

**DEEPVCF: A PROKARYOTIC SNP VARIANT CALLER USING DEEP  
NEURAL NETWORKS**

---

A Thesis

Presented to the

Faculty of

San Diego State University

---

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

In

Bioinformatics & Medical Informatics

---

by

Troy Michael Sincomb

Spring 2021

**SAN DIEGO STATE UNIVERSITY**

The Undersigned Faculty Committee Approves the

Thesis of Troy Michael Sincomb:

DeepVCF: A Prokaryotic SNP Variant Caller Using Deep Neural Networks



---

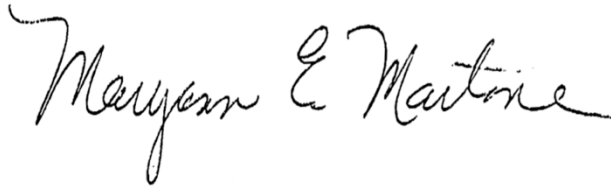
Scott T Kelley, Chair  
Bioinformatics and Medical Informatics Program



Parag Katira  
Department of Mechanical Engineering



Marina Kalyuzhnaya  
Department of Biology



Maryann E Martone  
Department of Neuroscience UCSD



Jeffrey S Grethe  
Department of Neuroscience UCSD

---

04/19/2021

Approval Date

Copyright © 2021  
by  
Troy Michael Sincomb  
All Rights Reserved

## **DEDICATION**

Dedicated to my wife, without whom sanity would have been lost.

I believe that at the end of the century the use of words and general educated opinion will have altered so much that one will be able to speak of machines thinking without expecting to be contradicted.

– Alan Turing

## ABSTRACT OF THE THESIS

DeepVCF: A Prokaryotic SNP Variant Caller Using Deep Neural  
Networks

by

Troy Michael Sincomb

Master of Science in Bioinformatics and Medical Informatics  
San Diego State University, 2021

Erroneous genomic data caused by sequencing platforms is unavoidable and can cause issues in downstream genomic analysis pipelines. These sequencing errors are significantly more prevalent when using Continuous Long Reads (CLR). Platforms such as PacBio and Nanopore creating CLRs are often used for sequencing Prokaryotic genomes by providing a way to keep any translocations that would have been fixed by a De Bruijn based assembly while also bridging long repeat regions. The drawback of CLRs is that they contain a high base error rate. Using Google's open source TensorFlow machine learning library in tandem with BioPython and Pysam, we created a Convolutional Neural Network based deep learning variant caller named DeepVCF that is shown here to outperform existing traditional Hidden Markov Model based variant callers, such as BCFtools, for erroneous genomic data caused by platform sequencing. DeepVCF accomplishes this by using the high-confidence variant dataset Genome in A Bottle (GIAB) as a baseline to prove model validity while training and testing on 10 Prokaryotic species datasets with variants created *in silico*. DeepVCF provides dynamic parameters for the user to alter the dimensions of the training tensors, heterozygous threshold for false positive training, minimum base quality, minimum read coverage, and complete control of Keras layers within the machine learning model. The current drawback of existing deep learning variant callers, such as Google's DeepVariant, are their fixed parameters that meet award winning accuracies with ideal training datasets from GIAB but underperform for less-than-optimal Prokaryotic variant datasets. By giving more control to the user, we show that DeepVCF can provide insight on erroneous genomic data to better determine novel variant calls with simple dynamic models.

## TABLE OF CONTENTS

	PAGE
ABSTRACT .....	vi
LIST OF TABLES .....	ix
LIST OF FIGURES .....	x
CHAPTER	
introduction.....	1
Methods .....	2
Preprocessing Alignments into a Pileup .....	2
Tensor Creation .....	5
Model.....	6
Validation Metrics .....	8
Variant Datasets.....	9
Usage .....	10
Future Release Notes .....	10
Results And Discussion .....	11
REFERENCES .....	18
A SUPPLEMENTAL MATERIALS .....	20

## LIST OF TABLES

	PAGE
Table 1. Keras model summary showing the dimensions and parameter totals in each layer. The parameters of value 0 means the number of possibilities has not changed in the network such as a convolutional layer altering the index value where dimensions are consistent from input to output after the kernel is used. ....	8
Table 2. GIAB hg38 chromosome 22 tests to find SNPs for homozygous alternative (hom_alt) and heterozygous (het) variant calls. DeepVCF scored considerably better than the baseline test of BCFtools when trying to see how the noisy alignment data affect traditional variant callers. ....	12
Table 3. List of bacteria strains used for training and testing variant callers. ....	20
Table 4. Complete Metric list for Heterozygous calls.....	22
Table 5. Complete Metric list for Homozygous Alternative calls.....	23



## LIST OF FIGURES

PAGE

- Figure 1. An alignment text view of the bases is converted into a pileup sum with dimensions 15 x 13. For a consistent view of the focused variant the pileup and tensor were transposed. The in-between pileup is constructed to save space in compression while also giving users an option to expand and create their own tensor matrices. In this window example the variant in question is not found to have a Boolean value of 1 from the broad internal variant caller and would not be accepted into the actual training. .... 4
- Figure 2. DeepVCFs default 9 layered CNN model with 3 CNN layers and 6 dense layers with a single input for the 2 outputs. The 2 outputs are for the multi-hot like encoding of the base prediction and the one-hot encoding for the genotype prediction. An example output for a heterozygous variant call for a predicted base set of Adenosine “A” and Thymine “T” would have a genotype output of list [1, 0, 0, 0] with a base prediction output of list [.5, 0, 0, .5]. .... 7
- Figure 3. The DeepVCF default model trained on hg38 chromosome 21 has a consistent reduction in loss for genotype and an increased accuracy for genotype calls for each epoch. The validation for base accuracy is unstable and suggests a possible replacement for the built-in Keras binary cross entropy loss function to stabilize the validation loss per epoch. In practice the base calls are leveraged using the genotype calls and will have a near exact relationship for accuracy as the genotype and why this issue was not addressed in the first release. .... 13
- Figure 4. Homozygous alternative SNV call catplot for F1 scores with a hue on species showing BCFtools performing better than DeepVCF for homozygous calls, but lower than chance for base errors of 10%. Hidden markov based models such as BCFtools takes calls at direct value and cannot discern error from real variants if enough base errors exist. .... 15
- Figure 5. Heterozygous alternative SNV call catplot for F1 scores showing BCFtools performed noticeably better for heterozygous calls when given noisy alignment data. The hue on species showed there was a noticeable difference in heterozygous calls between species suggesting the base errors are more concentrated in complex areas where bases lose predictable pattern. This is the reason GIAB includes bed files for focus areas to avoid hard to discern locations that would negatively impact training for a model. .... 15
- Figure 6. Pair plot of both 2% and 10 % base error for homozygous alternative SNP calls. General trend for BCFtools is a maintained precision and accuracy and sensitivity scores drop significantly for 10% base error. .... 16
- Figure 7. Pair plot of both 2% and 10 % base error for heterozygous SNP calls. General trend for BCFtools is a maintained precision and accuracy and sensitivity scores drop significantly for 10% base error. Similarly, to

homozygous alternative calls the precision is maintained indicating BCFtools get the value correct when it calls the correct position further supporting traditional variant calls using a hidden markov model to take variants at direct value without knowing context. ....	17
Figure 8. Example Model History for Prokaryotic datasets with the current datasets being from MRSA107 for this training history .....	21
Figure 9. Heterozygous statistical summary for base error rates of 2% for the Prokaryotic dataset. ....	24
Figure 10. Heterozygous statistical summary for base error rates of 10% for the Prokaryotic dataset. ....	25
Figure 11. Homozygous Alternative statistical summary for base error rates of 2% for the Prokaryotic dataset. ....	26
Figure 12. Homozygous Alternative statistical summary for base error rates of 10% for the Prokaryotic dataset. ....	27

## CHAPTER 1

### INTRODUCTION

Rapid advancements in next-generation sequencing (NGS) and Third generation sequencing have produced an exponentially growing amount of publicly available genome datasets. There have been parallel advancements within machine learning that could take advantage of the amount of genomic data available, but the variants between genomes are notoriously sparse with often poor documentation. The lack of variant datasets makes it difficult to find variants if there are significant platform sequencing errors that produce false positive variant calls in focus regions. This also poses an issue when using a deep learning model to predict variant calls if there are not enough high-confident variant calls to train the model. Machine learning cannot take full advantage of the vast amount of publicly available genomes without a baseline of high-confidence variants as a substrate. Lacking a gold-standard variant collection has led to the creation of a public-private academic consortium named Genome In A Bottle (GIAB) where high-confidence (or “truth”) variant calls can be publicly downloaded for human genomes (Zook et al., 2016). The cross platform high-confidence GIAB variant datasets used here were created with protocols found in the readme [https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/NA12878\\_HG001/NISTv3.3.2/README\\_NISTv3.3.2.txt](https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/NA12878_HG001/NISTv3.3.2/README_NISTv3.3.2.txt).

