Supplemental Materials for DeepSimulator: a deep simulator for Nanopore sequencing

Yu Li[1], Renmin Han[1], Chongwei Bi[2], Mo Li[2], Sheng Wang*[1], and Xin Gao*[1]

[1]*King Abdullah University of Science and Technology (KAUST), Computational Bioscience Research Center (CBRC), Computer, Electrical and Mathematical Sciences and Engineering (CEMSE) Division, Thuwal, 23955-6900, Saudi Arabia*

[2]*King Abdullah University of Science and Technology (KAUST), Biological and Environmental Sciences and Engineering (BESE) Division, Thuwal, 23955-6900, Saudi Arabia*

*All correspondence should be addressed to Sheng Wang (sheng.wang@kaust.edu.sa) and Xin Gao (xin.gao@kaust.edu.sa).

S1: Related works

In this section, we first provide an introduction to the three existing statistic-based simulators, followed by a brief introduction to Bi-LSTM, which we used for building the context-dependent pore model.

ReadSim

ReadSim (Lee *et al.*, 2014), to our knowledge, is the first work that is able to generate long reads, following the read length distribution of PacBio or Nanopore data. When proposed, it was used to simulate PacBio data. Later on, people also used it for simulating the Nanopore data since there was no simulator devoted to simulating the Nanopore data exclusively for a very long time. In principle, this software is a fast and simple reads simulator. However, it was not designed for Nanopore specifically and has not been updated since 2014. Because Nanopore sequencing has been developed dramatically, ReadSim may have difficulty in mimicking the error distribution and insertion-deletion pattern of the most recent Nanopore sequencing protocol.

SiLiCO

SiLiCO (Baker *et al.*, 2016) was claimed to be the first open source software that is designed specifically to simulate Nanopore and PacBio data. Through analysis, they found that the length distribution of Nanopore data follows the gamma distribution. Based on the user-provided parameters for the distribution, SiLiCO would generate genomic coordinates stochastically, resulting in the simulated data. Besides, SiLiCO can be scaled up to a Monte-Carlo simulation. Although it can generate reads with desirable read lengths and coverage with high scalability, this software does not mimic the error distribution and insertion-deletion pattern of the Nanopore data.

NanoSim

NanoSim (Yang *et al.*, 2017), published earlier in 2017, is the first formally published software with the ability of both modeling the read length distribution and the basecalling error pattern of the Nanopore data. There are two stages of NanoSim. The first stage is read characterization, in which the user would provide a reference and a training read set. During this stage, the training read set is aligned to the reference using LAST (Frith *et al.*, 2010), resulting in an alignment file, from which a set of profiles, such as substitution rate and insertion-deletion rate, could be summarized. Thus, the parameters of the predefined distribution, modeling the mismatch, insertion and deletion, are learned. The next stage is the read generation stage. With the user-provided reference sequence, the reads based on the length distribution learned from the previous stage are extracted from the input sequence. After that, the errors, which are drawn from the statistical model and whose type is determined by a predefined Hidden Markov Model (HMM), are introduced to the extracted reads. This work makes great contribution to the Nanopore data simulation field. However, we should notice that as the Nanopore technology evolves, the predefined error model may no longer fit. What is worse, the basecalling algorithm has been changed dramatically since September 2017, and Albacore no longer uses events to perform basecalling. As a result, the predefined HMM may not fit the current read error type transition well.

Bi-LSTM

Bi-LSTM is a well-known recurrent neural network architecture (Boža *et al.*, 2017). For a given input sequence Z, the uni-directional LSTM, generating the hidden vector $\mathbf{h}_t$ at position t, can be represented in the following way:

$$
\begin{aligned}
f_t &= \sigma(W_f z_t + U_f h_{t-1} + b_f), \\
n_t &= \sigma(W_n z_t + U_n h_{t-1} + b_n), \\
o_t &= \sigma(W_o z_t + U_o h_{t-1} + b_o), \\
c_t &= f_t * c_{t-1} + n_t * \tanh(W_c z_t + U_c h_{t-1} + b_c), \\
\mathbf{h}_t &= o_t * \tanh(c_t),
\end{aligned} \tag{1}
$$

where f, n, and o represent the forget, input, and output gates, respectively. c is the cell memory vector. σ is the sigmoid function, and tanh is the hyperbolic tangent function. W, U, b are the model parameters.

Let $\mathbf{h}_{f,t}$ denote the hidden vector generated by the forward LSTM and $\mathbf{h}_{b,t}$ denote the one generated by the backward LSTM. Then $\mathbf{h}_{b,f,t}$ generated by Bi-LSTM is the concatenation of $\mathbf{h}_{f,t}$ and $\mathbf{h}_{b,t}$.

