**PROTEIN ENGINEERING**

(Protein Sequence Generation)

[(Github)](https://github.com/En1gma02/Protein_Sequence_Generation)

-Team InfoMatrix

Akshansh Dwivedi

Pranav Pawar

Veer Raje

*2nd Year B.tech Artificial Intelligence(AI) & Data Science*

*Dwarkadas J. Sanghvi College of Engineering, Mumbai*

**Problem Understanding**

Introduction:

Proteins, the workhorses of biological systems, are crucial for numerous functions ranging from structural support to enzymatic activity. Protein engineering, the manipulation of protein sequences to achieve desired properties, holds immense promise in fields like drug discovery and biotechnology. Central to this endeavour is the generation of novel protein sequences with specific characteristics.

Objective:

This project aims to develop a model capable of generating realistic and meaningful protein sequences. Leveraging a dataset comprising diverse protein families, domains, and motifs, our goal is to employ machine learning, deep learning, or other computational approaches to decipher underlying sequence patterns. The generated sequences should be accurate, diverse, and biologically relevant, facilitating applications in drug design and protein engineering.

Approaches:

1. Utilising machine learning and deep learning techniques to learn sequence patterns and generate novel sequences.

2. Fine-tuning pre-trained language models tailored for protein sequence generation.

Challenges:

1. Sequence Patterns: Deciphering intricate sequence patterns underlying protein structure and function.

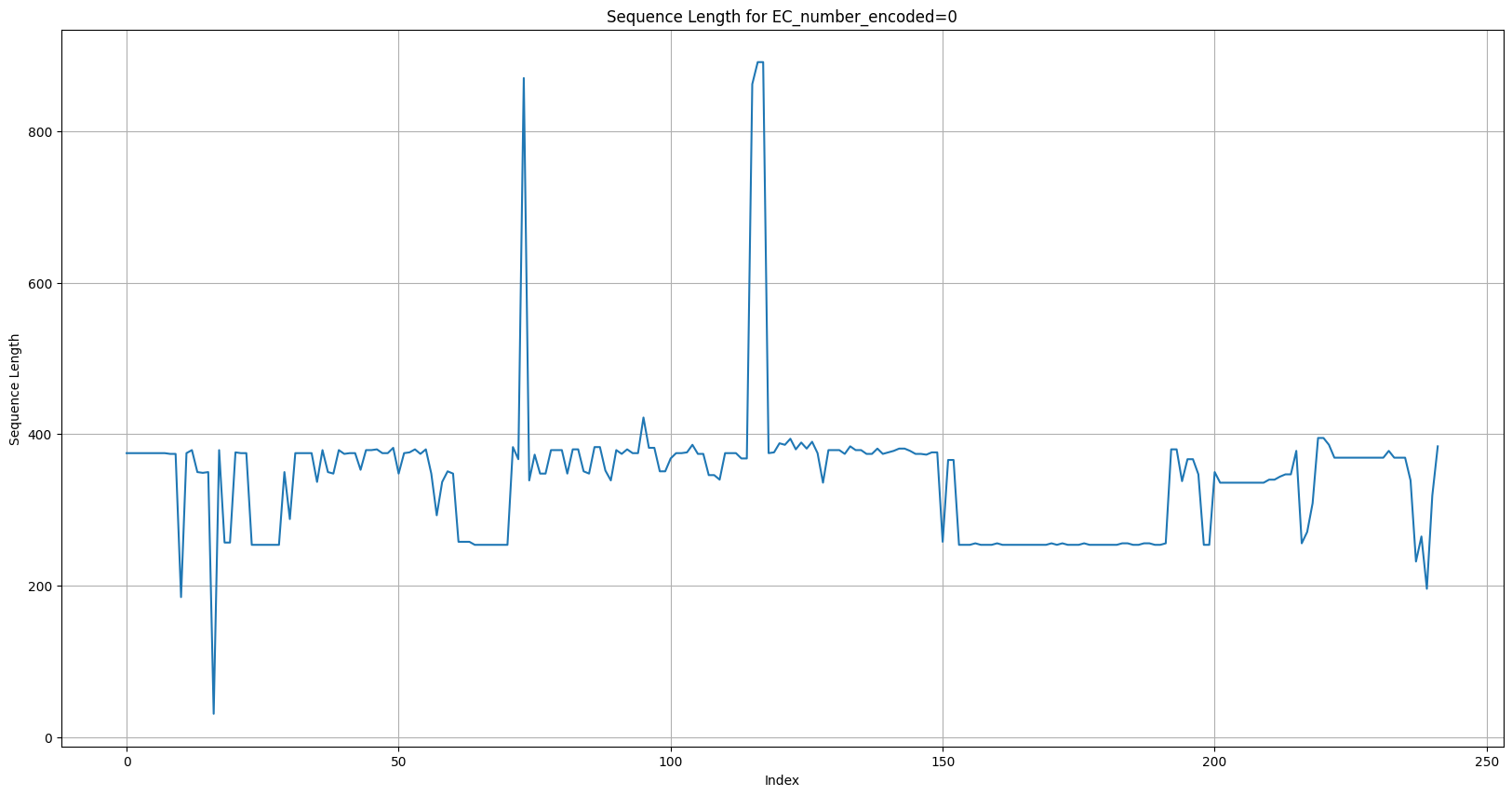
2. Biological Relevance: Ensuring generated sequences maintain functional integrity.

3. Computational Resources: Significant computational resources required for training models.

Significance:

Successfully addressing these challenges can revolutionise drug discovery, protein design, and biotechnology. Tailored protein sequences offer opportunities for personalised therapeutics and enhanced industrial processes.

In the subsequent sections, we will delve into data handling, methodology, evaluation metrics, benchmarking, future scope, and provide the GitHub link to access our code.



**Data Handling and Analysis**

Dataset Overview:

The dataset consists of 90,000 rows containing protein sequences along with associated metadata such as EC number, UniProt IDs, names, functions, genes, species, and sequence lengths.

Preprocessing Steps (splitting\_encoding\_visualization.ipynb):

1. Splitting and Encoding:

- The 'name' column was split into 'gene' and 'species' using '\_' as a delimiter.

- Label encoding was performed on 'EC\_number' and 'species' columns.

- Sequence lengths were calculated and added as a new feature.

