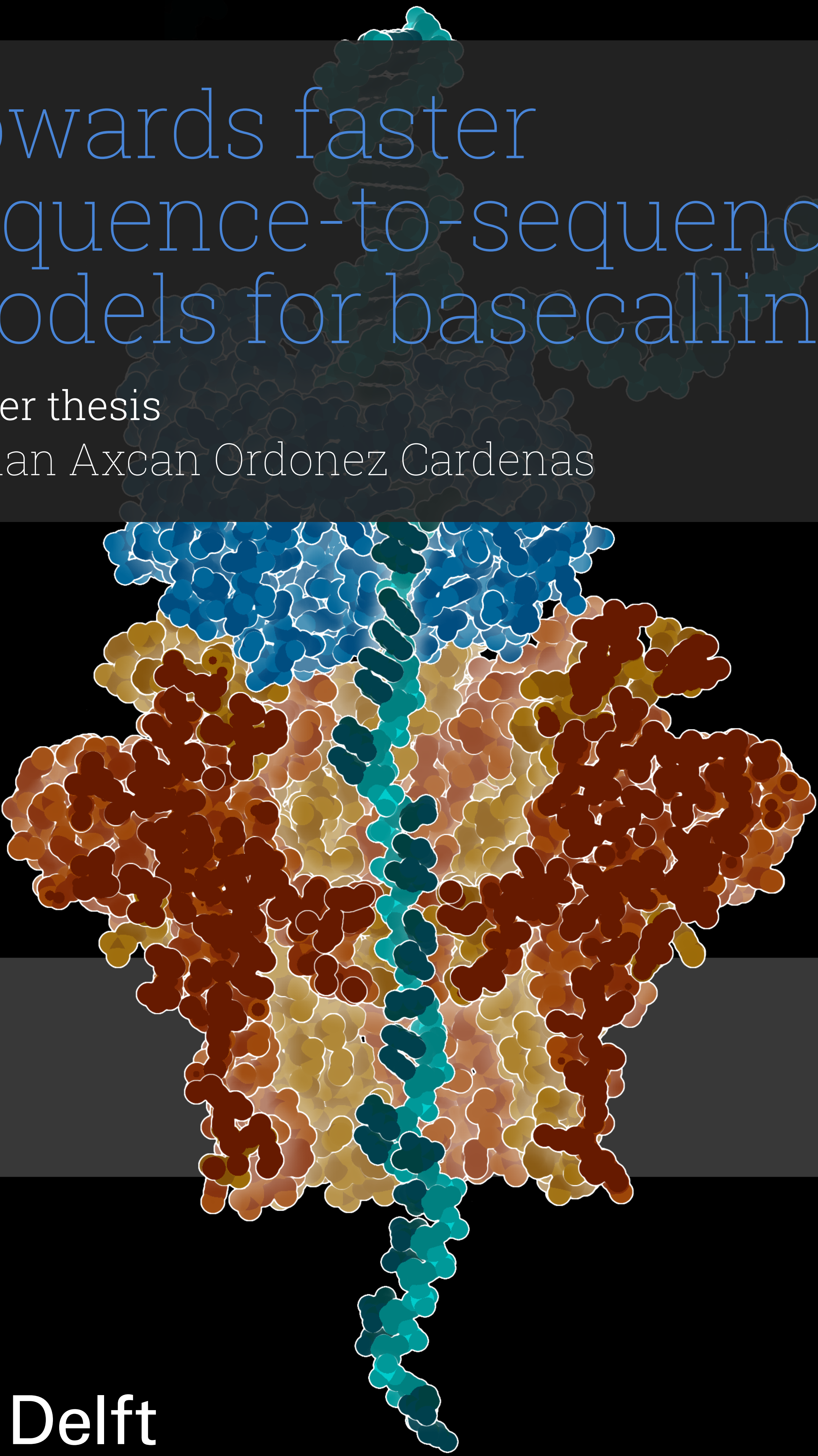
Towards faster sequence-to-sequence models for basecalling

Master thesis

Nathan Axcan Ordonez Cardenas

Delft University of Technology



TU Delft

Towards faster sequence-to-sequence models for basecalling

by

Nathan Axcan Ordonez Cardenas

Student Name	Student Number
Ordonez	4936205

Supervisor:	Zaid Al Ars
Co-supervisor:	Peter Hofstee
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

TU Delft

Preface

This thesis started with a very broad question as seems customary when one becomes a thesis student under Zaid. Indeed, the approach here is to tackle a problem with big eyes and an open mind. After being overwhelmed by the number of interesting ideas in deep learning's sequence modeling research, I worked on developing my sense of time and judgement. The need for these skills became especially apparent when, after having created a veritable mountain of data, it had to be sifted and moulded into a hopefully useful thesis. It is like having built a large number of paths and explorations through a forest, according to which one must choose which shall become highways, and which shall be forgotten.

I would like to thank my parents for every form of support possible, and for giving me precious opportunities. In particular, my mother for teaching me to always search for depth in reflection, and my father who taught me the value of structure and correctness in my own thoughts.

My many friends, in particular Bo Bakker who inspired a fixed, determined stare into complexity. My friends Ulf Torsten Kemmsies, Jaden Nierop, Stanislas Vuille-dit-Bille, Kamron Geijsen, Martin Verweij, all inspiring me in their special ways, and my girlfriend Enija for bringing me motivation and a greater sense of purpose.

Ecole Sofia in Lausanne and especially their directors, current and past, for giving me the possibility to pursue higher education. TU Delft for offering high quality education and an environment in which one can aim high and find success.

Peter Hofstee for being an inspiration and an example of noble academic values, Zaid Al-Ars for showing me how a supervisor can truly maximise a student's creativity and inspiration, and many others I've encountered during my education. All were important for me to reach this end of my official education, so that I can enter the next stage of life-long learning.

Nathan Axcan Ordonez Cardenas
Delft, November 2024

Summary

The rapid advancements in nanopore sequencing technology have revolutionized the field of genomics, enabling cost-effective and high-throughput long-read DNA sequencing. However, the basecalling process, which involves translating the raw electrical signals generated by nanopores into nucleotide sequences, remains a computational bottleneck. This thesis explores the potential of recent sequence-to-sequence (s2s) models from the deep learning literature to improve the accuracy and throughput trade-off in basecalling.

We begin by providing a comprehensive background on the complexity of nanopore sequencing, delving into the physical process and the challenges associated with mapping nanopore signals to DNA bases. We then introduce and compare various s2s models, including recurrent neural networks (RNNs), trans- formers, and state-space models (SSMs), highlighting their computational properties and suitability for the basecalling task. Additionally, we discuss techniques for producing variable-length outputs, such as conditional random fields (CRFs) and connectionist temporal classification (CTC), and consider hardware performance aspects crucial for efficient basecalling.

To gain a deeper understanding of basecalling, we conduct experiments to illustrate the properties of basecalling data and investigate the reasons behind the superior performance of long short-term memory (LSTM) networks in this domain. Through ablation studies and interpretability analysis, we uncover key insights into the architectural components that contribute to the effectiveness of RNNs for basecalling.

Building upon these findings, we explore a wide range of s2s models and propose novel architectures tailored for basecalling. We introduce the ParallelRNN, a parallel formulation of RNNs that leverages the locality of information in basecalling to achieve high throughput without compromising accuracy. Additionally, we present DenseBaseConv, a convolutional architecture designed to focus on learning the signal function and capture local dependencies efficiently.

Extensive experiments on large-scale datasets demonstrate the potential of our proposed models to push the Pareto frontier of throughput-accuracy trade-offs in basecalling. We showcase the impact of distillation techniques in enhancing the performance of existing models and highlight the competitive performance of ParallelRNN and DenseBaseConv compared to state-of-the-art basecallers. Our analysis also sheds light on the scalability challenges and the importance of custom kernel implementations for fully realizing the potential of these architectures.

This thesis makes significant contributions to the field of basecalling by providing a comprehensive overview of modern s2s models, introducing novel architectures tailored for high-throughput base- calling, and offering valuable insights into the computational properties and optimization strategies for efficient basecalling. Our findings pave the way for future research on developing even more accurate and computationally efficient basecallers, ultimately accelerating the progress in genomics and enabling groundbreaking discoveries in various domains, from personalized medicine to evolutionary biology.

Contents

1 Introduction

1.1. Context

The study of DNA is fundamental to answering what has been called ”one of the oldest questions in science” [58], namely that of nature and nurture. Genomics in particular is a fruitful and increasingly popular area of study that has added to humanity's informational heritage a novel deep and functional understanding of, broadly, life on our planet, from pathological patterns such as cancers, genetic diseases, to healthy ones such as GMOs and regenerative medicine.

A data that is essential to the study of DNA is the exact nucleotide sequence that forms a genome. Being able to read it at a relatively low cost has largely helped the aforementioned discoveries, but for a long time this has only been cost-effective when doing short-read sequencing, where short and disconnected parts of a genome are read, or genotyping, where only specific gene variants are measured. A more valuable data is the collection of the whole genome of a living being. This technique is called long-read sequencing. The advent of long-read DNA sequencers developed by i.e. Oxford Nanopore Technologies over the past decade represents a new large step in whole-genome sequencing because it is multiple orders of magnitude cheaper.

Long-read sequencing can be subdivided into two phases: producing signals that contain information potentially revealing a DNA sequence using chemical processes, and analyzing the signal to extract the underlying DNA sequence into a digital representation. This analysis, depicted in figure 1.1 consists of the recognition of bases from the raw signals, called basecalling, and a series of post-processing steps that combine the basecalling data into a highest-probability final sequence.

While both phases are important to the accuracy of long-read sequencing, recent findings indicate that the main challenges are in the basecalling part. Due to a multitude of factors including noise and the non-linearity of the relationship between the signal and the bases, only probabilistic models have achieved a high enough accuracy in basecalling to be useful for genomics research, in particular deep learning models, and while they have achieved significant accuracy levels, the best models have high computational requirements that can limit their usefulness. Different trade-offs of throughput (amount of signal processed per second) and accuracy are used by scientists depending on their use of the output. Specifically, current state-of-the-art models are potentially limited by their reliance on the LSTM architecture, which has unfavorable computational characteristics. Currently, basecalling has the longest duration part of the process in long-read genomics analysis workflows while additionally requiring expensive high-end compute hardware to run those probabilistic models.

To illustrate the computational requirements of long-read sequencing, we take the example use-case of reading DNA bases in real time. This would allow scientists to have fully-basecalled data right when the machine is done producing signals. For this, we take two popular long-read sequencing machines, Oxford Nanopore's MinION and PromethION. Both read DNA at the same speed per DNA strand, 400 bases per second. On average, we measure 8 electrical current samples per base, so the machine produces 3200 current samples per second. But because these machines have multiple

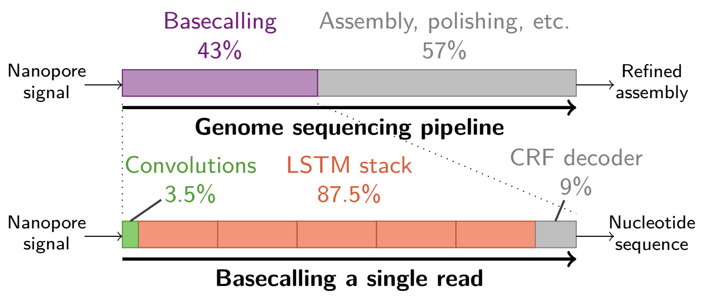


Figure 1.1: Top: basecalling takes 43% of the time in a nanopore genome sequencing pipeline. Bottom: during basecalling, almost 90% of the computing time is spent computing the LSTM layers' outputs. Illustration from [26].

nanopores that can read separate DNA strands in parallel, the MinION produces $512 \times 3200 \approx 1.6$M samples per second. PromethION generates $128'400 \times 3200 \approx 410$M million samples per second. A deep learning model attempting to basecall a running PromethION machine at its maximum throughput configuration has to process 128'400 current samples within a budget of $1/3200 = 312$μs. This high sample rate necessitates a software-hardware compute solution that is able to sustain the required compute throughput.

At the same time, recent developments in sequence-to-sequence (s2s) modeling present new avenues for potential improvements in basecalling accuracy and efficiency. Variations of the transformer architecture, originally designed for natural language processing tasks, have been developed to maintain its abilities while enhancing its hardware performance. Additionally, novel LSTM architectures and even models of a completely different nature, such as state-space models, have emerged as potential alternatives to traditional approaches. These advancements in deep learning architectures call for a thorough investigation into their applicability and potential benefits in the context of basecalling for long-read sequencing.

This thesis explores the potential of these new deep learning sequence-to-sequence models to accelerate the basecalling process. By examining various architectures, including transformer variants, advanced LSTM designs, and state-space models, we aim to identify novel approaches that could significantly improve both the speed and accuracy of basecalling. The research focuses on investigating those new sequence models and their potential for building faster, and more accurate, basecallers.

1.2. Problem definition and research questions

The main problem this thesis tackles is the lack of understanding of the impact of recent sequence-to-sequence models on the task of basecalling.

To address this problem, we focus on answering the following research questions:

1. What are the recent developments in efficient sequence-to-sequence modeling in deep learning?
2. What is the impact of recent developments in efficient sequence-to-sequence modeling on improving the accuracy/throughput trade-off, or the accuracy/latency trade-off for the task of basecalling?

1.3. Thesis outline

This thesis is organized as follows:

- **Chapter 1: Introduction**
  This chapter introduces the concept of basecalling, providing a description of the problem.
- **Chapter 2: Background**
  This chapter introduces basic notions that will be used throughout the thesis, including a theoretical analysis of the process of basecalling, the multiple sequence-to-sequence models from the deep learning literature, and important concepts concerning the hardware we will run benchmarks on.

- **Chapter 3: Understanding basecalling**
  Here, we present experiments illustrating the properties of basecalling data, investigate why LSTMs perform well in this task, and analyze state-of-the-art models using interpretability tools. A preliminary comparison of different sequence-to-sequence (s2s) models for basecalling acceleration is also included.
- **Chapter 4: Sequence to sequence models**
  We explore various candidate sequence-to-sequence methods and compare their computational properties.
- **Chapter 5: Novel architectures for basecalling**
  This chapter details the final candidate models chosen for evaluation based on throughput/accuracy and latency/accuracy trade-offs. We present the reasoning behind their selection and the experiments conducted for their development.
- **Chapter 6: Overall experimental results**
  We study the accuracy and throughput properties of the final models, comparing them with existing LSTM and Transformer-based models.
- **Chapter 7: Conclusions and future work**
  We wrap up the thesis with general conclusions from the paper, along with avenues for future research.

1.4. Contributions

In the interest of finding faster basecallers, this thesis made the following contributions:

- An overview of novel sequence-to-sequence neural network architectures
- A reproduction of LSTM and transformer-based state-of-the-art basecallers
- A detailed ablation study of state-of-the-art recurrent architecture
- Introduction of a new kind of sequence-to-sequence model block called the Markov model
- An analysis of the accuracy/throughput tradeoffs offered by selected sequence-to-sequence models

2 Background

In this section, we introduce important background knowledge that is relevant for the rest of this thesis. In general, they relate to the different stages of a basecaller as depicted in figure 2.1. Specifically, section 2.1 corresponds to the original signal, section 2.2 to the encoder, and 2.3 the decoder. Section 2.4 discusses the hardware considerations we must take into account given that this thesis involves the extensive use of performance benchmarks, which depend on the hardware involved.

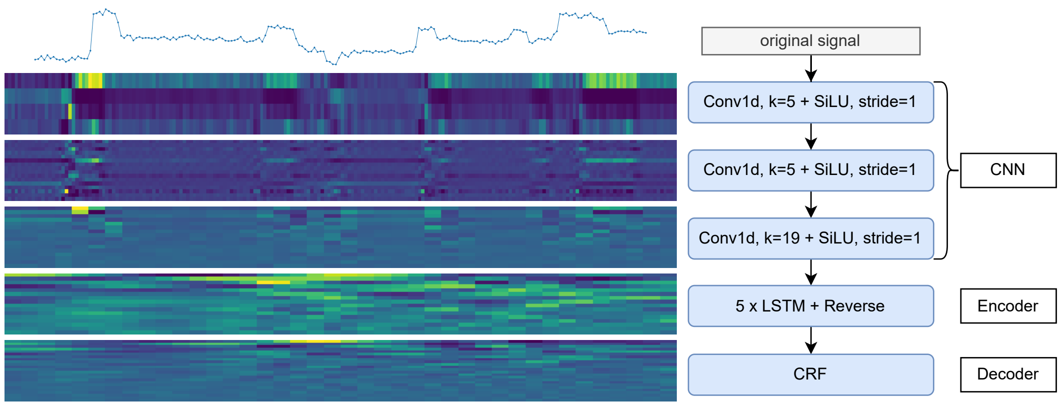

original signal
Conv1d, k=5 + SiLU, stride=1
Conv1d, k=5 + SiLU, stride=1
Conv1d, k=19 + SiLU, stride=1
CNN
5 x LSTM + Reverse
Encoder
CRF
Decoder

Figure 2.1: The architecture of the Bonito basecaller as implemented in [61], along with example outputs of the different layers with a given input sequence on the left column, the basic structure of the neural network in the middle column, and in the right column the basic structure which the different basecallers present in this thesis have in common (CNN, Encoder and Decoder). The model has dimensionality 96 and was trained for 5 epochs on the 52GB dataset. The outputs of the third convolution, the LSTM and CRF show lower-dimensionality vectors obtained through the PCA method [23].

2.1. Complexity of nanopore sequencing

To achieve better accuracy and throughput trade-offs in basecalling, it is crucial to identify and incorporate inductive priors based on the intrinsic nature of the input data. One example of this is the Xception architecture [12] which was found to be on the Pareto front of accuracy and throughput for the task of image recognition in a paper performing a comprehensive benchmark analysis of models designed for this task (see figure 3 of [7]). Their key insight was built upon previous work proposing to modify the model's layers such that correlations between different locations in the image, and correlations between different features extracted by the image at one location in the image, are separated into two layers. In this section, we describe the source of nanopore current signals and examine the challenges in mapping these signals to their corresponding DNA bases from its physical process. Therefore, this section describes a physical model for the process of basecalling, and section 3.1 presents a statistical model.

2.1.1. Overview of nanopore sequencing

At its core, basecalling aims to associate sequences of electrical current measurements (which we will call the signal) with the four DNA bases (A, C, G, and T). Ideally, each base would generate a characteristic current level, enabling a straightforward one-to-one mapping between current level and bases. Such as scenario would allow simple intervals of the signal intensity to be defined for every base, and the current to be converted to a list of bases. A visualization of the physical process of basecalling is shown in figure 2.2 to clarify how the DNA strand moves along the nanopore protein, blocking the flow of ions and producing a measurable perturbation to the measured electrical current.

However, real basecalling is more complex due to factors such as the presence of signal noise, temporal fluctuations inherent to nanopore sequencing, and the length of the nanopore itself, which causes multiple bases to be present in the nanopore at the same time such that they affect the signal together (this is denoted as a k-mer nanopore for k bases present in the nanopore at the same time). There are efforts to model the creation of this signal (by designing simulators), meaning they generate a realistic signal from a base sequence. Squigulator [27], which to the best of our knowledge is the state of the art in generating these simulations without the use of deep learning, models signal variation using temporal fluctuations and amplitude noise exclusively to achieve realistic signal generation, and proceed to show how they affect a trained basecaller's accuracy in realistic ways, so that for example when one kind of noise is removed, the basecaller shows better accuracy than on real basecalling data even though it was trained on real basecalling data.

