BMC Bioinformatics

METHODOLOGY ARTICLE Open Access

Lerna: transformer architectures for configuring error correction tools for short- and long-read genome sequencing

Atul Sharma[1†], Pranjal Jain[2†], Ashraf Mahgoub[1], Zihan Zhou[1], Kanak Mahadik[3] and Somali Chaterji[1*]

*Correspondence: schaterji@purdue.edu
†Atul Sharma and Pranjal Jain contributed equally to this work
[1] Purdue University, West Lafayette, US
Full list of author information is available at the end of the article

Abstract

Background: Sequencing technologies are prone to errors, making error correction (EC) necessary for downstream applications. EC tools need to be manually configured for optimal performance. We find that the optimal parameters (e.g., *k*-mer size) are both tool- and dataset-dependent. Moreover, evaluating the performance (i.e., Alignment-rate or Gain) of a given tool usually relies on a reference genome, but quality reference genomes are not always available. We introduce **Lerna** for the automated configuration of *k*-mer-based EC tools. **Lerna** first creates a language model (LM) of the uncorrected genomic reads, and then, based on this LM, calculates a metric called the *perplexity metric* to evaluate the corrected reads for different parameter choices. Next, it finds the one that produces the highest alignment rate *without* using a reference genome. The fundamental intuition of our approach is that the perplexity metric is inversely correlated with the quality of the assembly after error correction. Therefore, **Lerna** leverages the perplexity metric for automated tuning of *k*-mer sizes without needing a reference genome.

Results: First, we show that the best *k*-mer value can vary for different datasets, even for the same EC tool. This motivates our design that automates *k*-mer size selection without using a reference genome. Second, we show the gains of our LM using its component attention-based transformers. We show the model's estimation of the perplexity metric before and after error correction. The lower the perplexity after correction, the better the *k*-mer size. We also show that the alignment rate and assembly quality computed for the corrected reads are strongly *negatively* correlated with the perplexity, enabling the automated selection of *k*-mer values for better error correction, and hence, improved assembly quality. We validate our approach on both short and long reads. Additionally, we show that our attention-based models have significant runtime improvement for the entire pipeline—18× faster than previous works, due to parallelizing the attention mechanism and the use of JIT compilation for GPU inferencing.

Conclusion: Lerna improves *de novo* genome assembly by optimizing EC tools. Our code is made available in a public repository at: https://github.com/icanforce/lerna-genomics.

© The Author(s), 2021. **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/. The Creative Commons Public Domain Dedication waiver (http://creativecommons.org/publicdomain/zero/1.0/) applies to the data made available in this article, unless otherwise stated in a credit line to the data.

Keywords: Automated configuration tuning, Parameter search space, Natural language processing (NLP), Error correction, PacBio reads, Nanopore reads, Perplexity metric, Transformer networks

Background

The drop in high-throughput sequencing costs has offered unprecedented opportunities to characterize genomes, metagenomes, and single-cell genomes across the tree-of-life. Third-generation sequencing technologies [1, 2] have demonstrated the potential to produce unparalleled genome assemblies due to their capability to generate staggeringly longer reads, at a much faster pace, and remarkably low costs [3, 4], albeit, at low signal-to-noise ratios. This fundamental challenge makes genomic sequence identification and assembly, challenging, often necessitating hybrid correction approaches over self-correction from a consensus of long reads [5]. The error rates in base calling in these third-generation reads are nearly two orders of magnitude higher than their second-generation counterparts [6], with PacBio and Oxford Nanopore reaching error rates of 15% and 40% [7, 8] respectively, albeit improving with more sophisticated instrumentation [9]. In order to use long-read datasets, especially the more erroneous ones from the earlier versions of the PacBio and Nanopore sequencers, error correction tools in the assembly pipeline are needed [10]. A majority of already existing datasets have a significant error rate in them and performing error correction on these datasets reduces the cost compared to collecting the reads again with the new technology. Hence, diverse EC tools have been developed to remedy the erroneous reads [11].

k-mer based EC tools

The most dominant technique used by EC tools is *k*-mer-based error correction [12–16]. In these *k*-spectrum-based methods, the goal is to convert insolid *k*-mers (i.e., those likely to be erroneous) to solid ones (i.e., those likely to be correct). The *k*-mers that appear above a certain threshold frequency, and are therefore expected to be legitimate, are solid *k*-mers, the others are called insolid or untrusted *k*-mers. It has been found that performance of EC tools is extremely sensitive to the chosen *k*-value [4, 17], and the optimal value is both tool- and dataset-dependent. Small values of *k* result in an increase in the probability of overlap between reads at the cost of not allowing the algorithm to distinguish between erroneous and correct *k*-mers. In contrast, large *k*-values decrease the overlap probability and most *k*-mers appear to be unique, thus reducing the likelihood of correcting the errors.

Popular EC tools such as Lighter [18] and LoRDEC [11] for short- (< 400 base pairs) and long-read sequences (> 400 base pairs) respectively, require the user to select the *k*-value. Determining a favorable *k*-value among possible ones has been explicitly pointed out as an open area of work [4, 19, 20], since an arbitrary *k*-value could generate sub-optimal assemblies. In these scenarios, the best *k*-values need to be found by exploring all the possible *k*-values [20]. A large number of EC tools rely on *k*-mers to make corrections [12–16, 18]. We have used popular *k*-mer-based EC tools—Lighter [18] and LoRDEC [11] for short- and long-reads error correction respectively. Both of these exemplar tools require the user to select the *k*-value. Performance of error correction is extremely sensitive to this value. Some existing tools (e.g., KMERGENIE [21]) provide intuitive heuristics (e.g., abundance histograms) to guide *k*-value selection when performing de Bruijn Graph (DBG)-based genome assembly. However, experiments have shown that the optimum value of *k* varies across datasets *and* across different EC tools [19]. Thus, there is a need for a *data-driven and tool-specific* method to select the optimal *k*-value.

The runtimes of alignment algorithms like Bowtie2 [22] have superlinear complexity [23]. The perplexity computation in **Lerna**, in contrast, is linear in the length of the input (number of reads $\times$ read length). Because **Lerna** does not need to perform the alignment step, the runtime is significantly reduced. Furthermore, because **Lerna** subsamples reads, unlike alignment that requires entire read datasets, the time is further reduced. As seen in Table 3, the optimum value of *k* ranges from 15 to 25 for short reads when using Lighter. Performing an exhaustive search that requires alignment algorithms and entire reads is too time consuming. The optimum parameter configuration being dataset- and tool-dependent, further aggravates this process. For example, scanning over all *k*-mer values between 15 and 25 for dataset D6 and estimating the corresponding alignment rate with Bowtie2 took 1.5 hours on a server with 8 CPU cores and 12GB of RAM. In comparison, **Lerna** is 80x to 275x faster than Bowtie2 and was able to scan the same range in less than a minute.

State-of-the-art and our contribution

State-of-the-art automated EC tool tuner *Athena* [19] leverages NLP techniques to automatically compute the best value of *k* for error correction. However, Athena suffered from poor performance on datasets with larger read lengths, starting to degrade beyond reads of length 200 (as shown in Table 4). Further, as a result of the inability of RNNs to encode long-term dependencies, along with the character-level LMs being space inefficient, *Athena* failed to tune EC tools on long-read datasets. To address this problem, we engineered word-level transformer networks to deploy attention mechanisms, at different encoding granularities. This leads to improved alignment performance for lower read lengths and the ability to handle higher read lengths. With **Lerna** 's character- and word-level encodings encapsulated in transformer networks, our hypothesis of leveraging more sophisticated architectures for noisier and longer reads was validated. Specifically, **Lerna** either derived the optimal *k*-value or values corresponding to alignment rates within 0.31% of the optimal, plus improved runtimes of 18$\times$ faster than the best-in-class—Athena. Having improved the correlation rates for short reads, we moved on to solve the harder problem of finetuning EC tools for longer reads. This is because third-generation sequencing technologies have low signal-to-noise ratios and a high error rate in base calling, with rates as high as 15% and 40% for PacBio and Nanopore data, nearly two orders of magnitude higher than second-generation sequencing data. Athena used a character-level RNN in its algorithmic suite. We find through detailed analysis that character-level models, in contrast to word-level models, provide weaker correlations between perplexity and alignment rate, making them unsuitable to use over longer and/or noisier datasets. Intuitively, with word-level LMs, we expand the vocabulary size (4^W) and allow the model to learn identifiable patterns in the data, where W is the word length.

Additionally, the much slower inference [18× slower] of Athena (see Table 10) creates a bottleneck for downstream genomic analysis. Furthermore, as indicated before, RNN-based language models have the fundamental flaw of being unable to capture long-term dependencies in sequences [24]; hence it was unable to autotune parameters for *long-read* EC tools.

Our innovation, **Lerna**, schematically outlined in Fig. 1, employs a variant of Transformer networks [25] to perform automatic tuning of EC tools on *both* more error-prone, third-generation long reads and second-generation short reads. The original transformer used by Vaswani et al. is causal and directional, i.e., the predictions for position *i* can depend only on the elements at positions less than *i*. However, in our problem context of genomic error correction, such directionality is not apt. Therefore, we modify the encoder-decoder architecture in the transformer LM and allow it to train in a non-directional sense.

