Master 1 Computer science
Parcours Bioinformatique et Modélisation

BIM Project

Exploration of Nitroreductase Protein Sequence Datasets for Generative Modeling

Students: Arthur VINCENS & Sofia TERKI

Advisor : Roberto NETTI

2024-2025

Contents

1 Introduction **2**

2 Datas and alignement **2**
2.1 Data collection 2
2.2 Data Pre-processing 3
2.2.1 Alignement with Hmmer 3
2.2.2 Data encoding 3

3 Exploration of the dataset **3**
3.1 PCA 3
3.2 T-sne 5
3.3 K-means 6

4 Labels **9**
4.1 Reference Article 9
4.2 Computations 9

5 Generative model **10**
5.1 Variational Autoencoder 10
5.1.1 Model 10
5.1.2 Training 12
5.1.3 Performance 12
5.1.4 Sequences Analysis 13
5.2 Classification 15
5.2.1 Logistic Regression 15
5.2.2 Random Forest 16
5.2.3 Classification results 17

6 Conclusion **18**

A Amino Acid Numerical Encoding **20**

B VAE model **20**

1 Introduction

Nitroreductases are a class of enzymes that play a crucial role in the reduction of nitro compounds, which are commonly found in various environmental pollutants and pharmaceutical agents. These enzymes are of significant interest due to their potential applications in bioremediation, drug metabolism, and synthetic biology. Understanding the structural and functional diversity of nitroreductases is essential for harnessing their potential in various biotechnological applications.

The primary objective of this project is to explore the sequence space of nitroreductases using advanced bioinformatics techniques. By leveraging machine learning and generative models, we aim to generate novel nitroreductase sequences that retain the functional characteristics of natural sequences. This approach not only enhances our understanding of these enzymes but also opens up new avenues for designing custom nitroreductases with tailored properties.

The project is structured into several key sections. We begin with data collection and pre-processing, where we gather and align nitroreductase sequences using HMMER, a powerful tool for sequence alignment. The aligned sequences are then encoded into a suitable format for further analysis. Following this, we explore the dataset using dimensionality reduction techniques such as Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE), as well as clustering methods like K-means to identify patterns and groupings within the data.

A significant portion of the project is dedicated to the development and training of a Variational Autoencoder (VAE), a type of generative model capable of learning the underlying distribution of the input data. The VAE is trained to reconstruct and generate new nitroreductase sequences. The performance of the VAE is evaluated based on its ability to accurately reconstruct sequences and generate novel sequences that retain the functional characteristics of natural sequences.

The generated sequences are subjected to rigorous analysis to ensure their quality and similarity to natural sequences. This includes Principal Component Analysis (PCA) to visualize the distribution of natural and generated sequences, Shannon entropy analysis to assess the conservation of positions, and correlation analysis of covariance matrices to evaluate the similarity in covariance structures.

Furthermore, we employ classification techniques such as Logistic Regression and Random Forest to classify the generated sequences and assess their functional properties. The results of these analyses provide insights into the effectiveness of the VAE in generating high-quality nitroreductase sequences.

Overall, this project aims to bridge the gap between computational biology and machine learning, offering a comprehensive approach to understanding and generating nitroreductase sequences. The findings from this study have the potential to advance the field of enzyme engineering and open up new possibilities for the design of custom nitroreductases with tailored properties.

2 Datas and alignement

2.1 Data collection

All *Uniprot* sequences were collected beforehand by our supervisor. A total of $\sim$ **200** GB of data was processed. Then we preprocessed the data in order to retrieve our sequences of interest : the *Nitroreductase* sequences.

2.2 Data Pre-processing

2.2.1 Alignement with Hmmer

How does Hmmer work?

HMMER is used for searching sequence databases for homologs of protein sequences, and for making protein sequence alignments. It implements methods using probabilistic models called profile hidden Markov models (profile HMMs).

HMMER search was performed using the HMM PF00881 Pfam profile to filter relevant sequences and extract only the sequences that align with the profile model representing the Nitroreductase protein family

Data were cleaned by removing inserts, sequences with more than 80% gaps, and duplicates using preestablished scripts.

The final clean FASTA file contains $\mathbf{106,361}$ protein sequences.

2.2.2 Data encoding

1. **Numerical encoding:**

   Each sequence was encoded using a defined numerical encoding: each amino acid was replaced by a defined integer number in the interval (0,20) [Appendix A]

2. **One hot encoding:**

   One-hot encoding is a method to convert categorical data, such as amino acids in protein sequences, into numerical vectors for machine learning applications. Each of the 20 standard amino acids is represented as a binary vector of length 21, where a '1' indicates the presence of a specific amino acid, and all other positions are '0'.

   The sequences were transformed into one-hot encoded vectors to facilitate analysis and put all amino acids at the same distance between each other.

3 Exploration of the dataset

Before doing any further analysis, we wanted to explore the dataset and perform a preliminary analysis visualizing in reduced dimensional space (PCA, t-sne), highliting its clustered / non-clustered structure (k-means) to see if there are some clear patterns emerging from this analysis.

3.1 PCA

PCA is an orthogonal linear transformation that projects high-dimensional data into a lower-dimensional space while preserving maximal variance. The method identifies principal components (PCs), uncorrelated vectors that are linear combinations of the original features, ordered by their explained variance (eigenvalues of the covariance matrix). The first PC captures the largest data variance, and subsequent components account for residual variance in descending order. Mathematically, PCA involves:

1. **Centering the data**:
$$\mathbf{X}_{\text{centered}} = \mathbf{X} - \bar{\mathbf{X}}$$
2. **Computing the covariance matrix**:
$$\mathbf{C} = \frac{1}{n-1}\mathbf{X}_{\text{centered}}^{\top}\mathbf{X}_{\text{centered}}$$
3. **Eigenvalue decomposition**:
$$\mathbf{C} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\top}$$
where $\mathbf{V}$ contains eigenvectors and $\mathbf{\Lambda}$ is a diagonal matrix of eigenvalues.
4. **Projecting data**:
$$\mathbf{Z} = \mathbf{X}_{\text{centered}}\mathbf{V}_k$$
where $\mathbf{V}_k$ contains the top k components.

