Objective: Understand and analyze the new dataset and lay ground work for building a pipeline system.

Conclusion: Got all methods to work, embeddings looking good enough, understand the data well enough, training and inference functional. Moving to try to build the pipeline.

- Core data: Using critical role dataset, git
- Support data: Dolly15k Dataset, arena dataset, open orca, orca math
- Extra possible data: gov hearings
- Using Dolly V2 and pythia models allowing work with pretrained types or instruction tuned

Plan:

1. Explore CR3 Dataset

2. Setup Dolly V2 and pythia for inference then training

3. Setup train inference pipeline (use analysis work) 6-24-24 CL Pipeline

Analysis original based on bigchat: 6-10-24-BigChat Analysis

- Set up summaries as QA pair scripts.

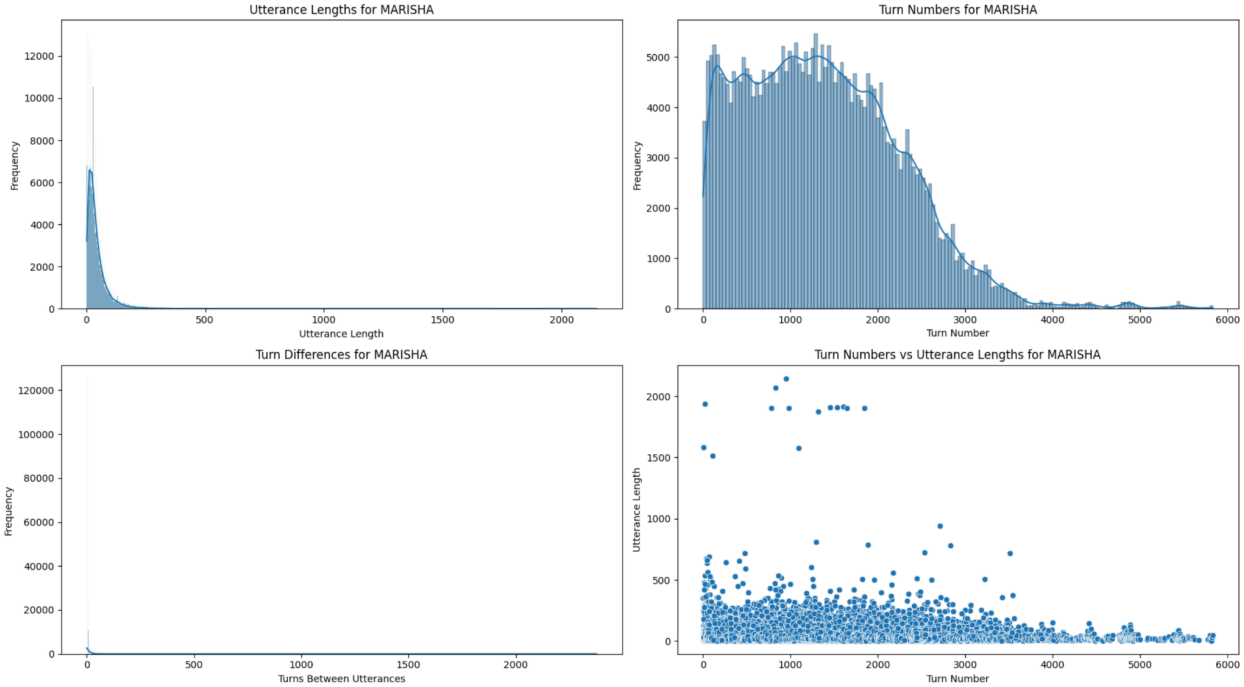
- training accuracies to follow:
    - General data
    - Questions on chunks
    - Response accuracy
    - Total inference output similarity size

NOTE: The initial data analysis here is on the uncleaned data. I was unaware that the dataset had massively repeated data.

