# Machine Learning in Applications GU03 Project Report

Tilocca Salvatore, Carlucci Francesco, Serra Matteo, Fiorio Ludovico
Politecnico di Torino

CONTENTS



LIST OF FIGURES

## List of Tables

# Machine Learning in Applications GU03 Project Report

***Abstract*—Nanopore-based DNA sequencing allows faster and cheaper generation of longer reads on portable devices. Basecalling means inferring DNA bases from nanopore's electric current signal. It requires neural networks to reach competitive performances. To further improve sequencing accuracy, new models with new architectures are continually being proposed. The state of the art model is Bonito [1], a deep learning-based basecaller recently developed by ONT, whose neural network architecture is composed of three convolutional layers followed by five LSTM layers. In order to improve its trade off between accuracy and energy efficiency we tried to create a Spiking version of this architecture. Our spiking architecture achieved a slightly lower accuracy with a drastically reduced theoretical energy footprint on neuromorphic hardware. This report showcases our findings, ideas, and eventual course of action that we applied to the project "Bio-inspired Basecallers" for the "Machine Learning in applications" course at Politecnico di Torino (A.Y. 2022/2023)[1].**

## I. Introduction

DNA or RNA sequencing performed using nanopore (as in Fig. 2) use specific devices that records the variations in electric signal caused by the passage of DNA or RNA strands through the pore. This process, known as 'squiggle,' results in a continuous modulation of current. Subsequently, employing dedicated software, a transformation is executed, converting the squiggle data into sequence reads, with each read representing a single DNA/RNA strand. These reads contains not only information about canonical bases but also details about epigenetic modifications, such as methylation. The signal information is subsequently processed by basecalling algorithms to decipher the sequence of bases. However, these signals are susceptible to sequencing noise and the occurrence of DNA mutations, making them challenging to decode efficiently. Currently, neural network-based basecalling algorithms are used for this purpose.

Machine learning approaches for basecalling include software tools like Albacore, Guppy, Scappie, and Flappie, developed by ONT. Additionally, the research community has contributed with various tools, such as Nanocall [2], DeepNano [3], Chiron [4], and Causalcall [5]. An important improvement in accuracy was achieved with the Bonito architecture, albeit limited by its speed, as a matter of fact Bonito was found to be a relatively slow model. The fact that Bonito exhibited slowness, coupled with the overarching goal of attaining superior energy efficiency, underscored the endeavour to craft a solution that leaned as heavily towards SNN-based architecture as feasible.

[1]Project Github repository: https://github.com/srrmtt/GU03-bioinspired-basecaller

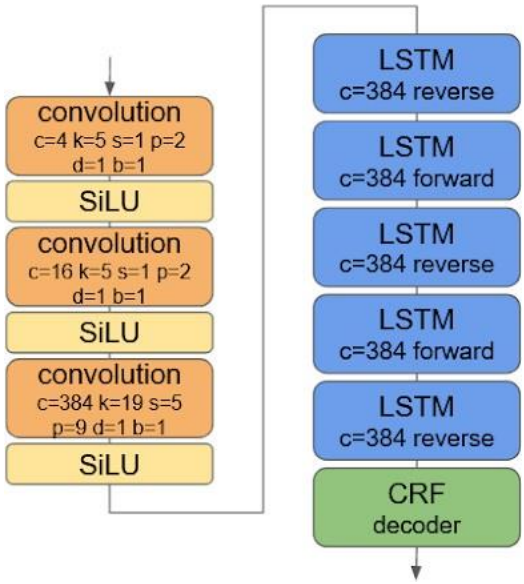


Fig. 1. The starting Bonito architecture. On the left the convolutional feature extractor, on the right the encoder and CRF decoder

## II. Related works

Since the 1980s when nanopore sequencing was first proposed many data analysis methods have been applied to the basecalling task [6]. As previously stated the application of machine learning techniques to basecalling already produced various network architectures and softwares in the last years [7]. Most of them aimed to improve model performances using consensus read, similarly to model ensemble concept [8], or applying recent breakthroughs of other field, like natural language processing [9]. Other works designed new components [10] or used NAS techniques to enhance speed, delivering the highest performances to real-time production applications [11]. More recent studies focused on developing new models able to run on the edge, e.g. on the coral accelerator [12]. At the same time spiking NN have proved to be a valuable tool for the sake of energy and memory footprint reduction, when deployed on specific neuromorphic hardware, especially for temporal data applications [13].

## III. Background

### A. Nanopore Sequencing

A nanopore is a microscopic structure with a diameter in the order of nanometers (a billionth of a meter). It is used