In summary, **Lerna** makes the following contributions:

1. We propose a mechanism to automatically select the best configurations for *k*-mer based error correction tools. Our mechanism handles both short and long reads by modifying Transformer Language Models [25] by disabling the masks to use non-directional word-level training, improving *k*-mer selection accuracy by 35% over uni-directional character-level RNNs used by prior works. Further, these non-directional models were found to be equivalent to the bi-directional models in our experiments and thus we disabled the masks, reducing the number of model hyperparameters.
2. We propose several runtime improvements such as leveraging parallelizable computations in self-attention layers, and utilizing JIT (Just-in-Time) compilers in order to make better optimizations based on the underlying GPU hardware. These improvements lead to efficient utilization of resources and 18× lower inference time than

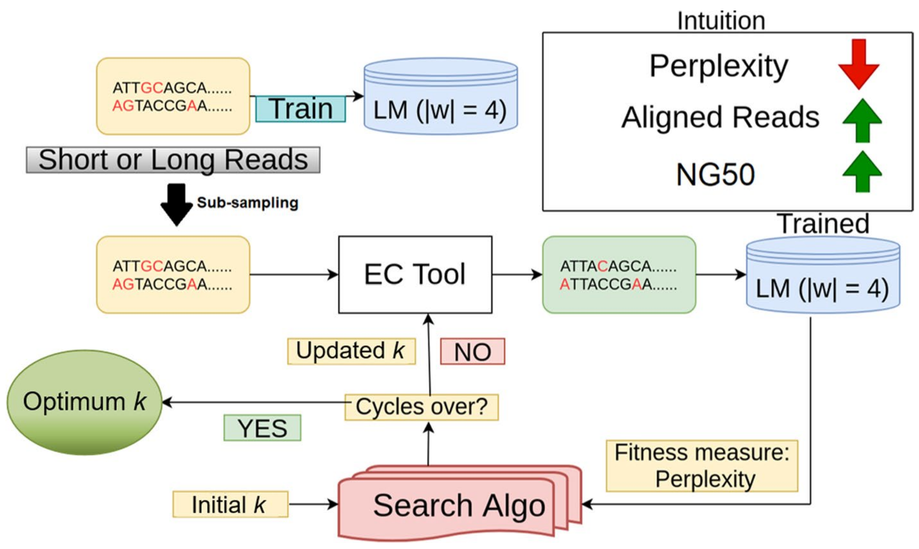

Fig. 1 Workflow of **Lerna**. **Lerna**'s high level intuition is that by reducing the perplexity metric for the generated reads, we increase the alignment rate and the assembly quality (NG50). Our evaluation shows that a word-level Transformer LM, with a word length of 4 ($|w| = 4$) works best across all datasets and tools. Our algorithmic suite works on both NGS short reads and PacBio and Nanopore long reads

the state-of-the-art tool Athena for choosing the best configuration parameters for genomic error correction (see Table 10).

3. We evaluate the generalizability of **Lerna** on second- and third-generation sequencing technologies, specifically, Illumina short reads, and PacBio and Nanopore long reads, validating that **Lerna** can optimize both short and longer, noisier reads.

Furthermore, **Lerna** reports a negative correlation in each and every dataset (unlike Athena, which has a positive correlation of + 0.95 on D6) with an average of − 0.92 over D1 to D7, degrading with longer read lengths. Our experiments have shown that the highest negative correlation is observed when word-level Transformer LMs are used, rather than character-level. Intuitively, with word-level LMs, we expand the vocabulary size and allow the model to learn identifiable patterns in the data.

Language modeling

Language modeling has played an integral role in improving speech recognition, text analysis, and sentiment analysis. A Language Model (LM) is a probability distribution over a sequence of tokens (e.g., words or characters), using which we can identify the sequences that are more likely to occur [26–29]. LMs can be count-based (e.g., estimating N-gram probabilities via counting and subsequent smoothing) [30] or continuous-space [31] language modeling, such as the use of Recurrent Neural Networks (RNNs). However, several papers have pointed out various drawbacks of using RNNs (see the next subsection). As an alternative, we make use of computationally efficient transformer LMs and further modify its architecture for speedup and parallelization.

N-gram-based language modeling

N-Gram models [32] are used to predict the probability of observing a token $[W_i]$ after the series of all already observed tokens $[W_0, \ldots W_{i-1}]$ (i.e., context). This is done for all possibilities present in the vocabulary. However, being a computationally expensive task, the above probability is approximated using Eq. 1.

$$\begin{aligned} P(W_0, W_1, \ldots, W_m) &= \prod_{i=1}^{m} P(W_i | W_{i-1}, \ldots, W_1) \\ &\approx \prod_{i=1}^{m} P(W_i | W_{i-1}, \ldots, W_{i-n}) \end{aligned} \tag{1}$$

where n represents the context. As we increase n, the accuracy increases, albeit at the cost of increasing complexity and training time. The conditional probabilities are stored as a lookup table, leading to a higher memory complexity. The size of the model on $D7$ for the N-gram LM is 2 GB while the size of the RNN model is only 3.5 MB for $D7$ (a description of the datasets is given in Table 2). Moreover, N-gram-based models suffer from increased data sparsity when trained on longer contexts. As N increases, the number of possible word combinations becomes 2^N and the amount of data becomes inadequate to train the N-grams. In contrast, Neural LMs (such as RNNs) overcome these limitations and are preferred over N-gram LMs [33].

RNN-based language modeling

RNNs are a class of LMs that rely on an internal "state" to make predictions, usually consisting of three or more layers. The input layer is given an input vector x_t. This may be a one-hot encoding vector, such as in our case, of the *t*th word or character. The hidden layer represents the memory of the network: the internal "state" s_t. A non-linear function f (e.g., tanh) is applied to the previous hidden states s_{t-1} and x_t (see Eq. 2)

$$s_t = f(Ux_t + Ws_{t-1}). \tag{2}$$

where U and W are the input and state weights respectively. However, RNNs suffer from the problems of vanishing and exploding gradients [34]. Due to the sequential nature of gradient computation results, training of RNNs is computationally demanding. Despite recent advances in training RNNs, *capturing long-term dependencies in sequences remains a fundamental challenge* [24]. As the sequence length increases, the path length of information flow increases, resulting in greater likelihood of the information being corrupted. The inherent sequential nature of RNNs precludes parallelization within training examples, which becomes critical at larger sequence lengths, as memory constraints limit batching across examples [25]. LSTMs [35] and GRUs [36] are alternatives but the *computation cannot be parallelized* because of the sequential nature of operations in that the internal state at any time step t depends on the state of previous time step $t-1$. This remains a fundamental issue in all sequential ML models. Various architectures have been proposed to solve these problems. The common target of all of them is to use attention mechanisms to model probability distributions [25, 37].

Transformer-based language modeling

Attention mechanisms are a way of selectively focusing on certain parts of a sequence [38]. Specifically, *self attention* relates different positions of a single sequence in order to compute a representation of the sequence and has been used in a variety of tasks [39], e.g., abstractive summarization [40], textual entailment [41], and learning task-independent sentence representations [42]. We incorporate self-attention in the Transformer network architecture. Learning long-range dependencies is a key challenge in many sequence transduction tasks. One key factor affecting the ability to learn such dependencies is the length of the paths that forward and backward signals have to traverse in the network. The shorter these paths between any combination of positions in the input and output sequences, the easier it is to learn long-range dependencies. In the transformer, this value is constant $\mathcal{O}(1)$. Furthermore, because the transformer contains no recurrence and convolutional operations, in order for the model to make use of the ordering of the input sequence, positional encoding of the input is performed. This encoding is done by using *sine* and *cosine* operations of different frequencies as follows:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}}) \tag{3}$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}}) \tag{4}$$

Here, d_{model} represents the embedding dimension, *pos* is the position, and *i* is the dimension index. Recurrence relations generate states s_t that depend on the previous state s_{t-1}

and the input at time t. This sequential nature of computations precludes parallelization, hence the removal of recurrence relations from the Transformer allows it to be parallelized [25].

Genomics preliminaries

This section gives brief definitions for a few genomics concepts that we focus on throughout the paper, such as error correction, alignment, and assembly. `Error correction` represents the process of identifying erroneous nucleotide base pairs in genomics reads and converting them to the correct pairs. This process is applied as a pre-processing step for the sequenced reads before further analysis is performed. It was shown that error correction can significantly improve the performance for all tasks downstream the analysis pipeline. `Alignment` is the process of matching the sequenced reads to a continuous error-free reference genome and identifying which segment in the reference genome matches a given read the best. `Assembly` is the process of merging the sequenced reads together into longer sequences in order to retrieve the original sequence. The design of **Lerna** focuses on improving the performance of error correction algorithms by identifying the best configuration parameters, which significantly improves the alignment rate and the assembly quality.

Perplexity of the language model

Perplexity is a metric that measures the goodness-of-fit of a sequence given an LM. The LM represents the probability distribution over the entire dataset [43]. Perplexity can be thought of as the probability of selecting a given word uniformly from the effective vocabulary, given some context. Thus, a lower perplexity indicates that the LM is better at making predictions. If an LM learns a uniform probability distribution over a given piece of text, the perplexity generated would be equal to the actual vocabulary size. The smaller the perplexity score is, compared to the actual vocabulary size, the better is the LM at learning patterns that occur with a high probability in the given text. In our context of genome error correction, the error correcting tools are expected to convert most of the insolid k-mers to solid ones. Since solid k-mers have a high frequency of occurrence, more the conversion of insolid to solid k-mers takes place, lower is the effective vocabulary size of the corrected data. This can be quantified using the perplexity metric. Mathematically, perplexity of a sentence (Eq. 5) is the inverse probability of the test set, normalized by the number of words [43].

$$PP(W) \approx \sqrt[m]{\frac{1}{\prod_{i=1}^{m} P(W_i|W_{i-1},\ldots,W_{i-n})}} \tag{5}$$

It is clear from (5) that minimizing perplexity maximizes the probability of the observed set of m words from W_1 to W_m.

