Lokatt: A hybrid DNA nanopore basecaller with an explicit duration hidden Markov model and a residual LSTM network

Xuechun Xu [1,*], Nayanika Bhalla [2], Patrik Ståhl [2] and Joakim Jaldén [1]

[1] ISE, KTH, Stockholm, 11423, Sweden and
[2] SciLife Lab, Stockholm, Sweden.

* To whom correspondence should be addressed.

Abstract

Motivation: Basecalling long DNA sequences is a crucial step in nanopore-based DNA sequencing protocols. Recently the CTC-RNN model has become the leading basecalling model and has replaced prior hidden Markov models (HMMs) based on pre-segmentation of the ion current measurements. However, CTC-RNN relies entirely on the neural network to make basecalling decisions while prior knowledge of the biological and physical processes is not involved. We believe that there are undiscovered potentials in using this prior knowledge that can benefit the basecalling performances and that HMMs can indeed be used to leverage this information if combined with an RNN in an end-to-end algorithm.
Results: We present the basecaller named **Lokatt**: explicit duration Markov model and residual-LSTM network. It is based on an Explicit Duration HMM (EDHMM), designed to model the nanopore sequencing processes. Trained on a new library created with free-methylation Ecoli genome with MinION R$9.4.1$ chemistry, the Lokatt basecaller achieves basecalling performances with a median single read identity score of 92%, on par with existing state-of-the-art.
Availability: https://github.com/chunxxc/lokatt
Contact: chunx@kth.se

1 Introduction

The concept of nanopore-sequencing was firstly drafted in 1989 as a hand-sketched illustration by David Deamer on a page of a notebook [1]. 30 years later, the technology is now a commercial success, available from Oxford Nanopore Technologies (ONT) [2]. Nanopore sequencing works by threading a single-strand DNA (ssDNA) through a protein-formed pore in a membrane. The sequence of nucleotides along the ssDNA can be indirectly recorded through their effect on an ion current flowing through the pore. However, transforming the current measurements into a sequence of bases, i.e., basecalling, is challenging for several reasons [3]. Firstly, multiple nucleotides along the ssDNA simultaneously affect the noisy current measurements at any given time. Secondly, the ssDNA's translocation speed through the pore is fast and unstable, leading to a random and apriori unknown number of current samples per nucleotide base in the measurements. Further, the short-term average translocation speed and the noise level are also variable across a long sequencing run due to slowly changing experimental conditions. These challenges made early nanopore sequencing unusable for most clinical and research applications. Although these problems have now been mitigated by, e.g., selecting and modifying various protein pores with narrower constriction, using improved DNA ratcheting enzymes to control the polymer's translocation speed, and better basecalling algorithms, the technology is still limited for many applications due to high error rates, amounts of material and costs [4][5].

Early basecalling algorithms worked by first segmenting the current measurements into a sequence of probabilistic events. These events were then treated as observations in a graphical model, usually in the form of a Hidden Markov model (HMM) with latent states representing k-mers, i.e., nucleotide sequences of length k. The choice of k corresponded to the number of bases that had a simultaneous effect on the measurements [1][6]. The representation of observation probabilities went from simple Gaussian distributions parameterized by the mean and variance of the ionic current during an event [7][8] to more elaborate models such as hierarchical Dirichlet processes [9]. With the probabilistic model in place, the final basecalling step could be completed using standard inference algorithms for HMMs, such as the Viterbi and the beam search algorithms. However, the performance of the HMM remained severely limited by the quality of the segmentation step and the choice of features used to model the distribution of the events. Most modern basecallers are now end-to-end and based on deep neural networks (DNN) due to the limitations of the HMM basecallers and following the pioneering work that led to the Chiron basecaller [10]. Specifically, Chiron applied a recurrent neural network (RNN) to process the current measurements, a connectionist temporal classification (CTC) layer from natural language processing (NLP) to replace the event-based HMM for data alignment and trained the resulting network end-to-end. The previously used event features have thus been replaced by neural networks, which obviate the need for explicit feature

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

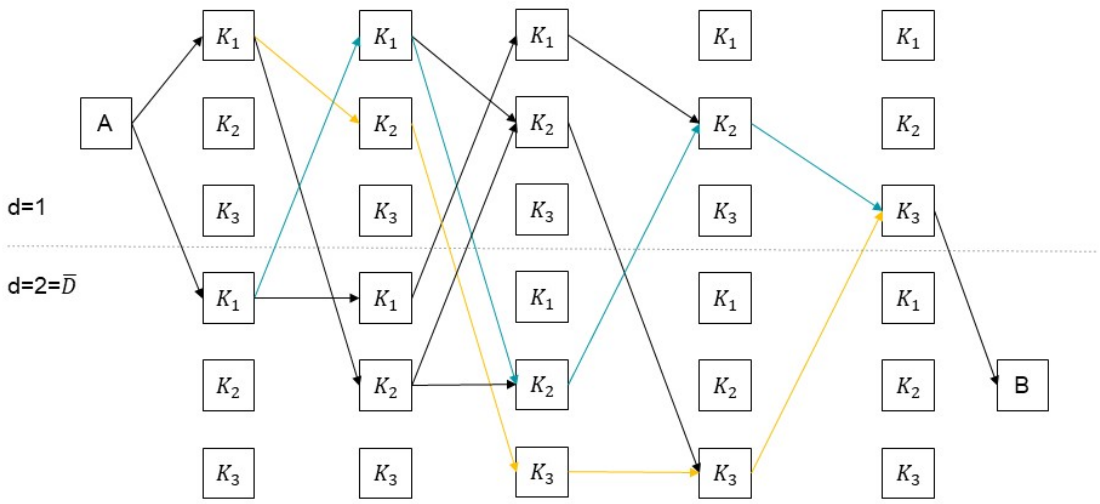

Fig. 1. Illustration of EDHMM structure, with example of $\mathbf{K}_3 = \{K_1, K_2, K_3\}$ and $\mathbf{X}_5 = \{X_1, ..., X_5\}$, hence $M = 3$, $\bar{D} = 2$ and $N = 5$. States in the upper half represent $(K_m, d = 1)$ while states in lower half represent $(K_m, d = 2)$. The blue path goes through five states $(K_1, d = 2), (K_1, d = 1), (K_2, d = 2), (K_2, d = 1), (K_3, d = 1)$, representing the sequence of pairs $(K_1, D_1 = 2), (K_2, D_2 = 2), (K_3, D_3 = 1)$; The yellow path goes through $(K_1, d = 1), (K_2, d = 1), (K_3, d = 2), (K_3, d = 2), (K_3, d = 1)$, representing the sequence of pairs $(K_1, D_1 = 1), (K_2, D_2 = 1), (K_3, D_3 = 3)$.

engineering. The ONT open-source software Bonito [11], which also use a CTC-RNN structure and end-to-end training, performs on par with the current state-of-art commercial software Guppy, which presumably also uses a deep learning solution.