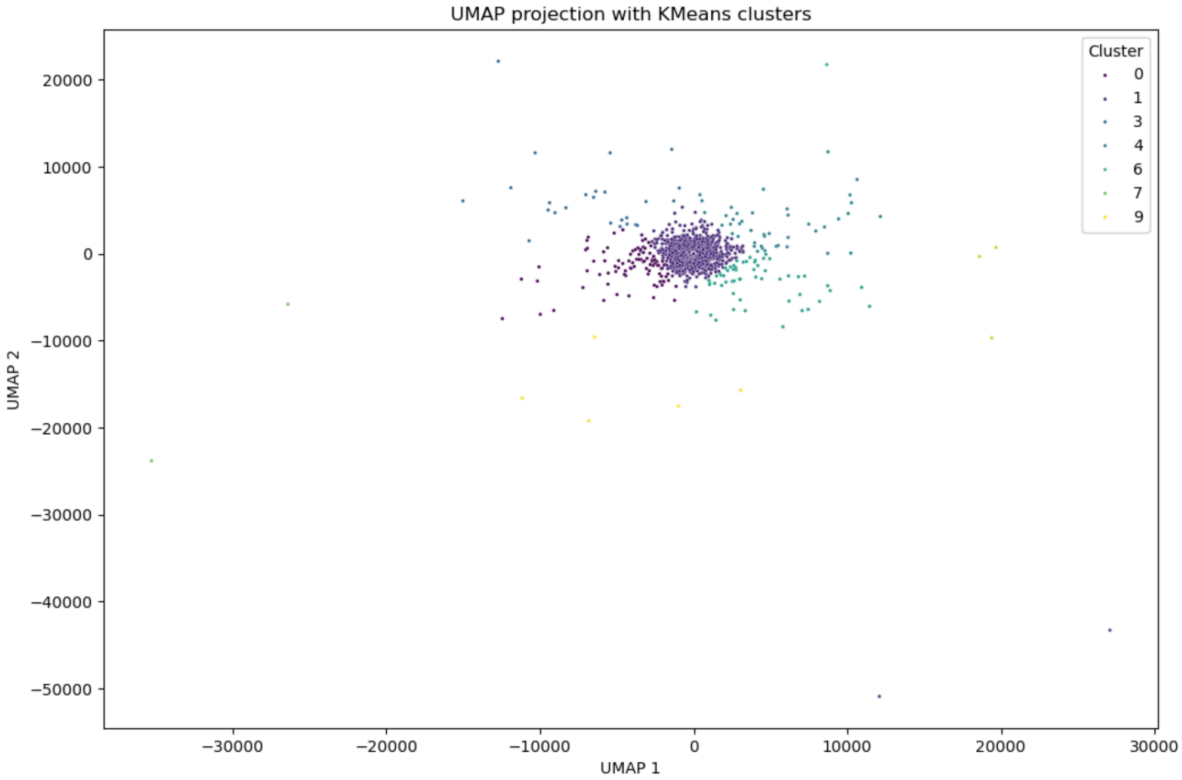
Text Analysis

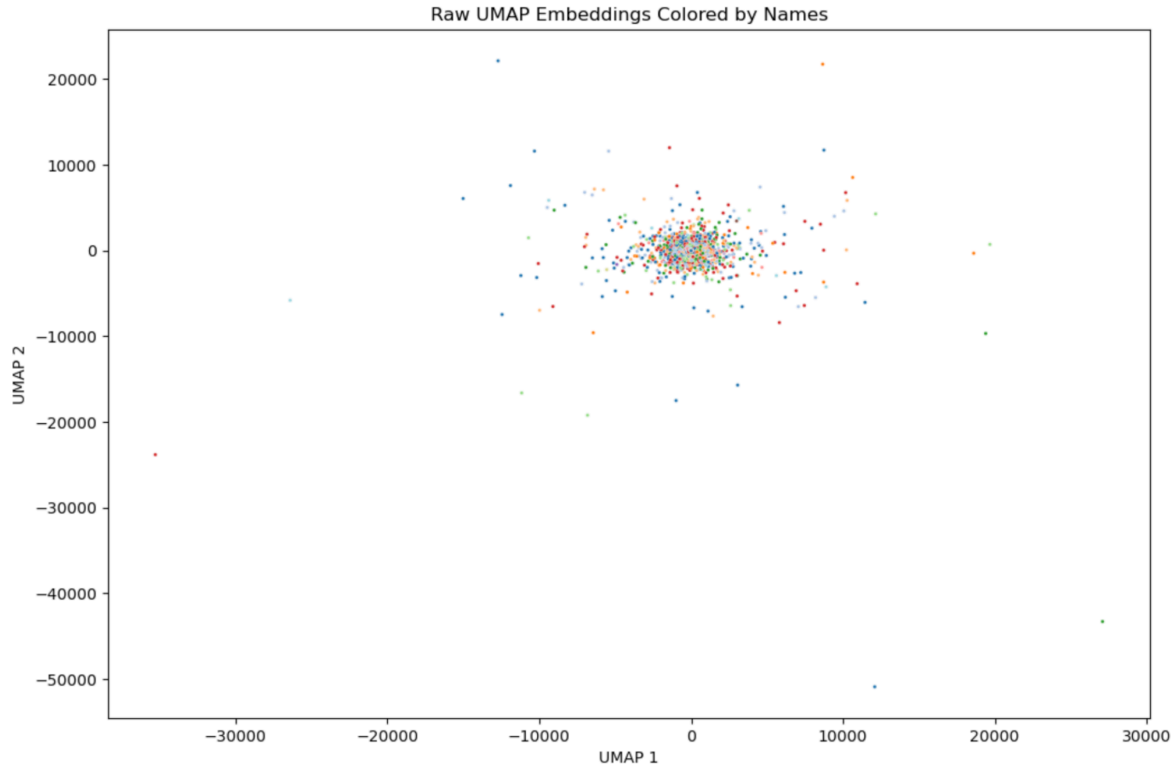
```
Campaign C2 Metrics:
Total Files: 411
Total Chunks: 19008
Total Turns: 1068103
Total Words: 15116857
Unique Names: TRAVIS, AUDIENCE MEMBER, DEBORAH, CAMERAMAN, BABS, MARK, SEAN, TOY IN BAG, JOEL, MATT, ANNE
T, TALIESIN, YURI, LAURA, JOEY, AMY, BRIAN, SOUND, MARISHA, JOE, KHARY, ASHLEY, ALL, WILL, LENORE, YASMIN
E, ALL - SAM, AUDIENCE, MAX, ERIC, LIAM, SUMALEE, SAM, ALLISON, CAST
Number of Unique Names: 35
Average Chunks per File: 46.25
Average Turns per File: 2598.79
Average Words per Utterance: 14.15
Common Names: LIAM, MARISHA, MATT, SAM, TALIESIN
Average Steps Between Utterances for Common Names:
  LIAM: 0.02
  MARISHA: 0.02
  MATT: 0.01
  SAM: 0.02
  TALIESIN: 0.03

Campaign C1 Metrics:
Total Files: 729
Total Chunks: 33788
Total Turns: 1874259
Total Words: 26115549
Unique Names: TRAVIS, ERIKA, ALL - TRAVIS - ORION - LIAM - AHSLEY - MATT, AUDIENCE MEMBER, STEVE, COMPUTE
R, CHRIS PRAMAS, WIL, KEVIN, KAI, NOELLE, IVAN, ORION, CHRIS HARDWICK, MATT, ZAC, ALL - MATT, PATRICK ROT
HFUSS, TALIESIN, ALL - ORION, TAYLOR, BECCA, LAURA, LUCAS, CHRIS PERKINS, ANNOUNCER, BRIAN, AMANDINE, DEN
ISE, SOUND, ALEC, DARIN, MARISHA, JOE, OFFSCREEN, ASHLEY, ALL, WILL, ALL - SAM - MATT, KIT, JON, SCREEN,
RYAN, HECTOR, AUDIENCE, JORE, IPHONE, LIAM, CREW, PRODUCER, IFY, FELICIA, SAM, GENERATED VOICE, MARY, CHR
IS WILLMOTT, ALL - TALIESIN
Number of Unique Names: 57
Average Chunks per File: 46.35
Average Turns per File: 2571.00
Average Words per Utterance: 13.93
Common Names: MARISHA, MATT