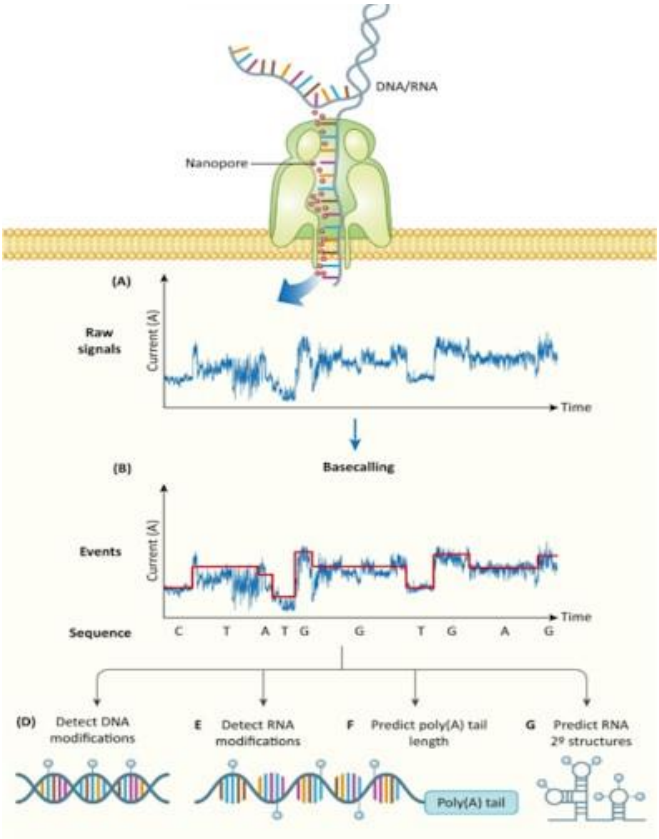


Fig. 2. How the nanopore sequencing pipeline works

to analyse and sequence biological molecules such as DNA or RNA, its sequencing pipeline is shown in figure 2. In the context of DNA sequencing, the sequence translocate through the nanopore one strand at a time, the different nucleotide combinations in the nanopore causes changes in electrical current, called ”squiggles”, that will be processed obtaining a raw signal, stored in a FAST5 file. This allows longer DNA reads to be directly analysed, with minimum library preparation in a wet lab. The whole procedure is nowadays miniaturized in devices like the ONT’s MinION, unlocking in field and real-time sequencing at a drastically cheaper price, lowering the entry barrier for researchers and laboratories in genetics and genomics.

### B. Basecalling

A ”basecaller” is a software or algorithm used in DNA sequencing to interpret raw signals generated by sequencing devices and identify the chemical bases (A, C, G, T) in a DNA sequence, then stored in FASTQ files. FASTQ is a text-based format for storing both a biological sequence (usually nucleotides) and its corresponding quality scores.

Basecalling requires deep learning techniques to achieve accuracies high enough for practical uses. Only modern basecallers allowed nanopore sequencing to compete against other short-read third generation technologies.

### C. Deep learning

A neural network is a computational model inspired by the human brain, comprising interconnected nodes (neurons) that process and learn from data to perform tasks such as pattern recognition and decision-making. The field of deep learning revolutionised the world of data analysis in the last decades, leveraging biologically inspired ideas and lessons from non-linear optimization.

We started with a basecaller model that used a neural network developed by ONT, called Bonito. We tackled the energy efficiency issue that prevents this neural network deployment on commonly used embedded hardware using a Spiking approach.

### D. Spiking Neural Network

Spiking neural networks (SNNs) replicate the brain’s communication using discrete events called spikes. Spike neural networks (SNNs) improve energy efficiency thanks to two factors: sparse activity and event-driven processing. The sparse activity characteristic ensures that only a fraction of neural units are active at any given time, reducing computational demands and conserving energy. Event-driven processing dictates that SNNs perform computations exclusively in response to ’spike’ events, preventing continuous power consumption during idle periods.

### E. SLSTM vs LSTM

The LSTM is a kind of recurrent neural network that is specifically designed to handle sequential data, such as time series, speech and text. LSTM networks are capable of learning long-term dependencies in sequential data, which makes them well suited for tasks such as basecalling. These nets are made by an hidden state and a memory cell, which can hold information for an extended period of time and it is controlled by three gates: the input, the forget and the output gates. Those decides what information to add, to remove from and output from the cell. The SLSTM layers are the most used cells in the spiking architecture version of the Bonito basecaller. In practice they are a spiking long short-term memory cell. Hidden states are membrane potential and synaptic current which corresponds to the hidden and cell states $h, c$ in the original LSTM formulation.

## IV. Materials and methods

The goal of our study is to develop a translation of the state of the art Bonito architecture into its SNN version. To implement this conversion we employed the `snnTorch` library compatible with the `pytorch` framework. This library allowed us to simulate SNN components on classical Von Neumann architectures. We followed a straightforward approach to compare the impact of the conversion on the starting architecture: We evaluated an already trained bonito model, provided from the developer of [14], and we retrained the same network from scratch on the same datasets, in order to reproduce their results. Then we trained a SNN version of

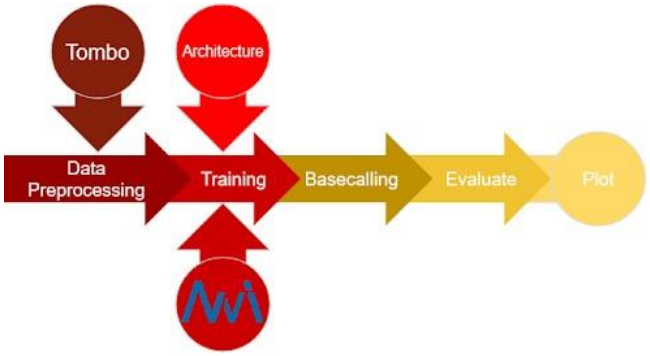


Fig. 3. A depiction of our pipeline, each phase is implemented through one or various commands

bonito, containing spiking LSTMs (**bonitosnn**) in the encoder part.