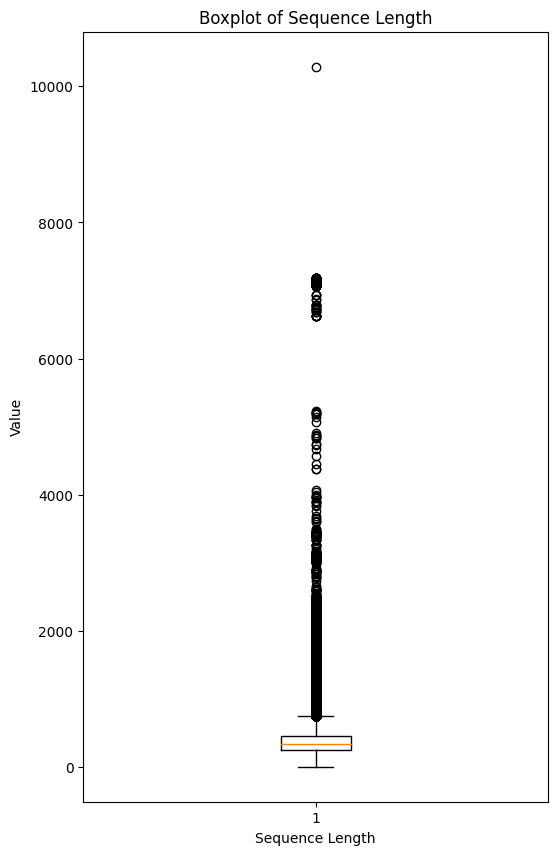
2. Outlier Detection:

- A box plot was generated to visualize the distribution of sequence lengths.

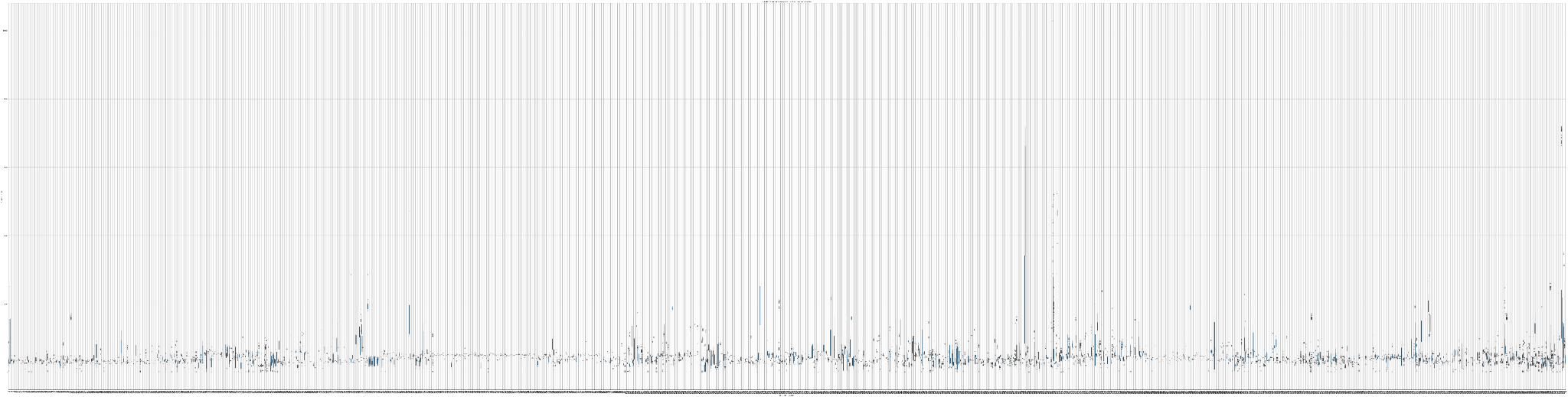
- Outliers were identified and addressed by filtering out sequences with length 0.

Line graph of length of sequences of EC\_number = ID 1.1.1.1

Box plot for sequence lengths of each row:

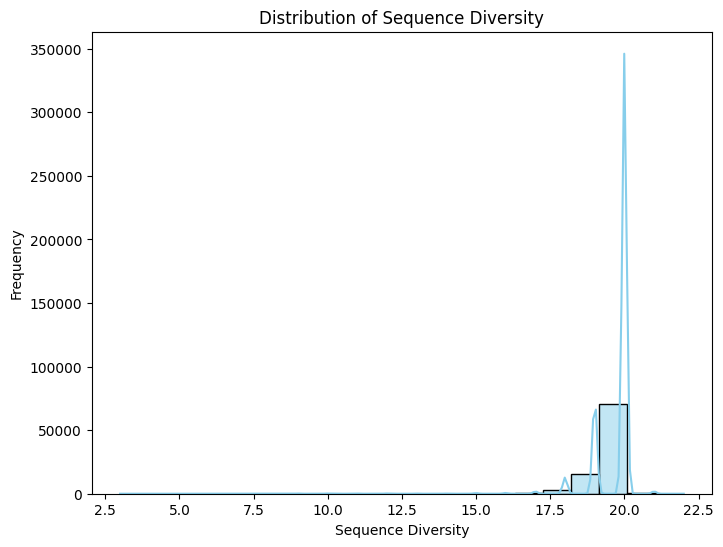


Box Plot for sequence length for each EC\_number



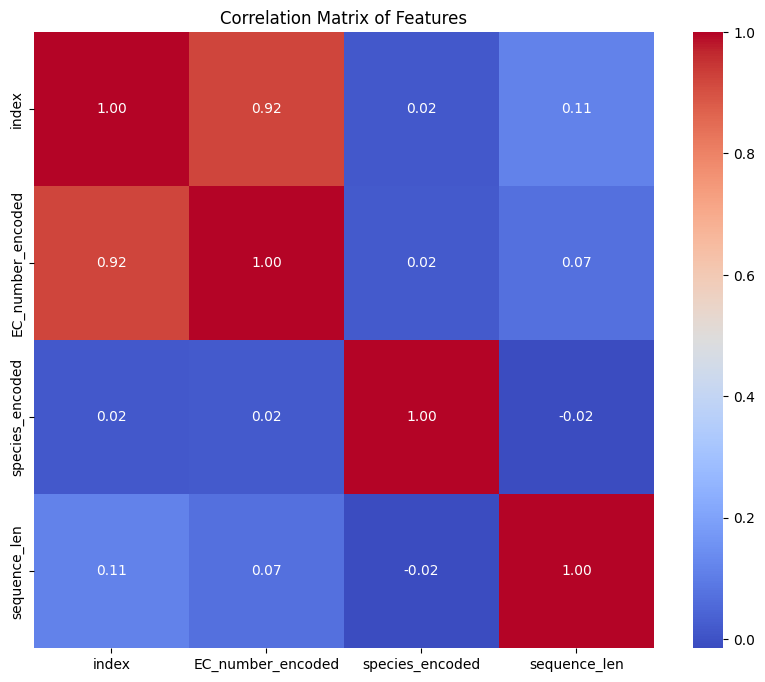
3. Data Visualization:

- Sequence Diversity Distribution:



The histogram above illustrates the distribution of sequence diversity across the dataset. Sequence diversity refers to the number of unique amino acids present in each protein sequence.