S2: Length distribution parameters

The parameters for the three different read length distributions are given in Table S1.

S3: Repeat time distribution

As shown in Figure S1 (A), it is not straightforward to use one predefined density function to fit such a distribution. Instead, we used a mixed distribution to fit it. Since if we ignore the value of 1, the remaining distribution could be easily fitted using an alpha distribution, we split the fitting into two steps. We first defined a ratio r, which specifies the ratio that belongs to the all 1 distribution, which means the remaining $1 - r$ should be drawn from the alpha distribution. Then we fitted the real data using the two-step distribution. Table S2 shows

Table S1: Parameters of the read length distribution. The definition of each parameter could be referred to Scipy document (https://docs.scipy.org/doc/scipy/reference/stats.html). In order to fit the third pattern, we ran an EM algorithm. Since the algorithm itself depends on the initialization, we reran the algorithm for 10,000 times and selected the best one as the final model, with the histogram interaction being the objective function. The final histogram interaction value is 0.8818.

Parameters	Exponential distribution	Beta distribution	Mixed Gamma distribution
a	-	1.778	6.369 & 1.676
b	-	7.893	0.538 & 0.229
loc	213.989	316.758	-
$scale$	6972.532	34191.257	-

Table S2: Parameters of the repeat time distribution. r means the ratio of the all 1 distribution in the mixed distribution. The definition of the remaining parameter could be referred to the Scipy document.

Parameters	r	a	loc	$scale$
Value	0.075	3.393	-7.645	50.874

the parameters of the two-step distribution. Figure S1 shows the comparison between the real distribution and the fitted distribution, which show similar pattern explicitly.

S4: Parameter manual of DeepSimulator

Table S3 shows all the parameters of DeepSimulator, which can be adjusted in the main.sh file by the users to simulate a variety of situations. Table S4 shows the parameters of the four sets of reads (DS(noise free), DS(high acc), DS(med

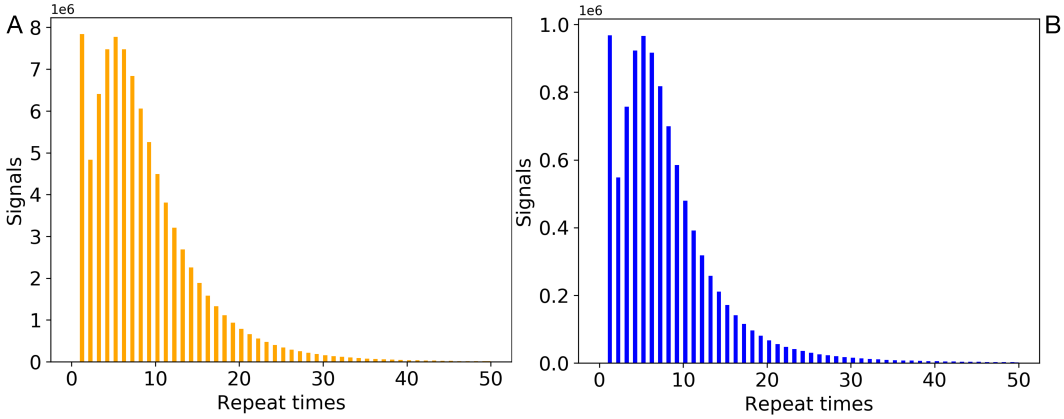

Figure S1: Comparison between the real repeat time distribution and the fitted distribution. (A) Histogram of the real data repeat times. (B) Histogram of 10,000,000 data points sampled from the fitted repeat time distribution.

acc), and DS(low acc)) in the main text.

Stage	Parameter	Explanation
Invoke main.sh	-f	The file path of the input reference genome or the contigs file in the fasta format.
Sequence sampling	-i	The file path of the input reference genome or the contigs file in the fasta format.
	-p	The prefix (including the path) of the output files in the fasta format, which contains the sample sequences.
	-n	The number of sampled sequences.
	-d	The length distribution used for the sequence sampling. 1: beta distribution, 2: exponential distribution, 3: mixed gamma distribution. The default is the mixed gamma distribution. If the read length drawn from the distribution is longer than the length of the genome, the value would be clipped to the length of the genome. The distribution can be directly modified in the samplying.py file in the sampling_from_genome directory so that users can use their desired distribution.

