Adaptive FEC for Cloud Gaming

Abdulmueez Eniola, Ozioma Okonicha, Pavel Tishkin

Introduction

Cloud gaming is an emerging technology that allows users to play high-quality video games without the need for powerful gaming hardware.

Introduction

Cloud gaming is a rapidly growing industry, with more and more people turning to streaming services to play games.

Problem Statement

However, one of the biggest challenges is ensuring a smooth and seamless experience for gamers, regardless of their internet connection quality.

Problem Statement

More specifically, one of which is the issue of network latency and packet loss. These issues can lead to a poor gaming experience, including visual artifacts, input lag, and dropped frames.

Problem Statement

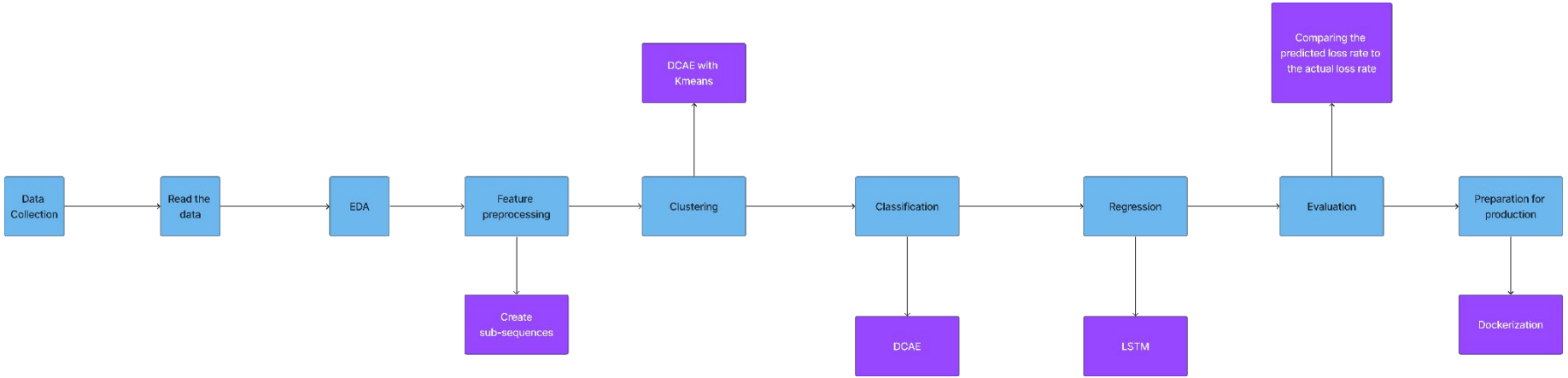
One solution to these problems is the use of Forward Error Correction (FEC), which adds redundant data to the packets to allow for error correction.

Problem Statement

Forward Error Correction (FEC) is a technique commonly used in data transmission to ensure data integrity by adding redundant information to the data packets. This technique can also be applied to cloud gaming to reduce the negative impact of packet loss on the player's experience.

