CNN and CDIL, respectively, for the same 1 kbp DNA sequence. The bottom shows the contribution scores of CDIL for a 10 kbp DNA sequence, with the central 1 kbp region being identical to the previous sequence.

the number of labeled training sequences, matching the settings used during pretraining. The performance is evaluated using average AUROC over all features of the plant. CDIL outperforms the other methods on all tasks. CDIL with self-supervised pretraining achieves the best for all plants. The $p$-values show that our method is significantly better than the runner-ups. CDIL without pretraining is the first runner-up in 10 of 14 cases and comparable to the first runner-up in the remainder. We also notice that self-supervised learning achieves higher performance improvement for limited labeled data. The average AUROC improvement from CDIL without pretraining to CDIL with pretraining is 1.64% when we use the 1% training data size configuration, while this number is only 0.22% for the 100% training data size. This means that self-supervised learning benefits from limited labeled data.

Next, we evaluate the performance of CDIL using five different input sequence lengths for the 1% data size configuration. The results are illustrated in Fig. 5. We observe similar patterns to those in human tissues. CDIL with pretraining consistently outperformed the non-pretrained version for all plant species and sequence lengths, demonstrating the universal benefit of self-supervised learning. In addition, the performance is improving with increasing sequence lengths in most cases, both with and without pretraining, suggesting that CDIL effectively captures the long-range interactions for OCR predictions.

Finally, we studied the contribution scores of different methods for one DNA sequence. The contribution scores are derived using DeepLIFT (Shrikumar, Greenside, & Kundaje, 2017), a tool to quantify the importance of each position within an input sequence by the backpropagation-based approach. We visualized the absolute contribution values (average over 10 bp and 100 bp for 1 kbp and 10 kbp sequences, respectively) in Fig. 6. The top and middle panels show the contribution scores of CNN and CDIL for a 1 kbp DNA sequence, respectively. The boundary positions of CNN demonstrate less contribution than CDIL, indicating that the model without circular dilated convolutions suffers from a stronger boundary effect. In contrast, CDIL shows the ability to alleviate the boundary effect. The bottom panel displays the contribution scores of a 10 kbp DNA sequence, with its central 1 kbp region identical to the previous sequence. This suggests that CDIL can handle long DNA sequences and capture important positions far from the center.

**Table 3**
The AUROC ×100% of CDIL with different supervised learning strategies after pretraining. Means and standard deviations are computed across five runs. The best mean AUROC values are highlighted in bold.

| | variant effect | *A.thaliana* | *B.distachyon* | *O.sativa-MH* |
|---|---|---|---|---|
| CDIL-probing (CP) | 81.31 ± 0.04 | 92.12 ± 0.10 | 93.24 ± 0.14 | 93.53 ± 0.05 |
| Fine-tuning (FT) | **88.08** ± 0.09 | 92.90 ± 0.08 | 93.47 ± 0.11 | 93.53 ± 0.24 |
| Combination (CP-FT) | **88.08** ± 0.03 | **92.95** ± 0.04 | **93.53** ± 0.09 | **93.74** ± 0.07 |
| | *O.sativa-ZS* | *S.italica* | *S.bicolor* | *Z.mays* |
| CDIL-probing (CP) | 92.77 ± 0.17 | 94.11 ± 0.04 | 96.03 ± 0.20 | 96.98 ± 0.10 |
| Fine-tuning (FT) | **93.09** ± 0.12 | 94.43 ± 0.07 | 96.42 ± 0.11 | 97.21 ± 0.08 |
| Combination (CP-FT) | **93.09** ± 0.15 | **94.47** ± 0.06 | **96.59** ± 0.05 | **97.25** ± 0.05 |