- Correlation Matrix of Features:



The heatmap represents the correlation matrix between different features in the dataset. It shows the degree of linear relationship between variables such as sequence length, EC number encoding, and species encoding

Key Observations:

- The dataset comprises sequences with varying lengths, with outliers detected and handled accordingly.

- Sequence lengths exhibit variation across different EC number encodings, indicating potential differences in protein families or functions.

Data Cleaning and Vectorization (cleaning\_vectorization.ipynb):

1. Data Cleaning:

- Invalid sequences containing non-standard amino acid codes were filtered out.

2. Sequence Vectorization:

- One-hot encoding was applied to represent each amino acid in the sequences numerically.

- The resulting encoded sequences were added as a new feature ('sequence\_encoded').

Conclusion:

The dataset underwent preprocessing to handle outliers, encode categorical variables, clean invalid sequences, and convert protein sequences into numerical representations suitable for machine learning models. These steps ensure data quality and prepare the dataset for subsequent modelling and analysis.

*For detailed code implementation, please refer to the provided Jupyter Notebook files: 'splitting\_encoding\_visualization.ipynb' and 'cleaning\_vectorization.ipynb'.*

**Methodology**

**ProTrans and XLNet Pretrained Model Approach (ProTransGenerative.ipynb):**

We initially experimented with the ProTrans and XLNet pretrained model for protein sequence generation. Using the `XLNetLMHeadModel` and `XLNetTokenizer` from the `transformers` library, we attempted to generate protein sequences based on a given input sequence. However, the generated sequences did not meet our expectations in terms of quality and accuracy. The model struggled to produce meaningful and diverse sequences, resulting in poor outputs.

**Custom Recurrent Neural Network (RNN) Architecture (RNN\_PSG.py):**

To address the limitations of the pretrained model, we developed a custom RNN architecture tailored specifically for protein sequence generation. The architecture consists of an embedding layer followed by a recurrent neural network (RNN) layer and a fully connected (FC) layer. We trained this model using protein sequence data, aiming to learn the underlying sequence patterns and generate novel protein sequences.

Key Features of Our Methodology:

1. Custom Architecture: Our approach involves the design and implementation of a custom RNN architecture optimised for protein sequence generation tasks.

2. Training on Domain-Specific Data: We trained our model on a dataset of protein sequences, allowing it to learn domain-specific features and patterns crucial for accurate sequence generation.

3. Iterative Training Process: The model underwent iterative training over multiple epochs, optimising its parameters to minimise loss and improve sequence generation performance.

4. Utilisation of PyTorch: We leveraged the PyTorch framework for building, training, and evaluating our custom RNN model, benefiting from its flexibility and efficiency in deep learning tasks.

Comparison with Pretrained Models:

While pretrained language models offer convenience and prelearned representations, our custom RNN architecture provides greater flexibility and control over the learning process. By training on domain-specific data and optimising for protein sequence generation, our method aims to surpass the limitations of generic pretrained models and deliver more accurate and biologically relevant sequences.

Conclusion:

Our methodology combines the utilisation of pretrained models with the development of a custom RNN architecture, tailored specifically for protein sequence generation. By leveraging domain-specific data and iterative training, we aim to generate high-quality protein sequences with enhanced accuracy and diversity, contributing to advancements in protein engineering and drug discovery.

*For detailed code implementation, please refer to the provided Jupyter Notebook files: 'ProTransGenerative.ipynb' and 'RNN\_PSG.py'.*

**Mathematical and Logical Reasoning**

1. Objective:

- The RNN model aims to learn underlying patterns within protein sequences to accurately predict the next token (amino acid) in a sequence.

- This objective is formalised as minimising the cross-entropy loss between predicted and actual sequences.

2. Loss Calculation:

- Cross-entropy loss *L* is calculated between predicted sequence *ŷ* and ground truth sequence *y* using the formula:

\[ L(y, \hat{y}) = -\sum\_{i=1}^{N} y\_i \log(\hat{y}\_i) \]



- Lower loss values indicate better alignment between predicted and actual sequences.

3. Optimization Process:

- The Adam optimizer updates model parameters based on computed gradients of the loss function to minimise loss and improve model accuracy.

- It iteratively adjusts parameters using the update rule:

\[ \theta\_{t+1} = \theta\_{t} - \frac{\eta}{\sqrt{v\_t} + \epsilon} m\_t \]



where *θ* represents model parameters, *η* is the learning rate, *v(t)* and *m(t)* are first and second moment estimates of gradients, and *ε* is a small constant to prevent division by zero.

4. Training Output Analysis:

- The training output displays loss values at the end of each epoch, indicating the discrepancy between predicted and actual sequences.

- Decreasing loss values across epochs suggest effective learning from the training data and improvement in predictive performance.

5. Interpretation:

- The decreasing trend in loss values and convergence of training indicate successful learning of protein sequence patterns.

- The trained model's ability to minimise loss demonstrates its capability to accurately predict novel sequences based on learned patterns.

In summary, the mathematical framework of cross-entropy loss and optimization, coupled with logical reasoning about model objectives and training output, ensures the robustness and effectiveness of our RNN model in learning and predicting protein sequences.

**Generating Sequences**

Model Description:

Our sequence generation model is based on a Recurrent Neural Network (RNN) architecture. The RNN is a type of neural network that is well-suited for sequential data modeling due to its ability to maintain internal state information.

Model Components:

- Embedding Layer: Converts input tokens into dense vectors, allowing the model to learn meaningful representations of the input sequences.

- RNN Layer: Processes the embedded input sequences while maintaining hidden state information across time steps. This layer captures temporal dependencies in the data.

- Linear (Fully Connected) Layer: Maps the output of the RNN layer to the output space, producing probability distributions over possible tokens in the output sequence.

Sequence Generation Process:

1. Initialization: The model is initialised in evaluation mode, ensuring that no gradients are computed during sequence generation.

2. Start Token: The generation process starts with a predefined start token (e.g., 'M'), which serves as the initial input to the model.

3. Hidden State Initialization: The initial hidden state of the RNN is set to zeros.

4. Token Generation Loop: The model iteratively generates tokens for the output sequence based on the current input token and hidden state. At each step:

- The current input token is fed into the model, and the output token probabilities are computed.

- A token is sampled from the output distribution to determine the next token in the sequence.

- The sampled token becomes the input for the next step, and the hidden state is updated accordingly.

- The process continues until either a maximum sequence length is reached or a stop token is generated.