We applied the sklearn PCA on our dataset. Then we plotted the explained cumulative variance

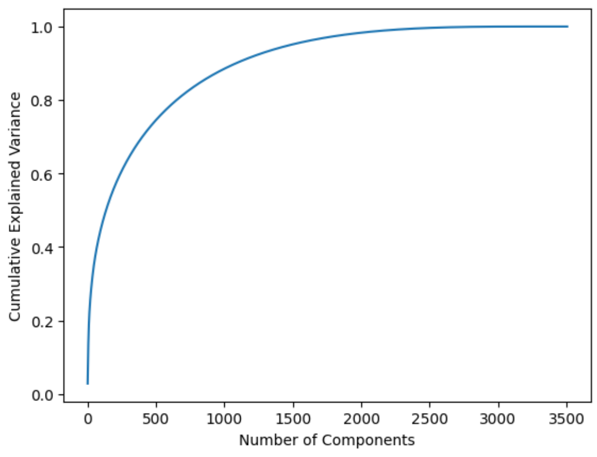


Figure 1: Cumulative Explained Variance vs. Number of Components

We can see that with 1000 components out of 3500 components, 90% of the variability of the data is explained

Then, we generated the heatmap of the first 5 principle components pair plots

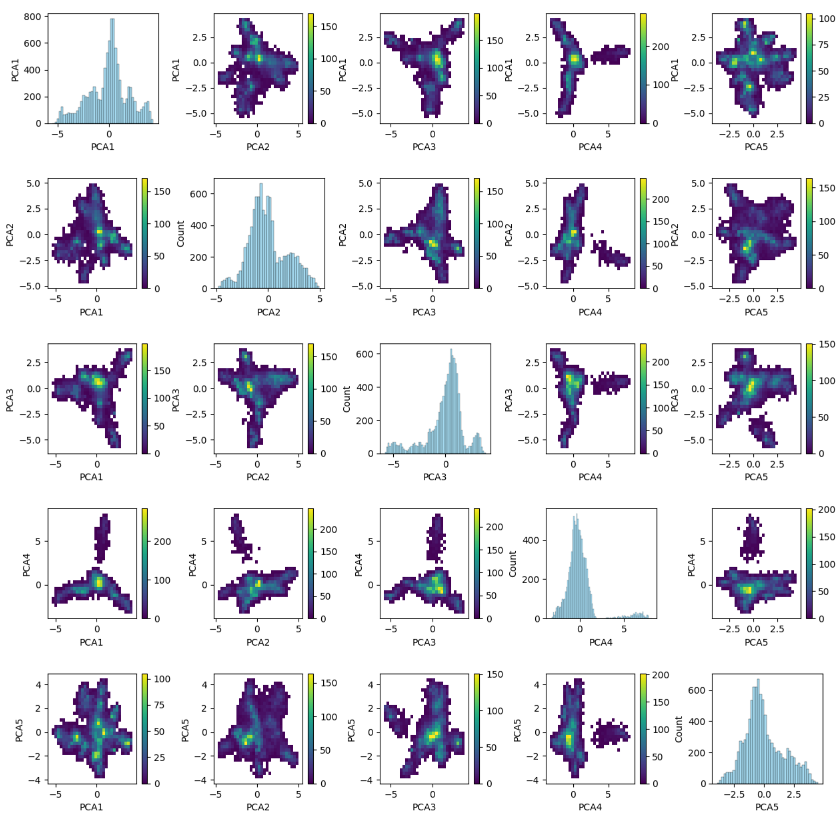


Figure 2: Heatmap of the first 5 principle components pair plots

3.2 T-sne

T-SNE [2] (t-Distributed Stochastic Neighbor Embedding) is a nonlinear dimensionality reduction technique that transforms high-dimensional data into a lower-dimensional space while preserving local neighborhood structures. The algorithm works by first computing probability distributions representing pairwise similarities in both the original high-dimensional space (using Gaussian distributions) and the target low-dimensional space (using heavier-tailed t-distributions). It then iteratively minimizes the *Kullback-Leibler* divergence between these two distributions through gradient descent. This unique approach using t-distributions in the low-dimensional space helps alleviate the "crowding problem" common in linear methods, allowing for effective visualization of complex data structures. The method is particularly sensitive to local patterns rather than global structure, making it especially useful for identifying clusters and local relationships in high-dimensional datasets.

We used the sklearn tsne function in order to visualize if our dataset naturally split into clusters:

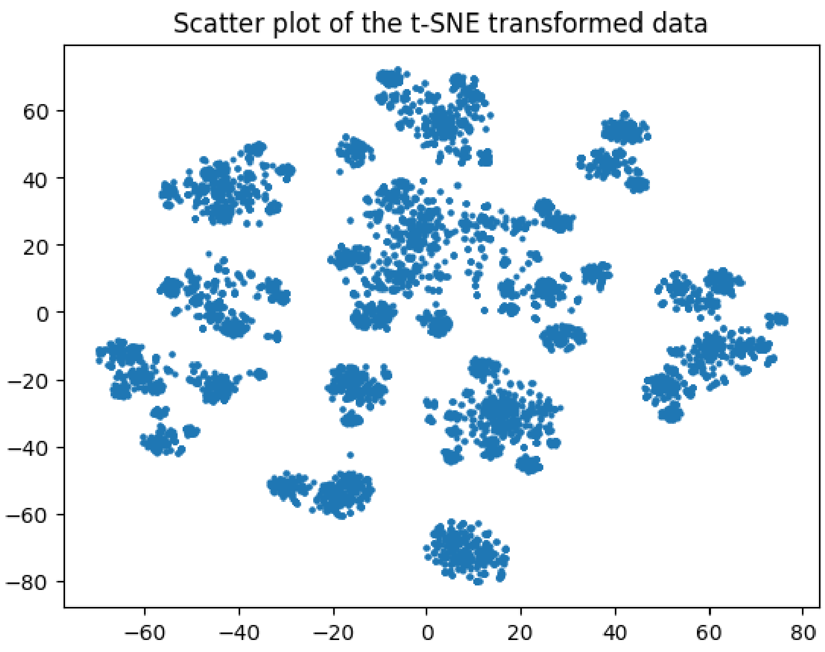


Figure 3: Scatter plot of the t-SNE transformed data

3.3 K-means

The K-means [3] algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares (see below). This algorithm requires the number of clusters to be specified. It scales well to large numbers of samples and has been used across a large range of application areas in many different fields.

