Pagès-Gallego and de Ridder
Genome Biology (2023) 24:71
https://doi.org/10.1186/s13059-023-02903-2

RESEARCH Open Access

Comprehensive benchmark and architectural analysis of deep learning models for nanopore sequencing basecalling

Marc Pagès-Gallego[1,2] and Jeroen de Ridder[1,2*]

*Correspondence: j.deridder-4@umcutrecht.nl

[1] Center for Molecular Medicine, University Medical Center Utrecht, Universiteitsweg 100, 3584 CG Utrecht, The Netherlands
[2] Oncode Institute, Utrecht, The Netherlands

Abstract

Background: Nanopore-based DNA sequencing relies on basecalling the electric current signal. Basecalling requires neural networks to achieve competitive accuracies. To improve sequencing accuracy further, new models are continuously proposed with new architectures. However, benchmarking is currently not standardized, and evaluation metrics and datasets used are defined on a per publication basis, impeding progress in the field. This makes it impossible to distinguish data from model driven improvements.

Results: To standardize the process of benchmarking, we unified existing benchmarking datasets and defined a rigorous set of evaluation metrics. We benchmarked the latest seven basecaller models by recreating and analyzing their neural network architectures. Our results show that overall Bonito's architecture is the best for basecalling. We find, however, that species bias in training can have a large impact on performance. Our comprehensive evaluation of 90 novel architectures demonstrates that different models excel at reducing different types of errors and using recurrent neural networks (long short-term memory) and a conditional random field decoder are the main drivers of high performing models.

Conclusions: We believe that our work can facilitate the benchmarking of new basecaller tools and that the community can further expand on this work.

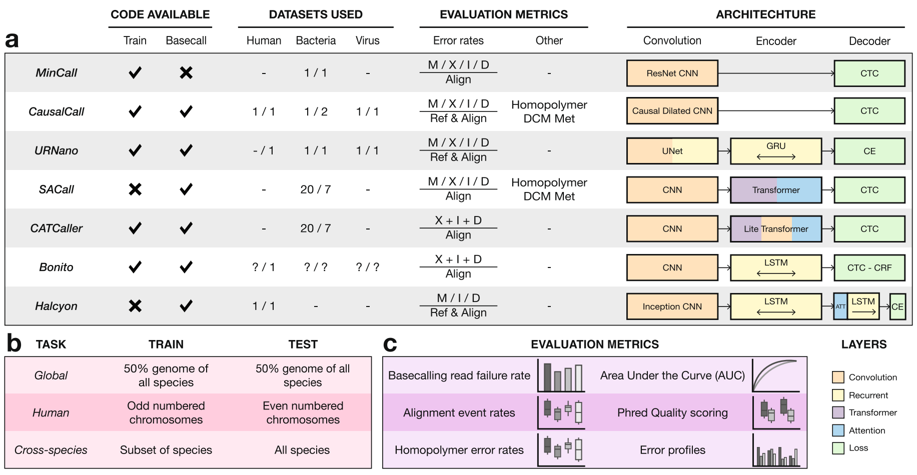
Keywords: Nanopore, Basecalling, Benchmark, Deep learning

Background

Sequencing of DNA (or RNA) can be achieved by translocating nucleic acids through a protein nanopore. By passing an electric current through the nanopore, a signal is measured that is representative for the chemical nature of the different nucleotides inside the pore. Therefore, capturing this current yields a signal that can be translated into a DNA sequence. In 2014, Oxford Nanopore Technologies (ONT) released the first commercial sequencing devices based on this principle.

Basecalling is the process of translating the raw current signal to a DNA sequence [1]. It is a fundamental step because almost all downstream applications depend on it [2]. Basecalling is a challenging task due to several reasons. First of all, the current signal level does not correspond to a single base but is most dominantly influenced by the several nucleotides that reside inside the pore at any given time. Secondly, DNA molecules do not translocate at a constant speed. Therefore, the number of signal measurements is not a good estimate of sequence length. Instead, the detection of signal changes is required to determine that the next base has entered the pore.

To address the basecalling challenge, a wide array of basecallers has been developed both by ONT and the wider scientific community. Basecallers evolved from statistical tests, to hidden Markov models (HMMs) and finally to the use of neural networks [3, 4]. Wick et al. (2019) benchmarked *Chiron* [5] and four other ONT basecallers that were being developed at the time: *Albacore*, *Guppy*, *Scrappie*, and *Flappie*. *Chiron* used developments in the speech to text translation field as it applied a convolutional neural network (CNN) to extract the features from the raw signal, a recurrent neural network (RNN) to relate such features in a temporal manner, and a connectionist temporal classification (CTC) decoder [6] to avoid having to segment the signal. Since then, many other basecallers have been developed and published by the community (Fig. 1a): *Mincall* [7] used a deep CNN with residual connections [8], *Causalcall* [9] used a CNN with causal dilations [10], *SACall* [11] and *CATCaller* [12] used transformers [13] and lite-transformers [14] respectively, *URNano* [15] used a convolutional U-net with integrated RNNs [16], and *Halcyon* [17] used a sequence-to-sequence (Seq2Seq) model with attention [18, 19]. ONT also updated its main



a

	CODE AVAILABLE		DATASETS USED			EVALUATION METRICS		ARCHITECHTURE		
	Train	Basecall	Human	Bacteria	Virus	Error rates	Other	Convolution	Encoder	Decoder
MinCall	✓	✗	-	1 / 1	-	M / X / I / D Align	-	ResNet CNN		CTC
CausalCall	✓	✓	1 / 1	1 / 2	1 / 1	M / X / I / D Ref & Align	Homopolymer DCM Met	Causal Dilated CNN		CTC
URNano	✓	✓	- / 1	1 / 1	1 / 1	M / X / I / D Ref & Align	-	UNet	GRU	CE
SACall	✗	✓	-	20 / 7	-	M / X / I / D Align	Homopolymer DCM Met	CNN	Transformer	CTC
CATCaller	✓	✓	-	20 / 7	-	X + I + D Align	-	CNN	Lite Transformer	CTC
Bonito	✓	✓	? / 1	? / ?	? / ?	X + I + D Align	-	CNN	LSTM	CTC - CRF
Halcyon	✗	✓	1 / 1	-	-	M / I / D Ref & Align	-	Inception CNN	LSTM	ATT LSTM CE

b

TASK	TRAIN	TEST
Global	50% genome of all species	50% genome of all species
Human	Odd numbered chromosomes	Even numbered chromosomes
Cross-species	Subset of species	All species

c

Fig. 1 Overview of the latest basecaller models and benchmark. Code available columns indicate whether their github repositories contained scripts to train a model from scratch or basecall using their trained models. Values in the datasets indicate the number of species they used for training (left) and testing (right). Alignment rates reported for matches (M), mismatches (X), insertions (I), and deletions (D) and whether they were normalized based on the length reference sequence (Ref) or the length of the alignment (Align). Metrics separated by a slash (/) are reported independently; others are reported as a combined error rate. Some models also report homopolymer and methylation site error rates. The architecture section shows a high level overview of the different models. Overview of the different tasks in the benchmark with their respective training and testing data (pink). Overview of the evaluation metrics in the benchmark (purple)

basecaller (*Guppy*) with the architecture from *Bonito*, which substituted the CTC decoder with a conditional random field (CRF) decoder [20].

With new AI methods being continuously developed and tailored to various applications, we can reasonably assume that further improvements to basecalling are around the corner. Problematically, there is currently no standardized benchmarking method to evaluate basecalling performance. Instead, evaluation metrics are often defined on a per-publication basis and are limited to only certain aspects of basecalling accuracy (Fig. 1a). This may bias conclusions as more favorable metrics are chosen for head to head algorithm comparisons. Furthermore, data used for training and testing is not uniform across publications, which makes it difficult to distinguish between data or neural network architecture driven improvements. This is in stark contrast to e.g., the field of machine learning itself, where new models and methods are benchmarked with community driven evaluation metrics and datasets, such as the CASP competition [21] and the CAMI benchmark [22]. Such efforts facilitate the comparison of new and existing models, and the identification of their respective strengths and weaknesses [23]. It also centralizes all the information, making it easier for users to find the best method for their end goal and lowering the bar to take on the challenge for new developers.

