Learning Structured Sparsity for Efficient Nanopore DNA Basecalling Using Delayed Masking

Frensel, Mees; Al-Ars, Zaid; Peter Hofstee, H.

DOI
10.1145/3698587.3701357

Publication date
2024

Document Version
Final published version

Published in
BCB '24

Learning Structured Sparsity for Efficient Nanopore DNA Basecalling Using Delayed Masking

Mees Frensel
meesfrensel@gmail.com
Delft University of Technology
Delft, The Netherlands

Zaid Al-Ars
Z.Al-Ars@tudelft.nl
Delft University of Technology
Delft, The Netherlands

H Peter Hofstee
hofstee@us.ibm.com
IBM/TU Delft
Austin, Texas, USA

Abstract

High accuracy nanopore basecalling uses large deep neural networks, requiring powerful GPUs, which is undesirable for sequencing experiments outside the lab. Research has shown that this can be circumvented by using smaller models to increase efficiency as well as basecalling speed. However, this comes at the cost of reduced accuracy, going against the trend of increasingly more complex models to extract the highest possible accuracy out of the source data. We propose learning structured sparsity during model training to find an improved trade-off between accuracy and model size, and thus basecalling speed. Our work introduces an improved pruning method with a delayed masking scheduler and removes redundant masks, saving compute, and is optimized for the basecaller training process. We find that the model size can be reduced by up to 21× with a reduction in match rate of 0.1% to 1.3% compared to Bonito-HAC, using a standardized benchmarking method. Our results indicate that the size of basecalling models can be reduced drastically without affecting accuracy, as long as researchers use appropriate training methods. Furthermore, our work helps democratize nanopore DNA sequencing, broadening the reach and impact of this technology.

The code with the masking mechanism to reproduce our results is available at https://github.com/meesfrensel/efficient-basecallers.

CCS Concepts

• **Computing methodologies** → **Neural networks**; **Regularization**; • **Applied computing** → *Computational genomics.*

Keywords

Recurrent neural networks, Deep neural networks, Learning sparse models, Model compression, Pruning, Nanopore sequencing, Basecalling, Genomics

ACM Reference Format:
Mees Frensel, Zaid Al-Ars, and H Peter Hofstee. 2024. Learning Structured Sparsity for Efficient Nanopore DNA Basecalling Using Delayed Masking. In *15th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics (BCB '24), November 22–25, 2024, Shenzhen, China.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3698587.3701357

BCB '24, November 22–25, 2024, Shenzhen, China

ACM ISBN 979-8-4007-1302-6/24/11
https://doi.org/10.1145/3698587.3701357

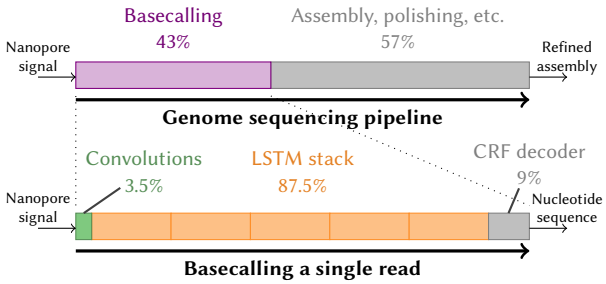


Figure 1: Top: basecalling takes 43% of the time in a nanopore genome sequencing pipeline. Bottom: during basecalling, almost 90% of the computing time is spent computing the LSTM layers' outputs.

1 Introduction

Nanopore sequencers measure electric current as DNA molecules pass through nanopores in a membrane, with these measurements being converted into DNA sequences by basecalling software. The current state-of-the-art basecaller, Oxford Nanopore Technologies' Dorado, reaches 99.5% raw read accuracy and higher when using the newest flowcells and most accurate model [4, 31]. Sequencers like the MinION are highly portable, enabling off-grid experiments in the field or remote locations. Examples of this are wastewater analysis for pathogen detection [8, 44] and off-grid/remote species analysis [11, 20], as well as relatively low-cost cancer detection and research [3, 30]. However, achieving high-quality reads, essential for downstream applications, requires a large amount of compute to perform basecalling.

Dorado and other basecallers rely on deep neural networks (DNNs) for their high accuracy. Over the years, these networks have become larger and more complex, requiring large, powerful and expensive GPUs to keep up with sequencing output. Figure 1 illustrates that about 43% of the time in the nanopore sequencing pipeline is spent on basecalling when using human data [25], with nearly 90% of this time dedicated to computing the Long Short-Term Memory (LSTM) layers, as shown by profiling Dorado. When using viral DNA data in a Read Until assembly, the basecalling step can take more than 95% of the time [9]. This hampers field deployment because, while mobile sequencing is very much possible, analyzing the data is not.

The size of many DNNs is prohibitive to deployment on resource-constrained devices due to their high memory requirements and the substantial number of operations needed to process inputs. Furthermore, the energy required to process DNNs is typically well above the limits of mobile devices [14]. Neural networks are often

over-parameterized, resulting in significant redundancy in DNNs [7]. Research indicates that it is often possible to prune at least half of the parameters without affecting accuracy, and with an effective retraining strategy, up to 90% of parameters can be removed without significantly impacting accuracy [7, 14]. Recent work [38] explored the possibilities of pruning basecallers specifically, but the authors conclude that this approach does not lead to significant savings in compute without sacrificing accuracy. However, their approach is a single-shot pruning and fine-tuning method, which is known to be suboptimal [15].

In this paper, we propose a pruning approach that is highly effective in reducing the size and computational complexity of DNNs and apply it to basecalling models. Results show that the required number of operations can be reduced by 21× and throughput is increased by 2.4×, while maintaining accuracy within 1.3% of the baseline model, Bonito-HAC. Pruning neurons decreases the number of calculations needed and the storage size on both disk and in memory, reducing the size and energy requirements of the basecalling pipeline.

Our contributions are threefold: we improve the speed and robustness of the original neuron selection method [41], firstly by removing the input mask and simplifying the objective function, and secondly by introducing a delayed masking penalty scheduler. Third, our work is, to the best of our knowledge, the first application of neuron selection on a real-world neural network, as well as the first successful pruning approach for basecalling neural networks.

The paper is organized as follows. We provide the necessary background on nanopore basecalling and related work on efficient basecalling and pruning neural networks in Section 2. In Section 3, the limitations of unstructured pruning and concept of learning structured sparsity are introduced, and we provide an in-depth rundown of the neuron selection concept and its theoretical benefits. Section 4 presents benchmark results as well as a basecalling speed comparison, followed by an ablation study. Finally, Sections 5 and 6 present the discussion and conclusions, respectively.

2 Background

For readers unfamiliar with the nanopore sequencing workflow, a brief background on its functionality and basecaller models is given in Section 2.1. Related work on efficient basecallers and structured RNN pruning is presented in Sections 2.2 and 2.3.