The k-means algorithm divides a set of N samples into K disjoint clusters, each described by the mean μ_j of the samples in the cluster. The means are commonly called the cluster "centroids"; note that they are not, in general, points from the original data, although they live in the same space.

The K-means algorithm aims to choose centroids that minimise the inertia, or within-cluster sum-of-squares criterion:

$$\text{Inertia} = \sum_{i=1}^{N} \min_{\mu_j \in C} \|x_i - \mu_j\|^2$$

where:

- N is the number of samples
- K is the number of clusters
- $C = \{\mu_1, \dots, \mu_K\}$ are the cluster centroids
- $\|x_i - \mu_j\|$ is the Euclidean distance between sample x_i and centroid μ_j

We used this algorithm to clusterise our data set. But first, we have tried to identify the optimal number of clusters using two methods:

1. **The elbow method:** The elbow method [4] is a graphical method for finding the optimal K value in a k-means clustering algorithm. The elbow graph shows the within-cluster-sum-of-square values on the y-axis corresponding to the different values of K (on the x-axis). The optimal K value is the point at which the graph forms an elbow.

2. **The silhouet method:** Silhouette analysis can be used to study the separation distance between the resulting clusters. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters and thus provides a way to assess parameters like number of clusters visually. This measure has a range of [-1, 1].

   The silhouette score measures cluster quality by comparing intra-cluster cohesion to inter-cluster separation. For each sample:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

   where:

   - $a(i)$: Average distance to other points in the *same* cluster
   - $b(i)$: Average distance to points in the *nearest neighboring* cluster

   The global score (average of all $s(i)$) ranges from -1 to 1:

   - Values near $+1$ indicate well-separated clusters
   - Values near 0 suggest overlapping clusters
   - Negative values reveal incorrect clustering

For each method (elbow and silhouette), we computed the score for K-means results with 2 to 40 clusters and we obtained these results:

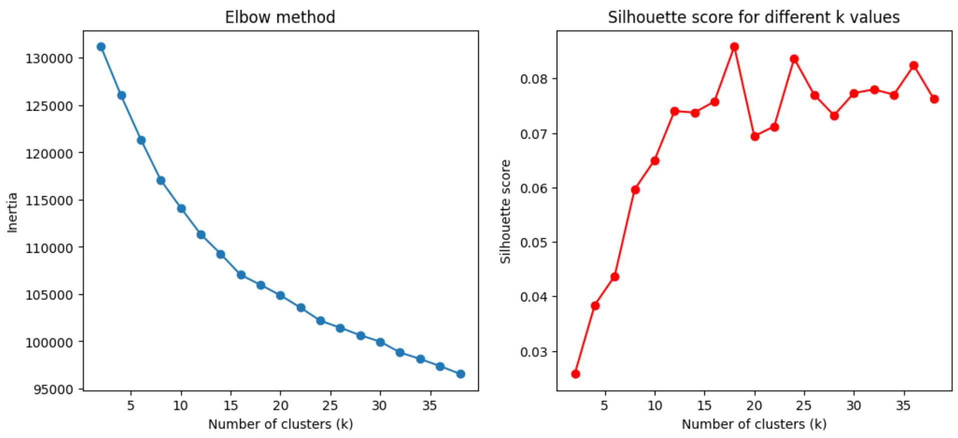


Figure 4: Elbow & Silhouhette curves

As we can see it on the Elbow curve, the elbow is approximately at $\mathbf{k = 15}$. For the Silhouette score, it is maximal between $\mathbf{k = 15}$ and $\mathbf{k = 18}$. We decided to choose $\mathbf{k = 15}$ for the rest of our computations.