The creation of the GIAB dataset has highlighted limitations with traditional variant callers using a Hidden Markov based model where noisy data populated with indels can significantly affect performance. Limitations of traditional variant callers were verified with PrecisionFDA's Truth Variant Challenge where Google's DeepVariant deep learning model had a 98-99% accuracy for the high-confidence variant calls within the GIAB dataset (Poplin, 2018). However, this model was trained with and for Eukaryotic variant calling and has been shown to underperform with Prokaryotic genomes compared to traditional variant callers using the model provided (Bush et al., 2020). DeepVariant, as of writing this paper,

does not provide a model trained for Prokaryotic variant calls. This is due to the current lack of a gold standard GIAB variant dataset equivalent prokaryotes that could be used as a training set for any publicly available Prokaryotic species. It has also been shown that Prokaryotic genomes produce a high set of variability between overlapping variants found amongst commonly used variant callers due to significant differences in the genomes between species (Bush et al., 2020). This means that there is a noticeable increase of unique sets of sequences between Prokaryotes than there are between Eukaryotes. Thus, a simpler more dynamic model might be more appropriate for smaller independent training sets to complement the wider range of genomic diversity found in Prokaryotes.

Our deep learning variant caller DeepVCF explores this approach of using a simpler dynamic model that can have internal parameters optimized for a specific Prokaryotic species. It is written using the Python programming language with a complementary C language script for matrix initializations to help with memory allocation for reduced runtime. DeepVCF imports the following third-party tools: a sequence parser BioPython (Cock et al., 2009), an alignment parser Pysam (Li et al., 2009), the built-in Keras library within TensorFlow (Abadi et al., 2015), the high-level matrices toolkit Pandas (“Pandas”, 2020), and the lower-level matrices toolkit NumPy (Harris et al., 2020). The plots using the pandas dataframe as an input were generated with the toolkit Seaborn (Waskom, 2021). The DeepVCF variant caller tool is hosted on GitHub (“DeepVCF,” 2021) as an open-source project under the MIT license and is currently under active development.

## METHODS

### Alignment Preprocessing

Alignment files are parsed using the Pysam AlignmentFile class and iterated through using the `get_alignment_pairs` method to pull each alignment. Transformation of the alignments into a pileup should minimize the data loss that could be associated with transforming the alignments into an image. Using Pysam reduced the amount of time creating the DeepVCFs pileup source code while also giving an optimized, already tested, library to use as the backbone when creating the tensor. With each alignment meeting a minimum coverage of 50%, the custom offset sum pileup was used to gather the complete reference alignment length. The resulting matrix dimension is  $[N \times 12]$  where  $N$  is the reference

segment length used. Each NumPy list of 12 elements were to keep track of the 3 independent sums of 4 bases ACGT position in that respective order. The first 4 sum ACGT positions are to keep track of the bases supporting the reference, the next 4 are to keep track of every ACGT positions of the query reads aligned to that pileup column site, and the last 4 sum ACGT positions are to keep track of insertions from the query reads aligned. These 3 segments will be separated for the tensor creation later one, but are stored in this one array to increase compression efficiency and reduce initiation overhead of NumPy arrays. After we have the  $N \times 12$  pileup, we filter each second 4 summed base position representing the query sequence bases and run it through a crude variant caller that gives all variant candidates that can be determined by the actual model variant caller. If the site is a candidate for a single nucleotide polymorphism (SNP), we prepend a 1 to that pileup segment and if the site is not a SNP candidate a 0 is prepended to the pileup segment. The final NumPy pileup array has dimensions  $[N \times 13]$ . A crude variant caller is implemented as an initial filter to remove any hard to predict variant calls that will negatively impact model training while reducing incorrect variant calls from edge cases. The final  $N \times 13$  matrix is optionally saved in a compressed NumPy file with the .npy extension to be used as a cache file in case there are any parameter changes among the preprocessing steps to reduce testing runtime.



## Tensor Creation

The [N x 13] NumPy matrix is sliced into 4 variables in their respective order: SNP candidate filter, reference, query alignment per reference position, and query alignment insertions per reference position. We will be addressing each segment by their variable names listed below for the remainder of the paper.

```
snp_candidate = pileup[ : , 0 ]
reference = pileup[ : , 1:5 ]
query_alignment = pileup[ : , 5:9 ]
query_alignment_insertions = [ : , 9:13 ]
```

The default tensor dimensions are a [15 x 4 x 3] tensor made up of 3 matrices with 15 columns and 4 rows. The 15 columns are denoted as a window for the DNA segment of interest with the variant call position directly in the middle of that window. The 4 rows of each matrix represent the sum of the DNA bases ACGT, in that order, where each of the 3 matrices serve as a separate training goal for the model. The first matrix supports the reference variant, the second matrix supports the query variant, and the third being to support the query insertions.

The first matrix is created to support the reference sequence by taking the integer sum of the query\_alignment and then using that sum as the reference value in the position representing the reference base. For example, if we just take a single pileup site with the reference base as a Thymine base “T” it would be represented as [0, 0, 0, 1] and the query\_alignment for that site has 8 Adenosine bases “A” and 2 Thymine bases “T” it would be represented as [8, 0, 0, 2]. The sum of the query\_alignment array would be 10 and then placed into the position representing the reference base with a resulting pileup of [0, 0, 0, 10]. 14 more of these pileup computations make up the window that forms the first matrix of the tensor that supports the reference variant.