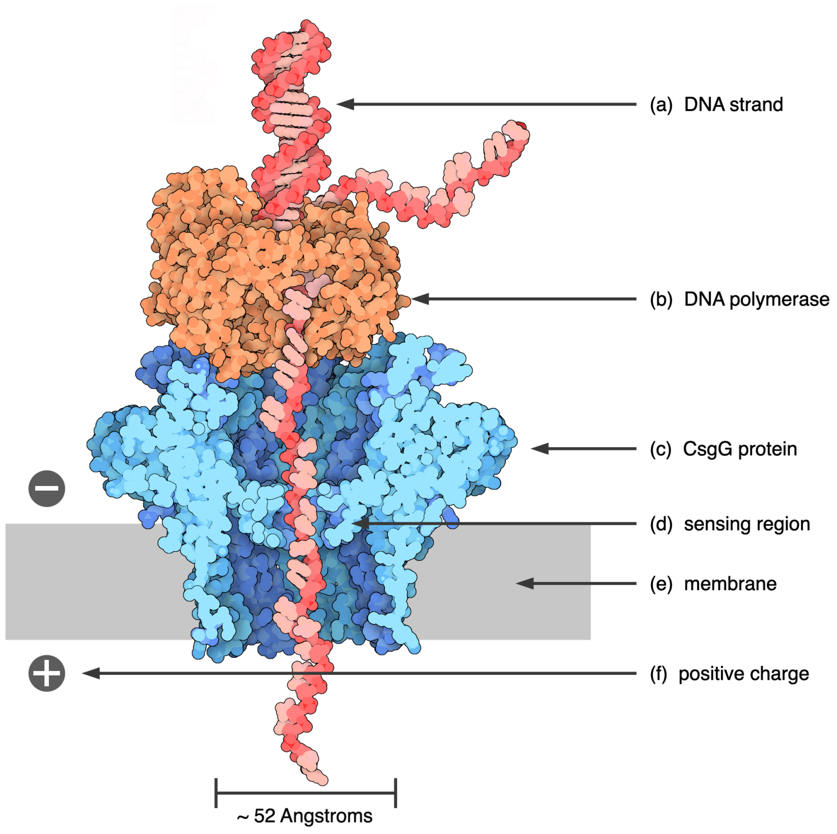


Figure 2.2: Depiction of the structure of a nanopore containing all essential components. (a) is the DNA strand, split into half by (b) the polymerase, which also serves to slow down the speed at which DNA goes through the (c) CsgG protein, a popular protein for nanopore sequencing. As charged ions go through the about 12 angstrom-wide (d) sensing region, the current is perturbed by the DNA strand, by an amount that depends on the specific bases present in the sensing region. The charged ions cannot go around the protein due to the (e) membrane, and are pulled through by an induced (f) electrical potential, or voltage. Measurements from [9], illustration modified from [43].

2.1.2. Nanopore sequencing as a complex deep learning task

As different deep learning tasks have different levels of complexity, so they have different requirements in terms of the neural network architectures that can best solve them (i.e. [21] showing a transformer-based model that captures long dependencies in images better than previous CNN-based models, [71] showing theoretical representational limitations in neural networks for a given depth). One way tasks like image classification or language modeling have accounted for those requirements is by increasing the model size, however in a task like basecalling where the desired throughput is on the order of millions of inputs per second, which generally limits the parameter count of the basecaller, neural network architectures with fundamentally different computational characteristics and types of patterns that can be recognised will lead to different accuracies at the task at hand. There have been previous work on the differing abilities of multiple sequence-to-sequence neural networks to model sequences of varying levels of complexity. One work uses the Chomsky hierarchy, which is a classification of languages that can be generated according to a series of rules (denoted as formal languages) to represent different levels of complexity in a dataset. It explored how single layers of the aforementioned neural networks (transformer, LSTM) fundamentally differ in their ability to model character sequences belonging to those languages, depending on the language's location in the hierarchy. This revealed that RNN models are able to represent more complex formal languages than transformer models[20]. Further work explored adding multiple layers, and found that with multi-layer neural networks, the number of layers of a model determines its ability to recognize more complex languages [56]. This is pertinent to our work because if we frame sequence-to-sequence models as inductive biases (since they are subsets of a dense and deep neural network taking the whole sequence as an input), the findings strongly suggest that those biases are key to achieving low layer count, and therefore greater efficiency in modeling complex data.

To better understand the question of what results into this signal, we refer to a paper describing a process using the physics-based finite element method to simulate basecalling at high temporal and spatial resolution [78]. Specifically, we examine the Poisson-Nernst-Planck-Stokes (PNPS) equations used to describe the signal:

$$\nabla \cdot (\epsilon_0 \epsilon_r \nabla \phi) = -(\rho^f_{\text{pore}} + \rho_{\text{ion}}) \tag{2.1}$$

$$\frac{\partial c_i}{\partial t} = -\nabla \cdot (\mathcal{D}_i \nabla c_i + z_i \mu_i c_i \nabla \varphi - \boldsymbol{u} c_i + \mathcal{D}_i \boldsymbol{\beta}_i c_i) \tag{2.2}$$

$$\rho \left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} \right) = -\nabla p + \eta \nabla^2 \boldsymbol{u} + \boldsymbol{F}_{\text{ion}} \tag{2.3}$$

Equation (2.1) is Poisson's equation relating the electric potential ϕ to the fixed charge density of the nanopore ρ^f_{pore} and the mobile ion charge density ρ_{ion}. Equation (2.2) is the Nernst-Planck equation describing the transport of ions with concentration c_i, where $\mathcal{D}_i$ is the diffusion coefficient, μ_i is the electrophoretic mobility, $\boldsymbol{u}$ is the fluid velocity, and $\boldsymbol{\beta}_i$ accounts for steric effects. Equation (2.3) represents the Navier-Stokes equations for fluid flow, with pressure p, viscosity η, and the ionic body force $\boldsymbol{F}_{\text{ion}}$.

The simulated ionic current I_{sim} through the nanopore is obtained by integrating the total ionic flux across the reservoir boundary S:

$$I_{\text{sim}} = \mathscr{F} \int_S \sum_i z_i \boldsymbol{J}_i \cdot \boldsymbol{n} \, dS \tag{2.4}$$

where $\mathscr{F}$ is Faraday's constant, z_i is the charge number of ion i, $\boldsymbol{J}_i$ is the ionic flux, and $\boldsymbol{n}$ is the unit normal vector to the surface S. This equation captures the contributions from diffusion, electrophoresis, convection, and steric effects, as well as the influence of the electrostatic potential and fluid flow, providing an accurate simulation of the ionic current in nanopore sequencing.

Upon closer examination of the terms in the PNPS equations, we can identify several potential sources of non-linearity in the current signal:

1. The Poisson equation establishes a relationship between the electric potential and the charge density, encompassing both the fixed charges of the nanopore and the mobile ion charges. The non-linear coupling between the potential and ion concentrations can give rise to intricate current patterns.

2. The Nernst-Planck equation describes the transport of ions, which is governed by the electric field and concentration gradients. The interplay between these factors can result in non-linear ion fluxes.

3. The Navier-Stokes equations dictate the fluid flow, driven by pressure gradients and body forces such as the electric field acting on the ions. The presence of the convective term $(\mathbf{u} \cdot \nabla \mathbf{u})$ introduces non-linearity into the fluid dynamics.

The existence of these non-linear terms in the PNPS equations suggests that the mapping between DNA bases and current signals is likely to be non-linear and context-dependent. Implicit in these equations is also the impact of geometry, which not only causes disturbances in the signal along the time dimension, but also can cause changes in the current depending on the specific position of the DNA strand within the nanopore.

2.2. Sequence to sequence models

Multiple models exist in the literature to model sequences, in this section we introduce the relevant equations and details to understand how they work. Further comparison in terms of accuracy at basecalling and maximum throughput can be found in Chapter 4, in figure 4.1.

2.2.1. Recurrent Neural Networks (RNNs)

Recurrent Neural Networks are a class of neural networks that are particularly well-suited for processing sequential data. Unlike feedforward neural networks, RNNs possess an internal state that allows them to capture temporal dependencies within the data.

Elman Recurrent Neural Networks (Elman RNNs)

Elman Recurrent Neural Networks [22] are one of the simplest forms of RNNs. They are composed of an input layer, a recurrent hidden layer that captures the temporal dependencies by maintaining an internal state, and an output layer. The network's equations are given by:

$$h_t = \tanh(W_h x_t + U_h h_{t-1} + b_h) \tag{2.5}$$
$$y_t = \phi(W_y h_t + b_y) \tag{2.6}$$

where $\tanh$ and ϕ denote the activation functions, and W_h, U_h, b_h, W_y, and b_y are parameters that need to be learned during training. While Elman RNNs can capture temporal dependencies, they often struggle with long-term dependencies due to their gradients either vanishing or exploding (this was called vanishing or exploding error flow in the original LSTM paper [37]). What is meant by this is that because the same weight matrix is applied at every iteration, the errors it accumulates from every timestep (due to the fact that every output depends on all previous inputs) increases or decreases over the sequence length in a trivial way (depending on distance in the sequence dimension, instead of depending on both distance and the specific vectors of the input sequence). Exploding gradients can cause large gradients that, when applied, cause the model's accuracy to collapse, and its loss to spike. Vanishing gradients can stop the RNN from learning longer distance relationships in the data.

Long short-term memory networks

Long Short-Term Memory (LSTM) networks [37] are a specific type of RNN designed to address the vanishing gradient problem that plagues vanilla RNNs (see previous section on Elman RNNs). LSTMs introduce a memory cell c_t and three gates: input gate i_t, forget gate f_t, and output gate o_t. The network's equations are given by:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{2.7}$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{2.8}$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{2.9}$$
$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{2.10}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{2.11}$$
$$h_t = o_t \odot \tanh(c_t) \tag{2.12}$$

where σ denotes the sigmoid activation function and $\odot$ represents the element-wise product. The gating mechanism of LSTMs equips them to learn long-term dependencies, making them effective for basecalling tasks.

The LSTM architecture includes three crucial gates that regulate the flow of information through the network, enabling it to capture long-term dependencies. The input gate i_t (Equation 2.8) controls how much of the new information from the current input x_t and the previous hidden state h_{t-1} is stored in the cell state c_t. The forget gate f_t (Equation 2.7) determines the extent to which the previous cell state c_{t-1} is forgotten, thereby allowing the network to discard irrelevant information. The output gate o_t (Equation 2.9) decides how much of the cell state c_t should be exposed to the next hidden state h_t, which affects the output for the current time step. These gates work together to update the cell state c_t and hidden state h_t, ensuring that important information is retained over long sequences while irrelevant data is discarded. The candidate cell state $\tilde{c}_t$ (Equation 2.10) represents the new candidate values for the cell state, which are added to the cell state based on the input and forget gate values.

Gated recurrent units

Gated Recurrent Unit (GRU) networks [11] are a variation of RNN designed to address the vanishing gradient problem while simplifying the LSTM architecture. GRU networks combine the forget and input gates into a single update gate, and merge the cell state and hidden state. The network's equations are given by:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{2.13}$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{2.14}$$
$$\hat{h}_t = \phi(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \tag{2.15}$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t \tag{2.16}$$

where σ denotes the sigmoid activation function, ϕ denotes the hyperbolic tangent activation function, and $\odot$ represents the element-wise product. The GRU, with its simplified gating mechanism, is effective in learning long-term dependencies and offers an efficient alternative to LSTMs for basecalling tasks. The rest of the variables are:

- $x_t \in \mathbb{R}^d$: input vector
- $h_t \in \mathbb{R}^e$: output vector
- $\hat{h}_t \in \mathbb{R}^e$: candidate activation vector
- $z_t \in (0,1)^e$: update gate vector
- $r_t \in (0,1)^e$: reset gate vector

And the parameters:

- $W \in \mathbb{R}^{e\times d}$, $U \in \mathbb{R}^{e\times e}$, $b \in \mathbb{R}^e$: parameter matrices and vector which need to be learned during training.

In our case, following the Bonito architecture as in [61], we consider dimensionalities e and d to be equal. In basecalling, input vectors of the first GRU layer typically represents information about the signal, whereas output vectors represent probabilities of bases in the nanopore at that step in time. The candidate activation vector represents the addition of new information into the previous output vector, so we may consider the GRU as applying modifications to the previous step's output vector along the whole input sequence. The update gate vector determines how much of the candidate activation vector should replace information from the previous output vector. As the candidate activation vector represents new information to include in the current output vector, the reset gate filters the previous output's information before it is used to compute the current step's output vector.

Notice that instead of performing eight matrix multiplications (denoted as W and U matrices in equations 2.7 to 2.16), the GRU allows to only perform six, thereby reducing total floating points by about 25% since most of the FLOPs (Floating Point Operations) come from the matrix multiplications because its computational complexity (number of mathematical operations depending on the input dimensions) is quadratic for varying input dimensionalities, whereas all other operations have linear computational complexity.

2.2.2. Transformers

Transformers [75] have revolutionized the field of sequence modeling [2], particularly in natural language processing (NLP). They rely entirely on the self-attention mechanism to capture dependencies across sequences, thereby eliminating the need for recurrence.

Self-attention mechanism

The self-attention mechanism computes the attention weights to focus on specific parts of the input sequence:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.17}$$

Here, Q, K, and V are the query, key, and value matrices, respectively, and d_k is the dimensionality of the key vectors. The Transformer architecture stacks multiple layers of self-attention and feedforward neural networks, enabling it to model complex dependencies efficiently.

Note that the softmax operation, run over the sequence length dimension, means that the attention operation cannot be entirely parallelized and therefore creates a data dependency between all elements of the sequence, which is also why it is able to attend to any part of the input sequence that is available to it.

This attention operation is usually in a block, such as the encoder-decoder blocks from the original transformer paper [75], which involves applying the attention operation followed by a normalization in the batch dimension and residual connection, and then a two-layer MLP over the sequence length, followed by another normalization and residual connection. The residuals and normalizations allow gradients to flow through even very large numbers of these layers.

Local attention mechanism

Local attention [62, 14] is an efficient variation of the self-attention mechanism where each position in the sequence only attends to a limited subset of positions within a certain window around it, rather than the entire sequence. The motivation behind local attention is to reduce the computational and memory complexity that arises from the quadratic dependence on sequence length L in the standard self-attention mechanism.

In local attention, each query q_i for the i-th element in the sequence only attends to the key-value pairs (k_j, v_j) within a window of size w centered around i:

$$\text{Attention}^{\text{local}}(q_i, K, V) = \sum_{j=i-w/2}^{i+w/2} \alpha_{ij} v_j \tag{2.18}$$

$$\alpha_{ij} = \frac{\exp(q_i \cdot k_j / \sqrt{d_k})}{\sum_{j=i-w/2}^{i+w/2} \exp(q_i \cdot k_j / \sqrt{d_k})} \tag{2.19}$$

Here, α_{ij} represents the attention weight that indicates the importance of the j-th position to the i-th position. By focusing only on a local context given by the window size w, the computation and memory requirements are significantly reduced from $O(L^2)$ to $O(Lw)$, with w being much smaller than L.

This approach is beneficial for many practical tasks where long-range dependencies are either less critical or can be approximated through sequentially composed local interactions. Typical applications include natural language processing tasks involving very long documents or time-series data where recent observations are more relevant.

Reformer: The Efficient Transformer

The Reformer [45] is one of the many variations of the Transformer architecture designed to handle long sequences more efficiently. It introduces two key innovations: locality-sensitive hashing (LSH) attention and reversible residual layers. These innovations collectively reduce the memory and computational requirements of the Transformer model.

Locality-Sensitive Hashing (LSH) Attention Unlike the standard attention mechanism, which computes attention weights for all pairs of positions in the sequence, LSH attention limits interactions to only a subset of positions that are likely to be similar. This is achieved using locality-sensitive hashing.

Locality-sensitive hashing is a technique for efficiently finding approximate nearest neighbors in high-dimensional spaces. The LSH attention mechanism hashes the query and key vectors into buckets such that similar vectors are likely to be assigned to the same bucket. Attention weights are then computed only within these buckets rather than across the entire sequence:

$$h(x) = \text{hash function}(x) \tag{2.20}$$

$$\text{Attention}^{\text{LSH}}(Q, K, V) = \sum_{j \in \text{same bucket as } i} \alpha_{ij} v_j \tag{2.21}$$

$$\alpha_{ij} = \frac{\exp(q_i \cdot k_j / \sqrt{d_k})}{\sum_{j \in \text{same bucket as } i} \exp(q_i \cdot k_j / \sqrt{d_k})} \tag{2.22}$$

By changing the complexity from $O(L^2)$ to $O(L \log L)$, this mechanism offers a significant reduction in both memory and computational cost, making it feasible to handle much longer sequences.

Reversible Residual Layers The Reformer also incorporates reversible residual layers, following the design of reversible residual networks [28]. In a standard residual network, activations need to be stored for backpropagation, leading to large memory consumption. However, in reversible residual networks, the activations can be reconstructed from the outputs, obviating the need to store intermediate states:

$$Y_1 = X_1 + F(X_2) \tag{2.23}$$

$$Y_2 = X_2 + G(Y_1) \tag{2.24}$$

$$X_2 = Y_2 - G(Y_1) \tag{2.25}$$

$$X_1 = Y_1 - F(X_2) \tag{2.26}$$

By making the Transformer layers reversible, the Reformer reduces memory usage during training, as only one copy of activations needs to be stored at any time.

Combining LSH attention with reversible residual layers makes the Reformer a powerful and efficient model, capable of handling tasks that involve very long sequences while maintaining or even improving performance relative to traditional Transformer models.

2.2.3. State Space Models (SSMs)

State Space Models (SSMs) are a class of models that have recently gained attention in the field of sequence modeling. SSMs are based on the idea of modeling a system's behavior using a set of hidden state variables that evolve over time according to a set of dynamic equations. The Structured State Space (S4) model [32] and the Mamba model [31] are two notable examples of SSMs that have shown promising results in various sequence modeling tasks.

Structured State Space (S4) Model

The S4 model is based on the state space model (SSM) defined by the following equations:

$$x'(t) = Ax(t) + Bu(t) \tag{2.27}$$

$$y(t) = Cx(t) + Du(t) \tag{2.28}$$

where $x(t) \in \mathbb{R}^N$ is the hidden state, $u(t) \in \mathbb{R}$ is the input signal, $y(t) \in \mathbb{R}$ is the output signal, and $A \in \mathbb{R}^{N \times N}$, $B \in \mathbb{R}^{N \times 1}$, $C \in \mathbb{R}^{1 \times N}$, and $D \in \mathbb{R}$ are the model parameters.

The key innovation of S4 is the parametrization of the state matrix A using a combination of a Normal Plus Low-Rank (NPLR) representation and the HiPPO (High-order Polynomial Projection Operator) framework [33]. The NPLR representation allows for efficient computation of the SSM, while the HiPPO framework enables the model to capture long-range dependencies effectively.

The S4 model can be computed efficiently in both recurrent and convolutional forms. The recurrent form is given by:

$$x_k = \bar{A}x_{k-1} + \bar{B}u_k \tag{2.29}$$

$$y_k = Cx_k \tag{2.30}$$

where $\bar{A}$ and $\bar{B}$ are the discretized versions of the continuous-time matrices A and B.

The convolutional form is given by:

$$y = K * u \tag{2.31}$$

where $K \in \mathbb{R}^L$ is the SSM convolution kernel, and $*$ denotes the convolution operation.

As the recurrent formulation is formed entirely of affine operations, a non-recurrent formulation can be written as a mathematical series largely composed of powers of the A, B, and C matrices, and therefore pre-computed, which is broadly how the convolution kernels are made. Using kernels, backpropagation through the sequence length is not needed and therefore training is accelerated, but this also means the kernels need to be recomputed after every update step, which is a computationally costly operation. To reduce the computational complexity of computing matrix powers, and to accelerate recurrent computation in general, this method makes use of structured matrices, where a weight matrix is the sum of a diagnosable matrix and a low-rank matrix.

The S4 model has shown impressive results on various benchmarks, including the Long Range Arena (LRA) benchmark [74], where it outperforms other efficient Transformer variants and sets a new state-of-the-art on the challenging Path-X task.

Mamba: Linear-Time Sequence Modeling with Selective State Spaces

The Mamba model [31] builds upon the S4 model by introducing a selection mechanism that allows the model to perform content-based reasoning. The selection mechanism is incorporated by making the SSM parameters Δ, B, and C functions of the input, enabling the model to selectively propagate or forget information along the sequence length dimension based on the current token.

The Mamba model also introduces a hardware-aware parallel algorithm for computing the selective SSMs in recurrent mode, which allows for efficient computation despite the input-dependent dynamics. The selective SSMs are integrated into a simplified end-to-end neural network architecture called Mamba, which does not include attention or MLP blocks.

Mamba achieves fast inference (5× higher throughput than Transformers) and linear scaling in sequence length, with performance improvements on real data up to million-length sequences. As a general sequence model backbone, Mamba has demonstrated state-of-the-art performance across several modalities, such as language, audio, and genomics. On language modeling tasks, the Mamba-3B model outperforms Transformers of the same size and matches the performance of Transformers twice its size, both in pretraining and downstream evaluation.