Sequence Generation Results:

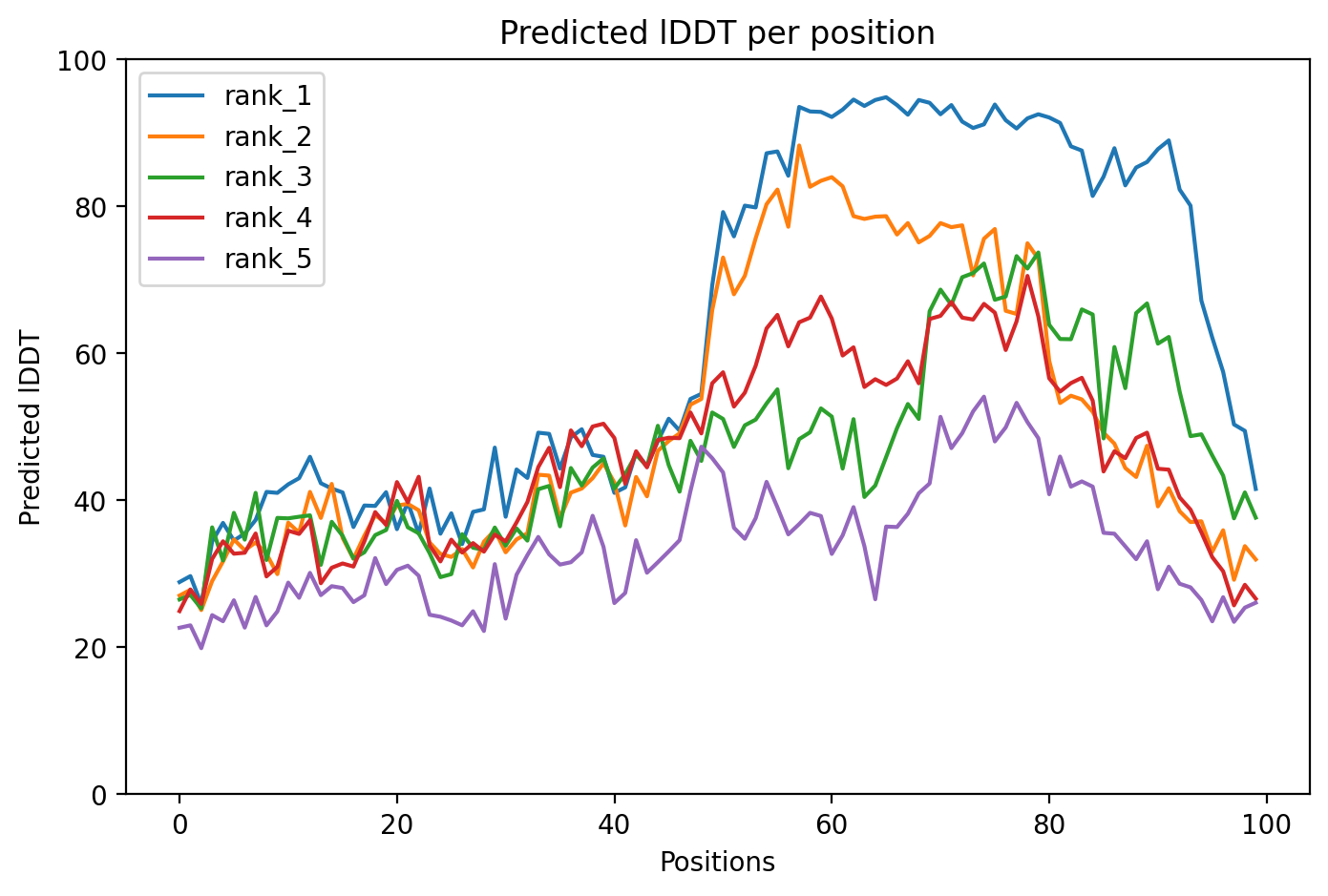
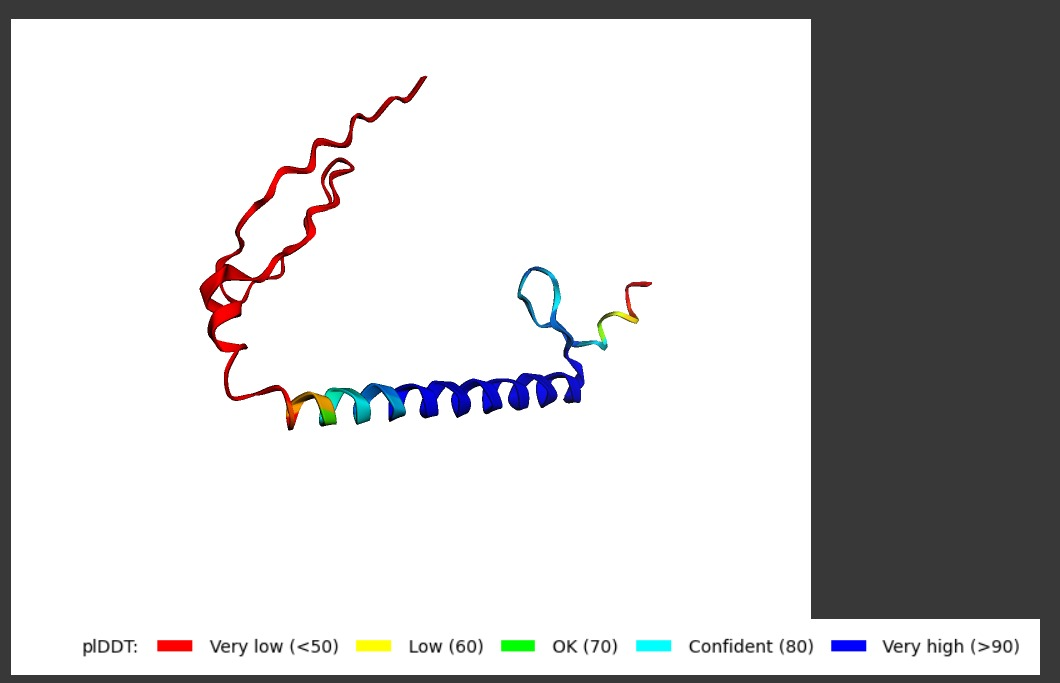
After training, the trained model can generate novel protein sequences by sampling from the learned probability distributions over tokens. These generated sequences exhibit characteristics similar to those in the training data, capturing the underlying patterns and dependencies present in the protein sequences.