2.1 Nanopore Basecalling

Basecalling is the process of translating raw electrical signals generated by (nanopore) sequencers into nucleotide sequences of bases (A, C, G, T/U). During sequencing, DNA or RNA molecules pass through a nanopore, causing disruptions in an ionic current that are characteristic of the sequence of bases. The changes in current are recorded as raw signal data. Basecalling software analyzes these signals to determine the corresponding DNA or RNA sequence. This process is crucial for converting the raw data into meaningful genetic information, which can then be used for various applications in genomics, such as genome assembly, variant detection, and gene expression analysis. The accuracy and efficiency of basecalling significantly impact the overall quality and usability of nanopore sequencing data.

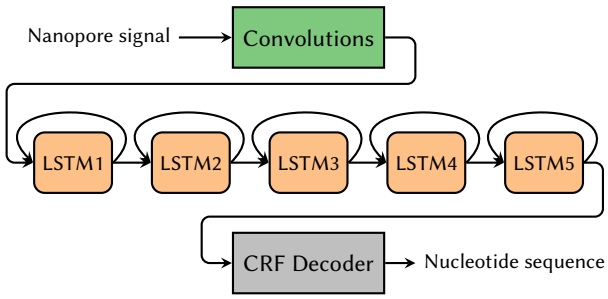


Figure 2: Bonito model architecture

Bonito, developed by Oxford Nanopore Technologies (ONT), is a research basecaller that utilizes deep learning techniques to achieve high accuracy in converting nanopore signals into nucleotide sequences [37]. Bonito's model architecture, visualized in Figure 2, primarily employs convolutional layers combined with LSTM layers to handle the sequential nature of the signal data. The convolutional layers are responsible for extracting features from the raw signal, capturing local patterns, and reducing noise. These features are then passed to LSTM layers, which are adept at handling sequential data and maintaining context over long sequences. The final output layer of the model decodes the processed signal into base sequences using a conditional random field (CRF) decoder, which aligns the neural network outputs with the original sequence. This hybrid architecture allows Bonito to effectively model the complex relationships in nanopore signals, resulting in accurate basecalling even with noisy inputs.

While Bonito is used as research software for basecaller (model) development, Dorado, ONT's production basecaller, represents the current state-of-the-art and is optimized for high throughput basecalling [4]. Building on the advancements of previous models like Bonito, Dorado combines the most accurate models with a high performance implementation of the neural network. It is optimized specifically for GPUs like NVIDIA's V100 and A100, making use of the newest architectural advances. Dorado uses 16-bit floating point numbers almost everywhere, and the LSTM layers are quantized to 8-bit integers to allow the highest throughput. Users can choose from three different models: Fast, HAC (for high accuracy), and SUP (for super accuracy), in order of increasing accuracy and compute requirements.

2.2 Efficient Basecallers

Most research on nanopore basecallers, both by ONT and independent researchers, has focused on using alternative neural network types and architectures, with the goal of creating a basecaller that is more accurate than ONT's own basecaller, i.e. Bonito or one of its predecessors [45, 27, 21, 18]. A small number of works have nonetheless attempted to develop efficient basecallers that can run on a battery-powered laptop for example, while keeping the accuracy at an acceptable level when compared to the state-of-the-art. Recent research explored the possibilities of pruning basecallers [38]. However, they only perform single-shot pruning and training,

and conclude that this approach does not lead to significant savings in compute without sacrificing accuracy.

DeepNano-blitz is one of the first basecallers focused on low-power hardware, without a GPU [5]. To get the highest throughput possible, DeepNano-blitz uses a small network with four GRU (gated recurrent unit) layers and a CTC decoder, which were also used by Bonito at that point, doing the bulk of the work. By the same authors, DeepNano-coral [33] is focused on power efficiency through the use of the Coral Edge Tensor Processing Unit. It achieves real-time basecalling of a single MinION device using just 10 W of power by shrinking and re-engineering the convolutional layers of Bonito and replacing the recurrent layers with a different type of convolution.

Grzesik and Mrozek take a different approach, opting to use existing ONT basecallers on the Jetson Xavier NX from NVIDIA [12]. With an onboard GPU, it is one of the few single board computers capable enough to run basecallers like Guppy. Impressively, the Jetson Xavier NX is able to basecall 3.8M samples/s in its lowest power configuration, or about two MinION sequencing outputs in real-time, with the fast model. However, HAC throughput peaks at only 480k samples/s, about a third of the MinION's output. Notably, by using the stock Guppy basecaller (now Dorado), accuracy is as good as one can get and users benefit from improvements in basecalling accuracy and speed that ONT provides in the future.

There is also a large body of research that uses dedicated hardware accelerators for genomics algorithms [17, 16, 1]. These algorithms can be effectively accelerated on hardware due to their dataflow nature [2]. However, the accelerated algorithms are classical algorithms commonly based on dynamic programming [35] such as Smith-Waterman or probabilistic analyses [34], which is different from the LSTM-based approach used in Bonito.

2.3 Structured Pruning of RNNs

Pruning (deep) neural networks is the process of removing redundant weights (connections) or neurons from a given network. This section discusses related work on pruning RNNs in general, and is not related to basecalling. Pruning received a lot of interest in the past decade, starting with the research of Han et al. who propose the three step iterative pipeline of training, pruning, then fine-tuning [14]. While their work results in a large reduction in the number of connections, it does not focus on neuron (or structured) pruning, thus preventing effective reduction in computing needs. Furthermore, this and many subsequent works focus on convolutional neural networks (CNNs) for computer vision tasks [22], whose concepts, e.g. channel pruning, cannot be directly applied to RNNs which operate on sequences.

While unstructured pruning methods that simply prune individual weights can be applied to LSTMs and other RNNs, these are of limited use and a shift towards structured pruning is needed. Some works focus on mathematically reducing the size of a network by monitoring neuron activity [36] or using new criteria to identify the optimal pruning rate [10]. Wang et al., focusing on an FPGA implementation, prune the columns of weight matrices with the smallest L_1 norm [39]. Also on FPGA, a compression technique with block-circulant is used instead of sparse matrices [40].

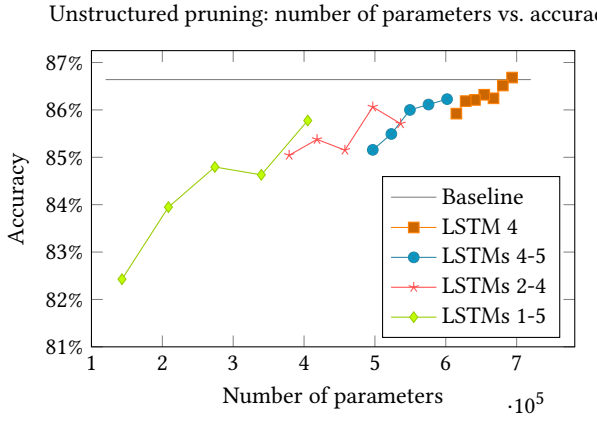


Figure 3: With a simple strategy of pruning some percentage of model weights and fine-tuning for two epochs, a significant reduction in model size can be achieved. However, this leads to unstructured sparsity (zeros in random places) which cannot be exploited for practical speedup.