Average Steps Between Utterances for Common Names:
  MARISHA: 0.01
  MATT: 0.00

Summary Metrics:
Total Files: 1140
Total Chunks: 52796
Total Turns: 2942362
Total Words: 41232406
Unique Names: TRAVIS, ERIKA, ALL - TRAVIS - ORION - LIAM - AHSLEY - MATT, STEVE, KEVIN, KAI, NOELLE, ORIO
N, MARK, SEAN, JOEL, ALL - MATT, ANNET, TALIESIN, ALL - ORION, YURI, TAYLOR, LAURA, LUCAS, JOEY, AMANDIN
E, DENISE, SOUND, JOE, WILL, ALL, LENORE, YASMINE, ALL - SAM, MAX, JORE, IPHONE, ERIC, CREW, IFY, PRODUCE
R, SUMALEE, FELICIA, ALLISON, MARY, ALL - TALIESIN, AUDIENCE MEMBER, COMPUTER, CHRIS PRAMAS, WIL, DEBORA
H, CAMERAMAN, IVAN, BABS, CHRIS HARDWICK, TOY IN BAG, MATT, ZAC, PATRICK ROTHFUSS, BECCA, CHRIS PERKINS,
AMY, ANNOUNCER, BRIAN, ALEC, DARIN, MARISHA, KHARY, OFFSCREEN, ASHLEY, KIT, SCREEN, ALL - SAM - MATT, JO
N, RYAN, HECTOR, AUDIENCE, LIAM, SAM, GENERATED VOICE, CAST, CHRIS WILLMOTT
Number of Unique Names: 77
Average Chunks per File: 46.31
Average Turns per File: 2581.02
Average Words per Utterance: 14.01
Common Names: MARISHA, MATT
Average Steps Between Utterances for Common Names:
  MARISHA: 0.01
  MATT: 0.00
```

```
Analyzing data for the name: MARISHA...
Total Utterances: 329039
Average Utterance Length: 44.29
Median Utterance Length: 31.00
Min Utterance Length: 2
Max Utterance Length: 2147
Average Turn Difference: 8.61
Median Turn Difference: 3.00
Min Turn Difference: 0
Max Turn Difference: 2377
Standard Deviation of Utterance Lengths: 58.43
Standard Deviation of Turn Differences: 16.02
Total Turn Numbers: 329039
Total Unique Turn Numbers: 3864
```