Here, we present a comprehensive set of benchmarking tools, including a range of evaluation metrics, that can be used to analyze the strengths and weaknesses of basecaller models. This toolbox can be used as benchmark for the standardized training and cross-comparison of existing and future basecallers. Using this toolbox, we benchmarked the neural network architectures of the latest basecaller models, as well as over eighty novel neural network architectures by combining the different components of existing basecaller models. This allowed us to study the influence of the different neural networks components, some of which greatly influence basecalling performance. Finally, we also benchmarked the top performing architectures using different training and testing datasets, allowing us to we evaluate the robustness of the models to training data biases. Together, our work provides insight into what the best model architecture is under what conditions and establishes that model architectures not currently implemented in existing basecalling tools outperform state of the art for certain metrics.

Results

Benchmark setup

We first developed a basecalling benchmarking framework enabling new and existing basecalling algorithms to be easily compared. Moreover, our benchmark facilitates the study of individual components of basecallers, as different combinations of basecaller components can readily be evaluated. The framework is divided into two main components: (i) standardized datasets for model training and testing (Fig. 1b) and (ii) evaluation metrics for extensive assessment of basecalling performance (Fig. 1c). The benchmark components are fully automated with minimal dependencies and publicly available on https://github.com/marcpaga/nanopore_benchmark, making it easy for developers to focus on algorithm development rather than benchmarking.

Datasets and tasks

We gathered datasets that have previously been used for benchmarking and were publicly available for download and used the R9.4 or R9.4.1 pore chemistry. This yielded 615,642 reads from 3 human datasets, 1 Lambda phage dataset, and 60 bacterial datasets encompassing 26 different bacterial species. After read annotation using *tombo*, 460,225 reads (75%) were suitable for benchmarking (Additional file 1: Fig. S1, Table S1). Because nanopore sequencing has several possible downstream applications, we defined three different tasks: global, human, and cross-species. Each task uses specific portions of training and test data to simulate different scenarios. (i) The *global* task evaluates the performance of a general-purpose model by training and testing the model using data from all available species. In this task, models have access to most data, which is a similar strategy employed in *Causalcall* and *URNano*. (ii) The *human* task evaluates the performance of a human specialized model by training and testing exclusively on human data. This strategy is used in *Halcyon* and *Mincall*, although in the latter *E. coli* is used. (iii) The *cross-species* task evaluates the performance of a trained model on never before seen species. This allows us to evaluate the robustness of the model to overfit on genomic features from the training set, such as k-mer distribution or base modifications. This is achieved by training using a subset of bacterial species and testing on data from all species. This is most similar to the benchmark by Wick et al., (2019) and similar to the evaluation strategy followed in *SACall* and *CATCaller* models. However, here, we explicitly take into account the genomic similarity between species when splitting them between train and test sets (see the "Methods" section).

Evaluation metrics

We define six basecalling evaluation metrics allowing making unbiased and quantitative comparisons of basecalling efficacy across various conditions. (i) Not all reads can be successfully basecalled. Since failed reads are excluded from any subsequent evaluation, it is important to report the *failure rate of the model*. Read basecalling failure falls into one of several categories: not basecalled (the model did not produce any basecalls), not aligned (during evaluation, the alignment algorithm did not produce an alignment), short alignment (the length of the alignment is less than 50% of the length of the reference), or other (other possible errors that prevent the read from being evaluated). (ii) After alignment of a basecalled read to its reference, four types of events can be distinguished: matches, mismatches, insertions, and deletions. We report the different types of *alignment rates* independently, since two models may have similar match rates but have different profiles in the types of errors they produce. (iii) *Homopolymers* are particularly difficult to basecall in nanopore sequencing; for this reason, an independent metric is used to report them. We defined homopolymers as any repetitive sequence of the same base of length 5 or longer. (iv) Basecalling errors can have different distributions depending on the bases that surround the event. *Error profiles*, similar to the trinucleotide context used to define mutational signatures [24], subdivide the errors based on its sequence context to study potential basecalling biases. Ideally, a model's error profile is unbiased and uniform, for example, to avoid biased variant calling. (v) *PhredQ scores* are values that indicate the confidence of the model in its prediction of a base.

Ideally, incorrect calls have low quality scores while high scoring bases are mostly correct. PhredQ scores can be calculated from the predicted distribution of probabilities; however, the calculation of the PhredQ score varies between decoder types; therefore, we evaluate the separation of their distributions. (vi) PhredQ scores can also be used to evaluate the model at read level. This is done by evaluating the trade-off between filtering out bad quality reads and the change in performance of the model by calculating the *area under the curve (AUC)*.

Benchmark results

To compare existing basecallers and benchmark them in a comparable manner, we re-implemented all their architectures. It is important to note, unless otherwise stated, that everytime a model is mentioned by name (*Bonito*, *Causacall*, *Halcyon*, etc.), we are referring to their neural network architectures and not the models as they were published. The results of the human benchmarking challenge are summarized in Fig. 2 and (Additional file 1: Table S2).

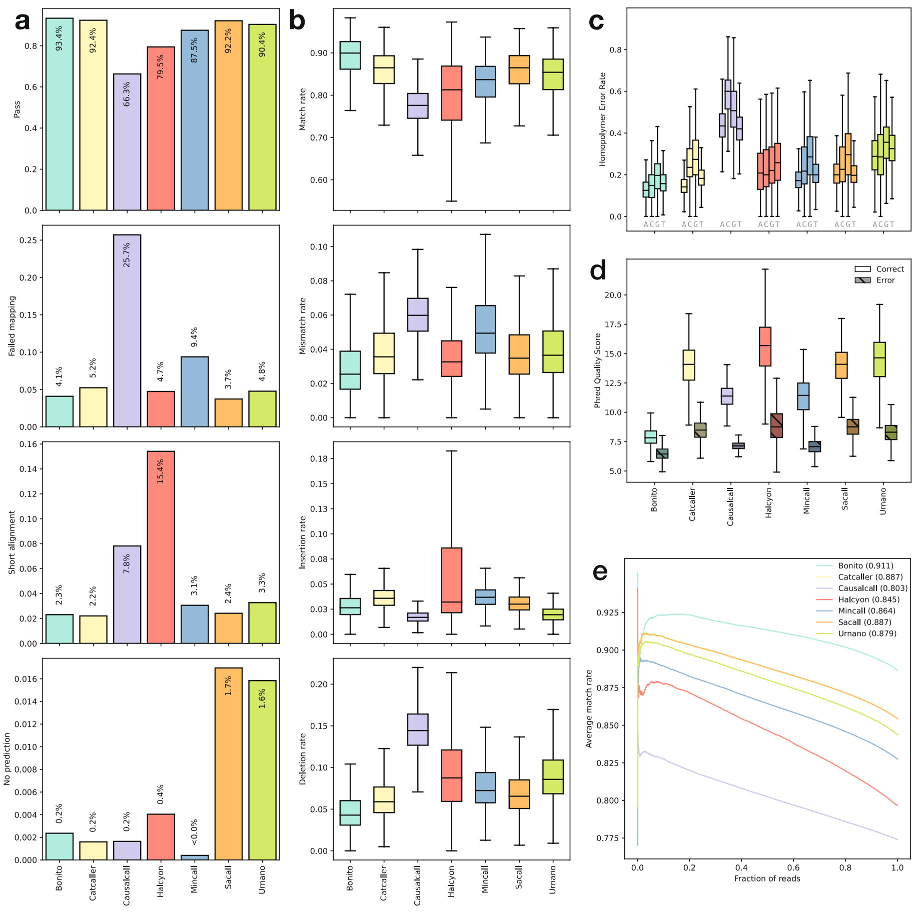


Fig. 2 Latest basecallers benchmark on the human task. **a** Basecalling reads failure rates: pass, failed mapping, short alignment, and no prediction (from top top bottom). **b** Alignment event rates: match, mismatch, insertion, and deletion (from top to bottom). **c** Homopolymer error rates per base. **d** PhredQ scores distributions for correctly (light) and incorrectly (dark) predicted bases. **e** AUC of match rate sorted by average read PhredQ score

Basecalling failure is a major determinant of reported performance

We first noted that the number of reads that failed basecalling varies substantially. Distinctively, *Causacall*, *Halcyon*, and *Mincall* only managed to properly basecall 66%, 79%, and 87% of the reads respectively, while the rest of the models managed > 90% (Fig. 2a). It is therefore critical to include such a metric in model evaluation since models could be skipping difficult to basecall reads, which would skew results towards a false higher performance.