Wen et al. present intrinsically sparse structures (ISS) for LSTMs [42], one of the first works proposing to learn sparsity. Later research adapts this method for gated recurrent units (GRUs) [24], which is then improved by introducing the concept of neuron selection to structurally shrink the size of LSTM layers [41]. We further adapt and improve this method in this paper.

3 Learning Structured Sparsity

The fundamental problem with most pruning methods is the lack of structured sparsity. Intuitively, an element-wise operation on a matrix that is 50% sparse, should take 50% less floating point operations (FLOPs) compared to a fully dense matrix. Unfortunately, conventional math and tensor libraries cannot take advantage of a low degree of unstructured sparsity: most require sparsity levels of 99% and above to reduce the required computations [15]. This is unrealistic for deep neural networks.

To get an idea of the obtainable level of sparsity, we follow the L_1 regularization procedure originally described in [14] to prune a pre-trained Bonito-HAC model by zeroing out the weights with the smallest magnitude, until the desired sparsity percentage is reached. Then, the resulting model is fine-tuned for two epochs while keeping the 'pruned' weights set at zero.

The results of this pruning and fine-tuning are shown in Figure 3, with different lines corresponding to pruning different sets of layers. For each configuration, we start at 90% pruning rate closest to the origin, and decrement this with 10% for each data point further away from the origin. While we do not fine-tune the model over the whole dataset for the same number of epochs, as described in [14], the pattern is clear: pruning is an effective way of decreasing the number of parameters with a minimal decrease in accuracy, i.e. about 1.5% when pruning the middle three layers by 90%. As mentioned before though, unstructured sparsity is not suitable for speedup. This section will therefore focus on learning structured

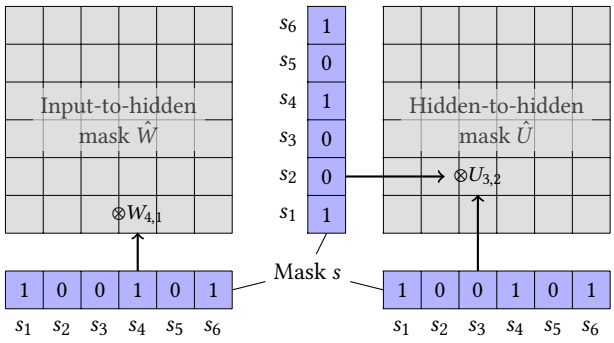


Figure 4: Illustration of the masking mechanism

sparsity simultaneously with training to effectively speed up inference.

3.1 Neuron Selection

The principal optimization method used in this paper is structured pruning of LSTMs through neuron selection, proposed originally by Wen et al., which regularizes a neural network by adding a mask over the neurons and penalizing nonzero mask entries [41]. By making this penalty part of the loss function, the network can learn the weights for accurate basecalling as well as which neurons are required to maximize the accuracy, at the same time.

More specifically, by introducing a set of binary random variables, which can be interpreted as switches for individual neurons, it is possible to structurally prune the LSTMs through *neuron selection*. Let $W = \{W_i, W_f, W_o, W_c\}$ and $U = \{U_i, U_f, U_o, U_c\}$ be the input-to-hidden and hidden-to-hidden weight matrices, respectively. Furthermore, let mask $s = \{s_i\}$, which controls the presence of the hidden neuron i, where $s_i \in \{0, 1\}$ and $|s|$ is the number of hidden neurons, or the 'hidden size'. While training, the weight matrices are masked by 'turning off' rows and columns when the input or output neurons are masked with s. This gating mechanism is visualized in Figure 4. For convenience, the original matrices W and U are re-parameterized to $\hat{W} = W \odot \mathbf{1}s^\top$ and $\hat{U} = U \odot ss^\top$ where $\mathbf{1}$ is the all-ones column vector and $\odot$ denotes the Hadamard or element-wise matrix product.

Conceptually, the binary random variable s is sampled from a Bernoulli distribution with $s_i \sim \text{Bern}(\pi_{s_i})$ where π_{s_i} denotes the probability of random variable s_i taking value 1. However, it is impossible to simply minimize its L_0 norm by gradient optimization, since the Bernoulli distribution is not differentiable. By remodeling to a hard concrete distribution [26], a modification of the concrete distribution [28], we obtain the following procedure to obtain mask variable s:

$$\begin{aligned} u &\sim \text{Uniform}_{[0,1]}, \\ \hat{\tau} &= \sigma((\log u - \log(1-u) + \log\alpha)/\beta), \\ \tau &= \hat{\tau}(\zeta - \gamma) + \gamma, \\ s &= \min(1, \max(0, \tau)), \end{aligned} \tag{1}$$

where $\hat{\tau}$ is a sample from the concrete distribution with location and temperature parameters $\log\alpha$ and β, respectively, which is then stretched to the (ζ, γ) interval and clamped in a hard-sigmoid fashion. Note that $\log\alpha$ is directly optimized, not α. This results in a function that is differentiable and can therefore be implemented in and optimized by machine learning frameworks with autograd like PyTorch and TensorFlow. The probability of s being non-zero, which is used in the loss function, is then computed by the cumulative density function Φ:

$$\Phi_s(s \neq 0|\phi_s) = \sigma(\log\alpha - \beta\log\frac{-\gamma}{\zeta}), \tag{2}$$

where $\phi = \{\alpha, \beta, \gamma, \zeta\}$ denotes the parameters of the hard concrete distribution.

3.2 Optimizing Training Time

The original method from [41] imposes an additional mask z over the input neurons of each LSTM layer. However, in our experiments we find that none or only a couple of input neurons get pruned in practice when the penalty is small. This can be attributed to the fact that there are 4096 possible 5-mers that could be in the nanopore[1]. Pruning away the input neurons removes too much state information from the model, which has a large impact on the model accuracy. This effect is observed in the original paper as well, but the authors do not suggest any remedy except for lowering the penalty value for the input neurons. However, this does not lead to better model compression, but instead results in keeping all input neurons for our model.

Since the neuron selection mechanism is not implemented in the standard LSTM implementations in PyTorch, we resort to a handwritten implementation in Python, with each tensor multiplication or other operation explicitly written out. This results in highly inefficient training due to overhead, data movement, and lack of parallelism. Therefore, by removing the input neuron mask and its random sampling and preprocessing, training is more efficient. This adjustment decreases the training time by 10–20% while maintaining the effectiveness of the neuron selection mechanism.