	-c	Whether the input genome is a circular genome or not.
Signal generator	-i	The file path of the input fasta file, containing the input sequences.
	-p	The prefix (including the path and signal file prefix) of the generated signal files.
	-t	The number of threads used for the program.
	-a	Change the read accuracy by modifying the signal repeat time distribution. The value is between 0 and 1. As the value increases, the standard deviation of the repeat time decreases which would increase the mapping accuracy of the final simulated reads. 0.1 would give the distribution that best simulates the real case while 0 would give the distribution whose result is slightly worse than the real case. 1 would give the almost perfect basecalling result. Note that due to the nonlinearity of the distribution changing, the value change in [0, 0.5] would affect the final result more dramatically than the value change in [0.5, 1].
	-s	Set the standard deviation of the random noise added to the signal. Default as 1.
	--perfect	Set it as 1 to get the almost perfect signal which could have 97% basecalling accuracy.
Signal to FAST5	-i	The file path of the template FAST5 file.
	-s	The directory of the input signal files.
	-d	The directory of the output FAST5 files.
Basecalling (Only the used)	-i	The directory of the input FAST5 files.
	-s	The directory of the output FASTQ files.
	-c	The configuration file of the basecaller.
	-o	Output format of Albacore.
	-t	Number of threads used in the basecaller.
Mapping accuracy check	-Hk19	The specific parameter of Minimap2 to perform mapping of the Nanopore reads onto the reference genome.

	-t	Number of threads used in the mapping.
	-c	Show the mapping in the cigar format.

Table S3: All the parameters of DeepSimulator

Table S4: DeepSimulator parameter settings for the DS(noise free), DS(high acc), DS(med acc), and DS(low acc) reads.

Readset	DS(noise free)	DS(high acc)	DS(med acc)	DS(low acc)
-a(Signal generator)	-	0.35	0.1	0
-s(Signal generator)	-	1	1.2	3.5
--perfect(Signal generator)	1	0	0	0

S5: Distribution of the length of simulated reads

Figure S2 shows the distribution of the length of the simulated and experimental reads for human, *E.coli* K-12 sub-strain MG1655, and lambda phage.

S6: Performance measure by dynamic time warping

The dynamic time warping (DTW) for mapping two input signals X, Y is denoted as DTW(X, Y). Specifically, given two input signal sequences $X = x_1, x_2, \ldots, x_{L_1}$ and $Y = y_1, y_2, \ldots, y_{L_2}$ of length L_1 and L_2, respectively, DTW constructs a warping path $W = w_1, w_2, \ldots, w_L$ to minimize the distance measurement $Dist(W)$ defined as follows:

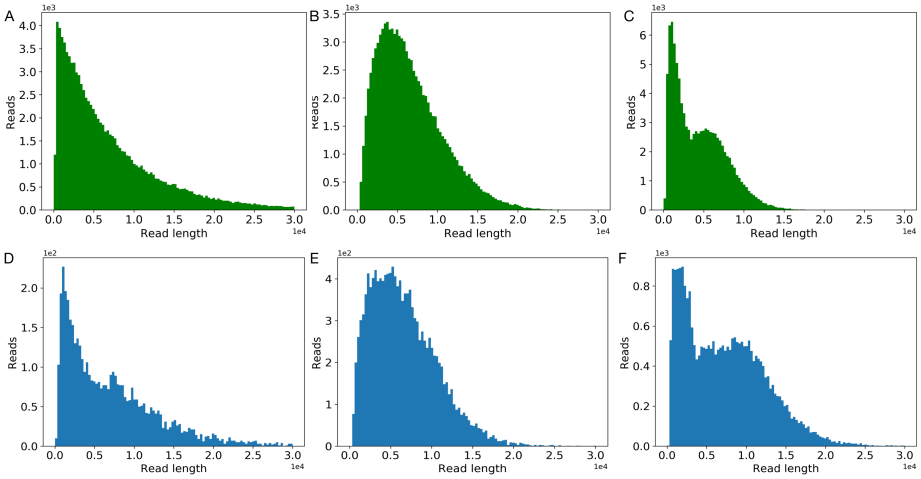

Figure S2: The distribution of the length of the simulated and experimental reads. The top panel shows the distribution of the length of the simulated reads from the exponential distribution (A), the beta distribution (B), and the mixed gamma distribution (C). The bottom panel shows the distribution of the experimental reads for human (D), *E.coli* K-12 sub-strain MG1655 (E), and lambda phage (F).

$$Dist(W) = \sum_{l=1}^{L} c(w_{li}, w_{lj}), \tag{2}$$

where L is the length of the warping path and $c(w_{li}, w_{lj})$ is the Euclidean distance of the lth aligned element between the two signal points x_i and y_j.

To determine the optimal path W, a $(L_1 \times L_2)$ matrix D is recursively computed as $D(i,j) = min\{D(i-1,j-1), D(i,j-1), D(i-1,j)\} + c(i,j)$. The matrix entry $D(n,m)$ is the total cost of an optimal path between $X(x_1,..,x_n)$ and $Y(y_1,..,y_m)$, which can be easily solved by dynamic programming which results in the global optimal distance. We used the normalized distance of a warping path $nDist(W)$ by dividing $Dist(W)$ by the maximal length of the input signals.