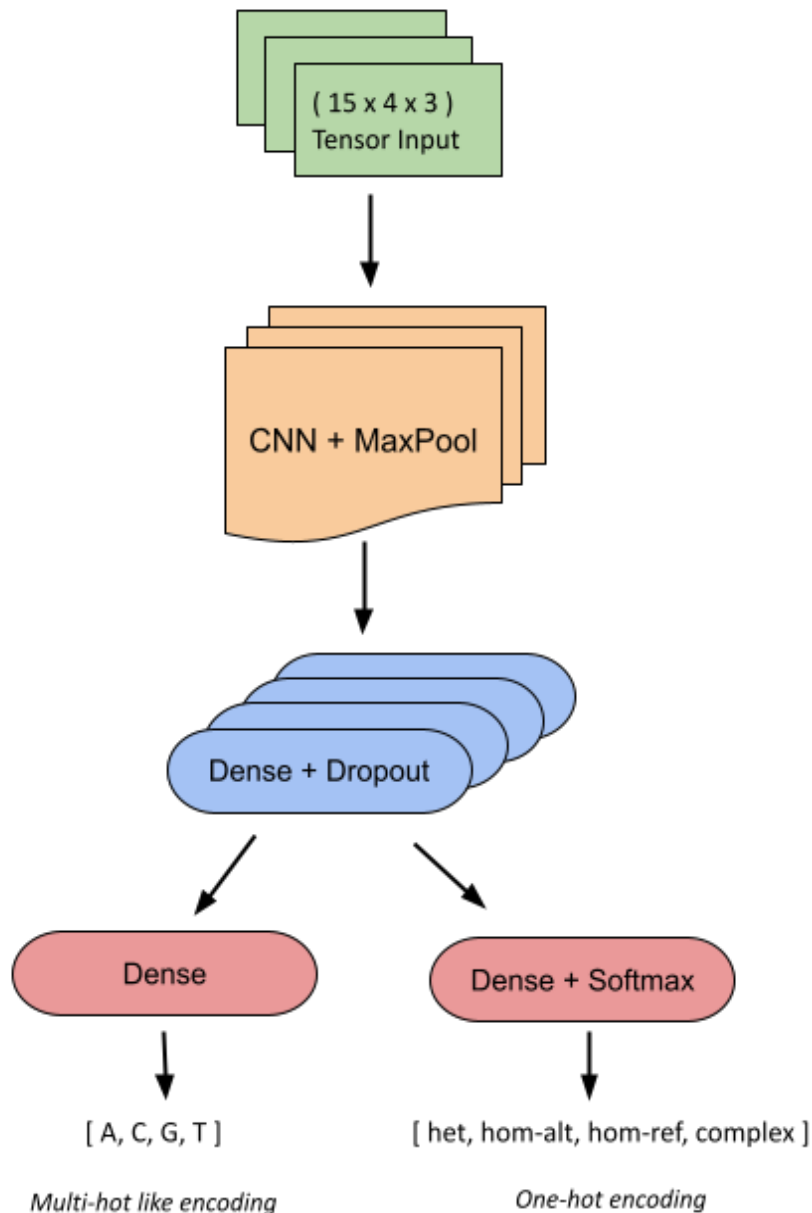
The next 2 matrices that make up a tensor are not as complicated to create. The second matrix to support the query variant is computed by using the Numpy subtraction method to broadcast the subtraction between the newly created matrix supporting the reference variant from the query\_alignment. The resulting second matrix will support the

base query alignment variants. The third matrix is the same as the second, but with the query\_alignment and query\_alignment\_insertions first added together before being subtracted by the first matrix of the tensor with the goal of accurately categorizing complex indel sites. The final tensor input for the model has a 15 x 4 x 3 shape, but the window size of 15 can be changed and each tensor can be separately omitted.

## **Model**

The default model provided for DeepVCF with 9 layers is considered a complex model due to it having 2 outputs, one for base calling and the other being for genotype prediction. The first 3 layers are Convolutional Neural Network (CNN) layers each with an associated Max Pooling layer. After the conventional flattening for dense layers, 6 dense layers are used with their associated dropout layers to avoid overtraining. The final 2 dense layers bifurcate to get the 2 outputs. The genotype dense layer is put through a SoftMax layer to better normalize the genotype output. A SoftMax is not used for the base output because the dense layer for the base output used a sigmoid activation. Every neural layer except for this dense layer for the base call uses an ELU activation. When constructing the DeepVCF default model, the activation ELU was chosen over the more widely used RELU activation. In tuner trial and error, RELU performed notably worse in all validation sets. This is most likely due to the relatively small tensor matrices being used for the CNN model and avoids the dead RELU issue where components of the network are most likely never updated to a new value. Default fitting parameters for the model will be ADAM as the optimizer and the built-in loss functions binary\_crossentropy and categorical\_crossentropy within Keras for the base and genotype outputs respectively.





**Figure 2.** DeepVCFs default 9 layered CNN model with 3 CNN layers and 6 dense layers with a single input for the 2 outputs. The 2 outputs are for the multi-hot like encoding of the base prediction and the one-hot encoding for the genotype prediction. An example output for a heterozygous variant call for a predicted base set of Adenosine “A” and Thymine “T” would have a genotype output of list [1, 0, 0, 0] with a base prediction output of list [.5, 0, 0, .5].

**Table 1. Keras model summary showing the dimensions and parameter totals in each layer. The parameters where the value is 0 means the number of possibilities has not changed in the network. Examples of this can be seen in each convolutional layer altering the index value where dimensions are consistent from input to output after the kernel is used.**

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 15, 4, 3)]	0	
conv2d (Conv2D)	(None, 15, 4, 16)	208	input_1[0][0]
max_pooling2d (MaxPooling2D)	(None, 11, 4, 16)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 11, 4, 32)	4128	max_pooling2d[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 8, 4, 32)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 8, 4, 48)	18480	max_pooling2d_1[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 6, 4, 48)	0	conv2d_2[0][0]
flatten (Flatten)	(None, 1152)	0	max_pooling2d_2[0][0]
dense_1 (Dense)	(None, 168)	193704	flatten[0][0]
dropout_1 (Dropout)	(None, 168)	0	dense_1[0][0]
dense_2 (Dense)	(None, 84)	14196	dropout_1[0][0]
dropout_2 (Dropout)	(None, 84)	0	dense_2[0][0]
dense_3 (Dense)	(None, 42)	3570	dropout_2[0][0]
dropout_3 (Dropout)	(None, 42)	0	dense_3[0][0]
dense_4 (Dense)	(None, 4)	172	dropout_3[0][0]
base (Dense)	(None, 4)	172	dropout_3[0][0]
genotype (Softmax)	(None, 4)	0	dense_4[0][0]
Total params: 234,630			
Trainable params: 234,630			
Non-trainable params: 0			

## Validation Metrics

The graph library to plot the validation metrics of the models is Seaborn. Seaborn accepts the resulting VCF Pandas DataFrame directly, allowing for quick and intricate plots from the metric data that can be accessed from the validation method in DeepVCF. The metrics performed are the standard metrics for machine learning models: precision (PPV), specificity, recall, accuracy and the F1 score.

$$Accuracy = \frac{TP + TF}{TP + TF + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN}$$

### Variant Datasets

For model validation we replicated Jason Chin’s VariantNet testing and training set by using the NA12878 PacBio read dataset aligned to GIABs GRCh38 genome with bwa mem (Li & Durbin, 2010) to test our model’s validity by training on chromosome 21 and testing on chromosome 22 (Chin, 2017). The HG001 datasets from GIAB that can be found here at [https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/NA12878\\_HG001/NISTv3.3.2/GRCh38](https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/NA12878_HG001/NISTv3.3.2/GRCh38) and the PacBio reads created by WUSTL that were aligned to GRCh38 can be found here <https://www.ncbi.nlm.nih.gov/bioproject/PRJNA323611>. Using the GIABs VCF file as the high-confidence variant set for the model to validate, we show the model can train towards a tangible confidence of having an F1 score of at least 80% using noisy diploid data with verified variant calls. After showing our model can train for variant calls, we tested our model against 10 different species of Prokaryotes with each species having a pair of variants where one variant is used to train the model on and the other to test for validation; a protocol replicated from the Bush Lab (Bush et al., 2020). The complete ascension list of each species trained and tested can be found under the Supplemental Materials section. To keep complete control over the SNPs we took the same Prokaryotic fasta file and generated the reads to align to it from itself. In order to simulate reads coming from variants of the same species we ran the read simulator DWGSIM (“DWGSIM,” 2021) with a high mutation rate of .5%. and a high error base rate of 10% among the reads to help replicate the same error rate shown in CLRs. The rest of the DWGSIM options were kept at default with 10% of the mutations being indels. Each bacterial species had approximately 25,000 mutations total introduced

with the .5% mutation rate. The list of options for DWGSIM and the defaults used can be found here <https://github.com/nh13/DWGSIM/wiki/Simulating-Reads-with-DWGSIM>. DWGSIM was chosen over dedicated long read simulating tools like SimLord (Stöcker et al., 2016) or PBsim2 (Ono, 2020) because it provided a mutation log to make the *in silico* variants traceable and usable for training. DWGSIM also had the added benefit of having the mutations log easily altered to be parsed as a VCF file making it ideal for our purposes.