The introduction of the selection mechanism in Mamba addresses the limitation of LTI (Linear Time-Invariant) models, which struggle with content-based reasoning. By allowing the model to selectively propagate information based on the input, Mamba can effectively handle tasks that require content-aware reasoning, such as the Selective Copying task and the Induction Heads task.

In summary, SSMs, particularly the S4 and Mamba models, have emerged as a promising alternative to traditional sequence models like RNNs and Transformers. By efficiently modeling long-range dependencies and incorporating content-based reasoning, these models have the potential to become a general-purpose backbone for foundation models operating on sequences across various domains.

2.2.4. comparison

While a task-specific comparison of these models is made in Chapter 4, it is useful to illustrate the main differences between the three overall sequence-to-sequence research directions as we have split them up (RNN-based, Attention (transformer) based, and state-space-based models). For a direct comparison, we use complexity notation to compare them computationally (during both training and inference), in terms of memory usage, their parameter count and memory usage during inference. This comparison is shown in table 2.1.

	Convolution	Recurrence	Attention	S4
Parameters	LH	H^2	H^2	H^2
Training	$\tilde{L}H(B+H)$	BLH^2	$B(L^2H+LH^2)$	$BH(\tilde{H}+\tilde{L})+B\tilde{L}H$
Space	BLH	BLH	$B(L^2+HL)$	BLH
Parallel	Yes	No	Yes	Yes
Inference	LH^2	H^2	L^2H+H^2L	H^2

Table 2.1: Complexity of various sequence models in terms of sequence length (L), batch size (B), and hidden dimension (H). Convolutions are efficient for training, while recurrence is efficient for inference, and structured state-space models (SSMs) like S4 combine the strengths of both. Table is taken from [32]

2.3. Producing variable-length outputs

In the context of nanopore sequencing and basecalling, the task involves mapping a continuous stream of electrical current measurements to a discrete sequence of DNA bases. This mapping inherently deals with variable-length outputs, as the number of observable signals does not directly correspond to the number of DNA bases due to factors such as overlapping bases and variable dwell times within the nanopore. To effectively handle this discrepancy, advanced decoding strategies are employed. Two prominent methods for decoding variable-length outputs in sequence-to-sequence tasks are Conditional Random Fields (CRFs) and Connectionist Temporal Classification (CTC). This section delves into the mathematical formulations, underlying principles, and operational mechanisms of CRF and CTC decoders.

2.3.1. Conditional Random Fields (CRFs)

Conditional Random Fields [47] are a class of discriminative probabilistic models used for predicting sequences of labels. Unlike generative models, which model the joint probability of inputs and outputs, CRFs directly model the conditional probability of the output sequence given the input sequence. This approach is particularly advantageous in scenarios where the input-output alignment is ambiguous or not explicitly provided, as is the case in basecalling.

Mathematical formulation

A CRF defines the conditional probability of an output label sequence $\mathbf{y} = (y_1, y_2, \ldots, y_T)$ given an input sequence $\mathbf{x} = (x_1, x_2, \ldots, x_T)$:

$$P(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left(\sum_{t=1}^{T} \sum_{k} \lambda_k f_k(y_{t-1}, y_t, \mathbf{x}, t)\right) \tag{2.32}$$

where:

- $f_k(y_{t-1}, y_t, \mathbf{x}, t)$ are feature functions that capture dependencies between labels and the input sequence.
- λ_k are the associated weights for the feature functions.
- $Z(\mathbf{x}) = \sum_{\mathbf{y}} \exp\left(\sum_{t=1}^{T} \sum_k \lambda_k f_k(y_{t-1}, y_t, \mathbf{x}, t)\right)$ is the partition function ensuring that the probabilities sum to one.

Decoding with CRFs

Decoding in CRFs involves finding the most probable label sequence $\mathbf{y}^*$ given the input $\mathbf{x}$:

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} P(\mathbf{y} \mid \mathbf{x}) \tag{2.33}$$

Due to the potential complexity of the label space, exact inference can be computationally intensive. However, for linear-chain CRFs, which are commonly used in sequence labeling tasks, dynamic programming algorithms such as the Viterbi algorithm [76] can efficiently compute the optimal sequence.

Advantages and applications

CRFs are particularly suitable for tasks where the output labels exhibit interdependencies, such as in part-of-speech tagging or named entity recognition in natural language processing. In the realm of nanopore sequencing, CRFs can effectively model the dependencies between successive DNA bases, capturing context-dependent variations in the current signal. This leads to more accurate basecalling by leveraging the structured relationships within the output sequence.

2.3.2. Connectionist Temporal Classification (CTC)

Connectionist Temporal Classification [29] is a loss function and decoding strategy specifically designed for sequence-to-sequence tasks where the alignment between input and output sequences is unknown. CTC is widely used in applications such as speech recognition and handwriting recognition, where the input sequence length does not necessarily match the output sequence length.

Mathematical formulation

CTC introduces an auxiliary *blank* symbol $\varnothing$ that allows for flexible alignment between the input and output sequences. Given an input sequence $\mathbf{x} = (x_1, x_2, \ldots, x_T)$ and an output label sequence $\mathbf{y} = (y_1, y_2, \ldots, y_U)$ where $U \leq T$, CTC defines the probability of $\mathbf{y}$ given $\mathbf{x}$ by summing over all possible alignments π that can be collapsed to $\mathbf{y}$:

$$P(\mathbf{y} \mid \mathbf{x}) = \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{y})} \prod_{t=1}^{T} P(\pi_t \mid x_t) \tag{2.34}$$

where:

- $\mathcal{B}$ is the *blank* collapsing function that removes blanks and merges repeated labels.
- $\mathcal{B}^{-1}(\mathbf{y})$ represents all possible extended label sequences π that collapse to $\mathbf{y}$.
- $P(\pi_t \mid x_t)$ is the emission probability of label π_t at time step t.

Decoding with CTC

Decoding with CTC typically involves finding the most probable output sequence $\mathbf{y}^*$ by maximizing the summed probabilities over all valid alignments:

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} P(\mathbf{y} \mid \mathbf{x}) = \arg\max_{\mathbf{y}} \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{y})} \prod_{t=1}^{T} P(\pi_t \mid x_t) \tag{2.35}$$

Due to the exponential number of possible alignments, efficient decoding algorithms such as the Beam Search algorithm [53] are employed to approximate the optimal sequence. Additionally, the forward-backward algorithm is utilized during training to efficiently compute gradients with respect to the model parameters.

Advantages and applications

CTC is advantageous in scenarios where explicit alignment between input and output sequences is either difficult to obtain or varies significantly between samples. In basecalling, CTC allows the model to learn the alignment between the noisy current signals and the underlying DNA bases without requiring pre-aligned training data. By introducing the blank symbol and considering all possible alignments, CTC provides a flexible framework that can handle the inherent variability and uncertainty in nanopore sequencing signals, leading to robust and accurate basecaller models.

2.3.3. Comparison: CRF vs. CTC

Both CRF and CTC are designed to handle sequence-to-sequence tasks with variable-length outputs, yet they differ in their approaches and suitable applications:

- **Alignment Handling**: CRFs explicitly model the dependencies between output labels and can incorporate rich feature representations, making them suitable for tasks where label interdependencies are crucial. CTC, on the other hand, implicitly handles alignment by summing over all possible labelings, making it more straightforward for tasks with less structured output dependencies.
- **Training Complexity**: CRFs generally require more complex training procedures due to the need to compute the partition function over all possible label sequences. CTC simplifies training by focusing on maximizing the probability of the correct label sequence regardless of alignment.

- **Flexibility**: CRFs offer greater flexibility in incorporating domain-specific features and constraints, which can be beneficial in specialized applications like nanopore sequencing where contextual information is valuable. CTC is more rigid but efficient, making it suitable for large-scale sequence labeling tasks.

In the context of basecalling, the choice between CRF and CTC depends on the specific requirements of the task. CRFs may offer higher accuracy by leveraging contextual dependencies, whereas CTC provides a more scalable and alignment-agnostic approach. Hybrid models that combine the strengths of both methods are also an area of ongoing research.

2.4. Hardware performance

As we focus on achieving both good accuracy and throughput with basecalling, it is crucial to understand the hardware we are using to run these benchmarks.

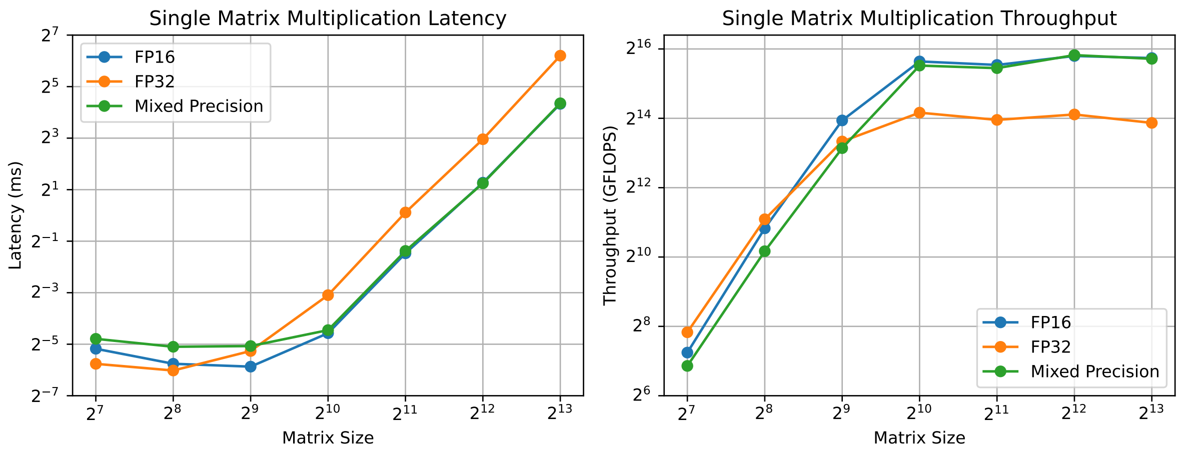


Figure 2.3: Throughput measured in GFLOP/s and latency in ms of matrix multiplications of different sizes on Nvidia 3090 GPU to illustrate the roofline model.

2.4.1. Structure of the GPU

Modern graphics processing units (GPUs) are complex systems designed for parallel processing, particularly the case where the same operation is performed on multiple different data at the same time. Between weight and data storage on the CPU and the GPU, there's the CPU-GPU connection through PCIe (Peripheral Component Interconnect Express). A PCIe 3.0 x16 connection, which is common in many systems, provides a theoretical bandwidth of 16 GB/s in each direction.

GPUs have their own main memory with high bandwidth, of which the type is typically either GDDR or HBM, which can provide bandwidths of hundreds of GB/s. This memory hierarchy continues with L2 cache shared among streaming multiprocessors (SMs) and L1 cache within each SM. The GPU cores are organized into these SMs, which execute warps (groups of 32 threads) in parallel. Each thread consists of a CUDA core which is equipped with floating point multiplication ability. The different CUDA cores are configured to execute parallel Single Instruction Multiple Data (SIMD) instructions, meaning they are able to work in parallel when the same operation has to be performed on multiple pieces of input data. Because of this, conditional statements, which CPUs accelerate by predicting their outcomes and pre-fetching memory that corresponds to an outcome in the conditional statement, run very slowly on the GPU. This is because the CUDA cores cannot all be active at the same time if they need to perform different instructions, which is further exacerbated by the fact that GPUs generally operate at more than two times lower clock speeds than CPUs.

GPUs perform best when asked to perform the same instruction on as much data as there are CUDA cores in parallel.

Streaming multiprocessors on newer GPUs are also equipped with Tensor cores, which allow the SM to break down matrix multiplications into grids of $N \times M$ sub-matrices (where N and M depend on the data type) to multiply them and accumulate their results, so that a Tensor core is able to do this in less instructions than if those values were multiplied and accumulated separately.

Different floating-point formats can encode the same weights. While FP32 (32-bit floating-point) has been the standard, FP16 (16-bit floating-point) is becoming increasingly common in deep learning. FP16 can halve memory bandwidth usage and main memory consumption, and it allows for more operations per unit of silicon, potentially doubling throughput.

2.4.2. Hardware for this thesis

For our benchmarks, we primarily use NVIDIA GPUs, specifically the RTX 3090 and RTX 2080 Ti. These GPUs feature tensor cores, which significantly increase FLOPS (floating-point operations per second) for matrix multiplication and convolution operations common in deep learning. In order to avoid slowdowns due to overheating, which can affect benchmark results, some tests done on the RTX 3090 are made with a power limit restriction on the GPU, restricting it to 250W instead of its default 350W.

Figure 2.3 illustrates the roofline model for the RTX 3090. The goal is to show the maximum amount of operations per second that can be executed on the hardware used to compute benchmarks, as they determine the fundamental upper-limit on how fast a given architecture with computational properties that vary as shown in table 2.1 can be executed on a given hardware. This gives clues for potential implementation inefficiencies. We do this by performing simple matrix multiplications for square matrices of increasing dimensionality. This is an operation that has high computational complexity (cubic) for relatively little memory bandwidth usage since the number of parameters, in comparison, is quadratic with respect to the dimensionality. The x-axis represents the size of the matrix, and the y-axis shows the attainable floating point operations per second (FLOPS) in gigaflops, which is roughly 10^9 floating point operations per second.

The roofline shows that as matrix size grows, the number of operations increases faster than the number of floats that need to be loaded into main memory. This means that for smaller matrices, performance is memory-bound (limited by memory bandwidth, or how fast the weights can be entered and exited from the GPU's compute cores), while for larger matrices, it becomes compute-bound (limited by computational capacity). This reflects the relationship between the growth in number of elements and number of operations of this operation. As seen in the figure, for FP16 operations, the RTX 3090 can achieve around 2^{16} GFLOPS/s, while when using FP32 floating points, it reaches about 2^{14} GFLOPS/s. This discrepancy is not explained by the structure of the CUDA cores, as in our GPU they have as many peak theoretical FLOPS in FP16 as in FP32. Instead, it is explained by the fact that the Tensor cores (which specifically accelerate matrix multiplication) are only able to use FP16 floating points as inputs and therefore offer additional performance when using that 16-bit precision. Mixed precision is a technique where FP32 floating points are automatically cast (or, converted) into FP16 before being given to the tensor cores, and turned back into FP32 to store the results, thereby achieving as much peak FLOPS as when we using FP16 floats.

It's worth noting that our benchmarks are run entirely on the GPU, with data transferred to GPU memory before timing begins. Therefore, CPU performance, system RAM performance and bandwidth between the CPU and the GPU do not significantly factor into our calculations.

3 Understanding basecalling

3.1. Representing basecalling data

Because basecalling is fundamentally a time-series state estimation problem, the inputs, which are measurements of the underlying physical process we want to estimate the state of, are structured in a unidimensionally-ordered way along the time dimension. Information density is a particular issue, with a sampling rate that is only ~8x higher than the rate at which bases go through the nanopore. This is very different from speech recognition, for example, where most recorded words last thousands of recording samples (150wpm is the average speaking rate and 8khz a typical speech recognition pipeline sampling rate, so $60/150 * 8000 = 3200$). Instead, data is very dense.

In order to think in a principled way about neural network development for basecalling, a principled approach would be to ask fundamental questions about the dataset. In this chapter we ask questions and run experiments to answer those questions, such that we can use this information to improve our understanding of what representation ability is characteristic of good accuracy for deep learning models in basecalling.

First, we must answer a theoretical question: how is a base represented, in the current measurement signal we receive as input? To answer that question, we start with understanding the locality of information in basecalling, so that we can establish how the presence and order of a base in a signal is a global or local phenomenon.

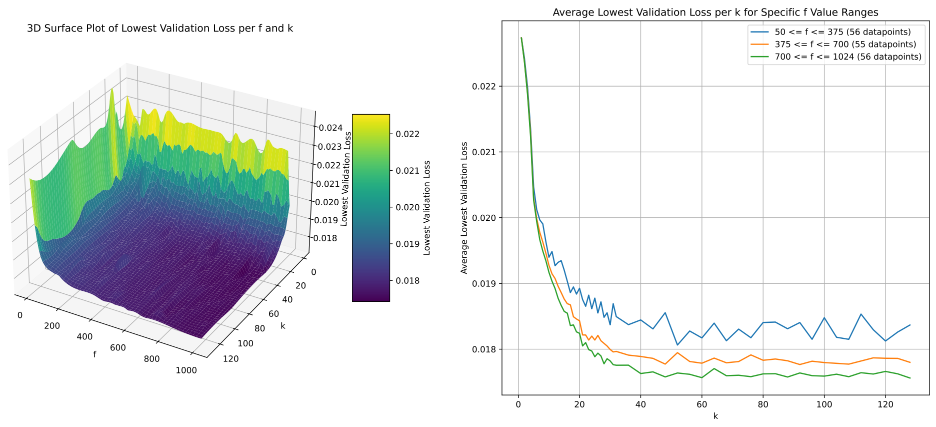


Figure 3.1: A series of single-layer CNNs of varying hyperparameters composed of a one-dimensional CNN layer followed by a ReLU non-linear activation function and a linear layer bringing the outputs of the non-linearity to 5 dimensions, trained for 5 epochs on 2GB of basecalling data using a CTC loss to find the most basic data dependencies. f is the number of filters applied to the input signal, k is the width of each filter's kernel. Validation loss for random outputs is 0.03.

As we can see in Figure 3.1, most of the gains in accuracy are found within around four bases (counting on average 8 datapoints per base, times four that's 32) and regardless of the number of filters, by kernel width 48 (6 bases) there does not seem to be any loss improvements left.

This observation raises several theoretical considerations regarding the representation and modeling of basecalling data. Fundamentally, basecalling seeks to determine the most probable sequence of bases given the observed signal, formally expressed as:

$$\hat{\text{seq}} = \arg\max_{\text{seq}} P(\text{seq} \mid \text{sig}) \tag{3.1}$$

where sig represents the entire nanopore signal. The conditional probability $P(\text{seq} \mid \text{sig})$ can be further decomposed into a sequence of translocations, capturing the sequence of events within the nanopore sequencing process:

$$P([\mathsf{T}_1, \ldots, \mathsf{T}_n] \mid \text{sig}), \tag{3.2}$$

with the sequence $\text{seq} = [\mathsf{T}_1, \ldots, \mathsf{T}_n]$ representing the translocation events. Each translocation event represents a pair of bases entering and exiting the nanopore, as well as the current state of the nanopore. Importantly, these translocations are non-overlapping and can be individually decomposed, ensuring a one-to-one correspondence between translocation events and segments of the signal.

To formalize the structure of the signal, we initially represent it as a sequence of translocation events, which have variable size in terms of measurement samples (which then involves a problem of learning the segmentation of those different translocation events in the signal, but for the purpose of this examination we do not consider this):

$$\begin{aligned} \text{sig} = [&tr(b_n, b_{n+5}),\, tr(b_{n+1}, b_{n+6}),\, tr(b_{n+2}, b_{n+7}), \\ &tr(b_{n+3}, b_{n+8}),\, tr(b_{n+4}, b_{n+9}),\, tr(b_{n+5}, b_{n+10})] \end{aligned}$$

Notice that b_{n+5} appears twice in this sequence, suggesting that if the characterization of a base were predominantly influenced by its own translocation events, we would expect a sharp decrease in the loss curve once the kernel width encompasses the second translocation involving b_{n+5}. However, empirical observations show a more gradual decrease, implying that translocation events depend on the five bases currently within the nanopore, thereby indicating a stateful process. We thus define a translocation event with state as:

$$\begin{aligned} &tr(b_n, b_{n+5}, s_n) \\ s_n = \{&b_n, b_{n+1}, b_{n+2}, b_{n+3}\} \end{aligned}$$

where s_n denotes the state of the nanopore, encompassing the five bases present during translocation. This stateful characterization is consistent with the two current best-performing simulators in research [50, 49, 27], which track the five bases within a nanopore to generate simulated signals. Consequently, incorporating state-tracking mechanisms, as evidenced by sequence-to-sequence neural network layers [56], serves as a beneficial inductive prior for basecalling tasks.