Without the extra mask z, the loss function can be simplified a bit. Given that $||x||$ is the input size, which replaces the expectation of the input masks with 'active' for each neuron, the loss and objective functions are formulated as follows:

$$\begin{aligned} \mathcal{L}(W, U, \phi) = \mathbb{E}_{\Phi(s|\phi)}[\mathbb{E}_D(W, U, s)] \\ + \lambda \sum_i (||x|| \cdot \Phi_{s_i}(s_i \neq 0|\phi_{s_i})) \\ + \lambda \sum_{i,j}^{i \neq j} (\Phi_{s_i}(s_i \neq 0|\phi_{s_i})\Phi_{s_j}(s_j \neq 0|\phi_{s_j})) \\ + \lambda \sum_i \Phi_{s_i}(s_i \neq 0|\phi_{s_i}) \end{aligned} \tag{3}$$

$$(W^*, U^*, \phi^*) = \arg\min_{W,U,\phi} \mathcal{L}(W, U, \Phi) \tag{4}$$

3.3 Delayed Masking

Another new contribution is the addition of a warm-up scheduler to prevent the model from collapsing at the start of training. We find that at high penalty rates, e.g. $> 3 \cdot 10^{-7}$, the autograd optimizer is eagerly pruning the model to reduce the value of the loss function,

[1]There are always 5 bases in a nanopore at the same time, each base can be one of 4, so there are $4^5 = 4096$ total possible states.

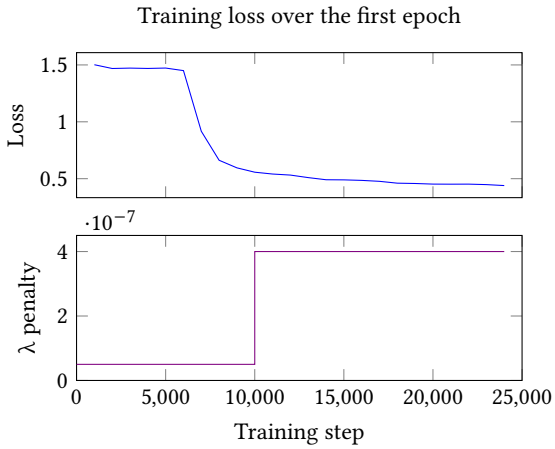


Figure 5: Top: training loss during the first epoch. After 6000 steps, the loss value sharply declines in a few thousand steps. When λ is too large, the model size decreases so much within this time, that it is not able to find patterns in the data at all. Bottom: the proposed delayed masking schedule ensures that a baseline accuracy has been established before starting to mask more and more of the model.

but this happens so quickly that the model does not have enough time to learn patterns in the data at all. The result is that the model is pruned to a hidden size of (nearly) 0, with the accuracy also sitting at 0%. This effect is a result of the model's performance during training: as can be seen in Figure 5 (top), the loss value stays consistently high at 1.4–1.5 until dropping off after 6000 training steps. It is at this state that the model grasps the structure of the nanopore squiggle, as the validation accuracy jumps from 0 to about 70% in subsequent steps.

We propose a penalty scheduler to prevent this issue from happening at larger λ penalty values. By delaying the masking to a later moment, we give the network time to train and learn before more aggressively turning off neurons. For example, using a step function from a modest $0.5 \cdot 10^{-7}$ at the start, to λ's intended value at 10000 training steps (see Figure 5) keeps the model at about 90% of its original size for the first part of training. After this time, the full penalty value is used and the model is masked much more aggressively. While we use a step function with a fixed step moment in this paper, more elaborate schemes using linear or sigmoid functions and a dynamic step moment can be tested in the future. The penalty warm-up is a novel contribution that we believe can benefit any application that is learning structured sparsity in a neural network.

3.4 Efficient Inference

During training, the weight mask is recalculated with every forward step, with each neuron being (de-)activated based on a random sample from its entry in s. As the neurons are 'selected' over the course of the training steps, the probability of a neuron being active is generally not truly zero, to allow correction based on the output accuracy. Consequently, no practical speedup is obtained over the complete starting model. When training is complete, the model can be structurally pruned by removing rows and columns that are all zeros from $\hat{W}$ and $\hat{U}$, as well as the bias vector. Then, these weights can be used for a plain LSTM layer with a reduced hidden size of $||s||_0$.

To motivate and quantify the benefit of smaller LSTM layers, consider that the computational intensity is dominated by the hidden size: the approximate number of FLOPs per batched LSTM step is $N \times D \times (I + D) \times 8$ FLOPs[2] where N is the batch size and I and D are the input and hidden size, respectively. An LSTM with an input and hidden size of 384, like Bonito's, therefore requires 2.4M FLOPs for a single timestep with batch size 1. If the hidden size is reduced to around 120, as shown in Table 1 with $\lambda = 6 \cdot 10^{-7}$ for example, the inference step requires just 484k or 0.2× the FLOPs. This reduction of the hidden size D for the LSTM's $O(D^2)$ computational complexity results in a substantially more efficient model.

3.5 Evaluation of Alternative Pruning Methods

In the process of enhancing the efficiency of nanopore basecalling models, we evaluated several alternative pruning methods proposed in the literature. Despite their promise, the methods discussed in this section did not perform as well in our specific application as reported in the original papers. For example, the Selfish-RNN sparse-to-sparse training method proposed in [23] did not perform nearly as well as the standard dense training method. Using the implementation provided by the authors, the final match rate was 10–13% lower than the baseline. This could be due to the model being small from the start, without being trained on the data at full size first.

The C-LSTM method involves compressing LSTM networks using block-circulant matrices, which significantly reduce the number of parameters and computational complexity [40]. The approach uses block-circulant instead of sparse matrices to compress weight matrices and reduces the storage requirement from $O(k^2)$ to $O(k)$ and FFTs to reduces the computational complexity from $O(k^2)$ to $O(k \log k)$. Although this technique demonstrated impressive results on FPGA implementations, it did not translate well to our GPU-based environment. The block-circulant structure introduced challenges in adapting the model to our specific basecalling tasks, and we were not able to successfully train a model.

A linear surrogate of the LSTM cell using linear recurrence [29] is a promising replacement for the standard LSTM cells in the LSTM layers. As a precursor to linear state-space models like S4 [13], it changes the iterative nature of recurrent models to a parallel scan algorithm. This parallel linear recurrence can be efficiently computed by parallel solvers like GPUs and thus appears to be a good alternative. For our basecalling application, however, experiments show that the accuracy did not get close to the baseline accuracy. Similar to Selfish-RNN, accuracy was about 9–11% below baseline.

Despite the initial promise of these methods, our experiments highlight the importance of domain-specific optimization. The unique challenges posed by nanopore basecalling, such as electrical noise and highly variable sequencing speed, necessitate pruning techniques that can maintain high accuracy while efficiently

[2] $\times 4 \cdot 2$ because the LSTM has 4 gates that each operate on the input, and a multiplication and addition have to be performed for each element.

reducing computational requirements. This suggests that not all sparsification methods are created equal, and that one can not directly apply a single best method to any application. As such, our focus shifted towards structured sparsity and neuron selection, which proved to be most effective for our purposes.

