ACTGap: LSTM and Transformer Decoder-Based DNA Sequence Imputation

Luke Nguyen (lnguye61), Ashley Xu (axu71), Romer Miranda (remirand), Khoi Le (kcle)

GitHub Repository: https://github.com/LukeLose/ACTGap.git

Introduction:

In biology, DNA sequencing refers to determining the exact order of nucleotides in a DNA fragment, using techniques like Next-Generation Sequencing (NGS). When an organism's genome is sequenced, it must be done in chunks, as current DNA sequencing technologies only work for certain lengths of reads. These reads are then aligned together into a reference genome for that organism. GRCh38, the latest human genome assembly, was generated by fragmenting the genome, sequencing the pieces, and then assembling them computationally.

We are trying to solve the problem of incomplete genome assemblies. Despite major advances in DNA sequencing, gaps remain in the human reference genome due to technological limitations, unresolved regions, and highly repetitive sequences that hinder accurate assembly into a continuous sequence. Previous tools made to fill in these gaps relied on greedy algorithms to solve the problem. However, these methods lack versatility and are unable to fully exploit all the sequencing data available. Some deep learning models are available, such as DNABert, but rely on heavy compute, encoder models. Thus, we propose a LSTM and Transformer Decoder approach to reconstruct missing DNA sequences in incomplete genome assemblies. The problem we are trying to solve is that of generation: we want our model to learn the underlying distribution of DNA sequence data and generate new sequences.

Methodology:

Data and Preprocessing:

K-mer Tokenization

For both of our approaches, we tokenized the original DNA sequence into a sequence of overlapping k-mers, with a k-mer length of 2 and stride of 1. Each k-mer was also mapped to a unique numerical ID after generating a k-mer dictionary. Below is an example tokenization:

Original Sequence:	ACTGGGAACATTA
K-mer Sequence:	AC CT TG GG GG GA AA AC CA AT TT TA
Tokenized Sequence:	1 6 11 15 15 12 0 1 4 2 10 8

Fig 1. Example of k-mer (2-mer) tokenization for a DNA sequence

We arrived at a k-mer length of 2 after testing various values of k > 1. K-mers are a common way in bioinformatics to analyze DNA sequence data, as they preserve contextual information in each token from the neighboring nucleotides, and this method is used in other DNA sequence models such as DNABERT2. In brief, we found that using a smaller value of k was ideal, as the vocabulary size scales exponentially with k, and with our limited time to train the models, the models learned quicker when there were less classes to predict.

LSTM-Based Model

We used sequences from an isolate of human chromosome 21 in GenBank fasta format, available from NCBI. Chromosome 21 is 45,090,682 bp long, but for our purposes, we used the first ~350,000 bp of the chromosome. The data was split into sequences of a certain window length, with a default value of 32. After k-mer tokenization, BEGIN and END tokens were included to provide start and stop context for the model. To simulate gaps in incomplete genome assemblies, we introduced artificial gaps through masking contiguous sequences of specified length in either random locations or at the end of each sequence. Masking allowed the model to know exactly where to predict the next token, and the loss and accuracy functions were calculated based only on the masked positions.

Transformer Decoder-Based Model

We used protein coding sequences from the CDS fasta downloaded from Ensembl. Sequences longer than 1024 bp were removed from the dataset, and padding tokens were added to the end of shorter sequences such that all input sequences were the same length. Preprocessing outputs were context and corresponding gap vectors each padded to be the size of the largest gene in the dataset (1024). There was still a significant subset of protein-coding genes below our max size, and thus, we limited it to 5000 protein-coding genes for reasonable training time. The data was shuffled with training, and the batch size was set to 16, a reduction reflecting the Transformer's more computationally intensive workflow.

Model Architecture and Training:

LSTM-Based Model

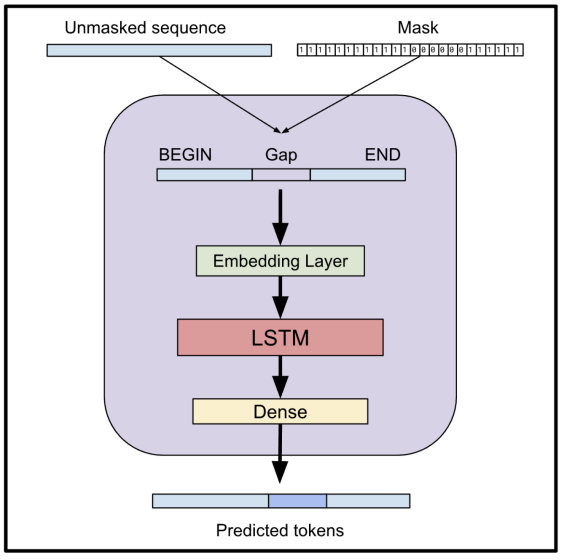

Fig 2. LSTM architecture model diagram

The model itself consists of an embedding layer, LSTM layer, and a dense layer to act as the classifier. The model was trained using the Adam optimizer with a learning rate of 0.0001 and sparse categorical cross entropy for the loss function. The data was split into a training and test fasta file, with 80% of the data being allotted for the training data and the remaining 20% for the testing data. We passed both the tokenized sequence as well as the mask, which consists of 0s and 1s where 0s indicated a masked or "gap" location and 1 indicating a

nucleotide being present, into the model. The model was trained for 3 epochs.