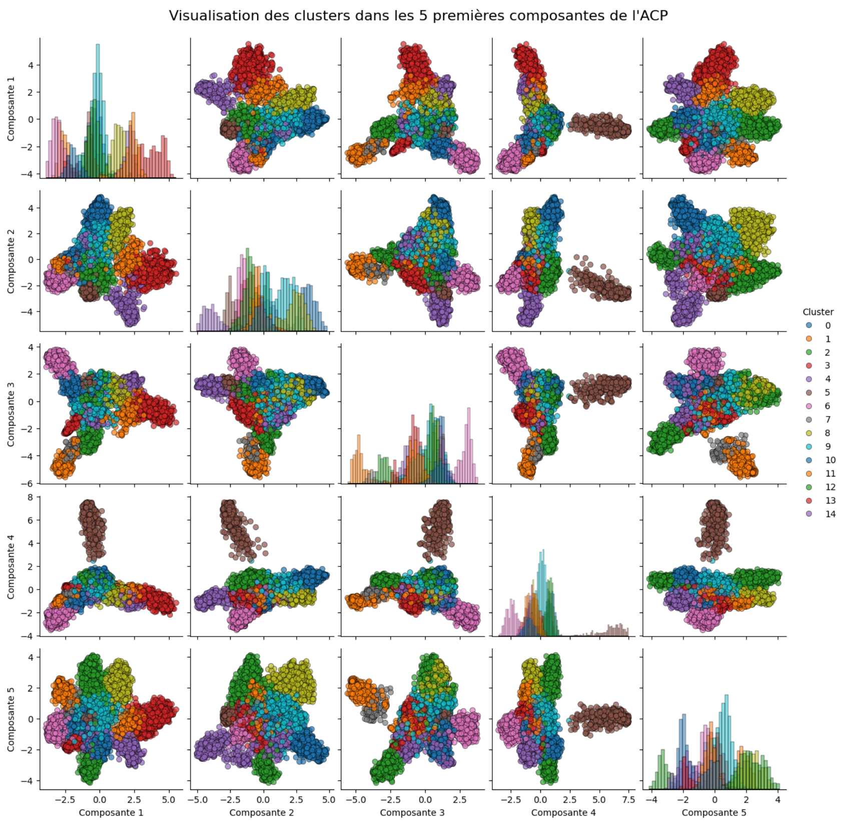


Figure 5: Pair plot of the 5 PC of K-means (k=15) PCA projection

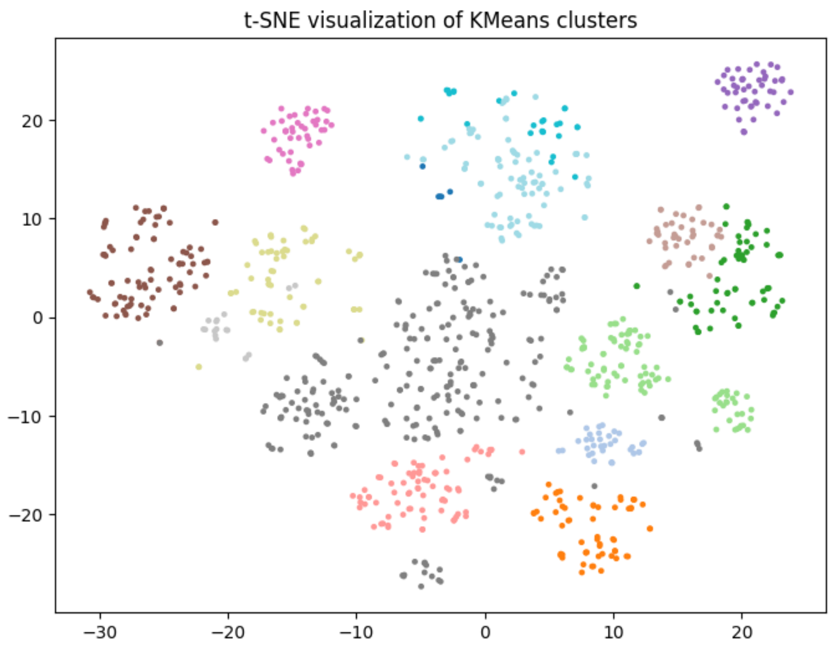


Figure 6: K-means (k=15) t-sne projection

4 Labels

4.1 Reference Article

Our study builds upon the classification framework established in the article [5], where researchers performed large-scale phylogenetic analysis of the nitroreductase (NTR) superfamily (¿24,000 sequences across all domains of life). Their work identified 22 functionally distinct subgroups, including 14 biochemically characterized subgroups (e.g., the NfsA subgroup containing E. coli NfsA homologs).

4.2 Computations

To validate whether our dataset conformed to this established classification, we atomatically retrived the labled sequences, cited in the paper, from Uniprot. We then aligned them using Hmmer with the Nitroreductase Pfam profile and we cleaned them (removed sequences with high percentage of gaps, insertions, duplicates).

We then wanted to "push the clustering to align with the experimental labels we collected from the paper". So we did the "supervised clustering" by initializing the centroids in the labels. The I would show just the image (that here you didn't show) were you have the supervised clusters obtained and the dots/stars corresponding to sequences belonging to that label corresponding to that cluster (same color)

textbfSupervised Cluster Initialization: We implemented K-means with initialized centroids based on reference subgroups:

- We omputed pairwise distance matrices within each labeled subgroup
- For each labeled subgroup, we selected the sequence with the minimal average distance to all other subgroup sequences as centroid

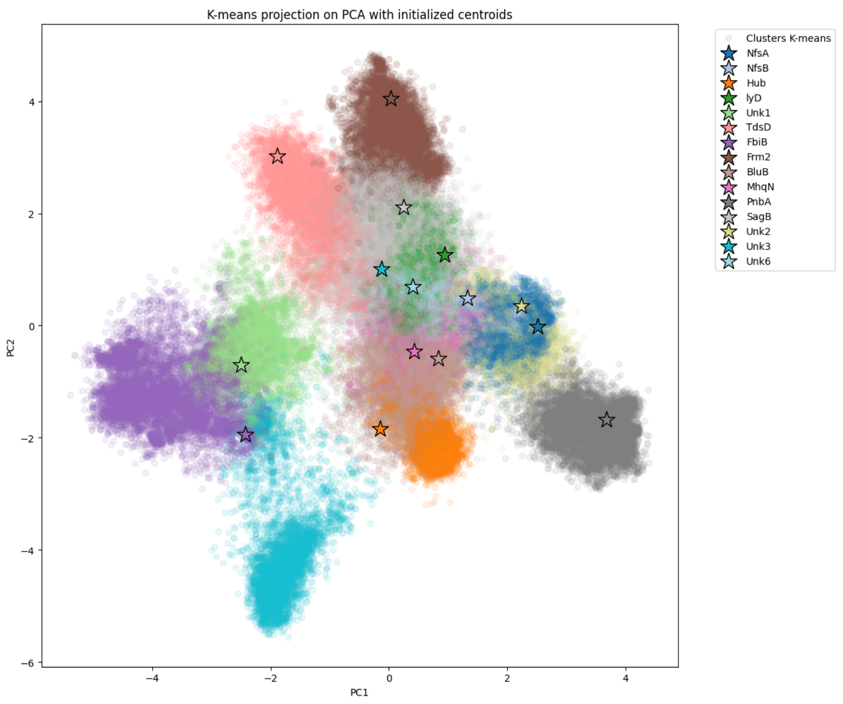


Figure 7: Pair plot of the 5 PC of K-means (k=15) PCA projection

At the end, we wanted to check how many labled sequences fell into the cluster initialized with their corresponding centroid, and we got a precision of **76.19%**.

5 Generative model

5.1 Variational Autoencoder

5.1.1 Model

The Variational Autoencoder (VAE) consists of two main components: the encoder and the decoder. The encoder maps the input data to a latent space, while the decoder reconstructs the input data from the latent space.

Encoder : The encoder is a neural network that takes an input sequence and maps it to a latent space. The architecture of the encoder is as follows:

- **Input Layer**: The input to the encoder is a sequence of shape $(167, 21)$.
- **Fully Connected Layers**: The encoder consists of two fully connected layers with ReLU activation functions. The first layer maps the input to a 256-dimensional space, and the second layer maps it to a 128-dimensional space.
- **Latent Space**: The encoder outputs the mean and log variance of the latent space, which are used to sample the latent vector z.