4 Experiments and Results

4.1 Experimental Setup

Hyperparameters. The model is trained with nearly the same hyperparameter settings as in [32], using the Adam optimizer with initial learning rate = 1.5e-3, $\beta_1 = 0.9$, $\beta_2 = 0.999$, weight decay = 0. The learning rate is linearly increased for the first 5000 training steps from 0 to the initial optimizer rate as a warm-up, then decreased using a cosine function to a minimum of 1e-5 by the final step. The batch size is doubled from 64 to 128, and the initial learning rate is increased accordingly from 1e-3 to 1.5e-3. Gradients are clipped between -2 and 2 to improve model stability. All standard LSTM parameters are initialized from $\mathcal{N}(0, 0.05)$.

For the neuron selection mechanism, we initially choose $\lambda = 2 \cdot 10^{-7}$ with smoothing parameter set $\phi = \{\beta = 2/3, \gamma = -0.1, \zeta = 1.1\}$. α is a learnable parameter directly trained as $\log \alpha$, initialized from $\mathcal{N}(1.5, 0.1)$ which corresponds to a 90% probability that a neuron is active on average. Since the model contains no dropout layers, the dropout keep ratio does not have to be increased.

Computational Environment. The experiments are run in an HPC environment [6] on nodes with Xeon E5-6448Y 32C CPUs @ 2.10 GHz and an NVIDIA A100 80GB PCIe GPU. In this environment, a full training run of 5 epochs takes 23.5 hours to complete. We use Python 3.9.8 with PyTorch 1.12.1. To deterministically find the throughput numbers in Figure 8, we disable the cuDNN backend in PyTorch. Other dependencies and versions are listed in the code repository on GitHub.

4.2 Datasets

The training, evaluation, and test datasets are the same benchmark datasets from [32], in which the authors propose a standardized benchmark for nanopore basecaller evaluation. We use the cross-species task for training and evaluation of all models. Therefore, our results are not directly comparable to [32], focusing solely on the human task.

The dataset consists of 3 human datasets from [19], 1 Lambda phage dataset sequenced in-house [32], and 60 bacterial datasets encompassing 26 different bacterial species originally published in [43]. All data is sequenced using R9.4 or R9.4.1 pore chemistry. Notably, performance improvements in newer pore chemistries, such as the current R10.1, are expected to similarly enhance our results, reflecting the ongoing advancements in nanopore sequencing technology. However, the larger number of bases present in R10.1 pores could increase sensitivity to pruning. We hypothesize that the convolutional layers extract all relevant information from the signal, therefore not negatively impacting pruning potential, but this point warrants further investigation.

4.3 Basecalling Accuracy

For evaluating our work, we reference the benchmark established in [32], which addresses the need for standardized benchmarking in nanopore sequencing, where basecalling accuracy is crucial and often improved through new neural network architectures. Due to varying evaluation metrics and datasets across different publications, it has been challenging to differentiate between data-driven and model-driven improvements. By using this comprehensive benchmark, we ensure fair and consistent comparisons of our basecaller with state-of-the-art models, thereby validating our performance claims.

The primary results are shown in Figure 6, highlighting the performance of Bonito fast, HAC, and our method with different pruning rates denoted by λ_x for $\lambda = x \cdot 10^{-7}$. The AUC (Figure 6a), with reads sorted by decreasing PhredQ score, shows that all models have a clear correlation between read quality and average match rate. This conclusion is backed up by the high amount of separation in the PhredQ quality scores (in Figure 6b), allowing downstream applications to use this as a measure of certainty that a base is correct. Bonito-HAC performs best with an AUC of 0.848 and our pruned model with $\lambda = 2 \cdot 10^{-7}$ following closely behind with an AUC of 0.847. The match rates differ by just 0.13%, while the number of FLOPs required is decreased by 2.8×.

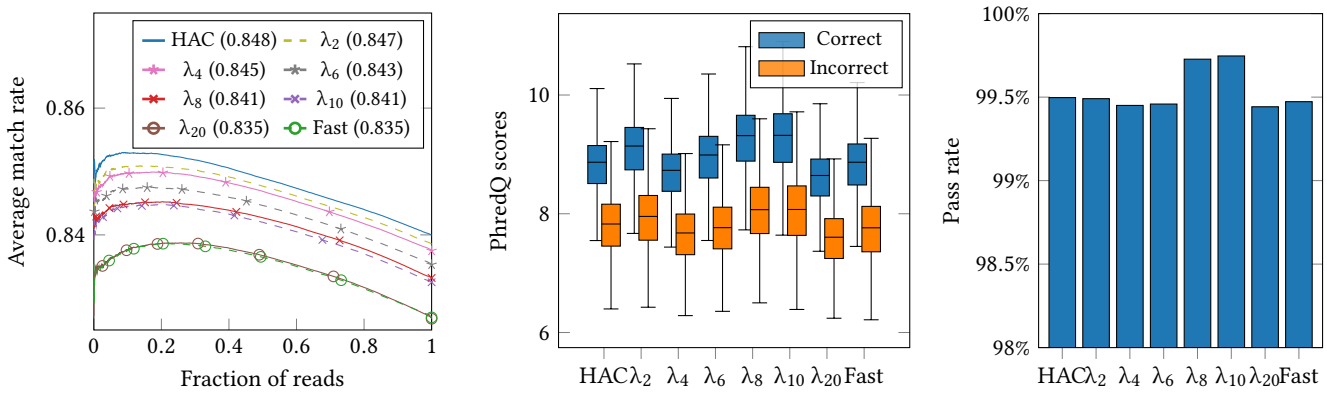
Pass rates are an important measure against 'cheating': basecalling models could achieve higher match rates by skipping reads that are harder to basecall correctly [32]. However, as can be seen in Figure 6c, the models with higher pruning rates do not sacrifice the pass rate for accuracy. This is important but not straightforward, as many experiments with prune/fine-tune methods resulted primarily in very low pass rates.

Overall, the trend is that at higher pruning rates all metrics tend to suffer somewhat equally: we do not observe that pruning affects some metric more than others. This is both positive and negative, as on one hand smaller models could be useful for users who do not care about that one metric, allowing them to save on compute. On the other hand, being able to play with the penalty value to get just the precision you need at the least amount of compute is useful for users who do not have a data center at their disposal.

4.4 Basecalling Throughput

Smaller models are theoretically faster at basecalling than large models. As explained in Section 3.4, the number of FLOPs for a single inference step is mostly dependent on the hidden size, and the input size has some influence as well. Our method's focus is on reducing the layer's hidden size, which has an $O(n^2)$ effect on the computational complexity, though by propagating the hidden mask to the next LSTM layer, this holds for input size as well. By reducing the hidden size with 1/3, the number of FLOPs decreases by 2×, while the accuracy is barely affected. Furthermore, by allowing the match rate to drop by 0.4%, it is possible to reduce the LSTM stack's theoretical FLOPs by 7.2×.