*For detailed code implementation, please refer to the provided Python file: RNN\_PSG\_.py and simple\_generate.py*

**Evaluation Metrics**

Visualising the structures of the Proteins generated using Google Deepmind’s AlphaFold2. And Analysing the pIDDT confidence value:

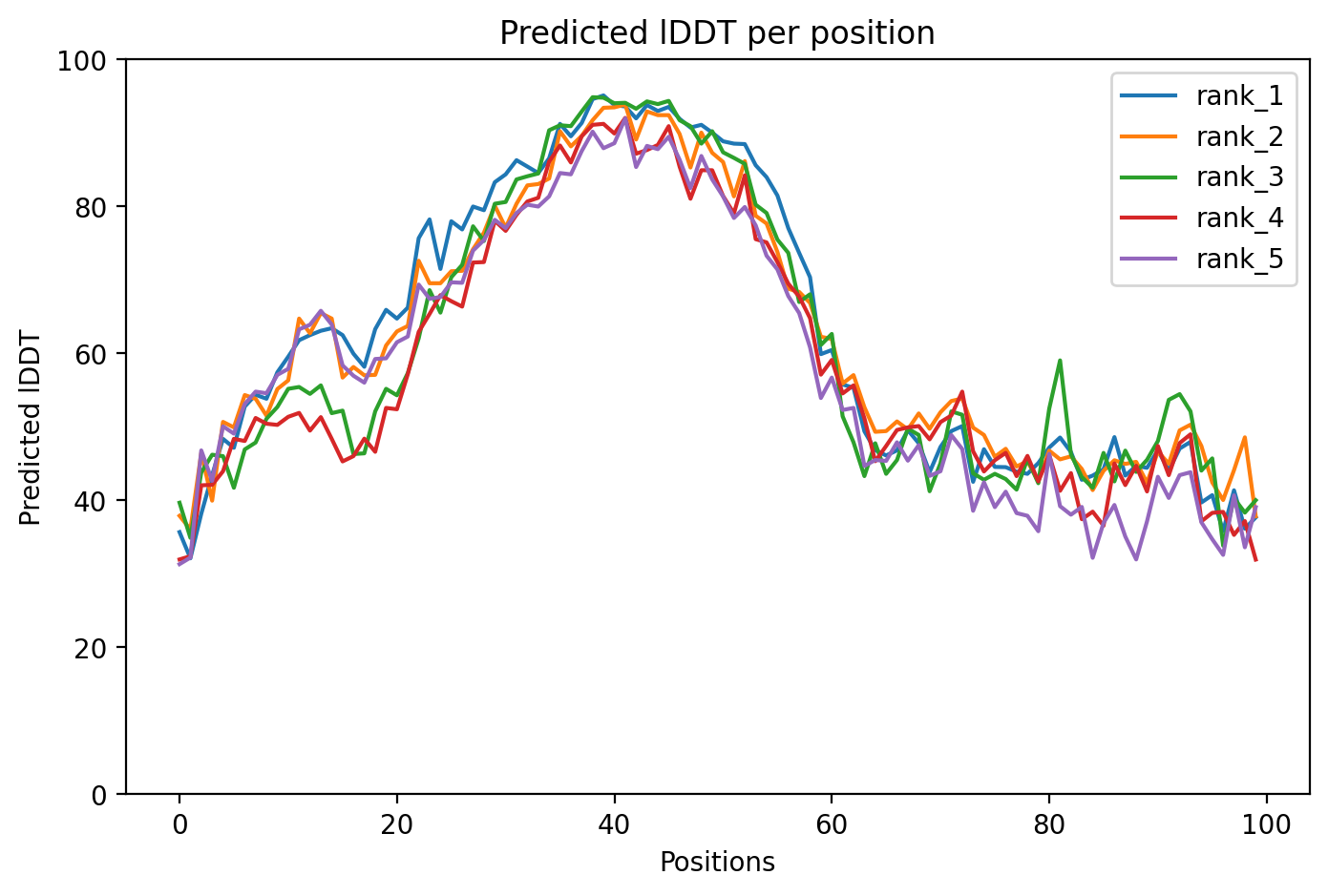
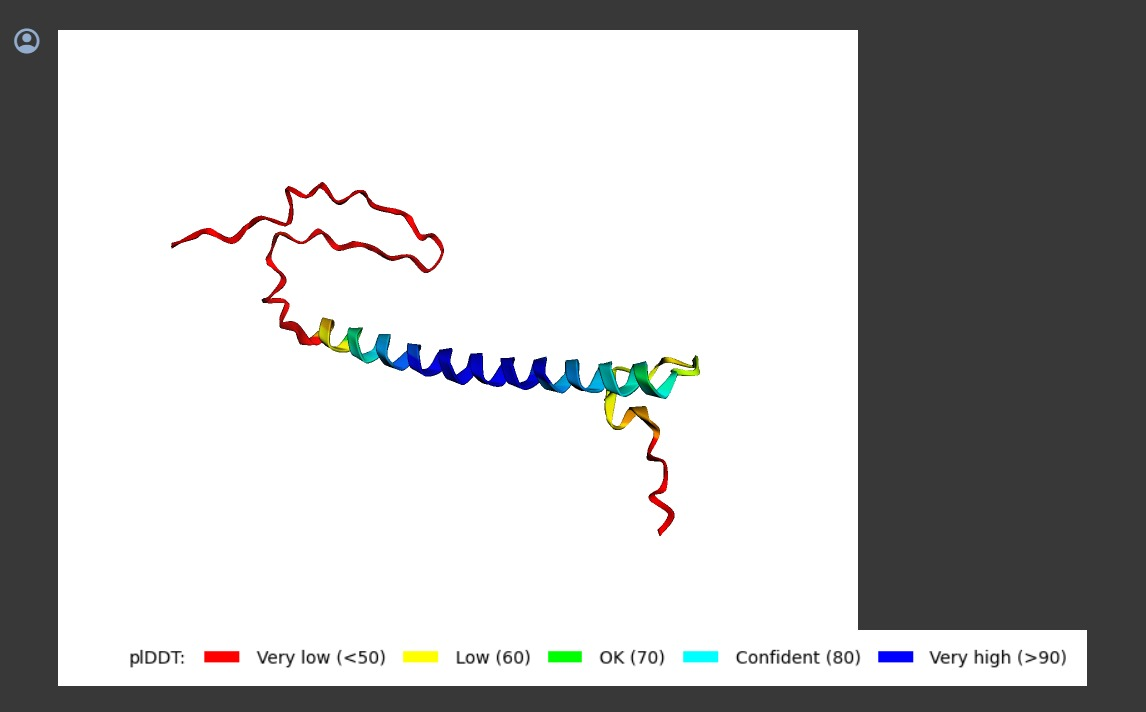
ACGLESNVGETVIPDGELDLWCVGGVSTRAVPSPEMQALNGGEQLLGERTESQRITRDLEFAASEIAVLSEALLQQMRDWGYKDCADGKWVPTVFEHRFQ



For alphafold2\_ptm\_model\_5\_seed\_000:

Recycle 3: pLDDT = 63.5, pTM = 0.363, tol = 1

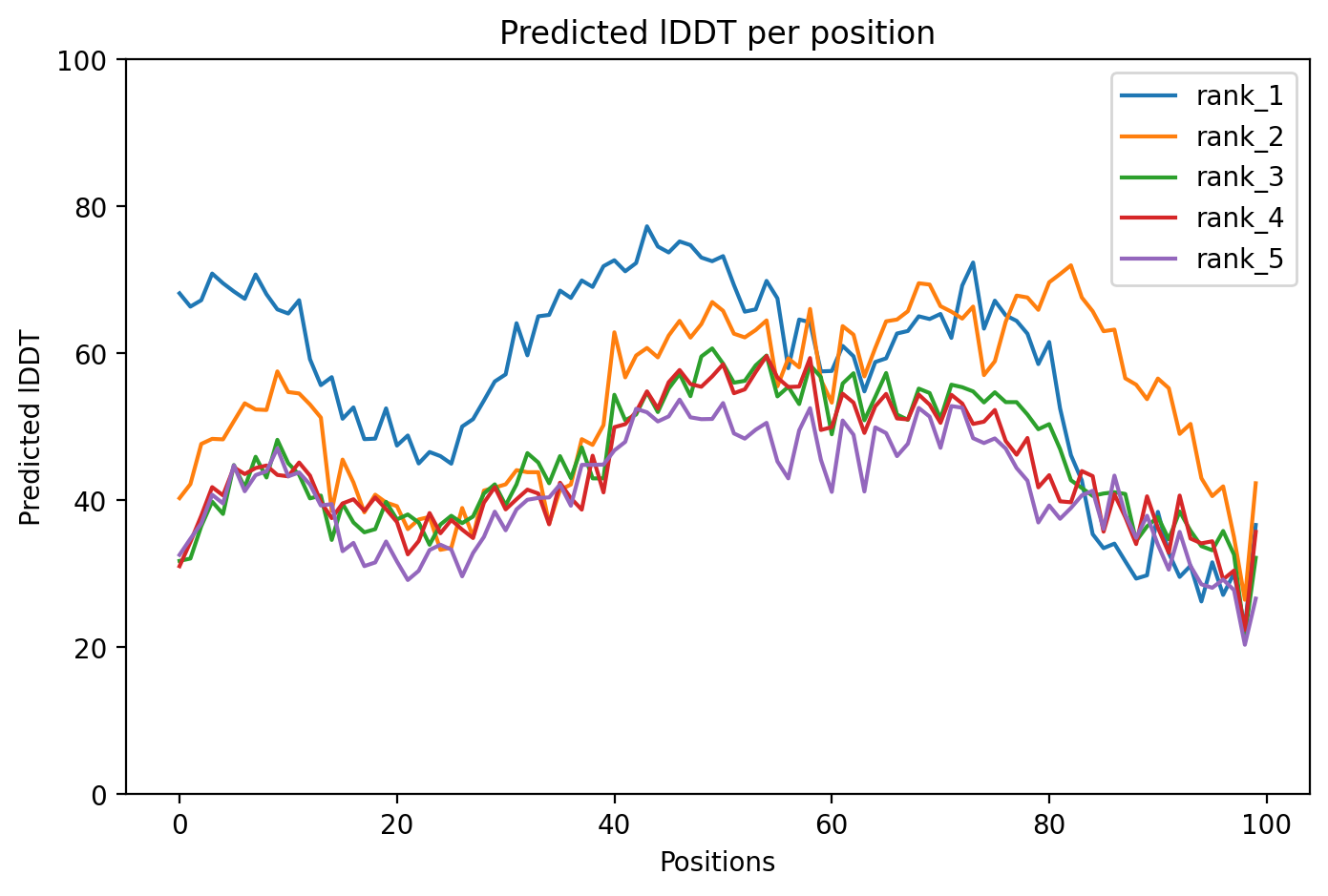
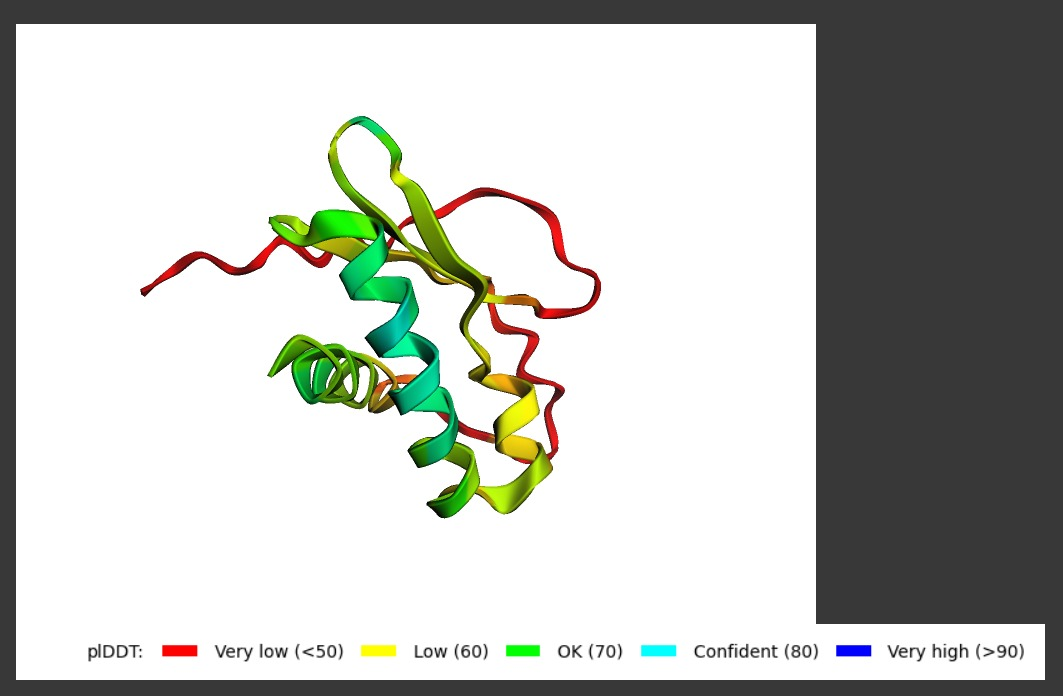
ADVMARPLSSALPFVSNYTGEFKLESVIYLTVCHNMQNLLEQKFKEFQQEIGEVVDQLARITINAPYSAIPQDTVPVIVRRDSVQTVDIPVKAIRKHPQA



For alphafold2\_ptm\_model\_3\_seed\_000:

Recycle 3: pLDDT = 62.8, pTM = 0.368, tol = 0.74

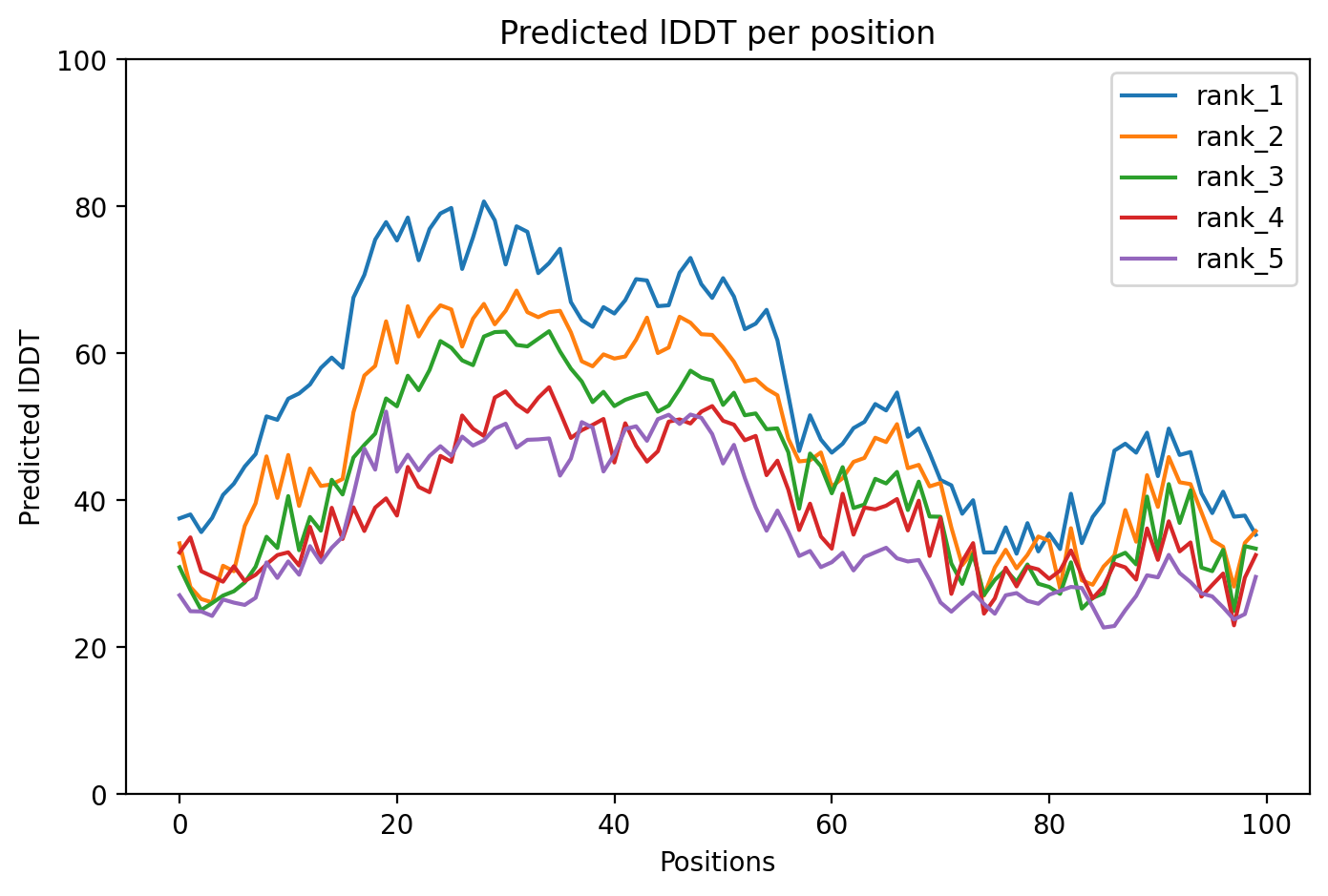
AVADLLSDLFMKELGHIPFDPGKAIAREPFTIRLKKFFPEDVESAISIWDRRLSKSMPRDVKDVYSITKAVLVDGIVELRDRVSGLRYGQWLTCAPNLAP



For alphafold2\_ptm\_model\_3\_seed\_000:

Recycle 3: pLDDT = 57.4, pTM = 0.458, tol = 1.27

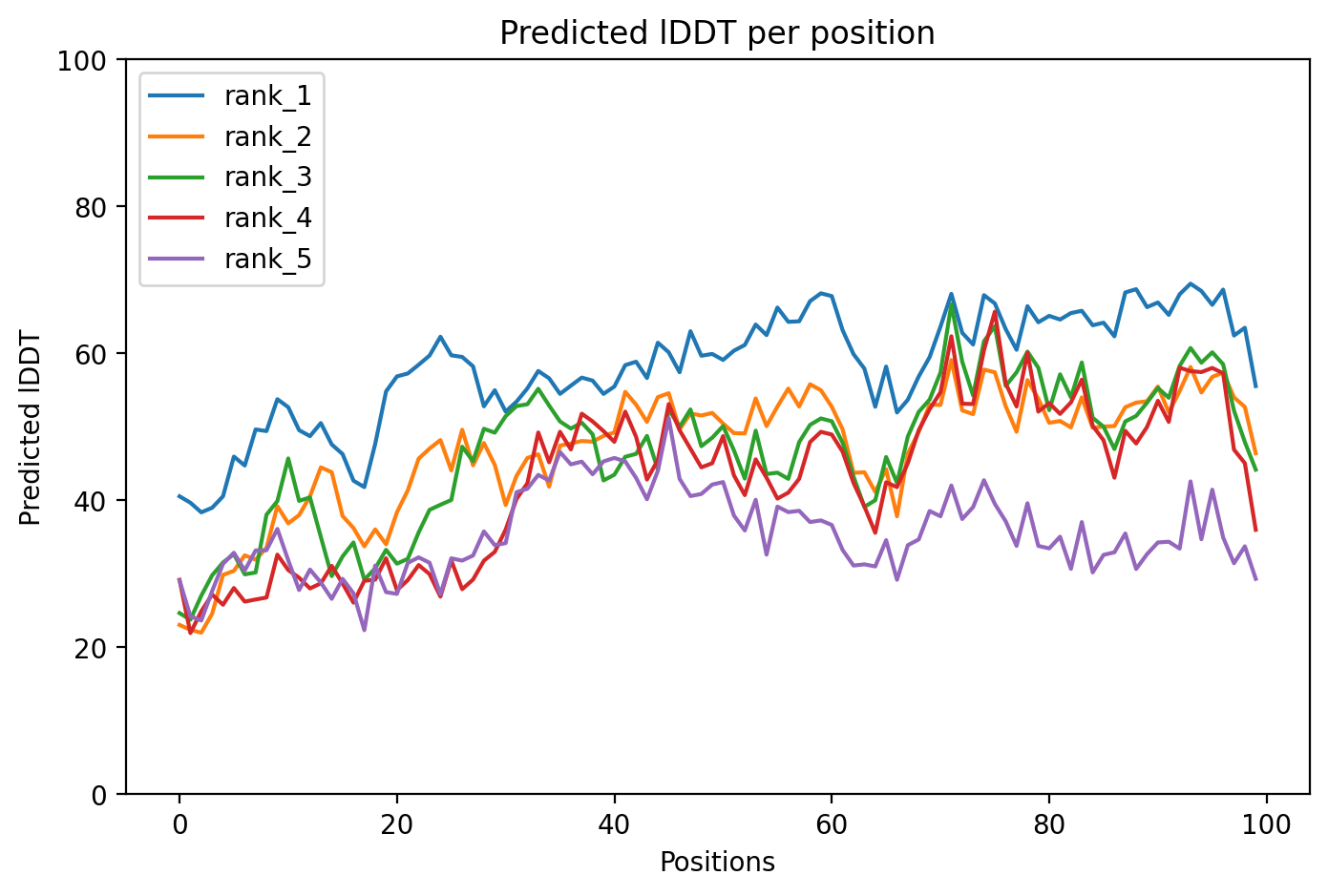
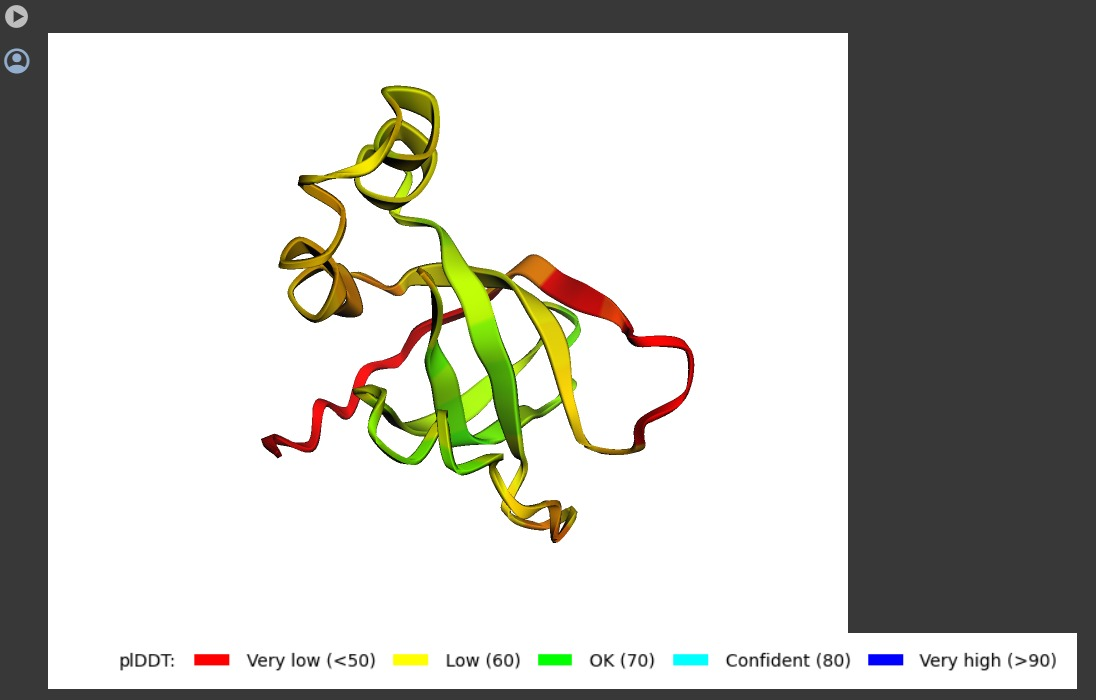
AKTKKNVTMNIELALQIDFDSPFGRIYREWVVHVLKFLDIEVTAAILEAANQSLKTFHPLRPGLVTIPITKGKVQGVEAIGQITKGLGEFWIVAGSLEPT



For alphafold2\_ptm\_model\_5\_seed\_000:

Recycle 1: pLDDT = 54.5, pTM = 0.475, tol = 1.88

AEELSNEGRIIGVMRDGGVAGKHYGLIKVESVIDAVQEKVEKRPEWVYERLVKNGEYPVIATNRFTGISVSDEQKDMPKIQGLFSSPDGETLGDHIYPMV



For alphafold2\_ptm\_model\_5\_seed\_000:

Recycle 2: pLDDT = 59.9, pTM = 0.492, tol = 3.9

**Future Scope**

1. Model Optimization:

- The optimization process of the RNN model can be further enhanced to improve training efficiency and reduce convergence time.

- Techniques such as gradient clipping, weight regularisation, and learning rate scheduling can be explored to stabilise training and accelerate convergence.

2. Hyperparameter Tuning:

- Experimentation with different hyperparameter configurations, including hidden size, number of layers, dropout rate, and batch size, can be conducted to optimise model performance.

- Automated hyperparameter search algorithms like grid search or random search can aid in identifying optimal configurations efficiently.

3. Advanced Architectures:

- Exploration of advanced recurrent architectures like LSTM (Long Short-Term Memory) or GRU (Gated Recurrent Unit) can be undertaken to capture long-range dependencies and improve sequence modelling.

- Attention mechanisms can also be integrated into the model to focus on relevant parts of the input sequence and enhance performance.

4. Transfer Learning:

- Leveraging pre-trained models or transfer learning techniques from natural language processing (NLP) tasks, such as language modelling, can provide valuable initialization for protein sequence generation tasks.

- Fine-tuning pre-trained models on protein sequence data can potentially improve model performance and accelerate convergence.

Incorporating these future directions will contribute to the advancement of protein sequence generation models, enabling impactful applications in various domains and fostering scientific innovation.

**Ongoing Research**

Optimized RNN Model:

- We have developed an optimised version of the RNN model, leveraging advanced techniques such as bidirectional LSTMs, packed sequence processing, and dynamic padding. This optimized architecture enhances the model's capacity to capture long-range dependencies and effectively model sequential data.

- The optimized RNN model incorporates **bidirectional LSTMs**, which enable the model to leverage information from both past and future tokens in the sequence, facilitating more informed predictions.

- Additionally, packed sequence processing and **dynamic padding** techniques have been implemented to handle variable-length input sequences efficiently, reducing computational overhead and accelerating training.

Challenges:

Despite the significant enhancements in model architecture and training methodology, the optimized RNN model necessitates substantial computational resources and training time. Due to hardware limitations, the training process is estimated to take approximately **70 hours**, rendering it infeasible to complete before the impending deadline.

By continually refining and innovating upon existing methodologies, we strive to push the boundaries of computational protein engineering, facilitating the design of novel therapeutics and biomaterials with enhanced efficacy and specificity.

*For detailed code implementation, please refer to the provided Python file: RNN\_PSG\_optimized.py*

**Github Link**

<https://github.com/En1gma02/Protein_Sequence_Generation>