Various models estimate the perplexity metric differently. However, they achieve the same purpose, which is estimating the correctness of a sequence given the trained probability distribution. Although the perplexity for Transformers and N-Grams is calculated in different ways, it gives us a measure of probability of a sequence given a trained LM. The Kullback–Leibler (KL) divergence (Eq. 6) is a fundamental equation of information theory that quantifies the proximity of two probability distributions. It specifies in bits how close a probability distribution P_i is to another distribution Q_i [44].

$$D(P\|Q) = \sum\nolimits_i P_i \log_2 \frac{P_i}{Q_i} \tag{6}$$

A metric similar to the KL divergence between two probability distributions is the cross entropy (Eq. 7). It is also used to measure the similarity between two probability distributions. However, there are subtle differences between the metrics. Cross entropy gives the average code length needed to represent one distribution by another distribution, and the excess code needed over the optimal coding is given by the KL divergence [45].

$$H(P,Q) = -\sum\nolimits_i P_i \log_2 Q_i = H(P) + D(P\|Q) \tag{7}$$

In **Lerna**, the Transformer LM uses the perplexity metric (Eq. 8), which is derived to be the exponential of the cross-entropy loss, where $\hat{y}$ is the predicted next character—the output of the LM—and $|V|$ is the vocabulary size used during training. Therefore, by using perplexity, we incorporate information-theoretic principles into our LM, and in effect, ensure that by minimizing the perplexity, we maximize the similarity between the ground truth and the predictions made by the LM.

$$Perplexity = e^{CE(y,\hat{y})} = \exp\,(-\sum_{i=1}^{|V|} p(y_i)\log(p(\hat{y}_i))) \tag{8}$$

Note that training our LM model on the uncorrected reads can still capture an accurate probability distribution, which we use to measure the quality of the corrected reads. The reason is that genomic reads usually have a high coverage (typically 30X or more), which makes the number of occurrences for an erroneous nucleotide much lower than that of the correct nucleotide. This means that the correct genomic reads still dominate the distribution.

Now, we discuss the key design elements of **Lerna** and the corresponding challenges addressed by each element.

Transformer-based language modeling in Lerna

The attention function (Eq. 9) used in a Transformer can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key. Specifically, scaled-dot product attention is done by taking dot product of *Query Q* and *Key K* vectors. This is then divided by $\sqrt{d_k}$, where d_k is the dimension of K. This scalar value is multiplied by value vector V, and a softmax is taken to get a probability distribution.

$$Attention(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}((\frac{\mathbf{QK}^T}{\sqrt{d_k}})\mathbf{V}) \tag{9}$$

Using multiple heads, rather than one, allows the language model to attend to information from different representation subspaces rather than from a single one. In the Transformer, the Attention module repeats its computations multiple times in parallel. Each of these is called an attention head. Because the layer after the concatenation is a feed forward network with an input dimension that does not depend on the number of heads, we take a linear combination of the concatenated vectors by multiplying with a matrix W^O so that the resulting vector has the same dimension as the input dimension of the feed forward network.

$$MultiHead(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(head_1, \ldots, head_h).\mathbf{W}^O \tag{10}$$

where $head_i = Attention(\mathbf{QW}_i^Q, \mathbf{KW}_i^K, \mathbf{VW}_i^V)$. The architecture described by [25] introduces masks in the self-attention layers. The self-attention sublayer in the decoder stack has been designed to prevent positions from influencing subsequent positions. This masking ensures that the predictions for position i can depend only on the known outputs at positions $< i$. We modify this in **Lerna** to remove these masks, allowing the representations to encode both past and subsequent nucleotides. This is important because each nucleotide in a k-mer provides critical information, so the encoding of a given nucleotide in k-mer should have information about what is to the left and to the right of it. BERT [46] successfully alleviates the unidirectionality constraint of the Transformer, by using deep bidirectional transformers. Drawing inspiration from BERT, we considered implementing two parallel architectures with masks activated—one for the forward direction and the other for the reverse direction. This would allow the implementation of bi-directionality into the LM. *However, empirically, we have shown that disabling masks and bi-directionality give the same results. Hence, we simply disable masks in order to reduce the number of parameters in the model.* RNNs compute an internal state at each time step s_t, which depends on the previous state s_{t-1} and the current input at time t. The Transformer, which gets rid of recurrent layers, are not restricted by this recursive relation, which must be computed one after the other, allowing parallelization. The use of parallel attention modules has previously been used in reducing training time and establishing a new state-of-the-art model for English to German translation [47]. The ability of the Transformer networks to be parallelized opens up greater scope for optimizations in the machine-translated code.

Language modeling of genome sequences

In general, LMs are trained to statistically determine if a sequence of tokens (e.g., words in the NLP domain, or base-pairs in our usecase) contains erroneous (insolid or untrusted) tokens. We make use of the fact that sequencing technologies require sequencing every base-pair multiple times to provide a better and higher coverage of the genome. Accordingly, the number of erroneous reads becomes far less than the number of error-free reads, allowing correct k-mers to outweigh erroneous ones. We train an LM on the set of reads before correction, expecting the correct sequences to mask the effect of the erroneous ones during the training phase. After correction of the reads by an EC tool, the resulting reads are tested against the trained language model. The evaluation metric chosen is *perplexity* that is used to model the *goodness of fit* of the corrected reads compared to the learned LM. The lower the perplexity, the better the fitness of the reads to the probability distribution captured by the LM.

Following this argument, the error-free reads with plenty of solid *k*-mers are expected to produce a lower perplexity than erroneous reads with insolid *k*-mers, and this is exactly what we observed in our experiments as seen in Table 1. We generated a data-set with the NanoSim [12] simulator using the reference genome of *E. coli*, strain K-12, substrain MG1655, with the characteristics of Nanopore sequencing [48]. The generated data set had 588 reads labeled "unaligned" (i.e., with nucleotide-level error rates over 90%) out of 10,000 total reads. We tested a word-level transformer model with word length $= 4$, on the entire dataset and separately on the erroneous and correct reads. From Table 1, it is clear that correct reads generate a lower perplexity than the erroneous ones for multiple values of *k*. We use this idea to optimize the performance of EC tools by varying the input parameter *k*, and select the *k*-value that minimizes the perplexity. We have validated the results by performing alignment with the reference genome for the erroneous *vs*. corrected reads. The corrected reads showed a higher alignment rate indicating a more effective correction of the sequences.

Challenges in adapting LM for genomic reads There are a few differences in the training of an LM on the English language (or on any other language for that matter) and a genome sequence. *First*, a genome sequence has no concrete concept of words unlike an actual language. One may train a character-level model on it, but the training may be negatively affected by the small vocabulary size of sequences consisting of only A, T, G, and C. Also, one may use a fixed word length and pre-process the sequences to generate words, by using a suitable stride value. We have reported the effects of word length on training in our results that a word length of 4 leads to stronger negative correlation between perplexity and alignment rate, in contrast to smaller or larger word lengths as too small or too large vocabulary sizes can make learning patterns difficult for the LM. *Second*, a genome sequence has no concept of punctuation marks such as periods. The sequenced reads, although having a finite read length, do not necessarily follow in succession. In the English language, the word following a period proceeds the word preceding it. In contrast, in a genome sequence, successive reads may or may not follow each other. Therefore, the batch creation for training or testing cannot be done arbitrarily. It has to be ensured that successive reads do not affect the training of other reads and that there are no unjustifiable breaks within a read during batch creation. Specifically, every read is tokenized into chunks of 100 base-pair length, before being fed to our transformer-based model. To preserve the overlap between the produced chunks, we use a sliding

Table 1 Effect of erroneous sequences on perplexity for multiple *k*-values on reads: these reads are generated by NanoSim using the *E.coli* reference genome

k	PPL_{err}	PPL_{corr}	PPL_{total}
15	1073.6	943.5	952.9
37	1072.8	944.5	946.8
81	1072.8	973.4	956.3

PPL_{err}, PPL_{corr}, and PPL_{total} denote the perplexity scores on erroneous and error-free reads, and the entire dataset (i.e., erroneous and error-free sequences) window with a step size (*a.k.a.* stride length) of 50. If length of a read in not divisible by 100, the residual portion of the read is trimmed as they account for a small fraction of the read. *Third*, genome sequences can be considered non-directional. A portion of a genome can as much be affected by the portion of the genome preceding it as by the portion of the genome in succession. This is also true for the English language [46] but most of the models available are directional. Recent developments have resulted in the transformer network architecture that we orchestrate in **Lerna** to conform to the non-directionality attribute of genomics reads. *Fourth*, since we train our model on the data *before* correction, expecting the solid k-mers to influence training more than the erroneous ones (by outnumbering the erroneous ones), it has to be taken care that the batch size during training $\geq$ the batch size during testing. A larger batch size during training will help circumvent the effect of noisy or erroneous data, and help the model better learn the patterns occurring with high frequency. On the contrary, a smaller batch size during the testing phase will help identify sequences that have not occurred frequently during the training. Following the above rationale, we justify the use of NLP techniques to model genome sequences for evaluating the performance of EC tools.

Search heuristic

Our aim is to find the k-value that will minimize the perplexity of the corrected sequence. This ensures that the sequence has the highest chance of occurring, signaling that the EC tool is working optimally. The perplexity function calculated using the LM is denoted by f (Eq. 11). Although we experiment with k-mer-based approaches, any configuration parameter can be used in the search depending on the tool used, such as the maximum number of passes per read in Jabba [49], or the value of the maximum boundary difference to split the subcontigs in HALC [50]. Whatever the configuration parameter may be, our search heuristic uses perplexity as a measure of "fitness". Our search heuristic aims to find the configuration parameter with the minimum perplexity, and in doing so, maximizes the alignment rate and the assembly quality.

$$k_{opt} = arg \min_{k_i} Perplexity = arg \min_{k_i} f(LM, D_0, k_i) \tag{11}$$

Here, LM: trained language model, D_0: uncorrected read set, and k: the configuration parameter we wish to tune.

In **Lerna**, we apply a Simulated Annealing (SA)-based search that improves an initial solution by walking randomly in the space of possible solutions and gradually adjusts a parameter called "temperature". At high temperature, the random walk is almost unbiased and it converges to essentially the uniform distribution over the whole space of solutions. As the temperature drops, each step of the random walk is less likely to explore sub-optimal choices. Unlike a Hill Climbing-based approach used by prior work Athena, Simulated Annealing does not get stuck at local optimum, and converges faster to the global optimum [51]. The problem with local optima is partially mitigated in Athena by multiple runs of the search with various initializations but this comes at the cost of higher execution times.