Different methods prevail at different measures

We evaluated the performance of the different architectures using the alignment event rates (Fig. 2b). *Bonito* performed best in three out of the four metrics. It has the highest median match rate (90%) and the lowest median mismatch (2.5%) and deletion (4.3%) rates. *Causalcall* achieved the lowest median insertion rate (1.7%); however, it performed worst in the other three metrics with the lowest median match rate (77.6%) and highest mismatch (6%) and deletion (14.4%) rates. *Halcyon* shows the highest variation in performance rates, demonstrating that in addition to the median, the distribution across the reads is important to consider while comparing basecallers. It is therefore critical to not only report the error rates but also their distributions.

Homopolymer error rates are correlated with alignment event rates

Homopolymers are especially difficult to basecall because, for long stretches of the same base, the signal does not change, and since the DNA translocation speed is not constant, the number of measurements is not a good indicator of the length of the homopolymer. For such stretches of DNA, *Bonito* performed best with the lowest median error rate (14.9% averaged across all four bases) and the lowest median error rate for each base individually. *Causalcall* performed significantly worse than the rest of the models with the highest median error rate (44.5% averaged across bases) (Fig. 2c). We observed a performance correlation between alignment event rates and homopolymer error rates; however, the latter have a significantly higher error rate likely due to the inherent difficulty of basecalling such stretches.

Utility of PhredQ scores varies across methods

To evaluate the relationship between the predicted bases and their PhredQ scores. We first consider the distributions of the scores of the correct and incorrect bases (Fig. 2d). For all model architectures, correct bases have higher scores than incorrect bases; *Causalcall* has the smallest overlap between the two distributions (0.7%), followed by the rest of the models with similar overlaps (6–8%) except *Halcyon* (12%) and *Bonito* (32%). Secondly, we calculated AUCs by sorting the reads based on their average quality scores and determining the area under the normalized cumulative score (Fig. 2e). All architectures showed a correlation between read quality and average match rate. Not surprisingly, *Bonito* performed best with an AUC of 0.91. *CATCaller* and *SACAll* are, however, close contenders both with an AUC of 0.886. Importantly, each model has its own PhredQ score offset that determines how their quality scores are calibrated. As a result, quality scores across models, even when compared in a standardized benchmark, are not directly numerically comparable.

The signatures of basecalling errors

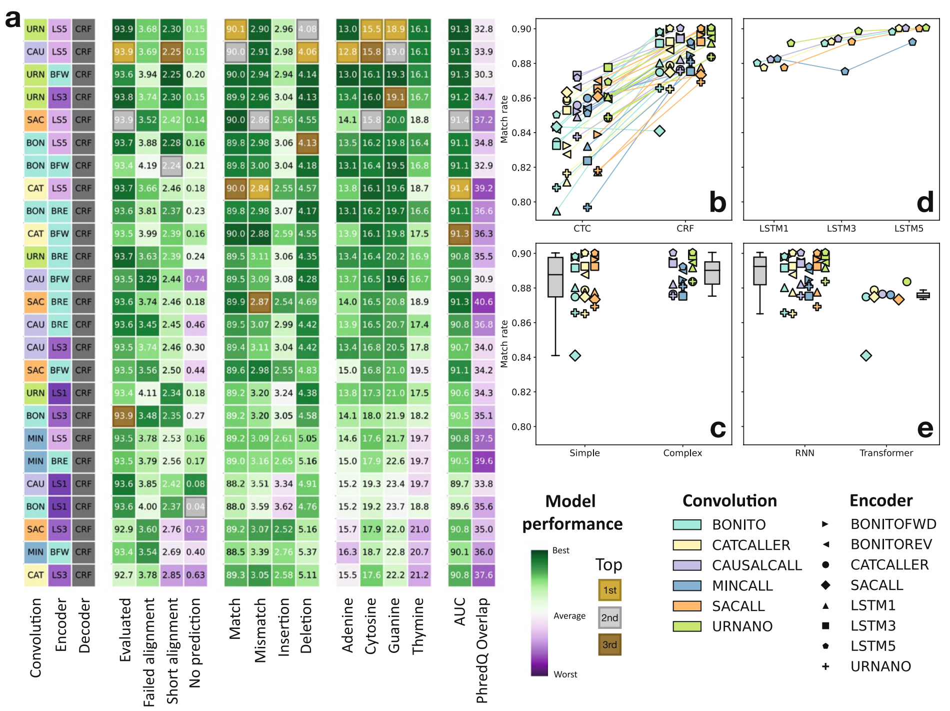
Finally, we evaluated the different types of mistakes in the context of the two neighboring bases in the basecalls (Additional file 1: Fig. S2). In general, these "error signatures" reveal that there are differences between the accuracy of the models depending on the predicted base context. Across all models, cytosine has low error rates (≈10%) when predicted in CCT or TCT context; however, it can have significantly higher error rates (> 30%) when predicted in the context of the 3-mers TCC or TCG. We noticed that many of the contexts with higher error rates contain a CG motif, suggesting the increased error might be due to the potential methylation status of cytosine. To evaluate if specific models have particular error biases we performed hierarchical clustering on the pairwise Jensen-Shannon divergences between the error signatures (Additional file 1: Fig. S3). This revealed that *Causalcall* and *Halcyon* are the two most different models in terms of "error signatures". The rest of the models have similar error profiles (lowest Pearson correlation coefficient between them: 0.95). We can conclude that basecalling errors are biased since they are not uniformly distributed across the 3-mer contexts. However, the error profiles are very similar between basecallers, suggesting that training data may play a stronger role in defining these error biases than the architecture of the model itself.

Architecture analysis

The benchmarking setup also allows straightforward investigation of which components of the neural networks provide the main performance gains. To this end, we created novel architectures by combining the convolution and encoder and decoder modules from existing basecallers, as well as some additional modules. We again used the human task to evaluate the different models. In total, ninety different models were evaluated and ranked based on the sum of rankings across all metrics (Fig. 3a, Additional file 1: Fig. S4). Out of the original models, *Bonito* again performs best but reached 9th place in the overall ranking. Consequently, our grid search reveals eight new model architectures that perform better in general. However, improvements in performance made by these models are small; for example, in comparison to the *Bonito* model, the alignment event rate improvements and homopolymer error rates are smaller than 1%, suggesting that we may be reaching the performance limits obtainable given the training data used.