Recent research into basecallers has mainly explored variations in the neural network structure. Networks with recurrent units such as long-short term memory (LSTM) networks and RNNs, temporal convolution networks (TCNs), and attention/transformer networks such as the convolution-augmented transformer have all showed acceptable basecalling accuracy when combined with a CTC layer [12][13][14]. The attention structure could also be used without the CTC layer, which has shown superior performance in NLP applications [15]. Basecallers built solely with attention have, however, not yet demonstrated higher accuracy for basecalling [16] unlike in NLP applications. Natural differences between DNA and speech are likely the reason for this. Nanopore signals have vague transitions between nucleotides, especially within homopolymers or repeated sequences of purines or pyrimidines, and no clear rules of order. On the other hand, speech signals have frequency domain information that separates phonemes, word-separated sentences, and linguistic rules.

This said, the lack of linguistic rules for DNA sequences does not mean there is no prior knowledge about the process that generated the nanopore data. For example, some studies models the ratcheting enzyme, e.g., a helicase, as finite state space machines with well-defined state transitions probabilities driven by ATP concentration [17]. However, it is not straightforward to incorporate such knowledge in basecallers solely built with deep learning. However, we believe that incorporating such prior knowledge could provide a pledge of in-depth understanding of nanopore sequencing and a new direction for future developments.

With this in mind, we wanted to revisit HMMs for basecalling while explicitly addressing some of the shortcomings of prior HMM basecallers. To this end, we propose a new hybrid basecaller called **Lokatt** that uses an explicit duration HMM (EDHMM) model with an additional duration state that models the dwell-time of the dominating k-mer. The duration state allows the basecaller to be applied directly to the raw current measurements by circumventing the need to pre-segment the data, which was problematic in previous HMM basecallers. It also allows us to explicitly model the probability distribution of k-mer dwell times within the pore, e.g., based on a physical understanding of the ratcheting enzyme [17]. However, we still use a neural network to estimate the individual k-mer probabilities, and we train our basecaller using end-to-end training.

We evaluate Lokatt on a methylation-free Ecoli-dataset sequence obtained with an ONT MinION device and Guppy 5 at the SciLifeLab facility in Solna, Sweden. Benchmarking against the Guppy basecaller, a locally trained Bonito basecaller, and the default Bonito basecaller shows that Lokatt provides comparable performance in terms of raw read accuracy. All data and code are available at https://github.com/chunxxc/lokatt.

2 The Lokatt Model

An HMM is a generative Bayesian model used to represent the relationship between a sequence of latent states, which are assumed to be generated according to a Markov process, and a corresponding sequence of observations. For the basecalling task at hand we focus on building a hierarchical HMM structure to encode the temporal dependencies between a k-mer sequence of length M, denoted by $\boldsymbol{K} \triangleq \{K_1, K_2, ..., K_M\}$ where $K_m \in \mathbb{K} \triangleq \{1, \ldots, 4^k\}$, and a current measurement sequence of length N, denoted by $\{X_1, X_2, ..., X_N\}$, where typically $M \ll N$.

The duration of any state with a self transition in a Bayesian state-space model is always geometrically distributed. This is inconsistent with the dwell-times reported for both polymers and helicase [18][19], two popular candidates for ratcheting enzymes. This inconsistency causes basecalling errors in regions rich in homopolymers since current variations are relatively small here, implying that the basecaller can only rely on the

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

statistical modeling of the translocation speed. We address this problem via the introduction of a sequence of M explicit duration variables [20], denoted by $\boldsymbol{D} \triangleq \{D_1, D_2, \ldots, D_M\}$ where $D_m \in \mathbb{N}$. This provides flexibility in terms of assigning an arbitrary dwell-time distribution. Each pair (K_m, D_m) for $m = 1, \ldots, M$ thus encodes the mth k-mer and its dwell time. However, to encode potentially very long dwell times using a limited number of states, we will allow self-transitions between the states encoding the distribution of D_m for $m = 1, \ldots, M$.

In the following two subsections, we formalize the hybrid data model on which Lokatt is based, beginning with the EDHMM and continuing with the DNN observation model.

2.1 The EDHMM structure

We consider a hierarchical and generative Bayesian EDHMM for the nanopore data, constructed as follows. We first draw the number of k-mers, $M \in \mathbb{N}$, from some distribution $\zeta(M)$ Then, we draw the first-order Markov sequence of k-mers $\boldsymbol{K}$ starting with K_1 from a distribution $\xi_0(K_1)$ followed by recursive draws of K_m from K_{m-1} according to a conditional distribution $\xi(K_m|K_{m-1})$ for $m = 2, \ldots, M$. At the same time, we draw the sequence of dwell-times $\boldsymbol{D}$ by drawing each D_m independently from some distribution $\eta(D)$ for $m = 1, \ldots, M$. Finally, D_m measurements $X_{m,d}$ for $d = 1, \ldots, D_m$ are drawn for each k-mer K_m for $m = 1, \ldots, M$, from a conditional distribution $\varphi(X_{m,d}|K_m)$. For notational convenience, we let $\boldsymbol{X}$ denote the sequence of measurements in the order of which they would be obtained, i.e.,

$$\boldsymbol{X} \triangleq \left\{X_{1,1}, \ldots, X_{1,D_1}, X_{2,1}, \ldots, X_{2,D_2}, X_{3,1}, \ldots, X_{M,D_M}\right\},$$

and use simply X_n for $n = 1, \ldots, N$, where $N = \sum_{m=1}^{M} D_m$, when speaking of the nth consecutive measurement. The joint probability distribution of the model is given in Eq. (1).