Table 2 NGS short-read datasets' description with coverage (estimated per Illumina's documentation), number of reads, read lengths, genome type, and the Accession #

Dataset	Coverage	#Reads	Read length (bp)	Genome type	Accession number
D1	80×	20.8M	36	*E. coli* str. K-12 substr	SRR001665
D2	71×	7.1M	47	*E. coli* str. K-12 substr	SRR022918
D3	173×	18.1M	36	*Acinetobacter* sp. ADP1	SRR006332
D4	62×	3.5M	75	*B. subtilis*	DRR000852
D5	166×	7.1M	100	*L. interrogans C* sp. ADP1	SRR397962
D6	70×	33.6M	250	*A. thaliana*	ERR2173372
D7	67×	202M	101	*Homo sapiens*	SRR1658570

Table 3 **Lerna** evaluated on 7 Illumina (Table 2) short read datasets

Dataset	Read length	Exhaustive search		Lerna (Char)		Lerna (word)		Athena (RNN)	
	Base pairs *bp*	Selected *k*	Alignment rate (%)	Selected *k*	Alignment rate (%)	Selected *k*	Alignment rate (%)	Selected *k*	Alignment rate (%)
D1	36	17	98.95	Same as exhaustive search		Same as exhaustive search		Same as exhaustive search	
D2	47	15	61.42	Same as exhaustive search		**19**	**61.27**%	**k=17**	**61.15**%
D3	36	15	80.44	**k=17**	**80.39%**	Same as exhaustive search		**k=17**	**80.39%**
D4	75	17	93.95	**k=15**	**93.72%**	Same as exhaustive search		Same as exhaustive search	
D5	100	17	92.15	Same as Exhaustive Search		Same as exhaustive search		**k=25**	**92.09**%
D6	250	25	86.16	Same as Exhaustive Search		Same as exhaustive search		**k=17**	**85.63**%
D7	101	17	40.53	**k=15**	**40.24%**	**k=15**	**40.24%**	Same as exhaustive search	

We test both the Transformer character- and word-level LMs. We observed that the best performance was attained by using the Transformer word-level LM with word length 4 ($|w|$=4). For the Transformer word-level LM ($|w| = 4$), **Lerna** finds either the optimal value or a value with an alignment rate within 0.31% of the theoretical best, consistent with the reported results by Lighter (Figure 5 in [18]). These slightly sub-optimal configurations are an artefact of sub-sampling

Results

Theoretically, **Lerna** can help tune any EC tool, but for the sake of evaluation, we have used Lighter for short reads correction and LoRDEC for long reads correction.

Evaluation on short reads

We begin with the demonstration of the effectiveness of **Lerna** on short reads datasets. The details of the seven NGS datasets used are given in Table 2. These are Illumina short reads and have been used in multiple prior studies (e.g., [12, 18]) with reference genomes available (used to evaluate the quality of error correction). The datasets have different read lengths (from 36 to 250 bp) and different error rates (< 3%) . After correction using EC tool Lighter [18], we run the Bowtie2 aligner [22] and measure the Alignment Rate. A higher value implies superior error correction. We do an exhaustive search over various *k*-values to see whether **Lerna** produces optimal values. Table 3 summarizes our findings. We see that the Transformer word-level LM performs slightly better than the character-level variant; the former predicts the correct value in 5 datasets while the latter makes accurate predictions in 4 datasets; these out of a total of 7 datasets. Furthermore, when optimal parameters are not chosen, the alignment rate lies within 0.31% of the best possible alignment rate for both char- and word-level models.

A comparison of **Lerna** and Athena can be seen in Table 4. We notice that **Lerna** 's Transformer-based model produces a higher correlation with the alignment rate compared to the RNN-based model used in Athena for longer reads (i.e., *D6* with 250 bp). This shows that more sophisticated NLP architectures are better suited for longer and noisier reads as is typical of the long-read sequences produced by PacBio and Nanopore sequencers. Recall that our refined NLP architecture uses parallelizable attention mechanisms in place of sequential recurrent layers, which affords higher optimizations. Furthermore, by restricting the maximum path length to $\mathcal{O}(1)$, we have prevented the corruption of information. This is unlike RNNs, in which the path length depends on the length of the input sequence.

Next we measure the improvement of **Lerna** 's pipeline on the assembly quality. In short, assembly is the process of merging the reads together to make longer reads (*a.k.a* contigs), and the longer the constructed contigs, the closer they are to the original sequence (consensus regions). We use SPAdes assembler [52] to perform *de novo* assembly for the reads before and after correction and then use QUAST [53] on the generated contigs to estimate the assembly quality metric NG50, which we show in Table 4. The N50 is a metric that indicates that contigs equal to or larger than its value represent 50% of the total assembly size, and the NG50 is similar to N50, but based on the estimated genome size. NG50 is similar to mean or median of the generated read contigs but with a higher weight assigned to longer sequences, hence the higher the better. We notice that **Lerna** is able to improve the NG50 by a geometric mean of 5.08× across the 7 real datasets as opposed to Athena, which shows an improvement of 4.72×. This shows the importance of using **Lerna** 's pipeline to improve the performance for alignment and assembly tasks significantly.

Table 4 A comparison between Athena and **Lerna** on short reads

Dataset	Read length (bp)	Coverage	Correlation Athena	Correlation Lerna	NG50 without correction	NG50 with Lerna	NG50 with Athena
D1	36	80×	− 0.93	− 0.94	3019	6827	6827
D2	47	71×	− 0.97	− 0.96	47	**2254**	2164
D3	36	173×	− 0.92	− 0.93	1042	**4873**	4164
D4	75	62×	− 0.86	− 0.97	118	858	858
D5	100	166×	− 0.96	− 0.98	186	**3524**	2799
D6	**250**	70×	**+0.95**	**− 0.84**	1098	**1344**	1237
D7	101	67×	− 0.72	− 0.82	723	739	**754**

The dataset is described in Table 2. We show the correlation between the perplexity metric and the alignment rate of the data after correction for **Lerna** vis-à-vis its closest competitor, Athena. On dataset *D6* (greatest read length of 250 bp), Athena fails and has a positive correlation of + 0.95 (instead of having a negative correlation, as desired), highlighted in the Table. This is in line with the fact that RNNs are unable to model longer sequences. We also show the improvement of the assembly quality (in NG50) after tuning the EC tool with **Lerna** versus using the uncorrected reads. The NG50 with Lerna is always higher than, or equal to, that with Athena, other than a small drop for Dataset D7. All the superior values are bolded

Evaluation on long reads

For long-reads evaluation, we use the PacBio simulator [54] to generate 10,000 reads with read lengths that vary between 2000 and 3000 base pairs and an accuracy of 78%, which matches the characteristics of real continuous long reads (CLR). Due to the varying read lengths in the dataset, an additional pre-processing step is performed before training our LM. Specifically, every read is tokenized into chunks of length 100 base-pairs before being fed to our transformer-based model. To preserve the overlap between the produced chunks, we use a sliding window with a step size (a.k.a stride length) of 50. If length of a read in not divisible by 100, the residual portion of the read is trimmed as they account for a small fraction of the read. This fraction is upper bounded at 1.6% in our evaluation.

We use the reference genome of *E. coli*, strain K-12, substrain MG1655. We used LoR-DEC [11] for error correction since it has proven to work well on PacBio reads and is computationally efficient, yielding results within just a few minutes. Since it is a hybrid error correction tool, the short reads we use are those of *E. coli*, strain K-12, substrain MG1655, accession number SRX000429. The LM was trained on the uncorrected reads, and tested on the corrected reads. The ground truth was generated by running alignment using Minimap2 [55] and the statistics of the aligned reads were generated using paftools (provided in the Minimap2 repository). The experiments were repeated for a character-level transformer and a word-level transformer network, where the input sequences were converted into words during the pre-processing phase.

Given that our data has 4 literals, A, T, G, and C, for a word length of $|w|$, the vocabulary size of the dataset is equal to $4^{|w|}$. The number of tokens in the model was set to $4^{|w|}$. The size of the word embedding vector was set to $4^{|w|}$ for $|w| < 4$, and to 128 otherwise, because an embedding size larger than the number of tokens is not a sensible choice. The number of encoder layers was set to 4, same as that of the model trained on short reads. The hyperparameter *bptt* represents back propagation through time, i.e., the data is divided into chunks of size *bptt*. It was set to 99 after it was observed that it is not sensitive to the quality of training for a significant range, from 50 to 500. We use perplexity as our evaluation metric, and its correlation with the percentage of aligned reads after correction was computed. The results are reported in Table 5. A more detailed result of variation of perplexity with k is shown in Fig. 2 for $|w| = 4$.