CRF decoder is vastly superior to CTC

We observed that most of the high performing models used the CRF decoder module. We therefore compared the change in performance between pairs of models whose only difference was the decoder (Fig. 3b, Additional file 1: Fig. S5). We see that for almost all models, using a CRF decoder leads to a general improvement of performance with a mean increase in match rate of 4% and mean decreases of mismatch, insertion, and deletion rates of 1%, 1%, and 2% respectively. Some exceptions are models that used the *Mincall* or *URNano* convolution modules, which have a mean increase in insertion rates of 1%, although their other alignment rate metrics still improve significantly. This is in concordance with our previous results, where *Causalcall* and *URNano* demonstrated



Convolution	Encoder	Decoder	Evaluated	Failed alignment	Short alignment	No prediction	Match	Mismatch	Insertion	Deletion	Adenine	Cytosine	Guanine	Thymine	AUC	PhredQ Overlap
URN	LS5	CRF	93.9	3.68	2.30	0.15	90.1	2.90	2.96	4.08	13.0	15.5	18.9	16.1	91.3	32.8
CAU	LS5	CRF	93.9	3.69	2.25	0.15	90.0	2.91	2.98	4.06	12.8	15.8	19.0	16.1	91.3	33.9
URN	BFW	CRF	93.6	3.94	2.25	0.20	90.0	2.94	2.94	4.14	13.0	16.1	19.3	16.1	91.3	30.3
URN	LS3	CRF	93.8	3.74	2.30	0.15	89.9	2.96	3.04	4.13	13.4	16.0	19.1	16.7	91.2	34.7
SAC	LS5	CRF	93.9	3.52	2.42	0.14	90.0	2.86	2.56	4.55	14.1	15.8	20.0	18.8	91.4	37.2
BON	LS5	CRF	93.7	3.88	2.28	0.16	89.8	2.98	3.06	4.13	13.5	16.2	19.8	16.4	91.1	34.8
BON	BFW	CRF	93.4	4.19	2.24	0.21	89.8	3.00	3.04	4.18	13.1	16.4	19.5	16.8	91.1	32.9
CAT	LS5	CRF	93.7	3.66	2.46	0.18	90.0	2.84	2.55	4.57	13.8	16.1	19.6	18.7	91.4	39.2
BON	BRE	CRF	93.6	3.81	2.37	0.23	89.8	2.98	3.07	4.17	13.1	16.2	19.7	16.6	91.1	36.6
CAT	BFW	CRF	93.5	3.99	2.39	0.16	90.0	2.88	2.59	4.55	13.9	16.1	19.8	17.5	91.3	36.3
URN	BRE	CRF	93.7	3.63	2.39	0.24	89.5	3.11	3.06	4.35	13.4	16.4	20.2	16.9	90.8	35.5
CAU	BFW	CRF	93.5	3.29	2.44	0.74	89.5	3.09	3.08	4.28	13.7	16.5	19.6	16.7	90.9	30.9
SAC	BRE	CRF	93.6	3.74	2.46	0.18	89.9	2.87	2.54	4.69	14.0	16.5	20.8	18.9	91.3	40.6
CAU	BRE	CRF	93.6	3.45	2.45	0.46	89.5	3.07	2.99	4.42	13.9	16.5	20.7	17.4	90.8	36.8
CAU	LS3	CRF	93.5	3.74	2.46	0.30	89.4	3.11	3.04	4.42	13.4	16.8	20.5	17.8	90.7	34.0
SAC	BFW	CRF	93.5	3.56	2.50	0.44	89.6	2.98	2.55	4.83	15.0	16.8	21.0	19.5	91.1	34.2
URN	LS1	CRF	93.4	4.11	2.34	0.18	89.2	3.20	3.24	4.38	13.8	17.3	21.0	17.5	90.6	34.3
BON	LS3	CRF	93.9	3.48	2.35	0.27	89.2	3.20	3.05	4.58	14.1	18.0	21.9	18.2	90.5	35.1
MIN	LS5	CRF	93.5	3.78	2.53	0.16	89.2	3.09	2.61	5.05	14.6	17.6	21.7	19.7	90.8	37.5
MIN	BRE	CRF	93.5	3.79	2.56	0.17	89.0	3.16	2.65	5.16	15.0	17.9	22.6	19.7	90.5	39.6
CAU	LS1	CRF	93.6	3.85	2.42	0.08	88.2	3.51	3.34	4.91	15.2	19.3	23.4	19.7	89.7	33.8
BON	LS1	CRF	93.6	4.00	2.37	0.04	88.0	3.59	3.62	4.76	15.2	19.2	23.7	18.8	89.6	35.6
SAC	LS3	CRF	92.9	3.60	2.76	0.73	89.2	3.07	2.52	5.16	15.7	17.9	22.0	21.0	90.8	35.0
MIN	BFW	CRF	93.4	3.54	2.69	0.40	88.5	3.39	2.76	5.37	16.3	18.7	22.8	20.7	90.1	36.0
CAT	LS3	CRF	92.7	3.78	2.85	0.63	89.3	3.05	2.58	5.11	15.5	17.6	22.2	21.2	90.8	37.6

Fig. 3 Benchmark of architecture components. **a** Top 25 best performing model combinations. **b** Comparison of CTC to CRF decoder. **c** Comparison of simple (*Bonito*, *CATCAller*, *SACall*) to complex (*Causalcall*, *Mincall*, *URNano*) convolutions. **d** Comparison of bidirection LSTM depth (1, 3, or 5 layers). **e** Comparison of RNN to Transformer encoders

the lowest insertion rates of all the models, showcasing that it is their convolutional architectures that boosts performance for this type of metric. Notably, a decrease in homopolymer error rates is also observed for the models with *Mincall* or *URNano* convolution modules that include a CRF decoder. However, results for the other models are more varied and depend on the base. Consistently with these improvements, we observe an average improvement of 3% in the AUCs. However, the PhredQ overlap between correct and incorrect predictions worsened with a median increase of 30%.

Complex convolutions are most robust, but simple convolutions are still very competitive

Another main architectural difference is the complexity and depth of the convolutional layers: ranging from two or three simple convolutional layers like in *Bonito*, *CATCaller*, and *SACall*, to more elaborate convolutional modules like *Causalcall*, *URNano*, or *Mincall*. We find that the top four ranked models use the *URNano* or *Causalcall* architecture (Fig. 3a). However, the six following models all use one of the simpler CNNs. More complex convolutional architectures perform better in general, specifically *Causalcall* and *URNano* (Fig. 3c, Additional file 1: Fig. S6). Simple convolutional architectures can also perform as good or even better, however they are more dependent on the encoder architecture that follows.

RNNs are superior to transformers and are depth dependent

Transformer layers have gained popularity in other fields due to increased performance and speed [25]. However, our top ten models all use RNN (LSTM) layers in their encoders. A direct comparison shows that RNNs outperform Transformer layers in all the metrics (Fig. 3d, Additional file 1: Fig. S7). Therefore, the attention mechanism appears to be less useful for this task as the RNNs are able to properly learn the time-axis relationships between samples. We also evaluated whether using deeper LSTM layers has any effect on the performance of the models. We see that there is a clear correlation between depth and performance, encouraging the use of deeper