In order to allow for computationally efficient inference using the model, we make two additional assumptions on the model, namely that $\zeta(M) = (1-\alpha)\alpha^{-M}$ for some $\alpha \in [0, 1)$, and $\eta(D+1) = \gamma\eta(D)$ for all $D \geq \bar{D}$ for some $\gamma \in [0, 1)$ and $\bar{D} \in \mathbb{N}$. In other words, we assume that the sequence length is geometrically distributed and that the duration distribution has a geometric tail. As noted above, the value of these assumptions is that they are inherently encoded in a *finite* state-space model using self transitions between states. The generative model can, with these assumptions, be encoded as a state model with a start state A, an end state B, and $|\mathbb{K}| \times \bar{D}$ intermediate states representing joint k-mer and duration states. The intermediate states are represented by pairs (K, d) where the state variable d acts as a counter that counts down to the next draw of a k-mer and is then stochastically reset according to the explicit duration probability η. The set of paths from A to B is isomorphic with the set of pairs $(\boldsymbol{K}, \boldsymbol{D})$, and a pair can be drawn from $p(\boldsymbol{K}, \boldsymbol{D})$ by a random walk from A to B with the following state transition rules:

1. State A transitions into state (K_1, d) with probability $\alpha\xi_0(K_1)\eta(D = d)$ for any $d = 1, \ldots, \bar{D}-1$, transitions into state $(K_1, \bar{D})$ with probability $\alpha(1-\gamma)^{-1}\xi_0(K_1)\eta(\bar{D})$, and directly transitions into state B with probability $1-\alpha$.
2. State $(K_{m-1}, 1)$ transitions into state (K_m, d) with probability $\alpha\xi(K_m|K_{m-1})\eta(d)$ for any $(K_{m-1}, K_m) \in \mathbb{K} \times \mathbb{K}$ and $d = 1, \ldots, \bar{D}-1$, transitions into state $(K_1, \bar{D})$ with probability $\alpha(1-\gamma)^{-1}\xi_0(K_m|K_{m-1})\eta(\bar{D})$, and transitions into state B with probability $1-\alpha$.
3. State (K_m, d) deterministically transitions into state $(K_m, d-1)$ for any $K_m \in \mathbb{K}$, $d = 2, \ldots, \bar{D}-1$, and $m \geq 1$.
4. State $(K_m, \bar{D})$ self transition into state $(K_m, \bar{D})$ with probability γ for any $K_m \in \mathbb{K}$, and transitions into state $(K_m, \bar{D}-1)$ with probability $1-\gamma$.

The state-space model is a standard implementation of an EDHMM [20]. The mth time the process returns to a state of the form $(K_m, 1)$ for $K_m \in \mathbb{K}$ marks the end of k-mer K_m in the data. Each measurement, $X_n = X_{m,d}$, can also be drawn directly based on the value of state (K_m, d), although we in our particular implementation assume that the observations are independent of d.

The value of the state-space representation is that it allows for efficient inference while maintaining model interpretability [21]. From the specific model presented above, the EDHMM can be constructed into a graph with size $(M \times \bar{D}) \times N$, as illustrated in Fig. 1. The joint data likelihood and k-mer sequence $p(\mathbf{K}, \mathbf{X})$ can, using this graph, be efficiently computed with Eq. 1 by applying the forward algorithm [22]. The data likelihood $p(\mathbf{X})$ can, similarly, be efficiently computed by applying the forward algorithm to a slightly altered graph [22], where $K_m \in \mathbb{K}$ can be arbitrary. The two graphs are sometimes referred to as the *clamped* and *free-running* models respectively [23], and allows us to efficiently compute the posterior sequence likelihood as $p(\mathbf{K}|\mathbf{X}) = p(\mathbf{K}, \mathbf{X})/p(\mathbf{X})$. The conditionally most likely sequence of states and dwell-times $(\boldsymbol{K}, \boldsymbol{D})$ can be computed using the Viterbi algorithm, and the conditionally most likely sequence of states $\boldsymbol{K}$ can be approximately obtained using the beam search algorithm.

Choosing the exact distributions in the model are also rather straightforward. The k-mer transition probabilities $\xi(K_m|K_{m-1})$ can, for example, be estimated with a maximum likelihood (frequency counting) estimator from a reference genome; or by using uniform probabilities for k-mers K_m that originate from K_{m-1}, e.g., set to $\frac{1}{4}$ for each possible one-base transition and zero for any other combination, in order not to bias the model towards any particular genome. The value of α needs to be set very close to 1 in order for the model to plausibly generate reads on the order of hundreds of thousands of bases, and for all intents and purposes, one can set $\alpha = 1$ in the implementation of the inference algorithms which we also do, although this does, strictly speaking, lead to an ill-defined prior distribution. The dwell-time distribution $\eta(D_m)$ can be estimated from sequences realigned at the sample-to-k-mer level. In the particular realization of Lokatt studied herein, we use a log-logistic distribution for η with parameters estimated using linear regression from the average number of large changes in the signal amplitude per sample. At the same time, we choose the tail factor γ so that $\eta(D)$ provides a representative mean dwell-time. We also perform a read-specific duration estimation during training and inference in our implementation since the average duration is longer for reads obtained later during the sequencing run due to ATP depletion. Finally, we replaced $\varphi(X_{m,d}|K_m)$ with scores obtained from a match network [24] $\varphi^{\theta}_{n,K}(\mathbf{X})$, where $K \in \mathbb{K}$, $n = 1, \ldots, N$ and where θ are the trainable weights of the (match) DNN described next.

$$p(\boldsymbol{X}, \boldsymbol{K}, \boldsymbol{D}) = \zeta(M) \times \underbrace{\xi_0(K_1) \prod_{m=2}^{M} \xi(K_m|K_{m-1})}_{\text{Transition model}} \times \underbrace{\prod_{m=1}^{M} \eta(D_m)}_{\text{Duration model}} \times \underbrace{\prod_{m=1}^{M} \prod_{d=1}^{D_m} \varphi(X_{m,d}|K_m)}_{\text{Observation model}} \tag{1}$$

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

2.2 The DNN structure

Lokatt relies on a neural network to dynamically extract the features from the current signal, then map them into scores associated with each k-mer, i.e., $\varphi^{\theta}_{n,K}(\mathbf{X})$. As we want the basecaller to be capable of handling various lengths of inputs, we construct a network with convolution kernels and recurrent units. In particular, we use two types of blocks: i) two residual-convolution blocks consisting of three convolution layers, with $32, 64, 32$ features respectively, and a skip connection [25]; and ii) two bi-directional LSTM blocks [26], with 256 features in the first block on each direction and 1024 for the second block. In the end a dense layer is applied to map the $2 \times 1024 = 2048$ features into $|\mathbb{K}| = 4^k$ dimensions, which in Lokatt is 1024 with $k = 5$. Lokatt contains two of each block type resulting in a total of 15.3 million weights. The complete network is shown in Fig. 2 with a decomposition of the residual-convolution block and the bi-directional LSTM block. We used Swish activation, which nonlinearly interpolates between a linear function and the ReLU function [27][28], between each layer in the network.