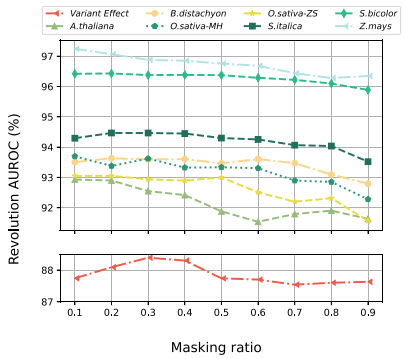



**Fig. 7.** The AUROC ×100% of CDIL on variant effect and OCR prediction with different masking ratios.

### 4.3. Ablation study

**Fine-Tuning vs. CDIL-Probing.** In the above experiments, we have used fine-tuning as the default approach for the prediction tasks. Kumar, Raghunathan, Jones, Ma, and Liang (2022) suggested that combining linear probing and fine-tuning could yield better results for distribution shift data. We also investigated whether the combination is helpful for DNA sequence-based inferences. Because we used an additional CDIL network on top of the pretrained network during pretraining, we refer to our probing method as CDIL-Probing (CP) instead of linear probing. We selected a sequence length of 1 kbp and compared the performance of three different strategies: Fine-Tuning (FT), CDIL-Probing (CP), and a combination of the two (CP-FT). In CP, we froze the pretrained encoder and used it to probe the supervised predictor. In CP-FT, we unfroze the encoder and retrained the whole network, with the supervised predictor initialized with the one obtained in CP.

The investigation result is shown in Table 3. We can see that CP-FT is better than CP and FT in all cases. We chose FT instead of CP-FT because the latter requires an extra round of training (CP) and the gain is very small.

**Masking Ratio.** Masked language models typically use a masking rate of 15%. However, some researchers have also argued that a masking rate of 40% is optimal for large pretraining models (Wettig, Gao, Zhong, & Chen, 2022). Therefore, we investigated how the masking ratio of CDIL influences the performance of DNA sequence-based inferences. We studied a range of different masking ratios from 10% to 90% for both variant effect and OCR prediction tasks. We used sequences of length 1 kbp and fixed all the other parameters. The results, as shown in Fig. 7, suggest that a lower masking ratio between 10% and 40% is often a good choice, and the optimal ratio can depend on the task and organisms. We empirically selected 30% and 15% for humans and plants, respectively.

## 5. Conclusion and future work

We have proposed a self-supervised learning framework for DNA sequences. The basic building blocks in our method are circular dilated convolutions, which enable wide receptive fields and circumvent boundary effects. The resulting neural network model can capture information from long-range interactions among nucleobases, especially when the supervised labels are limited. Experimental results showed that our design achieved state-of-the-art performance in both variant effect and OCR prediction.

In the future, we could apply the self-supervised pretraining to DNA sequences from multiple species. The jointly pretrained model could capture more structured information in DNA and learn even better representations for biological and medical inference tasks. Our framework could also be extended to other tasks, such as predicting gene expressions.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

We share the code and data.

## Acknowledgments

We acknowledge using the IDUN computing cluster (Själander, Jahre, Tufte, & Reissmann, 2019).

## Funding

This work was financed by the Research Council of Norway (grant number 287284).

## Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.neunet.2023.12.002.

## References

Al-Ajlan, A., & El Allali, A. (2019). CNN-MGP: Convolutional neural networks for metagenomics gene prediction. *Interdisciplinary Sciences: Computational Life Sciences*, *11*(4), 628–635.

Alipanahi, B., Delong, A., Weirauch, M. T., & Frey, B. J. (2015). Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning. *Nature biotechnology*, *33*(8), 831–838.

Alsallakh, B., Kokhlikyan, N., Miglani, V., Yuan, J., & Reblitz-Richardson, O. (2020). Mind the pad–CNNs can develop blind spots. arXiv preprint arXiv:2010.02178.

An, W., Guo, Y., Bian, Y., Ma, H., Yang, J., Li, C., et al. (2022). MoDNA: motif-oriented pre-training for DNA language model. In *Proceedings of the 13th ACM international conference on bioinformatics, computational biology and health informatics* (pp. 1–5).

Avsec, Ž., Agarwal, V., Visentin, D., Ledsam, J. R., Grabska-Barwinska, A., Taylor, K. R., et al. (2021). Effective gene expression prediction from sequence by integrating long-range interactions. *Nature Methods*, *18*(10), 1196–1203.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

Bai, S., Kolter, J. Z., & Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271.

Buenrostro, J. D., Giresi, P. G., Zaba, L. C., Chang, H. Y., & Greenleaf, W. J. (2013). Transposition of native chromatin for fast and sensitive epigenomic profiling of open chromatin, DNA-binding proteins and nucleosome position. *Nature Methods*, *10*(12), 1213–1218.

Chen, K., Cofer, E., Zhou, J., & Troyanskaya, O. (2019). Selene: A PyTorch-based deep learning library for sequence data. *Nature Methods*, *16*, 315. http://dx.doi.org/10.1038/s41592-019-0360-8.

Cheng, L., Khalitov, R., Yu, T., Zhang, J., & Yang, Z. (2023). Classification of long sequential data using circular dilated convolutional neural networks. *Neurocomputing*, *518*, 50–59.