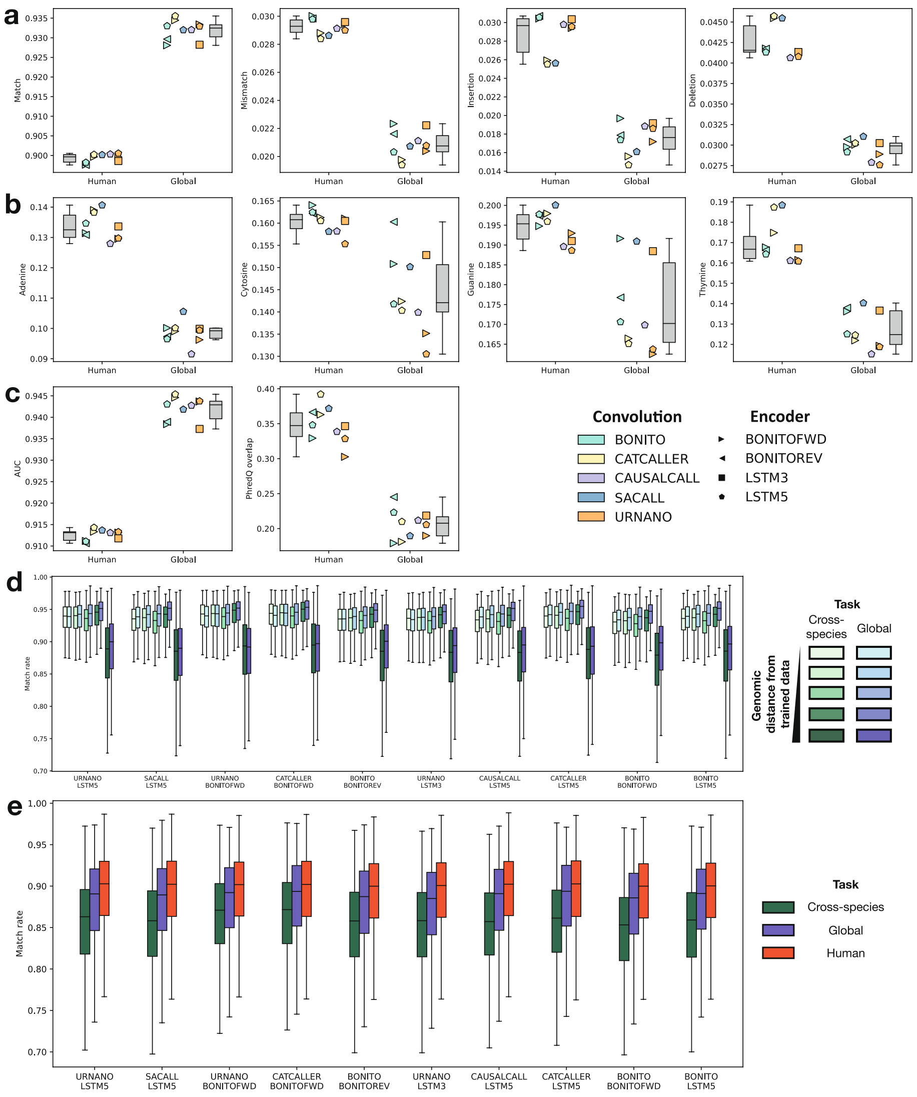


Fig. 4 Performance comparison on different tasks. Comparison between the human and global task on the top 10 best performing model combinations: **a** alignment event rates, **b** homopolymer error rates, and **c** PhredQ scoring. **d** Comparison between the cross-species (green) and global task (blue). Each boxplot contains a set of species split according to the cross-species task. Darker colors indicate increasing genomic distance from the species in the training set. The darkest colors contain lambda phage and human data. **e** Comparison between the human, global and cross-species task only on human data

encoders for this task, regardless of the preceding convolutional architecture (Fig. 3e, Additional file 1: Fig. S8).

Training on multiple species boosts model performance across all models

To assess adaptation of the model to the training dataset, we evaluated the top ten models (ranked based on their performance on the human task) on the global task. We observe that training on multiple species boosts the performance of all ten models across all metrics (Fig. 4). In particular, we observe a 3.2% improvement on mean match rate and 1.1% and 1.3% decrease in mean insertion and deletion rates, respectively (Fig. 4a). We also see a general improvement in homopolymer error rates, with decreases of 3.4% (A), 1.5% (C), 2% (G), and 4.2% (T) in mean error rates (Fig. 4b). As expected, there is also an improvement in AUCs of 2.9% (Fig. 4c). Surprisingly, we observe a significant decrease in the overlap between correct and incorrect base distribution, which was 35% for the human task trained models, and decreased to 21% for the global task trained models (Fig. 4d). These improvements in performance could be due to more variance in the data, but it could also be that non-human data is easier to basecall.

Lacking a species in the training data penalizes model performance

The cross-species benchmark task is aimed at evaluating the robustness of the models to data from species they have not been trained on. We find that all models performed similarly on all bacterial species, regardless of their k-mer genomics distance to the genomic training set. However, all models perform significantly worse on the two non-bacterial sets (*Homo sapiens* or *Lambda phage*) (Fig. 4d). This trend can be seen across all evaluation metrics (Additional file 1: Fig. S9). We therefore asked whether this drop in performance was due to the lack of training data on those species or whether those particular species are more difficult to basecall. To answer this question, we analyzed the data from the global task in the same manner by splitting the species as in the cross-species task. We observed that, in general, models trained on the global task perform better (0.1%, 0.3%, 0.8%, 0.7%, and 0.9% average mean match rate improvement for each set). There is however, a similar trend in which *Homo sapiens* and *Lambda phage* reads have a significantly lower performance than the rest of species (Fig. 4d).

We hypothesized that the *Homo sapiens* and *Lambda phage* genomes might be more difficult to basecall in general. To further investigate this we evaluated the read length distributions, error profiles, 3-mer, and homopolymer contents of the three datasets. We found no correlation between read length and match rate (Additional file 1: Fig. S10a-b). We found, however, that the human dataset contained more homopolymers than the two other datasets (Additional file 1: Fig. S10c). We also observe that, in the error profiles of the human dataset, that 3-mers containing a CG motif have the highest error rates (Additional file 1: Fig. S10d), which is not the case on the bacterial or virus datasets. Since these datasets are from native DNA, it is likely that CG methylation is the cause of that increased error rate, which has also been previously reported [26]. We finally calculated the average 3-mer error rate, weighted by their fraction in the test data: the human dataset has the highest weighted error (0.138), followed by the virus (0.126) and the bacteria (0.081). With this, we concluded that the the *Homo sapiens* and *Lambda phage* are overall more difficult datasets and that the lack of data in the cross-species task heavily contributed to the lower performance.

Finally, we compared the performance of these models only on the *Homo sapiens* data and included the results from the Human task. The results again show that the lack of data is detrimental to the performance of the model (Fig. 4e, Additional file 1: Fig. S11). Furthermore, we can observe that all models performed best when only trained with Homo sapiens data, with an average mean match rate improvement of 3.8% and 1.1% when compared to the cross-species and global tasks respectively. This result might encourage the use of species specific models, however, caution should be taken as it is possible that models start to overfit and memorize features like genomic sequences, GC-biases and homopolymer length distributions.

Discussion

Nanopore basecalling is a critical task in accurate DNA sequencing and currently relies on deep learning, a field in which new algorithms are regularly proposed. However, the lack of consensus in benchmarking data and evaluation metrics makes it complicated and cumbersome to value new contributions by the field. Here, we propose a series of tasks and a set of clearly defined evaluation metrics which can serve as a starting point for what should become a community effort: both by researchers and ONT. In the future, both additional tasks and metrics could be added [26]. For example, additional datasets could be added to further study the viability of species-specific models. And computational metrics, such as basecalling speed and compute requirements, could also be added in the future.

Given these proposed tasks and metrics, we re-implemented and benchmarked the latest existing basecallers to evaluate their neural network architectures performances at read-level. To do so, we implemented previously published methods in a coherent framework enabling training on the same data and making a fair comparison not influenced by implementation details or platform. Out of the original models, *Bonito*'s architecture achieved the best overall performance. However, our results show that other architectures can obtain better results in other metrics. For this reason, it is important to keep expanding the list of metrics so that end users can choose the best architecture for their end goals. While this work only focuses on read-level accuracy, users might consider consensus-level accuracy if high coverage data is available. We demonstrate that the basecalling error profiles are not uniform, and therefore theorize that a higher coverage will be required to achieve error-free consensus calling depending on the DNA sequence.