## Usage

The needed parameters for DeepVCF to train the model are the paths to the reference file, alignment file, and the subsequent high-confidence Variant Calling Format (VCF).

```
from DeepVCF.core import DeepVCF

deepvcf = DeepVCF()
deepvcf.train( reference_file, alignment_file, vcf_file )
```

The dynamic parameters provide optimal tuning for the user to alter the dimensions of the training tensors, heterozygous threshold for false positive training, minimum base quality, minimum read coverage, and complete control of Keras layers within the machine learning model. Once the DeepVCF model is trained and validated, the same parameters are used to be able to use DeepVCF to provide a VCF file from a different set of reference and alignment files. To reduce confusion between training and using DeepVCF as a method call, we made a separate method “create\_vcf” to be the variant caller functionality that will take the same parameters as to initialize DeepVCF but returns a VCF file with DeepVCFs variant calls.

```
vcf_file = deepvcf.create_vcf( reference_file, alignment_file )
```

## Future Release Notes

During optimization of the pileup class due to it being the computational bottleneck of DeepVCF, a Cython script was used with hybrid C and Python syntax for a Numpy matrix initialization due to testing showing memory allocation slowdowns in the python GIL.

Further optimizations were made to avoid this entirely, but there is a logic issue with the Pysam PileupColumns class where different values are being returned compared to the Pysam AlignedSegment Class causing a noticeable decline in model training. When the speed-up issue is solved, it will reduce the pileup step to a fourth of the time and will be in the version 0.1.1 release.

## RESULTS AND DISCUSSION

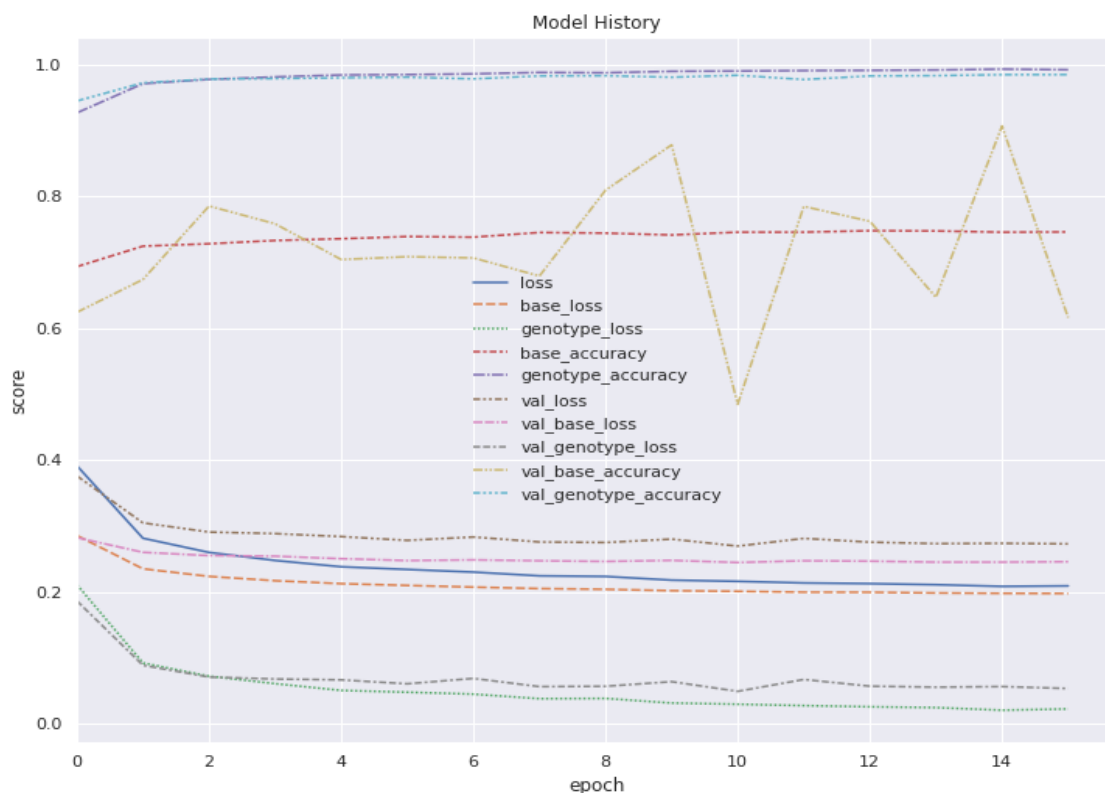
The performance found for DeepVCF among the validation datasets for hg38 chromosomes 21 and 22 had an average F1 score of 80.5% between the homozygous alternative and heterozygous single nucleotide variant (SNV) calls. The complete training on chromosome 21 and testing on chromosome 22 took 48 minutes and 37 seconds using a i5-10500 CPU, 32GBs of 3200hz memory, and a RTX 2080 desktop setup. A majority of that computational time is spent on each chromosome pileup tensor and leaves plenty of room for future optimizations. The resulting DeepVCF model produced an F1 score of 81% for homozygous alternative variant calls and an 79% for heterozygous variant calls. This shows the model is usable and the F1 score is well above the 25% chance from the 4 classes possible and meets an adequate baseline to show the default DeepVCF model can train on noisy high-confidence variant datasets and predict a viable SNV genotype and its respective base(s). However, in comparison to DeepVariants InceptionNet v3 model F1 score of 99%, DeepVariants scores highlight DeepVCFs limitations using a smaller CNN based model for large predictably noisy datasets such as the GIAB dataset. The BCFtools variant caller was also tested on chromosome 22 using the example pipeline, but had obtained an F1 score of 15.4% for homozygous alternative SNV calls and a 5% for heterozygous SNV calls while taking 47 hours, 22 minutes and 29 seconds to complete. Although BCFtools had a redeeming 99% PPV for homozygous alternative calls, this testing set shows how a traditional variant caller quickly reduces variant calling quality with noisy genomic datasets and the degradation of variant calling quality will be more apparent in the *in silico* SNPs datasets for the bacterial datasets.

With further testing of perfect datasets created *in silico*, the lower F1 score using the GIAB dataset was discovered to be from DeepVCFs current inability to handle and call indels in a diploid dataset. Similar to Jason Chin's VariantNets approach, we took all indels

and marked them as a “complex” feature, but unlike VariantNet we did not remove any variant calls from the truth variant call set to give a more robust metric of what DeepVCF should and should not be used for. Unlike the diploid dataset, the haploid variant calls created *in silico* using DWGSIM had upwards of an F1 between 98-99% depending on the Prokaryotic species.

**Table 2. GIAB hg38 chromosome 22 tests to find SNPs for homozygous alternative (hom\_alt) and heterozygous (het) variant calls. DeepVCF scored considerably better than the baseline test of BCFtools when trying to see how the noisy alignment data affect traditional variant callers.**

	Metric	DeepVCF_hom_alt	DeepVCF_het	BCFtools_hom_alt	BCFtools_het
0	Sensitivity	0.971285	0.908741	0.083543	0.974025
1	PPV	0.707775	0.700577	0.998927	0.026114
2	Accuracy	0.873678	0.738300	0.720906	0.028905
3	F1	0.818852	0.791196	0.154190	0.050864



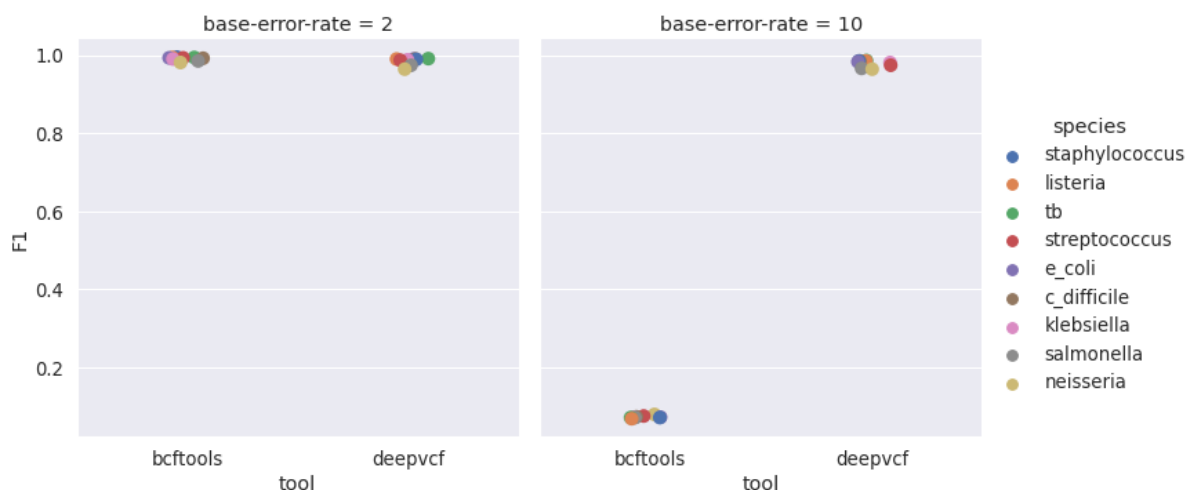
**Figure 3. The DeepVCF default model trained on hg38 chromosome 21 has a consistent reduction in loss for genotype and an increased accuracy for genotype calls for each epoch. The validation for base accuracy is unstable and suggests a possible replacement for the built-in Keras binary cross entropy loss function to stabilize the validation loss per epoch. In practice the base calls are leveraged using the genotype calls and will have a near exact relationship for accuracy as the genotype and why this issue was not addressed in the first release.**