In Table 5, $|w|$ refers to the word length selected for training and $|V|$ is the vocabulary size of the data. The *mean* and *standard deviation* of the perplexity score has been calculated on the corrected data for $k = 15,17,19,21,23,25,27,31,37,45$, since beyond this value there was no change in the percentage of alignment for the corrected reads with the reference genome. Furthermore, a lower k-value is not usually recommended; in most cases, it is not supported by the error correction tool. The correlation between perplexity and the percentage of aligned reads was computed for the above mentioned values of k after standard normalization of both, perplexity and the percentage of aligned reads. A negative correlation between perplexity and % aligned reads was expected but it was interesting to see a positive correlation for $|w| = 2$ and 3. A possible reason could be a low information gain in 2-mers and 3-mers; i.e., a 2-mer may not be followed by a character with a high probability, and the prediction of a 3-mer from a given 2-mer appears to be random because of a small vocabulary size. We observe a low negative correlation

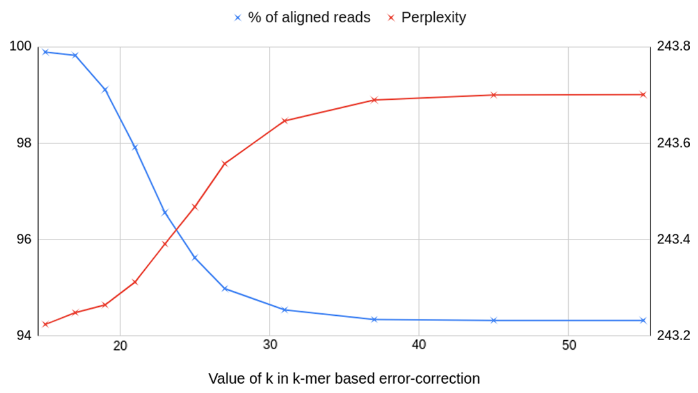

Fig. 2 Variation of perplexity and % of aligned reads with *k* on simulated *E. coli* PacBio reads, corrected by LoRDEC, mapped by Minimap2. The perplexity is computed by a Transformer word-level LM with $|w| = 4$. A strong negative correlation between the two metrics is clearly observed

Table 5 Effect of word length on correlation between perplexity and the percentage of aligned reads for *E. coli* PacBio simulated reads

$\|w\|$	$\|V\|$	Mean (PPL)	SD (PPL)	Correlation	Test time (s)
1	4	3.9997	0.00004	− 0.714	367.5
2	16	15.824	0.02298	+ 0.825	990.1
3	64	62.176	0.06331	+ 0.761	1439
4	256	243.45	0.17869	− 0.965	2026
5	1024	951.70	2.86750	− 0.904	1862
6	4096	3724.7	21.5649	− 0.889	2472
7	16,384	14675	133.207	− 0.882	2923
8	65,536	59199	811.820	− 0.877	7150

Strong correlation is observed for higher |*w*| values, with $|w| = 4$ producing the strongest correlation. The vocabulary size $4^{|w|}$ is represented by |*V*|

for $|w| = 1$. Although it is known that character-LMs perform well, the weak correlation is not good enough to eliminate the need for a reference genome. According to information theory, entropy is maximized when a uniform distribution is selected over a countable set [56]. The perplexity is the exponential of the cross entropy (Eq. 8). Hence, the perplexity, in essence, models the effective vocabulary size, i.e., it equals the vocabulary size when the probability distribution of the tokens in a text data is uniform, and it is less than the vocabulary size when there are motifs present in the data, with identifiable patterns that occur with high probability.

From the results, we see that the difference between the vocabulary size and the mean perplexity grows with the word size. The standard deviation of perplexity also grows with word size. The task of our search heuristic (of finding the best *k*) is made easier since our LM can differentiate easily between perplexities corresponding to *k*-values. To eliminate the need for the ground truth, that is the reference genome, the sole requirement was a high correlation between perplexity and % of aligned reads, and we see that strongest correlation was obtained for $|w| = 4$. Effectively, the word length only affects the effective *bptt* value, given that a stride of 1 is chosen for converting text to words. It also helps set the vocabulary size, allowing a model to look for high probability motifs in different

Table 6 Results of **Lerna** evaluated on PacBio reads: $|w|$ denotes the word length used for training, k is the selected k-value, *mapped reads* is the total number of reads out of the 10,000 reads that aligned with the reference genome after correction done by LoRDEC using the selected k-value

$\|w\|$	k	Mapped reads
1	67	9432
2	67	9432
3	67	9432
4	**15**	**9989**
5	17	9982
6	17	9982
7	17	9982
8	23	9656

Table 7 Results of **Lerna** evaluated on Nanopore simulated reads corrected using Canu

k	Mapped reads
11	9531
13	9549
15	9631
17	9723
19	**9813**
21	9724
23	9696
25	9657
27	9553
29	9545

We use a word length of 4 ($w = 4$) and find that the best value of the MhapMerSize comes out to be 19

vocabulary spaces. For example, consider a sequence of length 10, AATCGGCGCT. Let us choose *bptt=9*. A char-level model learns the following probability P(T|AATCGG CGCT) if trained on the given sequence. If we choose a word length of 4 with *stride=1*, the data is pre-processed to generate the sequence, AATC ATCG TCGG CGGC GGCG GCGC CGCT. With *bptt=6*, a word-level model learns the probability P(CGCT|AATC ATCG TCGG CGGC GGCG GCGC) that is the same as P(T|AATCGGCGC). Hence, a word-level model with *word length=4, stride=1, bptt=6* effectively learns the same probabilities that a character-level model with *bptt=9* learns, with the only difference being the vocabulary size. From our experiments, we have concluded that $|w| = 4$ is the best choice for our use case. We believe further research can be done on how training LMs on genome sequences is affected by the word length. This will help us gain more insights on how to detect solid k-mers using NLP techniques. The results of our pipeline on PacBio simulated long reads, are reported in Table 6. We find that the results are consistent with our claims made above. For $|w| = 4$, our search heuristic selected the value of k corresponding to the best alignment. 9989 out of 10,000 reads aligned with the reference genome for $k = 15$ which is selected only for $|w| = 4$. For all other word lengths $|w|$, the optimal k value for a minimum perplexity and the optimal k value for maximum alignment rate are different, largely depending on the correlation between the

Table 8 **Lerna** results on real PacBio *E.Coli K*-12 reads

k	Test PPL	NG50
15	240.57	101,440
17	**240.53**	133,690
19	240.59	**181,898**
21	240.62	166,229
23	240.65	92,537
25	240.67	92,859
27	240.69	74,776
31	240.70	58,332
37	240.64	32,160

Simulated annealing finds $k = 17$ as the best *k* value, which is also evident from the fact that it generated the minimum test perplexity. This value is quite close to $k = 19$ that generates the highest NG50 on assembly. Both of these *k*-values are highlighted in the Table.

Table 9 **Lerna** results on real Nanopore *Acinetobacter baumannii* reads

k	Test PPL	NG50
11	221.14	39,916
13	**192.64**	**41,358**
15	221.00	40,271
17	222.93	40,452
19	222.90	40,478
21	222.76	40,491
23	223.10	40,484
25	222.96	40,477
27	222.75	40,489
31	222.69	40,541
37	222.38	40,541
45	223.01	40,516

Simulated annealing here finds $k = 13$ as the best *k* value that also generates the highest NG50 on assembly, both of which are highlighted.

two metrics. We have, therefore, chosen $|w| = 4$ for evaluating **Lerna** on Nanopore simulated long reads as well, reported in Table 7.

Finally, we evaluate the improvement in assembly quality for the reads after correction using **Lerna** versus the original set of reads (uncorrected). We use SPAdes assembler to perform the assembly and generate the contigs, and we use QUAST measuring tool to estimate the NG50 for the generated contigs. The uncorrected reads had an NG50 of 26,034, whereas the corrected reads with LoRDEC+**Lerna** had an improved NG50 of 38,466, showing an improvement of 47% (over uncorrected reads). This shows the importance of applying **Lerna**'s pipeline to improve the performance for both alignment and assembly tasks. Compare this improvement to the improvement of 5.08× in assembly quality with short reads. This difference is due to the fact that assembly with shorter uncorrected reads is significantly worse than assembly with longer uncorrected reads. This is due to the fact that assembly is a harder problem with shorter reads, akin to piecing a jigsaw puzzle with a larger number of pieces.

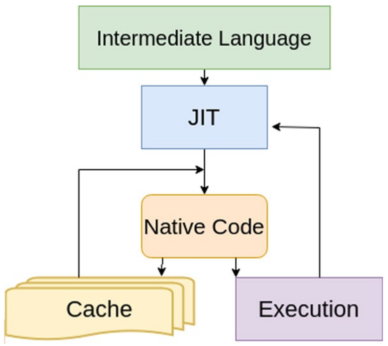

Fig. 3 A JIT compiler, which leverages the qualities of both static compilers and interpreters, has information about the system during execution, enabling better optimizations

Table 10 **Lerna** and Athena run on 7 Illumina short read datasets

Dataset	Coverage	Genome	Read length (bp)	#Reads	Athena	Lerna	Speedup
D1	80×	*E. coli* str. K-12 substr	136	20.8M	98s	5.5s	17.8×
D2	71×	*E. coli* str. K-12 substr	47	7.1M	49s	3s	16.3×
D3	173×	*Acinetobacter* sp. ADP1	36	18.1M	69s	4s	17.25×
D4	62×	*B. subtilis*	75	3.5M	52s	3s	17.3×
D5	166×	*L. interrogans C* sp. ADP1	100	7.1M	100s	5.5s	18.2×
D6	70×	*A. thaliana*	250	33.6M	400s	23s	17.4×
D7	67×	*Homo sapiens*	101	202M	960s	63s	15.2×

The time for calculating perplexity has been reported, along with the read lengths and number of reads. We observe that on average our pipeline is 18× faster than Athena. This translates to 80× to 275× faster than estimating the alignment rate with Bowtie2

We evaluate **Lerna** on two real datasets –*E. Coli K-12* PacBio reads (Accession number: SRX4909245) and *Acinetobacter baumannii* Nanopore reads (Accession number: SRX11521539). The data was corrected with LoRDEC and assembly was done using Canu. The data was tokenized into words with $|w| = 4$ for the training and testing of the language model. Tables 8 and 9 show the evaluation results of **Lerna** on these datasets. We have performed experiments with the maximum range of k values that each of LoRDEC and Canu could support on our machine.

Since **Lerna** uses a heuristic-based optimization strategy (i.e., Simulated annealing), finding the global optimum is not guaranteed. However, for all evaluated datasets, **Lerna** was able to tune different EC tools and find k-mer sizes that significantly improve the quality of the corrected reads when the assembly results of the corrected reads are compared across the different k-mer sizes. For example, in Table 8, $k = 17$ generates an NG50 value of 133,690 which is significantly better than for example, $k = 37$ which has an NG50 value of 32,160. Accordingly, it also improves the quality of assembly exemplified by the improvement in the NG50 metric. **Lerna** finds a value that is either identical or very close to the actual best k, but without relying on a reference genome. For example, we see in Table 8 that for the PacBio dataset, **Lerna** finds $k = 17$ as the optimal k whereas the actual best value was $k = 19$. In Table 9, we see that for the Nanopore dataset, **Lerna** finds the k value that results in the best assembly. Hence, we can conclude that **Lerna** successfully tunes LoRDEC on real long reads datasets as well.