The possible gains from reducing the hidden size are illustrated with the mask $\hat{U}$, as shown in Figure 7. Since we induce structured sparsity, we can 'physically' remove neurons and the corresponding rows/columns from the model. During training, which neurons are important, is automatically decided by backpropagation, and during



(a) AUC of match rate sorted by average read PhredQ score

(b) Separation of PhredQ scores of correct and incorrectly called reads

(c) Overall pass rates

Figure 6: Benchmark results showing Bonito Fast and HAC models, and our work for different λ penalties

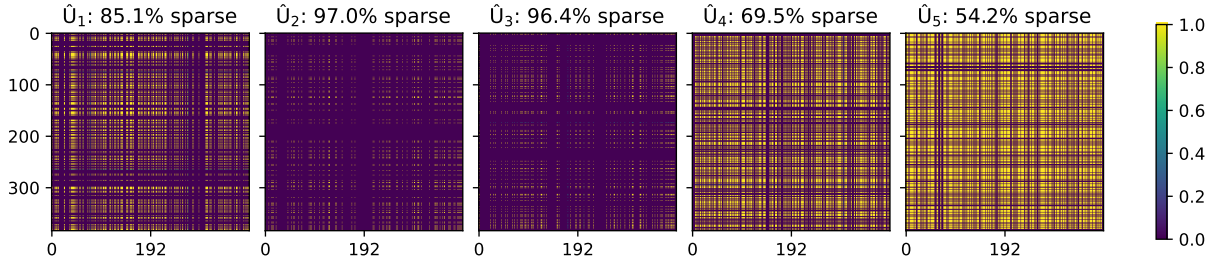


Figure 7: Illustration of the masks of the hidden-to-hidden weight matrix $\hat{U}$ for each LSTM layer in the λ_4 model

inference, we benefit by reloading the LSTM cells with smaller hidden sizes as well as the input sizes equal to the hidden size (output size) of the previous layer. This results in faster inference during basecalling, improving energy usage and throughput.

The resulting throughput speedup achieved with structured sparsity depends on many factors besides the number of FLOPs, which includes batch size, sequence length among others. Figure 8 shows the average throughput of the whole neural network (see the architecture in Figure 2) for different models, with constant batch size and sequence length. The curve shows the direct relationship between the theoretical gains and real throughput increase: at λ_2, throughput is already 1.5× higher than the baseline. Notably, the fastest model λ_{20} is 2.4× faster compared to the HAC baseline. By reducing the model size, memory requirements are reduced accordingly which opens up room for increasing the batch size. As a result, more data is processed in less time, increasing overall throughput even more.

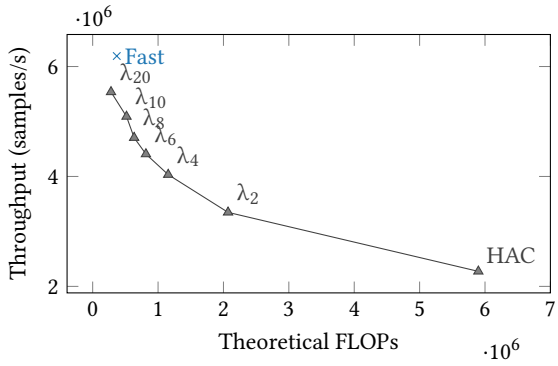


Figure 8: Theoretical FLOPs for the LSTM stack compared to the basecalling throughput of the whole neural network

4.5 Ablation Study

To thoroughly evaluate the impact of structured pruning on our basecalling model, we conduct an ablation study focusing on varying λ values, which control the neuron penalty during training. Another way to think about this, is how sensitive the model is to varying λ penalty values. This study is essential to understand the trade-offs between model size, throughput, and read accuracy. Our findings are illustrated in Figure 9 summarized in Table 1.

For $\lambda = 2 \cdot 10^{-7}$, we observed no significant change in basecalling accuracy compared to the baseline model, while achieving a 2.8× reduction in FLOPs. This indicates that a moderate sparsity penalty can effectively prune the network without compromising performance. Increasing the penalty to $4 \cdot 10^{-7}$, the match rate decreases by a further 0.2% while compressing the model to a fifth

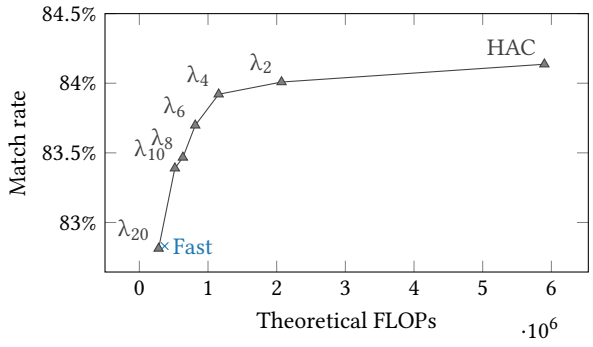


Figure 9: Results of the ablation study comparing FLOPs and match rate

λ penalty	Match	Hidden sizes	FLOPs (Improvement)
Baseline	84.14%	384, 384, 384, 384, 384	$5.90M$
$2 \cdot 10^{-7}$	84.01%	213, 126, 119, 291, 324	$2.07M$ (2.8×)
$4 \cdot 10^{-7}$	83.92%	149, 68, 74, 212, 261	$1.16M$ (5.1×)
$6 \cdot 10^{-7}$	83.70%	111, 48, 55, 179, 229	$0.81M$ (7.2×)
$8 \cdot 10^{-7}$	83.47%	96, 39, 48, 155, 201	$0.63M$ (9.3×)
$10 \cdot 10^{-7}$	83.39%	82, 36, 38, 133, 190	$0.52M$ (11.4×)
$20 \cdot 10^{-7}$	82.81%	55, 22, 21, 88, 145	$0.28M$ (21.0×)
Fast	82.83%	Input & hidden size: 96	$0.36M$ (16.0×)

Table 1: Ablation study of the penalty parameter of the LSTM layers

of its original size. This level of pruning strikes a good balance between maintaining accuracy and enhancing efficiency.

At higher penalty values, such as $\lambda = 8 \cdot 10^{-7}$, the model experiences a more noticeable drop in accuracy (around 0.7%), but the model FLOPs are drastically reduced, by more than 9× in the LSTM layers. This suggests that while aggressive pruning can significantly decrease computational requirements, it may also impact the model's performance. These results highlight the importance of selecting an appropriate sparsity penalty to balance the trade-offs between model efficiency and basecalling accuracy, which is similar to selecting a basecalling model from Bonito/Dorado's Fast, HAC or SUP models.

Overall, the ablation study underscores the potential of structured pruning to optimize nanopore basecalling models, making high-quality sequencing more accessible for field applications. The results show the great performance benefit of learning which neurons should be active at the same time as learning the connection weights between them. Between Bonito's Fast and HAC models, we consider $\lambda = 6 \cdot 10^{-7}$ the best trade-off between accuracy and speed, measuring just 0.44% below HAC's match rate while achieving a 7.2× reduction in FLOP count, ending nicely in between Bonito's models.