The 10 *in silico* bacterial variant datasets were broken up into 4 SNV calling groups: 2% base errors for heterozygous, 2% base errors for homozygous alternatives, 10% base errors for heterozygous, and 10% base errors for homozygous alternatives (see Supplemental Materials for list of bacterial strains used). The base percent error rate and the mutation of 5% where the only 2 alignment alerting options given for the *in silico* tool DWGSIM and relied on the defaults for noisy indel data regarding probability of generation, length, and location. From the initial analysis of 2% error rate BCFtools outperformed DeepVCF tools with acquiring a mean F1 scores of 99.5% and 99.8% compared to DeepVCFs mean F1 scores of 99.2% and 99.1% for homozygous alternatives and heterozygous SNV calls in that order. However, when BCFtools was tested with 10% base errors to simulated CLR percent errors, BCFtools returned a mean F1 score of 8% for homozygous calls and a mean F1 score

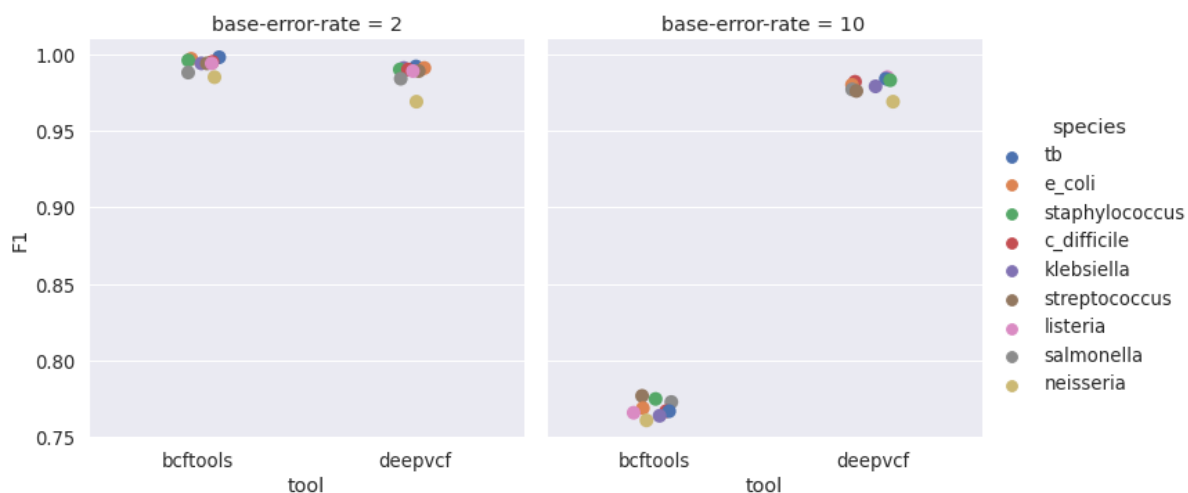
of 77% scores for heterozygous calls. DeepVCF still maintained consistent mean F1 scores of 98.6% and 98.5% for homozygous and heterozygous calls. The results confirm when given ideal datasets from an NGS platform a traditional Hidden Markov Model variant caller will perform as expected, but when a traditional variant caller similar to BCFtools is given erroneous sequence data the algorithm will not have context to decipher whether a base change is truly a variant or a platform sequencing error. DeepVCF having training on variant context will account for the base errors by leaning on the hidden features of the pileup window and maintain its effectiveness to call variants.

DeepVCF as a SNP variant caller shows practicality as a compliment application to make sure a traditional markov model based variant caller, such as BCFtools, will return adequate variant calls. BCFtools outperformed DeepVCF in perfect and predictable 2% base error alignments, but returned variant calls that were clustered base errors from a saturated 10% base error that is often seen from CLR datasets. DeepVCF could be used as a complementary variant directly by using it as a rough approximation of how many SNPs exist and compare a VCF output from a tool equivalent to BCFtools and see if the number of SNPs match up. Additionally, these results support those traditional variant calls that do not seek help with syntenic hybrid-based approaches should not be used for genomes sequenced from platforms known for erroneous continuous long reads such as the CLRs from PacBio. For complete metrics and figures please see Supplemental Materials.