We use Canu [57] for error correction on Nanosim [58] data. The Nanopore simulated reads are of *E. coli*. Using **Lerna**, we try to find the best value of MhapMerSize, i.e., the k-mer size for seeds in the MHAP part of Canu. Through our experiments, we found that a word length of 4 ($|w| = 4$) gives the optimum value of $k = 19$ and a corresponding 9,813 out of 10,000 reads, which aligned with the reference genome. This kind of configuration search is done in parallel using the Apache Spark framework as described in our prior work [59]. This is an example of what is called an embarrassingly parallel workload. The number of reads aligned and the corresponding value of MhapMerSize is given in Table 7. The mean perplexity output from the LM is 293.11, with a standard deviation of 0.16836, and the correlation between the aligned reads and perplexity is − 0.889. Hence, we show that **Lerna** works on both short- and long-read datasets.

Just-in-time compilation

We further improve the inference time of **Lerna** using Just-in-Time compilation [60]. The way bytecodes get converted to the appropriate native instructions for an application has a huge impact on the speed of an application. As seen in Fig. 3, they have access to run time information, such as, input parameters, control flow, and target machine specifics. While JIT compilers provide good performance, their performance is often a blackbox [61]. We employ JIT compilation in **Lerna** and find an immediate beneficial effect in the inference time. On using JIT compilers on Transformer LMs, we observe that the GPU utilization increases from 7 to 50%, and power usage of the GPU increases by 22W. This increase signifies that the JIT compiler allows for data parallelization in the GPU; more *data streams* and hence more GPU cores are being concurrently used. Clearly, by using JIT compilers, we have exploited the fact that the self-attention mechanisms can be parallelized, resulting in higher code runtime efficiency.

While the GPU utilization for a transformer increases ∼7× after using a JIT compiler, for an RNN LM, the power usage and GPU utilization are not improved. The Transformer efficiently uses the GPU by allowing computations to occur in parallel. The consequences of using an LM, which has parallelizable attention mechanisms, along with optimizations made by the JIT compiler, is that inference time for the Transformer is improved. The inferences are made using an NVIDIA Tesla P100 16GB GPU. We find that **Lerna** (using Transformer) is 18× faster than Athena (using RNN) (Table 10). The gains are quite consistent across the different datasets that span a wide range of characteristics (such as coverage and number of reads).

Discussion

We have presented **Lerna** that can automatically tune EC tools by exploring their parameter search space, using data-driven techniques without the need for a reference genome for the sequence being corrected. We have made use of the fact that trained language models (LMs) generate lower perplexity when used to evaluate corrected reads. Further, given that finding the optimal configuration traverses a non-convex trajectory, we have used simulated annealing to reduce the chances of getting stuck in a local minima. Further, some EC tools may have performance-sensitive configuration parameters that may have interdependencies and, in such cases, a surrogate model that can encode dependencies, as in our recent work [62, 63], can be used. Finally, we have compared **Lerna** 's findings with ground truth results on Illumina short reads and PacBio and Nanopore long reads. **Lerna** requires the user to first train an LM and then use the model to evaluate the corrected reads. Training can be a slow process and is directly proportional to the number of reads sampled for training. Our experiments have shown that **Lerna** works successfully even when as low as 5% of the input data is used for training. The user, therefore, needs to decide between the sample size and the training time. This is because a larger sample size is expected to have a larger input coverage that can result in a more rigorously trained model. Training is however required to be done only once, and the trained model is used to evaluate the corrected reads multiple times as **Lerna** searches through the parameter space of the EC tool being used. Evaluation is fast, and works in the order of a few seconds, as reported in Table 10.

The EC tool needs to be run for every input parameter that **Lerna** suggests as it searches through the parameter space. The runtime of the EC tool depends not only on the tool and the dataset, but also on the value of the input parameter. For example, the runtime of LoRDEC increases exponentially as k decreases. Therefore, the region of the parameter space to be explored must be carefully selected for the target EC tool. If the region is left too wide, then the search space becomes large and the search time, correspondingly, becomes long. If, on the other hand, the region is constrained too much, sub-optimal solutions may be found by **Lerna**. Our analysis in Table 6 captures some interesting empirical results. For example, choosing a word length of 4 for LM training produces better results than any other word length. This observation is consistent across multiple datasets. "This motivates us to study in future work the effect of word length on LM training in greater detail."

Conclusion

Configuration parameters play a crucial role in tuning EC tools. Manually selecting configuration parameters will degrade the quality of error correction (EC), and impact downstream genome assemblies [64]. **Lerna** automatically tunes these parameters without the need for a ground truth reference genome. Further, we improve our pipeline by leveraging character-level and word-level transformer networks. We observe that grouping nucleotides into groups of four ($|w| = 4$) in our LM causes the highest negative correlation with aligned reads, and gives the best results for both Lighter [18] and LoRDEC [11], exemplar short- and long-read EC tools, respectively. In all cases, **Lerna** can configure the EC tool so that the alignment rate is what would be found by an exhaustive search of the parameter space or the alignment rate is within 1% of the optimum. Further, **Lerna** has significant speed up of state-of-practice (actually doing the alignment, such as Bowtie2, over which we have a $80\times$ – $275\times$ speedup) or the state-of-the-art autotuner for genomic reads, namely *Athena* ($18\times$ speedup).

Further, the LMs we have developed may potentially impact future work, unraveling the nuances of the language of the genome. The use of attention mechanisms opens up several avenues toward the interpretability of LM. The values of the attention weights can be analyzed; we expect that beyond a certain distance from a token, the weights will fall below a certain threshold. By finding and quantifying an appropriate threshold, we can reduce the work of the LM by preventing it from encoding information beyond a certain point. Furthermore, the patterns observed through attention may allow future work to explore and find biologically significant portions of the genome. This can play a critical role in the analysis of genetic mutations and their consequences. **Lerna** can be applied to tune the configuration of many different EC tools. Due to its speed, it can be applied when datasets change or the instruments change, even transiently, to have different error characteristics.

We now discuss a few aspects of **Lerna**, such as supporting EC tools, which are not *k*-mer based, the impact of sub-sampling on the perplexity metric, and the tuning complexity for the hyperparameters in **Lerna**.

Support for non k-mer based EC tools Although our evaluation is performed on *k*-mer-based EC tools, such as Lighter and LoRDEC, the rationale for **Lerna** is not tightly coupled to *k*-mer-based correction techniques. Rather, it can be generalized to tune other parameters that impact the correction accuracy. This is achieved by **Lerna** 's pipeline that treats the EC tool under consideration as a blackbox. For example, tools such as Racer [65], which require users to specify the *genome length* for the sequenced portion of the genome, can also be tuned with **Lerna** 's pipeline by applying **Lerna** 's searching heuristic over the space of possible *genome lengths* (instead of the space of *k*-mer values). The selection criteria will be selecting the *genome length* that minimizes the perplexity metric for the corrected reads.

Impact of Sub-sampling on Perplexity As shown in Fig. 1, **Lerna** performs sub-sampling over the original set of reads before feeding it to the EC tool being tuned. This is performed to reduce the searching time. However, **Lerna** 's LM is trained over the *entire* (uncorrected) dataset to exploit the high coverage of the full set of reads, which allows the LM to accurately capture the probability distribution of the sequence. The sub-sampling ratio is lower bounded so that the resulting coverage of the sample is $\geq 30\times$, which is essential for the EC tool to perform efficient error correction [66]. Also notice that **Lerna** uses the perplexity scores that correspond to different *k*-values to pick the best among them. Accordingly, even if sub-sampling is impacting the perplexity scores, as long as the relative ordering of the perplexity scores with different *k*-values is maintained, **Lerna** can still select the best *k*-value.

Hyperparameter tuning for **Lerna** *'s search heuristic* **Lerna** relies on a Simulated Annealing (SA)-based searching to find the best *k*-value. However, SA has its own hyperparameters, such as initial temperature (T), temperature scale factor (α), and number of cycles. It may appear that we have replaced one parameter selection for a different, yet equally hard, parameter selection problem. Fortunately, we find empirically that **Lerna** 's performance is not sensitive to these parameters and a reasonable selection for their values (using common rules of thumb) provides the desirable performance. For example, a good rule of thumb for the "Initial Temperature" is to pick a value that accepts about 98% of the solutions to explore. As we can see from Fig. 4, the selection made by **Lerna** stays constant for any value of temperature between 2.00 and 3.25, and even outside of that, the fall in alignment quality is not that high. Figure 5 shows that the final choice of **Lerna** is relatively unaffected by the other parameter of SA, α. Moreover, prior works

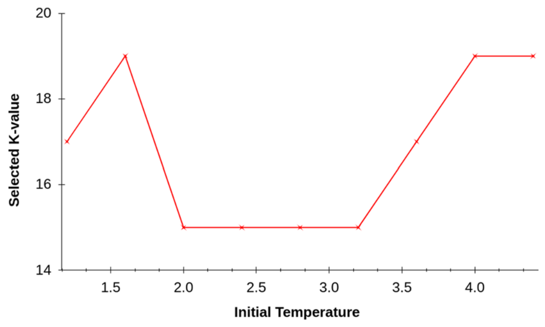

Fig. 4 The selected value of *k*-mer length for Illumina dataset *D2* (Table 2). The value of the parameter selected is relatively unaffected by the initial temperature (*T*). In the worst case, the alignment rate 61.15% compared to 61.42% at the optimum value. In this experiment, we set $\alpha = 0.7$

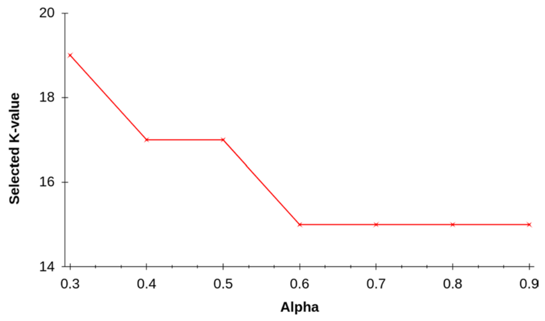

Fig. 5 Variation of selected *k* with parameter α keeping the initial temperature $(T) = 2.8$. In this case we observe that any α greater than 0.6 and less than 1 gives the optimum value for dataset *D2*

such as [67] proposed simple algorithms that can automatically select appropriate values for SA's hyperparameters, which can be easily integrated in **Lerna** 's pipeline.