Transformer Decoder-Based Model

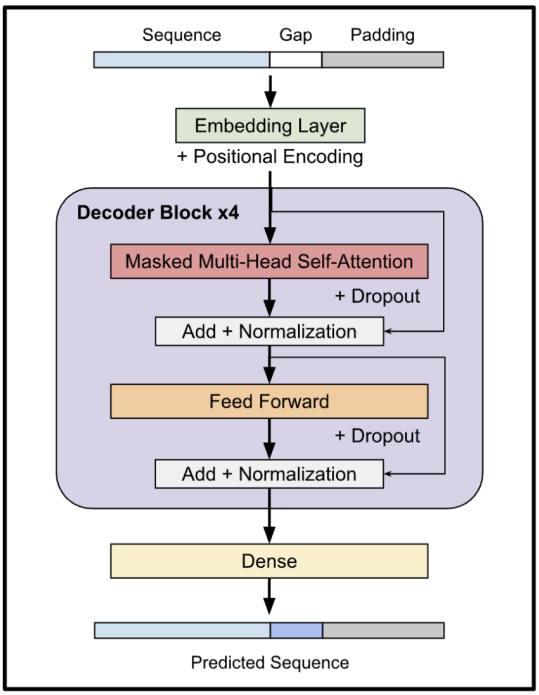

Fig 3. Transformer-decoder architecture model diagram

Our Transformer Decoder-Based Model includes a positional encoding layer, 4 Transformer Decoder Blocks, a normalization layer, and a classifier (dense) layer that produces logit values across the possible k-mer tokens. The Transformer Decoder Blocks each consist of a masked multi-headed self-attention layer, 2 feed-forward layers, 2 normalization layers, and a dropout layer. For hyperparameters, there are 8 attention heads, a dropout rate of 0.3, and an embedding size of 256. The model was trained using the Adam optimizer with a learning rate of 0.0003 and sparse categorical cross entropy for the loss function. Loss and accuracy were calculated based on only the gap sequence. The data was split into a training and testing fasta file, with 85% of the data being allotted for training and the remaining 15% for testing. The model was trained for 3 epochs.

Results:

LSTM-Based Model:

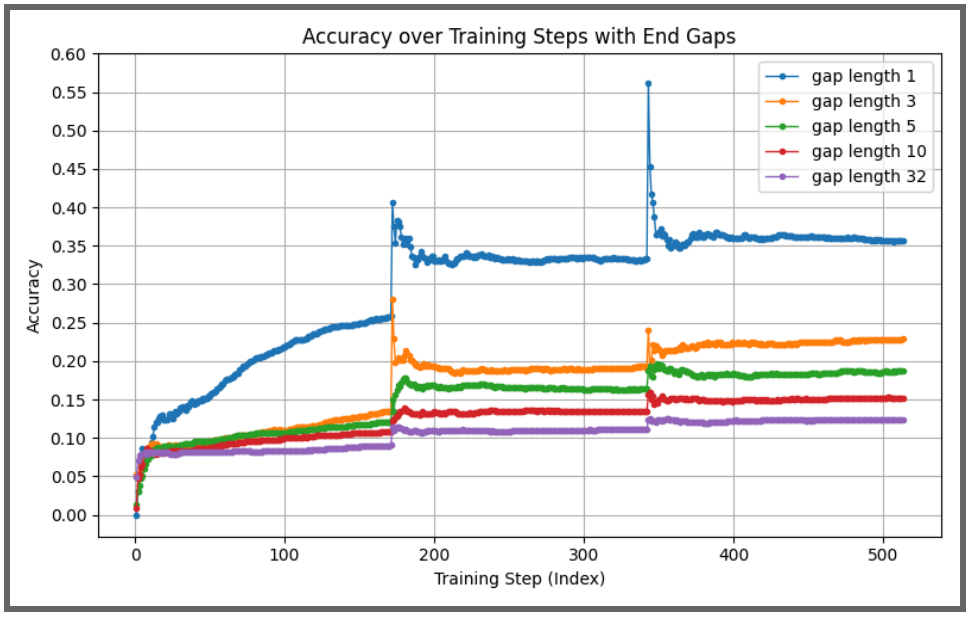

Fig 4. LSTM training accuracy with variable gap length at the end of the sequence