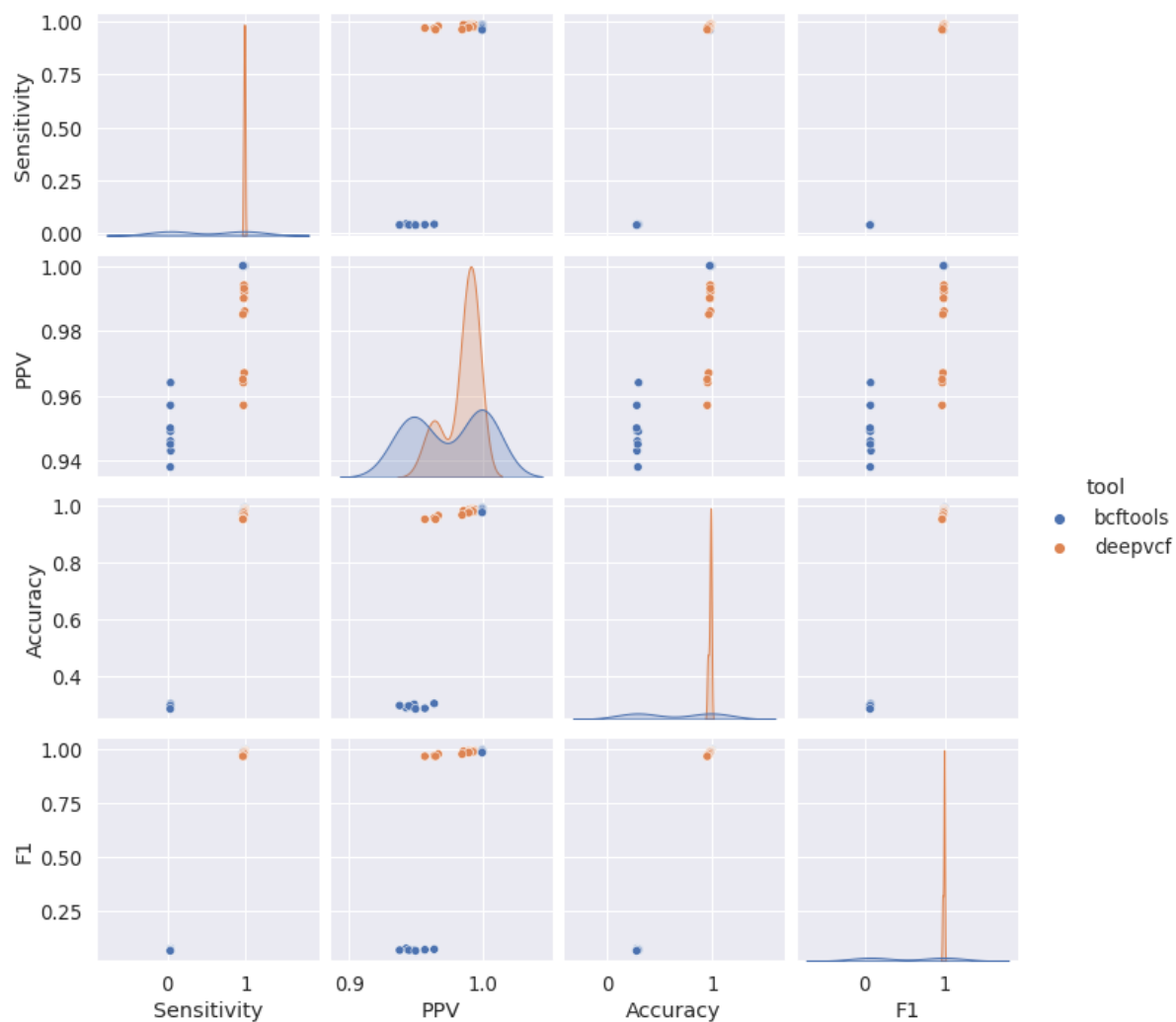




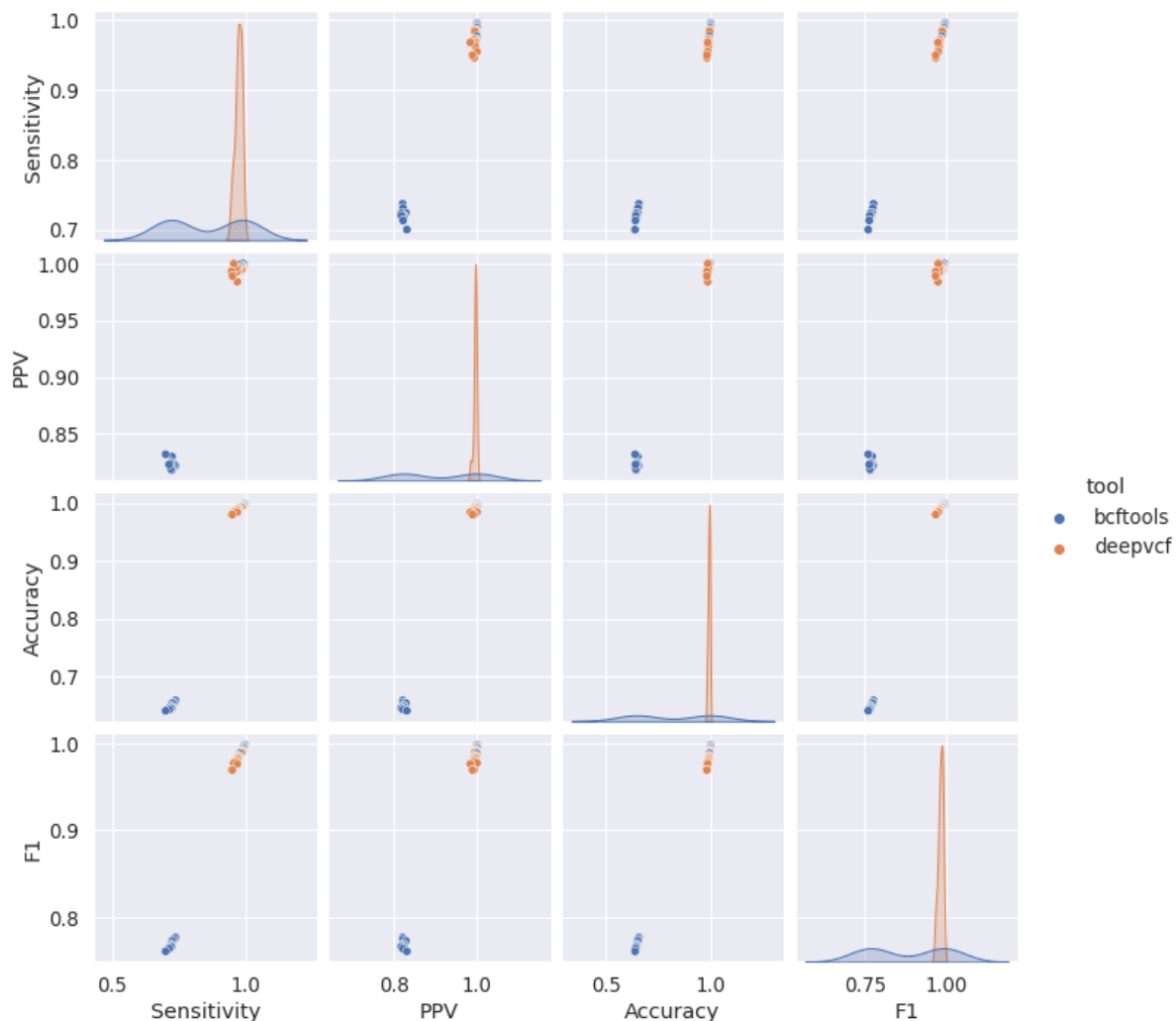
**Figure 4. Homozygous alternative SNV call catplot for F1 scores with a hue on species showing BCFtools performing better than DeepVCF for homozygous calls, but lower than chance for base errors of 10%. Hidden Markov based models such as BCFtools takes calls at direct value and cannot discern error from real variants if enough base errors exist.**



**Figure 5. Heterozygous alternative SNV call catplot for F1 scores showing BCFtools performed noticeably better for heterozygous calls when given noisy alignment data. The hue on species showed there was a noticeable difference in heterozygous calls between species suggesting the base errors are more concentrated in complex areas where bases lose predictable pattern. This is the reason GIAB includes bed files for focus areas to avoid hard to discern locations that would negatively impact training for a model.**



**Figure 6. Pair plot of both 2% and 10 % base error for homozygous alternative SNP calls. General trend for BCFtools is a maintained precision and accuracy and sensitivity scores drop significantly for 10% base error.**



**Figure 7. Pair plot of both 2% and 10 % base error for heterozygous SNP calls. General trend for BCFtools is a maintained precision and accuracy and sensitivity scores drop significantly for 10% base error. Similarly, to homozygous alternative calls the precision is maintained indicating BCFtools get the value correct when it calls the correct position further supporting traditional variant calls using a Hidden Markov model to take variants at direct value without knowing context.**

## REFERENCES

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E. & Chen, Z. TensorFlow: large-scale machine learning on heterogeneous systems (2015). Retrieved from <https://arxiv.org/abs/1603.04467>
- Cock PA, Antao T, Chang JT, Chapman BA, Cox CJ, Dalke A, Friedberg I, Hamelryck T, Kauff F, Wilczynski B and de Hoon MJL (2009) Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25, 1422-1423
- The Pandas Development Team, Pandas 1.1.0. Available at <https://pandas.pydata.org/>. <http://dx.doi.org/10.5281/zenodo.3964380>. (Accessed 19 August 2020).
- Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., & 1000 Genome Project Data Processing Subgroup (2009). The Sequence Alignment/Map format and SAMtools. *Bioinformatics (Oxford, England)*, 25(16), 2078–2079. <https://doi.org/10.1093/bioinformatics/btp352>
- Sincomb, Troy. (2021). DeepVCF: A variant caller using deep neural networks (Version 0.1.0dev) [Software]. Retrieved from <https://github.com/tmsincomb/DeepVCF>
- Homer, Nils. (2017). DWGSIM: Whole Genome Simulator for Next-Generation Sequencing (Version 0.1.12) [Software]. Retrieved from <https://github.com/nh13/DWGSIM>
- Bush, S. J., Foster, D., Eyre, D. W., Clark, E. L., De Maio, N., Shaw, L. P., ... Walker, A. S. (2020). Genomic diversity affects the accuracy of bacterial single-nucleotide polymorphism-calling pipelines. *GigaScience*, 9(2). <https://doi.org/10.1093/gigascience/giaa007>
- Chin, J. (2017, July 16). Simple Convolutional Neural Network for Genomic Variant Calling with TensorFlow [web log]. <https://towardsdatascience.com/simple-convolutional-neural-network-for-genomic-variant-calling-with-tensorflow-c085dbc2026f>.
- Poplin, R., Chang, P.-C., Alexander, D., Schwartz, S., Colthurst, T., Ku, A., ... DePristo, M. A. (2018). A universal SNP and small-indel variant caller using deep neural networks. *Nature Biotechnology*, 36(10), 983–987. <https://doi.org/10.1038/nbt.4235>
- Zook, J., Catoe, D., McDaniel, J. *et al.* Extensive sequencing of seven human genomes to characterize benchmark reference materials. *Sci Data* 3, 160025 (2016). <https://doi.org/10.1038/sdata.2016.25>
- The Pandas Development Team, Pandas 1.1.0 (2020). Available at <https://pandas.pydata.org/>. <http://dx.doi.org/10.5281/zenodo.3964380>. (accessed 19 August 2020).
- Waskom, M. L., (2021). seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021, <https://doi.org/10.21105/joss.03021>
- Harris, C.R., Millman, K.J., van der Walt, S.J. *et al.* Array programming with NumPy. *Nature* 585, 357–362 (2020). <https://doi.org/10.1038/s41586-020-2649-2>
- Stöcker, B. K., Köster, J., & Rahmann, S. (2016). SimLoRD: Simulation of Long Read Data. *Bioinformatics*, 32(17), 2704–2706. <https://doi.org/10.1093/bioinformatics/btw286>