5 Discussion

In this paper, we propose learning structured sparsity to make basecalling neural networks substantially more efficient while maintaining high accuracy. This makes nanopore sequencing faster, cheaper, and more accessible by using less compute resources. Moreover, our results show the extent to which the state-of-the-art models are over-parameterized when considering their accuracy. As previous works on efficient basecalling rightly point out, the ever-increasing accuracy, and as a result, size of these models does not outweigh the implications of resulting compute requirements for everyone. Moreover, available power and compute can be a decisive variable in selecting a basecalling model.

Aside from pruning, the other most common approach to model compression, be it for storage or compute savings, is quantization. By representing numbers with fewer bits or even as integers, one can save space and utilize specialized hardware to do these lower precision computations. Mixed precision math is widely used in Dorado and the Rubicon paper performed an in-depth analysis [38] that can be readily integrated with our work, as quantization is orthogonal to pruning. Furthermore, because the input size is fixed, the convolutional layers cannot be adjusted, limiting the overall reduction in parameters and FLOPs. We therefore recommend further research into whole-model structural pruning, in combination with quantization.

6 Conclusion

This paper presents a new pruning approach to enhance the efficiency of DNNs through the use of structured sparsity in the LSTM layers, and applies this to nanopore DNA basecalling models. By leveraging neuron selection techniques to prune redundant parameters, we can significantly reduce the computational and memory requirements of the basecalling models without compromising accuracy. Notably, our smallest model is 21× more efficient with a reduction in match rate of 1.3% compared to Bonito-HAC. This advancement is particularly beneficial for field deployments of portable sequencing devices like the MinION, where high-quality basecalling needs to be performed on resource-constrained hardware.

Our experimental results demonstrate that structured pruning can achieve substantial improvements in model size and basecalling throughput of up to 21× and 2.4×, respectively, enabling efficient basecalling on devices with limited compute resources. This approach facilitates DNA analysis in various field applications, by allowing users to select an appropriate trade-off between accuracy and basecalling time for their demands and available resources. There is ample room to maneuver this spectrum, with a throughput increase between 1.5 – 2.4× and an accuracy drop of 0.1% – 1.3% compared to the baseline Bonito-HAC model. Future work will explore further structured optimization techniques and better delayed masking schedulers to further improve the performance and efficiency of basecalling models.

Acknowledgments

This research was performed with the support of the Eureka Xecs project TASTI (grant no. 2022005).

References

[1] Nauman Ahmed et al. 2019. GASAL2: a GPU accelerated sequence alignment library for high-throughput NGS data. *BMC Bioinformatics*, 20, 1, (Oct. 25, 2019), 520. DOI: 10.1186/s12859-019-3086-9.

[2] Nauman Ahmed et al. 2015. Heterogeneous hardware/software acceleration of the BWA-MEM DNA alignment algorithm. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). (Nov. 2015), 240–246. DOI: 10.1109/ICCAD.2015.7372576.

[3] Timour Baslan et al. 2021. High resolution copy number inference in cancer using short-molecule nanopore sequencing. *Nucleic Acids Research*, 49, 21, (Dec. 2, 2021), e124. DOI: 10.1093/nar/gkab812.

[4] Mark Bicknell. Dorado — the future of basecalling. London Calling 2023, London, (May 19, 2023). Retrieved Apr. 18, 2024 from https://nanoporetech.com/resource-centre/london-calling-2023-dorado-future-basecalling.

[5] Vladimír Boža et al. 2020. DeepNano-blitz: a fast base caller for MinION nanopore sequencers. *Bioinformatics*, 36, 14, (July 30, 2020), 4191–4192. DOI: 10.1093/bioinformatics/btaa297.

[6] [SW] Delft High Performance Computing Centre (DHPC), DelftBlue Supercomputer (Phase 2) 2024. URL: https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2.

[7] Misha Denil et al. 2013. Predicting parameters in deep learning. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2* (NIPS'13). Curran Associates Inc., Red Hook, NY, USA, (Dec. 5, 2013), 2148–2156. Retrieved Mar. 15, 2024 from.

[8] Megan B. Diamond et al. 2022. Wastewater surveillance of pathogens can inform public health responses. *Nature Medicine*, 28, 10, (Oct. 2022), 1992–1995. Number: 10 Publisher: Nature Publishing Group. DOI: 10.1038/s41591-022-01940-x.

[9] Tim Dunn et al. 2021. SquiggleFilter: an accelerator for portable virus detection. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (MICRO '21). Association for Computing Machinery, New York, NY, USA, (Oct. 17, 2021), 535–549. ISBN: 978-1-4503-8557-2. DOI: 10.1145/3466752.3480117.

[10] Nikolaos Gkalelis et al. 2020. Structured pruning of LSTMs via eigenanalysis and geometric median for mobile multimedia and deep learning applications. In *2020 IEEE International Symposium on Multimedia (ISM)*. 2020 IEEE International Symposium on Multimedia (ISM). (Dec. 2020), 122–126. DOI: 10.1109/ISM.2020.00028.

[11] Glen-Oliver F. Gowers et al. 2019. Entirely off-grid and solar-powered DNA sequencing of microbial communities during an ice cap traverse expedition. *Genes*, 10, 11, (Nov. 2019), 902. Number: 11 Publisher: Multidisciplinary Digital Publishing Institute. DOI: 10.3390/genes10110902.

[12] Piotr Grzesik et al. 2021. Metagenomic analysis at the edge with jetson xavier NX. In *Computational Science – ICCS 2021* (Lecture Notes in Computer Science). Maciej Paszynski et al., (Eds.) Springer International Publishing, Cham, 500–511. ISBN: 978-3-030-77970-2. DOI: 10.1007/978-3-030-77970-2_38.

[13] Albert Gu et al. 2021. Efficiently modeling long sequences with structured state spaces. (Oct. 31, 2021). DOI: 10.48550/arXiv.2111.00396.

[14] Song Han et al. 2015. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1* (NIPS'15). MIT Press, Cambridge, MA, USA, (Dec. 7, 2015), 1135–1143. Retrieved Mar. 15, 2024 from.

[15] Torsten Hoefler et al. 2021. Sparsity in deep learning: pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research*, 22, 1, (Jan. 1, 2021), 241:10882–241:11005.

[16] Ernst Joachim Houtgast et al. 2015. An FPGA-based systolic array to accelerate the BWA-MEM genomic mapping algorithm. In *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. 2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS). (July 2015), 221–227. DOI: 10.1109/SAMOS.2015.7363679.

[17] Ernst Joachim Houtgast et al. 2018. Hardware acceleration of BWA-MEM genomic short read mapping for longer read lengths. *Comput. Biol. Chem.*, 75, (Aug. 1, 2018), 54–64, C, (Aug. 1, 2018). DOI: 10.1016/j.compbiolchem.2018.03.024.