Relevance of **Lerna** *with newer genome sequencing tools* As genomic reads become longer, the improvement due to **Lerna** will become more significant. This will happen because the longer-read technologies are more error prone, with PacBio and Oxford Nanopore reaching error rates of 15% and 40% [7, 8] respectively. Compare this to the error rates of less than 0.1% with short reads [68]. Second, with longer reads, the possible range of *k*-values in *k*-mer-based EC tools will increase. This will mean that it will be more time consuming to perform an exhaustive search through the parameter search space. Hence, the efficient design of **Lerna** will be of greater benefit to the community.

Methods

The proposed method for automatic tuning of EC Tools takes in the uncorrected reads and the EC Tool as the input, and involves the following 4 steps before generating the optimal input parameter (*k*) for the given EC Tool.

1 *Sub-sampling* The reads are usually obtained in the *fasta* file format. We sample out a small portion of the dataset, which typically constitutes 4–5% of the entire dataset. For example, for the dataset D7 (short reads, human genome), we sampled out 700k

reads out of 15M reads to work on. We have empirically found that this fraction has good enough coverage for the entire pipeline to be reliable. The sub-sampling step helps us achieve a significant improvement in runtime.

2 *LM training* We explore both character as well as word level training for both short and long reads. We train the transformer network, that is, our language model on the uncorrected subsampled dataset and use the trained model for evaluating the corrected reads.

3 *Data correction* Here we use the standard EC Tools to correct the uncorrected reads. We use Lighter for short reads and LoRDEC for long read correction. In principle, any EC tool can be used with **Lerna**. The EC Tool generates different outputs for different input *k* values and our aim is to find the *k* that corresponds to the best correction.

4 *Evaluation* We take the corrected outputs and evaluate them using our trained language model. To accelerate the process of evaluation, we have made use of JIT compiler. Our metric of evaluation is perplexity that essentially captures how confident the learned model is during prediction. A lower perplexity corresponds to a higher confidence and vice-versa. We repeat steps 3 and 4 iterating with different *k* values using a search technique called 'simulated annealing' to finally converge into the most optimal *k* value. We verify that the output of the EC Tool for this *k* value corresponds to the best alignment with the reference genome.

The entire workflow is visually represented in Fig. 1.

We show the steps of our searching mechanism in algorithms 1 and 2 . A Transformer LM is trained on a given dataset. The range of k; k_{min} and k_{max}, the LM, and the subsampled dataset S' are given as input to the algorithm. We set the parameter δ (i.e., step size) $= 1$, the initial temperature $T_0 = 2.8$, and the constant $\alpha = 0.7$ (α represents the factor by which temperature is scaled after each cycle). These parameters are empirically found to perform the best across all datasets (including short and long reads) and therefore no additional tuning efforts are needed here. The search is allowed to run for 8 cycles, with 3 trials per cycle.

Algorithm 1 Correct Set of Reads

```
Input: Dataset: $D_0$, Minimum value of $k$: $k_{min}$, Maximum value of $k$: $k_{max}$
Output: Corrected Dataset: $D_c$
1- Train a Transformer Language Model (LM) using $D_0$.
2- Select random sample $S'$ from $D_0$.
3- Call Search $k(S'$, LM, $k_{min}$, $k_{max})$: Algorithm 2
4- Use the value of $k$ from step 3 and do a complete correction on the entire dataset $D_0$ and return $D_c$.
```

The probability of selecting a proposed solution that corresponds to a higher perplexity (a less likely sequence) is given by $p = \exp(-\Delta E/(T \times \Delta E_{avg}))$, where E, the energy variable, determines the volume of the search space. Larger E values correspond to larger search space and viceversa. It is clear that as temperature T is reduced, p is reduced. Hence, the values assigned to parameters T_0 and α determine the overall performance of the search. If the temperature is reduced quickly over various cycles, by reducing α, the search is restricted to a local neighborhood (exploitation). However, taking a value of α close to 1 causes the algorithm to explore a greater search space, and prevents it from converging to a local optimum.

Algorithm 2 Search k

```
Input: Dataset: $S'$, Language Model: LM, Minimum value of $k$: $k_{min}$, Maximum value of $k$: $k_{max}$
Output: Best $k$ : $k^*$
1- Randomly chose a value $k_c$ in range $(k_{min}, k_{max})$.
2- Initialize temperature T and constant $\alpha$ in range (0,1)
3- Initialize $\Delta E_{avg} = 0$, $n_a = 0$ and constant $\delta$
4- Initialize the number of cycles and the number of trials in each cycle
5- for i in the range(0,cycles)
6-     for j in the range(0,trials)
7-         Select random integer $k_r$ in range $(max(k_{min}, k_c - \delta), min(k_{max}, k_c + \delta))$
8-         Calculate perplexity $f(LM, S', k_r)$ and $f(LM, S', k_c)$
9-         Calculate $\Delta E = |f(LM, S', k_r) - f(LM, S', k_c)|$
10-         if $f(LM, S', k_r) > f(LM, S', k_c)$ Then:
11-             if i=0 and j=0 Then:
12-                 $\Delta E_{avg} = \Delta E$
13-             $p = \exp(-\Delta E/(T \times \Delta E_{avg}))$
14-             Randomly select $\beta$ in range (0,1)
15-             if $\beta < p$ Then:
16-                 Set value of $k_c = k_r$
17-         else
18-             Set value of $k_c = k_r$
19-         Increment $n_a$
20-         Calculate $(\Delta E_{avg} \times (n_a - 1.0) + \Delta E)/n_a$
21-         Update the value of $\Delta E_{avg}$ with value from step 20
22-     Update temperature T = $T \times \alpha$
23- return $k_c$
```

Abbreviations
JIT: Just in time; EC tool: Error correction tool; NLP: Natural language processing; LM: Language model; RNN: Recurrent neural network; ML: Machine learning; CLR: Continuous long reads; SA: Simulated annealing.

Acknowledgements
None.

Author's contributions
A.S., P.J., A.M., Z.Z., K.M., and S.C. participated in the computational analyses. A.S worked on the language model training and long reads analysis. P.J. worked on the short reads analysis, simulated annealing, and language model testing speedup using JIT compiler. A.M. assisted with the EC tool usage, and Z.Z. evaluated the entire pipeline on real datasets reported in Tables 8 and 9. K.M. helped with finding the right datasets for the work. All authors wrote the manuscript. S.C. provided overall guidance and funding for the project. All authors read and edited the final manuscript.

Funding
This work is supported in part by the NIH R01 Grant 1R01AI123037, a Lilly Endowment grant, a gift from Adobe Research, and funding from Purdue's College of Engineering and Department of Agricultural and Biological Engineering (ABE). Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

Availability of data and materials
Our code has been made available at a public github repository at: https://github.com/icanforce/lerna-genomics.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
No competing interests by Adobe Systems Inc. Prof. Chaterji is an Associate Editor of BMC Bioinformatics.

Author details
[1]Purdue University, West Lafayette, US. [2]Indian Institute of Technology Bombay, Mumbai, India. [3]Adobe Research, San Jose, US.

Received: 7 April 2021 Accepted: 20 December 2021
Published online: 06 January 2022