Ono, Y., Asai, K., & Hamada, M. (2020). PBSIM2: a simulator for long-read sequencers with a novel generative model of quality scores. *Bioinformatics*.  
<https://doi.org/10.1093/bioinformatics/btaa835>

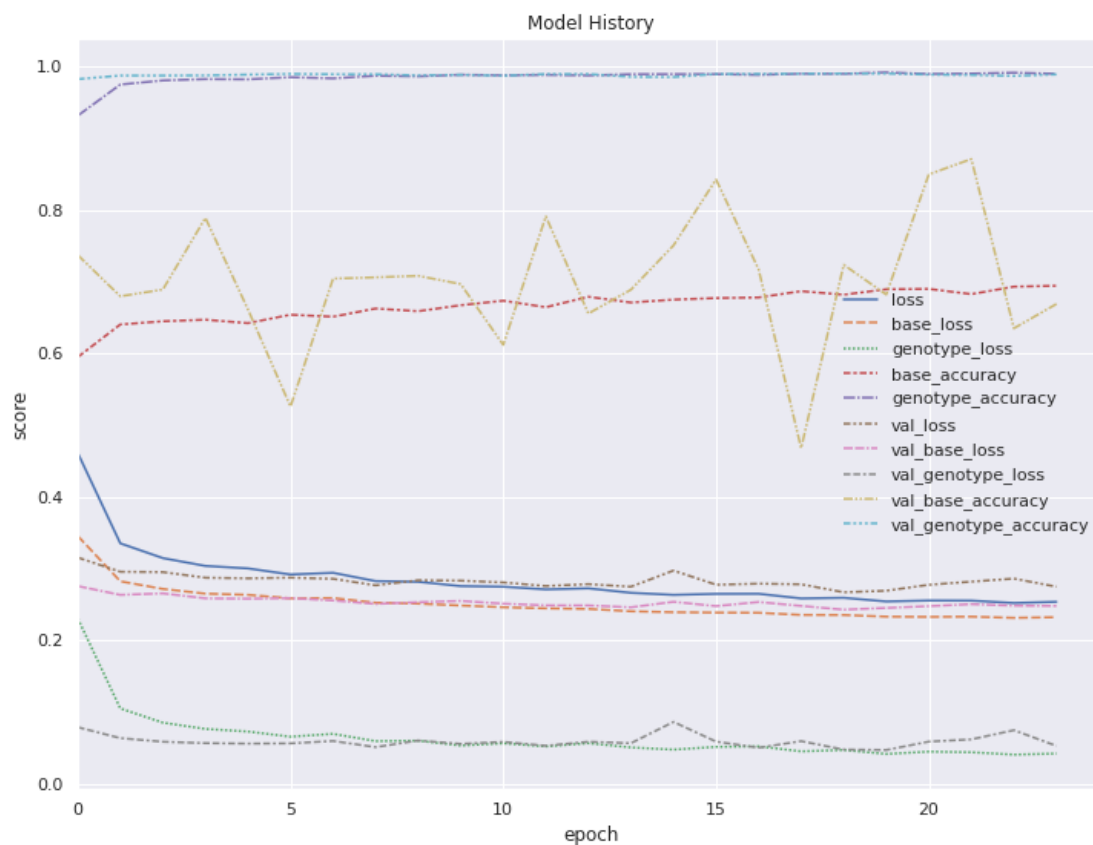
Li, H., & Durbin, R. (2010). Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics*, 26(5), 589–595.  
<https://doi.org/10.1093/bioinformatics/btp698>

## APPENDIX

### SUPPLEMENTAL MATERIALS

**Table 3. List of bacteria strains used for training and testing variant callers.**

	Strain Trained On	Strain Tested On
<i>Clostridioides difficile</i>	FDAARGOS_267	R0104a
<i>Escherichia coli</i>	ATCC25922	GCF_000005845
<i>Klebsiella pneumoniae</i>	AR_0087	ATCC43816KPPR1
<i>Listeria monocytogenes</i>	ATCC51775	GCF_000196035
<i>Mycobacterium tuberculosis</i>	CDC1551	GCF_000195955
<i>Neisseria gonorrhoeae</i>	FDAARGOS_205	GCF_000006845
<i>Salmonella enterica</i>	USDA-ARS-USMARC-1728	ATCC10720
<i>Shigella dysenteriae</i>	1617	GCF_000012005
<i>Staphylococcus aureus</i>	MRSA107	GCF_000013425
<i>Streptococcus pneumoniae</i>	G54	GCF_000007045



**Figure 8. Example Model History for Prokaryotic datasets with the current datasets being from MRSA107 for this training history**

Table 4. Complete Metric list for Heterozygous calls

	tool	species	base-error-rate	Sensitivity	PPV	Accuracy	F1
0	bcftools	tb	2	0.996000	1.000000	0.998000	0.998000
1	bcftools	e_coli	2	0.994000	0.999000	0.998000	0.997000
2	bcftools	staphylococcus	2	0.993000	1.000000	0.998000	0.996000
3	bcftools	c_difficile	2	0.991000	1.000000	0.997000	0.995000
4	bcftools	klebsiella	2	0.989000	0.999000	0.996000	0.994000
5	bcftools	streptococcus	2	0.988000	1.000000	0.996000	0.994000
6	bcftools	listeria	2	0.989000	1.000000	0.996000	0.994000
7	deepvcf	tb	2	0.986000	0.998000	0.995000	0.992000
8	deepvcf	e_coli	2	0.985000	0.998000	0.994000	0.991000
9	deepvcf	klebsiella	2	0.983000	0.998000	0.994000	0.991000
10	deepvcf	staphylococcus	2	0.986000	0.994000	0.993000	0.990000
11	deepvcf	c_difficile	2	0.981000	0.998000	0.993000	0.990000
12	deepvcf	streptococcus	2	0.983000	0.995000	0.993000	0.989000
13	deepvcf	listeria	2	0.984000	0.995000	0.993000	0.989000
14	bcftools	salmonella	2	0.977000	0.999000	0.992000	0.988000
15	bcftools	neisseria	2	0.972000	0.999000	0.991000	0.985000
16	deepvcf	listeria	10	0.973000	0.996000	0.990000	0.985000
17	deepvcf	salmonella	2	0.971000	0.996000	0.989000	0.984000
18	deepvcf	tb	10	0.971000	0.997000	0.989000	0.984000
19	deepvcf	staphylococcus	10	0.972000	0.994000	0.988000	0.983000
20	deepvcf	c_difficile	10	0.969000	0.996000	0.989000	0.982000
21	deepvcf	e_coli	10	0.967000	0.993000	0.987000	0.980000
22	deepvcf	klebsiella	10	0.962000	0.996000	0.986000	0.979000
23	deepvcf	salmonella	10	0.955000	1.000000	0.984000	0.977000
24	deepvcf	streptococcus	10	0.968000	0.984000	0.984000	0.976000
25	deepvcf	neisseria	10	0.946000	0.993000	0.980000	0.969000
26	deepvcf	neisseria	2	0.950000	0.989000	0.980000	0.969000
27	bcftools	streptococcus	10	0.737000	0.822000	0.658000	0.777000
28	bcftools	staphylococcus	10	0.731000	0.823000	0.654000	0.775000
29	bcftools	salmonella	10	0.724000	0.830000	0.653000	0.773000
30	bcftools	e_coli	10	0.720000	0.825000	0.648000	0.769000
31	bcftools	c_difficile	10	0.718000	0.824000	0.647000	0.767000
32	bcftools	tb	10	0.723000	0.818000	0.645000	0.767000
33	bcftools	listeria	10	0.720000	0.819000	0.644000	0.766000
34	bcftools	klebsiella	10	0.713000	0.823000	0.642000	0.764000
35	bcftools	neisseria	10	0.700000	0.832000	0.640000	0.761000