Cho, K. W. (2012). Enhancers. *Wiley Interdisciplinary Reviews: Developmental Biology*, *1*(4), 469–478.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

Gunasekaran, H., Ramalakshmi, K., Rex Macedo Arokiaraj, A., Deepa Kanmani, S., Venkatesan, C., & Suresh Gnana Dhas, C. (2021). Analysis of DNA sequence classification using CNN and hybrid models. *Computational and Mathematical Methods in Medicine*, *2021*.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., & Girshick, R. (2022). Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 16000–16009).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Ji, Y., Zhou, Z., Liu, H., & Davuluri, R. V. (2021). DNABERT: pre-trained bidirectional encoder representations from Transformers model for DNA-language in genome. *Bioinformatics*, *37*(15), 2112–2120.

Kalchbrenner, N., Espeholt, L., Simonyan, K., Oord, A. v. d., Graves, A., & Kavukcuoglu, K. (2016). Neural machine translation in linear time. arXiv preprint arXiv:1610.10099.

Kayhan, O. S., & Gemert, J. C. v. (2020). On translation invariance in CNNs: Convolutional layers can exploit absolute spatial location. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 14274–14285).

Kelley, D. R. (2020). Cross-species regulatory sequence activity prediction. *PLoS Computational Biology*, *16*(7), Article e1008050.

Kelley, D. R., Reshef, Y. A., Bileschi, M., Belanger, D., McLean, C. Y., & Snoek, J. (2018). Sequential regulatory activity prediction across chromosomes with convolutional neural networks. *Genome Research*, *28*(5), 739–750.

Khalitov, R., Yu, T., Cheng, L., & Yang, Z. (2022). ChordMixer: A scalable neural attention model for sequences with different lengths. arXiv preprint arXiv:2206.05852.

Kumar, A., Raghunathan, A., Jones, R., Ma, T., & Liang, P. (2022). Fine-tuning can distort pretrained features and underperform out-of-distribution. arXiv preprint arXiv:2202.10054.

Lea, C., Flynn, M. D., Vidal, R., Reiter, A., & Hager, G. D. (2017). Temporal convolutional networks for action segmentation and detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 156–165).

Lee, D., Gorkin, D. U., Baker, M., Strober, B. J., Asoni, A. L., McCallion, A. S., et al. (2015). A method to predict the impact of regulatory variants from DNA sequence. *Nature Genetics*, *47*(8), 955–961.

Liu, Q., Fang, L., Yu, G., Wang, D., Xiao, C.-L., & Wang, K. (2019). Detection of DNA base modifications by deep recurrent neural network on Oxford Nanopore sequencing data. *Nature Communications*, *10*(1), 1–11.

Min, X., Ye, C., Liu, X., & Zeng, X. (2021). Predicting enhancer-promoter interactions by deep learning and matching heuristic. *Briefings in Bioinformatics*, *22*(4), bbaa254.

Mo, S., Fu, X., Hong, C., Chen, Y., Zheng, Y., Tang, X., et al. (2021). Multi-modal self-supervised pre-training for regulatory genome across cell types. arXiv preprint arXiv:2110.05231.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., et al. (2016). Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499.

Rizzo, R., Fiannaca, A., La Rosa, M., & Urso, A. (2015). A deep learning approach to DNA sequence classification. In *International meeting on computational intelligence methods for bioinformatics and biostatistics* (pp. 129–140). Springer.

Sethi, A., Gu, M., Gumusgoz, E., Chan, L., Yan, K.-K., Rozowsky, J., et al. (2020). Supervised enhancer prediction with epigenetic pattern recognition and targeted validation. *Nature Methods*, *17*(8), 807–814.

Shrikumar, A., Greenside, P., & Kundaje, A. (2017). Learning important features through propagating activation differences. In *International conference on machine learning* (pp. 3145–3153). PMLR.

Själander, M., Jahre, M., Tufte, G., & Reissmann, N. (2019). EPIC: An energy-efficient, high-performance GPGPU computing research infrastructure. arXiv:1912.05848.

Taylor, W. L. (1953). "Cloze procedure": A new tool for measuring readability. *Journalism Quarterly*, *30*(4), 415–433.

Wang, Z., Tan, X., Li, B., Liu, Y., Shao, Q., Li, Z., et al. (2021). BindTransNet: A transferable Transformer-based architecture for cross-cell type DNA-protein binding sites prediction. In *Bioinformatics research and applications* (pp. 203–214).