Decoder : The decoder is a neural network that reconstructs the input data from the latent space. The architecture of the decoder is as follows:

- **Input Layer**: The input to the decoder is a latent vector z of dimension $latent\_dim$.
- **Fully Connected Layers**: The decoder consists of two fully connected layers with ReLU activation functions. The first layer maps the latent vector to a 128-dimensional space, and the second layer maps it to a 256-dimensional space.
- **Output Layer**: The final layer maps the 256-dimensional space to the original input shape $(167, 21)$ using a softmax activation function to ensure the output is a valid probability distribution.

The VAE combines the encoder and decoder into a single model. The encoder maps the input data to a latent space, and the decoder reconstructs the input data from the latent space. The VAE is trained to minimize the reconstruction loss and the Kullback-Leibler (KL) divergence between the learned latent distribution and a prior distribution (usually a standard normal distribution).

Hyperparameters :

- **Latent Dimension**: The dimension of the latent space, set to 10.
- **Batch Size**: The number of samples processed in one forward/backward pass, set to 256.
- **Epochs**: The number of times the entire dataset is passed through the network, set to 400.

- **Learning Rate**: The step size at which the model parameters are updated, set to $1e-3$.

For a detailed visualization of our VAE architecture , please refer to [Appendix B]

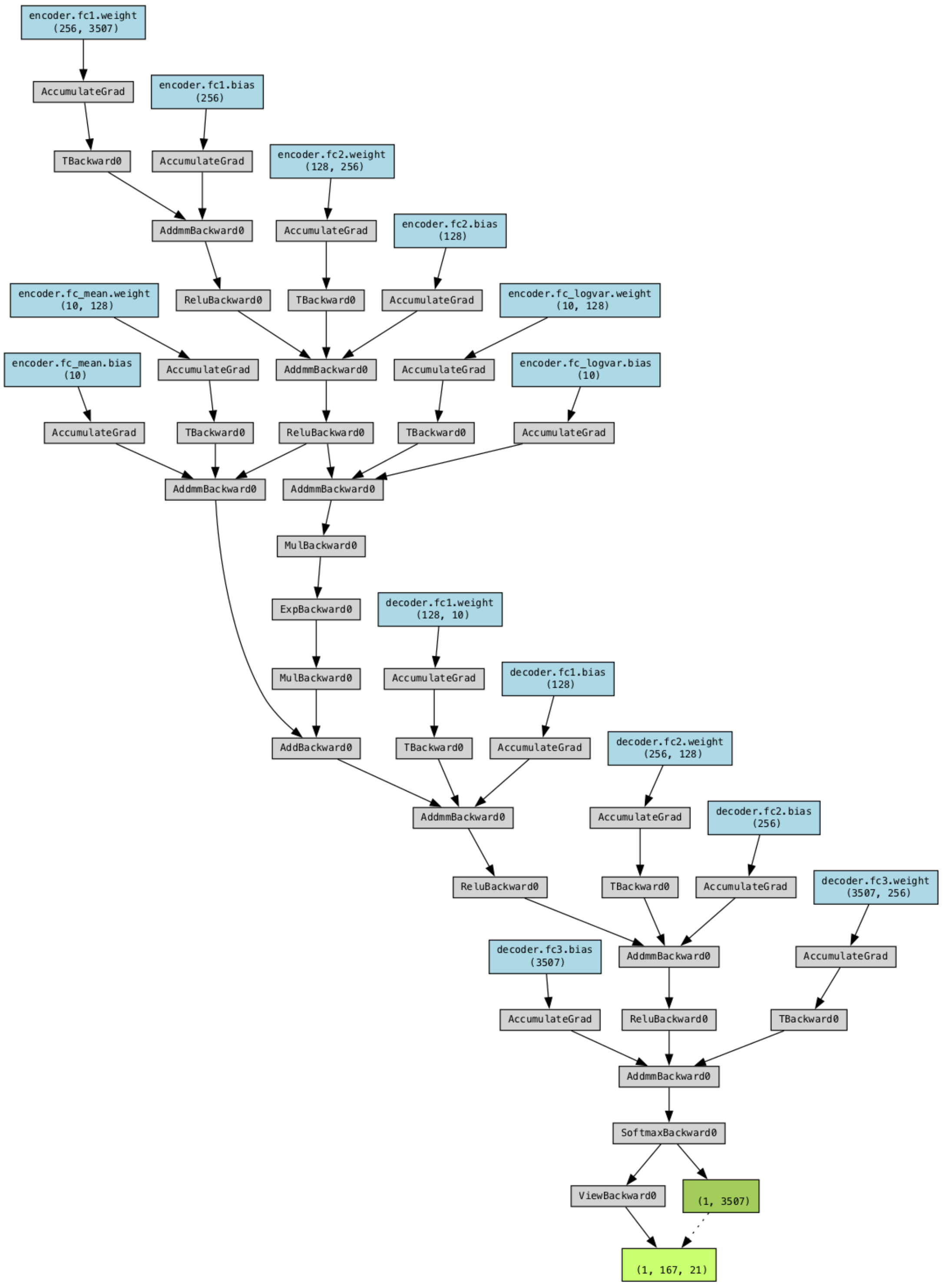


Figure 8: Architecture of the Variational Autoencoder

5.1.2 Training

The dataset consists of 100,000 sequences encoded in one-hot format with a shape of $(167, 21)$. The dataset is split into training and testing sets using a 75-25 split. The training set is used to train the model, while the testing set is used to evaluate the model's performance.

The VAE is trained using the Adam optimizer, which is an adaptive learning rate optimization algorithm. The Adam optimizer combines the advantages of two other extensions of stochastic gradient descent: AdaGrad and RMSProp. The learning rate is set to $1e-3$.

The training loop involves the following steps:

- **Forward Pass**: The input data is passed through the VAE to obtain the reconstructed data, the mean, and the log variance of the latent space.
- **Loss Calculation**: The loss is calculated using a combination of the reconstruction loss and the Kullback-Leibler (KL) divergence. The reconstruction loss measures the difference between the input data and the reconstructed data, while the KL divergence measures the difference between the learned latent distribution and a prior distribution (usually a standard normal distribution).
- **Backward Pass**: The gradients of the loss with respect to the model parameters are computed and used to update the model parameters using the Adam optimizer.