To train these models we started considering the same hyper-parameters used to instruct the original net:

- `window-size`: 2000
- `num-epochs`: 5
- `batch-size`: 64
- `starting-lr`: 0.001
- `warmup-steps`: 5000

We set the SLSTM threshold to 0.05, a value that proved feasible for training. In order to evaluate the impact of the translation of the LSTM layers into their SNN counterparts, we trained and tested the net with different configurations and number of SLSTM layers. We substituted one at the time each LSTM layer with the SNN version and evaluated the performance of the net. This protocol allowed us to inspect the existence of a trade-off between the SNN efficiency and the standard version performance results. Finally, to further explore the hyper-parameters space we took in consideration the employment of the NNI framework.

### A. Nanopore Benchmark

We've used as reference framework for both training and testing all our models the benchmark [14] and its related codebase. The benchmark includes three different dna datasets sources based on the R9.4 pore chemistry, along with a whole pipeline for basecallers benchmarking, on which we based the main branch of our pipeline, described in figure 3. In this way our models could also be compared with all the previously benchmarked models.

### B. Metrics

We introduce six fundamental evaluation metrics for assessing basecalling performance objectively and quantitatively across diverse conditions:

- **Failure Rate Analysis**: not all reads undergo successful basecalling. It is crucial to report the failure rate of the model. Failures can be categorised as follow:
  - **Not Basecalled**: the model does not produce any basecalls
  - **Not Aligned**: during the evaluation the alignment algorithm fails to produce an alignment.
  - **Short Alignment**: the alignment length is less than 50% of the reference's length
  - **Other**: various errors preventing the read from being evaluated are encountered.
- If the basecaller was able to align the read with its reference, we can distinguish four different types of events:
  - **Match rate**: a match happens when the base of the reading coincides with the base of the reference.
  - **Mismatch rate**: a mismatch happens when the base of the reading does not coincide with the base of the reference.
  - **Insertion rate**: an additional base is added compared to the target one.
  - **Deletion rate**: a deletion happens when the basecall process cannot identify a base that was present inside the target.
- **Homopolymer Assessment**: Homopolymers, which are repetitive sequences of the same base with a length of 5 or more, pose particular challenges in nanopore sequencing. Therefore, a dedicated metric is employed to evaluate the handling of homopolymers.
- **PhredQ Score**: The PhredQ scores reflect the confidence of the model in its base predictions. Ideally, incorrect calls should have low quality scores, while high-scoring bases should be predominantly correct. Calculating PhredQ scores involves assessing the predicted distribution of probabilities. In our study is computed as follow:

$$Q = -10 \log E$$

where $E$ is the error probability. To compute this metric we exploited the `fast_ctc_decode` ONT library.

### C. Proposed spiking architectures

The first architecture we tested, the aforementioned **bonitosnn** replace all the LSTM components in the encoder part, the one on the left in figure 1, with spiking equivalents, the SLSTM component of `snnTorch` library. In the following tests we show the behaviour of networks with different numbers of such spiking layers. This modification is the more straightforward step in translating the network to the spiking approach, as the spiking LSTM is a ready made component that work just as the conventional LSTM. Nonetheless, this modification delivers a substantial energy saving with comparable performances, as shown in section V.

Then we tested two others architectures that aim to convert the first convolutional part, the initial feature extractor, a shown in figure 4. The first, termed **SpikeLin**, is composed of 4 fully connected layers interleaved with Leaky neuron layers. It maintains the same shape of the output tensor as the original module, in order to be compatible with the following SLSTM encoder. The second, named **SpikeConv**, is obtained replacing the sigmoid linear unit activation functions with layers of Leaky neurons, preserving the convolutional layers as they are.

These architectures promises substantial improvements in energy efficiency but turned out to be way harder to train, as a matter of fact they could definitely be improved by testing different neuron types or through specific techniques for SNNs, such as population encoding.

### D. Hyperparameter Optimization

The first experiments on the bonito SNN architecture were carried out with the same hyper-parameters configuration of the pretrained benchmark model. The fact that this is the optimal set also for the SNN version cannot be taken as granted. For this reason, in our study we trained our network with different sets of hyper-parameters to assess the best configuration. To implement the search we didn't leverage the classical Grid Search algorithm due the large hyper-parameter space to explore, we used instead a different approach, exploiting the NNI framework.

NNI(Neural Network Intelligence) is an open source AutoML toolkit for hyperparameter optimization developed by Microsoft. This framework automatized the process of hyper-parameter tuning harnessing a tuning algorithm of choice. In our study we chose the simulated annealing algorithm which begins by sampling from the prior, but tends over time to sample from points closer and closer to the best ones observed. This algorithm is a simple variation of the random search that leverages smoothness in the response surface.

In order to run this toolkit we have to specify the limit of the hyper-parameters' space that the algorithm will explore. For each network version we defined a specific space, in particular we delineated a search space for the net with both the feature extractor and the decoder translated with SNN layers and one for the version with only the synaptic encoder. Precisely the two spaces differed only for one hyper-parameter: the threshold of the feature extractor spiking convolutional layers. Since that, for the sake of completeness only the second space is shown here:

- **batch size**: from 16 to 128,
- **starting learning rate**: from 0.0001 to 0.01,
- **SLSTM threshold**: from 0.01 to 0.2,
- **convolution threshold:** from 0.01 to 0.2.

### E. Training protocol

The training protocol is quite straightforward and follows the classic training of a neural network approach.

1) In the **preparation step** we downloaded the data from the nanopore banchmark links. Those are large datasets that can occupy up to 222GB in disk space. Before starting the training cycle two ulterior data processing are needed.
   - We annotated the raw data with the reference/true sequence, for this step we used the Nanopore tool Tombo. This script models the expected average raw signal level and aligns the expected signal with the measured signal. This remapping is referred to as "re-squiggle".
   - In the next step, we take the raw signal and splice it into segments of a fixed length so that they can be fed into the neural network.
2) **Model training**: In the model training phase we fed the training data we prepared and trained the models.
   In this step we trained the three different models: the classic Bonito architecture, bonitosnn (with *nlstm* as parameter that indicates how many LSTM layers we used in the decoder) or bonitospikeconv. All trainings use a cosine annealing learning rate schedule, the learning rate decrease following a cosine function but gets periodically restarted. They also use gradual warmup, the learning rate starts from a very small value and increase to the set starting value in the first 5000 training steps.
   All models have been trained on the cross-species task of the benchmark, using mostly bacterial DNA for training. This tasks is designed to evaluate the ability of the model to basecall many species and adapt to new species, never seen during training. Although most benchmarked models performed slightly worse on this task, we deemed it suitable to assess our models on a wide variety of possible applications.
3) (*optional*) **Parameters optimization**: This step is run only once for each model and it employed the NNI AutoML tool to explore the hyper parameter space and to pick the best configuration set (see IV-D for more details).
4) **Evaluation**: in this step we used the various metrics specified in Sub. IV-B.
5) **Report**: Finally a report is created with the evaluation of the trained model. This is done basecalling the test samples and computing the various metrics. This step generates a set of csv report files containing the different metrics results and it also generates different plots to graphically compare the different trainings.

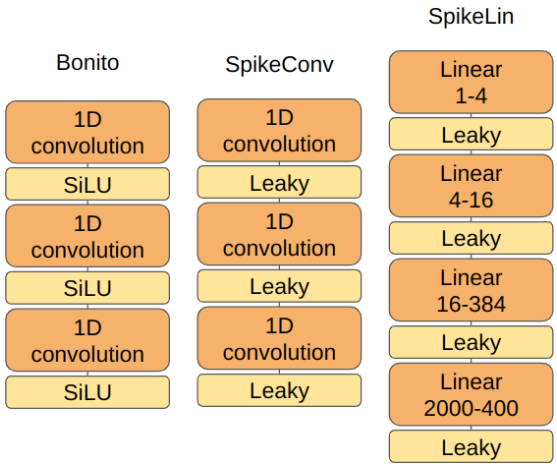


Fig. 4. The different feature extractor architectures proposed.The original one and our two spiking attempts

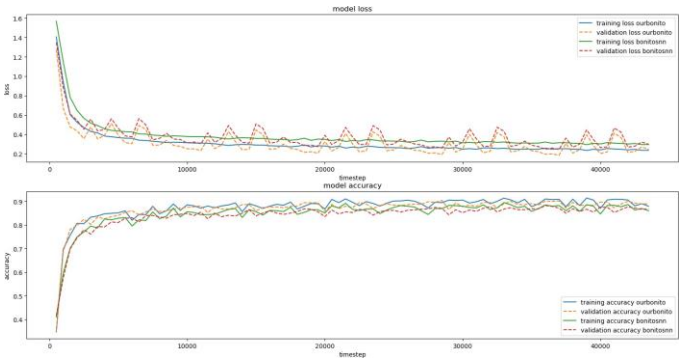


Fig. 5. Learning curves of the model we trained from scratch *ourbonito* and our spiking architecture *bonitosnn*

## V. Results and Discussion

TABLE I
Performance Comparison of Various Models in Terms of match rate and accuracy, including pre-trained Bonito and bonito with a number of SNN levels in the encoder ranging from 1 to 5

| Model | Match Rate | Accuracy |
|---|---|---|
| Pre-trained Bonito | 85.9% | - |
| Trained Bonito | 83.7% | 88.98% |
| FullSNN Bonito | 82% | 86.82% |
| Bonito 1LSTM4SNN | 82.2% | 88.04% |
| Bonito 2LSTM3SNN | 82.2% | 88.34% |
| Bonito 3LSTM2SNN | 82.90% | 89.27% |
| Bonito 4LSTM1SNN | 83.2% | 89.37% |

The experiments were executed in the HPC PoliTO, a free academic computing center, specifically we used the LEGION cluster with 24 nVidia Tesla V100 SXM2 -32GB - 5120 cuda cores. All the code is written in `Python` and the exploited frameworks are `pytorch` and `snnTorch` for the spiking transformation of the standard Bonito architecture.

First of all, we aimed to achieve a model evaluation match rate as close as possible to the one presented in the reference paper [14] employing the code of its repository [15]. The main factor hindering this effort was the publicly available dataset. Some of the dataset species, in fact, lacked a fasta format read reference file, only a fna file with the whole species DNA was provided. Training on such data would require an alignment operation that proved to be complex and would complicate the whole training process. So we decided to discard the incomplete data, losing around 13,4% of the train set and 31,6% of the test set, and use the still vast amount of remaining reads, 58251 out of the total 72368 of the benchmark cross-species task. The learning curve of the models 2 and 3 in table I are reported in figure 5.

One of the objectives of our study was to provide the opportunity to assess various amounts of SNN layers based on the user needs.

### A. Event Rates

In reference to the plots in figure 6, it can be observed that the model with the highest match rate is the one pre-trained provided by the repository [15], namely 'pretrainedbonito'

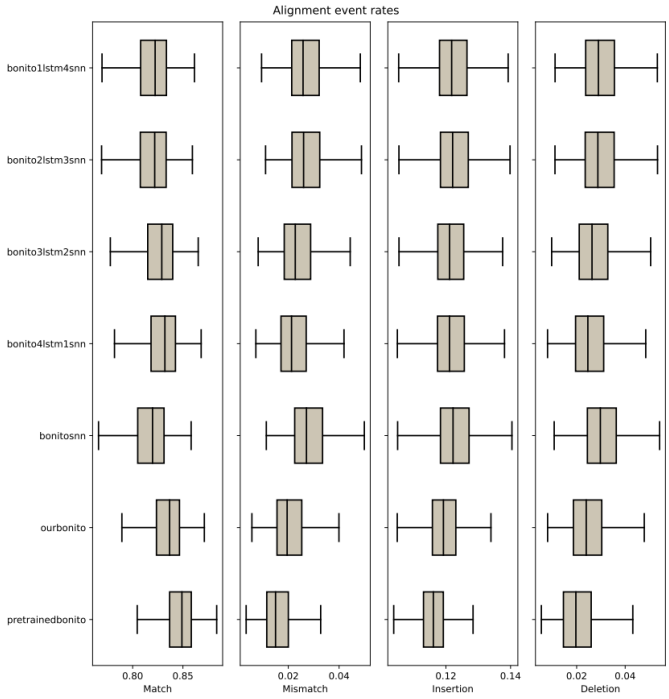


Fig. 6. Event rates distributions of the various models

with LSTM layers only (approximately 85%). Following this, the model trained in our project with the same architecture featuring only LSTM layers achieved a match rate of around 84%.

The match rate of the SNN-based networks exhibited an expected behaviour, wherein an increase in the number of SNN layers led to a decrease in the match rate. Despite the substitution of LSTM layers with SNN layers, the model's performance remained favourable, with no significant deviation in the match rate among the various models. Specifically, the match rate consistently ranged from 83% in the full SNN model to 85% in the full LSTM model.

These results are in accordance with the insertion rate and deletion rate, which both hover around 12% for the insertion rate and 2-4% for the deletion rate. Generally, there is a proportional relationship between the insertion/deletion rate and the number of SNN layers.

### B. Homopolymer rates

Homopolymer rates represent sequences of identical bases, which are particularly challenging to detect due to the absence of signal variations. Consequently, they were assessed separately. Among the evaluated models, the pre-trained model exhibited the best performances, followed by the fully LSTM-trained model, both achieving values around 20%. Subsequently, various SNN models were examined, displaying values ranging from 20% to 23%. Notably, there is a correlation between event rates and homopolymer rates, as the models performing well in event rates also excel in homopolymer rates. This consistency is observed from this perspective as well.

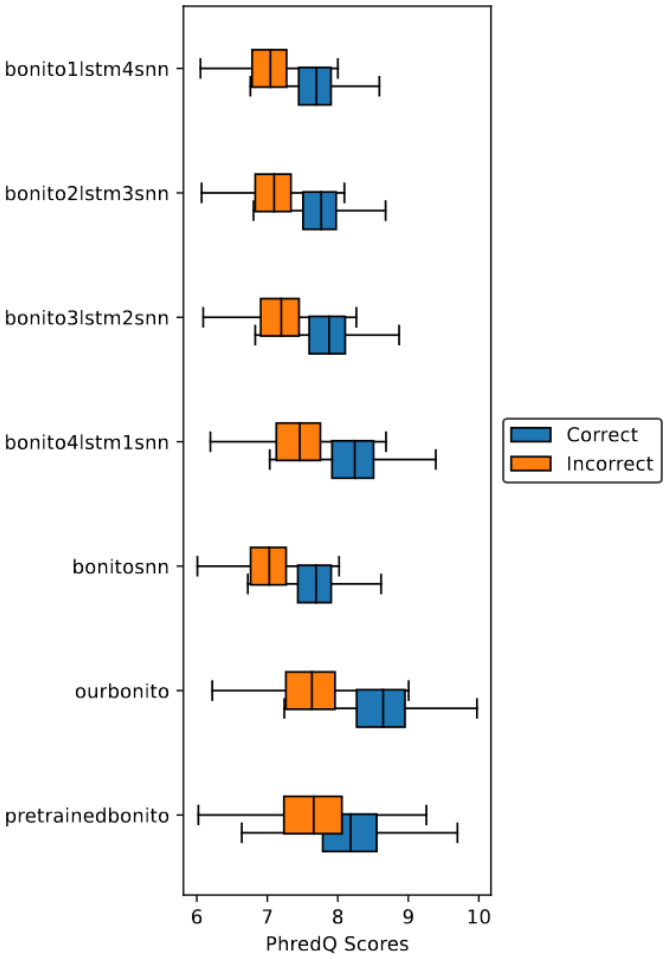


Fig. 7. Phredq scores distributions for correct and incorrect bases

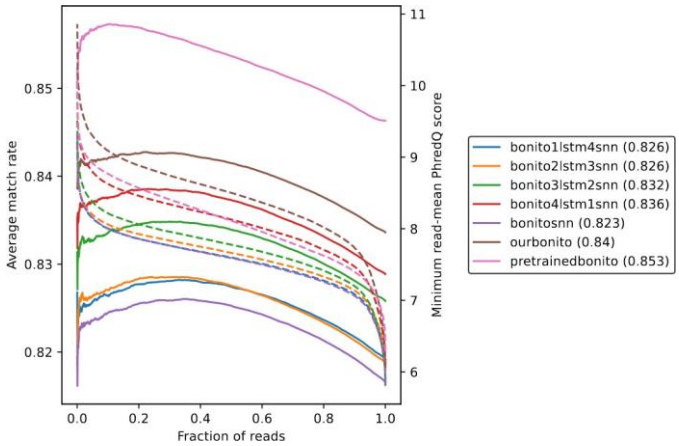


Fig. 8. AUC of match rate sorted by average read PhredQ score, comparison of the models with increasing numbers of spiking LSTMs

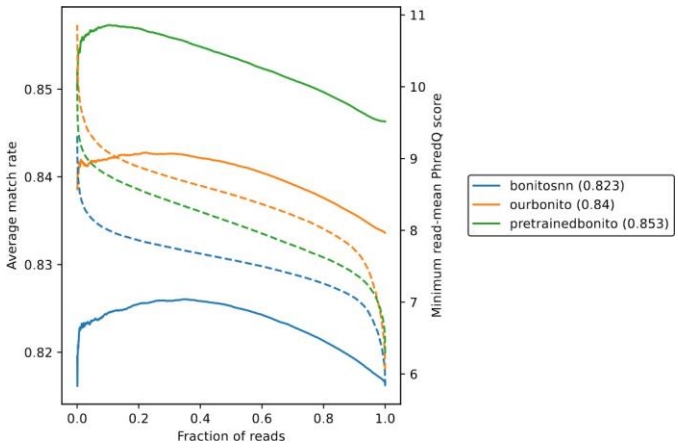


Fig. 9. AUC of match rate sorted by average read PhredQ score, comparison of the already trained model *pretrainedbonito*, the model we trained from scratch *ourbonito* and our spiking architecture *bonitosnn*

### *C. PhredQ scores*

In the following box plot graph 7 we can explore the PhredQ scores data distributions of both correctly and incorrectly called bases. We can see key elements as the median, quartiles or eventual anomalous values. Correct bases have higher scores than incorrect ones among all the considered models, with an overlap between the two distributions around 38% for the pre-trained model, 55% for the model trained by us, 63% for the model with all spiking LSTM, and values around 55-58% for all the models with both LSTM and spiking layers. We can conclude that PhredQ scores are indeed a good index of the overall model confidence.

### *D. AUC*

In order to investigate the trade-off between reads quality and accuracy we sorted the reads based on their PhredQ score and plot the average match rate against the fraction of accepted reads, as in figure 9. The AUC measure how well each model maximise this trade-off. We can see how the model we reproduced achieved slightly lower AUC (0.840) compared to the pre-trained one (0.853), maybe because of the aforementioned reference files problem.

On the other hand we can also note that the spiking model performed a bit worse (0.823) than the non spiking model trained from scratch. These values led us to conduct a more thorough study of the impact of spiking layers by replacing different numbers of final LSTMs to their spiking counterpart. As highlighted in figure 8 the less spiking layers we introduce, the higher performances, and closer to the non-spiking bonito, we obtain. We also searched for the optimal bonitosnn hyperparameters in order to better compare the models.

### *E. Hyperparameter optimization*

We performed a total of 5 NNI trials for the *BonitoSnn* architecture, each one performing a 5 epochs training. Each trial is quite time-consuming, taking from 10 to 12 hours, but we could not assume the best hyperparameter after one epoch to be optimal for the complete training too.

The results of all this trials are reported in table II, the respective learning curve are showcased in figure 10. The trials haven't really managed to improve on the default hyperparameters we had previously used, that already achieved good performances on all the metrics. To actually do so, longer anneal experiments and a wider hyperparameter space search would be needed.

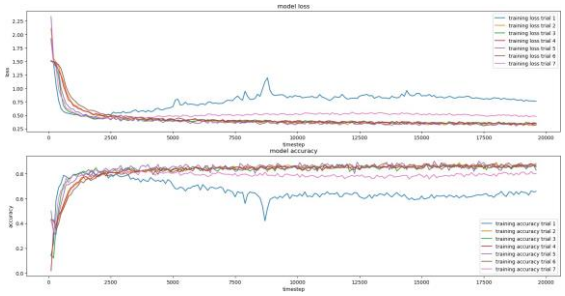


Fig. 10. Learning curves of all the *BonitoSnn* NNI trials

| *BonitoSnn* hyperparameter optimization trials | | | | |
|---|---|---|---|---|
| Batch size | Learning rate | SLSTM threshold | Train | Validation |
| 80 | $6,66 \cdot 10^{-4}$ | $2,98 \cdot 10^{-2}$ | 0, 6567 | 0, 6567 |
| 79 | $6,43 \cdot 10^{-3}$ | $9,36 \cdot 10^{-2}$ | 0, 8640 | 0, 8636 |
| 68 | $1,6 \cdot 10^{-3}$ | $9,87 \cdot 10^{-2}$ | 0, 8293 | 0, 8613 |
| 60 | $9,01 \cdot 10^{-4}$ | $8,67 \cdot 10^{-2}$ | 0, 8780 | 0, 8696 |
| 24 | $1,6 \cdot 10^{-4}$ | $5,54 \cdot 10^{-2}$ | 0, 8816 | 0, 8532 |
| 116 | $1,35 \cdot 10^{-3}$ | $2,51 \cdot 10^{-2}$ | 0, 8535 | 0.8583 |
| 103 | $4,48 \cdot 10^{-3}$ | $2,62 \cdot 10^{-2}$ | 0, 8006 | 0.8148 |

TABLE II
RESULTS OF THE *BonitoSnn* NNI TRIALS, WITH TRAINING AND VALIDATION ACCURACIES

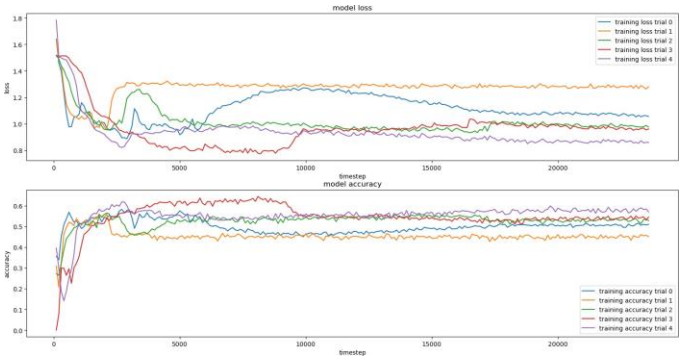


Fig. 11. Learning curves of all the *BonitoSpikeConv* NNI trials

| Estimated energy consumption comparison | | | | |
|---|---|---|---|---|
| Metric | Bonito | BonitoSnn | SpikeConv | SpikeLin |
| Accuracy | 0.840 | 0.823 | 0.536 | - |
| FLOPs ($x10^3$) | 107139.52 | 95329.6 | - | - |
| SOPs ($x10^3$) | - | 153.3 | 390.75 | 770.03 |
| Movidius energy ($\mu J$) | 80676.06 | 71783.19 | - | - |
| Loihi energy ($\mu J$) | - | 97.32 | 187.30 | 390.41 |

TABLE IV
ESTIMATED ENERGY CONSUMPTION OF ALL THE PROPOSED ARCHITECTURES

### *F. Feature Extractors*

The spiking feature extractors have been trained always coupled with all spiking encoders, their purpose is, after all, precisely obtaining a completely spiking architecture. The SpikeLin architecture always failed to even start learning, suggesting that the architecture is really unstable and easily falls in a state of either no spikes at all or always saturated. The SpikeConv feature extractor instead showed promising results during the NNI hyperparameter optimization, even if it has yet to reach performances comparable to the non-spiking bonito.

We performed 5 different NNI trials using the Anneal tuner, that refined the parameters during the course of the experiment. The best all-spiking architecture so far achieved an accuracy of 58,63%, as shown in table III, the respective learning curves are in figure 11.

### *G. Estimated energy footprint*

We estimated the energy it would take to run each model on a specialized hardware, Intel Movidius Neural Compute Stick 2 for non spiking modules, Intel Loihi for the spiking ones. Quantities reported in table IV are estimated using FLOPs for non spiking modules and SOPs, spiking operations, for the spiking layers. Energy estimations are based on the costs per operation estimated in [16] with a procedure equivalent to the one followed in [13]. We can see how the initial convolutional part is responsible for most of the operations, while the spiking components theoretically have drastically lower energy requirement. The most energy efficient architecture seems to be SpikeConv, although that may depend on the implementation of 1d convolutions over spikes, and it has yet to achieve acceptable performances. All considered, the BonitoSnn architecture save around 8000 $\mu J$ replacing the LSTMs, still achieving comparable performances on all the metrics.

| *BonitoSpikeConv* hyperparameter optimization trials | | | | |
|---|---|---|---|---|
| Batch size | learning rate | SLSTM thresh. | Leaky thresh. | Accuracy |
| 114 | $2,52 \cdot 10^{-3}$ | $8,93 \cdot 10^{-2}$ | $1,88 \cdot 10^{-1}$ | 0, 5085 |
| 95 | $1,05 \cdot 10^{-3}$ | $8,37 \cdot 10^{-2}$ | $1,18 \cdot 10^{-1}$ | 0, 5863 |
| 49 | $9,56 \cdot 10^{-4}$ | $1,00 \cdot 10^{-1}$ | $1,76 \cdot 10^{-1}$ | 0, 4492 |
| 118 | $2,68 \cdot 10^{-4}$ | $1,63 \cdot 10^{-1}$ | $7,13 \cdot 10^{-2}$ | 0, 5385 |
| 110 | $1,75 \cdot 10^{-3}$ | $1,41 \cdot 10^{-1}$ | $1,69 \cdot 10^{-1}$ | 0, 5355 |

TABLE III
RESULTS OF THE *BonitoSpikeConv* NNI TRIALS WITH DIFFERENT BATCH SIZES, LEARNING RATES, SLSTM AND LEAKY THRESHOLDS.

## VI. CONCLUSIONS AND FUTURE WORKS

In our studies, various types of datasets of different species were employed. The utilisation of diverse dataset types offers numerous advantages in research, such as robust validation, enhanced generalization, and comprehensive exploration. This approach ensure more reliable, valid, and applicable conclusions across a variety of scenarios. The same principle applies to the evaluation metrics; using diverse metrics provides a more comprehensive view of performance and facilitates better model comparison. The favorable performance of our model can be seen in the results obtained during experimentation. Our model consistently demonstrated behaviour in line with established expectations. As the SNN levels decreased, model performance increased without significantly compromising reliability.

The model presented in our study could be used as a foundation for future optimizations aimed at improving energy and addressing the issue of the fact the base Bonito model it's slow. There are significant improvement opportunities, an

improvement idea could be converting the feature extractor into an SNN as we achieved sub-optimal performance in this aspect. Therefore, a more in depth network design and a bespoke neural architecture search may be highly desirable to create a better-performing spiking equivalent. Additionally, further enhancements could be achieved by conducting hyper-parameter optimization over an extended time period to obtain more robust parameter values. Numerous avenue remain un-explored in this continually evolving field, warranting further investigation into these research directions.

## REFERENCES


[1] [Online]. Available: https://github.com/nanoporetech/bonito

[2] M. David, L. Dursi, D. Yao, P. Boutros, and J. Simpson, "Nanocall: an open source basecaller for oxford nanopore sequencing data," *Bioinformatics*, vol. 33, pp. 49–55, 2017.

[3] V. Boža, B. Brejová, and T. Vinař, "Deepnano: Deep recurrent neural networks for base calling in minion nanopore reads," *PLOS ONE*, vol. 12, no. 6, pp. 1–13, 06 2017. [Online]. Available: https://doi.org/10.1371/journal.pone.0178751

[4] H. Teng, M. D. Cao, M. B. Hall, T. Duarte, S. Wang, and L. J. M. Coin, "Chiron: translating nanopore raw signal directly into nucleotide sequence using deep learning," *GigaScience*, vol. 7, no. 5, p. giy037, 04 2018. [Online]. Available: https://doi.org/10.1093/gigascience/giy037

[5] e. a. J. Zeng, "Causalcall: Nanopore basecalling using a temporal convolutional network," *Frontiers in Genetics*, vol. 10, no. 1332, 2020.

[6] F. J. Rang, W. P. Kloosterman, and J. de Ridder, "From squiggle to basepair: computational approaches for improving nanopore sequencing read accuracy," *Genome Biol*, vol. 19, p. 90, 12 2018.

[7] Y. K. Wan, C. Hendra, P. N. Pratanwanich, and J. Göke, "Beyond sequencing: machine learning algorithms extract biology hidden in nanopore signal data," *Trends in Genetics*, vol. 38, p. 246–257, 03 2022.

[8] J. S.-R. e I. Holmes, "Sacall: A neural network basecaller for oxford nanopore sequencing data based on self-attention mechanism," *Genome Biology*, 1 2021.

[9] 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*, vol. 19, p. 614–623, 1 2022.

[10] V. Boža, P. Perešíni, B. Brejová, , and T. Vinař, "Dynamic pooling improves nanopore base calling accuracy." *arXiv*, 5 2021.

[11] Z. X. et al., "Fast-bonito: A faster deep learning based basecaller for nanopore sequencing," *Artificial Intelligence in the Life Sciences*, vol. 1, 12 2021.

[12] P. Perešíni, V. Boža, B. Brejová, and T. Vinař, "Nanopore base calling on the edge," *Bioinformatics*, vol. 37, no. 24, pp. 4661–4667, 07 2021. [Online]. Available: https://doi.org/10.1093/bioinformatics/btab528

[13] V. Fra, E. Forno, R. Pignari, T. Stewart, E. Macii, and G. Urgese, "Human activity recognition: suitability of a neuromorphic approach for on-edge aiot applications," *Neuromorphic Computing and Engineering*, vol. 2, 01 2022.

[14] d. R. J. Pagès-Gallego M, "Comprehensive benchmark and architectural analysis of deep learning models for nanopore sequencing basecalling." *Genome Biol.*, Apr 2023. [Online]. Available: https://doi.org/10.1186/s13059-023-02903-2

[15] [Online]. Available: "https://github.com/marcpaga/basecalling_architectures"

[16] P. Blouw and C. Eliasmith, "Event-driven signal processing with neuromorphic computing systems," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 8534–8538.