Wettig, A., Gao, T., Zhong, Z., & Chen, D. (2022). Should you mask 15% in masked language modeling? arXiv preprint arXiv:2202.08005.

Yang, B., Liu, F., Ren, C., Ouyang, Z., Xie, Z., Bo, X., et al. (2017). BiRen: Predicting enhancers with a deep-learning-based model using the DNA sequence alone. *Bioinformatics*, *33*(13), 1930–1936.

Yu, T., Khalitov, R., Cheng, L., & Yang, Z. (2022). Paramixer: Parameterizing mixing links in sparse factors works better than dot-product self-attention. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 691–700).

Zhang, Y., Qiao, S., Ji, S., & Li, Y. (2020). DeepSite: bidirectional LSTM and CNN models for predicting DNA–protein binding. *International Journal of Machine Learning and Cybernetics*, *11*(4), 841–851.

Zhao, H., Tu, Z., Liu, Y., Zong, Z., Li, J., Liu, H., et al. (2021). PlantDeepSEA, a deep learning-based web service to predict the regulatory effects of genomic variants in plants. *Nucleic Acids Research*, *49*(W1), W523–W529.

Zhou, J., & Troyanskaya, O. G. (2015). Predicting effects of noncoding variants with deep learning–based sequence model. *Nature Methods*, *12*(10), 931–934.

Zhuang, Z., Shen, X., & Pan, W. (2019). A simple convolutional neural network for prediction of enhancer–promoter interactions with DNA sequence data. *Bioinformatics*, *35*(17), 2899–2906.

# Self-supervised Learning for DNA Sequences with Circular Dilated Convolutional Networks (supplemental document)

Lei Cheng*, Tong Yu*, Ruslan Khalitov, and Zhirong Yang**

*Department of Computer Science, Norwegian University of Science and Technology, Norway*

**Abstract**

In this supplemental document, we provide more details of the datasets and hyperparameters.

## 1. Human

### *1.1. Data Description*

In our experiment on the variant effect prediction task, we used a dataset consisting of 97,992 instances across 49 tissues. We randomly split the data into 80% training, 10% validation, and 10% test sets. The DNA sequences are extracted from GRCh38 to construct both the pretraining dataset and the downstream task.

### *1.2. Hyperparameters*

The hyperparameters of our model and the compared models are shown in Table 1. We maintained a constant batch size across all models and used a mask ratio of 30% during the pretraining stage. Our masking strategy included random masking, replace masking, and no masking in an 80-10-10 ratio. We ran the pretraining for 100 epochs with a cosine scheduler, using a maximum learning rate of 0.003.

We used different hidden sizes for the encoder and decoder according to the sequence length. For example, we used 16 for the encoder and 8 for the decoder when pretraining on 1kbp sequences. The dimensionality of the decoder is set to half of the encoder to keep it lightweight. In the fine-tuning stage, we ran the model for 100 epochs with a learning rate of 0.0003. The model first loaded the pretrained encoder and then a CDIL predictor was fine-tuned. The hidden dimensionality of the CDIL predictor is also set to half of the pretrained encoder.

## 2. Plants

### *2.1. Data Description*

In our experiment of OCR prediction, we used two data size configurations, 1% (64,000 sequences from maize and 51,200 sequences from other species) and 100% (6,400,000 sequences from maize and 5,120,000 sequences from other species). Following the strategy of Zhao et al. (2021), we selected 1 or 2 chromosomes as the validation or test set and excluded them from the train set. For each plant, we made predictions about multiple tissues. Table 2 summarizes the details of different plants.

### *2.2. Hyperparameters*

All plants achieved the best average AUROC on downstream tasks when the masking ratio was either 10% or 20%, and we thus selected 15% for all plants. Random masking, replace masking, and no masking followed an 80-10-10 split. We ran the pretraining for 50 epochs for the 1% data size configuration and 5 epochs for the 100% data size configuration. More hyperparameters are shown in Table 3.

## References

Zhao, H., Tu, Z., Liu, Y., Zong, Z., Li, J., Liu, H., Xiong, F., Zhan, J., Hu, X., Xie, W., 2021. PlantDeepSEA, a deep learning-based web service to predict the regulatory effects of genomic variants in plants. Nucleic Acids Research 49, W523–W529.

*Equal contribution.