[18] Neng Huang et al. 2022. SACall: a neural network basecaller for oxford nanopore sequencing data based on self-attention mechanism. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 19, 1, (Jan. 2022), 614–623. Conference Name: IEEE/ACM Transactions on Computational Biology and Bioinformatics. DOI: 10.1109/TCBB.2020.3039244.

[19] Miten Jain et al. 2018. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature Biotechnology*, 36, 4, (Apr. 2018), 338–345. Publisher: Nature Publishing Group. DOI: 10.1038/nbt.4060.

[20] Ineke Knot. How do i use portable genomics in the field? WILDLABS Tech Tutors, (Aug. 13, 2020). Retrieved Feb. 20, 2024 from https://www.wildlabs.net/event/how-do-i-use-portable-genomics-field.

[21] Hiroki Konishi et al. 2021. Halcyon: an accurate basecaller exploiting an encoder–decoder model with monotonic attention. *Bioinformatics*, 37, 9, (June 9, 2021), 1211–1217. DOI: 10.1093/bioinformatics/btaa953.

[22] Tailin Liang et al. 2021. Pruning and quantization for deep neural network acceleration: a survey. *Neurocomputing*, 461, (Oct. 21, 2021), 370–403. DOI: 10.1016/j.neucom.2021.07.045.

[23] Shiwei Liu et al. 2021. Selfish sparse RNN training. (June 15, 2021). arXiv: 2101.09048[cs]. DOI: 10.48550/arXiv.2101.09048.

[24] Ekaterina Lobacheva et al. 2020. Structured sparsification of gated recurrent neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34, 4, (Apr. 3, 2020), 4989–4996. Number: 04. DOI: 10.1609/aaai.v34i04.5938.

[25] Qian Lou et al. 2020. Helix: algorithm/architecture co-design for accelerating nanopore genome base-calling. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques* (PACT '20). Association for Computing Machinery, New York, NY, USA, (Sept. 30, 2020), 293–304. ISBN: 978-1-4503-8075-1. DOI: 10.1145/3410463.3414626.

[26] Christos Louizos et al. 2018. Learning sparse neural networks through l_0 regularization. (June 22, 2018). arXiv: 1712.01312[cs,stat]. DOI: 10.48550/arXiv.1712.01312.

[27] Xuan Lv et al. 2020. An end-to-end oxford nanopore basecaller using convolution-augmented transformer. In *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. 2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM). (Dec. 2020), 337–342. DOI: 10.1109/BIBM49941.2020.9313290.

[28] Chris J. Maddison et al. 2017. The concrete distribution: a continuous relaxation of discrete random variables. (Mar. 5, 2017). arXiv: 1611.00712[cs,stat]. DOI: 10.48550/arXiv.1611.00712.

[29] Eric Martin et al. 2018. Parallelizing linear recurrent neural nets over sequence length. In International Conference on Learning Representations. https://openreview.net/forum?id=HyUNwulC-.

[30] Josephine B. Oehler et al. 2023. The application of long-read sequencing in clinical settings. *Human Genomics*, 17, 1, (Aug. 8, 2023), 73. DOI: 10.1186/s40246-023-00522-3.

[31] Oxford Nanopore Technologies. 2024. Nanopore sequencing accuracy. Oxford Nanopore Technologies. Retrieved Jan. 23, 2024 from https://nanoporetech.com/platform/accuracy.

[32] Marc Pagès-Gallego et al. 2023. Comprehensive benchmark and architectural analysis of deep learning models for nanopore sequencing basecalling. *Genome Biology*, 24, 1, (Apr. 11, 2023), 71. DOI: 10.1186/s13059-023-02903-2.

[33] Peter Perešíni et al. 2021. Nanopore base calling on the edge. *Bioinformatics*, 37, 24, (Dec. 11, 2021), 4661–4667. DOI: 10.1093/bioinformatics/btab528.

[34] Shanshan Ren et al. 2018. Efficient acceleration of the pair-HMMs forward algorithm for GATK HaplotypeCaller on graphics processing units. *Evolutionary Bioinformatics Online*, 14, 1176934318760543. DOI: 10.1177/1176934318760543.

[35] Shanshan Ren et al. 2019. GPU accelerated sequence alignment with traceback for GATK HaplotypeCaller. *BMC Genomics*, 20, 2, (Apr. 4, 2019), 184. DOI: 10.1186/s12864-019-5468-9.

[36] Nikita Semionov. 2019. *Pruning of Long Short-Term Memory Neural Networks: Passes of Redundant Data Patterns*. Master thesis. Tilburg University. Cognitive science and artificial intelligence, (Dec. 2019). 63 pp. https://arno.uvt.nl/show.cgi?fid=153975.

[37] [SW] Chris Seymour, Bonito: A PyTorch Basecaller for Oxford Nanopore Reads 2019. URL: https://github.com/nanoporetech/bonito.

[38] Gagandeep Singh et al. 2024. RUBICON: a framework for designing efficient deep learning-based genomic basecallers. *Genome Biology*, 25, 1, (Feb. 16, 2024), 49. DOI: 10.1186/s13059-024-03181-2.

[39] Shaorun Wang et al. 2019. Acceleration of LSTM with structured pruning method on FPGA. *IEEE Access*, 7, 62930–62937. Conference Name: IEEE Access. DOI: 10.1109/ACCESS.2019.2917312.

[40] Shuo Wang et al. 2018. C-LSTM: enabling efficient LSTM using structured compression techniques on FPGAs. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (FPGA '18). Association for Computing Machinery, New York, NY, USA, (Feb. 15, 2018), 11–20. ISBN: 978-1-4503-5614-5. DOI: 10.1145/3174243.3174253.

[41] Liangjian Wen et al. 2020. Structured pruning of recurrent neural networks through neuron selection. *Neural Networks*, 123, (Mar. 1, 2020), 134–141. DOI: 10.1016/j.neunet.2019.11.018.

[42] Wei Wen et al. 2018. Learning intrinsic sparse structures within long short-term memory. (Feb. 11, 2018). arXiv: 1709.05027[cs]. DOI: 10.48550/arXiv.1709.05027.

[43] Ryan R. Wick et al. 2019. Performance of neural network basecalling tools for oxford nanopore sequencing. *Genome Biology*, 20, 1, (June 24, 2019), 129. DOI: 10.1186/s13059-019-1727-y.

[44] Karin Yaniv et al. 2023. Wastewater monitoring of SARS-CoV-2 in on-grid, partially and fully off-grid bedouin communities in southern israel. *Frontiers in Water*, 5. DOI: 10.3389/frwa.2023.1136066.

[45] Yao-Zhong Zhang et al. 2020. Nanopore basecalling from a perspective of instance segmentation. *BMC bioinformatics*, 21, (Apr. 23, 2020), 136, Suppl 3, (Apr. 23, 2020). DOI: 10.1186/s12859-020-3459-0.