S7: *De novo* assembly of mitochondrial genome

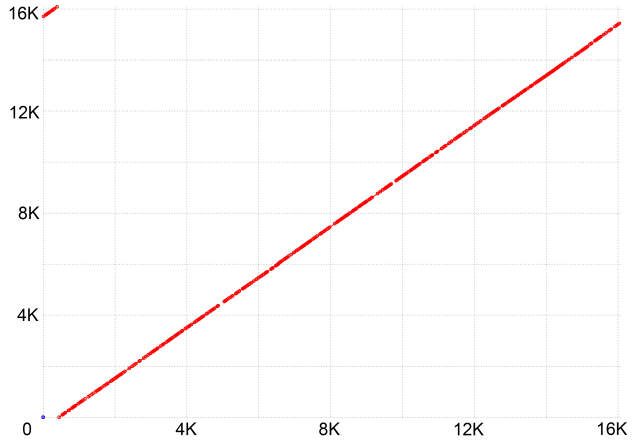

Figure S3: The Mummer plot comparing the reference mitochondrial genome on the x-axis with the assembled genome on y-axis using simulated reads from DeepSimulator.

Figure S3 shows the assembling result of the DeepSimulator simulated reads from the mitochondrial genome using Miniasm (Li, 2016) with Racon (Vaser *et al.*, 2017).

S8: Runtime and memory usage of DeepSimulator

In this section, we report the running time and memory usage of DeepSimulator, compared with NanoSim. We ran the experiment on a workstation with 56 cores and 128GB memory. The details could be referred to Table S5. As we can see from the table, the running time of DeepSimulator increases linearly with the number of simulated reads. To simulate a large amount of sequences, it would indeed take a significant amount of CPU time. However, DeepSimulator takes advantage of the multiprocessing cores, reducing the real time greatly and

enabling large scale simulation.

Table S5: Running time and memory usage analysis of DeepSimulator and NanoSim. The second and third rows show the running time of DeepSimulator(DP) and NanoSim(NS) when simulating different numbers of reads. Inside each cell, the first number is the real time and the number inside the bracket is the total CPU time. The fourth and fifth rows show the memory usage of DeepSimulator and NanoSim, respectively.

#Reads	10	20	50	100	200	400	1000	2000
DP(sec)	23(68)	30(101)	36(360)	43(683)	66(1227)	116(2563)	270(6650)	524(13598)
NS(sec)	0.4(0.4)	0.5(0.5)	0.6(0.6)	0.8(0.8)	1.2(1.2)	2.2(2.2)	4.2(4.2)	8.8(8.8)
DP(GB)	13.1	13.9	15.2	16.5	16.2	16.4	17.6	17.7
NS(GB)	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5

S9: NanoSim Experiment

Since the original NanoSim would perform random sampling for a given reference genome and output the simulated reads, which means we are unable to run the simulation on a specific given sequence. To solve the problem, we modified the original NanoSim to eliminate the random sampling step so that it can output the simulated read for a given sequence. Downloading the pre-trained R9 profile for the office website[1], we ran NanoSim over the dataset in Section 3.2 and obtained the simulated read for each input sequence. After that, we performed BLAST between the simulated reads and the input sequences and averaged the results over all the sequences to gain the profile shown in Table 1.

[1] ftp://ftp.bcgsc.ca/supplementary/NanoSim/

References

Baker, E. A. G., Goodwin, S., McCombie, W. R., and Mendivil Ramos, O. (2016). Silico: A simulator of long read sequencing in pacbio and oxford nanopore. *bioRxiv*, page 76901.

Boža, V., Brejová, B., and Vinař, T. (2017). Deepnano: Deep recurrent neural networks for base calling in minion nanopore reads. *PloS one*, **12**(6), e0178751.

Frith, M. C., Hamada, M., and Horton, P. (2010). Parameters for accurate genome alignment. *BMC Bioinformatics*, **11**(1), 80.

Lee, H., Gurtowski, J., Yoo, S., Marcus, S., McCombie, R. W., and Schatz, M. (2014). Error correction and assembly complexity of single molecule sequencing reads. *BioRxiv*, page 6395.

Li, H. (2016). Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, **32**(14), 2103–2110.

Vaser, R., Sovic, I., Nagarajan, N., and Sikic, M. (2017). Fast and accurate de novo genome assembly from long uncorrected reads. *Genome Research*.

Yang, C., Chu, J., Warren, R. L., and Birol, I. (2017). Nanosim: nanopore sequence read simulator based on statistical characterization. *GigaScience*, **6**(4), 1–6.