References

1. Biosciences P. Detecting DNA base modifications using single molecule, real-time sequencing. White Paper Base Modifications. 2015.
2. Eisenstein M. Oxford Nanopore announcement sets sequencing sector abuzz. Berlin: Nature Publishing Group; 2012.
3. Schadt EE, Turner S, Kasarskis A. A window into third-generation sequencing. Hum Mol Genet. 2010;19(R2):227–40.
4. Mahadik K, Wright C, Kulkarni M, Bagchi S, Chaterji S. Scalable genome assembly through parallel de Brujin graph construction for multiple k-mers. Sci Rep. 2019;9(1):1–15.
5. Fu S, Wang A, Au KF. A comparative evaluation of hybrid error correction methods for error-prone long reads. Genome Biol. 2019;20(1):26.
6. Laehnemann D, Borkhardt A, McHardy AC. Denoising DNA deep sequencing data-high-throughput sequencing errors and their correction. Brief Bioinform. 2016;17(1):154–79.
7. Korlach J. Understanding accuracy in smrt® sequencing. Pac Biosci. 2013;1–9.
8. Laver T, Harrison J, O'Neill P, Moore K, Farbos A, Paszkiewicz K, Studholme DJ. Assessing the performance of the Oxford nanopore technologies minion. Biomol Detect Quantif. 2015;3:1–8.
9. Amarasinghe SL, Su S, Dong X, Zappia L, Ritchie ME, Gouil Q. Opportunities and challenges in long-read sequencing data analysis. Genome Biol. 2020;21(1):1–16.
10. Berlin K, Koren S, Chin C-S, Drake JP, Landolin JM, Phillippy AM. Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. Nat Biotechnol. 2015;33(6):623.
11. Salmela L, Rivals E. LoRDEC: accurate and efficient long read error correction. Bioinformatics. 2014;30(24):3506–14.
12. Yang X, Dorman KS, Aluru S. Reptile: representative tiling for short read error correction. Bioinformatics. 2010;26(20):2526–33.
13. Liu Y, Schröder J, Schmidt B. Musket: a multistage k-mer spectrum-based error corrector for Illumina sequence data. Bioinformatics. 2013;29(3):308–15.
14. Heo Y, Wu X-L, Chen D, Ma J, Hwu W-M. Bless: bloom filter-based error correction solution for high-throughput sequencing reads. Bioinformatics. 2014;30(10):1354–62.
15. Benoit G, Lavenier D, Lemaitre C, Rizk G. Bloocoo, a memory efficient read corrector. In: European conference on computational biology (ECCB). 2014.
16. Lim E-C, Müller J, Hagmann J, Henz SR, Kim S-T, Weigel D. Trowel: a fast and accurate error correction module for Illumina sequencing reads. Bioinformatics. 2014;30(22):3264–5.
17. Akogwu I, Wang N, Zhang C, Gong P. A comparative study of k-spectrum-based error correction methods for next-generation sequencing data analysis. Hum Genom. 2016;10(2):20.
18. Song L, Florea L, Langmead B. Lighter: fast and memory-efficient sequencing error correction without counting. Genome Biol. 2014;15(11):509.
19. Abdallah M, Mahgoub A, Ahmed H, Chaterji S. Athena: automated tuning of k-mer based genomic error correction algorithms using language models. Sci Rep. 2019;9(1):1–13.
20. Kao W-C, Chan AH, Song YS. Echo: a reference-free short-read error correction algorithm. Genome Res. 2011;21:1181–92.
21. Chikhi R, Medvedev P. Informed and automated k-mer size selection for genome assembly. Bioinformatics. 2013;30(1):31–7.
22. Langmead B, Salzberg SL. Fast gapped-read alignment with Bowtie 2. Nat Methods. 2012;9(4):357–9.
23. Baichoo S, Ouzounis CA. Computational complexity of algorithms for sequence comparison, short-read assembly and genome alignment. Biosystems. 2017;156:72–85.
24. Trinh TH, Dai AM, Luong M-T, Le QV. Learning longer-term dependencies in RNNs with auxiliary losses. 2018. arXiv:1803.00144.
25. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. In: Advances in neural information processing systems. 2017;5998–6008.
26. Zhai C. Statistical language models for information retrieval. Synth Lect Hum Lang Technol. 2008;1(1):1–141.
27. Agarwal A, Xie B, Vovsha I, Rambow O, Passonneau RJ. Sentiment analysis of twitter data. In: Proceedings of the workshop on language in social media (LSM 2011). 2011;30–38.
28. Elaraby MS, Abdallah M, Abdou S, Rashwan M. A deep neural networks (DNN) based models for a computer aided pronunciation learning system. In: International conference on speech and computer. Springer. 2016;51–58.
29. Lhoussain AS, Hicham G, Abdellah Y. Adaptating the Levenshtein distance to contextual spelling correction. Int J Comput Sci Appl. 2015;12(1):127–33.
30. Siivola V, Pellom BL. Growing an n-gram language model. In: Ninth European conference on speech communication and technology. 2005.
31. Schwenk H, Dechelotte D, Gauvain J-L. Continuous space language models for statistical machine translation. In: Proceedings of the COLING/ACL on main conference poster sessions. Association for Computational Linguistics. 2006;723–730.
32. Brown PF, Desouza PV, Mercer RL, Pietra VJD, Lai JC. Class-based n-gram models of natural language. Comput Linguist. 1992;18(4):467–79.
33. Kombrink S, Mikolov T, Karafiát M, Burget L. Recurrent neural network based language modeling in meeting recognition. In: Twelfth annual conference of the international speech communication association. 2011.
34. Pascanu R, Mikolov T, Bengio Y. On the difficulty of training recurrent neural networks. In: International conference on machine learning. 2013;1310–1318.
35. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997;9(8):1735–80.

36. Chung J, Gulcehre C, Cho K, Bengio Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. 2014. arXiv:1412.3555.
37. Liu Y, Ott M, Goyal N, Du J, Joshi M, Chen D, Levy O, Lewis M, Zettlemoyer L, Stoyanov V. RoBERTa: a robustly optimized BERT pretraining approach. 2019. arXiv:1907.11692.
38. Luong M-T, Pham H, Manning CD. Effective approaches to attention-based neural machine translation. 2015. arXiv:1508.04025.
39. Cheng J, Dong L, Lapata M. Long short-term memory-networks for machine reading. 2016. arXiv:1601.06733.
40. Parikh AP, Täckström O, Das D, Uszkoreit J. A decomposable attention model for natural language inference. 2016. arXiv:1606.01933.
41. Paulus R, Xiong C, Socher R. A deep reinforced model for abstractive summarization. 2017. arXiv:1705.04304.
42. Lin Z, Feng M, Santos CNd, Yu M, Xiang B, Zhou B, Bengio Y. A structured self-attentive sentence embedding. 2017. arXiv:1703.03130.
43. Azzopardi L, Girolami M, Van Risjbergen K. Investigating the relationship between language model perplexity and IR precision-recall measures. In: Proceedings of the 26th annual international acm sigir conference on research and development in information retrieval. 2003. p. 369–370.
44. Shlens J. Notes on Kullback–Leibler divergence and likelihood. 2014. arXiv:1404.2000.
45. Sbert M, Chen M, Poch J, Bardera A. Some order preserving inequalities for cross entropy and Kullback–Leibler divergence. Entropy. 2018;20(12):959.
46. Devlin J, Chang M-W, Lee K, Toutanova K. Bert: pre-training of deep bidirectional transformers for language understanding. 2018. arXiv:1810.04805.
47. Medina JR, Kalita J. Parallel attention mechanisms in neural machine translation. In: 2018 17th IEEE international conference on machine learning and applications (ICMLA). 2018. p. 547–552. IEEE
48. Branton D, Deamer DW, Marziali A, Bayley H, Benner SA, Butler T, Di Ventra M, Garaj S, Hibbs A, Huang X, et al. The potential and challenges of Nanopore sequencing. In: Nanoscience and technology: a collection of reviews from nature journals. World Scientific. 2010. p. 261–268.
49. Miclotte G, Heydari M, Demeester P, Rombauts S, Van de Peer Y, Audenaert P, Fostier J. Jabba: hybrid error correction for long sequencing reads. Algorithms Mol Biol. 2016;11(1):10.
50. Bao E, Lan L. Halc: high throughput algorithm for long read error correction. BMC Bioinform. 2017;18(1):204.
51. Kalai AT, Vempala S. Simulated annealing for convex optimization. Math Oper Res. 2006;31(2):253–66.
52. Prjibelski A, Antipov D, Meleshko D, Lapidus A, Korobeynikov A. Using SPADES de novo assembler. Curr Protoc Bioinform. 2020;70(1):102.
53. Gurevich A, Saveliev V, Vyahhi N, Tesler G. Quast: quality assessment tool for genome assemblies. Bioinformatics. 2013;29(8):1072–5.
54. Ono Y, Asai K, Hamada M. PBSIM: PacBio reads simulator-toward accurate genome assembly. Bioinformatics. 2013;29(1):119–21.
55. Li H. Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics. 2018;34(18):3094–100.
56. Conrad K. Probability distributions and maximum entropy. Entropy. 2004;6(452):10.
57. Koren S, Walenz BP, Berlin K, Miller JR, Bergman NH, Phillippy AM. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. Genome Res. 2017;27(5):722–36.
58. Yang C, Chu J, Warren RL, Birol I. Nanosim: nanopore sequence read simulator based on statistical characterization. GigaScience. 2017;6(4):010.
59. Ghoshal A, Grama A, Bagchi S, Chaterji S. An ensemble SVM model for the accurate prediction of non-canonical microRNA targets. In: Proceedings of the 6th ACM conference on bioinformatics, computational biology and health informatics (BCB). 2015. p. 403–12.
60. Aycock J. A brief history of just-in-time. ACM Comput Surv (CSUR). 2003;35(2):97–113.
61. Rompf T, Sujeeth AK, Brown KJ, Lee H, Chafi H, Olukotun K. Surgical precision JIT compilers. In: Proceedings of the 35th ACM SIGPLAN conference on programming language design and implementation. 2014. p. 41–52.
62. Mahgoub A, Wood P, Ganesh S, Mitra S, Gerlach W, Harrison T, Meyer F, Grama A, Bagchi S, Chaterji S. Rafiki: a middleware for parameter tuning of NoSQL datastores for dynamic metagenomics workloads. In: Proceedings of the 18th ACM/IFIP/USENIX middleware conference. ACM. 2017. p. 28–40.
63. Mahgoub A, Medoff AM, Kumar R, Mitra S, Klimovic A, Chaterji S, Bagchi S. OPTIMUSCLOUD: heterogeneous configuration optimization for distributed databases in the cloud. In: 2020 USENIX annual technical conference (USENIX ATC) 20). 2020. p. 189–203.
64. Heydari M, Miclotte G, Demeester P, Van de Peer Y, Fostier J. Evaluation of the impact of Illumina error correction tools on de novo genome assembly. BMC Bioinform. 2017;18(1):374.
65. Ilie L, Molnar M. Racer: rapid and accurate correction of errors in reads. Bioinformatics. 2013;29(19):2490–3.
66. Liu B, Shi Y, Yuan J, Hu X, Zhang H, Li N, Li Z, Chen Y, Mu D, Fan W. Estimation of genomic characteristics by analyzing k-mer frequency in de novo genome projects. 2013. arXiv:1308.2012.
67. Ben-Ameur W. Computing the initial temperature of simulated annealing. Comput Optim Appl. 2004;29(3):369–85.
68. Ma X, Shao Y, Tian L, Flasch DA, Mulder HL, Edmonson MN, Liu Y, Chen X, Newman S, Nakitandwe J, et al. Analysis of error profiles in deep next-generation sequencing data. Genome Biol. 2019;20(1):1–15.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.