The training loop is implemented as follows:

- The model is trained for a maximum of 400 epochs.
- The training loss and test loss are calculated for each epoch.
- The model's performance is evaluated on the test set every 10 epochs.
- Early stopping is used to prevent overfitting. If the test loss does not improve for 10 consecutive epochs, the training is stopped early.

Early stopping is a technique used to prevent overfitting by stopping the training process if the model's performance on the test set does not improve for a certain number of epochs. In this case, early stopping is triggered if the test loss does not improve for 10 consecutive epochs.

The training was stopped at epoch 151 with a training loss of 1034.74 and a test loss of 1047.34 due to the early stopping criterion being met.

5.1.3 Performance

The performance of the VAE is evaluated based on its ability to reconstruct sequences and generate new sequences. The evaluation metrics include the percentage of identity and the similarity score using the BLOSUM62 matrix.

Test Dataset :

- Average Identity : 69.56%
- Average Similarity Score : 533.39

Training Dataset :

- Average Identity : 70.83%
- Average Similarity Score : 545.47

Now that the model is able to reconstruct a one-hot amino acid sequence from the latent space, 10 000 sequences were generated by sampling from the latent space and decoding the latent vectors using the decoder part of the VAE. The generated sequences were saved to a FASTA file.

5.1.4 Sequences Analysis

The analysis of the generated sequences involves several key steps to evaluate their quality and similarity to natural sequences. These steps include Principal Component Analysis (PCA), Shannon entropy analysis, and correlation analysis of covariance matrices.

Principal Component Analysis (PCA): PCA was performed to reduce the dimensionality of the one-hot encoded sequences and visualize the distribution of natural and generated sequences. The PCA results show that the natural and generated sequences overlap very well, indicating that the generated sequences have retained the functional characteristics of the natural sequences.

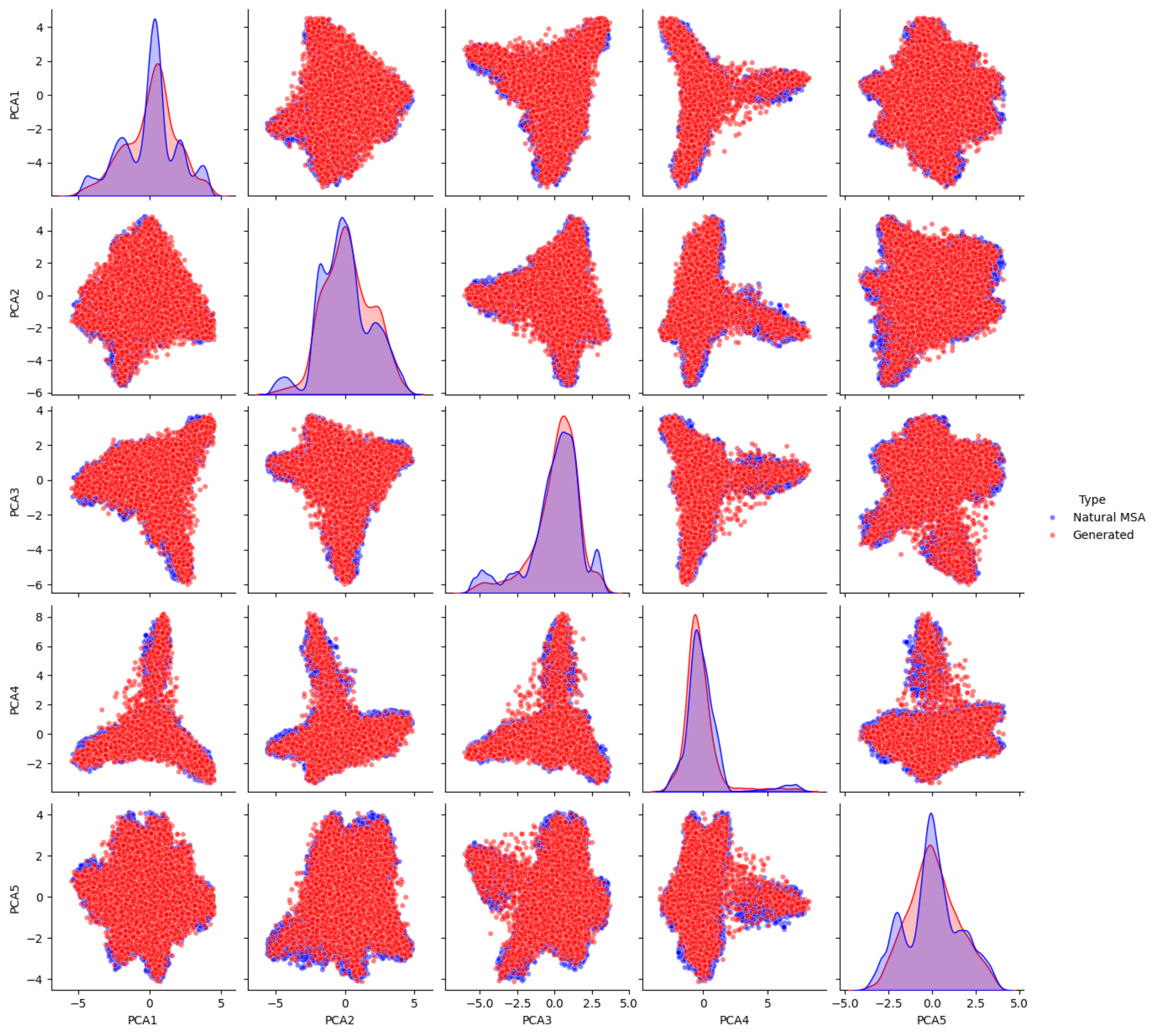


Figure 9: PCA of Natural vs Generated Sequences

Shannon Entropy Analysis: Shannon entropy was calculated for each position in the multiple sequence alignments of natural and generated sequences. The entropy plots for natural and generated sequences align perfectly, showing that the generated sequences have captured the conserved positions well.