2.3 Model training

The weights in Lokatt's DNN are trained end-to-end using a modified conditional maximum likelihood (CML) approach. The vanilla CML objective maximizes the conditional log-likelihood, i.e. $\log p(\mathbf{K}|\mathbf{X})$, which is often decomposed as $\log p(\mathbf{K}|\mathbf{X}) = \log p(\mathbf{K}, \mathbf{X}) - \log p(\mathbf{X})$. Then clamped model is used to compute $\log p(\mathbf{K}, \mathbf{X})$ and the free running model is used to compute $\log p(\mathbf{X})$, respectively, the former with latent states representing the particular target k-mer sequence and latter with latent state from $\mathbb{K}$ representing all possible sequences [23, 24]. Through the two graphs, the gradients can be computed with respect to $\varphi^{\theta}_{n,K}(\mathbf{X})$ and passed down to update the weights in the DNN with back-propagation.

During the training of Lokatt, we realized that this strategy lead to a network that did not generalize well and gave low final scores. In a sense, the model learned to maximize $\log p(\mathbf{K}|\mathbf{X})$ by minimizing $\log p(\mathbf{X})$ and hence did not provide a very good model of the data. This lead the neural network to output low values of the score $\varphi^{\theta}_{n,K}(\mathbf{X})$ overall. Therefore, we instead adopted a modified CML training strategy, where we maximized a weighed linear combination of $\log p(\mathbf{K}|\mathbf{X})$ and $\log p(\mathbf{X})$ to regularize the model in order to provide a balance between posterior decisions and modelling of the data. In particular, we explicitly used

$$\log p(\mathbf{K}|\mathbf{X}) + \lambda \log p(\mathbf{X}) = \log p(\mathbf{K}, \mathbf{X}) - (1 - \lambda) \log p(\mathbf{X})$$

with regularization factor $\lambda = \frac{1}{2}$ as the overall cost function. We also trained the network with a cost function where $p(\mathbf{X})$ was replaced by $p(\mathbf{K}_{\mathrm{BS}}, \mathbf{X})$, i.e, with

$$\log p(\mathbf{K}, \mathbf{X}) - (1 - \lambda) \log p(\mathbf{K}_{\mathrm{BS}}, \mathbf{X}) \,, \tag{2}$$

again with $\lambda = \frac{1}{2}$, but where $\mathbf{K}_{\mathrm{BS}}$ is the output sequence of k-mers from the local-focus beam search decoder explained in the section 3.4. The intuition behind Eq. 2 is to make the training focus more on segments of the data where $\mathbf{K}_{\mathrm{BS}}$ differ from the (correct) reference sequence $\mathbf{K}$. The model trained with this approach showed higher final basecalling accuracy. Therefore, it was used to benchmark Lokatt with the other basecallers.

3 Methods

3.1 Data generation

To be able to evaluate Lokatt, we sequenced *E.coli* non-methylated genomic DNA (D5016, Zymo Research) at two dates using an ONT MinION device. The reason for the choose of DNA was that we initially thought that DNA methylation may play a more important role when basecalling than it did in the end.

The sequencing libraries were prepared by fragmenting the genomic DNA using Covaris g-TUBE and Ligation sequencing kit (SQK-LSK109, Oxford Nanopore). The first sequencing run was performed in December 2019 using Flow Cell chemistry R9.4.1 and basecalled using Guppy 3.2.2. The second sequencing run was performed in November 2021 using Flow Cell chemistry R9.4.1 and basecalled with Guppy 5.0. The reason for performing the second sequencing run was to, first of all, obtain a more recent comparison to the state-of-the-art commercial software, as well as to provide separate data sets for training and evaluation.

The data was divided into training and testing batches based on their position on the genome. We divided the data from the first sequencing run into two batches, batch 1.A and batch 1.B, by separating reads corresponding to the first and second half of the genome, respectively. The second sequencing run was similarly divided into batch 2.A and batch 2.B. The data in batch 1.A was used as training data for the model, while the remaining batches were used for validation and performance evaluations. This allowed us to ensure that the model was not over-fitted to the underlying genome by comparing the performance on batch 2.A, where a homology exists, with the performance on batch 1.B and 2.B where no homology between the training and test data should exist. Similar data division strategies have been used for human genome data sets, where the training and testing were based on different chromosomes. [16]

3.2 Data preparation and DNN pre-training

We obtained a ground-truth nucleotide sequence for each raw read by mapping a tentative read from Guppy to a reference genome with `Minimap2` [29]. We next obtained a sample-to-k-mer level re-alignment using an early implementation of our EDHMM basecaller, which used an observation probability table obtained using the Baum-Welch algorithm [22] instead of a dynamic output from a neural network. The Lokatt model, once in existence, could also be used for this purpose as alignment information is an inherent part of the model.

Based on the alignment, we divided the long reads into segments of 4096 samples with corresponding reference sequences to enable parallel computation with GPUs. A small portion of re-aligned training data was first used to pre-train the neural network structure to reduce the total training time of the computationally costly modified CML training. We trained the network to predict the corresponding k-mer per sample probabilities using a cross-entropy loss.

3.3 End to end training

As described in section 2.3, training with the regularized CML loss from Eq.2 requires two graphs representing $p(\mathbf{K}, \mathbf{X})$ and $p(\mathbf{K}_{\mathrm{BS}}, \mathbf{X})$ to compute their gradients with respect to the output of the DNN, i.e., $\varphi^{\theta}_{n,K}(\mathbf{X})$, for $n = 1, ..., N$. Note here that while $p(\mathbf{X})$ is usually computed using the free running mode, we use the clamped model for both $p(\mathbf{K}, \mathbf{X})$ and $p(\mathbf{K}_{\mathrm{BS}}, \mathbf{X})$. In our implementation, the size of the two graphs are of $\bar{D} \times M \times 4096$ and of $\bar{D} \times M_{\mathrm{BS}} \times 4096$, where the lengths of $\mathbf{K}$ and $\mathbf{K}_{\mathrm{BS}}$, i.e., M and M_{BS}, are on the order of a few hundred bases of overlapping k-mers. To manage the complexity of the inference on these graphs we reply on a custom GPU implementations that can efficiently utilize the sparsity of the graphs. In particular, we implemented the gradient calculation for the EDHMM as custom CUDA kernels [30] and registered them as a customized operations in Tensorflow2 [31]. The gradients are then passed on to the DNN with a standard batch based Adam optimizer [32] from Tensorflow. We trained the whole DNN on the

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

NSC Berzelius compute cluster using 8 Nvidia A100 GPUs [1], for which training on one epoch of data took 576 GPU hours. The CUDA kernel code is available in the Lokatt repository: https://github.com/chunxxc/lokatt.