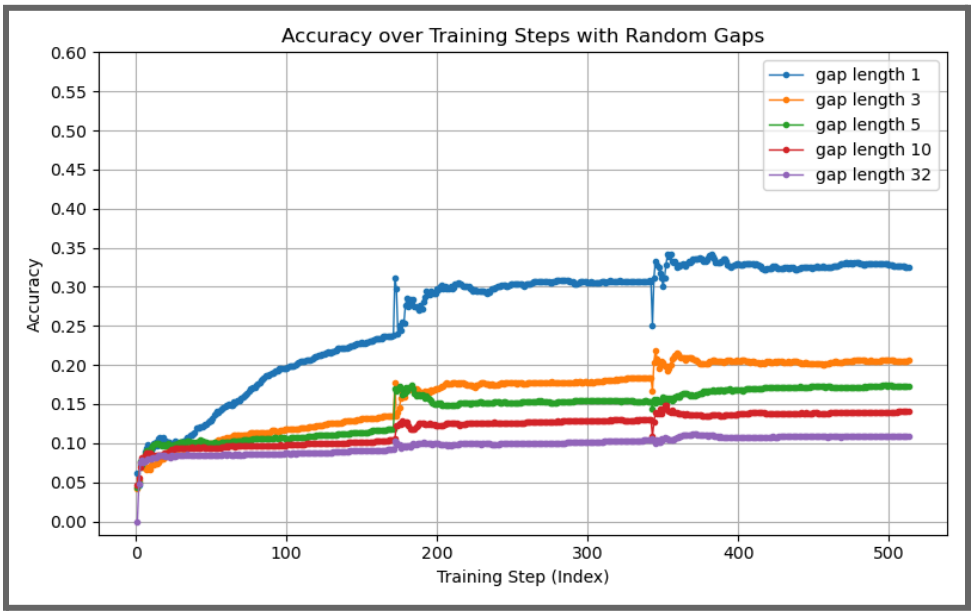

Fig 5. LSTM training accuracy with variable gap length at random position in the sequence

Within the LSTM model, we wanted to explore whether or not the location of the gap, as well as the gap length, affects the accuracy of our model. With random insertion of a contiguous gap, the model only examined the sequence up to the start of the gap, omitting any information after the gap. Therefore, we hypothesized that the model would better perform on data with gaps inserted at the end, as the model would have the most prior information available. The figures revealed that we were correct, as accuracies were slightly higher when adding gaps to the end compared to when they were randomly inserted. Additionally, smaller gaps were expected to have higher accuracy as the model has less to predict, and this was supported by the figures.

Below is an example of the LSTM model's prediction on a gap length of 10 k-mers, demonstrating the difficulty with which the model has with predicting sequences correctly.

Predicted Sequence:	5 4 5 15 15 15 15 15 15 15
Actual Sequence:	4 3 13 6 9 4 0 3 15 22

Fig 6. Example prediction from the LSTM-based model

Transformer Decoder-Based Model:

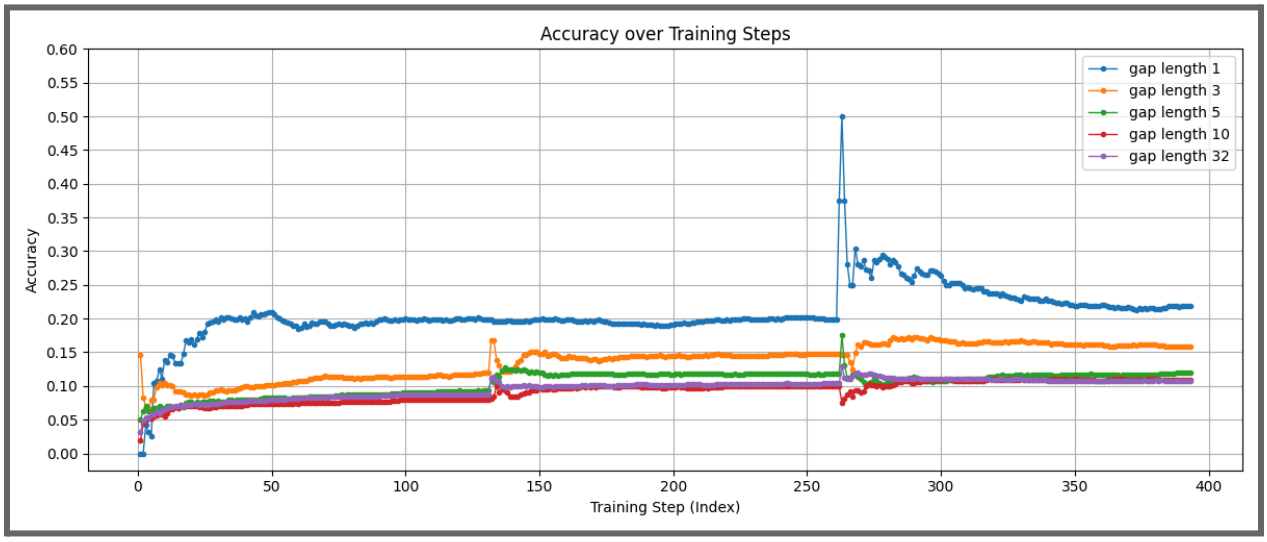

Fig 7.