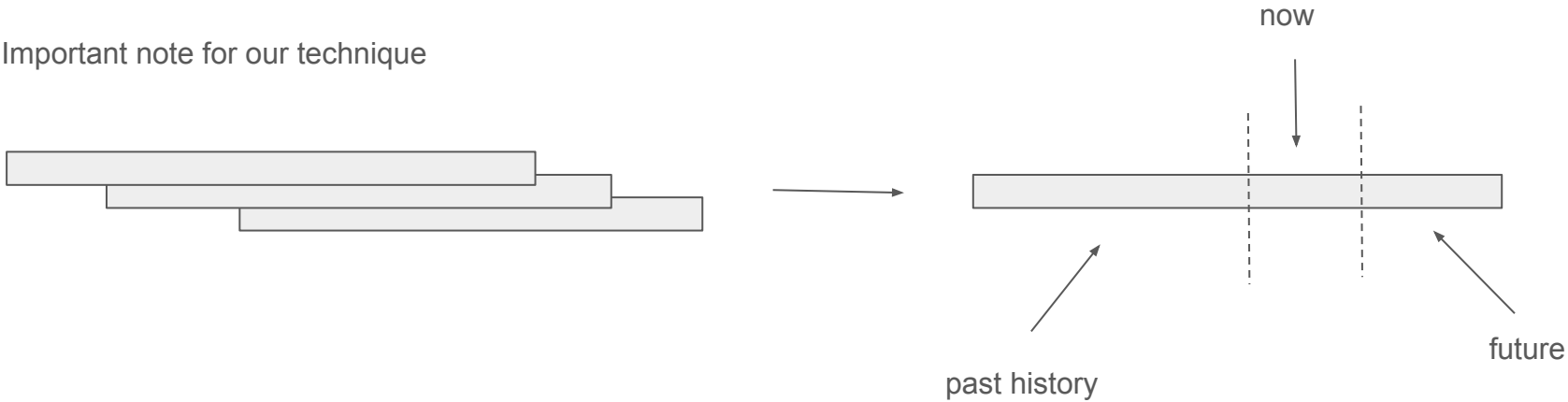
Proposed Solution

For our task, we will leverage neural networks to predict the loss rate in some steps in the future, hence we can send additional data corresponding to the amount of lost packets.



Proposed Solution

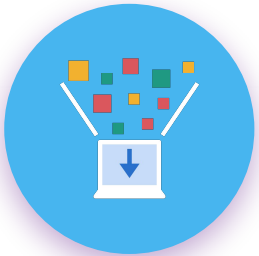
Important note for our technique



Proposed Solution

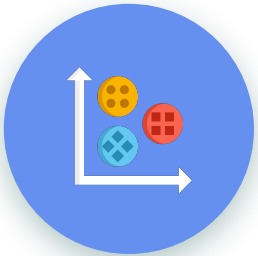
Below is the core ML part of our initial solution

Data Collection



Collecting raw data, dropping outliers, duplicated data, and splitting into subsequences

Clustering



Using kmeans to group the subsequences into clusters

Classification

Categorizing the subsequences into different classes.

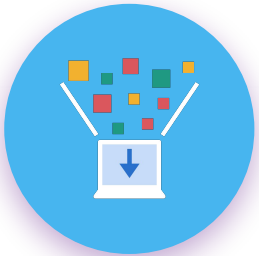
Regression

Predicting the loss rate based on the relationship between the other features in the data

Proposed Solution

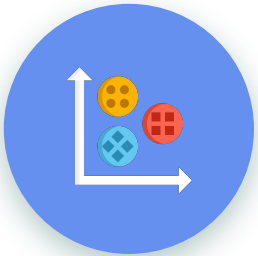
Below is the core ML part of our initial solution

Data Collection



Collecting raw data, dropping outliers, duplicated data, and splitting into **subsequences**

Clustering



Using a **DCAE model with kmeans** to group the subsequences into clusters and the classification is the cluster label