In modeling these translocations, the probability of each base entering the nanopore, B_{n+1}, is conditioned on the sequence of preceding bases $[B_1, \ldots, B_n]$, denoted as $P_{\text{seqm}}(B_{n+1} \mid [B_1, \ldots, B_n])$. Given the Markovian nature of the process, the model maintains a probability distribution over approximately $4^5 = 1024$ possible nanopore states. Consequently, the number of potential trajectories for a sequence of length 250 bases scales exponentially to approximately 10^{150} possibilities, as calculated by:

$$4^{250} \approx 3.27 \times 10^{150} \approx 10^{150}$$

However, empirical observations suggest that the actual distribution p_{seq}, defined as:

$$p_{\text{seq}} = P([\mathsf{T}_1, \ldots, \mathsf{T}_n] \mid \text{sig}) \tag{3.3}$$

is significantly sparse within this vast space, with the number of feasible distinct 250-base sequences estimated to be on the order of 10^{39}. This estimation is meant as a soft upper bound, based on the consideration that the longest known DNA sequences encompass approximately 10^{10} bases and there are roughly 10^{31} cells on Earth (each potentially containing a number of mutations), leading to an order of magnitude of possible distinct 250-base sequences of about $10^{10} \times 10^{31} \div 250 \approx 10^{38}$. The discrepancy between the total possibilities and the upper bound real-world estimate, which we refer to as the *compression gap*, implies that effective basecalling models must learn a highly constrained manifold within the expansive space of possible sequences.

This necessity likely contributes to the efficacy of Conditional Random Field (CRF) decoders in enhancing basecalling accuracy, as they directly model p_{seq}, thereby capturing the underlying sequence dependencies more efficiently and providing more information about the structure of this constrained manifold.

To represent the signal generation by the system, conditioned on the state of the DNA strand and the nanopore, we define a function 3.4, which produces a sequence of signal measurements based on a translocation event tr characterized by the base entering the nanopore (b_n), the base exiting the nanopore (b_{n+5}), and the current state of the nanopore (s_n). This is the function a neural network needs to model apart from P_{seqm}, which we could measure independently for example by generating synthetic signals based on uniformly-sampled DNA sequences.

$$\text{sig\_gen}(tr(b_n, b_{n+5}, s_n)) \tag{3.4}$$

On the signal processing side, each translocation event comprises approximately eight 16-bit signed integers, equating to around 128 bits of information per translocation, compared to the 11 bits of information associated with the nanopore states. This further exacerbates the compression gap, as models must distill high-dimensional signal information into the much sparser sequence space. As models attempt to leverage longer-range dependencies within the signal to improve basecalling accuracy, they encounter the curse of dimensionality, struggling to generalize across the exponentially increasing sequence lengths.

Mutual information between different segments of the signal is thus crucial for effectively utilizing longer-range dependencies. Specifically, when models consider additional segments of the signal beyond the immediate vicinity of a base, there must exist significant mutual information between these segments to justify their inclusion in the predictive process. This necessitates robust modeling of the system's state using sequences of translocations, as this approach aligns with the successful strategies employed by CRF decoders.

In summary, the intricate balance between the high-dimensional signal space and the relatively sparse sequence space underpins the challenges in basecalling. Effective models must navigate this compression gap by exploiting mutual information and leveraging structured probabilistic frameworks to accurately infer base sequences from dense, time-ordered signal measurements.

3.2. Datasets

The dataset used is downloaded from [61], totaling approximately 375 gigabytes (GBs). From this, a subset was chosen by copying training data files from uniformly-sampled random species in the bacteria dataset. The size of this subset totals 52GB, and is chosen based on model training time and to have a similar dataset size to the one used in [61]. The human data from the original dataset was retained to measure generalization, though due to limited time and resources, the primary focus of the thesis is not on generalization to human data. Therefore, it will not be extensively discussed in this thesis. We call this the main dataset.

For more efficient model development, a representative 2GB subset taken using the same bacterial sequences was created, enabling the full training of multiple models within an hour on a single GPU. Training on this dataset was done only using a CTC decoder, for simplicity and speed of training. We found that this was enough data to study the inductive biases of different sequence-to-sequence models, most especially of the RNNs (GRU, LSTM) which are of highest interest for this thesis given that they achieve the highest accuracy on the main dataset. We call this 2GB subset the small dataset.

The 52GB dataset is constituted of data from the following samples:

Folder Name	Size	% Contribution
Klebsiella_pneumoniae-NUH11	6.3G	12.12%
Klebsiella_pneumoniae-NUH27	5.1G	9.81%
Klebsiella_pneumoniae-SGH07	4.7G	9.04%
Klebsiella_pneumoniae-QMP_B2_170	0.5G	0.90%
Klebsiella_quasipneumoniae-INF291	3.6G	6.92%
Klebsiella_variicola-KSB1_8J	5.4G	10.38%
Klebsiella_variicola-INF022	5.3G	10.19%
Morganella_morganii-MSB1_1E	6.4G	12.31%
Moraxella_lincolnii-51409	1.5G	2.88%
Pseudomonas_aeruginosa-MINF_7A	6.2G	11.92%
Salmonella_enterica-2010_06152	4.2G	8.08%
Stenotrophomonas_pavanii-MSB1_4D	2.6G	5.00%

Table 3.1: Composition of the 52GB dataset sampled from the data source used by [61]. Sample details are indicated as codes at the end of the folder names. For example, NUH11 indicates a specific strain.

3.3. Synthetic datasets

Initial attempts were made to create synthetic datasets for the purpose of studying sequence-to-sequence models abilities on simplified tasks that preserve the same essential challenge as basecalling but with controllable levels of difficulty and noise.

3.3.1. Damped harmonic oscillators

The first attempt involved a simulation of a damped harmonic oscillator using the following equations:

$$m\frac{d^2x}{dt^2} + b\frac{dx}{dt} + k(x - x_r) = 0 \tag{3.5}$$

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{3.6}$$

In Equation 3.5, m represents the mass, b is the damping coefficient, k is the spring constant, x is the position, and x_r is the resting height. The second equation, Equation 3.6, represents the Poisson distribution where λ is the average rate, and k is the number of occurrences.

The oscillator's x_r resting height would change with a rate determined by sampling time intervals from the Poisson distribution which were tuned to match basecalling data from our dataset. Multiple values for k, m, and b were attempted to vary the complexity of the task. The task is therefore, at each datapoint, to determine the resting height of the equation at that point in time.

Unfortunately, this task ended up being so simple to solve for even just two convolution layers, even with large amounts of non-linear noise, that modifications in model architecture did not have a sufficient impact on accuracy to be useful to compare them. Therefore this research direction was dropped.

3.3.2. Simulated data

The second attempt at generating an artificial dataset was to use DeepSimulatore [49], a neural-network based model that can generate synthetic basecalling data when given a nucleotide base sequence. The advantages of this approach were that we could generate synthetic sequencing data for base sequences where the bases are selected completely randomly (so that neural network models do not learn a bias towards certain nucleotide sequences rather than others), and also with reduced numbers of base pairs. Models trained using this dataset could also be trained to perform segmentation on the input data, so that we can more easily avoid issues of instability during training.

When developing neural network models on this approach, it unfortunately was the case that models that model performance on the synthetic dataset was not sufficiently predictive of performance on actual basecalling data. One interesting finding worth mentioning was that CRF decoders tended to collapse (meaning, their validation loss suddenly grows intensely and does not come back down) when trained on the synthetic dataset generated from uniform random nucleotide sequences, when applied to the Bonito architecture, and when applied to the SACall transformer-based CNN and encoder (per [61] nomenclature).

We stopped using this dataset for model development because the training dynamics were also too different from actual basecalling data. One particular example is that CNNs trained with receptive fields shorter than 21 showed approximately 40% lower accuracy at the segmentation task when compared to models with exactly 21 or longer receptive field. This is in stark contrast with equivalent experiments from Figure 3.1.

4
Sequence to sequence models

4.1. General model considerations

In the context of sequence-to-sequence (s2s) models, particularly for tasks like basecalling, it is crucial to understand and distinguish between several key performance metrics. Throughout this thesis, we will frequently refer to three primary concepts: performance, accuracy, and throughput. Performance is a broad term that encompasses various aspects of a model's efficiency and effectiveness. It can include factors such as computational speed, resource utilization, and the quality of results. In our discussion, we often use performance as an umbrella term that includes both accuracy and throughput. Accuracy refers to the model's ability to correctly perform its intended task. In the case of basecalling, this would be the proportion of correctly identified nucleotide bases in a DNA or RNA sequence. A higher accuracy indicates that the model is more reliable in its predictions. Throughput, on the other hand, is a measure of the model's processing speed. It quantifies how much data the model can process in a given time frame. For basecalling, this might be expressed as the number of base pairs processed per second. In our case, we will count throughput in terms of input samples per second. Higher throughput is desirable for applications requiring real-time or near-real-time processing of large volumes of data. It's important to note that there's often a trade-off between accuracy and throughput. Models that achieve high accuracy may require more complex computations, potentially reducing throughput. Conversely, models optimized for high throughput might sacrifice some degree of accuracy. Throughout this thesis, we will explore various approaches to balance these competing factors, aiming to develop models that offer both high accuracy and efficient throughput for practical basecalling applications.

4.1.1. Performance-related properties of s2s models

Since the 1990's, convolutions and recurrent neural network models achieved landmark results in deep learning [2, 68], and those results, which evolved over time, were not especially designed with computational efficiency as a goal for the model architecture [2, 68], instead the improved ability to perform a task (i.e. speech recognition, character recognition) was the main focus. Since the paper introducing the transformer network was released however, model and dataset scales became the relevant variables for many of the most popular deep learning tasks, as exemplified by the paper introducing the transformer network architecture itself [75] (in the sequence-to-sequence domain, as it formed a new state of the art for tasks such as language modeling, language translation and speech recognition), and the vision-transformer paper [21].

Some recent papers researching sequence modeling focus on inference-time and training-time computational improvements of sequence-to-sequence models, either by providing an improved implementation of the models [17, 15], by modifying the model's formulation [77, 51], or by attempting to create new models entirely [16, 63, 81, 13, 72]. In this thesis, we will refer to the ability to perform the task as the ”accuracy” of the model, and its computational characteristics as its performance.

The mentioned papers presenting improvements in sequence-to-sequence modeling tend to understudy the computational characteristics of their proposed improvements. They describe performance in some of the following ways. Some compare algorithms on the basis of the time taken for inference of a single input sequence [77], a practice which we will also study and call latency. Others compare the computational complexity [77, 32] of their proposed algorithms, with the problem noted by [17] that computational complexity can be misleading as to how different neural network models will perform in terms of wall-clock time to complete a task. In general, these papers face the inherent limitations of hardware-dependency, meaning that using different hardware (of which there are many [67]) will result in different advantages and disadvantages for two compared neural network models. Another fundamental problem is that there is a combinatorial explosion in the multiple ways an algorithm can be implemented, some often running better than others, and some being discovered years after intense research has been put into the task in question [24]. In this domain, this problem can be exemplified by [17] and [15], where not only big performance improvements were achieved by changing the implementation of the algorithm, but it also involved counter-intuitively increasing the total amount of calculations performed by the hardware. The FlashAttention paper [17] showed that by improving the implementation alone, speedups in training and inference time in practice could be higher than by improving the computational complexity of the algorithm.

We posit that applications where: large amounts of parallelizable inputs have to be computed at the same time, and where cost is an issue, as in basecalling, then maximum throughput is the most relevant performance variable. Throughput in sequence-to-sequence models could be counted in samples per second, where a sample is a unitary input data along the time dimension of a sequence. In our case it is a current measurement of the nanopore reading machine. This metric can be optimized for in the same ways discussed earlier, and there can be a trade-off when increasing the batch size as it increases the throughput but but also increases the latency (see Figures 12 and 13 in [70], or Figure 5.2 which is just one example. This is true of most models benchmarked in this thesis).

Once we have designed a basecalling model, how can we adjust its throughput and accuracy, in order to achieve a desired trade-off level? In this thesis, we will use what we call model dimensionality in order to do this. We will restrict ourselves to designing models that contain a series of modules that can be repeated in order to increase the depth of the model. However, because we find that in general, increasing the depth of models for basecalling leads to instabilities and it is often the case that a certain number of layers is optimal, we decide to instead change the number of features that come in and out of a module in order to increase or decrease the throughput, and vice versa the accuracy, of our models. Generally, this has the effect of increasing the size of the parameter matrices of the different models, and therefore the total amount of computation and data movement required to perform inference. For models in this chapter, we did not put a particular emphasis on finding deeper networks with higher layer counts than around 2, because the layers we were working with already produced much lower throughputs than our RNN-based baseline, so that researching higher-accuracy models with even lower throughput would be counter-productive to the goal of this thesis, which is to find models that push the throughput/accuracy Pareto curve.

4.1.2. Accuracy-related properties of s2s models

Sequence to sequence models are often tested on multiple sequence-related tasks when proposed, such as Chomsky language recognition [13] or long range arena [74], where we see high variability between different models. This means that different models have different strengths in their ability to perform certain tasks, rather than having a generalized ability to perform, at least in these standardized settings. The same is true in basecalling, where a previous work comparing different basecalling models present in the literature [61] including transformers (vanilla transformer [75] and a modified lite-transformer [54]), recurrent models (LSTM, GRU) and causal convolutions. One conclusion of that work was that recurrent models definitely perform better than attention-based models, but there was no further exploration of other models. In general, their reproduction of Oxford Nanopore's Bonito architecture was the vanilla model that performed best, and may be the largest company doing research in this domain.

In contrast, Oxford Nanopore has announced that a LocalTransformer-based architecture [34, 62] they developed has achieved best accuracy at lower-throughput regimes [19], making them worth investigating in a research setting. Worth noting is that the model they end up with uses rotary embeddings, which has been found to be superior to positional embeddings in learning hierarchical language structures [1].

Models based on state-space models also show promise. S4 [32] has been shown to outperform transformers in tasks such as speech recognition or image recognition, with the important property that it is timescale-invariant and therefore does not suffer from a large decrease in accuracy when the sequence length increases by even an order of magnitude, while at the same time enjoying large improvements in latency. Mamba [31] is another model in this series, which offers a new model with accelerated kernels which contribute to fast execution on deep learning accelerators. It performs language modeling with accuracy on par with the best transformer model found so far, but with throughput around 4 times higher at language generation. Because of its selective attention mechanism, it is said to have more complex modeling capabilities, as was confirmed by [56].

xLSTM [4] is another recent s2s model which outperforms other models at language recognition and language modeling, offering improvements on the original LSTM architecture, optimized kernels as well and faster inference than transformer networks. Finally, Retentive Network [72] is yet another model, this time showing both improvements in throughput and latency, and improvements in language modeling.

4.1.3. Selected techniques

In the field of sequence modeling, there are more models and techniques than there is time to test on a task like ours. For each, some amount of hyperparameter tuning and architecture adjustment for properties like stability during training, choice of functions according to their computational properties, and choice of implementation is involved in order to properly judge its potential in solving a certain task. For example, batch normalization layers [42] and gated activation functions [18] are both techniques to enhance training stability, but at large dimensionalities the normalisation layer has a much smaller computational impact on performance than gating. Because of this limitation, we select architectures and techniques that show promise in basecalling and that represent a conceptually representative sample of the line of research in which they are contextualized..

We have chosen to include the following s2s models in our analysis, and link to the sections discussing them:

- RNN models (Section 4.2.1)
  - LSTM
  - GRU
  - Elman RNN
  - xLSTM
- Attention-based models (Section 4.2.2)
  - Transformer
  - Local-Transformer
  - Reformer
- State-space models (Section 4.2.3)
  - S4
  - Mamba

One might also wonder why it could be beneficial to focus on architectures whose focus is on modelling very long input sequences above thousands of elements such as xLSTM, Local-Transformer, Reformer, S4, Mamba when in our case the input sequence is never longer than 500. We argue that the problem of long sequences is one where the solutions being pursued are ones increasing efficiency in terms of compute needed per input vector, and that this has a lot in common with increasing efficiency of basecalling. A limitation is that these architectures do not generally explicitly pursue efficiency increases at very large batch sizes (but in basecalling, 512 is the number of simultaneous input streams of the smallest machine sold by Oxford Nanopore). Nevertheless, it is because of this overlap that we chose to pursue this research direction.

We show preliminary accuracy/throughput numbers for the models mentioned so far, in Figure 4.1. It is important to point out how much the RNNs have an advantage over other architectures. In the rest of this chapter, we document our attempts to either modify RNNs or to attempt to make other sequence-to-sequence models competitive with RNNs. In Chapter 5 we will instead propose our own attempt at improving upon the RNN's very good accuracy and throughput properties.

The LSTM and GRU we will consider our baselines, because their accuracy/throughput is ahead of others and the current best-performing open-source basecalling model uses LSTM layers. xLSTM represents improvements in recurrent nets. The Transformer is included as the main competitor to recurrent models [60] and to include a new variant we include the Local-Transformer. ReFormer represents advances in making a Transformer-based s2s model. S4 and Mamba represent the line of research into state-space-based models. All together, we believe this set covers a substantially wide range of modern developments in model architectures for sequence modeling.

To also consider more general model modifications, we have chosen to also consider the following:

- Fast-feed-forward (FFF), a neural network layer that replaces feedforward layers with layers that learn a $\log_2(n)$ computational complexity linear layer equivalent (in Section 4.2.5)
- Mixture of experts, a way to increase accuracy without adding computational complexity (in Section 4.2.5)
- An exploration of different pre-processing functions to apply to the signal given as input (in Section 4.3)
- Two implementations of Kolmogorov-Arnold networks, one as a replacement for convolution, and one as a replacement for linear layers (in Section 4.2.5)
- Knowledge distillation, a technique to use patterns learned by a teacher model to assist the training of a student model (in Section 4.2.4)

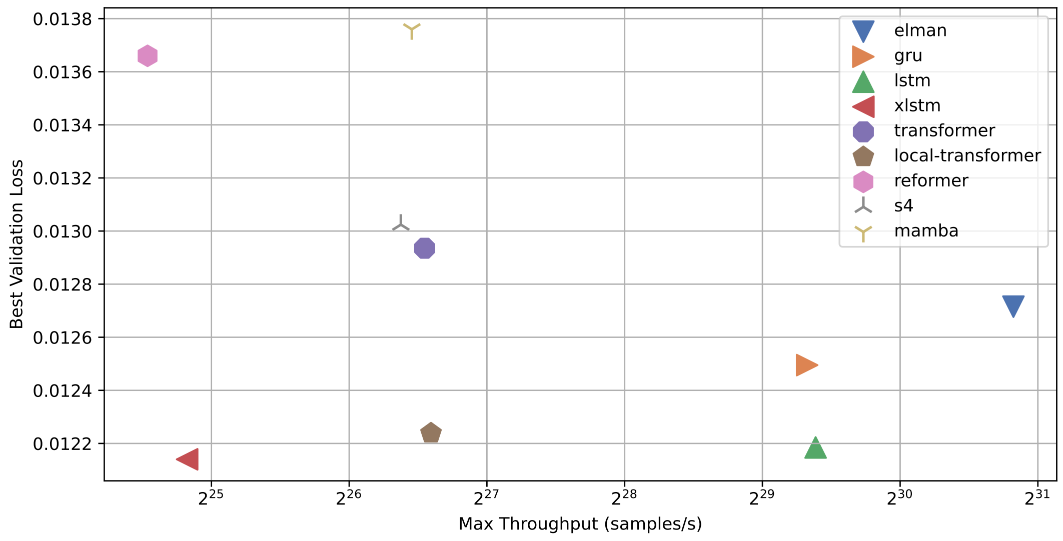


Figure 4.1: Comparison of all base models considered in this chapter in terms of their throughput and validation loss. Models have two layers and a dimensionality of 128, which is the dimensionality used for all models when working with the 2GB dataset. Benchmarks run on RTX 3090 GPU. Model types are separated into groups of shapes (triangles for recurrent models, three-lines for state-space models, and polygons for transformer-based models). Three models form the Pareto frontier on this figure: the elman model, the lstm model and the xlstm model. Results are discussed in section

4.2. Model development

After seeing Figure 4.1, one might want to only study models that lie on the Pareto curve. But we find that a given model can be modified in many different ways that could have an impact both on accuracy and throughput. And while the impact of a modification on the model's achievable throughput is often intuitive, this is often not the case with accuracy, and therefore we cannot know in advance whether an improvement might cause a model to yield a useful trade-off between accuracy and throughput. This is why we decided to spend extensive amounts of time studying many different techniques in this thesis. In this section, we detail our wide-ranging search for useful model architectures, primarily using the small dataset.