3.4 Decoding with a local-focus beam search

The final task of basecalling, also known as *decoding*, is to find the sequence of k-mer with the highest posterior probability, i.e. $\mathbf{K} = \arg\max_{\mathbf{K}} p(\mathbf{K}|\mathbf{X})$). The optimal solution to this problem is intractable. It essentially requires an exhaustive search over all possible k-mer sequences of length $M \leq N$, which is infeasible. The Viterbi algorithm, popular for its low complexity, can not solve the target decoding problem using the EDHMM graph. It will instead find the *jointly* optimal sequence $(\mathbf{K}, \mathbf{D})$ that maximizes $p(\mathbf{K}, \mathbf{D}|\mathbf{X})$. However, decoding the optimal k-mer sequence requires marginalizing over the duration states, for which the Viterbi algorithm has no strategy.

We instead rely on the beam search algorithm, which approximates the maximum a posteriori solution by recursively marginalizing the probabilities over a fixed-size list of k-mer sub-sequences $\mathbf{K}_m \triangleq (K_1, \ldots, K_m)$ for $m \leq n$, i.e. *beams*, with high posterior probability $p(\mathbf{K}_m|\mathbf{X}_n)$ for $\mathbf{X}_n \triangleq (X_1, \ldots, X_n)$ and $m \leq n$. Each beam also keeps track of the probability of $p(\mathbf{K}_m, D_m|X_n)$ from which it can compute the probability of new beams and marginalize over D_m as needed. From experimental experience, we know that the beam search algorithm achieves a similar performance to the Viterbi algorithm on our particular EDHMM when we use at least 4096 beams, which already have a significant memory footprint. The beam search performances can be improved by increasing the number of beams, but the pruning and sorting operations inherent to the algorithm become prohibitively slow as sorting complexity scales like $B\log(B)$ for a beam list size of B [33].

To alleviate these issues, we introduce a local-focus beam search algorithm based on the intuition that the correlation between decoding errors reduces quickly over time. Since the sequence to be decoded is handled at the k-mer level, errors can be identified and corrected locally without significantly affecting other k-mers in the sequence. The difference to the standard beam search algorithm is that our algorithm discriminates between beams according to their ending k-mer at the current position during forward propagation and only sorts and prunes the leaf beams according to their end state, similar to the maximum forward propagation in the Viterbi algorithm. We have experimentally observed that the local-focus beam search algorithm achieves the same performance as vanilla beam search decoding despite dramatically reducing execution time. The functional description of our proposed algorithm is given below:

1. *Initialization*
   a. Create $B = 4^k$ beams, indexed by $b = 1, \ldots, B$, corresponding to each possible single k-mer sequence $\mathbf{K}_1 = (K_1 = b)$.
   b. In each beam, store $p(\mathbf{K}_1, D_1)$ for $D_1 = 1, ..., \bar{D}$
2. *Forward beam search from* $n = 2, \ldots, N$
   a. Index each of the B beams according to the last k-mer $K_{m_b} = b$ of $\mathbf{K}_{m_b}$, where we use m_b to stress that the different beams can correspond to k-mer sequences of different lengths.
   b. Expand each beam $\mathbf{K}_{m_b}$ into 5 leaf beams: four beams that correspond to a transition to a compatible next k-mer ending with either one of 'A, C, G, T', and one 'staying' at the current k-mer.
   c. Update $p(\mathbf{K}_m, D_m)$ for $D_m = 1, ..., \bar{D}$ in each of the $5B$ leaf beam following the transition rules described in Section 2.1.
   d. Group the leaf beams into B groups of size 5 according to the last k-mer K_m of $\mathbf{K}_m$, and merge leaf beams that represent the same k-mer sequence $\mathbf{K}_m$ by adding their respective probabilities $p(\mathbf{K}_m, D_m)$.
   e. Sort the leaf beams in each group according to $p(\mathbf{K}_m)$, obtained by marginalizing $p(\mathbf{K}_m, D_m)$ over $D_m \in \{1, ..., \bar{D}\}$. Keep the top leaf beam in each group and prune the rest, which brings the total number of beams back to B.

The above procedure is functionally equivalent to the standard vanilla beam search algorithm apart from the final pruning step (2e), where the beam list is reduced by keeping the single best beam in each of the B groups rather than the B globally best beams. This has obvious benefits in terms of complexity as it is computationally much simpler to in parallel sort the B groups of size (at most) 5 rather than finding the B globally best beams in a list of length $5B$. There is, however, also a more nuanced benefit to inference performance as the local-focus beam search algorithm will keep a beam list with a much more diverse set of tail k-mers K_{m_b} whereas the beam list of the standard beam search algorithm quickly degrades to a list of beams that are different in terms of $\mathbf{K}_{m_b}$ but that all more or less share the same tail k-mers. We believe that this explains our empirical observation that the local-focus beam search algorithm can match the inference performance of the vanilla implementation even when the latter uses a larger list.

In our implementation of the local-focus beam search algorithm, we use a tree structure to store $\mathbf{K}_m$ in memory, where common ancestry represents common initial sub-strings of $\mathbf{K}_m$. Since this tree has at most NB nodes, it can be efficiently implemented on the GPUs without needing dynamic memory allocation. This also allows us to evaluate in fixed time if two expanded leaf beams correspond to the same $\mathbf{K}_m$ in the beam merger step, as this can only happen when they share a common parent beam. One can also use this from a conceptual viewpoint to see that the local merging of beams within each of the B groups is functionally equivalent to a global merging operation. Finally, in order to extract the final approximate maximum a posteriori k-mer sequence $\mathbf{K} = \mathbf{K}_M$, we can read this tree backward from the highest-scoring leaf node to the root in a fashion similar to the backtracking step of the Viterbi algorithm.