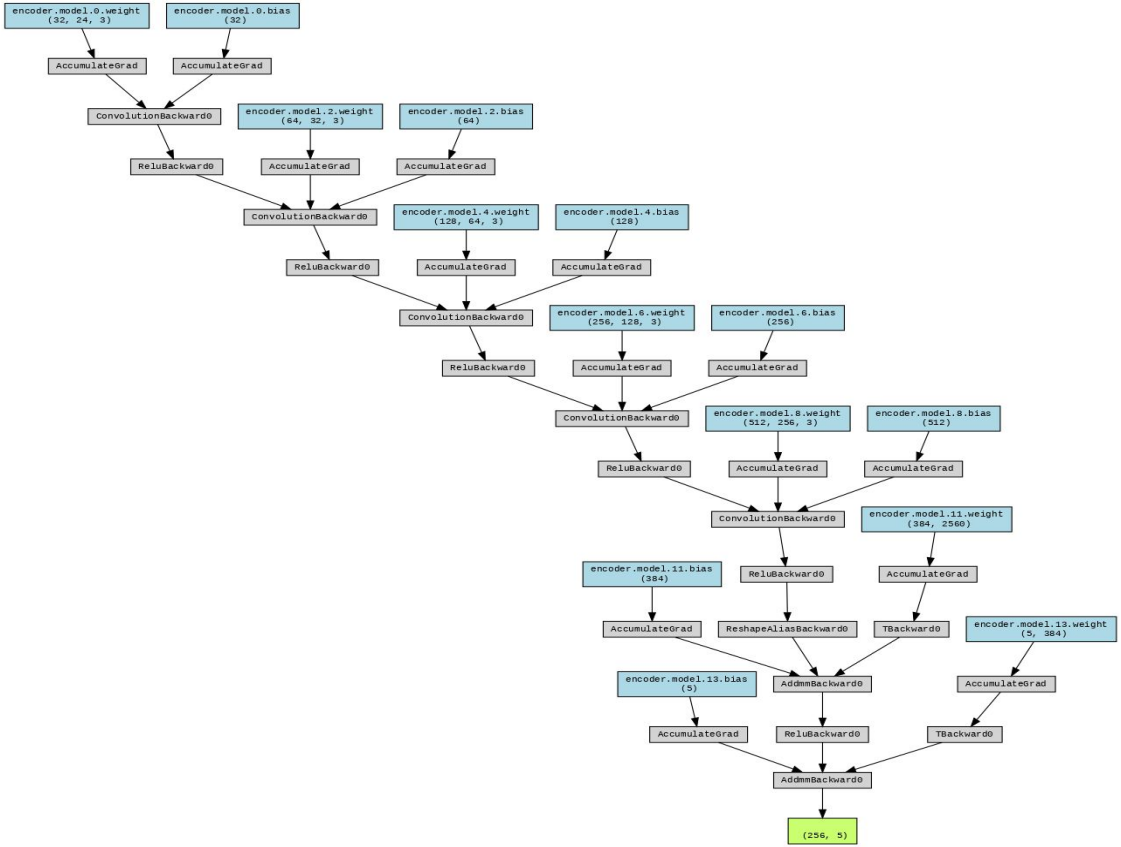
Classification

Categorizing the subsequences into different classes.

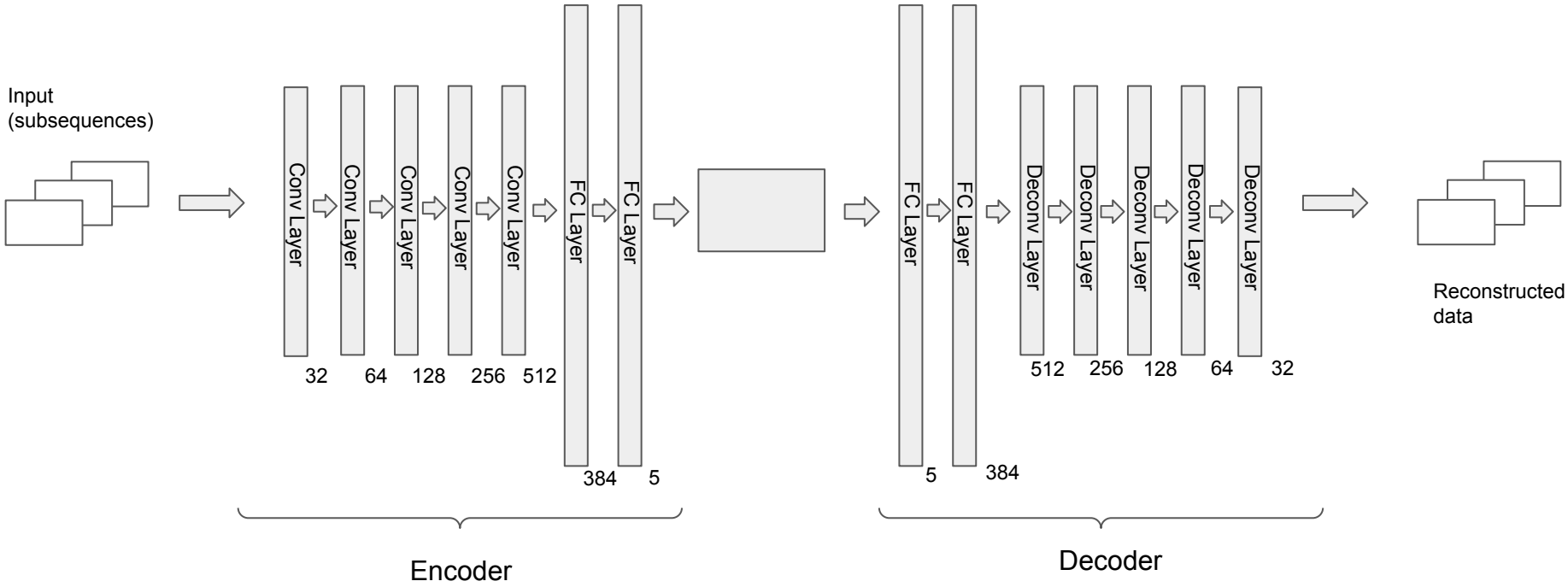
Regression

Using **lstm** to predict the loss rate based on the relationship between the other features in the data