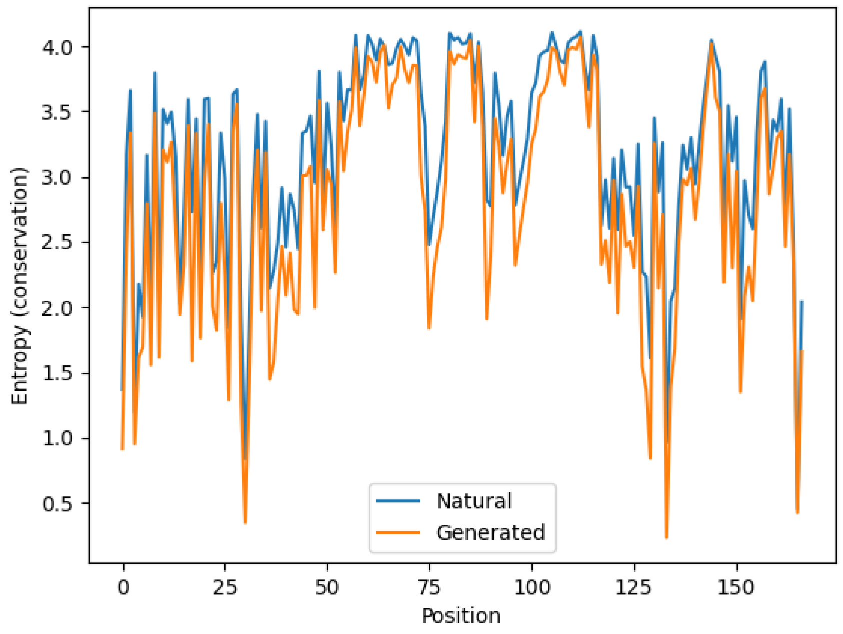


Figure 10: Shannon Entropy of Natural vs Generated Sequences

Correlation Analysis: The covariance matrices of the PCA results for natural and generated sequences were compared using Pearson correlation.

- Pearson correlation : 0.9933497993556529
- p-value : 4.23×10^{-23}

This high correlation indicates a strong similarity between the covariance structures of natural and generated sequences.

Overall, the analysis demonstrates that the generated sequences closely resemble the natural sequences in terms of their functional characteristics, conserved positions, and covariance structures. This indicates that the VAE has successfully learned the underlying patterns and relationships in the natural sequences and can generate high-quality sequences that retain these properties.

5.2 Classification

After generating artificial sequences using the VAE, the goal was to see if a simple classifier could easily distinguish between the natural sequences and the generated one.

Two classifiers were trained:

5.2.1 Logistic Regression

Logistic regression [6] is a supervised machine learning algorithm used for classification tasks in which the objective is to predict the probability that an instance belongs to a given class or not. Logistic regression is a statistical algorithm that analyzes the relationship between two data factors.

Logistic regression is used for binary classification, where we use sigmoid function, which takes input as independent variables and produces a probability value between 0 and 1.

The logistic regression model computes the probability $P(y = 1|\mathbf{x})$ as:

$$P(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x}+b)}}$$

where:

- $\mathbf{x}$ is the feature vector
- $\mathbf{w}$ is the weight vector (model parameters)
- b is the bias term
- σ is the sigmoid function

The model is trained by maximizing the likelihood function:

$$\mathcal{L}(\mathbf{w}, b) = \prod_{i=1}^{n} P(y_i|\mathbf{x}_i)^{y_i}(1 - P(y_i|\mathbf{x}_i))^{1-y_i}$$

Parameters are optimized using gradient descent or other optimization algorithms.

5.2.2 Random Forest

Random Forests is an ensemble learning method that operates by constructing multiple decision trees during training. For a classification problem, multiple trees are built from random sub-samples of the data set (with replacement), and each tree votes for a class. The final prediction is obtained by majority vote or by averaging. This algorithm combines the concepts of *bootstrap aggregating* (bagging) and *random feature selection* to create a robust and accurate model that resists overfitting.

The Random Forest algorithm works through the following key steps:

1. **Bootstrap Sampling**: Construct B decision trees using B bootstrap samples from the training data of size n. Each sample is drawn with replacement.

2. **Random Feature Selection**: For each node in every tree, select a random subset of m features from the total p features ($m \leq p$, typically $m \approx \sqrt{p}$ for classification). The best split is chosen from among these m features.

3. **Tree Construction**: Grow each tree to maximum depth without pruning.

4. **Aggregation**: Combine predictions from all trees:

$$\hat{y}_{\text{RF}} = \Big\{ \text{mode}(\{\hat{y}_b\}_{b=1}^{B})$$

5.2.3 Classification results

	Logistic Regression	**Random Forests**
train	99.41%	100%
test	98.55%	93.69%

Table 1: Accuracy in train and test of 2 classification algorithms

We can see on the Table 1, accuracies in train and test of both our classifiers (Logistic regression & Random forest) are really high. This means that the classifiers are easily able to distinguish between the natural sequences and the generated ones.

6 Conclusion

This project has successfully demonstrated the application of advanced bioinformatics techniques and machine learning models to explore and generate nitroreductase sequences. Through a comprehensive approach involving data collection, pre-processing, dimensionality reduction, clustering, and generative modeling, we have gained valuable insights into the structural and functional diversity of nitroreductases.

The initial exploration of the dataset using PCA and t-SNE provided a clear visualization of the sequence space, highlighting the underlying patterns and groupings within the data. The use of K-means clustering further elucidated the clusterized structure of the dataset, setting the stage for more detailed analysis.

The supervised clustering approach, where centroids were initialized based on reference subgroups, provided an initial validation of our clustering methodology. By selecting sequences with minimal average distances as centroids, we achieved a precision of 76.19% in clustering labeled sequences. This result, while promising, indicates that there is still room for improvement in aligning generated sequences with established classifications, suggesting the need for further refinement of our clustering techniques.

The development and training of a Variational Autoencoder (VAE) marked a significant milestone in this project. The VAE's ability to reconstruct and generate new nitroreductase sequences with high accuracy and similarity to natural sequences underscores its potential as a powerful tool for sequence generation. The performance metrics, including the high percentage of identity and similarity scores, as well as the strong correlation in covariance structures, attest to the VAE's effectiveness in capturing the essential characteristics of natural sequences.