It is also worth mentioning that in order to achieve fast iteration between models, our training procedure is to first train a model for 5 epochs on the small dataset, and when results show clear improvements over a baseline model, we move to training it on 15 epochs instead, to see whether the model is under-trained or not.

4.2.1. A closer look at RNNs

Ablating the GRU

In order to better understand why RNNs perform so well at language modeling, we decided to create ablations of the GRU, to measure the impact of the modifications on validation accuracy after training for five epochs on 2GB basecalling data:

Ablation Code	Best Validation Loss (Percentage Change)	Description
5.4.0	0.01327 (-1.11%)	Set h_{t-1} to k=1 1dconv output
5.4.2	0.01339 (-0.22%)	Set h_{t-1} to k=4 1dconv output
5.4.1	0.01341 (-0.07%)	Set h_{t-1} to k=2 1dconv output
baseline	0.01342 (0.00%)	Baseline GRU model
5.5	0.01352 (+0.74%)	Set h_{t-1} to transformer layer output
5.4.3	0.01363 (+1.56%)	Set h_{t-1} to k=8 1dconv output
6.2	0.01363 (+1.56%)	Replace each weight matrix one-by-one by a point-wise product with a parameter vector
4.5.4	0.01364 (+1.63%)	Set r_t to 1.0
4.5.3	0.01373 (+2.30%)	Set r_t to 0.75
4.5.2	0.01388 (+3.42%)	Set r_t to 0.5
2.4.0	0.01401 (+4.39%)	Set z_t to 1.0
4.5.1	0.01404 (+4.61%)	Set r_t to 0.25
alt baseline	0.01419 (+5.73%)	Equivalent Transformer-based architecture
1	0.01423 (+6.03%)	Remove h_{t-1} from h_t
2.4.1	0.01424 (+6.11%)	Set z_t to 0.75
4.5.0	0.01437 (+7.07%)	Set r_t to 0.0
2.4.2	0.01470 (+9.53%)	Set z_t to constant 0.5
2.4.3	0.01504 (+12.07%)	Set z_t to constant 0.25
6.1	0.01638 (+22.05%)	Remove all non-linearities
2.1	0.02367 (+76.37%)	Only consider x_t
3.1	0.02367 (+76.37%)	Remove the non-linearity
4.4	0.02367 (+76.37%)	Replace non-linearity with ReLU
5.3	0.02367 (+76.37%)	Set h_{t-1} to x_{t-1}
2.3	0.02371 (+76.67%)	Remove the activation function
3.3	0.02371 (+76.67%)	Only consider x_t
3.4	0.02372 (+76.75%)	Only consider h_{t-1} with r_t
4.1	0.02372 (+76.75%)	Only consider x_t
2.2	0.02374 (+76.90%)	Only consider h_{t-1}
3.2	0.02374 (+76.90%)	Replace non-linearity with a ReLU
2.4.4	0.02375 (+76.97%)	Set z_t to 0.0
4.2	0.02375 (+76.97%)	Only consider h_{t-1}
4.3	0.02375 (+76.97%)	Remove non-linearity
5.1	0.02375 (+76.97%)	Set h_{t-1} to always zero
5.2	0.02375 (+76.97%)	Set h_t to always equal x_t
3.5	0.02376 (+77.04%)	Only consider h_{t-1} without r_t

Table 4.1: Impact of various ablations on the GRU model equations as shown in section 2.2.1. Ablation codes indicate how the GRU equations were changed with syntax x.y.z where x is a part of the equations that is changed, y is the specific change that was made, and optionally z indicates different values for a variable which is part of the change. Here are the meanings of the x values: (1) Remove h_{t-n} from h_t from equation 2.16, (2) Modify z_t from equation 2.13 (3) Modify $\hat{h}_t$ from equation 2.15 (4) Modify r_t from equation 2.14 (5) Modify h_{t-1} from all GRU equations (6) General modifications. $17/35$ modifications increased the loss by more than 50%

Some ablations are simplifications such as removing a term or replacing a function by a simpler function. Others are modifications that aim to remove properties, such as when we replace h_{t-1} with the output of another s2s layer, either a transformer encoder or 1d convolution. In that case, the model effectively becomes a mix between a recurrent model and a parallel s2s model.

We will consider that models that have $+76\%$ increases in validation loss have failed to learn basecalling during training. In general, modifications to $\hat{h}_t$ (which is the part of h_t that contains information from the current input) cause the model to fail, and since no other ablation code (see caption of Figure 4.2.1) has all of its sub-ablations fail, this indicates that $\hat{h}_t$ is the most crucial part of the GRU architecture for basecalling.

Gating could be replaced with a fixed value and retain much of its validation loss. The update gate vector z_t had an impact of +4.39% on loss when set to one and fails to learn when set to a value of zero (for obvious reasons, as it then doesn't use any inputs. At $z_t = 1$, only equation 2.15 is used to calculate the output, which means there is no more update gate, so how much of the input vector is added to the hidden state only depends on matrix W_h and the reset gate vector.

The reset gate vector r_t could also be replaced by a fixed value and had minimal impact on validation loss when set to 1, which means that the $\hat{h}_t$ vector always contains maximal information from h_{t-1}. Generally, gating in recurrent networks serves to enhance information flow through the iteration steps [44], and in this case we are decreasing the rate at which hidden states are forgotten, increasing the time horizon of the flowing information. This could imply that the neural network is using contextual information that goes beyond the approximately 48 datapoints of information locally-available to determine the base at a certain point in the input sequence.

One last point of interest is to note that removing all non-linearities did lead to a model that succeeded in learning basecalling, however training collapsed about halfway during training, rather consistently. It is its instability that lead to the +22% increase in loss. This implies that a more stable approach to training such a model would make it parallelizable in the same way state-space models like S4 [32] are. The model then can be formulated as a size-n kernel that can be computed either iteratively through the input sequence or in parallel as a convolution operation. Similar approaches have been explored multiple times in the past [25, 8].

Overall exploration of RNNs

As seen in Figure 4.1, recurrent neural networks (RNNs) show great promise for the task of basecalling. The main question we aim to address in this subsubsection is: which RNN variant performs best at this task, and why? We focus our analysis on three RNN variants included in PyTorch: LSTM, GRU, and Elman. Despite differences in their floating point operations, we observe minimal variation in throughput across these models. This is likely due to the inability to utilize batch sizes large enough to saturate the available memory bandwidth.

Given the advantage of recurrent networks, we concentrated our efforts on model development using the large dataset. Our goal was to develop an Elman RNN-based model that surpasses the Pareto front of the LSTM. However, we have not been entirely successful in this endeavor. We found that incorporating residual connections and layer normalization layers improves accuracy and training stability. Additionally, applying a small amount of dropout (around 10-20%) proves beneficial. Nevertheless, we observed that the model struggles to effectively utilize model dimensions beyond 64, as indicated by the increase in generalization and validation accuracies when employing distillation techniques.

We hypothesize that a fundamental limitation might exist, as evidenced by the GRU ablation results in Table 4.2.1. Gating, particularly the "forgetting" mechanism in GRU and LSTM layers, allows for implicit segmentation of the input. In the ablation where we replace the reset gate with a fixed scalar (ablation codes 4.5), we constrain the model's ability to implicitly segment the input. If we consider the update gate z as representing the extent to which the nanopore state has changed since the last timestep (otherwise, we could always return the same hidden state), we can interpret it as handling the temporal fluctuations of basecalling, which determine the rate at which the nanopore state evolves over time. On the other hand, the reset gate could be interpreted as estimating the mutual information between the previous and next nanopore states, as it determines the proportion of the previous state used to compute a new hidden state (the candidate hidden state in equation 2.15). Given that state-of-the-art basecallers appear to be most sensitive to temporal fluctuations [27], our intuition suggests that the update gate would be the most crucial for achieving the best loss, which aligns with the ablation results.

Regarding LSTMs, in Chapter 6, we discuss the advantage of the GRU over the LSTM when using distillation. This indicates that the GRU can, to some extent, represent the same information as the LSTM. The difference can be primarily attributed to the memory cell c (see equation 2.11), which allows for the utilization and propagation of information over longer temporal distances. The fact that we could sometimes train GRUs to achieve similar accuracy to LSTMs implies that the ability to learn over longer temporal distances mainly impacts training dynamics. If the GRU can represent the same accuracy, it would seem unlikely that the model couldn't find those weights independently through gradient descent, suggesting that either more sophisticated regularisation or optimization could be helpful.

Concerning xLSTM, we only utilize their sLSTM layers, as the mLSTM layers are extremely slow despite using 16-bit floating-point arithmetic and handcrafted CUDA kernels provided by the authors. The sLSTM layer, although slower, demonstrates even lower loss compared to LSTMs, which is intriguing. Given that the core idea behind sLSTM is to increase representational capacity and extend the complexity of its representational space, we would expect substantial improvements in basecalling. However, a significant issue appears to stem from the higher FLOP count in sLSTMs due to the increased size complexity of their internal state. Furthermore, we were unable to achieve high enough accuracy on the main dataset to pursue this direction further.

4.2.2. Exploring the transformer layer

While transformer layers are slow due to the $O(n^2)$ computational requirement during inference [32], they are worth investigating for their ability to achieve top accuracy in highly competitive tasks such as automatic speech recognition [55] and image classification [69] and occasional works that find their throughput and latency to be competitive with other competing architectures [6]. The idea being that if a technique can increase the accuracy of a transformer network, we can later tune the model for throughput using various other techniques (such as by manipulating which parts of the attention matrix we compute, or using operations that are an alternative to attention). Our first dataset and task for model development was the 2GB small dataset.

Chain-of-Thought model development

To enhance the performance of our transformer-based models on the base recognition task, we explored the incorporation of the chain-of-thought (CoT) mechanism. Inspired by recent research indicating that CoT allows models to recognize patterns at higher levels of the Chomsky hierarchy [57], we hypothesized that integrating a CoT approach could improve accuracy without significantly impacting inference time. Given that the theoretical benefit can be achieved with a chain length of $\log_2(n)$ (approximately 8 for our sequence length of $n = 400$), this technique was particularly appealing.

We developed several versions of our model, each progressively integrating the CoT mechanism in different ways. Our base architecture consists of a 1D convolutional layer for initial feature extraction, followed by ReLU activation and positional encoding [75]. The core of the model employs transformer layers to capture dependencies in the sequence data.

Model architecture details All model variants share a common backbone architecture comprising:

1. **Convolutional Layer:** A 1D convolutional layer processes the raw input sequences to extract local features. The kernel size, stride, padding, and dilation parameters are adjusted across versions to optimize performance.

2. **Activation Function:** A ReLU activation function introduces non-linearity.
3. **Positional Encoding:** Positional encoding is applied to the embeddings to retain sequence order information, which is crucial for processing sequential data.
4. **Transformer Layers:** Depending on the version, transformer encoder and decoder layers are utilized to model long-range dependencies and capture complex patterns within the data.
5. **Chain-of-Thought Mechanism:** Implemented through reasoning tensors and iterative processing steps, the CoT mechanism enables the model to simulate a reasoning process over the input data.
6. **Fully Connected Layer:** A final linear layer maps the transformer outputs to the desired output dimensions (e.g., classification logits or regression targets).

SequenceNetCoT two-step variants To incorporate the CoT mechanism, we introduced a series of models labeled *SequenceNetCoTTwostep*, each refining the approach to integrate reasoning steps into the model's processing pipeline.

- **Version 1 (SequenceNetCoTTwostep V1):** In this version, we introduced a reasoning tensor initialized to zeros, representing a fixed number of reasoning steps (*reasoning_steps*). Both the input sequence and the reasoning tensor undergo positional encoding. We then pass the reasoning tensor through a transformer decoder layer, using the encoded input sequence as the memory (context). This allows the model to generate reasoning representations based on the input data.
- **Version 2 (SequenceNetCoTTwostep V2):** Building upon the previous model, we added special tokens to act as separators between the input sequence and the reasoning steps. Specifically, we introduced two learnable parameters: a special token inserted between the input and reasoning tensors, and another to denote the end of the sequence. This aims to help the model distinguish between different segments of the input, potentially improving the alignment between the reasoning steps and the relevant parts of the input.
- **Version 3 (SequenceNetCoTTwostep V3):** This iteration incorporates an iterative reasoning process. For each reasoning step, the model updates the reasoning tensor by attending over both the input sequence and the accumulated reasoning from previous steps. This is achieved by looping over the reasoning steps and sequentially updating the reasoning tensor. This process allows the model to refine its reasoning iteratively, simulating a step-by-step thought process.
- **Version 4 (SequenceNetCoTTwostep V4):** In this version, we enabled multiple reasoning iterations controlled by a hyperparameter (*reasoning_iterations*). We also replaced the zero-initialized reasoning tensor with learnable parameters, allowing the model to learn an initial state for the reasoning steps. This change provides the model with greater flexibility to determine the starting point of its reasoning process.
- **Version 5 (SequenceNetCoTTwostep V5):** The final iteration explores several architectural modifications:
  - *Residual Connections:* We added residual connections between the input embeddings and the outputs of the transformer encoder, allowing the model to leverage both the original input and the transformed representations.
  - *Encoded Context Inclusion:* We experimented with including the encoded context (the output of the transformer encoder applied to the combined input and reasoning tensors) in the final reasoning steps. This aims to enhance the information flow between the input and the reasoning mechanism.
  - *Separator Tokens:* Similar to Version 2, we included separator tokens to delimit different segments within the sequence, aiding the model in distinguishing between the input data and reasoning steps.

Implementation considerations Throughout the development of these models, we carefully considered the computational complexity introduced by the transformer layers and the CoT mechanism. While transformer layers have a quadratic time complexity with respect to sequence length ($O(n^2)$), we aimed to mitigate this by limiting the length of the reasoning steps and optimizing the model architecture.

Experimental results We conducted extensive experiments on the 2GB small dataset. Initially, the CoT models showed worse validation loss than vanilla transformers. However, after increasing the number of training epochs to 15, the V5 CoT mechanism reached lower validation loss (by about 3%) than the baseline two-layer transformer-encoder-based model, and both reached better validation loss than even a two-layer GRU.

Model	Best Validation Loss
two-layer transformer baseline	0.01186
SequenceNetCoTTwostep V5	0.01176
two-layer GRU baseline	0.01192

Table 4.2: Validation loss of chain-of-thought-based transformer network and baselines when trained on small 2GB basecalling dataset

Benchmarks also showed that this updated block structure achieves throughput 10 to 15 percent lower than the transformer baseline, so it was tested on the SACall architecture [40] as a drop-in replacement for the transformer encoder blocks. This led to a negligible increase in accuracy of about 0.5%, however, interestingly it led to a greater increase in out-of-distribution generalisation of 3.63%

Given the impact of the chain-of-thought technique on accuracy, we tried to use it with a Mamba-based architecture, both as a drop-in replacement and, because the Mamba architecture does not attend to all previous tokens directly (but rather through a recurrent state), we hypothesized that the chain-of-thought vectors should be interleaved with the input vectors (so, instead of concatenating the chain-of-thought vectors to the block's input, we interleave them with the input, initialized as a trainable vector so that the model is able to distinguish them from input data). This did not have any favorable impact on validation loss.

Finally, we did run our best model on the 52GB dataset (see table 5.1). We notice that the chain-of-thought model performs slightly better in validation accuracy, while having a bigger impact on generalization. Throughput is, of course, lower due to the increased number of operations.

Locality-enhanced transformer models

In our exploration of transformer models for basecalling, we investigated the impact of incorporating local attention mechanisms, as proposed in the BigBird architecture [80]. The motivation behind this approach was to capture local dependencies within the input sequences while maintaining the ability to model long-range interactions.

Our experiments with local attention yielded mixed results. On the small dataset, we observed an improvement in validation loss compared to the standard transformer model. However, when we applied the same technique to the main dataset, the improvement did not replicate, and the model's throughput remained unchanged. Our finding that this technique does not seem to increase throughput remained true across various implementations using their own optimizations, including xformers [48], fairseq [59], and the recent flexattention library [35], indicating that the issue was not related to the specific implementation.

Further analysis revealed that the primary factor limiting the throughput in both the local attention model and the Reformer [45] was the number of attention heads. Reducing the number of heads led to a significant degradation in validation loss, suggesting an inherent overhead associated with computing the heads separately. Figure 4.1 illustrates that the Reformer model does not exhibit an advantage in validation loss over the standard transformer, and reducing the number of heads substantially worsens the validation loss. For this reason we did not pursue these two models.

Conclusion: transformer improvements for basecalling

Overall, it is evident that there is some room for improving transformer networks for basecalling, as demonstrated by our findings and the CATCaller [54] paper. Additionally, it would be beneficial to investigate how these improvements can be coupled with throughput optimization techniques for effective deployment of models. Ultimately, our findings emphasize the pay-offs in accuracy that can come from delving into these various routes for enhancement of transformer-based basecalling models.

4.2.3. State-space model development

The model using the S4 layer was the model offering the best throughput-validation loss trade-off when not considering transformer-based models, so that extensive work was made to find block structures around it that could improve the model's accuracy or its throughput. This involved parameter sweeps, experimentation with residual connections, normalization layers, linear layers between the S4 layers, all to develop the best-performing block that we can repeat multiple times, as is the approach used in most of the s2s model papers we're including in this thesis. The resulting model exhibited good training dynamics and improved validation loss on the small dataset, so we went on to continue model development on the 52GB dataset.

The best S4-based block developed had the following structure: batch norm, a bidirectional s4 block [32] with dropout 0.1 maximum kernel length 64 (which was found to have a large positive impact on validation loss on the 2GB dataset, as well as a large impact on throughput), the GELU activation function, and using the original S4 kernel (no HiPPO or other techniques shown in [32]). With this architecture, we were able to achieve validation accuracy on the 52GB training set of around 5.6% lower than SACall, at 0.817 versus 0.866, so it was not pursued further. We hypothesize that because S4 does not perform state tracking [56], it is not able to track the change in the internal state of the nanopore during basecalling along the time dimension. Similar efforts were applied to Mamba and Mamba2 because it was shown to be an improvement upon S4 on the task of language modeling and in state tracking tasks, but none of our attempts were able to achieve good accuracy or validation loss.

4.2.4. Distillation

Knowledge distillation is a technique where the behavior of one neural network (either its outputs, or some intermediary values taken from its architecture) is put in a loss function along with equivalent values from another network which is considered to have knowledge, or abilities, which the former model does not have, so that the gradient descent process not only uses ground truth labels but a reference coming from another model which has some advantage over the main model to train. The seminal work on knowledge distillation focused on expert models distilling their knowledge into models that combine their expertise [36], but as recent transformer-based models show improvements in ability that scale with the size of the training data and the model's parameter counts [38], more recent works in distillation have focused instead on treating the larger or longer-trained (or both) model as the "expert" model from which a model with more favorable computational characteristics learns richer information than just its training data during training [10].

We attempt this on the 52GB dataset directly since it is a simple modification of the training process, by first training a Bonito model with dimensionality 512 (to maximize accuracy), and then using this pre-trained model to distill its abilities to models with dimensionalities 64, 96, 128, 384. We distill the large model into the small model by taking the output of either their encoders (the LSTM layers) or their decoders (the CRF model's input, see [61]), applying a linear layer to the small model's encoder output so it matches the dimensionality of the big model's (when distilling encoder outputs). When distilling on decoder outputs, we found that a mean-square error loss worked best, however this constrained distillation to models with the same decoder layer (CRF, whereas SACall for example uses a CTC decoder). This is why encoder-based distillation was pursued, and we found that in this case, using a mean-square error loss collapses model accuracies during training, but that Kullback-Leibler divergence works much better. Thus, decoder distillation uses mean-square error and encoder distillation uses Kullback-Leibler divergence as a loss function. Guided by [10], we stop the distillation before ending training, in our case after reaching 50% of the training steps, which we found empirically was a good ratio, as can be seen in Figure 4.2, where we also notice that the model with dimensionality 128 actually suffers from distillation. We found this to hold in general, and were not able to increase accuracy for a model at or below 128 using distillation, a fact we tentatively attribute to model capacity being saturated on the smaller models, and them therefore not being able to represent more complex patterns.