Architecture- Autoencoder

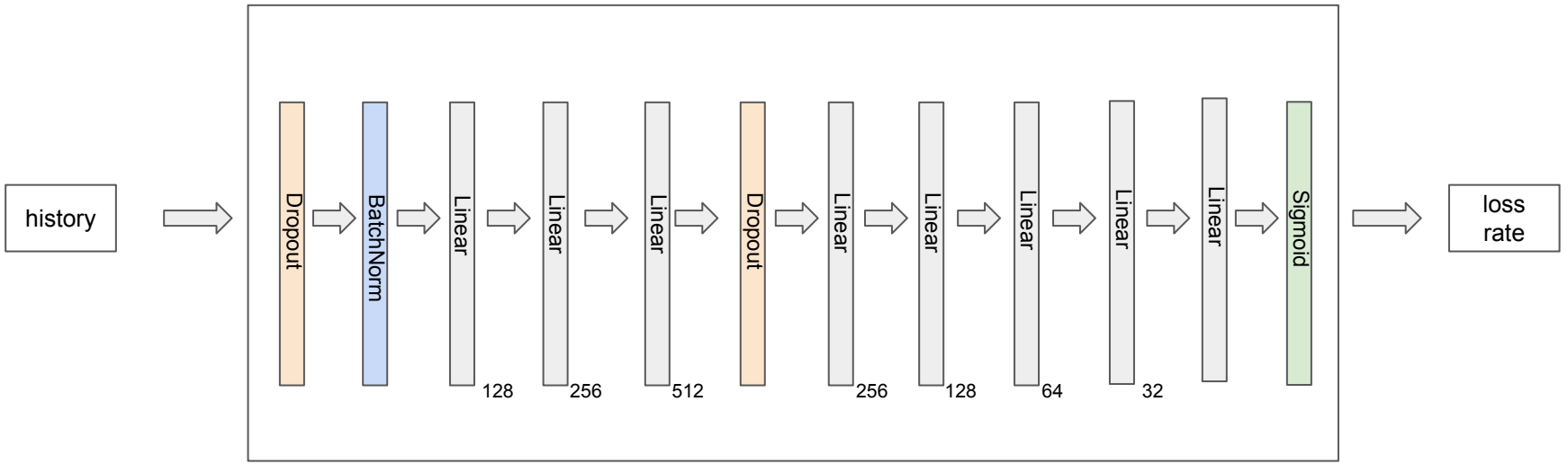


Architecture- DCAE

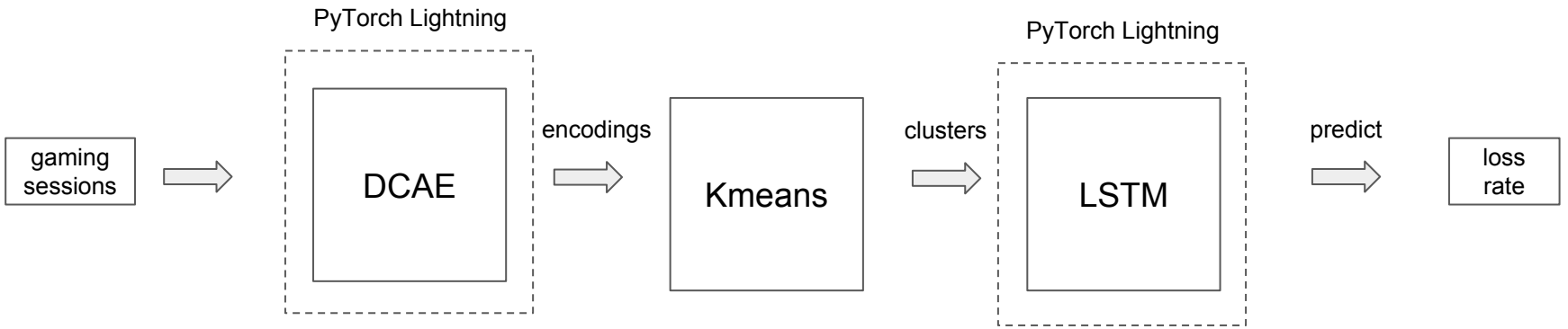


Architecture- LossRatePredictor

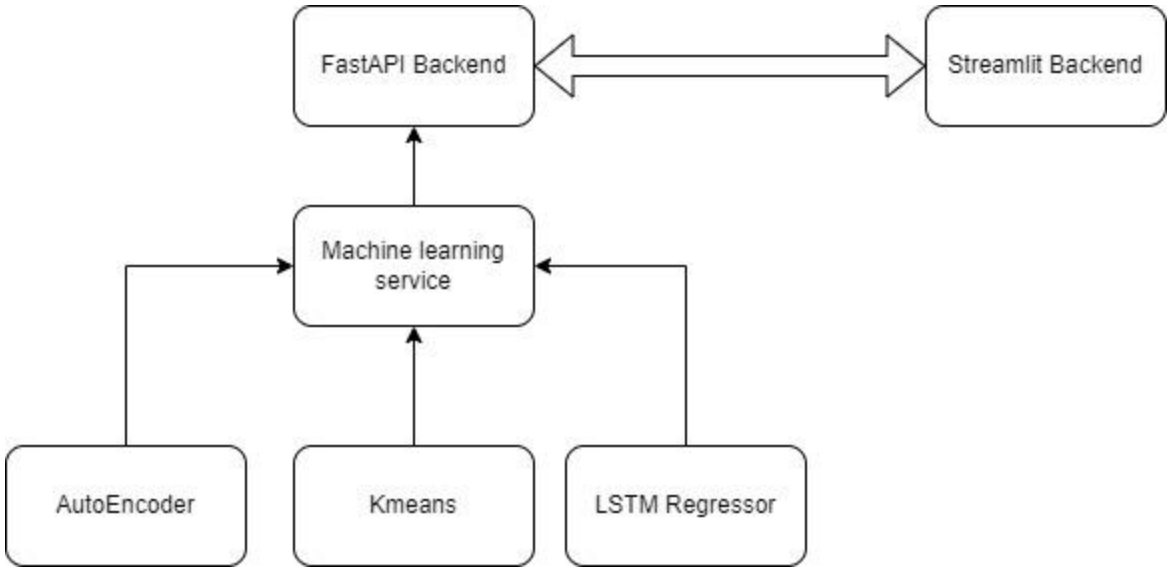
Architecture- LSTM



Architecture- Full view



Backend Architecture



Challenges and Changes

- So, we ended up **removing the classification section** as our clustering model automatically does that
- Also, our dataset was quite erroneous so we will need to **collect more adequate data**
- While we aimed to achieve no loss but we couldn't, this can be solved by **increasing model size** and getting more accurate data
- We eventually made use of **PyTorch Lightning** in both clustering and regression parts due to resources limitation

Challenges and Changes

- Since our technique involved **past, present and future**, we tried various combinations of subsequences and gaps to find the optimal
- Our **loss function was modified** in order to ensure that the predicted loss rate is never lesser than the actual loss rate.
- We needed to fine tune several deep learning models and then wire it all together
- In general, there is **not much research done in the field** compared to other ml fields like llm