4 Results

4.1 Benchmarking

We benchmark the Lokatt basecaller with ONT's Guppy and Bonito basecallers. Guppy 5.0 is ONT's current commercial basecaller, which we only had access to for the 2.A and 2.B data batches. The 1.A and 1.B batches were sequenced with Guppy 3.2.2, whose performance is $2\% - 5\%$ lower than Guppy 5.0 and therefore disregarded in the benchmark. Bonito is an open-source research tool released by ONT that has been actively updated recently. We take a model from Bonito release 0.5.0, choose $dna_r9.4.1_e8_sup@v3.3$, and refer to this as 'Bonito Sup' in what follows as it is supposed to give 'super accuracy' [34] for the $r9.4.1$ pore used in this project. We also trained a Bonito model from scratch using the same basecaller model structure as Bonito Sup, but with the data used in Lokatt's training. We will refer to this as 'Bonito Local'. Bonito Local and Bonito Sup thus use the same decoding strategy. The only difference is that ONT trained Bonito Sup on a different, presumably larger, data set. Both Lokatt and Bonito Local were trained over 5 epochs of training data before they started to over-fit. In the following session, we will discuss the basecalling quality in terms of raw read identities and assembly results.

[1] Berzelius: https://www.nsc.liu.se/support/systems/berzelius-getting-started/

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

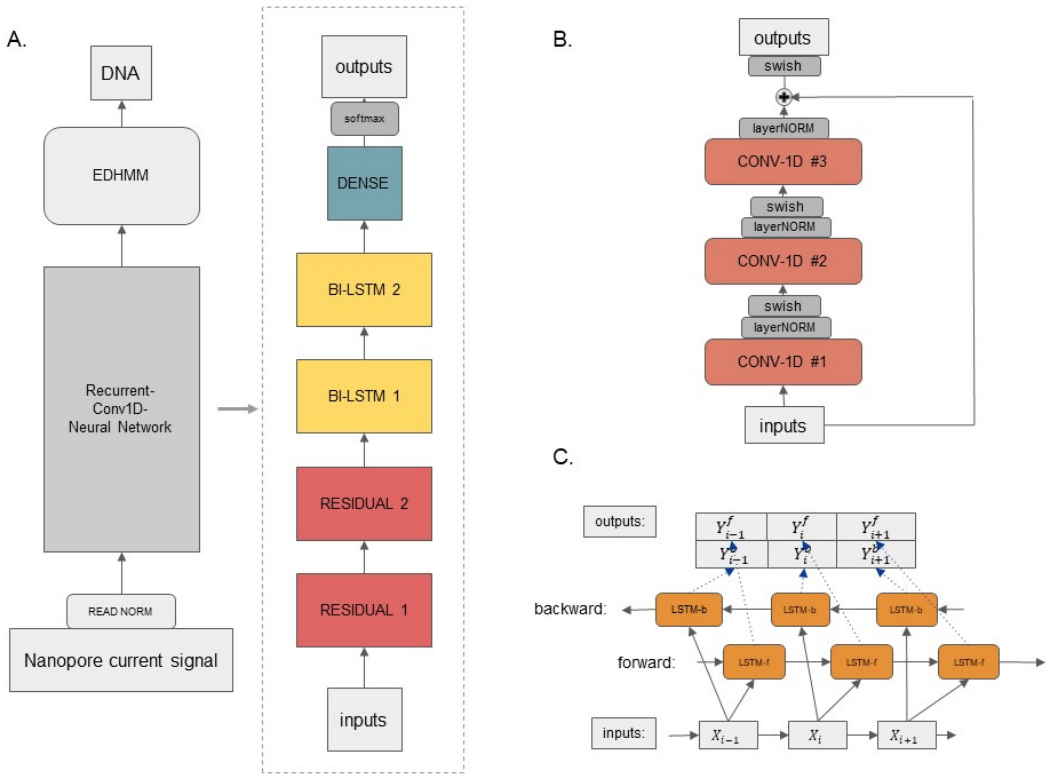

Fig. 2. The overall structure of the Basecaller: A. The main components of Lokatt, from the bottom up, are normalized input, the neural network, the EDHMM layer, and output. The neural network is expanded into main component layer blocks on the right side: two residual blocks, two bi-directional LSTM clocks, and a dense layer. B. The inner structure of the residual block. It consists of three layers of 1D convolution layer, followed by layer-normalization and Swish activation. Then the results are taken sum with the inputs followed by cross-layer normalization. C. The inner structure of the bi-directional LSTM layer. The inputs go through two independent LSTM layers, one in the forward direction and the other backward. The two results are then concatenated along the feature dimension, making the output sequence the same length as the inputs.

4.2 Raw read accuracy

The evaluation of basecalling performance is measured in terms of sequence accuracy, output base length per read, the total number of mapped reads as well as the total number of mapped bases. We define the sequence accuracy in terms of the identity score as computed by Minimap2 [29] and given by the ratio between the number of matches and the total alignment columns:

$$\text{identity} = \frac{\text{matches}}{\text{matches} + \text{mismatches} + \text{insertions} + \text{deletions}}$$

The alignments are obtained with Minimap2 v.2.17-r941 using the default parameters. The training performance with the 1.A data set is reported in Table. 1. The testing performances with 1.B, 2.A, 2.B are reported in Table. 2, Table. 3 and Table. 4 respectively. Results with Guppy 5.0 are included in latter two tables.

Overall the two basecaller provided by ONT, Guppy 5.0 and Bonito Sup, give better performance than the two locally trained basecallers. Bonito Sup, with weights provided by ONT, gives the best basecalling performance for all tests, achieving identity 1.5%-2.7% higher than the rest of the basecallers and outputs 100 to 500 more bases per read. Bonito Sup is significantly better than Bonito Local, despite using the same model structure, and it is plausible that Lokatt would also benefit from training on a larger or different data set. However, since the training data and tuning strategy used for Bonito Sup is not publicly available, we could not assess this hypothesis.

Both Lokatt and Bonito Local are evaluated at the point before they overfit the training data set, which is illustrated by the similar training and testing performances. Between the two basecaller trained with local data, Lokatt achieves higher raw read identity and more mapped bases than Bonito Local. For data set 1.A and 1.B Lokatt did, however, basecall fewer reads, although it achieved a longer average read length than Bonito Local.

We also observed that, during the training procedure of Lokatt, the pre-trained neural network could already achieve a median basecalling accuracy of around 90.5% when coupled with the EDHMM. The regularized CML training increased the median identity score by around 2.0%.

4.3 Basecaller complexity

Bonito utilize the CTC-Conditional Random Field [35] structure on top of 3 convolution layers and 5 LSTM layers alternating in forward and reverse directions, with in total 26.9 million parameters. Despite being nearly twice large as Lokatt with 15.3 million parameters, Bonito is roughly 30 times faster than Lokatt in decoding speed. The high decoding speed results from a smaller CTC graph in Bonito and, thereby, a fast decoding algorithm. Firstly, Bonito's DNN runs with a stride of 5, reducing the costly recurrent computation and shortening the CTC graph size for decoding. Secondly, the CTC graph has only 5 states: blank and 1-mers, i.e., single nucleotides, while Lokatt uses 5-mer with 1024 states. Therefore, for

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

CTC, it is sufficient to use the Viterbi algorithm, which in the CTC graph is equivalent to just taking the maximum output per sample.