Other techniques attempted are self-distillation [82], where a model is distilled from a trained model of

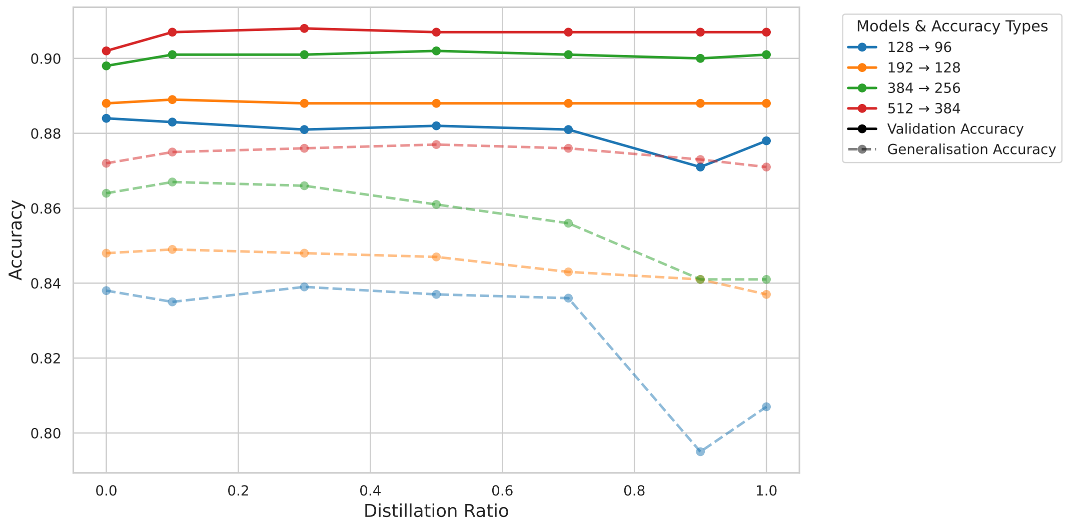


Figure 4.2: Validation and generalistaion accuracy of Bonito models with different dimensionalities, depending on what percentage of total training steps we stop the distillation loss. 0.5 means we stop distillation halfway through training. 0.0 means that the model is not distilled at all. Number before the arrow indicates teacher model dimensionality, whereas number after the arrow is the student model dimensionality.

the same dimensions as the student. This was repeated multiple times and found to increase accuracy and generalisation for models with dimensionality 256 or higher. Best results are in Table 5.1 and are chosen from the mutliple results collected during training. Complete sweeps and accuracy numbers for all methods are not available due to time limitations.

4.2.5. Other modifications

Mixture of experts models

Another strategy we pursued to improve our model's performance was a technique commonly used to increase accuracy (or perplexity in the case of language modeling) by increasing the parameter count without increasing inference complexity. This has successfully been done in transformer and Mamba-based architectures [41, 3]. The idea is essentially that a single layer is replaced with a set of identical layers with different parameters (termed "experts"), and a lightweight gating layer selects a small number of them which the input is given to, and their outputs are averaged. Thus, one could have 128 feed-forward layers, but only select 2 for every input, thereby not loading or computing the remaining 126. We found that this approach does not benefit basecalling at our levels of throughput because just setting the batch size at 16 would mean that every single batch chooses different experts, which means all experts end up being run. We also tried gathering sequence elements by chosen expert in order to group computation, but this was found to cause significant overhead. We advise against pursuing mixture-of-experts architectures in this way for the purpose of basecalling.

Fastfeedforward layer

Fast feed-forward [5] layers are an alternative to feed-forward layers in neural networks that perform sparsification so that during inference, a multi-layer perceptron (such as the one present in the transformer encoder block) which normally takes a runtime of $O(d^2)$ where d is the dimensionality of the input vectors, can be replaced with their layer that has a runtime of $O(log_2(n))$ with comparable accuracy. In our experiments, the mechanism through which computational complexity is reduced, meaning the sparse matrix operation, led to lower throughput than an equivalent perceptron on our Nvidia accelerators (2080ti, 3090), and therefore was not pursued further. This is despite attempting multiple implementations, including the one offered by the authors themselves. We theorize that such an approach would work better if custom CUDA kernels were written that implement these operations.

Kolmogorov-Arnold networks

Kolmogorov-Arnold networks (KAN) [52] are another replacement for multi-layer-perceptrons, however instead of their theoretical base being based on the universal approximation theorem [39] where the subject of approximation is broadly any differentiable function, they are based on the Kolmogorov-Arnold representation theorem [46] which focuses instead on continuous multivariate functions, stating that they can be approximated by compositions of univariate continuous functions.

Batch Size	Conv1dOperatorKAN	Conv1d	Two-Layer KAN	Two-Layer MLP
1	1’626.53	13’917.09	2’081.21	9’387.37
16	25’467.84	219’266.50	8’580.51	47959.33
64	30’892.11	835’149.76	8’749.77	52188.53
256	31’310.66	1’150’916.88	8’794.13	53’755.35

Table 4.3: Throughput in batches/second for various models with different batch sizes. Conv1dOperatorKAN is the kan-based 1-dimensional convolution (a two-layer KAN replaces the linear layer and non-linearity of the conv1d model), Two-Layer KAN is just a two-layer KAN network, and two-layer MLP is an equivalent two-layer multi-layer perceptron. Convolutions have kernel size 5, 4 input filters and 16 output filters. The MLP and two-layer KAN model have two layers each, both with input and output neuron count of 64.

We start by measuring the throughput of equivalently sized perceptron-based and KAN-based networks using the highest-performance implementation we could find [66], which is a version where the univariate continuous functions are learnable grids of re-weighted sine functions. We notice that there are multiple order of magnitude differences between the KAN-based models and their perceptron counterparts, as seen in table 4.3. The impact on accuracy is measured by replacing convolution layers in the Bonito architecture [61], specifically the first and second layer, in order to minimize the impact on throughput, since those operations take a small percentage of total execution time.

Model	Best Train Acc	Best Val Acc
1st conv kan	0.920f	0.901f
baseline	0.923f	0.905f
2nd conv kan	0.923f	0.907f

Table 4.4: Train, and validation accuracy measurements on 52GB basecalling data for Bonito model where we replace either the first or second

Results are in Table 4.4, where we notice that using a KAN on the first convolution worsens validation accuracy. This, along with the fact that the gradient magnitudes were higher on this model, lead us to think that the model does not stabilize during training. Replacing the second convolution with a kan-based one however, does have a positive impact on validation accuracy, which could be explained by its higher representative power. Overall, because of the impact on throughput, we do not consider this result worth pursuing further.

4.3. Exploring alternative input representations

In our pursuit to enhance the accuracy and efficiency of the basecalling models, we explored various transformations and parameterizations of the input sequence. The primary motivation was to present the neural network with an input that encapsulates more meaningful features from the raw signal. Two prominent methods we experimented with were the Fourier transform and Taylor series approximation.

The Fourier transform was our initial approach, converting the time-domain signal into the frequency domain. The mathematical formulation of the discrete Fourier transform (DFT) we used is given by:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j2\pi kn/N}$$

where $x[n]$ is the input sequence, N is the total number of samples, and $X[k]$ represents the frequency bin. Although this technique decomposes the signal into its constituent frequencies, we observed that it reduced the accuracy of our models on the task of basecalling. Given that this type of information is highly successful for tasks such as speech recognition, we attribute this to a loss of information with respect to the original signal. This could be because the Fourier transform focuses primarily on the frequency components and might not retain the phase information as effectively as needed.

Acknowledging the limitations of the Fourier transform in our context, we considered the Taylor series approximation as an alternative. The Taylor series provides a polynomial approximation of the function around a specific point, effectively capturing the local behavior of the signal. The n-th degree Taylor polynomial for a function $f(x)$ around a point a is defined as:

$$T_n(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n$$

For our discrete input signal, we calculated the finite differences to obtain the derivatives up to the desired degree. This method was compelling because it maintains the positional height information of the signal and approximates the local trends, which could be beneficial for our sequence-to-sequence models.

Initial testing involved converting small chunks of the input signal into their polynomial approximations and using these as inputs to our neural network models. This was useful on the small dataset, and a simple change for the Bonito model, so we tested it on the 52GB dataset. On the 52GB dataset, this lead to an increase in validation accuracy, but the impact on throughput and memory usage was drastic, as we could not compute it efficiently on our GPUs, so it was not pursued further.

The best hyper-parameters for this Taylor approximation we could find were using a 17-element sliding window (of stride 1) and a Taylor polynomial of degree 81.

5 Novel architectures for basecalling

In this chapter we design new architectures for the goal of pushing the Pareto optimal line of throughput and accuracy for the task of basecalling. We compare our proposed model to two baselines, Bonito and SACall, using the 52GB dataset, and by varying their model dimensionalities in order to adjust their position in the accuracy/throughput trade-off. This is based on looking at Oxford Nanopore's multiple Bonito architectures made available, and noticing that in order to adjust that trade-off themselves in the direction of increased throughput, they do in fact lower the dimensionality of the model and sometimes make no other changes at all. The Bonito-fast model, for example, has a dimensionality of 96, which is why it is included in table 5.1 in this chapter, alongside the one with dimensionality 384, which is considered the "high-accuracy" model by Oxford Nanopore.

5.1. ParallelRNN

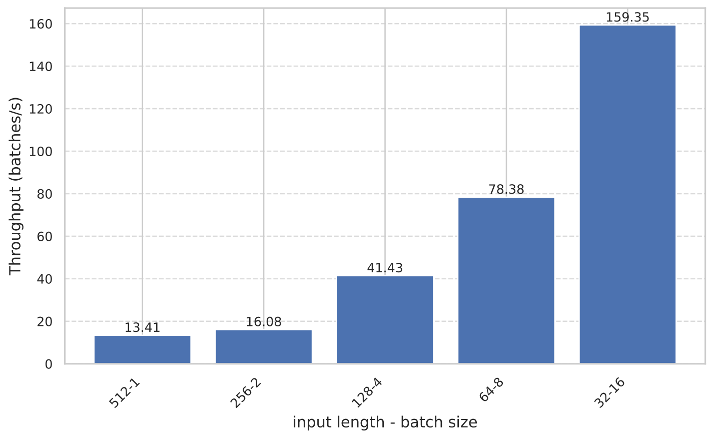


Figure 5.1: Throughput measurements of a 5-layer LSTM with input and output dimensions 384 (as in Bonito) where we repeatedly divide the sequence length by two by concatenating two halves of the input sequence along the batch dimension, starting with 512 to be close to the actual Bonito sequence length of 400. Run on Nvidia 3090 GPU. Average of 10 iterations.

5.1.1. LSTM performance and parallelization

In our goal of finding more efficient architectures than LSTMs, the problem we are trying to solve is the theoretical limitation of the recurrent dependence on all previous outputs, which means for any element j of its input sequence, the LSTM requires the computation of all previous inputs regardless of their relevance to the current element. This is illustrated in Figure 5.1, where we see that an input sequence that is split up and concatenated along the batch dimension before being computed by the LSTM can be computed up to an order of magnitude faster. To explore this further, we implement a variant of the LSTM layer, where we first split the input sequence into n chunks, and then concatenate the chunks along the batch dimension so they are computed in parallel. Since the goal is still to process as many input batches per second as we can, and increasing the batch size increases throughput for LSTMs, we measure throughput for different input batch sizes in Figure 5.2. Initially we see an advantage, which is the effect of parallelization, at small batch sizes. The advantage of using this layer seems to disappear as the batch size of the input sequence increases. At batch size 256 and after, the models have near-equal throughput.

There are multiple reasons this could conceivably happen, but because the different models perform practically the same number of floating point operations, the performance stagnation must be due to data locality issues such as memory bandwidth or cache sizes. The differences between all models on the figure can also only be attributed to this, because computationally they only differ in terms of the order in which the floating point operations are made, so that Split LSTM allows a greater proportion of data (set by the splitting factor) to be computed in parallel.

We then halve the precision of the model and its inputs, meaning all floating points then occupy half the space. It goes from using 32-bit full-precision floating points to using 16-bit half-precision floating points. It also is able to double its compute throughput, as instead of using 16-bit tensor cores with 32-bit accumulates, it uses 32-bit accumulates, which on our GPU offers double the theoretical peak FLOPS. We will show that this corresponds directly to what we see on the graph. We approximate the FLOP count of an LSTM layer using the following equation:

$$\text{FLOP count}_{\text{fwd}} = ((d + h) \times (4 \times h)) \times \text{seq\_len} \times \text{num\_lay}$$

Here, d is the dimensionality of the input, h of the hidden state (both equal in our case), seq_len is the length of the input sequence and num_lay the number of layers. We are only considering the matrix multiplications in the LSTM cell, as the rest of the operations are comparably negligible at our scale.

Plugging in our variables in this equation and dividing it by the throughput we've measured in Figure 5.2, we find peak FLOPS of around 181 in float32, and 363 for float16. This means we reach peak theoretical FLOPS for our GPU, but also that if our GPU had less memory, only the Split LSTM would achieve peak FLOPS.

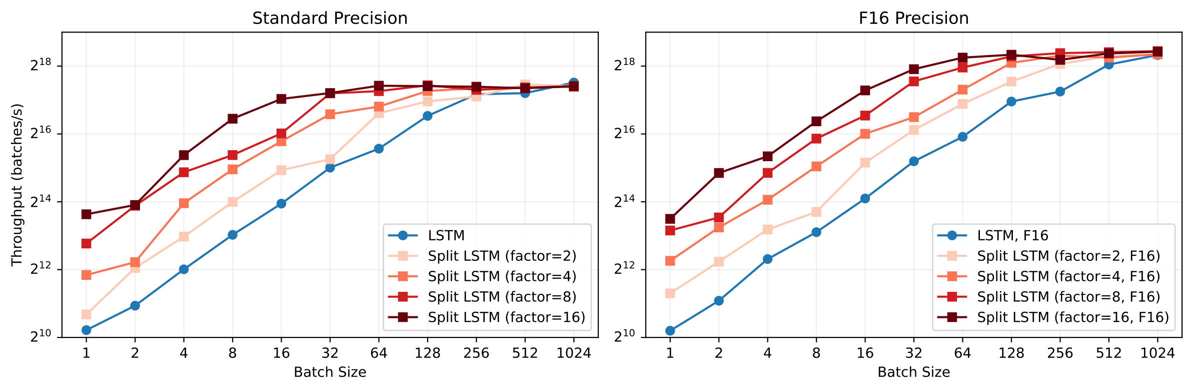


Figure 5.2: Throughput measurements of two LSTM layers with same dimensions as Figure 5.2 and "Split LSTM" where the input is split into two along the sequence dimension and concatenated along the batch dimension. Run on Nvidia 3090 GPU. Left uses full-precision (32-bit) floating points, right has half-precision (16-bit) floating points. Average of 32 iterations. Split LSTM is put through PyTorch's JIT compiler to minimize split-related overhead.

When we switch to 16-bit floating points, the Split LSTM model preserves its advantage over the default LSTM for input batch sizes more than 2x bigger. In summary, we find that the advantage of parallelization over recurrence is reaching higher throughput at earlier batch size. Batch size here is effectively a way of increasing GPU occupancy until peak FLOPS are reached, meaning that before reaching that threshold the GPU is slowed down by having to wait for previous iterations. This also implies that to get good LSTM performance (and therefore maximize basecalling throughput using Bonito), hardware should be picked such that a batch size large enough to reach peak FLOPS can be run, and where memory bandwidth is large enough to support what is essentially one big matrix multiplication (in contrast to, say, the attention operation, where bandwidth constraints are more difficult [17]).

In fact, deeper analysis using Nsight Systems on an Nvidia T4 GPU shows that once the model reaches its maximum throughput, the DRAM bandwidth is only around 12% of maximum bandwidth (for a GPU with 320GB/s bandwidth) and general compute utilization of about 40% (including tensor cores and active warps).

5.1.2. Parallel formulation of RNNs

After using an inverse gradient method on an LSTM-based basecalling model, we noticed that even though the LSTM could use an arbitrarily long amount of its previously-seen inputs, it seems to periodically activate its forget gate and thereby restrict itself to using a range of around 3 of its input vectors.

Inspired by the Split LSTM and the advantages of parallelization, we propose the insight that a single step of the LSTM model can be computed in parallel, if it is applied to all elements of a sequence at once. By doing this for multiple iterations, the LSTM operation can be run in parallel, at the cost of a much reduced receptive field. Since base predictions generally do not seem to depend on data more than 5 bases away from the current one when learning the underlying signal function, and the CRF decoder takes explicit care of modeling the sequence probability distribution (see section 3.1 for this terminology) this should not result in a significant reduction in accuracy. If we consider one LSTM cell iteration as a single function, the algorithm looks like this:

Algorithm 1 Parallel LSTM Pseudocode

```
Require: Input sequence $X \in \mathbb{R}^{T \times D}$, where $T$ is the length and $D$ is the dimensionality
Ensure: Output hidden states $H \in \mathbb{R}^{T \times D}$ and cell states $C \in \mathbb{R}^{T \times D}$
 1: Initialize $H \leftarrow$ zeros$(T, D)$
 2: Initialize $C \leftarrow$ zeros$(T, D)$
 3: for j = 1 to number of iterations do
 4:     for i = 1 to T do
 5:         $(H[i], C[i]) \leftarrow LSTM(X[i], H[i], C[i])$
 6:     end for
 7:     for i = 1 to T - 1 do
 8:         $H[i] \leftarrow H[i+1]$
 9:         $C[i] \leftarrow C[i+1]$
10:     end for
11: end for
12: return $H, C$
```

Note that the second and third for loops do not have data dependencies along their iterations, so they can be run in parallel. This is how we implemented the layer. When implemented this way, the LSTM cell computes all gates and outputs with sequence and batch-wise parallelism. There is also an increase in total computation, because a single iteration, for sequence length T computes T times the LSTM cell function, which is as many as a standard LSTM layer, and we want to do this for multiple iterations in order to profit from the advantage of LSTM's at basecalling (see Chapter 4).

Preliminary performance benchmarks of this approach reveal that a lack of custom CUDA kernels like used in PyTorch's official LSTM implementation make it hard to compare both algorithms directly. To account for this discrepancy, we compare our models to three different LSTM implementations:

- **Bonito** is the Bonito architecture using an implementation of LSTM using Pytorch base layers
- **BonitoJIT** uses PyTorch's Just-in-Time compilation to perform automated kernel fusion and other techniques to automatically optimize our custom LSTM implementation
- **BonitoOptimized** uses the default PyTorch LSTM layer with its custom kernels, and is the implementation used until now in this thesis

We choose to use the parallel LSTM as a drop-in replacement for the LSTM in the Bonito architecture for a direct comparison. Results are shown in table 5.1. Our model's training dynamics were similar to LSTMs and the gradients were even more stable than LSTMs. We ascribe this to the shorter sequence length of individual cell outputs, which give less opportunity for exploding or vanishing gradients. The accuracy, however, is not on the level of Bonito, but is still superior to transformer-based SACall, showing further that it retains its advantage over alternative architectures. Given this architecture is essentially an LSTM where each output element is computed from a small number of input elements, these results indicate that the remaining accuracy delta with standard Bonito is related to the receptive field of the LSTM cells (meaning, the length of the previous context they use to generate their final output).

Model Name	**Dimensionality**	**Validation Accuracy**	**Generalization Accuracy**
SACall	96	0.826	0.747
	128	0.847	0.765
	256	0.866	0.779
	512	0.869	0.792
SACall CoT (4 layer)	**256**	0.872	0.790
Bonito/best distill	64	0.871/0.851	0.817/0.810
	96	0.878/0.871	0.828/0.828
	128	0.883/0.879	0.840/0.839
	256	0.893/0.897	0.856/0.863
	384	0.899/0.905	0.899/0.871
	512	0.901/**0.906**	0.869/**0.871**
Bonito GRU/best distill	64	/0.847	/0.785
	96	0.875/0.865	0.823/0.813
	128	0.879/0.873	0.836/0.826
	384	0.894/0.900	0.833/0.866
	512	/0.901	/0.871
Bonito (parallel LSTM)	64	0.858	0.775
	96	0.859	0.794
	128	0.862	0.789
	384	0.874	0.801
DenseBaseConv	64	0.861	0.786
	96	0.868	0.799
	128	0.873	0.812
	256	0.881	0.828
	384	0.885	0.835
	512	0.887	0.840
Bonito ParallelRNN	64	0.808	0.706
	96	0.827	0.727
	128	0.839	0.741
	256	0.858	0.761
	384	0.864	0.748
	512	0.860	0.746
Bonito 1stconvkan	384	0.894	0.857
2ndconvkan	384	0.899	0.865
2ndconvkan + wider + sinekan	**384**	0.899	0.867

Table 5.1: Validation and generalization (on human DNA) accuracy of different models after training on the 52GB dataset for 5 epochs. Dimensionality in bold when it's the dimensionality used in [61]. Bold numbers in dimensionality column indicate the best-performing dimensionality in the model group, bold numbers in accuracy columns indicate best model of all.