**Corresponding author, zhirong.yang@ntnu.no

Table 1: Hyperparameter details for every model on variant effect prediction task. *N*, *L*, *B*, *H*, *E*, and *lr* refer to max sequence length, layer number, batch size, hidden state size, running epoch, and learning rate, respectively. The first value of *H* is the dimensionality of the encoder while the second value is the dimensionality of the decoder.

| Model | *N* | L | n_heads | *B* | *H* | pos_embed | *E* | *lr* |
|---|---|---|---|---|---|---|---|---|
| Transformer | 1kbp | 4 | 4 | 8 | 16/8 | True | 100 | 0.0003 |
| Performer | 1kbp | 4 | 4 | 8 | 16/8 | True | 100 | 0.0003 |
| Nyströmformer | 1kbp | 4 | 4 | 8 | 16/8 | True | 100 | 0.0003 |
| CDIL | 1kbp | 9 | - | 8 | 16/8 | False | 100 | 0.0003 |
| CDIL | 2kbp | 10 | - | 8 | 16/8 | False | 100 | 0.0003 |
| CDIL | 3kbp | 11 | - | 8 | 16/8 | False | 100 | 0.0003 |
| CDIL | 5kbp | 12 | - | 8 | 32/16 | False | 100 | 0.0003 |
| CDIL | 10kbp | 13 | - | 8 | 64/32 | False | 100 | 0.0003 |

Table 2: Summary of plants. #Training, #Testing, and #Feature refer to the number of training sequences, the number of validation/testing sequences, and the number of features from multiple tissues, respectively.

| Species | #Training(1%) | #Training(100%) | #Testing | #Feature |
|---|---|---|---|---|
| *A.thaliana* | 51200 | 5120000 | 9984 | 19 |
| *B.distachyon* | 51200 | 5120000 | 14848 | 9 |
| *O.sativa-MH* | 51200 | 5120000 | 14848 | 15 |
| *O.sativa-ZS* | 51200 | 5120000 | 14848 | 15 |
| *S.italica* | 51200 | 5120000 | 19968 | 9 |
| *S.bicolor* | 51200 | 5120000 | 29952 | 14 |
| *Z.mays* | 64000 | 6400000 | 79872 | 19 |

Table 3: Hyperparameters of every model on OCR prediction task. *N*, *L*, *H*, *B*, *E*, and *lr* refer to the sequence length, the number of layers, hidden states size, batch size, running epoch, and learning rate, respectively. PlantDeepSEA increases the hidden size as layers are stacked. The two parts of *L* and *H* refer to hyperparameters of the encoder and decoder or supervised predictor, respectively. The two parts of *E* and *lr* refer to hyperparameters for w/o pretraining and w/ pretraining, respectively.

| data size | Model | *N* | *L* | *H* | n_heads | pos_embed | *B* | *E* | *lr* |
|---|---|---|---|---|---|---|---|---|---|
| 100% | CDIL | 1kbp | 9/9 | 128/32 | - | False | 256 | 5/2 | 0.001/0.0001 |
| | PlantDeepSEA | 1kbp | 6 | 320-480-960 | - | False | 256 | 5 | 0.001 |
| 1% | CDIL | 1kbp | 9/9 | 128/32 | - | False | 256 | 50/20 | 0.001/0.0001 |
| | PlantDeepSEA | 1kbp | 6 | 320-480-960 | - | False | 256 | 50 | 0.001 |
| | Nyströmformer | 1kbp | 4/2 | 128/128 | 4 | True | 64 | 50/20 | 0.0001/0.0001 |
| | Performer | 1kbp | 4/2 | 128/128 | 4 | True | 64 | 50 | 0.0001 |
| | Transformer | 1kbp | 4/2 | 128/128 | 4 | True | 64 | 50 | 0.0001 |
| | CDIL | 1kbp | 9/9 | 128/32 | - | False | 8 | 15/5 | 0.001/0.0001 |
| | | 2kbp | 10/10 | 128/32 | - | False | 8 | 15/5 | 0.001/0.0001 |
| | | 3kbp | 11/11 | 128/32 | - | False | 8 | 15/5 | 0.001/0.0001 |
| | | 5kbp | 12/12 | 128/32 | - | False | 8 | 15/5 | 0.001/0.0001 |
| | | 10kbp | 13/13 | 128/32 | - | False | 8 | 15/5 | 0.001/0.0001 |

ISBN 978-82-326-8600-1 (printed ver.)
ISBN 978-82-326-8599-8 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)





NTNU

Norwegian University of
Science and Technology