The classification results using Logistic Regression and Random Forest models highlighted the distinguishable differences between natural and generated sequences. With accuracies of 99.41% and 100% on the training set, and 98.55% and 93.69% on the test set, respectively, these models demonstrated a remarkable ability to differentiate between the two types of sequences. This high level of accuracy, while indicating the effectiveness of the classifiers, also suggests that the generated sequences may lack certain characteristics present in natural sequences. This finding underscores the need for further analysis and refinement of the generative model to better capture the functional properties of natural nitroreductases.

Overall, this project has not only advanced our understanding of nitroreductases but also showcased the potential of machine learning in enzyme engineering. The integration of computational biology and machine learning techniques has opened up new avenues for designing custom nitroreductases with tailored properties. The findings from this study have significant implications for bioremediation, drug metabolism, and synthetic biology, paving the way for future research and applications in these fields.

In conclusion, the successful implementation of this project underscores the importance of interdisciplinary approaches in addressing complex biological questions. By leveraging the strengths of both computational and experimental methods, we have made substantial progress in the study of nitroreductases, with promising prospects for future advancements.

References

1. GeeksforGeeks. *Principal Component Analysis (PCA).* [Online]. Available: https://www.geeksfor component-analysis-pca/

2. MathWorks. *t-SNE Documentation.* [Online]. Available: https://www.mathworks.com/help/stat sne.html

3. IBM. (Year). *K-means Clustering.* [Online]. Available: https://www.ibm.com/fr-fr/think/topics/k-means-clustering

4. GeeksforGeeks. *Elbow Method for Optimal Value of K in KMeans.* [Online]. Available: https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/

5. Akiva, E., Copp, J. N., Tokuriki, N., & Babbitt, P. C. (2017). *Evolutionary and molecular foundations of multiple contemporary functions of the nitroreductase superfamily.* Proceedings of the National Academy of Sciences, 114(45), E9569-E9578. https://www.pnas.org/doi/full/10.1073/pnas.1706849114

6. Jurafsky, D. & Martin, J. H. (Year). *Speech and Language Processing* (3rd ed.). [Online]. Available: https://web.stanford.edu/ jurafsky/slp3/5.pdf

7. IBM. *Random Forest.* [Online]. Available: https://www.ibm.com/fr-fr/think/topics/random-forest

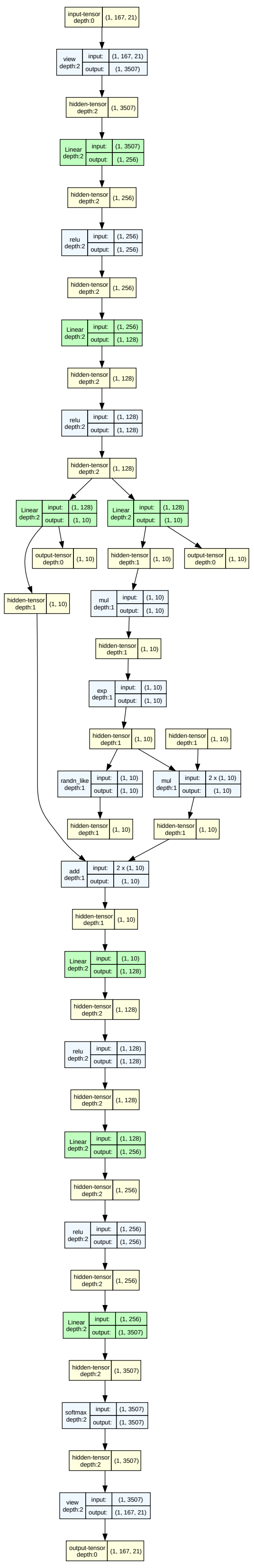
Appendix

A Amino Acid Numerical Encoding

Amino Acid	**Value**	**Amino Acid**	**Value**
- (gap)	0	M	11
A	1	N	12
C	2	P	13
D	3	Q	14
E	4	R	15
F	5	S	16
G	6	T	17
H	7	V	18
I	8	W	19
K	9	Y	20
L	10		

Table 2: Numerical encoding scheme for amino acids and gaps

B VAE model


input-tensor depth:0 (1, 167, 21)
view depth:2 input: (1, 167, 21) output: (1, 3507)
hidden-tensor depth:2 (1, 3507)
Linear depth:2 input: (1, 3507) output: (1, 256)
hidden-tensor depth:2 (1, 256)
relu depth:2 input: (1, 256) output: (1, 256)
hidden-tensor depth:2 (1, 256)
Linear depth:2 input: (1, 256) output: (1, 128)
hidden-tensor depth:2 (1, 128)
relu depth:2 input: (1, 128) output: (1, 128)
hidden-tensor depth:2 (1, 128)
Linear depth:2 input: (1, 128) output: (1, 10)
Linear depth:2 input: (1, 128) output: (1, 10)
output-tensor depth:0 (1, 10)
hidden-tensor depth:1 (1, 10)
output-tensor depth:0 (1, 10)
hidden-tensor depth:1 (1, 10)
mul depth:1 input: (1, 10) output: (1, 10)
hidden-tensor depth:1 (1, 10)
exp depth:1 input: (1, 10) output: (1, 10)
hidden-tensor depth:1 (1, 10)
hidden-tensor depth:1 (1, 10)
randn_like depth:1 input: (1, 10) output: (1, 10)
mul depth:1 input: 2 x (1, 10) output: (1, 10)
hidden-tensor depth:1 (1, 10)
hidden-tensor depth:1 (1, 10)
add depth:1 input: 2 x (1, 10) output: (1, 10)
hidden-tensor depth:1 (1, 10)
Linear depth:2 input: (1, 10) output: (1, 128)
hidden-tensor depth:2 (1, 128)
relu depth:2 input: (1, 128) output: (1, 128)
hidden-tensor depth:2 (1, 128)
Linear depth:2 input: (1, 128) output: (1, 256)
hidden-tensor depth:2 (1, 256)
relu depth:2 input: (1, 256) output: (1, 256)
hidden-tensor depth:2 (1, 256)
Linear depth:2 input: (1, 256) output: (1, 3507)
hidden-tensor depth:2 (1, 3507)
softmax depth:2 input: (1, 3507) output: (1, 3507)
hidden-tensor depth:2 (1, 3507)
view depth:2 input: (1, 3507) output: (1, 167, 21)
output-tensor depth:0 (1, 167, 21)