We recognized that LSTM model limitations include its inability to capture long-range dependencies, and thus we implemented a transformer decoder based model. However, compared to the accuracy in both LSTM models, we get significantly worse performance across all gap lengths. In the LSTM, accuracy varied noticeably between gap lengths of 5, 10, and 32, but the decoder's accuracy collapsed onto each other—indicating it did not learn the gap-specific patterns. We believe this performance stems from insufficient training, where compute heavy decoders were penalized more heavily than LSTMs.

Challenges:

One challenge implementing the model was implementing the masking mechanism whereby parts of the input are masked and then the model has to predict the masked tokens. We went through various different implementations, but the way we ended up deciding on was to pass in

the full unmasked input to the model along with a vector of booleans with each boolean corresponding to whether the associated position in the input sequence is masked.

However, our biggest challenge during the project came from switching gears, from initially implementing a encoder-only transformer model to an LSTM and decoder-only transformer model. We decided on this switch after our initial encoder-only model, inspired by DNABert, had very low accuracies, leading us to believe that this model was not actually learning, and that there may be some error in our training process.

After this switch, a big topic of discussion was the k-mers. We soon realized that because we were using overlapping k-mers, predicting one k-mer limited the choices for the next k-mer, as the next k-mer had to start with the last k-1 characters of the previous k-mer. Additionally, we spent a lot of time deciding on what the length of the k-mer to be. Because increasing k-mer length exponentially increases the total number of possible k-mers, we learned that making the k-mer size too big would make the problem much harder for our model. We ultimately decided that in order to simplify the problem as much as possible, using k-mer size of 2 was the best approach, as it gave the model less classes to choose from when making predictions. This also boosted our new models' accuracies to greater than random guessing.

Reflection:

Overall, our project turned out satisfactory, achieving our base goal of training models to predict gap sequences in the human genome. Because we did not achieve as high of an accuracy as we initially predicted, our efforts were focused on improving our existing models instead of shooting for our target goal of testing our model on other species' reference genomes. In the end though, our models did not perform the way we expected them to, possibly due to limitations with time and compute. However, we were successful in training models that performed better than random chance. Initially, our approach was to use a BERT-style encoder-only model to predict DNA sequences, but that did not work because our model ended up guessing the same token for every masked position instead of learning the patterns in the genome. As a result, we shifted to a simpler LSTM model, which improved our accuracy to higher than random chance. We also tried out a transformer decoder only model, which also had more successful results than our initial approach but did not perform as well as the LSTM model.

If we were to do the project over again, we would have spent less time trying to make the encoder-only model work and tried out different architectures sooner instead of being bogged down on one. If we had had more time, we would have tested the results of training our models for more epochs and training our models on a larger dataset and/or longer sequences to see if the models would be able to learn more specific patterns and long-range dependencies in the human genome. In addition, we would have tested our model on the data of other species and

attempted to do model interpretation to see which input features in the DNA were important for predicting gap sequences.

We realized from this project that deep learning is not an exact science with a lot of trial and error, and that flexibility is needed when one approach is not working. After our initial encoder only transformer model did not perform as we expected, we had to take a step back, examine possible sources of error, and determine the best step forward. Transitioning from an encoder only based model to an LSTM and decoder only based model forced us to consider the differences between these different deep learning models, and why one might be better than another at solving a specific problem. Overall, we learned a lot about how deep learning models are used to tackle the problem of incomplete genome assemblies, both from reading previous implementations such as DNABert and GapPredict, and from attempting it ourselves!

References:

Ji, Y., Zhou, Z., Liu, H., & Davuluri, R. V. (2021). DNABERT: pre-trained Bidirectional Encoder Representations from Transformers model for DNA-language in genome. Bioinformatics, 37(15), 2112-2120. https://doi.org/10.1093/bioinformatics/btab083

Chen, E., Chu, J., Zhang, J., Warren, R. L., & Birol, I. (2021). GapPredict - A Language Model for Resolving Gaps in Draft Genome Assemblies. IEEE/ACM transactions on computational biology and bioinformatics, 18(6), 2802–2808. https://doi.org/10.1109/TCBB.2021.3109557