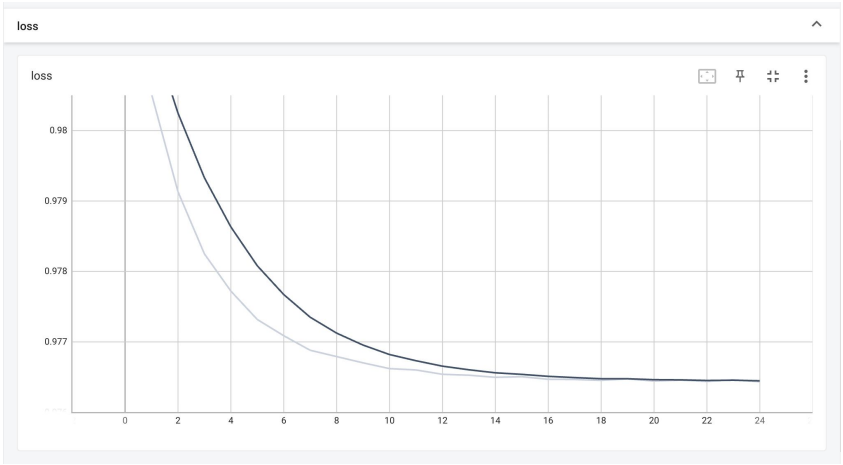
Metrics & Duration

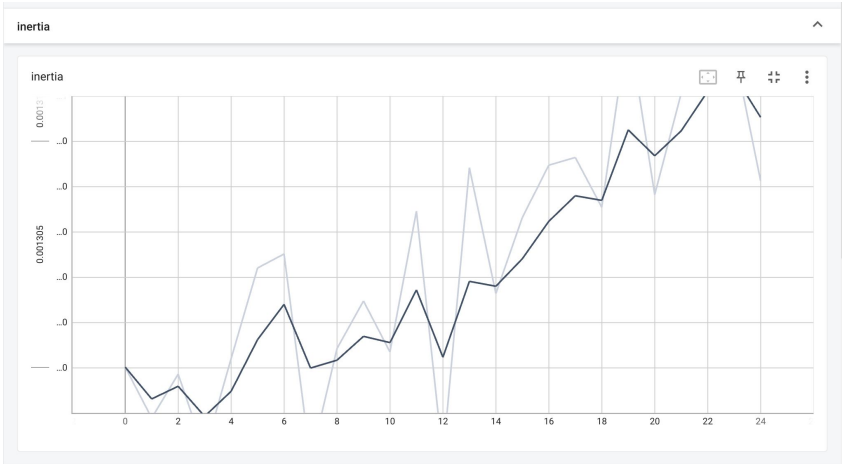
Below is a table to show our scores and time taken

	Clustering	Regressor 1	Regressor 2	Regressor 3
Training loss	0.9764			
Training inertia	0.0013			
Training mse loss		11.84	10.029	0.071
Custom training loss		0.071	0.070	0.071
Duration	0.25 (without gpu)			

Metrics & Duration

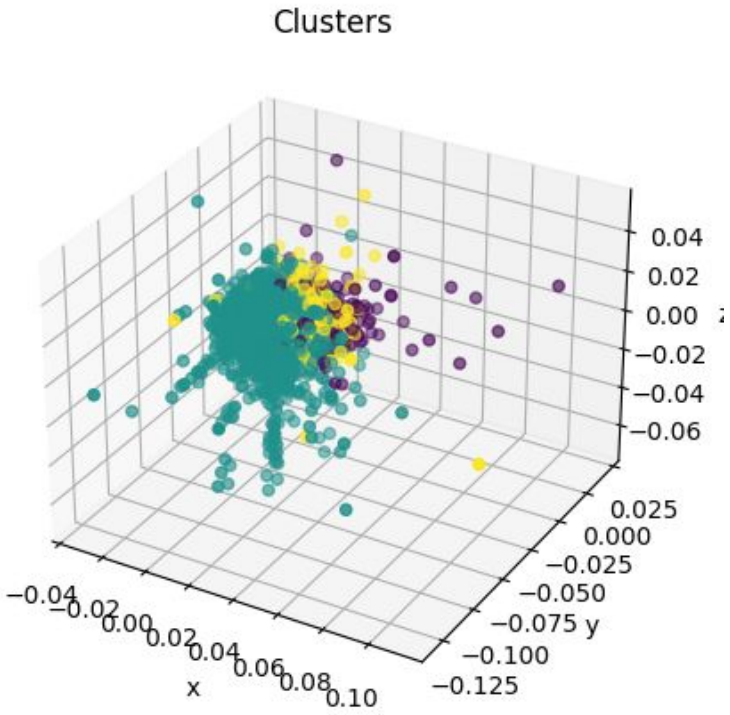
Below are the specifics for clustering training