We choose to proceed with k-mer instead of single nucleotide as used in CTC to gain a high resolution per sample which will benefit the local-focus decoding algorithm. Using k-mer, however, requires a large-sized graph, i.e., a graph of size $4^k \times N$, leading to a computationally expensive and slow inference process. However, it is certainly within the capability of modern GPU. We have observed that the main bottlenecks are, in fact, the N times recurrent calculation in the LSTM layer and the estimation of the duration distribution.

In theory, Lokatt could also adapt the striding structure and use a smaller k to reduce the graph size. However, we consider the main point of this project to be the introduction of a computational model which can explicitly leverage prior knowledge of the nanopore device. Thus, we did not spend further resources to improve the basecalling speed.

Table 1. Training performances on data batch 1.A

Basecaller	identity	read length	total reads	bases ($\times 10^9$)
Lokatt	$0.909/0.925$	$6073/4738$	453103	2.751
Bonito_Local	$0.898/0.913$	$5901/4581$	454202	2.680
Bonito_Sup	$0.923/0.942$	$6067/4672$	464310	2.817

*identity and read length are listed as 'mean/median'.

Table 2. Testing performances on data batch 1.B

Basecaller	identity	read length	total reads	bases ($\times 10^9$)
Lokatt	$0.910/0.925$	$6112/4823$	376431	2.301
Bonito_Local	$0.896/0.911$	$5928/4661$	377161	2.235
Bonito_Sup	$0.923/0.942$	$6105/4759$	385767	2.355

*identity and read length are listed as 'mean/median'.

Table 3. Testing performances on data batch 2.A

Basecaller	identity	read length	total reads	bases ($\times 10^9$)
Lokatt	$0.896/0.929$	$7773/6754$	1202862	9.351
Bonito_Local	$0.894/0.924$	$7327/6215$	1181169	8.655
Bonito_Sup	$0.915/0.954$	$7859/6777$	1234763	9.704
Guppy 5.0	$0.900/0.933$	$7779/6730$	1205477	9.377

*identity and read length are listed as 'mean/median'.

Table 4. Testing performances on data batch 2.B

Basecaller	identity	read length	total reads	bases ($\times 10^9$)
Lokatt	$0.896/0.930$	$7740/6713$	1010791	7.823
Bonito_Local	$0.892/0.921$	$7289/6169$	990751	7.222
Bonito_Sup	$0.915/0.954$	$7859/6777$	1037837	8.116
Guppy 5.0	$0.900/0.933$	$7743/6687$	1012931	7.843

*identity and read length are listed as 'mean/median'.

5 Discussion

In this project, we have designed, trained, and tested a hybrid basecaller **Lokatt** with a performance that is at least as good as the CTC-CRF structured Bonito basecaller trained on the same data set and comparable to the ONT-trained Bonito and Guppy basecallers.

Lokatt and Bonito differ not just in model design but also the training strategy. Bonito utilizes the CTC loss, where a blank state is manually inserted between every nucleotide. Hence the reference sequence, including the problematic homopolymers, is divided into single bases. The CTC loss typically leads to a dominance of blank predictions in the output of the CTC trained RNN [10]. One interpretation is that the RNN has learned to detect the transition moment of the nucleotides and treat everything else as blank, which is presumably a less complicated task compared with learning the complete characteristic of the measurements as given by the ion current through the nanopore and ratcheting enzyme. However, the dominance of blank predictions could limit the model design and make it more difficult to incorporate changes in the sequencing device.

On top of the CTC loss, the training criteria share a similarity. Specifically, Bonito is trained with a conditional random field (CRF) loss [35], which is equivalent to the CML loss on an undirected graph model such as the CTC graph. In Lokatt's implementation, we have regularized the CML loss with the data likelihood. We have observed that the best regularization factor λ depends on the specific model, e.g., replacing our CNN-LSTM model with a U-net model requires a higher factor than $\lambda = \frac{1}{2}$. One plausible reason that the regularization is needed in the first place could be the mismatch between the model and the real device, like the assumption of independence between the duration distribution and the k-mer as well as between the duration state and the observations. In future versions of Lokatt, we would like to lift the constraints of the EDHMM design and replace it with a more dynamic structure based on more prior knowledge about the sequencing device and chemistry.

Acknowledgements

The authors acknowledge support from the National Genomics Infrastructure in Stockholm funded by Science for Life Laboratory, the Knut and Alice Wallenberg Foundation and the Swedish Research Council, and SNIC/Uppsala Multidisciplinary Center for Advanced Computational Science for assistance with massively parallel sequencing and access to the UPPMAX computational infrastructure. The model training of Lokatt and basecalling were enabled by the supercomputing resource Berzelius provided by National Supercomputer Centre at Linköping University and the Knut and Alice Wallenberg foundation.

Funding

This work has been supported by the Swedish Research Council Research Environment Grant QuantumSense [VR 2018-06169]; and the Swedish Foundation for Strategic Research (SSF) grant ASSEMBLE [RIT15-0012].

References

[1]D. Deamer, M. Akeson, and D. Branton, "Three decades of nanopore sequencing," pp. 518–524, 5 2016.

[2]M. Jain, H. E. Olsen, B. Paten, and M. Akeson, "The oxford nanopore minion: delivery of nanopore sequencing to the genomics community," *Genome Biology*, vol. 17, 12 2016.

[3]D. Branton, D. W. Deamer, A. Marziali, H. Bayley, S. A. Benner, T. Butler, M. D. Ventra, S. Garaj, A. Hibbs, X. Huang, S. B. Jovanovich, P. S. Krstic, S. Lindsay, X. S. Ling, C. H. Mastrangelo, A. Meller, J. S. Oliver, Y. V. Pershin, J. M.

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

Ramsey, R. Riehn, G. V. Soni, V. Tabard-Cossa, M. Wanunu, M. Wiggin, and J. A. Schloss, "The potential and challenges of nanopore sequencing." *Nature biotechnology*, vol. 26, pp. 1146–53, 10 2008.