5.1.3. ParallelRNN

To push this further, we decided to use the insights found in the GRU ablations performed in Chapter 4 (see table 4.2.1) in order to modify this architecture. First, we started with a parallel implementation of the GRU recurrent model. The z and r gates (see equations 2.13 and 2.14 respectively) were removed to improve throughput. Because the model does not go through a large number of layers, this should have even less of an impact on accuracy, and to deal with instability caused by recurrent iterations, the layer output is normalized using a GroupNorm [79] layer, which struck a balance between minimizing validation loss and the impact on throughput.

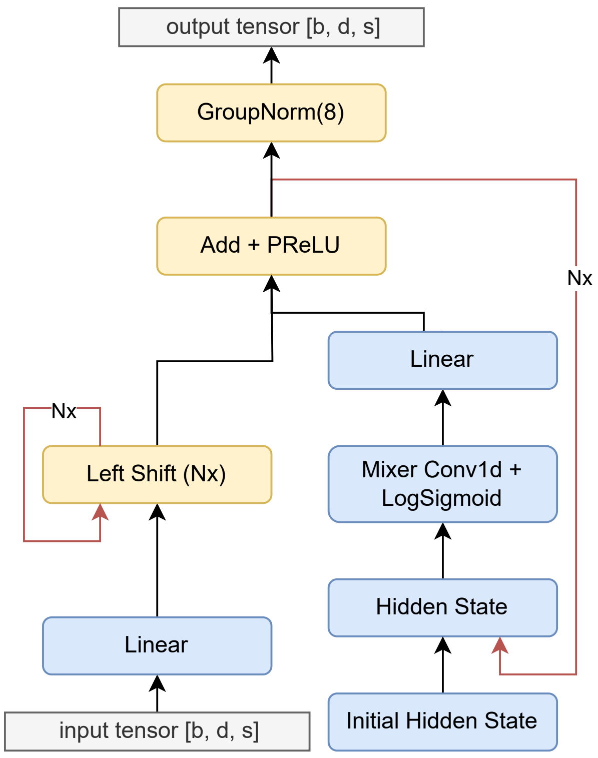


Figure 5.3: The architecture of the ParallelRNN layer, meant as a replacement for the LSTM layers in the Bonito model architecture. Nx denotes the multiple recurrent iterations, so at each nth iteration, we use the output of the iteration as the new hidden state, and the nth left-shifted output of the Linear layer on the lower left. Mixer convolution is a convolution for which in the final ParallelRNN models in Table 5.1 the kernel size was 5, and the initial hidden state is a trainable vector that is duplicated to have the same shape as the input tensor. The input tensor has shapes b, d, and s, which correspond respectively to batch size, model dimensionality, and sequence length. Activation functions were determined through trial and error.

Another improvement inspired from table 4.2.1 aiming to increase accuracy by increasing the receptive field of the recurrent cells was to add what we term a 'mixer' layer, which is a 1d convolution applied to the hidden state before each iteration. This mixer allowed to reach validation losses remarkably lower than the strong GRU baseline with a number of iterations set only to 3, as seen on Figure 5.4. This finding did not transfer to the larger dataset, where mixer width did not have the same effect on accuracy, which was simply lower than all tested dimensionalities of the LSTM-based Bonito (see table 5.1), though it does show a better pareto frontier than SACall (see Figure 6.1). Further results are discussed in the next chapter. This is after tuning the mixer kernel size, which was found to cause low throughput and had to be reduced to 5. In the final model, we use two ParallelRNN layers, as this was determined to yield the best accuracy/throughput trade-off, and the architecture is depicted in detail in Figure 5.3.

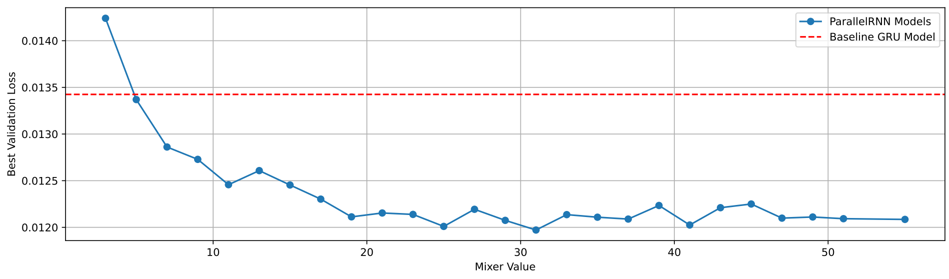


Figure 5.4: Validation accuracy of ParallelRNN model on 2GB dataset depending on hyperparameter 'mixer' compared to a GRU-based baseline.

5.2. DenseBaseConv

Another idea inspired by the analysis in Chapter 3 was based on the hypothesis that in the pursuit of higher throughput/accuracy ratios, one might want the 'computational budget' to be spent where the most information is present to make predictions. By focusing on learning the signal function, the model would be able to avoid spending computation on learning longer-distance relationships in the data. For example, the architecture of the LSTM is engineered to retain information from as many inputs back as there are in the sequence, and the transformer layer is meant to learn relationships on the whole sequence, forwards and backwards.

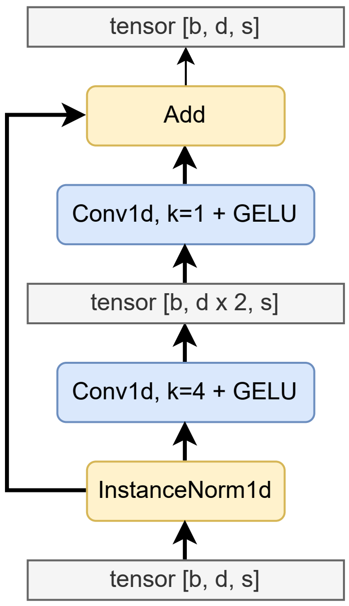


Figure 5.5: Architecture of a single DenseBaseConv layer parameters b, d, and s are the batch size, dimensionality, and sequence length of the input. Parameter k is the kernel width. All convolution layers have padding chosen such that it does not change the length of the sequence, and the stride is always set to one.

In order to focus on local relationships, the initial version of the DenseBaseConv (Dense Base Convolution) block was an instance norm, then a 1d convolution with kernel size 2 and activation function GELU, and two linear layers with GELU activations, and finally a residual connection from the output of the instance norm to the output of the last linear layer. This architecture initially achieved better validation accuracy than the LSTM baseline on the 2GB dataset, so it was developed further.

Further changes were made to the CNN convolutions when compared to the base Bonito architecture, including replacing the SiLU activations with LeakyReLU activations, reducing the first two convolution kernel sizes to 3 and 4, and changing the stride of the second convolution to 3, which increases the sequence length to 666. We also add a Batch Norm after each of the two convolutions, and after the DenseBaseConv block.

In the final architecture, shown in Figure 5.5 there is only one linear layer, and the kernel size of the convolution is set to 5 There is one DenseBaseConv block and one LSTM from the original Bonito architecture. This was the trade-off we found was best to maximize throughput and accuracy. Its accuracy measurements on the 52GB dataset are shown in table 5.1 as DenseBaseConv and on Figure 6.1. As can be seen, the model performs close to, but not as well as, the optimized Bonito model.

6

Overall experimental results

In this chapter, we present and discuss the overall experimental results obtained from our exploration of sequence-to-sequence models for basecalling. We focus on two key aspects: the impact of model dimensionality and scaling, and the throughput-accuracy trade-off offered by the models under consideration.

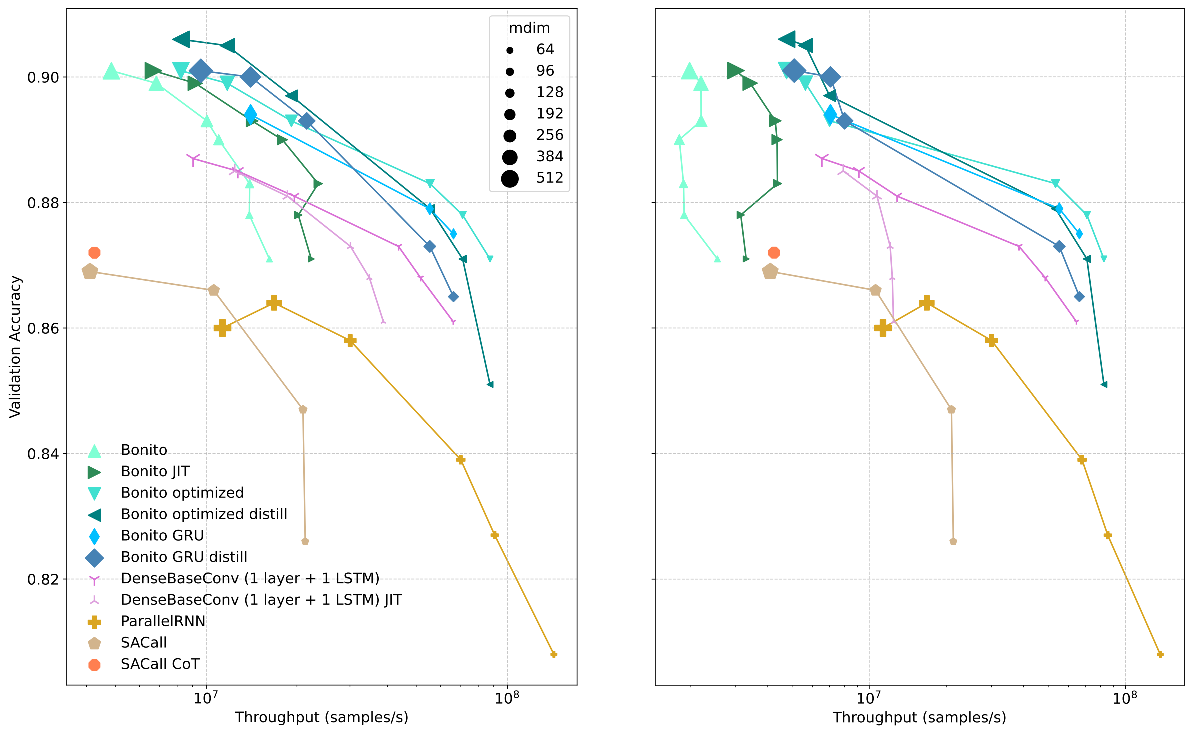


Figure 6.1: Validation accuracy and throughput for different models, on the **left** we sweep through batch sizes of increasing powers of two until 1024, after which we increase by 256, and finally we pick the largest throughput regardless of batch size. On the **right** we show throughputs at batch size 512, because this is the batch size a practitioner would face when using a MinION nanopore sequencer (see Section 1.1). We show the main two baselines, SACall and Bonito, two models we developed with the goal to push the Pareto frontier, DenseBaseConv and ParallelRNN, and one model Bonito GRU where we simply replace the optimized LSTM layer with an optimized GRU layer. Also present are distilled versions of Bonito and Bonito GRU, showing their utility in surpassing the pareto frontier, and an enhanced version of SACall implementing a chain-of-thought mechanism and thereby beating the transformer-based pareto curve. All models are benchmarked with JIT, Inductor and Cudagraphs compilation (when supported, but JIT compilation rarely works when applied to a whole model), and without compilation, and the fastest throughput value among them is chosen. All benchmarks are averages of 3 runs during which the model processes 16 groups of batches (on the rightward plot, one batch group contains 512 sequences). See Section 6.2 for clarifications of terms ”optimized” ”JIT”, and other considerations affecting throughput. All benchmarks run on RTX 3090.

6.1. Discussion on dimensionality and scaling accuracy

One of the primary challenges encountered during our experiments was the scaling behavior of the models when transitioning from small to large datasets. The ParallelGRU model, which showed promising results on the smaller 2GB dataset, required significant adjustments to maintain its performance on the larger 52GB dataset. Similarly, the DenseBaseConv model, despite its initial success, struggled to achieve comparable accuracy levels when applied to the larger dataset. Local-attention, as seen in table 5.1 becomes more unstable as dimensionality is increased, but we have not found a reason why.

These observations highlight a crucial aspect of model development for basecalling: the effective utilization of increased dimensionality. As the dataset size grows, the models' ability to leverage the additional information becomes a critical factor in their performance. Our current approaches, while effective on smaller datasets, do not seem to scale well with increased dimensionality.

6.2. Levels of optimization

Throughput-accuracy comparisons like in Figure 6.1 are difficult to interpret for a few reasons. One issue is of fused operations. Essentially, the usual case is that when two operations are to be performed and the results of one are used in the second operation, then the first operation is performed, the relevant data is taken from the GPU memory into the on-chip memory, the operation is performed, and the results are brought back to GPU memory, after which the second operation will again retrieve the data from GPU memory, causing extra waiting time. A fused operation, in our case written as a CUDA kernel, allows the second operation to be performed before the data from the first operation is copied back into GPU memory, hence having a positive impact on throughput. BonitoOptimized, for example, computes one LSTM iteration, including non-linearities and gate point-wise multiplications, in a single CUDA kernel call. ParallelRNN however, uses distinct kernel calls for the different operations that are performed on all sequence vectors in parallel.

Another issue is of automated optimizers. the PyTorch deep learning framework provides a just-in-time compiler (JIT) [65] which attempts to use compiler techniques to optimize the execution algorithm of the model, i.e. by fusing multiple operations into a single kernel. There are multiple compilers available, such as the Cudagraphs [30] or the PyTorch Inductor [64] compiler. Whether these methods are used or not, sometimes determines the final measured throughput. We find that the impact of these systems is largely dependent on the specific network type, so that an LSTM layer is slowed down by both the Inductor and Cudagraphs optimizers, while the version using custom kernels (as BonitoOptimized does) is much faster.

Finally, there are sources of overhead that lead, for example, the unoptimized Bonito and JIT Bonito models to actually decrease in throughput as their dimensionality is reduced, which could be related to different kernels being incompatible with certain specific dimensionalities. In this case, it seems the issue is related to computational intensity, as the greater batch size on the left side of the figure causes both models to behave more closely to the optimized Bonito models.

To clarify the comparability of the different models presented here, we detail the optimization levels of the multiple models compared in this chapter:

- **All models**: CRF and CTC modules utilize a CUDA-optimized decoder provided by Seqdist, and CNNs are composed of individual convolution layers which use their own kernels to execute (PyTorch official kernels).
- **Bonito**: Per-iteration operations of the LSTM layers are individual PyTorch modules, each with their own kernel.
- **BonitoJIT**: The LSTM layers are the same as in the unoptimized Bonito model, but they are put through PyTorch's JIT compiler before running benchmarks
- **BonitoOptimized**: LSTM layers are the PyTorch official ones, meaning individual iterations are performed in a single kernel call.
- **ParallelRNN**: The RNN component is made of individual PyTorch modules, each having their own kernels.

- **DenseBaseConv**: The DenseBaseConv blocks (see Figure 5.5) are made of individual PyTorch modules, and automated optimizers did not yield statistically significant increases in throughput.
- **DenseBaseConv JIT**: Same as DenseBaseConv, but the LSTM is the same as in Bonito JIT
- **SACall**: Transformer layers are highly optimized PyTorch official implementations, which use the highly optimized FlashAttention kernels [17]. Automated optimizers did not yield statistically significant increases in throughput.
- **SACall CoT**: Same as SACall

6.3. Throughput-accuracy trade-off

In this section, we analyze the results presented in Figure 6.1, discussing the performance of various models in terms of their throughput-accuracy trade-offs. We will examine these results in several parts, focusing on the impacts of distillation, the performance of SACall, and the potential of our proposed ParallelRNN and DenseBaseConv architectures.

6.3.1. Impact of Distillation

Distillation emerges as the most significant contribution of this thesis, with particularly strong results observed for the base Bonito model. The distilled versions of Bonito consistently outperform their non-distilled counterparts, with the exception of smaller models, including the 'bonito-fast' configuration.

For the Bonito GRU model at dimensionality 384, we observe a throughput increase of approximately 20% ($7016 \div 5868 \approx 1.20$) compared to the optimized Bonito model. This demonstrates that we have successfully expanded the Pareto frontier not only with the existing Bonito model but also with a new architecture (the GRU Bonito).

Notably, at batch size 512, which corresponds to the practical scenario of a MinION nanopore sequencer, the distilled Bonito GRU model surpasses the Pareto curve of all other Bonito variants in the high-accuracy regime (dimensionality 384). It achieves a throughput increase of approximately 25% ($3529 \div 2816 \approx 1.25$) while maintaining slightly higher accuracy.

6.3.2. SACall Performance

The results indicate that almost all models outperform SACall, suggesting relatively weak performance for this transformer-based approach. However, it's interesting to note that the implementation of a chain-of-thought mechanism improves SACall's Pareto curve. While we were unable to train more variants of this model due to time constraints, this observation supports the potential of research directions motivated by comparisons of models along the Chomsky hierarchy for finding more efficient sequence-to-sequence models.

6.3.3. ParallelRNN and DenseBaseConv Performance

The ParallelRNN model achieved the highest throughput among all tested models. While its accuracy is lower than that of the 96-dimensional Bonito model by about 2.6% ($0.864 \div 0.878 \approx 0.974$), it appears to potentially beat the Pareto optimal curve of the optimized Bonito, especially when the batch size is limited to 512.

An important observation is the significant difference in performance between unoptimized and optimized Bonito architectures. This suggests that both ParallelRNN and DenseBaseConv could benefit substantially from custom kernel implementations, particularly ParallelRNN due to its unique computation method.

To further explore optimization potential, we benchmarked DenseBaseConv using the JIT-compiled unoptimized LSTM implementation. We found that JIT compilation has a more pronounced effect at smaller batch sizes and dimensionalities.

When compared to JIT-compiler-optimized Bonito, which represents a more comparable level of optimization, our DenseBaseConv model surpasses the Pareto curve:

- In a constrained batch size scenario (512), all tested DenseBaseConv models outperform Bonito. The best DenseBaseConv achieves approximately 80% throughput increase ($3960 \div 2191 \approx 1.80$)

with accuracy between Bonito's 128 and 192 model dimensionalities. This means we outperform the Bonito-fast architecture in this scenario, which is particularly relevant for the MinION sequencer's output of 1.6×10^6 bases per second.

- In unconstrained scenarios, we surpass the smallest Bonito JIT model with a speedup of approximately $3.64\times$ ($6037 \div 1654 \approx 3.64$) while maintaining slightly better accuracy.

These results suggest that DenseBaseConv could offer a better trade-off than Bonito in some situations, particularly very-high-throughput scenarios. Furthermore, it implies lower hardware requirements at high-throughput regimes for larger numbers of concurrent nanopores.

In conclusion, our proposed models, particularly when optimized, show promising potential to push the Pareto frontier of throughput-accuracy trade-offs in basecalling. The success of distillation techniques and the performance of our novel architectures open up new avenues for improving basecalling efficiency, especially in resource-constrained scenarios.

7 Conclusions and future work

7.1. Conclusions

In this thesis, we explored various sequence-to-sequence models and their potential for improving the accuracy and throughput trade-off in the task of basecalling for nanopore sequencing. Our investigations led to several key findings:

- Distillation has allowed us to go beyond the state-of-the-art in throughput/accuracy, including with new models (GRU-based Bonito) and at comparable levels of optimization, a new architecture (ParallelRNN).
- Transformer-based models, particularly those enhanced with chain-of-thought mechanisms and locality-aware attention, showed only slight improvements in accuracy compared to baseline models. These modifications demonstrate the potential for further optimization of transformer architectures for basecalling, but they also show that the transformer network is not very well adapted for the task of basecalling at this scale of throughput.
- Recurrent neural networks, specifically LSTMs and GRUs, remain strong contenders for basecalling at high-throughput regimes. Our ablation studies provided insights into the crucial components of these architectures that contribute to their effectiveness.
- The analysis of computational properties and hardware utilization revealed that memory bandwidth requirements for basecalling are relatively low, suggesting that consumer-grade GPUs could be a cost-effective solution for this task.