Metrics & Duration

Below are the specifics for clustering training



Metrics & Duration

Below are the specifics for regression training

Regressor 1

	Training loss	Penalty loss	epoch	step
2	28.79806137084961	0.07111626118421555	0	208
3	11.836403846740723	0.07170604169368744	1	417
4	11.836403846740723	0.07170604169368744	2	626
5	11.836403846740723	0.07170604169368744	3	835
6	11.836403846740723	0.07170604169368744	4	1044
7	11.836403846740723	0.07170604169368744	5	1253
8	11.836403846740723	0.07170604169368744	6	1462
9	11.836403846740723	0.07170604169368744	7	1671
10	11.836403846740723	0.07170604169368744	8	1880
11	11.836403846740723	0.07170604169368744	9	2089
12	11.836403846740723	0.07170604169368744	10	2298
13	11.836403846740723	0.07170604169368744	11	2507
14	11.836403846740723	0.07170604169368744	12	2716
15	11.836403846740723	0.07170604169368744	13	2925
16	11.836403846740723	0.07170604169368744	14	3134
17	11.836403846740723	0.07170604169368744	15	3343
18	11.836403846740723	0.07170604169368744	16	3552
19	11.836403846740723	0.07170604169368744	17	3761
20	11.836403846740723	0.07170604169368744	18	3970
21	11.836403846740723	0.07170604169368744	19	4179

Regressor 2

Training loss	Penalty loss	epoch	step
65.08642578125	0.06843326985836029	0	141
10.02912712097168	0.07014656066894531	1	283
10.02912712097168	0.07014656066894531	2	425
10.02912712097168	0.07014656066894531	3	567
10.02912712097168	0.07014656066894531	4	709
10.02912712097168	0.07014656066894531	5	851
10.02912712097168	0.07014656066894531	6	993
10.02912712097168	0.07014656066894531	7	1135
10.02912712097168	0.07014656066894531	8	1277
10.02912712097168	0.07014656066894531	9	1419
10.02912712097168	0.07014656066894531	10	1561
10.02912712097168	0.07014656066894531	11	1703
10.02912712097168	0.07014656066894531	12	1845
10.02912712097168	0.07014656066894531	13	1987
10.02912712097168	0.07014656066894531	14	2129
10.02912712097168	0.07014656066894531	15	2271
10.02912712097168	0.07014656066894531	16	2413
10.02912712097168	0.07014656066894531	17	2555
10.02912712097168	0.07014656066894531	18	2697
10.02912712097168	0.07014656066894531	19	2839

Regressor 3

Training loss	Penalty loss	epoch	step
2422.181396484375	0.0	0	0
1988.323486328125	0.0	1	1
480.4396057128906	0.0357142873108387	2	2
0.07079140841960907	0.0714285746216774	3	3
0.0714285746216774	0.0714285746216774	4	4
0.0714285746216774	0.0714285746216774	5	5
0.0714285746216774	0.0714285746216774	6	6
0.0714285746216774	0.0714285746216774	7	7
0.0714285746216774	0.0714285746216774	8	8
0.0714285746216774	0.0714285746216774	9	9
0.0714285746216774	0.0714285746216774	10	10
0.0714285746216774	0.0714285746216774	11	11
0.0714285746216774	0.0714285746216774	12	12
0.0714285746216774	0.0714285746216774	13	13
0.0714285746216774	0.0714285746216774	14	14
0.0714285746216774	0.0714285746216774	15	15
0.0714285746216774	0.0714285746216774	16	16
0.0714285746216774	0.0714285746216774	17	17
0.0714285746216774	0.0714285746216774	18	18
0.0714285746216774	0.0714285746216774	19	19

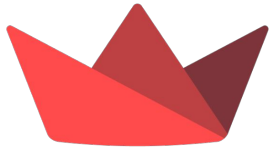
Future Improvements

- We need to improve on the clustering algorithm as ideal clusters for our task couldn’t be found
- Fine tune the models better, and try out other architectures to see how the performance differs
- Oversee the data collection process in order to get more accurate clusters. Data collection should focus on getting bad network.

Deploying

We made use of streamlit and FastAPI to prepare an interface that interacts with our models and is able to train and predict.

- Streamlit allows the end user to easily upload new datasets to finetune our preliminary model.
- While the fast api allows the end user to easily make predictions for new subsequences.



Streamlit

FastAPI

Demo