To investigate the ideal architecture for basecalling more generally, we created new models by combining their different components. We show that using a CRF decoder, over the more traditional CTC decoder, boosts performance significantly and it is likely the reason why *Bonito*'s architecture performs so well in the initial benchmark. A CTC decoder predicts each state independently, and it has been its main criticism over the years. For this reason, Seq2Seq models, like *Halcyon*, were preferred over CTC in other fields since they are able to predict the sequence of states in a conditional manner. However, Seq2seq models are significantly slower than non-recurrent models (CTC and CRF) and require several training steps to be able to deal with large windows of data. The CRF decoder brings the best of both approaches by predicting in a conditional manner while still having a non-recurring decoding step. We also show that deep RNNs (LSTM) are superior to transformer layers. Transformer layers have gained significant popularity in language processing tasks due to their attention mechanism and speed. However, it appears the attention mechanism is not beneficial for the basecalling task. We hypothesize that, while the attention mechanism might be good for long distance relationships between inputs, the temporal relationships in the electric signal are local enough that RNNs are sufficient for the task. Our results also show that both simple and complex convolutional architectures can achieve competitive performance. Finally, we demonstrate that lack of training data for a particular species decreases model performance, and for species-specific tasks, models trained solely on that particular species have the potential to perform better than more general-purpose models.

Conclusions

In this work, we gathered several benchmark datasets, defined train and test sets for different tasks, and proposed a plethora of evaluation metrics with the goal to establish a basecalling benchmark baseline. We hope that this facilitates the comparison of further basecalling algorithm improvements, such as new neural network architectures, but also other data processing steps like normalization. We also analyzed the latest basecaller neural networks in order to understand their architecture-performance relationships. As a rule of thumb, we conclude that the CNN-LSTM-CRF combination will give the most competitive performance. However, one must also carefully consider the training data, as large differences between training and inference, in k-mer composition or DNA modifications, will significantly lower the performance of the basecaller.

Methods

Data

We gathered datasets that had previously been used for benchmarking, were available and were sequenced using a R9.4 or R9.4.1 pore chemistry (Additional file 1: Table S3). With that criteria, we used the bacterial dataset from [27] and the human genome reference (NA12878/GM12878, Ceph/Utah pedigree) dataset from [28]. The human dataset contained many different sequencing runs. We arbitrarily chose three experiments so that each different ligation kit (rapid, ligation and ultra) would be included: FAB42828, FAF09968 and FAF04090. We also included a Lambda phage dataset that we sequenced. The Lambda phage genome DNA material was purchased from NEB (N3011S). The sample was prepared according to the ONT Ligation KIT and sequenced with a MinION flow cell.

Data annotation

Data was annotated using the Tombo resquiggle tool (v1.5.1). First, reads were aligned to the reference sequences using their basecalls. Reference genomes were used as reference sequences for all the datasets; with the exception of the train portion of the bacterial dataset from Wick et al. (2019), which was provided already with a reference sequence for each read. Using Tombo, we aligned the raw signal to the expected signal according to the reference. Reads that did not align to the reference genome or provided a bad resquiggle quality according to Tombo were discarded (Additional file 1: Fig. S1).

Tasks definitions

We decided to define three tasks in order to simulate different case scenarios of Nanopore sequencing applications. We simulate this by controlling the data that is used for training and testing in each case. The global task resembles a general purpose model, where the model has been trained on most of the data available. The human task is used to evaluate the feasibility of a species specific model. Finally, the cross-species task is used to evaluate the robustness of the models to unseen species. For each task, we defined a set of reads to be used for training and testing (Additional file 1: Table S1). In the global task, reads were split according to their mapping position in their respective reference genome (human, bacterial, or lambda phage). Bacterial and lambda phage reads that mapped to the first 50% of the genome were used for training, and the rest were used for testing. Reads that mapped to both halves of the genome were discarded. Human reads that mapped odd numbered chromosomes were used for training and reads that mapped to even numbers were used for testing. A total of 100k reads were selected for training and 50k reads for testing. The number of reads was equally split for each species; therefore, each species would contribute with a maximum of 3571 for training and 1785 for testing. Some species did not have sufficient reads to reach this quota. A total of 81,955 reads were used for training and 47088 reads for testing. For the human task, reads were split as described in the global challenge. For this challenge, reads were also split equally between chromosomes, and a total of 42,812 and 25,000 reads were used for training and testing. For the cross-species task, species were split between training and testing. First, we defined the similarity between species based on the the distribution of 9-mers in their reference genomes. We calculated the pairwise Jensen-Shannon divergence between all species pairs. We then performed single-linkage clustering on the distance matrix and binned the distances into 4 bins (Additional file 1: Fig. S12). We then recursively selected species for training. At each iteration, we selected one species and added it to the training pool. We then recalculated the distance between the species in the training pool as a whole and the rest of the species. We pseudo-randomly added species to the training pool in a manner that species in the test pool would have a different grades of distance to species in the training pool. We did this by counting the number of test species in each distance bin relative to the training pool and then picking the species which would change the distribution of counts the least in the next iteration. We continued this process until at least 10 species were chosen for training, with a maximum of 12 allowed. Testing species were divided into different difficulty categories based on their distance to the closest species in the train set (Additional file 1: Table S4). Similarly to the global task, a total of 50k and 25k reads were selected for training and testing. The testing set was further subdivided between 20k reads, that would come from the test species, and 5k reads, that would come from the train species. Each set of reads was equally divided between the different species. A total of 48,013 reads were used for training, 5000 reads were used for testing from the training species, and 24,335 reads were used for testing from the testing species.

Models

Model architectures were recreated using Pytorch (v1.9.0) based on their description in their publications and their github repositories. If their implementation was done using Pytorch, code was reused as much as possible (Additional file 1: Fig. S13-27).

Model training

Non-recurrent models (all except *Halcyon*) were trained for 5 epochs with a batch size of 64. All models were trained on the same task data which was also given as input in the same order. Models inital random parameters were initialized via a uniform distribution with values ranging from − 0.08 to 0.08. Reads were sliced in non-overlapping chunks of 2000 data points. Models were trained using an Adam optimizer (initial learning rate = $1e^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, weight decay = 0). Learning rate was initially increased linearly for 5000 training steps from 0 to the initial learning rate of the optimizer as a warm-up; the learning rate was then decreased using a cosine function until the last training step to a minimum of $1e^{-5}$. To improve model stability, gradients were clipped between − 2 and 2. Halcyon was trained similarly to non-recurrent models with the following differences: models were trained first for 1 epoch with non-overlapping chunks of 400 data points, then for 2 epochs with chunks of 1000 data points and finally for 2 epochs with chunks of 2000 data points. This was necessary because training directly using 2000 data points chunks led to unstable model training. This phenomenon is also described in the original Halcyon publication [17], requiting this transfer learning approach to ameliorate the issue. Recurrent models were also trained without warm-up and with a 0.75 scheduled sampling. During training, 5% of the training data was used for validation from which accuracy and loss were calculated without gradients. Validation data was the same for all models. The state of the model was saved every 20,000 training steps. The model state was chosen based on the best validation accuracy during training. Models were evaluated on hold out test data from the task being evaluated.

Original model recreation and benchmark

URNano used cross-entropy as its loss; however, since the objective of the benchmark was basecalling and not signal segmentation, we used a CTC decoder instead. All the other models were recreated as stated in their respective publications; when in doubt, their github implementations were used as reference.