[4]D. Sheka, N. Alabi, and P. M. Gordon, "Oxford nanopore sequencing in clinical microbiology and infection diagnostics," *Briefings in Bioinformatics*, vol. 22, 9 2021.

[5]Y. Wang, Y. Zhao, A. Bollas, Y. Wang, and K. F. Au, "Nanopore sequencing technology, bioinformatics and applications," *Nature Biotechnology*, vol. 39, pp. 1348–1365, 11 2021.

[6]P. Goyal, P. V. Krasteva, N. V. Gerven, F. Gubellini, I. V. D. Broeck, A. Troupiotis-Tsaïlaki, W. Jonckheere, G. Péhau-Arnaudet, J. S. Pinkner, M. R. Chapman, S. J. Hultgren, S. Howorka, R. Fronzes, and H. Remaut, "Structural and mechanistic insights into the bacterial amyloid secretion channel csgg," *Nature*, vol. 516, pp. 250–253, 12 2014.

[7]N. J. Loman, J. Quick, and J. T. Simpson, "A complete bacterial genome assembled de novo using only nanopore sequencing data," *Nature Methods*, vol. 12, pp. 733–735, 7 2015.

[8]C. L. C. Ip, M. Loose, J. R. Tyson, M. D. Cesare, B. L. Brown, M. Jain, R. M. Leggett, D. A. Eccles, V. Zalunin, J. M. Urban, P. Piazza, R. J. Bowden, B. Paten, S. Mwaigwisya, E. M. Batty, J. T. Simpson, T. P. Snutch, E. Birney, D. Buck, S. Goodwin, H. J. Jansen, J. O'grady, and H. E. Olsen, "Minion analysis and reference consortium: Phase 1 data release and analysis," *F1000Res.*, 2015.

[9]A. C. Rand, M. Jain, J. M. Eizenga, A. Musselman-Brown, H. E. Olsen, M. Akeson, and B. Paten, "Mapping dna methylation with high-throughput nanopore sequencing," *Nature Methods*, vol. 14, pp. 411–413, 2 2017.

[10]H. Teng, M. D. Cao, M. B. Hall, T. Duarte, S. Wang, and L. J. Coin, "Chiron: translating nanopore raw signal directly into nucleotide sequence using deep learning," *GigaScience*, 2018.

[11]O. N. T. plc., "Nanoporetech/bonito: A pytorch basecaller for oxford nanopore reads." https://github.com/nanoporetech/bonito, Feb. 2020.

[12]J. Zeng, H. Cai, H. Peng, H. Wang, Y. Zhang, and T. Akutsu, "Causalcall: Nanopore basecalling using a temporal convolutional network," *Frontiers in Genetics*, vol. 10, 1 2020.

[13]X. Lv, Z. Chen, Y. Lu, and Y. Yang, "An end-to-end oxford nanopore basecaller using convolution-augmented transformer." Institute of Electrical and Electronics Engineers Inc., 12 2020, pp. 337–342.

[14]N. Huang, F. Nie, P. Ni, F. Luo, and J. Wang, "Sacall: a neural network basecaller for oxford nanopore sequencing data based on self-attention mechanism," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2020.

[15]A. Vaswani, G. Brain, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Łukasz Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[16]H. Konishi, R. Yamaguchi, K. Yamaguchi, Y. Furukawa, and S. Imoto, "Halcyon: an accurate basecaller exploiting an encoder–decoder model with monotonic attention," *Oxford Bioinformatics*, vol. 37, no. 9, pp. 1211–1217, 2020.

[17]J. M. Craig, A. H. Laszlo, H. Brinkerhoff, I. M. Derrington, M. T. Noakes, I. C. Nova, B. I. Tickman, K. Doering, N. F. D. Leeuw, and J. H. Gundlach, "Revealing dynamics of helicase translocation on single-stranded dna using high-resolution nanopore tweezers," *PNAS*, vol. 114, pp. 11 932–11 937, 2017.

[18]P. Sarkozy, Jobbágy, and P. Antal, "Calling homopolymer stretches from raw nanopore reads by analyzing k-mer dwell times," *IFMBE Proceedings*, vol. 65, pp. 241–244, 2017.

[19]A. Fornili, A. N. Kapanidis, M. Meli, M. Sustarsic, T. D. Craggs, and G. Colombo, "Dna polymerase conformational dynamics and the role of fidelity-conferring residues: Insights from computational simulations," *Frontiers in Molecular Biosciences | www.frontiersin.org*, vol. 1, p. 20, 2016. [Online]. Available: www.frontiersin.org

[20]S. Yu and H. Kobayashi, "Practical implementation of an efficient forward-backward algorithm for an explicit-duration hmm," *Environment Protection Engineering*, 2007.

[21]M. J. Wainwright, M. I. Jordan *et al.*, "Graphical models, exponential families, and variational inference," *Foundations and Trends® in Machine Learning*, vol. 1, no. 1–2, pp. 1–305, 2008.

[22]A. Krogh, "Hidden markov models for labeled sequences," 1994.

[23]A. Krogh and S. K. Riis, "Hidden neural networks," 1999.

[24]S. Kamaric, "Hidden markov models and neural networks for speech recognition," 1999.

[25]K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9908 LNCS, pp. 630–645, 2016.

[26]M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, pp. 2673–2681, 1997.

[27]P. Ramachandran, B. Zoph, and Q. V. L. G. Brain, "Searching for activation functions," *6th International Conference on Learning Representations, ICLR 2018 - Workshop Track Proceedings*, 10 2017. [Online]. Available: https://arxiv.org/abs/1710.05941v2

[28]S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *Neural Networks*, vol. 107, pp. 3–11, 2 2017. [Online]. Available: https://arxiv.org/abs/1702.03118v3

[29]H. Li, "Minimap2: Pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, pp. 3094–3100, 9 2018.

[30]NVIDIA, P. Vingelmann, and F. H. Fitzek, "Cuda, release: 10.2.89," 2020. [Online]. Available: https://developer.nvidia.com/cuda-toolkit

[31]M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[32]D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," 2015.

[33]E. Sintorn and U. Assarsson, "Fast parallel gpu-sorting using a hybrid algorithm," *Journal of Parallel and Distributed Computing*, vol. 68, pp. 1381–1388, 10 2008.

[34]iiSeymour, "Release v0.5.0 · nanoporetech/bonito," Dec 2021. [Online]. Available: https://github.com/nanoporetech/bonito/releases/tag/v0.5.0

[35]L. John, M. Andrew, and P. F. C.N., "Conditional random fields: Probabilistic models for segmenting and labeling sequence," 2001.

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture