# A Framework for Designing Efficient Deep Learning-Based Genomic Basecallers

Gagandeep Singh[a] Mohammed Alser*[a] Alireza Khodamoradi*[b]
Kristof Denolf[b] Can Firtina[a] Meryem Banu Cavlak[a]
Henk Corporaal[c] Onur Mutlu[a]
[a]ETH Zürich [b]AMD [c]Eindhoven University of Technology

Nanopore sequencing is a widely-used high-throughput genome sequencing technology that can sequence long fragments of a genome. Nanopore sequencing generates noisy electrical signals that need to be converted into a standard string of DNA nucleotide bases (i.e., A, C, G, T) using a computational step called *basecalling*. The accuracy and speed of basecalling have critical implications for every subsequent step in genome analysis. Currently, basecallers are mainly based on deep learning techniques to provide high sequencing accuracy without considering the compute demands of such tools. We observe that state-of-the-art basecallers (i.e., Guppy, Bonito) are slow, inefficient, and memory-hungry as researchers have adapted deep learning models from other domains without specialization to the basecalling purpose. Our goal is to make basecalling highly efficient and fast by building the first framework for specializing and optimizing machine learning-based basecaller. We introduce `RUBICON`, a framework to develop hardware-optimized basecallers. `RUBICON` consists of two novel machine-learning techniques that are specifically designed for basecalling. First, we introduce the quantization-aware basecalling neural architecture search (`QABAS`) framework to specialize the basecalling neural network architecture for a given hardware acceleration platform while jointly exploring and finding the best bit-width precision for each neural network layer. Second, we develop `SkipClip`, the first technique to remove all the skip connections present in modern basecallers to greatly reduce resource and storage requirements without any loss in basecalling accuracy. We demonstrate the benefits of `QABAS` and `SkipClip` by developing `RUBICALL`, the first hardware-optimized basecaller that performs fast and accurate basecalling. Our experimental results on state-of-the-art computing systems show that `RUBICALL` is a fast, accurate and hardware-friendly, mixed-precision basecaller. Compared to a highly-accurate state-of-the-art basecaller, `RUBICALL` provides a 16.56× speedup without losing accuracy, while also achieving a 6.88× and 2.94× reduction in neural network model size and the number of parameters, respectively. Compared to the fastest state-of-the-art basecaller, `RUBICALL` provides a 3.19× speedup with 2.97% higher accuracy. We show that `QABAS` and `SkipClip` can help researchers develop hardware-optimized basecallers that are superior to expert-designed models.

## 1. Main

The rapid advancement of genomics and sequencing technologies continuously calls for the adjustment of existing algorithmic techniques or the development of entirely new computational methods across diverse biomedical domains [1–14]. Modern sequencing machines [15, 16] are capable of sequencing complex genomic structures and variants with high accuracy and throughput using long-read sequencing technology [17]. Oxford Nanopore Technologies (ONT) is the most widely used long-read sequencing technology [17–22]. ONT devices generate long genomic reads, each of which has a length ranging from a few hundred to a million base pairs or nucleotides, i.e., A, C, G, and T in the DNA alphabet [23–27].

*These authors contributed equally to this work.

ONT devices sequence a genome by measuring changes to an electrical signal as a single strand of DNA is passed through a nanoscale hole or *nanopore* [28]. The generated noisy electrical signal or *squiggle* is decoded into a sequence of nucleotides using a computationally-expensive step, called *basecalling* [19, 29–32]. Basecallers need to address two key challenges to accurately basecall a raw sequencing input. First, providing accurate predictions of each and every individual nucleotide, as the sensors measuring the changes in electrical current can only measure the effect of multiple neighboring nucleotides together [29]. Second, tolerating low signal-to-noise ratio (SNR) caused by thermal noise and the lack of statistically significant current signals triggered by DNA strand motions [30].

Modern basecallers use deep learning-based models to significantly (by at least 10%) improve the accuracy of predicting a nucleotide base from the squiggle compared to traditional non-deep learning-based basecallers [16–18, 33–36]. The success of deep learning in genome basecalling is attributed to the advances in its architecture to model and identify spatial features in raw input data to predict nucleotides. However, we observe the following five shortcomings with the current basecallers [33, 37–44]. First, current state-of-the-art basecallers are slow and show poor performance on state-of-the-art CPU and GPU-based systems, bottlenecking the entire genomic analyses. For example, `Guppy-fast`, the fastest state-of-the-art basecaller, takes ∼6 hours to basecall a 3 Gbps (Giga basepairs) human genome on a powerful server-grade GPU, while the subsequent step, i.e., read mapping, takes a fraction of basecalling time (∼0.11 hours using minimap2 [45]).

Second, since basecalling shares similarities with automatic-speech recognition (ASR) task, many researchers have directly adapted established ASR models, such as Quartznet [46], Citrinet [47], and Conformers [48], for basecalling without customizing the neural network architecture specifically for the basecalling problem. Such an approach might lead to higher basecalling accuracy but at the cost of large and unoptimized neural network architecture. For example, `Guppy` has ∼7-27 million neural network model parameters[1], and its research version, `Bonito`, even has ∼10 million model parameters. We show in Section 2.1.1 that we can eliminate up to 85% of the model parameters to achieve a 6.67× reduction in model size without any loss in basecalling accuracy. Therefore, current basecalling models are costly to run, and the inference latency becomes a major bottleneck.

Third, modern basecallers are typically composed of convolution layers with skip connections[2] [49] (allow reusing of activations from previous layers) that creates two major performance issues: (a) skip connections increase the data lifetime: the layers whose activations are reused in future layers must either wait for this reuse to occur before accepting new input or store the activations for later use by utilizing more memory. Thus, leading to high resource and storage requirements; and (b) skip connections often need to perform additional computation to match the channel size at the input of the non-consecutive layer, which increases the number of model parameters; e.g., `Bonito` requires ∼21.7% additional model parameters due to the skip connections.

Fourth, current basecallers use floating-point precision (32 bits) to represent each neural network layer present in a basecaller. This leads to high bandwidth and processing demands [50, 51]. Thus, current basecallers with floating-point arithmetic precision have inefficient hardware implementations. We observe in Section 2.1.2 that the arithmetic precision requirements of current basecallers can be reduced ∼4× by adjusting the precision for each neural network layer based on the target hardware and desired accuracy.

Fifth, basecallers that provide higher throughput have lower basecalling accuracy. For example, we show in Section 2.2 and Supplementary S3 that `Guppy-fast` provides up to 22.7× higher basecalling performance

[1]The parameters are weights of a neural network that are *learned* during neural network training.

[2]A skip connection allows to skip some of the layers in the neural network and feeds the output of one layer as the input to the next layers.

using 36.96× fewer model parameters at the expense of the 5.37% lower basecalling accuracy compared to more accurate basecallers.

These five problems concurrently make basecalling slow, inefficient, and memory-hungry, bottlenecking all genomic analyses that depend on it. Therefore, there is a need to reduce the computation and memory cost of basecalling while maintaining their performance. However, developing a basecaller that can provide fast runtime performance with high accuracy requires a deep understanding of genome sequencing, machine learning, and hardware design. At present, computational biologists spend significant time and effort to design and implement new basecallers by an extensive trial-and-error process.

**Our goal** is to overcome the above issues by developing a framework for specializing and optimizing a machine learning-based basecaller that provides high efficiency and performance for Nanopore sequencing machines.

To this end, we introduce `RUBICON`, the first framework for specializing and optimizing a machine learning-based basecaller. `RUBICON` uses two machine learning techniques to develop hardware-optimized basecallers that are specifically designed for basecalling. First, we propose `QABAS`, a quantization-aware basecalling architecture search framework to specialize basecaller architectures for hardware implementation while considering hardware performance metrics (e.g., latency, throughput, etc.). `QABAS` uses neural architecture search (NAS) [52] to evaluate millions of different basecaller architectures. During the basecaller neural architecture search, `QABAS` quantizes the neural network model by exploring and finding the best bit-width precision for each neural network layer, which largely reduces the memory and computational complexity of a basecaller. Adding quantization to the basecaller neural architecture search dramatically increases the model search space ($\sim 6.72 \times 10^{20}$ more viable options in our search space). However, jointly optimizing basecalling neural network architecture search and quantization allows us to develop accurate basecaller architectures that are optimized for hardware acceleration. Second, we develop `SkipClip` to remove all the skip connections presented in modern basecallers to reduce resource and storage requirements without any loss in basecalling accuracy. `SkipClip` performs a skip removal process using knowledge distillation [53], where we train a smaller network (*student*) without skip connections to mimic a pre-trained bigger network (*teacher*) with skip connections. We demonstrate the effectiveness of `QABAS` and `SkipClip` to develop `RUBICALL`, the first hardware-optimized basecaller that performs fast and accurate basecalling, outperforming the state-of-the-art basecallers.

**Key results.** We compare `RUBICALL` to three different basecallers. We demonstrate five key results. First, `RUBICALL` provides 16.56× higher basecalling throughput without any loss in basecalling accuracy compared to the most accurate state-of-the-art basecaller by leveraging mixed precision computation when implemented on a cutting-edge spatial vector computing system, i.e., the AMD-Xilinx Versal AI. Compared to the fastest basecaller, `RUBICALL` provides, on average, 2.97% higher basecalling accuracy with 3.19× higher basecalling throughput. Second, we show that `QABAS`-designed models are 5.74× smaller in size with 2.41× fewer neural network model parameters than the best state-of-the-art basecaller. Third, by further using our `SkipClip` approach, `RUBICALL` achieves a 6.88× and 2.94× reduction in neural network model size and number of parameters, respectively. Fourth, assemblies constructed using reads basecalled by `RUBICALL` lead to higher quality, more contiguous, and more complete assemblies for all evaluated species than that provided by other basecallers. Fifth, `RUBICALL` provides a 1.82%-26.49% lower number of base mismatches with the largest number of mapped based and mapped reads compared to the baseline basecaller. Our experimental results on state-of-the-art computing systems show that `RUBICALL` is a hardware-friendly, accurate, mixed-precision basecaller.

### 1.1. RUBICON

Figure 1 shows the key components of RUBICON. It consists of five modules. QABAS (❶) and SkipClip (❷) are two novel techniques that are specifically designed for specializing and optimizing machine learning-based basecallers. RUBICON provides support for Pruning (❸), which is a popular model compression technique where we discard network connections that are unimportant to neural network performance [54–57]. We integrate Training (❹) and Basecalling (❺) modules from the official ONT basecalling pipeline [58]. For both the Pruning and Training modules, we provide the capability to use knowledge distillation [53, 59] for faster convergence and to increase the accuracy of the target neural network.

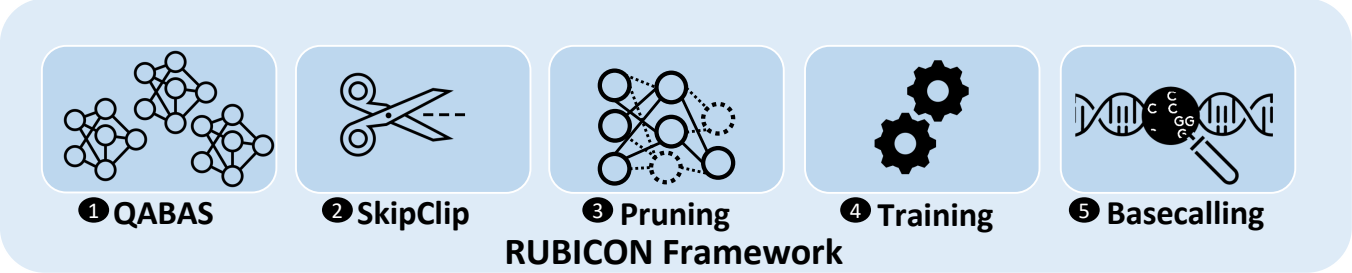


**Figure 1: Overview of RUBICON framework.**

#### 1.1.1. Quantization-Aware Basecaller Architecture Search (QABAS).

QABAS automates the process of finding hardware-aware genomics basecaller. Figure 2 shows the workflow overview of QABAS. The raw sequencing data ❶ is provided as input to QABAS, which can be obtained through sequencing a new sample, downloading from publicly-available databases, or computer simulation. QABAS uses such a set of data as training ($\mathbb{D}_{train}$) and evaluation set ($\mathbb{D}_{eval}$) while automatically designing a basecaller. To achieve a basecaller design that provides high throughput, we add hardware constraints ❷, in terms of latency or throughput, to QABAS. A hardware-aware basecaller can better use the underlying hardware features and greatly accelerate inference speed. As a result, it improves the overall basecalling efficiency.

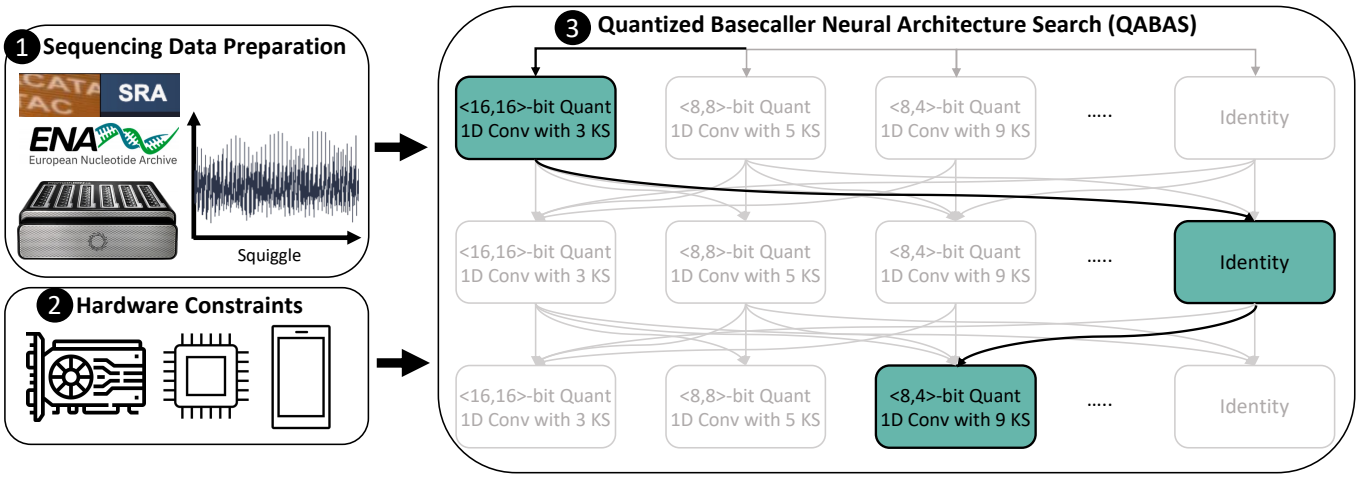


**Figure 2: Overview of QABAS. QABAS evaluates a different set of candidate operations for convolution (conv) and quantization bits. In the figure, we show different options for kernel size (KS) (e.g., 3, 5, 9, etc.) and quantization bits (4-b, 8-b, and 16-b) for each network layer. The identity operator removes a layer to get a shallower network.**

QABAS ❸ leverages automated machine learning (AutoML) algorithms [52] using neural architecture search (NAS) to design an efficient hardware basecaller by exploring and evaluating different neural

network architectures from a pre-defined search space. The search space $\mathcal{M}$ consists of the possible neural network architectural options while $\mathbb{M} \in \mathcal{M}$ is a sub-architecture from $\mathbb{M}$. The goal is to find an optimal sub-architecture $\mathbb{M}^*$ using Equation 1 that minimizes the training loss ($\mathcal{L}_{train}$) while going over $\mathbb{D}_{train}$ and gives maximum accuracy with the $\mathbb{D}_{eval}$.

$$\mathbb{M}^* = \arg\max_{\mathbb{M} \in \mathcal{M}} Eval(\mathbb{M}, \arg\min_{w^*} \mathcal{L}_{train}(w^*(\mathbb{M}), \mathbb{D}_{train}); \mathbb{D}_{eval}) \quad (1)$$

where $w^*(\mathbb{M})$ represent the weights of sub-architecture $\mathbb{M}^*$.

**QABAS Search Space.** We define the search space $\mathcal{M}$ as sufficiently large to enable a powerful neural architecture search. A larger space enables the search algorithm to cover more architectures so that the chance of finding a powerful architecture is increased. However, a larger search space makes the search algorithm more difficult to converge.

Our model search space has sequentially connected blocks, where each block receives input from its direct previous block. We formulate the NAS problem for hardware-aware genomics basecaller as finding: (a) the computational operations in each basic block [3] of a basecaller, including operations in a skip connection block, and (b) quantization bit-width for weights and activations for each neural network layer to perform low precision computation. Quantization is the reduction of the bit-width precision at which calculations are performed in a neural network to reduce memory and computational complexity. Adding quantization exploration dramatically increases the model search space ($\sim 6.72 \times 10^{20}$ additional viable options in our search space). However, performing a joint search for computational blocks and quantization bits is crucial because: (1) optimizing these two components in separate stages could lead to sub-optimal results as the best network architecture for the full-precision model is not necessarily the optimal one after quantization, and (2) independent exploration would also require considerable search time and energy consumption because of many viable design options [62]. Therefore, QABAS searches for both the computational operations present in each basic block of a basecaller and the quantization bits used by these computational operations.

**QABAS Seach Algorithm.** QABAS evaluates different neural network architectures using differentiable neural architecture search (DNAS) [63–65]. DNAS follows a weight-sharing approach of reusing weights of previously optimized architectures from the neural architecture search space. For example, if sub-architecture $\mathbb{M}_1$ has only one additional layer compared to sub-architecture $\mathbb{M}_2$. In such a scenario, $\mathbb{M}_1$ can use most weights from $\mathbb{M}_2$. Therefore, the search procedure gets accelerated in DNAS compared to training each sub-architecture individually.

DNAS formulates the entire search space as a super-network and distills a target network from this super-network. Traditional NAS approaches [52] often sample many different architectures from the search space and train each architecture from scratch to validate its performance. Such an approach requires heavy computational resources that could lead to thousands of GPU hours of overhead. One way to overcome this issue is to use NAS with heuristic-based methods [66, 67], such as genetic algorithms that select individual architectures from the current *population* to be *parents* and uses them to produce the *children* for the next generation. However, such methods still suffer from the problem of retraining each sample architecture from scratch. Therefore, DNAS provides an efficient solution by sharing computation among different architectures, as many of them have similar properties.

In QABAS, we construct an over-parameterized super-network with all possible candidate options. The super-network shares weights among sub-architecture. During the search phase, QABAS searches for the optimal: (a) architectural parameter $\alpha$: likelihood that a computational operation will be preserved in the final architecture; and (b) network weights $w$: weights of convolution layers. We use ProxylessNAS [68] to

[3] Our basic block consists of one-dimensional (1-D) convolution, batch normalization [60], and rectified linear unit (ReLU) [61].

binarize architectural parameter (i.e., $\alpha \in \{0,1\}$) to reduce memory consumption during the search phase. At the end of the search phase, the operators with the highest architectural weight are preserved, while others are eliminated. Since the NAS search procedure is focused on optimizing the super-network, the final sub-network architecture $\mathbb{M}^*$, with all the preserved operations, is retrained to convergence to fully optimize its network weights.

**Quantization-Aware Hardware Metric.** Current state-of-the-art basecallers [29, 30, 39–42, 44, 69] are hardware-agnostic. They only focus on improving the accuracy without paying attention to its inference efficiency. Therefore, these basecallers are over-provisioned with a large number of parameters and model size (see Section 2.1). We overcome this inefficiency in QABAS by adding hardware constraints, in terms of inference latency, to the QABAS search phase. Thus, QABAS aims to find an efficient neural network architecture for basecalling that is also optimized for hardware implementation. During the search process, QABAS sequentially selects a sub-network from the super-network. The expected latency of the sub-network is the sum of the latencies of each operation in the network. Before the start of the QABAS search phase, we profile the latencies of operations present in the search space on a targeted hardware to build a latency estimator. We also incorporate the latency while using different quantization bit-widths for the weights and activation in our latency estimator. This latency estimator is utilized to guide the QABAS search process.

QABAS's objective function ($\mathcal{L}_{\texttt{QABAS}}$) minimizes a joint cross-entropy error to: (a) provide better basecalling accuracy by minimizing the training loss ($\mathcal{L}_{train}$) while going over $\mathbb{D}_{train}$, and (b) minimize a regularization term ($\mathcal{L}_{reg}$) to find a sub-network $\mathbb{M}$ with inference latency ($\mathbb{L}_{\mathbb{M}}$) that satisfies our inference latency constraints. We add latency constraints by using a target latency parameter ($\mathbb{L}_{tar}$) to the regularization term $\mathcal{L}_{reg}$ to guide the search process. For example, in case if we want a small model, then we can provide a higher $\mathbb{L}_{tar}$ value, or vice versa.

$$\mathcal{L}_{\texttt{QABAS}} = \mathcal{L}_{train} + \lambda \mathcal{L}_{reg}$$
$$\mathcal{L}_{reg} = (\mathbb{L}_{\mathbb{M}} - \mathbb{L}_{tar})/\mathbb{L}_{tar}$$

where $\lambda$ is a parameter to control the tradeoff between the basecalling accuracy and the model latency.

#### 1.1.2. SkipClip: Skip Connection Removal by Teaching.

Deep neural networks often rely on skip connections or residual connections to improve training convergence [49]. Skip connections help mitigate the vanishing gradient problem [70] that occurs during the training stage. As a neural network trains under stochastic gradient descent (SGD) [71], it adjusts its trainable parameters (e.g., weights) via backpropagation [71]. Backpropagation passes the partial computations of one layer's gradient back to the previous layer, repeating this process for all layers via the chain rule [72]. However, as more layers are added to a neural model, the partial gradient computation may lead to extremely small (i.e., *vanishing*) values for the layers far back in the backpropagation path, preventing a deep neural model from converging. Skip connections address this issue by connecting one layer's output to a non-consecutive layer's input, providing a direct path for propagating the error through the layers so that the gradient no longer vanishes. In the forward path, skip connections provide an identity mapping of their input to their output, allowing deep neural networks to overcome the saturation problem (insignificant increase in accuracy while increasing the model's size and layers) [49].

Similarly, deep learning-based basecallers [29, 30, 39–42, 44, 69] incorporate skip connections to help mitigate the vanishing gradient and saturation problems. However, adding skip connections introduces the following three issues for hardware acceleration. First, skip connections increases the data-lifetime. The layers whose activations are reused in subsequent layers must wait for this activation reuse (or buffer the activations in memory) before accepting new input and continuing to compute. This leads to high

resource and storage requirements due to data duplication. Second, they introduce irregularity in neural network architecture as these connections span across non-adjacent layers. Third, skip connections require additional computation to adjust the channel size to match the channel size at the non-consecutive layer's input. Thus, increasing model parameters and model size. Therefore, networks without skip connections have more regular topologies that translate better to hardware acceleration. To this end, we propose `SkipClip`, a first skip connection remover for basecallers.

`SkipClip` performs a gradual skip removal process with knowledge distillation (KD) [53, 59]. KD is a model compression technique where a shallower model (*student*) learns to mimic a pre-trained bigger model (*teacher*) by transferring learned knowledge and label representation from the teacher to the student. As shown in Figure 3, `SkipClip` starts with a pretrained over-parameterized model as the teacher, which is not updated during the training of the student network. We use our final `QABAS` model as the student network. We achieve skip removal by letting the teacher teach the student to perform well on basecalling. At the start of every training epoch, `SkipClip` removes a skip connection from a block, starting from the input side, while performing KD. This is done until all skip connections are removed from the student network. `SkipClip` gets the best of both worlds: a highly accurate and topologically regular neural network without skip connections.

During the `SkipClip`, we perform a forward pass of both the student and the teacher model, while we perform a backward pass only for the student model to update its weights. The loss to update the student network's weight during the backward pass ($\mathcal{L}_{\texttt{SkipClip}}$) is calculated with Equation 2, where we use a weighing of the actual student loss ($\mathcal{L}_S$) and distillation loss ($\mathcal{L}_D$) using an alpha ($\alpha$) hyper-parameter. The student and the teacher model compute probabilities $f_T$ and $f_S$ for output labels (i.e., nucleotides A, C, G, T) in the forward pass, respectively. We use cross entropy ($\mathcal{L}_{CR}$) in the probability distributions to calculate the distillation loss ($\mathcal{L}_D$) as in Equation 3. The temperature ($\tau$) variable is used for *softening* the probability distributions, i.e., it controls the weight of knowledge from the teacher network for a student network to absorb. As we raise the $\tau$, the resulting soft label probability distribution becomes richer in information.

$$\mathcal{L}_{\texttt{SkipClip}} = \alpha\mathcal{L}_S - (1-\alpha)\mathcal{L}_D \tag{2}$$

$$where \mathcal{L}_D = \mathcal{L}_{CR}(f_T/\tau, f_S/\tau) \tag{3}$$

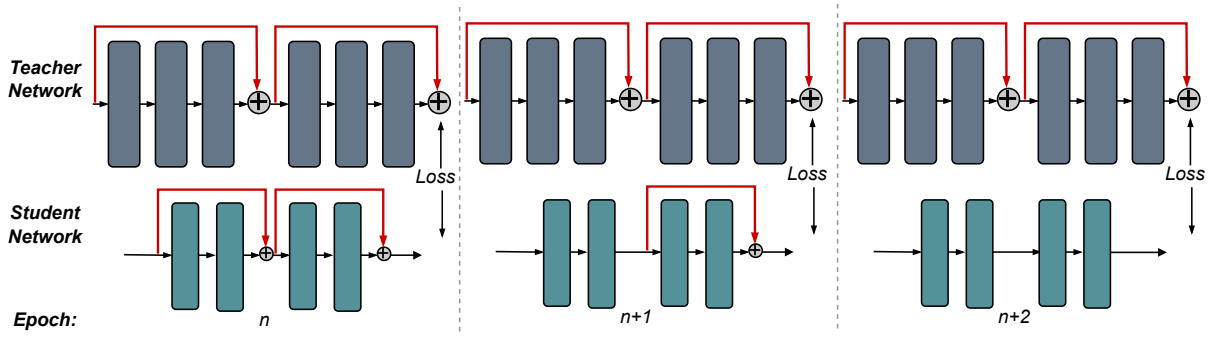


**Figure 3: Overview of `SkipClip` process for three epochs. We start with a large, overprovisioned floating-point precision model as the teacher network and our `QABAS` mixed-precision model as the student network. During the training, `SkipClip` removes a skip connection from the student network every `n` epoch, starting with the first skip connection encountered in the network from the input.**

### 1.1.3. Pruning: Pushing the Limits of a Basecaller.

Pruning is a model compression technique where we discard network connections that are unimportant to network performance without affecting the inference accuracy [54–57]. In a neural network, weights very

close to zero contribute very little to the model's inference. Performing convolution on such weights is equivalent to performing multiplication with zero [73]. Therefore, removing such weights could reduce redundant operations, in turn providing higher throughput and lower memory footprint both during the training and inference phase of a neural network.

Figure 4 shows two different pruning techniques that we use to evaluate the limits of a basecaller: (a) unstructured pruning, and (b) structured pruning. Unstructured or element pruning is a fine-grain way of pruning individual weights in a neural network without applying any structural constraints. This approach leads to the highest model compression [74]. However, this method presents a major problem that the non-pruned weights have a very sparse structure that is unsuitable for acceleration on any hardware platform. Therefore, unstructured pruning does not improve the actual inference run-time performance of a network.

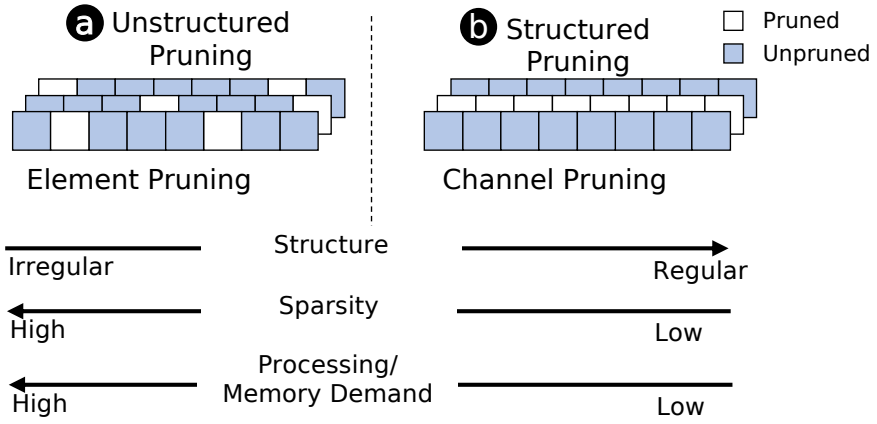


**Figure 4: Two different pruning techniques applied to 1-dimensional convolution.**

A structured way of pruning weights from a neural network is more amenable to hardware acceleration. In structure pruning, we remove a larger set of weights while maintaining a dense structure of the model [75, 76]. Hence, this leads to a network with fewer parameters that require not only fewer computations but also less memory overhead during the runtime by generating lighter intermediate representation. However, structured pruning dramatically affects the structure of subsequent layers present after the pruned layer. Therefore, structured pruning is a coarse-grain pruning method where higher amount of sparsity can have drastic effect on the model accuracy.

#### 1.1.4. `RUBICALL` Architecture.

Figure 5 shows the architecture of `RUBICALL`. We develop `RUBICALL` using `QABAS` and `SkipClip`. The `RUBICALL` architecture is composed of 28 quantized convolution blocks containing ~3.3 million model parameters. Each block consists of quantized grouped 1-dimensional convolution and quantized pointwise 1-dimensional convolution where every layer is quantized to a different domain. The convolution operation is followed by batch normalization (Batch Norm) [60] and a quantized rectified linear unit (ReLU) [61] activation function. The final output is passed through a connectionist temporal classification (CTC) [77] layer to produce the decoded sequence of nucleotides (A, C, G, T). CTC is used to provide the correct alignment between the input and the output sequence.

In a learning task, $\mathcal{X}$ represents feature space with label $\mathcal{Y}$, where a machine learning model is responsible for estimating a function $f\colon \mathcal{X} \rightarrow \mathcal{Y}$. `RUBICALL` first splits a long read in electrical-signal format (e.g., millions of signals) into multiple smaller chunks (e.g., thousands of samples per chunk) and then basecalls these chunks. `RUBICALL` uses the input signal (or squiggle) as $\mathcal{X}$ to predict nucleotides as label $\mathcal{Y}$. The

CTC layer assigns a probability for all possible labels in $\mathcal{Y}$ given an $\mathcal{X}$ at each time-step. The nucleotide with the highest probability is selected as the final output.

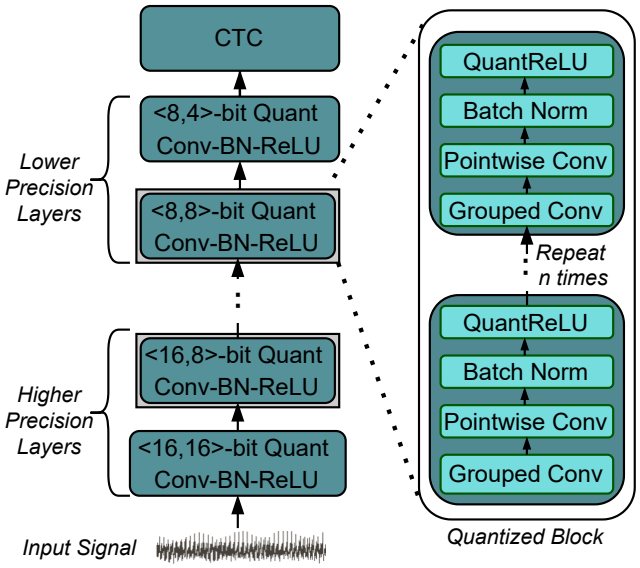


**Figure 5: Overview of `RUBICALL` architecture. The normalized input signal is passed through a succession of quantized convolution blocks. Each block is composed of several processing steps (convolution, batch normalization, and activation). We represent the quantization as a tuple <weight, activation>. Initial layers use a higher precision for weights and activation, while the final layers use a lower precision. The final output is passed through a connectionist temporal classification (CTC) to produce the decoded sequence of nucleotides.**

## 2. Results

### 2.1. Analyzing the State-of-the-Art Basecaller

We observe established automatic-speech recognition (ASR) models being directly applied to basecalling without optimizing it for basecalling. Such an approach leads to large and unoptimized basecaller architectures. We evaluate the effect of using two popular model compression techniques on the `Bonito` basecaller: (1) Pruning, and (2) Quantization.

**2.1.1. Effect of Pruning.** We show the effect of pruning `Bonito` on the validation accuracy and model size in Figure 6(a) and Figure 6(b), respectively. We use unstructured element pruning and structured channel pruning with different degrees of sparsity.

We make three major observations. First, pruning up to 85% of the `Bonito` model weights using unstructured pruning reduces the model size by 6.67× while maintaining the same accuracy as the baseline, unpruned `Bonito` model. While pruning 30-40% of the `Bonito` model filters, using structured pruning reduces the model size by 1.46-1.66× while maintaining the same accuracy of the baseline, unpruned `Bonito` model. Such a high pruning ratio shows that most of the weights are redundant and do not contribute to the actual accuracy. Second, after pruning 97% (60%) of the model weights, `Bonito` provides 81.20% (72.66%) basecalling accuracy while using 33.33× (2.62×) smaller model using unstructured pruning (structured pruning). Third, the *knee point*[4] for unstructured pruning and structured pruning is at 98% and 60% where `Bonito` provides 65.14% and 72.66% of basecalling accuracy, respectively. Beyond the knee-point, `Bonito` losses its complete prediction power. We conclude that `Bonito` is over-parameterized and contains redundant logic and features.

[4]We define knee point as the point beyond which a basecaller is unable to basecall at an acceptable level of accuracy.

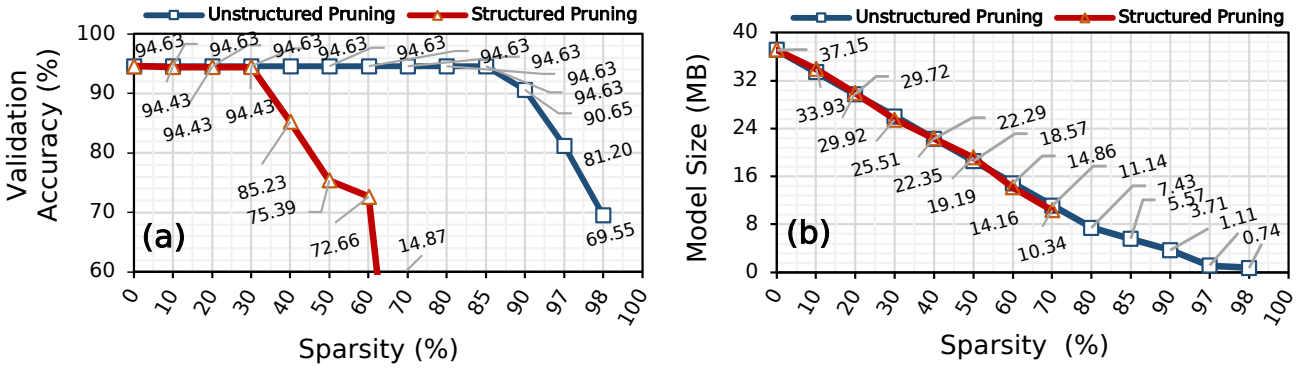


**Figure 6: Effect of pruning the elements and channels of `Bonito` using unstructured and structured pruning, respectively, on: (a) validation accuracy and (b) model size.**

**2.1.2. Effect of Quantization.** Figure 7 shows the effect of using a quantized model to basecall on the basecalling accuracy for four different species. In Figure 8, we show the effect of quantization on the model size. We quantize both the weight and activation using six different bit-width configurations (<3,2>, <4,2>, <4,4>, <4,8>, <8,4>, and <16,16>). We also show the results with the default floating-point precision (<fp32,fp32>). We use static quantization that uses the same precision for each neural network layer.

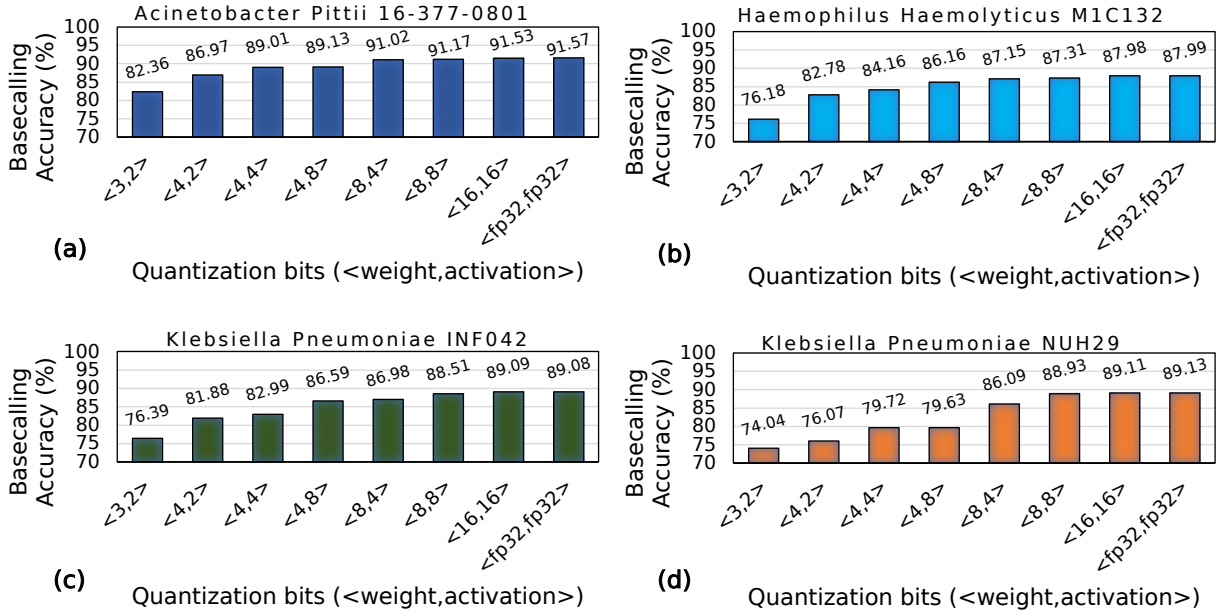


**Figure 7: Basecalling using quantized models.**

We make four main observations. First, using a precision of <8,8> for weight and activation for all the layers of `Bonito` causes a negligible accuracy loss (0.18%-0.67%), while reducing the model size by 4.03×. Second, `Bonito` is more sensitive to weight precision than activation precision. For example, we observe a loss of 1.82%-9.48% accuracy when using a precision of <4,8> instead of <16,16> bits compared to an accuracy loss of only 0.51%-3.02% when using a precision of <8,4> instead of <16,16> bits. Third, we observe a significant drop in accuracy (by 9.17%-15.07%), when using less than 4 bits for weights (e.g., using <3,2> configuration). Fourth, using bit-width precision of <16,16> bits provides ∼2× reductions in model size and without any accuracy loss compared to using full precision (<fp32,fp32>) floating-point implementation. We conclude that the current state-of-the-art basecaller, `Bonito`, can still efficiently

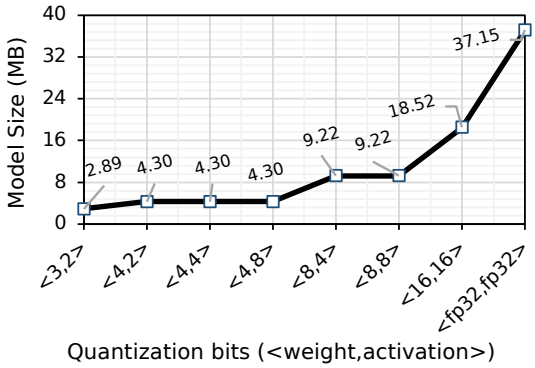


**Figure 8: Effect of quantizing weight and activation of `Bonito` on model size. We quantize both the weight and activation with static precision. Since weights are the trainable parameters in a neural network, only weights contribute to the final model size.**

perform basecalling even when using lower precision for both the weight and activation.

### 2.2. `RUBICALL`: Overall Trend

We compare the overall basecalling performance of `RUBICALL` with that of the baseline basecallers in terms of basecalling accuracy, model parameters, and model size in Figure 9(a), 9(b), and 9(c), respectively.

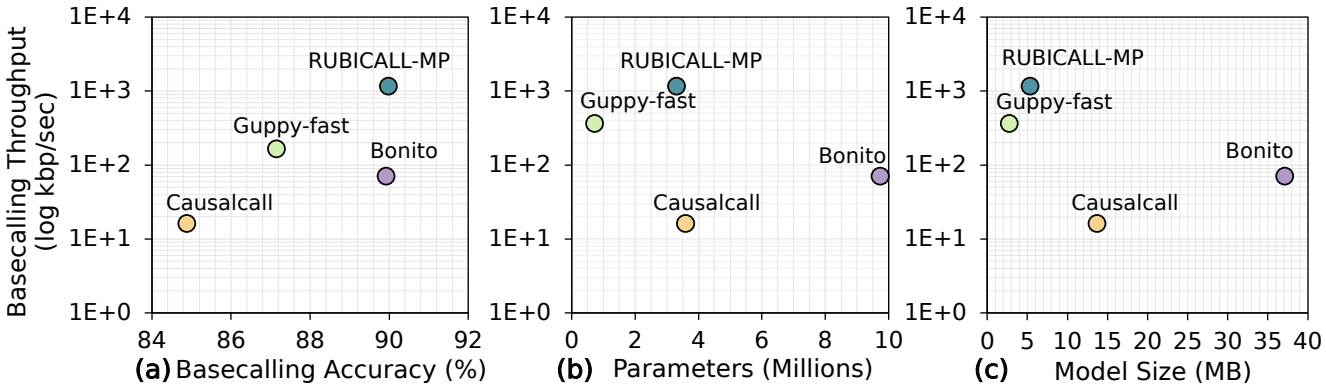


**Figure 9: Comparison of average basecalling throughput for `RUBICALL-MP` with baseline basecaller in terms of: (a) average basecalling accuracy, (b) model parameters, and (b) model size.**

We make six key observations. First, compared to `Bonito`, which is the most accurate basecaller, `RUBICALL-MP` provides 16.56× higher basecalling throughput without any loss in accuracy. This is because `RUBICALL-MP` has a mixed precision neural architecture that can exploit low-precision compute units present on our evaluated spatial vector computing systems, i.e., the AMD-Xilinx Versal AI Engine (AIE). Compared to `Guppy-fast`, which is the fastest basecaller, `RUBICALL-MP` provides, on average, 2.97% higher accuracy with 3.19× higher basecalling throughput. Therefore, `RUBICALL-MP` provides both accuracy and high basecalling throughput compared to the baseline basecallers. Second, `Bonito` has the highest number of neural network model parameters, which are 2.71×, 13.33×, and 2.94× more than `Causalcall`, `Guppy-fast`, and `RUBICALL-MP`, respectively. Third, `Causalcall` (`Bonito`) has 2.71× fewer (2.93× greater) neural network model parameters but is 72.55× (16.56×) slower in performance when compared to `RUBICALL-MP`. This is because skip connections in `Causalcall` and `Bonito` create a performance bottleneck by increasing the data lifetime and adding additional computation. Therefore, we observe a performance degradation instead of higher performance due to the linear relation usually present between the number of model parameters and the speed of a model. Fourth, `Guppy-fast` has the lowest number of trainable model parameters that are 4.92×, 13.33×, and 4.54× lower than `Causalcall`, `Bonito`, and `RUBICALL-MP`. Fifth, `RUBICALL-MP` provides 2.55× and 6.93× smaller model size compared to `Causalcall` and `Bonito`, respectively. The decrease in model size is due to: (1) a lower number of neural network layers; and (2)

optimum bit-width precision for each neural network layer. Sixth, all the baseline basecallers use floating-point arithmetic precision for all neural network layers. This leads to very high memory bandwidth and processing demands. We conclude that RUBICALL-MP provides the ability to basecall accurately, quickly, and efficiently scale basecalling by providing a reduction in both model size and neural network model parameters.

### 2.3. Performance Comparison

We compare the speed of RUBICALL-MP against baseline basecallers in Figure 10. We make three major observations. First, RUBICALL-MP consistently outperforms all the other basecallers for all the evaluated species. RUBICALL-MP improves average performance by 72.55×, 3.19×, and 16.56× over Causalcall, Guppy-fast, and Bonito, respectively. Second, by leveraging low precision capabilities, RUBICALL-MP provides 7.45× higher performance when compared to its floating-point implementation (RUBICALL-FP). Third, RUBICALL-FP, by using floating-point precision, provides 9.72×, 0.43×, and 2.22× performance compared to Causalcall, Guppy-fast, and Bonito, respectively. We conclude that using mixed-precision computation, RUBICALL-MP consistently performs better than the baseline basecallers.

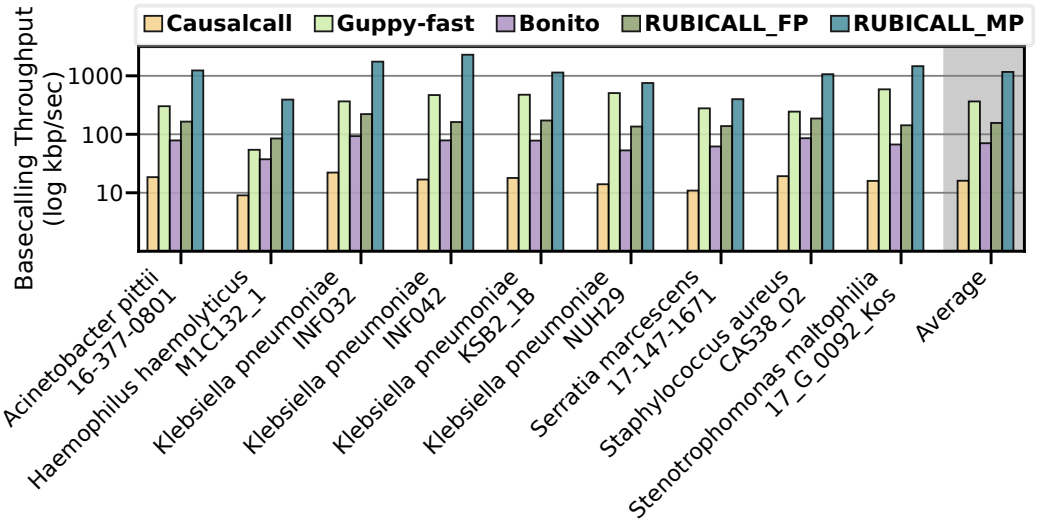


**Figure 10: Performance comparison of RUBICALL (using floating-point precision (RUBICALL-FP) and mixed-precision (RUBICALL-MP)) and three state-of-the-art basecallers.The y-axis is on a logarithmic scale.**

### 2.4. Basecalling Accuracy

We compare the basecalling accuracy of RUBICALL against baseline basecallers in Figure 11. We make three major observations. First, RUBICALL provides 6.54% and 0.57% higher accuracy than CNN-based basecaller Causalcall and Bonito, respectively. Compared to a state-of-the-art RNN-based basecaller, Guppy-fast, RUBICALL achieves 2.97% higher accuracy. Second, Bonito has 2.93× higher parameters (Figure 9(a)) but is 0.57% less accurate than RUBICALL. Third, Causalcall is unable to align half of *Haemophilus haemolyticus M1C132_1* reads to its reference. Therefore, it is deemed unaligned and cannot be used to determine its read accuracy. We conclude that RUBICALL provides the highest accuracy compared to other basecallers.

### 2.5. Downstream Analysis

**2.5.1. De Novo Assembly.** We provide the statistics related to the accuracy, completeness, and contiguity of assemblies we generate using the basecalled reads from Causalcall, Guppy-fast, Bonito, and RUBICALL in Table 1.

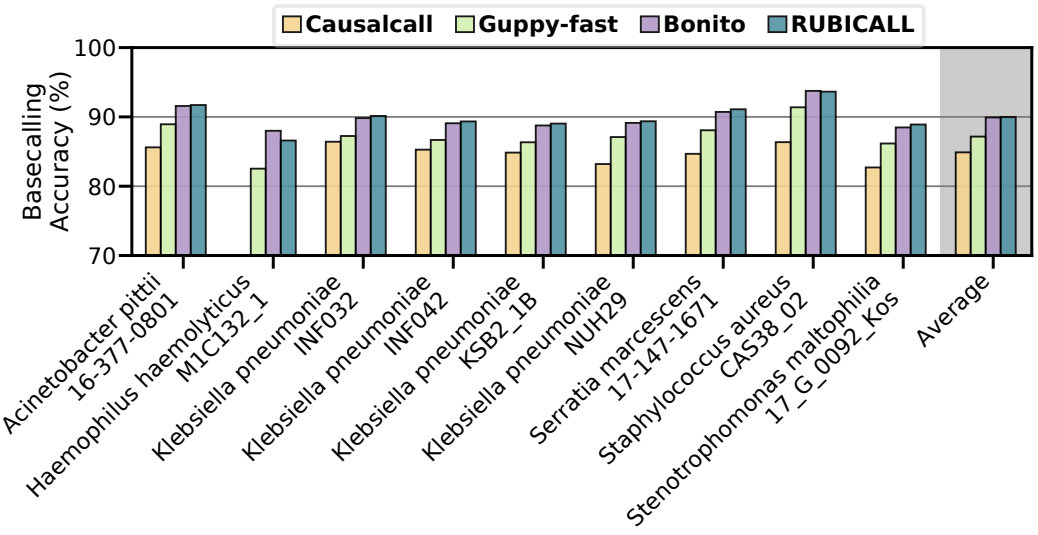


**Figure 11: Basecalling accuracy comparison of `RUBICALL`.**

We make four key observations. First, assemblies constructed using reads basecalled by `RUBICALL` provide the best reference genome coverage for *all* datasets ("Genome Fraction" in Table 1). This means that assemblies built using `RUBICALL`-basecalled reads are more complete than assemblies built using reads from other basecallers since a larger portion of the corresponding reference genomes align to their assemblies using `RUBICALL`-basecalled reads compared to that of using reads from other basecallers. Second, assemblies constructed using the `RUBICALL` reads usually have a higher average identity than that of `Causalcall`, `Guppy-fast`, and `Bonito`. These average identity results are tightly in line with the basecalling accuracy results we show in Figure 11. Although `Guppy-fast` provides a higher average identity for the Haemophilus haemolyticus M1C132_1 dataset (i.e., 91.51%), the genome coverage provided by `Guppy-fast` is 2.2% lower than that provided by `RUBICALL` for the same dataset. This means a large portion of the assembly provided by `Guppy-fast` has low-quality regions as the reference genome cannot align to these regions due to high dissimilarity. Third, assemblies constructed using the `RUBICALL` reads provide better completeness and contiguity as they have 1) assembly lengths closer to their corresponding reference genomes and 2) higher NG50 results in most cases than those constructed using the `Guppy-fast` and `Bonito` reads. Fourth, although `Causalcall` usually provides the best results in terms of the assembly lengths and NG50 results, we suspect that these high NG50 and assembly length results are caused due to highly repetitive and inaccurate regions in these assemblies due to their poor genome fraction and average GC content results. The average GC content of the assemblies constructed using the `Causalcall` reads is significantly distant from the GC content of their corresponding reference genomes in most cases. This poor genome fraction and average GC content results suggest that such large NG50 and assembly length values from `Causalcall` may also be caused by poorly basecalled reads that lead to unresolved repetitive regions (i.e., bubbles in genome assembly graphs) or a strong bias toward certain error types (i.e., homopolymer insertions of a certain base) in the assembly [78, 79].

We conclude that, in most cases, the reads basecalled by `RUBICALL` lead to higher quality, more contiguous, and more complete assemblies than that provided by other state-of-the-art basecallers, `Causalcall`, `Guppy-fast`, and `Bonito`.

**2.5.2. Read Mapping.** We provide the comparison of `RUBICALL` with `Causalcall`, `Guppy-fast`, and `Bonito` in terms of the total number of base mismatches, the total number of mapped bases, the total

**Table 1: Assembly quality comparison of the evaluated basecallers for different species. We measure assembly accuracy in terms of genome fraction (Genome Fraction (%)) and average identity (Average Identity (%)). Genome fraction is the portion of the reference genome that can align to a given assembly, while average identity is the average of the identity of assemblies when compared to their respective reference genomes. We measure statistics related to the contiguity and completeness of the assemblies in terms of the overall assembly length (Assembly Length), Average GC content (Average GC (%)) (i.e., the ratio of G and C bases in an assembly), and NG50 statistics (NG50) (i.e., shortest contig at the half of the overall reference genome length).**

| Dataset | Basecaller | Genome Fraction (%) | Average Identity (%) | Assembly Length | Average GC (%) | NG50 |
|---|---|---|---|---|---|---|
| Acinetobacter pittii 16-377-0801 | Causalcall | 92.45 | 86.18 | **3,826,077** | 42.23 | **3,826,077** |
| | Guppy-fast | 94.67 | 89.29 | 3,628,317 | **38.82** | 3,628,317 |
| | Bonito | **96.87** | 91.44 | 3,676,821 | 38.90 | 3,676,821 |
| | RUBICALL | **96.87** | **91.51** | 3,694,086 | 38.97 | 3,694,086 |
| | Reference | 100 | 100 | 3,814,719 | 38.78 | 3,814,719 |
| Haemophilus haemolyticus M1C132_1 | Causalcall | NA | NA | NA | NA | NA |
| | Guppy-fast | 94.67 | **91.51** | **2,046,024** | 37.98 | **2,046,024** |
| | Bonito | **96.87** | 90.70 | 1,957,480 | **38.87** | 1,957,480 |
| | RUBICALL | **96.87** | 90.54 | 1,966,781 | 38.92 | 1,966,781 |
| | Reference | 100 | 100 | 2,042,591 | 38.56 | 2,042,591 |
| Klebsiella pneumoniae INF032 | Causalcall | 92.45 | 87.35 | **4,959,127** | 56.9 | **4,959,127** |
| | Guppy-fast | 90.50 | 87.53 | 4,761,297 | **57.19** | 4,761,297 |
| | Bonito | 94.50 | 90.20 | 4,897,352 | 56.65 | 4,897,352 |
| | RUBICALL | **94.51** | **90.30** | 4,924,240 | 56.85 | 4,924,240 |
| | Reference | 100 | 100 | 5,111,537 | 57.63 | 5,111,537 |
| Klebsiella pneumoniae INF042 | Causalcall | 91.44 | 87.36 | **5,288,166** | **56.94** | **5,288,166** |
| | Guppy-fast | 91.30 | 88.49 | 5,052,889 | 56.8 | 5,052,889 |
| | Bonito | **93.12** | 90.49 | 5,111,083 | 56.61 | 5,111,083 |
| | RUBICALL | **93.12** | **90.60** | 5,146,050 | 56.72 | 5,146,050 |
| | Reference | 100 | 100 | 5,337,491 | 57.41 | 5,337,491 |
| Klebsiella pneumoniae KSB2_1B | Causalcall | 91.58 | 86.97 | **5,175,311** | **57.09** | **5,175,311** |
| | Guppy-fast | 88.55 | 88.00 | 4,932,626 | 56.71 | 4,932,626 |
| | Bonito | **93.07** | **90.11** | 5,003,377 | 56.69 | 5,003,377 |
| | RUBICALL | **93.07** | 89.89 | 5,023,639 | 56.75 | 5,023,639 |
| | Reference | 100 | 100 | 5,228,889 | 57.59 | 5,228,889 |
| Klebsiella pneumoniae NUH29 | Causalcall | 89.08 | 86.01 | **5,158,874** | 56.78 | **5,158,874** |
| | Guppy-fast | 91.60 | 89.34 | 4,942,833 | 57.01 | 4,942,833 |
| | Bonito | **94.36** | 90.26 | 4,918,147 | 57.04 | 4,918,147 |
| | RUBICALL | **94.36** | **90.43** | 4,940,813 | **57.18** | 4,940,813 |
| | Reference | 100 | 100 | 5,134,281 | 57.61 | 5,134,281 |
| Serratia marcescens 17-147-1671 | Causalcall | 89.91 | 86.23 | **5,532,953** | 57.86 | **5,422,052** |
| | Guppy-fast | 96.31 | 89.56 | 5,479,812 | **58.85** | 5,282,474 |
| | Bonito | **96.76** | 91.38 | 5,534,329 | 58.41 | 5,316,651 |
| | RUBICALL | **96.76** | **91.59** | 5,597,251 | 58.52 | 5,346,640 |
| | Reference | 100 | 100 | 5,517,578 | 59.13 | 5,517,578 |
| Staphylococcus aureus CAS38_02 | Causalcall | 94.35 | 87.29 | 2,849,123 | 36.59 | 2,810,038 |
| | Guppy-fast | 95.10 | 91.49 | 2,790,895 | 33.05 | 2,752,169 |
| | Bonito | **97.03** | **93.57** | 2,858,986 | **32.86** | 2,819,356 |
| | RUBICALL | **97.03** | 93.36 | **2,860,885** | 33.24 | **2,821,276** |
| | Reference | 100 | 100 | 2,902,076 | 32.82 | 2,902,076 |
| Stenotrophomonas maltophilia 17_G_0092_Kos | Causalcall | 94.85 | 85.73 | **4,823,177** | 63.66 | **4,823,177** |
| | Guppy-fast | 94.89 | 89.74 | 4,596,898 | **65.5** | 4,596,898 |
| | Bonito | 95.42 | 90.14 | 4,664,226 | 64.82 | 4,664,226 |
| | RUBICALL | **95.46** | **90.49** | 4,693,744 | 65.03 | 4,693,744 |
| | Reference | 100 | 100 | 4,802,733 | 66.28 | 4,802,733 |

number of mapped reads, and the total number of unmapped reads in Figure 12(a), 12(b), 12(c), and 12(d), respectively.

We make four key observations. First, RUBICALL provides the lowest number of base mismatches, which are 26.49%, 21.35%, and 1.82% lower compared to Causalcall, Guppy-fast, and Bonito, respectively. This indicates that RUBICALL provides more accurate basecalled reads that share large similarity with the reference genome. This is in line with the fact that RUBICALL provides the highest basecalling accuracy, as we evaluate in Section 2.4. Second, RUBICALL provides, on average, the highest number of mapped bases,

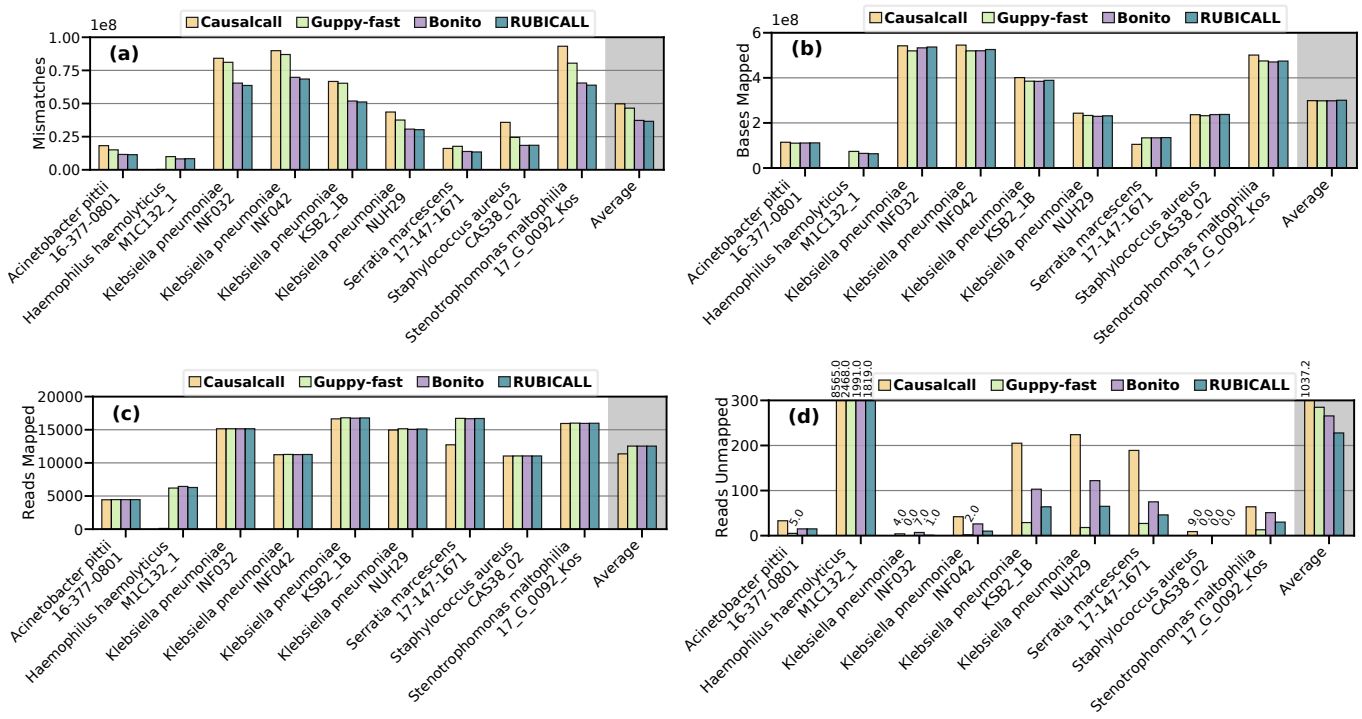


**Figure 12: Comparison of `RUBICALL` for (a) mismatches, (b) bases mapped, (c) reads mapped, and (d) reads unmapped.**

which are 0.56%, 0.83%, and 0.77% higher compared to `Causalcall`, `Guppy-fast`, and `Bonito`. Mapping more bases to the target reference genome confirms that the careful design and optimizations we perform when building `RUBICALL` have no negative effects on the basecalling accuracy. Third, unlike `Causalcall`, `RUBICALL`, `Guppy-fast`, and `Bonito` all provide a high number of mapped reads. However, `RUBICALL` is the only basecaller that provides high-quality reads that have the highest number of base matches and the lowest number of base mismatches. Fourth, `RUBICALL` provides the lowest number of unmapped reads compared to `Causalcall`, `Guppy-fast`, and `Bonito`. `RUBICALL` achieves 78.04%, 19.98%, and 14.23% lower unmapped reads compared to `Causalcall`, `Guppy-fast`, and `Bonito`, respectively. This indicates that using `Causalcall`, `Guppy-fast`, and `Bonito` wastes a valuable expensive resource, i.e., sequencing data, by not mapping reads to the reference genome due to basecalling inaccuracies during basecalling. If a read is flagged as unmapped during read mapping, then this read is excluded from all the following analysis steps affecting the overall downstream analysis results.

We conclude that `RUBICALL` reads provides the highest-quality read mapping results with the largest number of mapped bases and mapped reads.

### 2.6. `SkipClip` Analysis

Figure 13 shows the effect of `SkipClip` on validation accuracy using three different strides at which we remove a skip connection from a block. Our `QABAS`-designed model has five blocks with skip connections. We make three observations. First, `Stride 1` converges faster to the baseline accuracy compared to `Stride 2` and `Stride 3`. By using `Stride 1`, we quickly remove all the skip connections (in five epochs) giving enough fine-tuning iterations for the model to recover its loss in accuracy. Second, all the strides show the maximum drop in accuracy (1.27%-2.88%) when removing skip connections from block 1 and block 4. We observe these blocks consist of the highest number of neural network model parameters due to the skip connections (30.73% and 25.62% of the total model parameters are present in skip connections in block 1

and block 4, respectively). Therefore, the model requires more training epochs to recover its accuracy after the removal of skip connections from these blocks. Third, a lower stride can get rid of skip connections faster than using a higher stride. However, all strides eventually converge to the baseline accuracy at the expense of more training iterations. We conclude that `SkipClip` provides an efficient mechanism to remove hardware-unfriendly skip connections without any loss in basecalling accuracy.

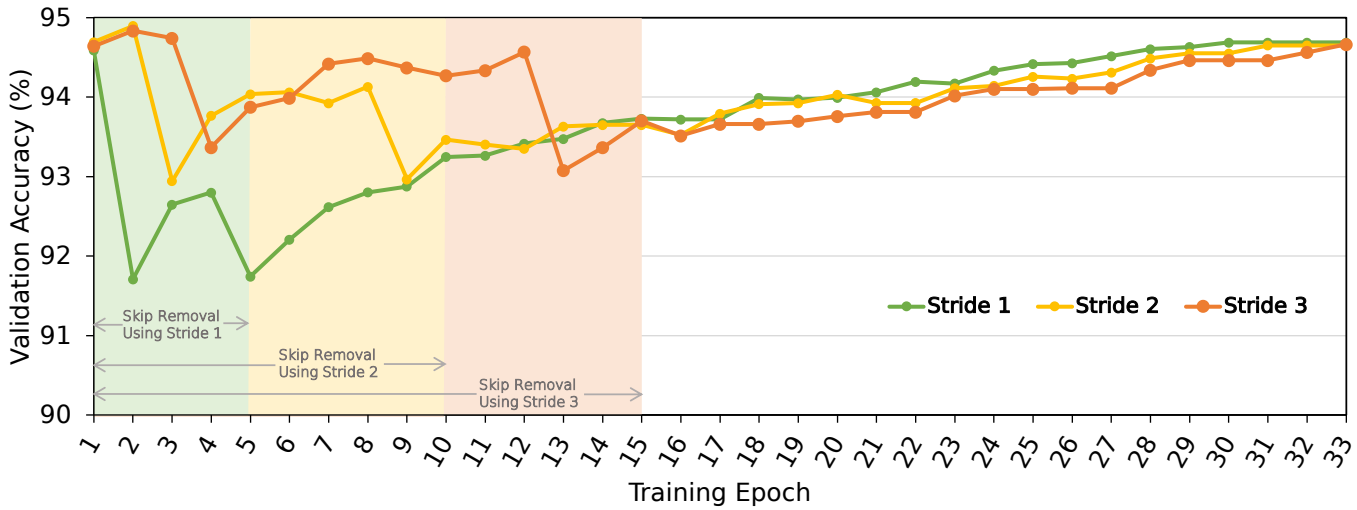


**Figure 13: Effect of different strides while removing skip connections.**

## 2.7. Effect of Pruning `RUBICALL`

Figure 14 shows the effect of pruning `RUBICALL` using two different pruning methods: unstructured element pruning and structured channel pruning.

We make four major observations. First, we can remove up to 15% and 5% of model parameters providing 1.18% and 1.05% reductions in model size without any loss in accuracy by using unstructured pruning and structured pruning, respectively. However, unstructured pruning is unsuitable for hardware acceleration due to irregular structure, and structured pruning provides minimal model size (or parameters) savings. Therefore, we do not apply these pruning techniques to optimize `RUBICALL` further. Second, we observe a drop in accuracy for pruning levels greater than 15% and 5% for unstructured and structured pruning, respectively. This shows that `QABAS` found an optimal architecture as there is little room for pruning `RUBICALL` further without loss in accuracy.

Third, we observe that the *knee point* for unstructured pruning and structured pruning lies at 90% and 50%, where we achieve 80.65% and 70.10% of accuracy with 9.99× and 1.99× savings model size, respectively. After the knee point, we observe a sharp decline in accuracy. Fourth, below the knee point, we can trade accuracy for speed to further accelerate `RUBICALL` for hardware computation and resources by removing unimportant network weights. We conclude that pruning provides a tradeoff between accuracy and model size that can lead to further reductions in processing and memory demands for `RUBICALL`, depending on the type of device on which genomic analyses would be performed.

## 2.8. Explainability Into `QABAS` Results

We perform an explainability analysis to understand our results further and explain `QABAS`'s decisions. The search performed by `QABAS` provides insight into whether `QABAS` has learned meaningful representations in basecalling. In Figure 15(a) and 15(b), we extract the number of model parameters and precision of each parameter in a neural network layer to calculate the total size for each layer for `Bonito` and `RUBICALL`,

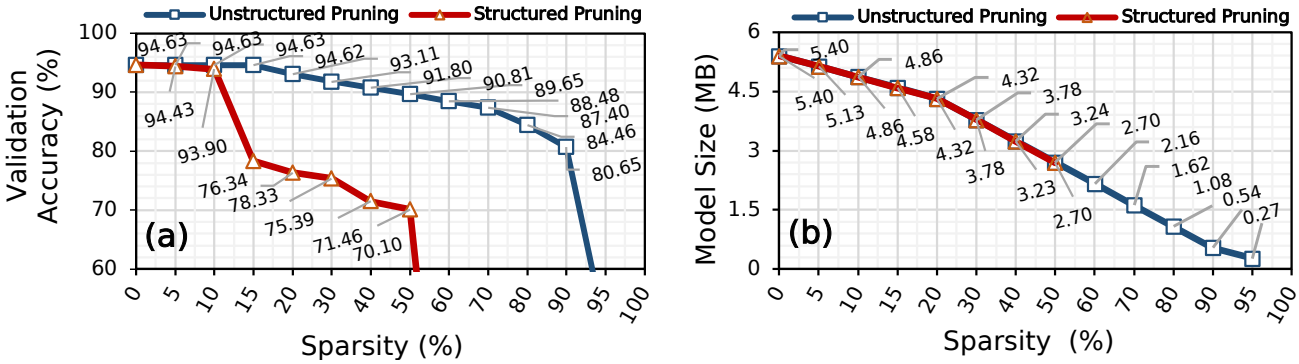


**Figure 14: Effect of pruning RUBICALL on: (a) validation accuracy and (2) model size.**

respectively. We make three observations. First, QABAS uses more bits in the initial layers than the final layers in RUBICALL. QABAS learns that the input to RUBICALL uses an analog squiggle that requires higher precision, while the output is only the nucleotide bases (A, C, G, T), which can be represented using lower precision.

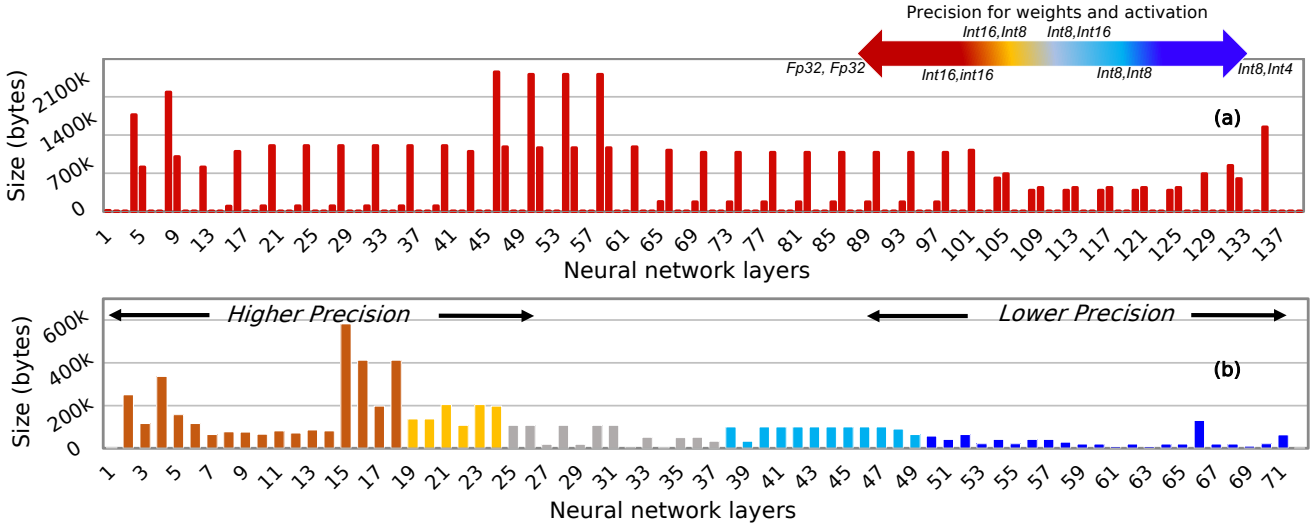


**Figure 15: Layer size comparison for basecallers: (a) Bonito, and (b) RUBICALL.**

Second, RUBICALL uses 1.97× less number of neural network layers than Bonito while providing similar or higher basecalling accuracy on the evaluated species (Section 2.4). Thus, the superior performance of a basecaller architecture is not explicitly linked to its model complexity, and QABAS-designed models are parameter efficient. Third, Bonito uses the same single-precision floating-point representation (FP32) for all neural network layers, which leads to very high memory bandwidth and processing demands. Whereas RUBICALL has every layer quantized to a different quantization domain. We conclude that QABAS provides an efficient automated method for designing more accurate, efficient, and hardware-friendly genomic basecallers compared to expert-designed basecallers.

## 3. Discussion

We are witnessing a tremendous transformation in high-throughput sequencing to significantly advance omics and other life sciences. The bioinformatics community has developed a multitude of software tools to leverage increasingly large and complex sequencing datasets. Deep learning models have been especially powerful in modeling basecalling. Basecalling is the most fundamental computational step in the high-throughput sequencing pipeline. Modern basecallers generally employ convolution neural networks

to extract features from raw genomic sequences. However, designing a basecaller comes with a cost that a neural network model can have many different computational elements making the neural network tuning a major problem. At present, the vast majority of deep learning-based basecallers are manually tuned by computational biologists through manual trial and error, which is time-consuming. To a large extent, basecallers are being designed to provide higher accuracy without considering the compute demands of such networks. Such an approach leads to computationally complex basecallers that impose a substantial barrier to performing end-to-end time-sensitive genomic analyses. This vast dependence of computational biologists and biomedical researchers on these deep learning-based models creates a critical need to find efficient basecalling architectures optimized for performance.

To address this challenge, we developed: (a) `QABAS`: an automatic architecture search framework that jointly searches for computation blocks in basecaller and best bit-with precision for each neural network layer, and (b) `SkipClip`: a dynamic skip removal module to remove hardware-unfriendly skip connections to greatly reduce resource and storage requirements without any loss in basecalling accuracy. We demonstrate the capabilities of `QABAS` and `SkipClip` by designing `RUBICALL`, the first hardware-optimized basecaller that provides both accuracy and inference efficiency.

During our evaluation, we ran `QABAS` for 96 GPU hours to sample architectures from our search space. Using complete sampling to evaluate all the $1.8\times10^{32}$ viable options would take at least $\sim4.3\times10^{33}$ GPU hours. Thus, `QABAS` accelerates the basecaller architecture search to develop high-performance basecalling architectures. The final model architecture can be further fine-tuned for other hyperparameters [80,81], such as learning rate and batch size (for example, with grid search or neural architecture search). Throughout our experiments, we build general-purpose basecalling models by training and testing the model using an official, open-source ONT dataset that consists of a mix of different species. We did not specialize basecalling models for a specific specie. Past works, such as [29], show that higher basecalling accuracy can be achieved by building species-specific models.

As future work, `QABAS` can be extended in two ways: (1) evaluate advance model architectures (such as RNN, transformer, etc.), and (2) perform more fine-grain quantization. First, extending `QABAS` to other model architectures is important for researchers to quickly evaluate different computational elements. We focus on convolution-based networks because matrix multiplication is the fundamental operation in such networks that is easily amenable to hardware acceleration. As the field of machine learning is rapidly evolving, it is non-trivial for researchers to adapt their models with the latest deep learning techniques. Second, currently, we perform mixed precision quantization, where every layer is quantized to a different domain. In the future, we can quantize every dimension of the weights to different precision. Such an approach would increase the design space of neural network architectural options to many folds. `QABAS` enables easy integration to explore such options automatically. Thus, `QABAS` is easily extensible and alleviates the designer's burden in exploring and finding sophisticated basecallers for different hardware configurations.

For `SkipClip`, we demonstrate its applicability on basecalling only, while there are other genome sequencing tasks where deep learning models with skip connections are actively being developed, such as predicting the effect of genetic variations [69, 82], detecting replication dynamics [83], and predicting super-enhancers [84]. In Supplementary S1, we show the effect of manual skip removal, where we manually remove all the skip connections at once. We observe that the basecaller achieves 90.55% accuracy (4.08% lower than the baseline model with skip connections). By manual skip removal, the basecaller is unable to recover the loss in accuracy because CNN-based basecallers are sensitive to skip connections. Therefore, `SkipClip` provides a mechanism to develop hardware-friendly deep learning models for other genomic tasks.

We would explore two future directions for pruning a basecaller. First, currently, we perform one-shot pruning, whereby we prune the model once and then fine-tune the model until convergence. Another approach could be to perform iterative pruning, where after every training epoch, we can re-prune the model using certain pruning criteria. Such an approach would further evaluate the fine-grained pruning limit of a basecaller. Second, an interesting future direction would be to combine multiple pruning techniques, e.g., structured channel pruning with structured group pruning (where we maintain the structure of the tensors without causing sparsity). Such an approach could lead to higher pruning ratios without substantial accuracy loss.

Our extensive evaluation of real genomic organisms shows that `RUBICALL` provides the ability to basecall quickly, accurately, and efficiently enough to scale the analysis, leading to ∼6.88× reductions in model size with 2.94× fewer neural network model parameters. We observe researchers building larger and larger basecallers in an attempt to gain more accuracy without heeding to the disproportionately higher amount of power these basecallers are consuming. Moreover, none of the previous basecallers [29, 30, 39–42, 44, 69] have been optimized for mixed-precision execution to reduce energy consumption. As energy usage is proportional to the size of the network, energy-efficient basecalling is essential to enable the adoption of more and more sophisticated basecallers. We hope that our open-source implementations of `QABAS` and `SkipClip` inspire future work and ideas in genomics and general omics research and development.

## 4. Methods

**Evaluation setup.** Table 2 provides our system details. We evaluate `RUBICALL` using: (1) a cutting-edge spatial vector computing system from AMD-Xilinx (`RUBICALL`-MP) using mixed-precision computation, and (2) AMD Mi50 GPU [85] (`RUBICALL`-FP) using floating-point precision computation. We train and evaluate `RUBICALL` and our baseline basecallers (`Causalcall`, `Guppy-fast`, and `Bonito`) using the same MI50 GPU. The AMD-Xilinx system is the Versal ACAP VC2802 featuring Versal AI Engine (AIE) ML [86], with 304 AIE-ML cores. We estimate the performance on AIE-ML by calculating bit operations (BOPs) [87] that measures the number of bitwise operations in a given network from its ONNX (Open Neural Network Exchange) [88] representation. The AIE vector datapath implements two-dimensional single instruction, multiple data (SIMD) [89] operations using precisions ranging from int4×int8 to int16×int16 operands that can execute 512 to 64 multiply-accumulate operations (MACs) per cycle, respectively.

**Table 2: System parameters and hardware configuration for the CPU, GPU, and the AMD-Xilinx Versal ACAP.**

| | |
|---|---|
| **CPU** | AMD EPYC 7742 [90]<br>@2.25GHz, 4-way SMT [91] |
| **Cache-Hierarchy** | 32×32 KiB L1-I/D, 512 KiB L2, 256 MiB L3 |
| **System Memory** | 4×32GiB RDIMM DDR4 2666 MHz [92] PCIe 4.0 ×128 |
| **OS details** | Ubuntu 21.04 Hirsute Hippo [93],<br>GNU Compiler Collection (GCC) version 10.3.0 [94] |
| **GPU** | AMD Radeon Instinct™ MI50 [85] 3840 Stream Processors<br>32GB HBM2 PCIe 4.0 ×16, ROCm version 5.1.1 [95] |
| **AMD-Xilinx Versal ACAP** | Versal ACAP VC2802 [86], 304×AIEngine-ML@1GHz,<br>19MB local memory, Dual-Core Arm Cortex-A72 [96] |

**`QABAS` setup details.** We use the publicly available ONT dataset [58] sequenced using MinION Flow Cell (R9.4.1) for the training and validation during the `QABAS` search phase. We randomly select 30k

samples from the training set for the search phase (specified using the `--chunks` parameter). We use nni [97] with nn-meter [98] to implement hardware-aware NAS. We use the Brevitas library [99] to perform quantization-aware training. The architectural parameters and network weights are updated using AdamW [100] optimizer with a learning rate of $2e^{-3}$, a beta value of 0.999, a weight decay of 0.01, and an epsilon of $1e^{-8}$. We set the hyperparameter $\lambda$ to 0.6. We choose these values based on our empirical analysis. After the `QABAS` search phase, the sampled networks are trained until convergence with knowledge distillation using the same ONT dataset that we use during the `QABAS` search phase, with a batch size of 64, based on the maximum memory capacity of our evaluated GPU. We set knowledge distillation hyperparameters alpha ($\alpha$) and temperature ($\tau$) at 0.9 and 2, respectively.

**`QABAS` search space.** For the computations operations, we search for a design with one-dimensional (1D) convolution with ten different options: ten kernel size (KS) options (3, 5, 7, 9, 25, 31, 55, 75, 115, and 123) for grouped 1-D convolutions. We also use an identity operator that, in effect, removes a layer to get a shallower network. For quantization bits, we use bit-widths that are a factor of $2^n$, where 2<n<4 (since we need at least 2 bits to represent nucleotides A, C, G, T and 1 additional bit to represent an undefined character in case of a misprediction). We use four different quantization options for weights and activation (<8,4>, <8,8>, <16,8>, and <16,16>). We choose these quantization levels based on the precision support provided by our evaluated hardware and the effect of quantization on basecalling (Section 7). We use five different channel sizes with four repeats each. We choose the number of repeats based on the maximum memory capacity of our evaluated GPU. In total, we have $\sim 1.8 \times 10^{32}$ distinct model options in our search space $\mathcal{M}$.

**`SkipClip` details.** We use `Bonito` as the teacher network, while the `QABAS`-designed model is the student network. We remove skip connections with a stride 1 (using parameter `--skip_stride`). Based on hyper-parameter tuning experiments (Supplementary S2), set knowledge distillation hyperparameters alpha ($\alpha$) and temperature ($\tau$) at 0.9 and 2, respectively. We use Kullback-Leibler divergence loss to calculate the loss [101].

**Pruning details.** We use PyTorch [102] modules for both unstructured and structured pruning [103] with L1-norm, i.e., prune the weights that have the smallest absolute values. We apply one-shot pruning, where we first prune a model with a specific amount of sparsity, then train the model until convergence on the full ONT dataset [58].

**Baseline basecallers.** We compare `RUBICALL` against three different basecallers: (1) `Causalcall` [38] is a state-of-the-art hand-tuned basecaller optimized for performance, (2) `Guppy-fast` [104] is an official production version of basecaller from ONT that is optimized for throughput for real-time basecalling on Nanopore devices, and (3) `Bonito` [58] is an official development version of basecaller from ONT used for research and development. Since `Guppy-fast` required packages are supported only on NVIDIA GPUs, we estimate the performance of `Guppy-fast`.[5] We are aware of other basecallers such as `Halcyon` [42], `SACall` [43], `Helix` [40], `Fast-bonito` [41], and `Dorado` [105]. However, these basecallers are either not open-source [40], do not provide training code [41–43, 105], or are under development with support only for specific reads [105].

**Basecalling reads.** To evaluate basecalling performance, we use a set of reads generated using a MinION R9.4.1 flowcell. Table 3 provides details on different organisms used in our evaluation.

**Basecaller evaluation metrics.** We evaluate the performance of `RUBICALL` using two different metrics: (1) basecalling throughput (kbp/sec), i.e., the throughput of a basecaller in terms of kilo basepairs generated per second, and (2) basecalling accuracy (%), i.e., the total number of bases of a read that are exactly matched

[5]Extrapolated from comparable NVIDIA GPU measurements with a constant throughput factor observed for the other basecallers.

**Table 3: Details of datasets used in evaluation.**

| Organism | Chemistry | # Reads[6] | Reference[7] Genome Size (bp) |
|---|---|---|---|
| Acinetobacter pittii 16-377-0801 | R9.4.1 | 4,467 | 3,814,719 |
| Haemophilus haemolyticus M1C132_1 | R9.4 | 8,669 | 2,042,591 |
| Klebsiella pneumoniae INF032 | R9.4 | 15,154 | 5,111,537 |
| Klebsiella pneumoniae INF042 | R9.4 | 11,278 | 5,337,491 |
| Klebsiella pneumoniae KSB2_1B | R9.4 | 15,178 | 5,228,889 |
| Klebsiella pneumoniae NUH29 | R9.4 | 11,047 | 5,134,281 |
| Serratia marcescens 17-147-1671 | R9.4.1 | 16,847 | 5,517,578 |
| Staphylococcus aureus CAS38_02 | R9.4.1 | 16,742 | 2,902,076 |
| Stenotrophomonas maltophilia 17_G_0092_Kos | R9.4 | 16,010 | 4,802,733 |

[1]Read set: https://bridges.monash.edu/articles/dataset/Raw_fast5s/7676174
[2]Reference genome: https://bridges.monash.edu/articles/dataset/Reference_genomes/7676135

to the bases of the reference genome divided by the total length of its alignment including insertions and deletions. We measure the basecalling throughput for the end-to-end basecalling calculations, including reading FAST5 files and writing out FASTQ or FASTA file using Linux */usr/bin/time -v* command. For basecalling accuracy, we align each basecalled read to its corresponding reference genome of the same species using the state-of-the-art read mapper, minimap2 [106].

**Downstream analysis.** We evaluate the effect of using `RUBICALL` and other baseline basecallers on two widely-used downstream analyses, *de novo* assembly [107] and read mapping [108].

***De novo* assembly.** We construct *de novo* assemblies from the basecalled reads and calculate the statistics related to the accuracy, completeness, and contiguity of these assemblies. To generate *de novo* assemblies, we use minimap2 [106] to report all read overlaps and miniasm [45] to construct the assembly from these overlaps. We use miniasm because it allows us to observe the effect of the reads on the assemblies without performing additional error correction steps on input reads [109] and their final assembly [35]. To measure the assembly accuracy, we use dnadiff [110] to evaluate 1) the portion of the reference genome that can align to a given assembly (i.e., Genome Fraction) and 2) the average identity of assemblies (i.e., Average Identity) when compared to their respective reference genomes. To measure statistics related to the contiguity and completeness of the assemblies, such as the overall assembly length, average GC content (i.e., the ratio of G and C bases in an assembly), and NG50 statistics (i.e., shortest contig at the half of the overall reference genome length), we use QUAST [111]. We assume that the reference genomes are high-quality representative of the sequenced samples that we basecall the reads from when comparing assemblies to their corresponding reference genomes. The higher the values of the average identity, genome fraction, and NG50 results, the higher quality of the assembly and hence the better the corresponding basecaller. When the values of the average GC and assembly length results are closer to that of the corresponding reference genome, the better the assembly and the corresponding basecaller.

**Read mapping.** We basecall the raw electrical signals into reads using each of the subject basecallers. We map the resulting read set to its reference genome of the same species using the state-of-the-art read mapper, minimap2 [106]. We use the default parameter values for mapping ONT reads using the preset parameter *-x map-ont*. We use the *stats* tool from the SAMtools library [112] to obtain four key statistics on the quality of read mapping results, the total number of mismatches, the total number of mapped bases, the total number of mapped reads, and the total number of unmapped reads.

## 5. Data Availability

The read set used in this study was downloaded from https://bridges.monash.edu/articles/dataset/Raw_fast5s/7676174, while the reference set can be downloaded from https://bridges.monash.edu/articles/dataset/Reference_genomes/7676135. We use the official ONT dataset [58] for training and evaluating all our models.

## 6. Code Availability

Scripts used to perform basecalling accuracy analysis are available at: https://github.com/rrwick/Basecalling-comparison. Source code, including pre-trained models, scripts to generate basecalled reads, and reproducible results instruction, will be made publicly available upon publication.

## 7. Acknowledgments

We thank the SAFARI Research Group members for their valuable feedback and the stimulating intellectual and scholarly environment they provide. SAFARI Research Group acknowledges the generous gifts of our industrial partners, including Google, Huawei, Intel, Microsoft, VMware, and Xilinx. This research was partially supported by the Semiconductor Research Corporation. Special thanks to Alessandro Pappalardo for his support with quantization-aware training. We appreciate valuable discussions with Giovanni Mariani. Thanks to AMD for providing access to the HPC fund cluster [113].

## 8. Competing Interests

All authors declare no competing interests.

# References

[1] G. Ginsburg and K. Phillips, "Precision Medicine: From Science To Value," *Health Affairs*, vol. 37, pp. 694–701, 05 2018.

[2] Z. Aryan, A. Szanto, A. Pantazi, T. Reddi, C. Rheinstein, W. Powers, E. Wilson, R. C. Deo, S. Chowdhury, L. Salz, D. Dimmock, S. Nahas, W. Benson, S. F. Kingsmore, C. A. MacRae, and D. Vuzman, "Moving Genomics to Routine Care: An Initial Pilot in Acute Cardiovascular Disease," *Circulation. Genomic and precision medicine*, vol. 13, no. 5, p. 406—416, October 2020. [Online]. Available: https://doi.org/10.1161/CIRCGEN.120.002961

[3] M. M. Clark, A. Hildreth, S. Batalov, Y. Ding, S. Chowdhury, K. Watkins, K. Ellsworth, B. Camp, C. I. Kint, C. Yacoubian, L. Farnaes, M. N. Bainbridge, C. Beebe, J. J. A. Braun, M. Bray, J. Carroll, J. A. Cakici, S. A. Caylor, C. Clarke, M. P. Creed, J. Friedman, A. Frith, R. Gain, M. Gaughran, S. George, S. Gilmer, J. Gleeson, J. Gore, H. Grunenwald, R. L. Hovey, M. L. Janes, K. Lin, P. D. McDonagh, K. McBride, P. Mulrooney, S. Nahas, D. Oh, A. Oriol, L. Puckett, Z. Rady, M. G. Reese, J. Ryu, L. Salz, E. Sanford, L. Stewart, N. Sweeney, M. Tokita, L. Van Der Kraan, S. White, K. Wigby, B. Williams, B. Williams, T. Wong, M. S. Wright, C. Yamada, P. Schols, J. Reynders, K. Hall, D. Dimmock, N. Veeraraghavan, T. Defay, S. F. Kingsmore, and S. F. Kingsmore, "Diagnosis of Genetic Diseases in Seriously Ill Children by Rapid Whole-Genome Sequencing and Automated Phenotyping and Interpretation," *Science translational medicine*, vol. 11, no. 489, p. eaat6177, April 2019. [Online]. Available: https://doi.org/10.1126/scitranslmed.aat6177

[4] S. F. Kingsmore, L. D. Smith, C. M. Kunard, M. Bainbridge, S. Batalov, W. Benson, E. Blincow, S. Caylor, C. Chambers, G. Del Angel, D. P. Dimmock, Y. Ding, K. Ellsworth, A. Feigenbaum, E. Frise, R. C. Green, L. Guidugli, K. P. Hall, C. Hansen, C. A. Hobbs, S. D. Kahn, M. Kiel, L. Van Der Kraan, C. Krilow, Y. H. Kwon, L. Madhavrao, J. Le, S. Lefebvre, R. Mardach, W. R. Mowrey, D. Oh, M. J. Owen, G. Powley, G. Scharer, S. Shelnutt, M. Tokita, S. S. Mehtalia, A. Oriol, S. Papadopoulos, J. Perry, E. Rosales, E. Sanford, S. Schwartz, D. Tran, M. G. Reese, M. Wright, N. Veeraraghavan, K. Wigby, M. J. Willis, A. R. Wolen, and T. Defay, "A Genome Sequencing System for Universal Newborn Screening, Diagnosis, and Precision Medicine for Severe Genetic Diseases," *American journal of human genetics*, vol. 109, no. 9, p. 1605—1619, September 2022. [Online]. Available: https://doi.org/10.1016/j.ajhg.2022.08.003

[5] G. S. Ginsburg and H. F. Willard, "Genomic and Personalized Medicine: Foundations and Applications," *Translational Research*, vol. 154, no. 6, pp. 277–287, 2009, special Issue on Personalized Medicine. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1931524409002746

[6] J. S. Bloom, L. Sathe, C. Munugala, E. M. Jones, M. Gasperini, N. B. Lubock, F. Yarza, E. M. Thompson, K. M. Kovary, J. Park, D. Marquette, S. Kay, M. Lucas, T. Love, A. Sina Booeshaghi, O. F. Brandenberg, L. Guo, J. Boocock, M. Hochman, S. W. Simpkins, I. Lin, N. LaPierre, D. Hong, Y. Zhang, G. Oland, B. J. Choe, S. Chandrasekaran, E. E. Hilt, M. J. Butte, R. Damoiseaux, C. Kravit, A. R. Cooper, Y. Yin, L. Pachter, O. B. Garner, J. Flint, E. Eskin, C. Luo, S. Kosuri, L. Kruglyak, and V. A. Arboleda, "Massively Scaled-Up Testing for SARS-CoV-2 RNA via Next-Generation Sequencing of Pooled and Barcoded Nasal and Saliva Samples," *Nature Biomedical Engineering*, vol. 5, no. 7, pp. 657–665, Jul 2021. [Online]. Available: https://doi.org/10.1038/s41551-021-00754-5

[7] J. Quick, N. J. Loman, S. Duraffour, J. T. Simpson, E. Severi, L. Cowley, J. A. Bore, R. Koundouno, G. Dudas, A. Mikhail, N. Ouedraogo, B. Afrough, A. Bah, J. H. Baum, B. Becker-Ziaja, J. P. Boettcher, M. Cabeza-Cabrerizo, A. Camino-Sanchez, L. L. Carter, J. Doerrbecker, T. Enkirch, I. Garcia-Dorival, N. Hetzelt, J. Hinzmann, T. Holm, L. E. Kafetzopoulou, M. Koropogui, A. Kosgey, E. Kuisma, C. H. Logue, A. Mazzarelli, S. Meisel, M. Mertens, J. Michel, D. Ngabo, K. Nitzsche, E. Pallasch, L. V. Patrono, J. Portmann, J. G. Repits, N. Y. Rickett, A. Sachse, K. Singethan, I. Vitoriano, R. L. Emanaberhan, E. G. Zekeng, T. Racine, A. Bello, A. A. Sall, O. Faye, O. Faye, N. Magassouba, C. V. Williams, V. Amburgey, L. Winona, E. Davis, J. Gerlach, F. Washington, V. Monteil, M. Jourdain, M. Bererd, A. Camara, H. Somlare, A. Camara, M. Gerard, G. Bado, B. Baillet, D. Delaune, K. Y. Nebie, A. Diarra, Y. Savane, R. B. Pallawo, G. J. Gutierrez, N. Milhano, I. Roger, C. J. Williams, F. Yattara, K. Lewandowski, J. Taylor, P. Rachwal, D. J. Turner, G. Pollakis, J. A. Hiscox, D. A. Matthews, M. K. O'Shea, A. M. Johnston, D. Wilson, E. Hutley, E. Smit, A. DiCaro, R. Woelfel, K. Stoecker, E. Fleischmann, M. Gabriel, S. A. Weller, L. Koivogui, B. Diallo, S. Keita, A. Rambaut, P. Formenty, S. Guenther, and M. W. Carroll, "Real-Time, Portable Genome Sequencing for Ebola Surveillance," 2016-02-11.

[8] R. Yelagandula, A. Bykov, A. Vogt, R. Heinen, E. Özkan, M. M. Strobl, J. C. Baar, K. Uzunova, B. Hajdusits, D. Kordic, E. Suljic, A. Kurtovic-Kozaric, S. Izetbegovic, J. Schaeffer, P. Hufnagl, A. Zoufaly, T. Seitz, VCDI, M. Födinger, F. Allerberger, A. Stark, L. Cochella, and U. Elling, "Multiplexed Detection of SARS-CoV-2 and Other Respiratory Infections in High Throughput by SARSeq," *Nature communications*, vol. 12, no. 1, p. 3132, May 2021. [Online]. Available: https://europepmc.org/articles/PMC8149640

[9] V. T. M. Le and B. A. Diep, "Selected Insights from Application of Whole-Genome Sequencing for Outbreak Investigations," *Current Opinion in Critical Care*, vol. 19, p. 432–439, 2013.

[10] V. Nikolayevskyy, K. Kranzer, S. Niemann, and F. Drobniewski, "Whole Genome Sequencing of M.tuberculosis for Detection

of Recent Transmission and Tracing Outbreaks: A Systematic Review," *Tuberculosis*, vol. 98, 03 2016.

[11] F. Meyer, A. Fritz, Z.-L. Deng, D. Koslicki, A. Gurevich, G. Robertson, M. Alser, D. Antipov, F. Beghini, D. Bertrand *et al.*, "Critical Assessment of Metagenome Interpretation-The Second Round of Challenges," *bioRxiv*, 2021.

[12] N. LaPierre, M. Alser, E. Eskin, D. Koslicki, and S. Mangul, "Metalign: Efficient Alignment-Based Metagenomic Profiling via Containment Min Hash," *Genome biology*, vol. 21, no. 1, pp. 1–15, 2020.

[13] N. LaPierre, S. Mangul, M. Alser, I. Mandric, N. Wu, D. Koslicki, and E. Eskin, "MiCoP: Microbial Community Profiling Method for Detecting Viral and Fungal Organisms in Metagenomic Samples," *BMC Genomics*, vol. 20, p. 423, 06 2019.

[14] F. Meyer, A. Fritz, Z.-L. Deng, D. Koslicki, T. R. Lesker, A. Gurevich, G. Robertson, M. Alser, D. Antipov, F. Beghini, D. Bertrand, J. J. Brito, C. T. Brown, J. Buchmann, A. Buluç, B. Chen, R. Chikhi, P. T. L. C. Clausen, A. Cristian, P. W. Dabrowski, A. E. Darling, R. Egan, E. Eskin, E. Georganas, E. Goltsman, M. A. Gray, L. H. Hansen, S. Hofmeyr, P. Huang, L. Irber, H. Jia, T. S. Jørgensen, S. D. Kieser, T. Klemetsen, A. Kola, M. Kolmogorov, A. Korobeynikov, J. Kwan, N. LaPierre, C. Lemaitre, C. Li, A. Limasset, F. Malcher-Miranda, S. Mangul, V. R. Marcelino, C. Marchet, P. Marijon, D. Meleshko, D. R. Mende, A. Milanese, N. Nagarajan, J. Nissen, S. Nurk, L. Oliker, L. Paoli, P. Peterlongo, V. C. Piro, J. S. Porter, S. Rasmussen, E. R. Rees, K. Reinert, B. Renard, E. M. Robertsen, G. L. Rosen, H.-J. Ruscheweyh, V. Sarwal, N. Segata, E. Seiler, L. Shi, F. Sun, S. Sunagawa, S. J. Sørensen, A. Thomas, C. Tong, M. Trajkovski, J. Tremblay, G. Uritskiy, R. Vicedomini, Z. Wang, Z. Wang, Z. Wang, A. Warren, N. P. Willassen, K. Yelick, R. You, G. Zeller, Z. Zhao, S. Zhu, J. Zhu, R. Garrido-Oter, P. Gastmeier, S. Hacquard, S. Häußler, A. Khaledi, F. Maechler, F. Mesny, S. Radutoiu, P. Schulze-Lefert, N. Smit, T. Strowig, A. Bremges, A. Sczyrba, and A. C. McHardy, "Critical Assessment of Metagenome Interpretation: The Second Round of Challenges," *Nature Methods*, vol. 19, no. 4, pp. 429–440, Apr 2022. [Online]. Available: https://doi.org/10.1038/s41592-022-01431-4

[15] M. O. Pollard, D. Gurdasani, A. J. Mentzer, T. Porter, and M. S. Sandhu, "Long Reads: Their Purpose and Place," *Human Molecular Genetics*, vol. 27, no. R2, pp. R234–R241, Aug. 2018.

[16] D. Senol Cali, J. S. Kim, S. Ghose, C. Alkan, and O. Mutlu, "Nanopore Sequencing Technology and Tools for Genome Assembly: Computational Analysis of the Current State, Bottlenecks and Future Directions," *Briefings in Bioinformatics*, vol. 20, no. 4, pp. 1542–1559, Jul. 2019.

[17] S. L. Amarasinghe, S. Su, X. Dong, L. Zappia, M. E. Ritchie, and Q. Gouil, "Opportunities and Challenges in Long-Read Sequencing Data Analysis," *Genome biology*, vol. 21, no. 1, pp. 1–16, 2020.

[18] G. A. Logsdon, M. R. Vollger, and E. E. Eichler, "Long-Read Human Genome Sequencing and its Applications," *Nature Reviews Genetics*, vol. 21, no. 10, pp. 597–614, 2020.

[19] Y. Wang, Y. Zhao, A. Bollas, Y. Wang, and K. F. Au, "Nanopore Sequencing Technology, Bioinformatics and Applications," *Nature biotechnology*, vol. 39, no. 11, pp. 1348–1365, 2021.

[20] M. Jain, S. Koren, K. H. Miga, J. Quick, A. C. Rand, T. A. Sasani, J. R. Tyson, A. D. Beggs, A. T. Dilthey, I. T. Fiddes, S. Malla, H. Marriott, T. Nieto, J. O'Grady, H. E. Olsen, B. S. Pedersen, A. Rhie, H. Richardson, A. R. Quinlan, T. P. Snutch, L. Tee, B. Paten, A. M. Phillippy, J. T. Simpson, N. J. Loman, and M. Loose, "Nanopore Sequencing and Assembly of a Human Genome with Ultra-Long Reads," *Nature Biotechnology*, vol. 36, no. 4, pp. 338–345, Apr 2018. [Online]. Available: https://doi.org/10.1038/nbt.4060

[21] L. Gong, C.-H. Wong, J. Idol, C. Y. Ngan, and C.-L. Wei, "Ultra-Long Read Sequencing for Whole Genomic DNA Analysis," *JoVE*, no. 145, p. e58954, Mar 2019. [Online]. Available: https://www.jove.com/t/58954

[22] D. Branton, D. W. Deamer, A. Marziali, H. Bayley, S. A. Benner, T. Butler, M. Di Ventra, S. Garaj, A. Hibbs, X. Huang *et al.*, "The potential and challenges of nanopore sequencing," *Nature biotechnology*, vol. 26, no. 10, pp. 1146–1153, 2008.

[23] E. L. Van Dijk, Y. Jaszczyszyn, D. Naquin, and C. Thermes, "The Third Revolution in Sequencing Technology," *Trends in Genetics*, vol. 34, no. 9, pp. 666–681, 2018.

[24] S. Ardui, A. Ameur, J. R. Vermeesch, and M. S. Hestand, "Single Molecule Real-Time (SMRT) Sequencing Comes of Age: Applications and Utilities for Medical Diagnostics," *Nucleic acids research*, vol. 46, no. 5, pp. 2159–2168, 2018.

[25] M. Jain, S. Koren, K. H. Miga, J. Quick, A. C. Rand, T. A. Sasani, J. R. Tyson, A. D. Beggs, A. T. Dilthey, I. T. Fiddes, S. Malla, H. Marriott, T. Nieto, J. O'Grady, H. E. Olsen, B. S. Pedersen, A. Rhie, H. Richardson, A. R. Quinlan, T. P. Snutch, L. Tee, B. Paten, A. M. Phillippy, J. T. Simpson, N. J. Loman, and M. Loose, "Nanopore Sequencing and Assembly of a Human Genome with Ultra-Long Reads," *Nature Biotechnology*, vol. 36, no. 4, pp. 338–345, Apr. 2018.

[26] M. Kchouk, J.-F. Gibrat, and M. Elloumi, "Generations of Sequencing Technologies: From First to Next Generation," *Biology and Medicine*, vol. 9, no. 3, 2017.

[27] J. L. Weirather, M. de Cesare, Y. Wang, P. Piazza, V. Sebastiano, X.-J. Wang, D. Buck, and K. F. Au, "Comprehensive Comparison of Pacific Biosciences and Oxford Nanopore Technologies and Their Applications to Transcriptome Analysis," *F1000Research*, vol. 6, 2017.

[28] M. Jain, H. E. Olsen, B. Paten, and M. Akeson, "The Oxford Nanopore MinION: Delivery of Nanopore Sequencing to the

Genomics Community," *Genome biology*, vol. 17, no. 1, pp. 1–11, 2016.

[29] R. R. Wick, L. M. Judd, and K. E. Holt, "Performance of Neural Network Basecalling Tools for Oxford Nanopore Sequencing," *Genome biology*, vol. 20, no. 1, pp. 1–10, 2019.

[30] M. Pages-Gallego and J. de Ridder, "Comprehensive and Standardized Benchmarking of Deep Learning Architectures for Basecalling Nanopore Sequencing Data," *bioRxiv*, 2022.

[31] M. Alser, J. Lindegger, C. Firtina, N. Almadhoun, H. Mao, G. Singh, J. Gomez-Luna, and O. Mutlu, "From Molecules to Genomic Variations: Accelerating Genome Analysis via Intelligent Algorithms and Architectures," *Computational and Structural Biotechnology Journal*, 2022.

[32] M. Alser, J. Rotman, D. Deshpande, K. Taraszka, H. Shi, P. I. Baykal, H. T. Yang, V. Xue, S. Knyazev, B. D. Singer, B. Balliu, D. Koslicki, P. Skums, A. Zelikovsky, C. Alkan, O. Mutlu, and S. Mangul, "Technology Dictates Algorithms: Recent Developments in Read Alignment," *Genome Biology*, vol. 22, no. 1, p. 249, Aug. 2021.

[33] Y.-z. Zhang, A. Akdemir, G. Tremmel, S. Imoto, S. Miyano, T. Shibuya, and R. Yamaguchi, "Nanopore Basecalling from a Perspective of Instance Segmentation," *BMC bioinformatics*, 2020.

[34] R. Dias and A. Torkamani, "Artificial Intelligence in Clinical and Genomic Diagnostics," *Genome medicine*, vol. 11, no. 1, pp. 1–12, 2019.

[35] C. Firtina, J. S. Kim, M. Alser, D. Senol Cali, A. E. Cicek, C. Alkan, and O. Mutlu, "Apollo: A Sequencing-Technology-Independent, Scalable and Accurate Assembly Polishing Algorithm," *Bioinformatics*, vol. 36, no. 12, pp. 3669–3679, Jun. 2020.

[36] F. J. Rang, W. P. Kloosterman, and J. de Ridder, "From Squiggle to Basepair: Computational Approaches for Improving Nanopore Sequencing Read Accuracy," *Genome Biology*, vol. 19, no. 1, p. 90, Jul 2018. [Online]. Available: https://doi.org/10.1186/s13059-018-1462-9

[37] X. Lv, Z. Chen, Y. Lu, and Y. Yang, "An End-to-End Oxford Nanopore Basecaller Using Convolution-Augmented Transformer," in *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2020, pp. 337–342.

[38] J. Zeng, H. Cai, H. Peng, H. Wang, Y. Zhang, and T. Akutsu, "Causalcall: Nanopore Basecalling Using a Temporal Convolutional Network," *Frontiers in Genetics*, p. 1332, 2020.

[39] P. Perešíni, V. Boža, B. Brejová, and T. Vinař, "Nanopore Base Calling on the Edge," *Bioinformatics*, vol. 37, no. 24, pp. 4661–4667, 2021.

[40] Q. Lou, S. C. Janga, and L. Jiang, "Helix: Algorithm/Architecture Co-design for Accelerating Nanopore Genome Basecalling," in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, 2020, pp. 293–304.

[41] Z. Xu, Y. Mai, D. Liu, W. He, X. Lin, C. Xu, L. Zhang, X. Meng, J. Mafofo, W. A. Zaher *et al.*, "Fast-bonito: A Faster Deep Learning Based Basecaller for Nanopore Sequencing," *Artificial Intelligence in the Life Sciences*, vol. 1, p. 100011, 2021.

[42] H. Konishi, R. Yamaguchi, K. Yamaguchi, Y. Furukawa, and S. Imoto, "Halcyon: An Accurate Basecaller Exploiting an Encoder–Decoder Model with Monotonic Attention," *Bioinformatics*, vol. 37, no. 9, pp. 1211–1217, 2021.

[43] N. Huang, F. Nie, P. Ni, F. Luo, and J. Wang, "SACall: A Neural Network Basecaller for Oxford Nanopore Sequencing Data Based on Self-Attention Mechanism," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2020.

[44] D. Neumann, A. S. Reddy, and A. Ben-Hur, "RODAN: A Fully Convolutional Architecture for Basecalling Nanopore RNA Sequencing Data," *BMC bioinformatics*, vol. 23, no. 1, pp. 1–9, 2022.

[45] H. Li, "Minimap and Miniasm: Fast Mapping and De Novo Assembly for Noisy Long Sequences," *Bioinformatics*, vol. 32, no. 14, pp. 2103–2110, Jul. 2016.

[46] S. Kriman, S. Beliaev, B. Ginsburg, J. Huang, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, and Y. Zhang, "QuartzNet: Deep Automatic Speech Recognition with 1D Time-Channel Separable Convolutions," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 6124–6128.

[47] S. Majumdar, J. Balam, O. Hrinchuk, V. Lavrukhin, V. Noroozi, and B. Ginsburg, "Citrinet: Closing the Gap Between Non-Autoregressive and Autoregressive End-to-End Models for Automatic Speech Recognition," *arXiv preprint arXiv:2104.01721*, 2021.

[48] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu *et al.*, "Conformer: Convolution-Augmented Transformer for Speech Recognition," *arXiv preprint arXiv:2005.08100*, 2020.

[49] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[50] G. Singh, D. Diamantopoulos, S. Stuijk, C. Hagleitner, and H. Corporaal, "Low precision processing for high order stencil computations," in *International Conference on Embedded Computer Systems*. Springer, 2019, pp. 403–415.

[51] G. Singh, D. Diamantopoulos, C. Hagleitner, J. Gómez-Luna, S. Stuijk, O. Mutlu, and H. Corporaal, "Nero: A near high-bandwidth memory stencil accelerator for weather prediction modeling," in *2020 30th International Conference on*

*Field-Programmable Logic and Applications (FPL)*. IEEE, 2020, pp. 9–17.

[52] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," *arXiv preprint arXiv:1611.01578*, 2016.

[53] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model Compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 535–541.

[54] Y. LeCun, J. Denker, and S. Solla, "Optimal Brain Damage," *Advances in neural information processing systems*, vol. 2, 1989.

[55] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *arXiv*, 2015.

[56] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both Weights and Connections for Efficient Neural Network," *Advances in neural information processing systems*, vol. 28, 2015.

[57] J. Frankle and M. Carbin, "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks," *arXiv preprint arXiv:1803.03635*, 2018.

[58] "Bonito, https://github.com/nanoporetech/bonito."

[59] G. Hinton, O. Vinyals, J. Dean *et al.*, "Distilling the Knowledge in a Neural Network," *arXiv preprint arXiv:1503.02531*, vol. 2, no. 7, 2015.

[60] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[61] A. F. Agarap, "Deep Learning Using Rectified Linear Units (ReLU)," *arXiv*, 2018.

[62] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–34, 2021.

[63] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable Architecture Search," *arXiv*, 2018.

[64] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural Architecture Optimization," *Advances in neural information processing systems*, vol. 31, 2018.

[65] L. Xie, X. Chen, K. Bi, L. Wei, Y. Xu, L. Wang, Z. Chen, A. Xiao, J. Chang, X. Zhang *et al.*, "Weight-Sharing Neural Architecture Search: A Battle to Shrink the Optimization Gap," *ACM Computing Surveys (CSUR)*, vol. 54, no. 9, pp. 1–37, 2021.

[66] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-Scale Evolution of Image Classifiers," in *International Conference on Machine Learning*. PMLR, 2017, pp. 2902–2911.

[67] L. Xie and A. Yuille, "Genetic CNN," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1379–1388.

[68] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct Neural Architecture Search on Target Task and Hardware," *arXiv*, 2018.

[69] Z. Zhang, C. Y. Park, C. L. Theesfeld, and O. G. Troyanskaya, "An Automated Framework for Efficiently Designing Deep Convolutional Neural Networks in Genomics," *Nature Machine Intelligence*, vol. 3, no. 5, pp. 392–400, 2021.

[70] S. Hochreiter, "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[71] S.-i. Amari, "Backpropagation and Stochastic Gradient Descent Method," *Neurocomputing*, vol. 5, no. 4-5, pp. 185–196, 1993.

[72] P. J. Werbos, "Backpropagation Through Time: What It Does and How To Do It," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[73] H.-J. Kang, "Accelerator-Aware Pruning for Convolutional Neural Networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 7, pp. 2093–2103, 2019.

[74] T. Gale, E. Elsen, and S. Hooker, "The State of Sparsity in Deep Neural Networks," *arXiv preprint arXiv:1902.09574*, 2019.

[75] J. K. Kruschke and J. R. Movellan, "Benefits of Gain: Speeded Learning and Minimal Hidden Layers in Back-Propagation Networks," *IEEE Transactions on systems, Man, and Cybernetics*, vol. 21, no. 1, pp. 273–280, 1991.

[76] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the Value of Network Pruning," *arXiv preprint arXiv:1810.05270*, 2018.

[77] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 369–376.

[78] M. Ferrarini, M. Moretto, J. A. Ward, N. Šurbanovski, V. Stevanović, L. Giongo, R. Viola, D. Cavalieri, R. Velasco, A. Cestaro, and D. J. Sargent, "An Evaluation of the PacBio RS Platform for Sequencing and De Novo Assembly of a Chloroplast Genome," *BMC Genomics*, vol. 14, no. 1, p. 670, Oct. 2013.

[79] Y.-C. Chen, T. Liu, C.-H. Yu, T.-Y. Chiang, and C.-C. Hwang, "Effects of GC Bias in Next-Generation-Sequencing Data on De Novo Genome Assembly," *PLOS ONE*, vol. 8, no. 4, p. e62856, Apr. 2013.

[80] G. Singh, J. Gómez-Luna, G. Mariani, G. F. Oliveira, S. Corda, S. Stuijk, O. Mutlu, and H. Corporaal, "Napel: Near-memory computing application performance prediction via ensemble learning," in *2019 56th ACM/IEEE Design Automation*

*Conference (DAC)*. IEEE, 2019, pp. 1–6.

[81] G. Singh, R. Nadig, J. Park, R. Bera, N. Hajinazar, D. Novo, J. Gómez-Luna, S. Stuijk, H. Corporaal, and O. Mutlu, "Sibyl: Adaptive and extensible data placement in hybrid storage systems using online reinforcement learning," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 320–336. [Online]. Available: https://doi.org/10.1145/3470496.3527442

[82] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey, "Predicting the Sequence Specificities of DNA-and RNA-Binding Proteins by Deep Learning," *Nature biotechnology*, vol. 33, no. 8, pp. 831–838, 2015.

[83] M. A. Boemo, "DNAscent v2: Detecting Replication Forks in Nanopore Sequencing Data with Deep Learning," *BMC genomics*, vol. 22, no. 1, pp. 1–8, 2021.

[84] S. Sabba, M. Smara, M. Benhacine, and A. Hameurlaine, "Residual Neural Network for Predicting Super-Enhancers on Genome Scale," in *International Conference on Artificial Intelligence and its Applications*. Springer, 2021, pp. 32–42.

[85] "AMD Radeon Instinct™ MI50 Accelerator (32GB), https://www.amd.com/system/files/documents/radeon-instinct-mi50-datasheet.pdf."

[86] "Versal ACAP AI Core Series Product Selection Guide, https://www.xilinx.com/content/dam/xilinx/support/documents/selection-guides/versal-ai-core-product-selection-guide.pdf."

[87] C. Baskin, N. Liss, E. Schwartz, E. Zheltonozhskii, R. Giryes, A. M. Bronstein, and A. Mendelson, "Uniq: Uniform noise injection for non-uniform quantization of neural networks," *ACM Transactions on Computer Systems (TOCS)*, vol. 37, no. 1-4, pp. 1–15, 2021.

[88] "Open Neural Network Exchange (ONNX), https://github.com/onnx/onnx."

[89] G. H. Barnes, R. M. Brown, M. Kato, D. J. Kuck, D. L. Slotnick, and R. A. Stokes, "The ILLIAC IV Computer," *IEEE Transactions on Computers*, 1968.

[90] "Introducing 3rd Gen AMD EPYC™ Processors, https://www.amd.com/en/events/epyc."

[91] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism," in *ISCA*, 1995.

[92] "RDIMM, https://www.micron.com/products/dram-modules/rdimm."

[93] "Ubuntu 20.04.3 LTS (Focal Fossa), https://releases.ubuntu.com/20.04/."

[94] GCC, the GNU Compiler Collection. [Online]. Available: https://gcc.gnu.org/

[95] "ROCm, https://github.com/RadeonOpenCompute/ROCm."

[96] "ARM Cortex-A72 MPCore Processor Technical Reference Manual r0p3, https://developer.arm.com/documentation/100095/0003."

[97] "NNI, https://github.com/microsoft/nni."

[98] M. R. nn Meter Team, "nn-Meter: Towards Accurate Latency Prediction of Deep-Learning Model Inference on Diverse Edge Devices," 2021. [Online]. Available: https://github.com/microsoft/nn-Meter

[99] A. Pappalardo, "Xilinx/brevitas," 2021. [Online]. Available: https://doi.org/10.5281/zenodo.3333552

[100] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv*, 2014.

[101] "KLDivLoss, https://pytorch.org/docs/stable/generated/torch.nn.KLDivLoss.html."

[102] "PyTorch, https://pytorch.org/."

[103] "TORCH.NN, https://pytorch.org/docs/stable/nn.html."

[104] "How Basecalling Works, https://nanoporetech.com/how-it-works/basecalling."

[105] "Dorado, https://github.com/nanoporetech/dorado.git."

[106] H. Li, "Minimap2: Pairwise Alignment for Nucleotide Sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, Sep. 2018.

[107] G. Robertson, J. Schein, R. Chiu, R. Corbett, M. Field, S. D. Jackman, K. Mungall, S. Lee, H. M. Okada, J. Q. Qian *et al.*, "De Novo Assembly and Analysis of RNA-seq Data," *Nature methods*, vol. 7, no. 11, pp. 909–912, 2010.

[108] B. Li, V. Ruotti, R. M. Stewart, J. A. Thomson, and C. N. Dewey, "RNA-Seq Gene Expression Estimation with Read Mapping Uncertainty," *Bioinformatics*, vol. 26, no. 4, pp. 493–500, 2010.

[109] C. Firtina, Z. Bar-Joseph, C. Alkan, and A. E. Cicek, "Hercules: A Profile HMM-based Hybrid Error Correction Algorithm for Long Reads," *Nucleic Acids Research*, vol. 46, no. 21, pp. e125–e125, Nov. 2018.

[110] G. Marçais, A. L. Delcher, A. M. Phillippy, R. Coston, S. L. Salzberg, and A. Zimin, "MUMmer4: A Fast and Versatile Genome Alignment System," *PLOS Computational Biology*, vol. 14, no. 1, p. e1005944, Jan. 2018.

[111] A. Gurevich, V. Saveliev, N. Vyahhi, and G. Tesler, "QUAST: Quality Assessment Tool for Genome Assemblies," *Bioinformatics*, vol. 29, no. 8, pp. 1072–1075, Apr. 2013.

[112] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, . G. P. D. P. Subgroup *et al.*, "The Sequence Alignment/Map (SAM) Format and SAMtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.

[113] "AMD HPC Fund, https://www.amd.com/en/corporate/hpc-fund."

# Supplementary Material for A Framework for Designing Efficient Deep Learning-Based Genomic Basecallers

## S1. Sensitivity to Skip Connection

Many state-of-the-art deep learning-based basecallers [29, 30, 39–42, 44, 69] incorporate skip connections to improve their basecalling accuracy. Figure S1 shows the accuracy of `Bonito` using two different configurations of skip connections (`s1` and `s2`) and one configuration without any skip connections (`s3`) and compares it to the baseline `Bonito` architecture. In `s1` configuration, we reduce the number of repeats in each block to one, while in `s2` configuration, we use only one block with maximum channel size, maximum kernel size, and maximum number of repeats.

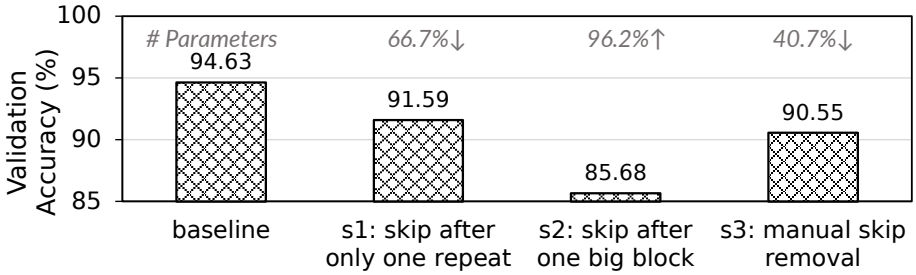


**Figure S1: Basecaller sensitivity to skip connections.**

For `s3` configuration, we manually remove all the skip connections from each block in `Bonito`. We also annotate the change in model parameters compared to the baseline model. `Bonito` architecture comprises several blocks, each consisting of a time channel separable convolution sub-block (referred to as repeat). We make two major observations. First, the number sub-block we provide skip connection plays an important role. In `s1` configuration, we observe that by using only one repeat, we reduce the accuracy by 2.84% with 66.7% lower model parameters, while by merging all the blocks into one big block in `s2` configuration, we observe 8.75% lower accuracy with $96.2\%$ higher model parameters. Second, manually removing all the skip connections in `s3` configuration leads to $40.7\%$ lower model parameters at the expense of a $3.88\%$ loss in accuracy. This performance degradation is because, during neural network training, these connections provide a direct path for propagating the error through the layers and dealing with the vanishing gradient problem, allowing deep networks to learn properly and converge during training. Therefore, manual removal of skip connections can lead to lower basecalling performance. We conclude that skip connections are critical for basecalling accuracy.

## S2. Hyper-parameter Tuning for `SkipClip`

In Figure S2, we show the effect of two critical hyper-parameters (alpha ($\alpha$) and temperature ($\tau$)) on `SkipClip`. We observe that as we raise $\alpha$ while keeping $\tau$ constant, the basecaller accuracy increases. At higher $\alpha$, `SkipClip` gives more importance to the student loss than the distillation loss during the backward pass. We use $\alpha$ =0.9 throughout our experiments.

For $\tau$, we experiment with values ranging from 0.5 to 5.0. Increasing $\tau$ provides more knowledge from the teacher network for a student network to absorb. We observe at $\tau$=2, `SkipClip` provides the highest accuracy. Further increasing $\tau$ does not provide benefits because the student network cannot absorb knowledge provided by the teacher network.

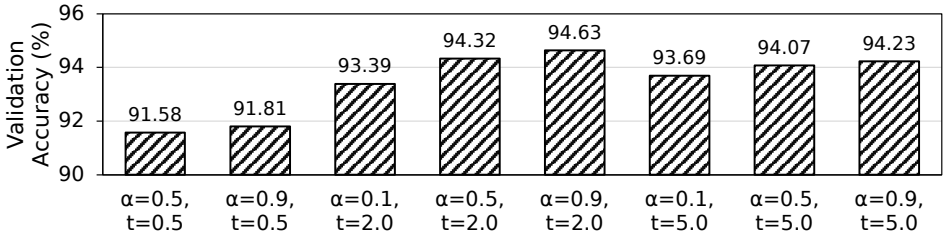


**Figure S2: Sensitivity of `SkipClip` to hyper-parameters alpha ($\alpha$) and temperature ($\tau$).**

## S3. Comparison to More Accurate Basecallers

Our goal is to make basecalling highly efficient and fast by building the first framework for specializing and optimizing machine learning-based basecaller. Currently, we focus on CNN-based basecallers because: (1) they are the most widely used basecallers, and (2) the fundamental multiply-accumulate (MAC) operation in a CNN model is amenable to hardware acceleration, unlike the operations in RNN-based basecallers. Guppy's super high accuracy (`Guppy-sup`) model is an RNN-based basecaller that provides more accuracy than `Guppy-fast` at the expense of a much larger model. We compare the overall basecalling performance of `RUBICALL` with that of the baseline basecallers in terms of basecalling accuracy, model parameters, and model size in Figure S3(a), S3(b), and S3(c), respectively.

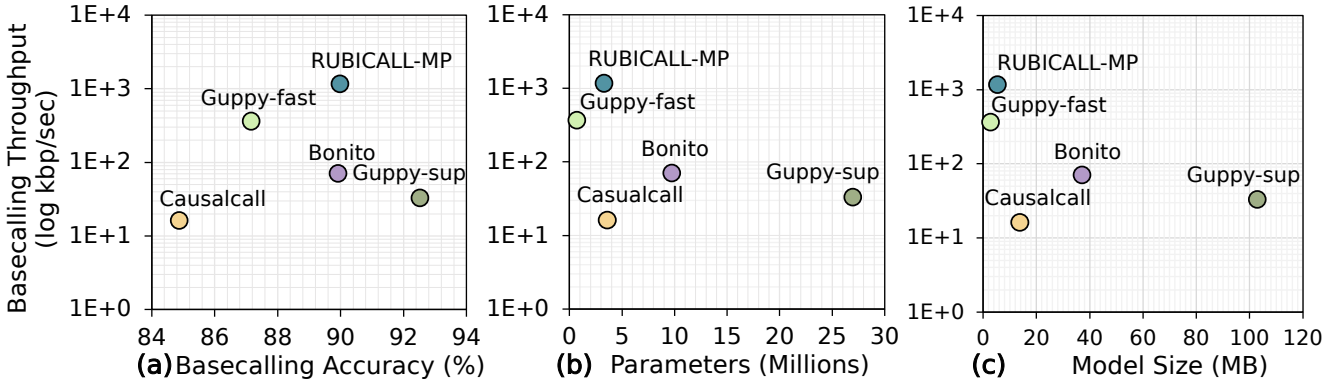


**Figure S3: Comparison of average basecalling throughput for `RUBICALL-MP` with baseline basecaller in terms of: (a) average basecalling accuracy, (b) model parameters, and (b) model size.**

**Table S1: Comparison of `RUBICALL-MP` with baseline basecallers in terms of basecalling throughput, basecalling accuracy, parameters, and model size. For basecalling throughput and basecalling accuracy, we report average (Avg.), minimum (Min.), maximum (Max.), 25th percentile (25th %tile), and 75th percentile (75th %tile) values for all the basecallers.**

| Basecaller | Architecture | Precision | Basecalling Throughput (kbp/sec) | | | | | Basecalling Accuracy (%) | | | | | Parameters | Model Size (MB) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Avg. | Min. | Max. | 25th %tile | 75th %tile | Avg. | Min. | Max. | 25th %tile | 75th %tile | – | – |
| `Causalcall` | CNN | FP16/FP32 | 16.06 | 9.04 | 22.16 | 12.45 | 18.86 | 84.88 | 82.7 | 86.42 | 83.57 | 86.17 | 3,589,893 | 13.69 |
| `Guppy-sup` | RNN | FP16/FP32 | 32.89 | 17.43 | 42.7 | 29.75 | 36.98 | **92.53** | **90.61** | **95.95** | **91.43** | **93.72** | 26,992,744 | 103.03 |
| `Guppy-fast` | RNN | FP16/FP32 | 364.63 | 54.32 | 587.53 | 260.98 | 490.8 | 87.16 | 82.53 | 91.39 | 86.25 | 88.51 | 730,344 | 2.79 |
| `Bonito` | CNN | FP16/FP32 | 70.36 | 37.41 | 93.01 | 57.49 | 82.15 | 89.92 | 87.99 | 93.75 | 88.62 | 91.15 | 9,738,573 | 37.15 |
| `RUBICALL-MP` | CNN | Mixed-Precision | **1165.21** | **392.03** | **2293.95** | **576.16** | **1604.91** | 89.98 | 86.59 | 93.64 | 88.97 | 91.41 | 3,314,578 | 5.36 |

In addition to our previous observations from Figure 9, we make three new observations from Figure S3 and Table S1. First, `RUBICALL-MP` has 35.42× the performance of the highly-accurate `Guppy-sup`. Second, `Guppy-sup` uses 7.52×, 36.96×, 2.77×, and 8.14× model parameters leading to a model size of 7.53×, 36.93×, 2.77×, and 19.22× compared to `Causalcall`, `Guppy-fast`, `Bonito`, and `RUBICALL-MP`, respectively. Third, `Guppy-sup` is 5.37% more accurate than its throughout-optimized version, `Guppy-fast`, which provides 22.71× higher basecalling performance. We conclude that the high accuracy of a basecaller comes at a substantial cost in terms of lower throughput due to the higher number of model parameters and model size.