Overall, our findings emphasize the importance of considering both architectural innovations and hardware characteristics when designing basecalling models. By carefully balancing accuracy and throughput, and leveraging insights from our experiments, future research can continue to push the boundaries of efficient and accurate basecalling for nanopore sequencing.

7.2. Future work

This thesis suggests the following avenues for potentially fruitful research on this topic:

- Developing custom CUDA kernels and optimized implementations for promising architectures like ParallelRNN and DenseBaseConv could significantly enhance their performance and make them more competitive with state-of-the-art models.
- Exploring variants of RNN models (as there are many in the literature) would be the next logical domain to explore in order to search for more efficient basecalling architectures (as long as custom CUDA kernels are also written for these RNN models, and given the right compute budget)
- Conducting a more extensive examination of the CATCaller architecture, which showed promising results but could not be fully explored due to time constraints, may provide additional insights

and opportunities for improvement, perhaps through integration with more efficient transformer architectures from the literature, such as the local-attention operation.

- Exploring the use of consumer-grade GPUs and parallelization strategies across multiple GPUs could lead to more cost-effective and scalable basecalling solutions. Especially for LSTMs, it seems using one GPU for the first half of a sequence and the next GPU for the other half would increase the ratio of data in memory which can be computed in parallel, thereby increasing utilization until the GPU reaches peak theoretical FLOPS. Exploring the newest GPU architectures would also be important, to understand how compute resource trade-offs have changed.
- Exploring the use of cheaper accelerators would also make sense, as we found that FLOPS was the main bottleneck in running Bonito and memory bandwidth was not, which is opposite of what modern GPUs are focused on (because they target most primarily transformer networks, which are more bandwidth-bottlenecked than compute). Accelerators similar to Meta's Next Gen MTIA [73] could offer better trade-offs as they are designed with FLOPS and total memory as a bigger priority than memory bandwidth.
- Creating an efficient GPU implementation of Taylor approximations (or a training/inference setup that performs it on the CPU) could be a catalyst for getting better accuracy from existing models, as an alternative to the discrete Fourier transform which did not work for us.
- Examining large-batch-size custom kernels for Transformer inference, while existing techniques often focus on latency at batch sizes close to 1 and dimensionalities higher than 1000, Basecalling requires at least a batch size of 512 and a dimensionality of 128 could suffice, meaning that a good execution algorithm should be able to maximise utilisation and minimize the overhead associated with low-dimensional heads. It does not seem like this is currently the case.
- Models seem to struggle in effectively using their dimensions past a dimensionality of 128. It would seem that specialised regularisation techniques could be a useful way to get more accuracy out of bigger models.

By pursuing these research directions, we can continue to advance the field of basecalling for nanopore sequencing, enabling faster, more accurate, and more accessible genomic analysis. The insights gained from this thesis lay the foundation for future innovations in sequence-to-sequence modeling and their application to the crucial task of basecalling.

References

[1] Zeyuan Allen-Zhu and Yuanzhi Li. “Physics of Language Models: Part 1, Learning Hierarchical Language Structures”. In: *arXiv preprint arXiv:2305.13673* (2023). V2+V3 polishes writing; V3 includes Figures 6 and 10 for better illustrations of our results. URL: `https://doi.org/10.48550/arXiv.2305.13673`.

[2] M.Z. Alom et al. “A State-of-the-Art Survey on Deep Learning Theory and Architectures”. In: *Electronics* 8.3 (2019), p. 292. DOI: `10.3390/electronics8030292`.

[3] Quentin Anthony et al. “BlackMamba: Mixture of Experts for State-Space Models”. In: *arXiv preprint arXiv:2402.01771* (Feb. 2024). Version 1. URL: `https://doi.org/10.48550/arXiv.2402.01771`.

[4] Maximilian Beck et al. “xLSTM: Extended Long Short-Term Memory”. In: *arXiv preprint arXiv:2405.04517* (2024). URL: `https://doi.org/10.48550/arXiv.2405.04517`.

[5] Peter Belcak and Roger Wattenhofer. “Fast Feedforward Networks”. In: *arXiv preprint arXiv:2308.14711* (Aug. 2023). Version 2, revised in September 2023. URL: `https://doi.org/10.48550/arXiv.2308.14711`.

[6] Lucas Beyer. *On the speed of ViTs and CNNs*. `http://lb.eyer.be/a/vit-cnn-speed.html`. 2024.

[7] Simone Bianco et al. “Benchmark Analysis of Representative Deep Neural Network Architectures”. In: *IEEE Access* 6 (Oct. 2018), pp. 64270–64277. DOI: `10.1109/ACCESS.2018.2877890`.

[8] James Bradbury et al. “Quasi-Recurrent Neural Networks”. In: *arXiv preprint arXiv:1611.01576* (2016). `https://doi.org/10.48550/arXiv.1611.01576`.

[9] Baohua Cao et al. “Structure of the nonameric bacterial amyloid secretion channel”. In: *Proceedings of the National Academy of Sciences* 111.50 (2014), E5310–E5318. DOI: `10.1073/pnas.1411942111`.

[10] Jang Hyun Cho and Bharath Hariharan. “On the Efficacy of Knowledge Distillation”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 4794–4802.

[11] Kyunghyun Cho et al. “On the Properties of Neural Machine Translation: Encoder-Decoder Approaches”. In: *arXiv preprint arXiv:1409.1259* (Sept. 2014). DOI: `10.48550/arXiv.1409.1259`. URL: `https://doi.org/10.48550/arXiv.1409.1259`.

[12] François Chollet. “Xception: Deep Learning with Depthwise Separable Convolutions”. In: *arXiv preprint arXiv:1610.02357* (Apr. 2017). Version 3. URL: `https://doi.org/10.48550/arXiv.1610.02357`.

[13] K. Choromanski et al. “Rethinking Attention with Performers”. In: *arXiv preprint arXiv:2009.14794* (2021). Published as a conference paper and oral presentation at ICLR 2021. URL: `https://doi.org/10.48550/arXiv.2009.14794`.

[14] Zihang Dai et al. “Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context”. In: *arXiv preprint arXiv:1901.02860* (Jan. 2019). DOI: `10.48550/arXiv.1901.02860`. URL: `https://doi.org/10.48550/arXiv.1901.02860`.

[15] T. Dao. “FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning”. In: *arXiv preprint arXiv:2307.08691* (2023).

[16] T. Dao and A. Gu. “Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality”. In: *arXiv preprint arXiv:2405.21060* (2024).

[17] T. Dao et al. “FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness”. In: *arXiv preprint arXiv:2205.14135* (2022).

[18] Yann N. Dauphin et al. “Language Modeling with Gated Convolutional Networks”. In: *arXiv preprint arXiv:1612.08083* (2017). https://doi.org/10.48550/arXiv.1612.08083.

[19] Sam Davis. *Transforming Basecalling in Genomic Sequencing*. https://nanoporetech.com/about-us/blog/transforming-basecalling-genomic-sequencing. Accessed: 2020-XX-XX. 2020.

[20] G. Delétang et al. “Neural networks and the Chomsky hierarchy”. In: *arXiv preprint arXiv:2207.02098* (2022).

[21] A. Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *arXiv preprint arXiv:2010.11929* (2021).

[22] Jeffrey L. Elman. “Finding structure in time”. In: *Cognitive Science* 14.2 (Mar. 1990), pp. 179–211.

[23] Karl Pearson F.R.S. “LIII. On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: 10.1080/14786440109462720.

[24] Alhussein Fawzi et al. “Discovering faster matrix multiplication algorithms with reinforcement learning”. In: *Nature* 610 (2022). Published: 05 October 2022, pp. 47–53. URL: https://doi.org/10.1038/s41586-022-05172-4.

[25] Leo Feng et al. “Were RNNs All We Needed?” In: *arXiv preprint arXiv:2410.01201* (2024). https://doi.org/10.48550/arXiv.2410.01201.

[26] Mees Frensel, Zaid Al-Ars, and H Peter Hofstee. “Learning Structured Sparsity for Efficient Nanopore DNA Basecalling Using Delayed Masking”. In review for the ACM Conference on Bioinformatics, Computational Biology, and Health Informatics. 2024.

[27] Hasindu Gamaarachchi et al. “Squigulator: simulation of nanopore sequencing signal data with tunable noise parameters”. In: *bioRxiv* (2023). This article is a preprint and has not been certified by peer review. DOI: 10.1101/2023.05.09.539953.

[28] Aidan N. Gomez et al. “The Reversible Residual Network: Backpropagation Without Storing Activations”. In: *arXiv preprint arXiv:1707.04585* (July 2017). DOI: 10.48550/arXiv.1707.04585. URL: https://doi.org/10.48550/arXiv.1707.04585.

[29] Alex Graves et al. “Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks”. In: *Proceedings of the 23rd International Conference on Machine Learning (ICML ’06)*. June 2006, pp. 369–376. DOI: 10.1145/1143844.1143891. URL: https://doi.org/10.1145/1143844.1143891.

[30] Alan Gray. “Getting Started with CUDA Graphs”. In: *NVIDIA Developer Blog* (Sept. 2019). Available at https://developer.nvidia.com/blog/getting-started-with-cuda-graphs.

[31] A. Gu and T. Dao. “Mamba: Linear-Time Sequence Modeling with Selective State Spaces”. In: *arXiv preprint arXiv:2312.00752* (2024).

[32] A. Gu et al. “Efficiently Modeling Long Sequences with Structured State Spaces”. In: *arXiv preprint arXiv:2111.00396* (2022).

[33] Albert Gu et al. “HiPPO: Recurrent Memory with Optimal Polynomial Projections”. In: *Advances in Neural Information Processing Systems* 33 (2020). URL: https://proceedings.neurips.cc/paper/2020/hash/1160462a35df945d433ff7b044658700-Abstract.html.

[34] Ali Hassani et al. “Neighborhood Attention Transformer”. In: *arXiv preprint arXiv:2204.07143* (2022). To appear in CVPR 2023. URL: https://doi.org/10.48550/arXiv.2204.07143.

[35] Horace He et al. “FlexAttention: The Flexibility of PyTorch with the Performance of FlashAttention”. In: *PyTorch Blog* (2024). Available at https://pytorch.org/blog/flexattention.

[36] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the Knowledge in a Neural Network”. In: *arXiv preprint arXiv:1503.02531* (2015). NIPS 2014 Deep Learning Workshop. DOI: 10.48550/arXiv.1503.02531. URL: https://doi.org/10.48550/arXiv.1503.02531.

[37] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[38] Jordan Hoffmann et al. “Training Compute-Optimal Large Language Models”. In: *arXiv preprint arXiv:2203.15556* (Mar. 2022). Version 1. URL: `https://doi.org/10.48550/arXiv.2203.15556`.

[39] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366. DOI: `10.1016/0893-6080(89)90020-8`. URL: `https://doi.org/10.1016/0893-6080(89)90020-8`.

[40] Neng Huang et al. “SACall: A Neural Network Basecaller for Oxford Nanopore Sequencing Data Based on Self-Attention Mechanism”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 19.1 (Jan. 2022), pp. 614–623. DOI: `10.1109/TCBB.2020.3039244`. URL: `https://doi.org/10.1109/TCBB.2020.3039244`.

[41] Peter J. Huber. “Robust Estimation of a Location Parameter”. In: *Annals of Mathematical Statistics* 35.1 (Mar. 1964), pp. 73–101. DOI: `10.1214/aoms/1177703732`. URL: `https://doi.org/10.1214/aoms/1177703732`.

[42] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *arXiv preprint arXiv:1502.03167* (2015). `https://doi.org/10.48550/arXiv.1502.03167`.

[43] Jennifer Jiang, Katherine H. Park, and Kiranmayi Vemuri. *Molecule of the Month: DNA-Sequencing Nanopores*. `https://pdb101.rcsb.org/motm/261`. Produced as part of a boot camp hosted by the Rutgers Institute for Quantitative Biomedicine. Sept. 2021.

[44] Rafal Jozefowicz et al. “Exploring the Limits of Language Modeling”. In: *arXiv preprint arXiv:1602.02410* (2016). URL: `https://doi.org/10.48550/arXiv.1602.02410`.

[45] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. “Reformer: The Efficient Transformer”. In: *arXiv preprint arXiv:2001.04451* (Jan. 2020). DOI: `10.48550/arXiv.2001.04451`. URL: `https://doi.org/10.48550/arXiv.2001.04451`.

[46] A. N. Kolmogorov. “On the representation of continuous functions of several variables as superpositions of continuous functions of a smaller number of variables”. In: *Dokl. Akad. Nauk* 108.2 (1956).

[47] John Lafferty, Andrew McCallum, and Fernando Pereira. “Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data”. In: *Proceedings of the 18th International Conference on Machine Learning (ICML ’01)*. June 2001, pp. 282–289.

[48] Benjamin Lefaudeux et al. *xFormers: A modular and hackable Transformer modelling library*. `https://github.com/facebookresearch/xformers`. 2022.

[49] Y. Li et al. “DeepSimulator: a deep simulator for Nanopore sequencing”. In: *Bioinformatics* 34.17 (2018), pp. 2899–2908. DOI: `10.1093/bioinformatics/bty223`.

[50] Y. Li et al. “DeepSimulator1.5: a more powerful, quicker and lighter simulator for Nanopore sequencing”. In: *Bioinformatics* 36.8 (2020), pp. 2578–2580. DOI: `10.1093/bioinformatics/btz963`.

[51] T. Lin et al. “A survey of transformers”. In: *AI Open* 3 (2022), pp. 111–132. DOI: `10.1016/j.aiopen.2022.10.001`.

[52] Ziming Liu et al. “KAN: Kolmogorov-Arnold Networks”. In: *arXiv preprint arXiv:2404.19756* (Apr. 2024). Version 4, revised in June 2024. URL: `https://doi.org/10.48550/arXiv.2404.19756`.

[53] Bruce T. Lowerre. “The HARPY Speech Recognition System”. Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy. Research supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense and monitored by the Air Force Office of Scientific Research (Contract F44520-73-C-0074). PhD thesis. Pittsburgh, Pennsylvania: Carnegie-Mellon University, Apr. 1976.

[54] Xuan Lv et al. “An End-to-end Oxford Nanopore Basecaller Using Convolution-augmented Transformer”. In: *bioRxiv preprint bioRxiv:2020.11.09.374165* (2020). This article is a preprint and has not been certified by peer review. URL: `https://doi.org/10.1101/2020.11.09.374165`.

[55] Pingchuan Ma et al. “Auto-AVSR: Audio-Visual Speech Recognition with Automatic Labels”. In: *arXiv preprint arXiv:2303.14307* (2023). `https://doi.org/10.48550/arXiv.2303.14307`.

[56] W. Merrill, J. Petty, and A. Sabharwal. "The Illusion of State in State-Space Models". In: *arXiv preprint arXiv:2404.08819* (2024). To appear at ICML 2024.

[57] William Merrill and Ashish Sabharwal. "The Expressive Power of Transformers with Chain of Thought". In: *International Conference on Learning Representations (ICLR)*. 2024. URL: `https://doi.org/10.48550/arXiv.2310.07923`.

[58] National Academy of Sciences. "From Molecules to Minds: Challenges for the 21st Century: Workshop Summary". In: *Proceedings of the National Academy of Sciences Workshop on From Molecules to Minds*. Washington, DC: National Academies Press, 2008, pp. 1–35. URL: `https://www.ncbi.nlm.nih.gov/books/NBK50991/`.

[59] Myle Ott et al. "fairseq: A Fast, Extensible Toolkit for Sequence Modeling". In: *Proceedings of NAACL-HLT 2019: Demonstrations*. 2019.

[60] Marc Pages. *Nanopore Benchmark for Basecallers*. `https://github.com/marcpaga/nanopore_benchmark`. 2024.

[61] M. Pagès-Gallego and J. de Ridder. "Comprehensive benchmark and architectural analysis of deep learning models for nanopore sequencing basecalling". In: *Genome Biology* 24 (2023), p. 71. DOI: `10.1186/s13059-023-02903-2`.

[62] Xuran Pan et al. "Slide-Transformer: Hierarchical Vision Transformer with Local Self-Attention". In: *arXiv preprint arXiv:2304.04237* (2023). Accepted to CVPR2023. URL: `https://doi.org/10.48550/arXiv.2304.04237`.

[63] B. Peng et al. "RWKV: Reinventing RNNs for the Transformer Era". In: *arXiv preprint arXiv:2305.13048* (2023).

[64] PyTorch Contributors. *torch.compiler: Graph Compilation for PyTorch 2.x*. Available at `https://pytorch.org/docs/main/torch.compiler.html`. PyTorch Foundation. 2023.

[65] PyTorch Contributors. *TorchScript: A Guide to Creating and Using TorchScript Models*. Available at `https://pytorch.org/docs/main/torchscript.html`. PyTorch Foundation. 2023.

[66] Eric A. F. Reinhardt, P. R. Dinesh, and Sergei Gleyzer. "SineKAN: Kolmogorov-Arnold Networks Using Sinusoidal Activation Functions". In: *arXiv preprint arXiv:2407.04149* v2 (July 2024). URL: `https://doi.org/10.48550/arXiv.2407.04149`.

[67] Albert Reuther et al. "Survey of Machine Learning Accelerators". In: *arXiv preprint arXiv:2009.00993* (2020). This article is a preprint and has not been certified by peer review. URL: `https://doi.org/10.48550/arXiv.2009.00993`.

[68] H. Salehinejad et al. "Recent Advances in Recurrent Neural Networks". In: *arXiv preprint arXiv:1801.01078* (2018).

[69] Siddharth Srivastava and Gaurav Sharma. "OmniVec: Learning robust representations with cross modal sharing". In: *arXiv preprint arXiv:2311.05709* (2023). `https://doi.org/10.48550/arXiv.2311.05709`.

[70] Jovan Stojkovic et al. "Towards Greener LLMs: Bringing Energy-Efficiency to the Forefront of LLM Inference". In: *arXiv preprint arXiv:2403.20306* (2024). This article is a preprint and has not been certified by peer review. URL: `https://doi.org/10.48550/arXiv.2403.20306`.

[71] Shizhao Sun et al. "On the Depth of Deep Neural Networks: A Theoretical View". In: *arXiv preprint arXiv:1506.05232* (2015). AAAI 2016.

[72] Y. Sun et al. "Retentive Network: A Successor to Transformer for Large Language Models". In: *arXiv preprint arXiv:2307.08621* (2023). URL: `https://doi.org/10.48550/arXiv.2307.08621`.

[73] Eran Tal et al. "Our next-generation Meta Training and Inference Accelerator". In: *Meta AI Blog* (Apr. 2024). `https://ai.meta.com/blog/next-generation-meta-training-inference-accelerator-AI-MTIA/`.

[74] Yi Tay et al. "Long Range Arena: A Benchmark for Efficient Transformers". In: *arXiv preprint arXiv:2011.04006* (2020). Submitted: 08 November 2020. URL: `https://doi.org/10.48550/arXiv.2011.04006`.

[75] A. Vaswani et al. "Attention Is All You Need". In: *arXiv preprint arXiv:1706.03762* (2017).

[76] Andrew Viterbi. “Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm”. In: *IEEE Transactions on Information Theory* 13.2 (1967), pp. 260–269. DOI: 10.1109/TIT.1967.1054010.

[77] S. Wang et al. “Linformer: Self-Attention with Linear Complexity”. In: *arXiv preprint arXiv:2006.04768* (2020).

[78] K. Willems et al. “Accurate modeling of a biological nanopore with an extended continuum framework”. In: *Nanoscale* 12.32 (2020), pp. 16775–16795.

[79] Yuxin Wu and Kaiming He. “Group Normalization”. In: *arXiv preprint arXiv:1803.08494* v3 (June 2018). URL: https://doi.org/10.48550/arXiv.1803.08494.

[80] Manzil Zaheer et al. “Big Bird: Transformers for Longer Sequences”. In: *arXiv preprint arXiv:2007.14062* (2020). https://doi.org/10.48550/arXiv.2007.14062.

[81] Y. Zhan et al. “Griffon v2: Advancing Multimodal Perception with High-Resolution Scaling and Visual-Language Co-Referring”. In: *arXiv preprint arXiv:2403.09333* (2024).

[82] Linfeng Zhang et al. “Be Your Own Teacher: Improve the Performance of Convolutional Neural Networks via Self Distillation”. In: *arXiv preprint arXiv:1905.08094* (2019). https://doi.org/10.48550/arXiv.1905.08094.