Comparison of original models to model recreations

Since *Causalcall* and *Halcyon* performed worse than the rest of the models, we evaluated the original models *Causalcall*, *Halcyon*, and *Guppy* model published by the authors and compared them against our PyTorch implementations (SAdditional file 1: Fig. S28, Additional file 1: Table S5). When evaluating the original *Halcyon*, we were unable to completely basecall all 25k reads in the test set due to memory limitations; we therefore compared our recreation based only on the ≈13k reads that were basecalled by the original *Halcyon* model from the human task test set. We used *Guppy* (v5.0.11, latest version) for the comparison between original models and our PyTorch implementations. In terms of reads that we consider evaluable, we see small differences (less than 3%) between the original versions and our implementations of *Causalcall* and *Guppy*. However, we see some differences in the types of failed reads between the original *Causalcall*, which has 7% more reads that failed mapping, whereas our recreation had 3% more reads with short alignments. Surprisingly, basecalls from the original *Halcyon* produce only 46% of reads suitable for evaluation, (30% less than our recreation). A significant 34% of reads failed mapping (30% more than our recreation) to the reference. There is also an increase, although smaller, on the 19% of reads that have short alignments to the reference (5% more than our recreation) (Additional file 1: Fig. S28a). Regarding our implementation of *Guppy*, we find that differences are small, with at most a 2% difference. We then looked at the alignment event rates (Additional file 1: Fig. S28b). Differences between the two *Guppy* models were very small, with the largest being a 1.3% difference in increased match rate from the original version. The original *Causalcall* showed improved match performance, with increased match (2%) and a decreased deletion (6%) rates; however, it showed a slight increase in mismatch (1%) and insertion (1%) rates as well as higher variability across reads. Finally, the original *Halcyon* performed worse in all metrics except deletion rate. However, its performances are less variable across reads. Homopolymer error rates show a similar trend (Additional file 1: Fig. S28c), the original *Causalcall* performs significantly better with a more similar to the other models average error rate (25.6%), while the other two models show very similar performances. We finally compared the models regarding their PhredQ scores: when comparing *Bonito* to *Guppy*, we saw a large difference in the scale of the scores (Additional file 1: Fig. S28d); however, *Guppy* still had an overlap between the distributions of 28%. On the other hand, the original *Causalcall* showed a significant increase in the overlap between distributions (48%). (Additional file 1: Fig. S28e). Correlating with the event rates results, the original versions of *Causalcall* and *Guppy* performed slightly better than our recreated counterparts with AUCs of 0.837 and 0.937 respectively. The original *Halcyon* does not report any PhredQ scores. With these results, we concluded that although there are some differences between the original models and our recreations, these are minor and could be attributed to training strategies and used data.

Architecture analysis

Most models contain a convolutional module that later directly feeds into an encoder (recurrent/transformer) module. To be able to combine modules from different models without changing the original number of channels, we included a linear layer in between the convolution and encoder modules to up-scale or down-scale the number of channels. After this additional linear layer, we applied the last activation function of the preceding convolutional module. Contrary to the other models, the convolution modules from *URNano* and *Causalcall* do not reduce the amount of input timepoints. For those modules, we also included an extra convolution layer with the same configuration as the last convolution layer in *Bonito* (kernel size = 19, stride = 5, padding = 9). This layer had the same number of channels as the last convolutional layer of *URNano* or *Causalcall*. This convolution layer was necessary in order to both use transformer encoders and/or a CRF decoder due to memory requirements. We also included three non-used encoder architectures: either one, three or five RNN-LSTM bidirectional stacked layers with 256 channels each.

Evaluation metrics

Evaluation metrics are based on the alignment between the predicted sequence and the reference sequence. Alignment is done using Minimap2 (2.21) [29] with the ONT configuration for all metrics except accuracy. Accuracy is based on the Needleman-Wunsch global alignment algorithm implemented in Parasail (1.2.4) [30]. The global alignment is configured with a match score of 2, a mismatch penalty of 1, a gap opening penalty of 8 and a gap extension penalty of 4. Accuracy is used to evaluate the best performing state of the models during training based on the validation fraction of the data. During training, short sequences have to be aligned; however, during testing, complete reads have to be aligned, for which Minimap2 is necessary.

Accuracy Accuracy is defined as the number of matched bases in the alignment divided by the total number of bases in the reference sequence.

Alignment rates Match, mismatch, insertion, and deletion rates are calculated as the number of events of each case divided by the length of the reference unless otherwise stated.

Homopolymer error rates Homopolymer regions are defined as consecutive sequences of the same base of length 5 or longer. Error rates on homopolymer regions are calculated by counting the number of homopolymers with errors (one or more mismatches, insertions, or deletions) and dividing it by the number of homopolymer bases.

PhredQ scoring PhredQ scores are calculated using the fast_ctc_decode library from ONT. Average quality scores are calculated for all the correct and incorrect bases for each read. Differences between mean scores between correct and incorrect bases are reported. AUCs are calculated by sorting the basecalled reads according to their mean Phred quality score and calculating the average match rate for cumulative fraction of reads in steps of 50.

Error profiles Error profiles are calculated for all 3-mers by counting the number of events (mismatches for each base, insertions and deletions) in the context of the two neighboring bases of the event itself according to the basecalls. Rates for each event are calculated by dividing each event count by the total number of 3-mer occurrences in the read. Error profiles are also calculated for each base, independently of their context. Randomness of error is defined as the Jensen-Shannon divergence between each 3-mer error profile and their corresponding base error profile.

Software and hardware requirements Packages and their versions used for training and evaluation can be found on (Additional file 1: Table S6). All analysis were run on Python 3.7.8 and CUDA version 10.2. We used the following hardware requirements: 32 CPU cores and 64Gb of RAM (data processing and model performance evaluation); it is possible to reduce these requirements at the expense of longer compute time; 4 CPU cores, 128Gb of RAM, and 1 NVIDIA RTX6000 GPU (model training basecalling).

Supplementary Information

The online version contains supplementary material available at https://doi.org/10.1186/s13059-023-02903-2.

Additional file 1. Figs. S1-S28, Tables S1-S6.

Additional file 2. The review history.

Acknowledgements
We thank Tobias Dansen for critical reading of the manuscript. We thank Carlo Vermeulen for critical reading of the manuscript and contribution of the Lambda phage sequencing data. We thank Mike Vella for critical reading of the manuscript. We thank Vlado Menkovski for very helpful discussions.

Peer review information
Andrew Cosgrove was the primary editor of this article and managed its editorial process and peer review in collaboration with the rest of the editorial team.

Review history
The review history is available as Additional file 2.

Authors' contributions
MPG and JdR designed the analysis and benchmark. MPG performed the analysis and wrote the code. MPG drafted the first version of the manuscript with guidance from JdR. JdR contributed major parts of the manuscript and revised the manuscript. All authors read and approved the final manuscript.

Funding
This work was funded by Health Holland (No. LSHM19029)

Availability of data and materials
Datasets
Nanopore sequencing data: human raw data and basecall datsets (FAF04090, FAF09968, FAB42828) are available at https://github.com/nanopore-wgs-consortium/NA12878/blob/master/Genome.md [28]; bacterial raw data are available at the Monash University repository, download links are available at https://github.com/marcpaga/nanopore_benchmark/tree/main/download [27]; lambda phage raw data is available on the Sequence Read Archive under the PRJNA926802 bioproject ID [31].
Code
Source code and scripts used to recreate and train the models are available at https://github.com/marcpaga/basecalling_architectures [32]. Source code and scripts used for benchmarking (data download and evaluation) are available at https://github.com/marcpaga/nanopore_benchmark [33]. Both repositories are under the Unlicense license and accessible at: https://zenodo.org/record/7657037 (DOI: 10.5281/zenodo.7657037) [34].

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
Mike Vella works at Oxford Nanopore Technologies; he had no influence on the design or conclusions of the analysis. JdR is cofounder of Cyclomics. JdR has received reimbursement of travel and accommodation expenses to speak at meetings organized by Oxford Nanopore Technologies.