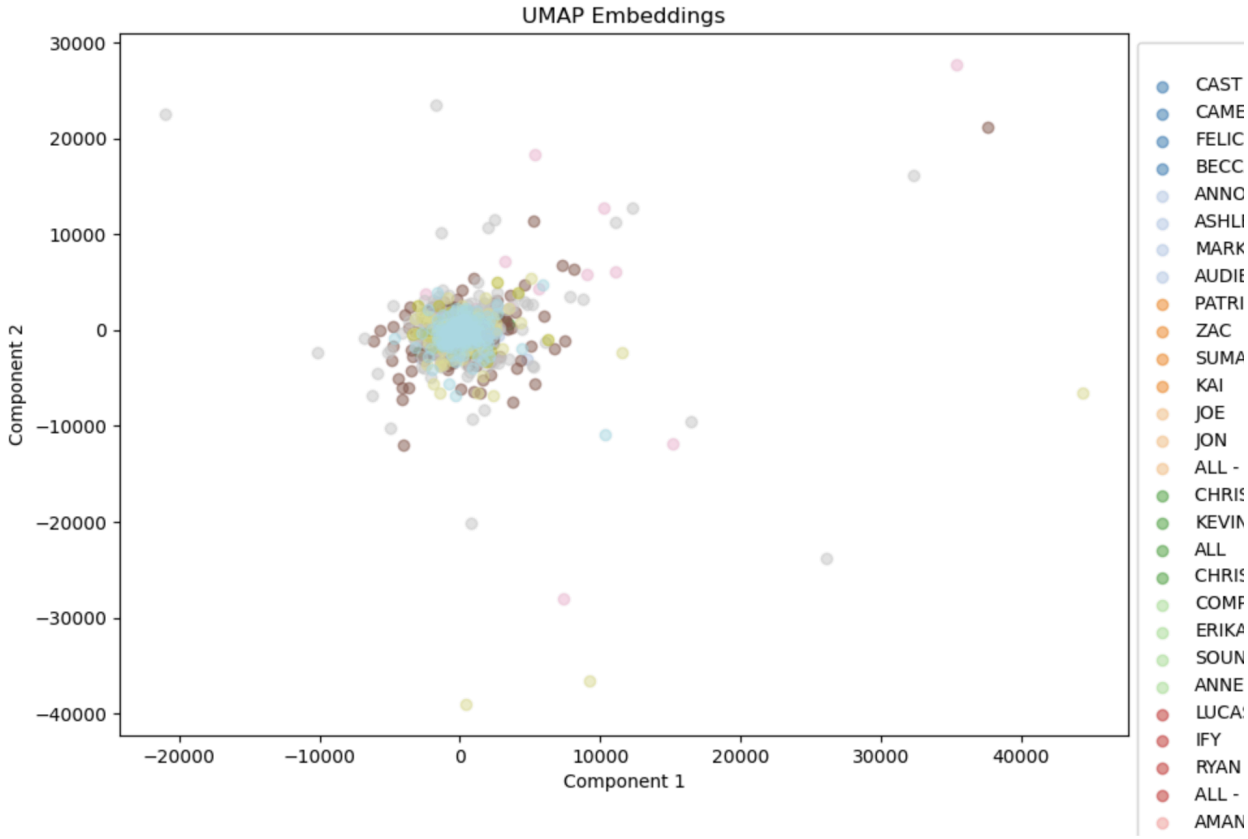
6-18-24





This embedding method sucks, need a better one. Using original bert text encoder training.

Still massive outliers? Having trouble getting it to embed well, but this is a very undertrained embedding model below. Trying much longer training. Also going to run the previous progressive training method.
5000 samples, 10 epochs, standard training.



Training is crazy slow on all data, going to embed with openai and try again. Also reducing size to 64. Using c=3 data, ~630000 utterances

Tried retraining model accuracy was poor

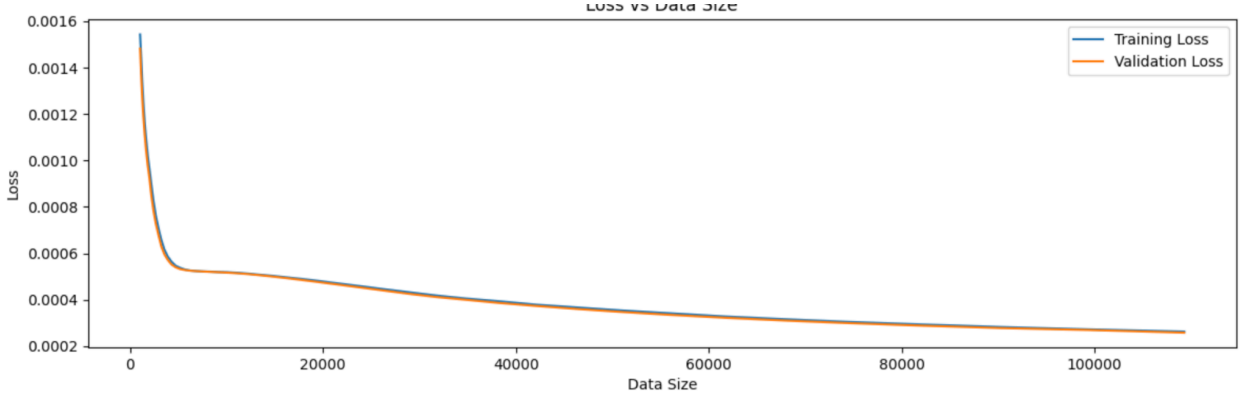
Following work is using openai Embedding methods. I generated two datasets (repeated embedding same code) So I will color the graphs plots and models training accordingly to what the data was that generated it. I did often switch up models inference different data so I will only label by data used

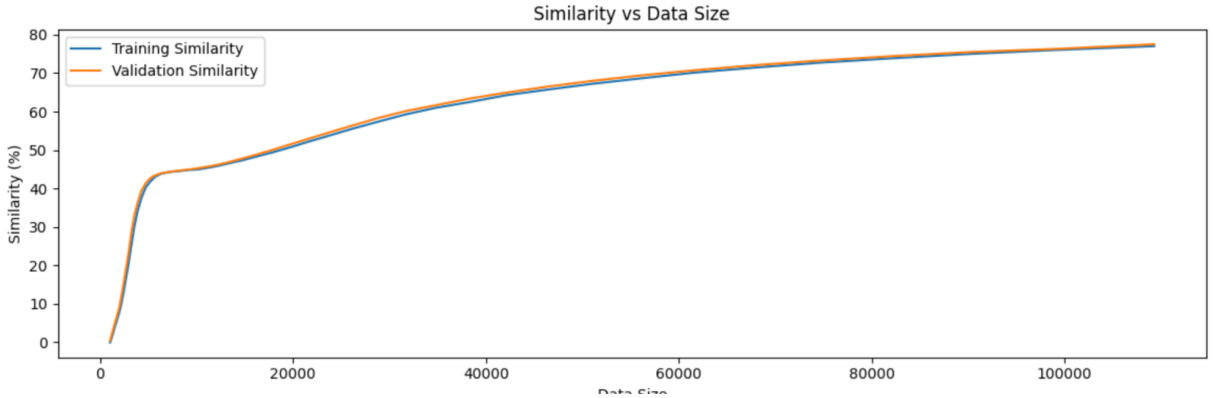
DS1
DS2

6-19-24 going to try AE on embeddings and progress training

- I was using a bad accuracy calculation method, not at all the same as previous methods.