Table 5. Complete Metric list for Homozygous Alternative calls

	tool	species	base-error-rate	Sensitivity	PPV	Accuracy	F1
0	bcftools	staphylococcus	2	0.989000	1.000000	0.993000	0.995000
1	bcftools	listeria	2	0.988000	1.000000	0.993000	0.994000
2	bcftools	tb	2	0.988000	1.000000	0.992000	0.994000
3	bcftools	streptococcus	2	0.987000	1.000000	0.991000	0.993000
4	bcftools	e_coli	2	0.987000	1.000000	0.992000	0.993000
5	bcftools	c_difficile	2	0.984000	1.000000	0.990000	0.992000
6	deepvcf	tb	2	0.991000	0.992000	0.989000	0.991000
7	deepvcf	e_coli	2	0.989000	0.992000	0.987000	0.990000
8	deepvcf	c_difficile	2	0.986000	0.993000	0.986000	0.990000
9	bcftools	klebsiella	2	0.979000	1.000000	0.987000	0.990000
10	deepvcf	listeria	2	0.986000	0.994000	0.987000	0.990000
11	deepvcf	staphylococcus	2	0.987000	0.990000	0.985000	0.989000
12	deepvcf	klebsiella	2	0.984000	0.992000	0.984000	0.988000
13	deepvcf	streptococcus	2	0.987000	0.986000	0.982000	0.987000
14	deepvcf	tb	10	0.980000	0.992000	0.982000	0.986000
15	deepvcf	listeria	10	0.976000	0.994000	0.980000	0.985000
16	deepvcf	c_difficile	10	0.976000	0.994000	0.979000	0.985000
17	bcftools	salmonella	2	0.971000	1.000000	0.981000	0.985000
18	deepvcf	staphylococcus	10	0.978000	0.993000	0.981000	0.985000
19	deepvcf	e_coli	10	0.976000	0.990000	0.977000	0.983000
20	bcftools	neisseria	2	0.962000	1.000000	0.976000	0.981000
21	deepvcf	klebsiella	10	0.973000	0.990000	0.975000	0.981000
22	deepvcf	streptococcus	10	0.964000	0.985000	0.967000	0.974000
23	deepvcf	salmonella	2	0.980000	0.967000	0.965000	0.974000
24	deepvcf	salmonella	10	0.969000	0.964000	0.956000	0.966000
25	deepvcf	neisseria	2	0.971000	0.957000	0.952000	0.964000
26	deepvcf	neisseria	10	0.963000	0.965000	0.952000	0.964000
27	bcftools	neisseria	10	0.042000	0.943000	0.287000	0.080000
28	bcftools	streptococcus	10	0.040000	0.964000	0.302000	0.076000
29	bcftools	c_difficile	10	0.038000	0.957000	0.285000	0.074000
30	bcftools	e_coli	10	0.038000	0.946000	0.290000	0.073000
31	bcftools	klebsiella	10	0.038000	0.949000	0.287000	0.073000
32	bcftools	salmonella	10	0.037000	0.938000	0.295000	0.072000
33	bcftools	staphylococcus	10	0.038000	0.949000	0.299000	0.072000
34	bcftools	tb	10	0.037000	0.945000	0.294000	0.072000
35	bcftools	listeria	10	0.036000	0.950000	0.283000	0.069000

	ml_metric	stat_metric	bcftools	deepvcf
<b>0</b>	Sensitivity	mean	0.987667	0.978778
<b>1</b>	Sensitivity	min	0.972000	0.950000
<b>2</b>	Sensitivity	max	0.996000	0.986000
<b>3</b>	PPV	mean	0.999556	0.995667
<b>4</b>	PPV	min	0.999000	0.989000
<b>5</b>	PPV	max	1.000000	0.998000
<b>6</b>	Accuracy	mean	0.995778	0.991556
<b>7</b>	Accuracy	min	0.991000	0.980000
<b>8</b>	Accuracy	max	0.998000	0.995000
<b>9</b>	F1	mean	0.993444	0.987222
<b>10</b>	F1	min	0.985000	0.969000
<b>11</b>	F1	max	0.998000	0.992000

**Figure 9. Heterozygous statistical summary for base error rates of 2% for the Prokaryotic dataset.**

	ml_metric	stat_metric	bcftools	deepvcf
<b>0</b>	Sensitivity	mean	0.720667	0.964778
<b>1</b>	Sensitivity	min	0.700000	0.946000
<b>2</b>	Sensitivity	max	0.737000	0.973000
<b>3</b>	PPV	mean	0.824000	0.994333
<b>4</b>	PPV	min	0.818000	0.984000
<b>5</b>	PPV	max	0.832000	1.000000
<b>6</b>	Accuracy	mean	0.647889	0.986333
<b>7</b>	Accuracy	min	0.640000	0.980000
<b>8</b>	Accuracy	max	0.658000	0.990000
<b>9</b>	F1	mean	0.768778	0.979444
<b>10</b>	F1	min	0.761000	0.969000
<b>11</b>	F1	max	0.777000	0.985000

**Figure 10. Heterozygous statistical summary for base error rates of 10% for the Prokaryotic dataset.**

	ml_metric	stat_metric	bcftools	deepvcf
0	Sensitivity	mean	0.981667	0.984556
1	Sensitivity	min	0.962000	0.971000
2	Sensitivity	max	0.989000	0.991000
3	PPV	mean	1.000000	0.984778
4	PPV	min	1.000000	0.957000
5	PPV	max	1.000000	0.994000
6	Accuracy	mean	0.988333	0.979667
7	Accuracy	min	0.976000	0.952000
8	Accuracy	max	0.993000	0.989000
9	F1	mean	0.990778	0.984778
10	F1	min	0.981000	0.964000
11	F1	max	0.995000	0.991000

**Figure 11. Homozygous Alternative statistical summary for base error rates of 2% for the Prokaryotic dataset.**

	ml_metric	stat_metric	bcftools	deepvcf
<b>0</b>	Sensitivity	mean	0.038222	0.972778
<b>1</b>	Sensitivity	min	0.036000	0.963000
<b>2</b>	Sensitivity	max	0.042000	0.980000
<b>3</b>	PPV	mean	0.949000	0.985222
<b>4</b>	PPV	min	0.938000	0.964000
<b>5</b>	PPV	max	0.964000	0.994000
<b>6</b>	Accuracy	mean	0.291333	0.972111
<b>7</b>	Accuracy	min	0.283000	0.952000
<b>8</b>	Accuracy	max	0.302000	0.982000
<b>9</b>	F1	mean	0.073444	0.978778
<b>10</b>	F1	min	0.069000	0.964000
<b>11</b>	F1	max	0.080000	0.986000

**Figure 12. Homozygous Alternative statistical summary for base error rates of 10% for the Prokaryotic dataset.**