Received: 1 August 2022 Accepted: 20 March 2023
Published online: 11 April 2023

References

1. Rang FJ, Kloosterman WP, de Ridder J. From squiggle to basepair: computational approaches for improving nanopore sequencing read accuracy. Genome Biol. 2018;19(1):90. https://doi.org/10.1186/s13059-018-1462-9. https://genomebiology-biomedcentral-com.proxy.library.uu.nl/articles/10.1186/s13059-018-1462-9.
2. Wang Y, Zhao Y, Bollas A, Wang Y, Au KF. Nanopore sequencing technology, bioinformatics and applications. Nat Biotechnol. 2021;39(11):1348–65. https://doi.org/10.1038/s41587-021-01108-x.
3. Boža V, Brejová B, Vinař T. DeepNano: Deep recurrent neural networks for base calling in MinION nanopore reads. PLoS ONE. 2017;12(6):e0178751. https://doi.org/10.1371/journal.pone.0178751. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0178751.
4. Stoiber M, Brown J. BasecRAWller: streaming nanopore basecalling directly from raw signal. bioRxiv. 2017. https://doi.org/10.1101/133058.

5. Teng H, Cao MD, Hall MB, Duarte T, Wang S, Coin LJMM. Chiron: translating nanopore raw signal directly into nucleotide sequence using deep learning. GigaScience. 2018;7(5). https://doi.org/10.1093/gigascience/giy037. http://dx.doi.org.proxy.library.uu.nl/10.1093/gigascience/giy037.
6. Graves A, Fernández S, Gomez F, Schmidhuber J. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In: Proceedings of the 23rd international conference on Machine learning - ICML '06. New York: ACM Press; 2006. p. 369–376. https://doi.org/10.1145/1143844.1143891. http://portal.acm.org.proxy.library.uu.nl/citation.cfm?doid=1143844.1143891.
7. Miculinić N, Ratković M, Šikić M. MinCall - MinION end2end convolutional deep learning basecaller. GitHub. 2019. arXiv preprint arXiv:1904.10337.
8. He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 2016. p. 770–8. https://doi.org/10.1109/CVPR.2016.90.
9. Zeng J, Cai H, Peng H, Wang H, Zhang Y, Akutsu T. Causalcall: nanopore basecalling using a temporal convolutional network. Front Genet. 2020;10:1332. https://doi.org/10.3389/fgene.2019.01332. https://www.frontiersin.org/articles/10.3389/fgene.2019.01332/full.
10. van den Oord A, Dieleman S, Zen H, Simonyan K, Vinyals O, Graves A, et al. WaveNet: a generative model for raw audio. 2016. p. 1–15. arXiv preprint arXiv:1609.03499.
11. Huang N, Nie F, Ni P, Luo F, Wang J. SACall: a neural network basecaller for Oxford Nanopore sequencing data based on self-attention mechanism. IEEE/ACM Trans Comput Biol Bioinforma. 2020;XX(X):1–10. https://doi.org/10.1109/TCBB.2020.3039244.
12. Lv X, Chen Z, Lu Y, Yang Y. An end-to-end Oxford nanopore basecaller using convolution-augmented transformer. IEEE/ACM Trans Comput Biol Bioinforma. 2020:6. https://doi.org/10.1101/2020.11.09.374165.
13. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. Adv Neural Inf Process Syst. 2017;2017-Decem(Nips):5999–6009.
14. Wu Z, Liu Z, Lin J, Lin Y, Han S. Lite transformer with long-short range attention. ICLR 2020. 2020. p. 1–13. arXiv preprint arXiv:2004.11886.
15. Zhang YZ, Akdemir A, Tremmel G, Imoto S, Miyano S, Shibuya T, et al. Nanopore basecalling from a perspective of instance segmentation. BMC Bioinformatics. 2020;21(136). https://doi.org/10.1186/s12859-020-3459-0.
16. Ronneberger O, Fischer P, Brox T. In: U-Net: Convolutional Networks for Biomedical Image Segmentation. Cham: Springer International Publishing; 2015. p. 234–41.
17. Konishi H, Yamaguchi R, Yamaguchi K, Furukawa Y, Imoto S. Halcyon: an accurate basecaller exploiting an encoder-decoder model with monotonic attention. Bioinformatics. 2021;37(9):1211–1217. https://doi.org/10.1093/bioinformatics/btaa953. https://academic-oup-com.proxy.library.uu.nl/bioinformatics/article/37/9/1211/5962086.
18. Sutskever I, Vinyals O, Le QV. Sequence to sequence learning with neural networks. Adv Neural Inf Process Syst. 2014;4(January):3104–12.
19. Luong MT, Pham H, Manning CD. Effective approaches to attention-based neural machine translation. In: Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing. 2015. p. 1412–1421. https://doi.org/10.18653/v1/d15-1166.
20. Lafferty J, McCallum A, Pereira FCN. Conditional random fields: probabilistic models for segmenting and labeling sequence data. Deparmental papers (CIS), University of Pennsylvania. 2001;11(1):1–84. https://doi.org/10.29122/mipi.v11i1.2792.
21. Moult J, Krzystof F, Kryshtafovych A, Schwede T, Tramontano A. Critical assessment of methods of protein structure prediction (CASP) – round x. Proteins. 2014;82(02):1–6. https://doi.org/10.1002/prot.24452.Critical.
22. Sczyrba A, Hofmann P, Belmann P, Koslicki D, Janssen S, Dröge J, et al. Critical assessment of metagenome interpretation - a benchmark of metagenomics software. Nat Methods. 2017;14(11):1063–71. https://doi.org/10.1038/nmeth.4458.
23. Going for algorithm gold. 2008. https://doi.org/10.1038/nmeth0808-659.
24. Alexandrov LB, Nik-Zainal S, Wedge DC, Campbell PJ, Stratton MR. Deciphering signatures of mutational processes operative in human cancer. Cell Rep. 2013;3(1):246–259. https://doi.org/10.1016/j.celrep.2012.12.008. https://linkinghub.elsevier.com/retrieve/pii/S2211124712004330.
25. Karita S, Chen N, Hayashi T, Hori T, Inaguma H, Jiang Z, et al. A comparative study on transformer vs RNN in speech applications. ASRU 2019. 2019. arXiv preprint arXiv:1909.06317.
26. Delahaye C, Nicolas J. Sequencing DNA with nanopores: troubles and biases. PLoS ONE. 2021;16(10):1–29. https://doi.org/10.1371/journal.pone.0257521.
27. Wick RR, Judd LM, Holt KE. Performance of neural network basecalling tools for Oxford nanopore sequencing. Genome Biol. 2019;20(1):1–10. https://doi.org/10.1186/s13059-019-1727-y. https://genomebiology.biomedcentral.com/articles/10.1186/s13059-019-1727-y.
28. Jain M, Koren S, Miga KH, Quick J, Rand AC, Sasani TA, et al. Nanopore sequencing and assembly of a human genome with ultra-long reads. Nat Biotechnol. 2018;36(4):338–45. https://doi.org/10.1038/nbt.4060.
29. Li H. Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics. 2018;34(18):3094–100. https://doi.org/10.1093/bioinformatics/bty191.
30. Daily J. Parasail: SIMD C library for global, semi-global, and local pairwise sequence alignments. BMC Bioinformatics. 2016;17(1). https://doi.org/10.1186/s12859-016-0930-z.
31. Pagès-Gallego M, de Ridder J. Nanopore sequenced (R9.4.1) Lambda phage dataset. 2023. https://doi.org/10.5281/zenodo.7728175.
32. Pagès-Gallego M, de Ridder J. Deep learning architectures for basecalling. Github; 2023. https://github.com/marcpaga/basecalling_architectures.
33. Pagès-Gallego M, de Ridder J. Nanopore benchmark for basecallers. Github; 2023. https://github.com/marcpaga/nanopore_benchmark.
34. Pagès-Gallego M, de Ridder J. Comprehensive benchmark and architectural analysis of deep learning models for Nanopore sequencing basecalling. 2023. https://doi.org/10.5281/zenodo.7657037.

Publisher's Note