- Simple linear model with correct accuracy metric:
  initial_size = 1024 # Initial training data size and batch size
  increment_ratio = 0.1 # Increment ratio for progressive training
  max_epochs = 50 # Maximum number of epochs
  Epoch 50/50 | Data size: 120208 | Train Loss: 0.00026219 | Val Loss: 0.00025741 | Train Similarity: 77.03% | Val Similarity: 77.52%



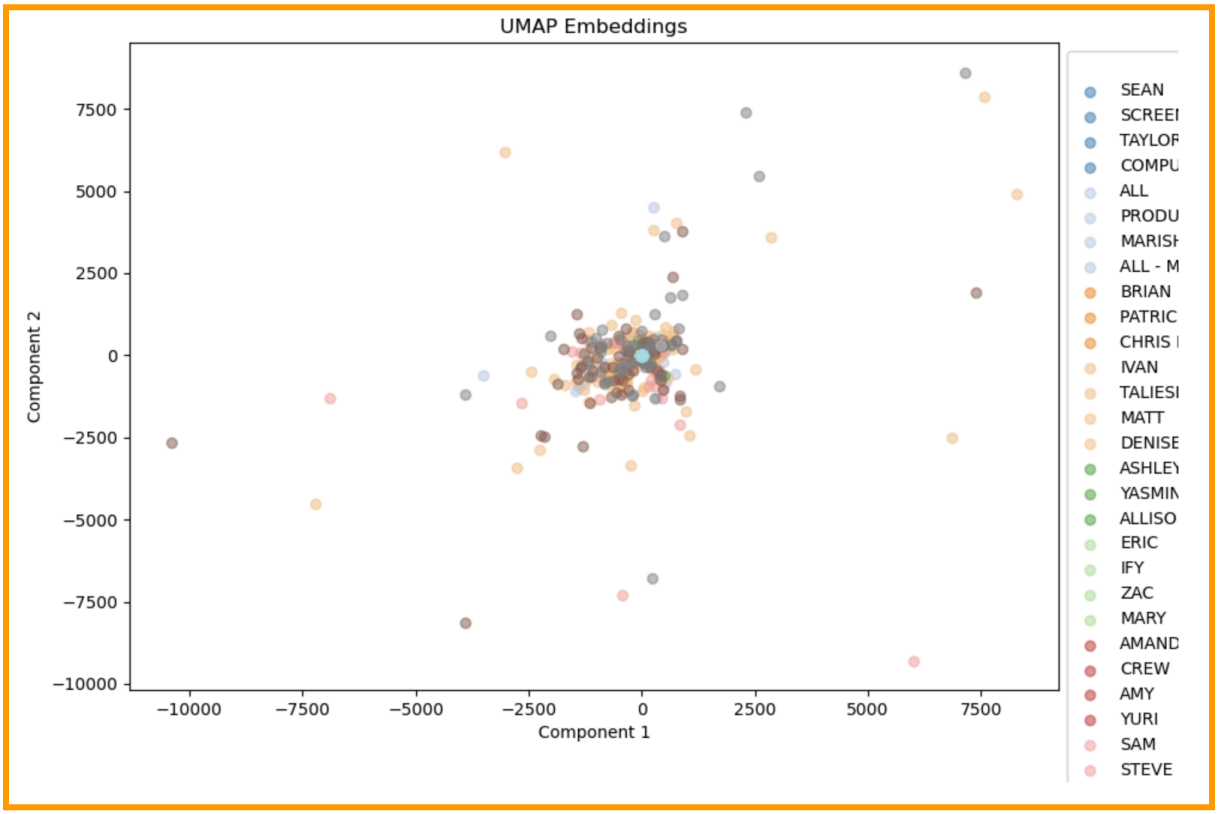


- Even original training method was still pretty good at first epoch:
  Epoch 3/5000 | Data size: 1055.028224 | Train Loss: 9.9562 | Val Loss: 9.8111 | Train Acc: 5.78% | Val Acc: 8.64%

- Bert embedding training

Epoch 33/500 | Data size: 190260 | Train Loss: 0.00054292 | Val Loss: 0.00052701 | Train Similarity: 41.12% | Val Similarity: 43.65%

Bert standard training trains ok similar just worry about overfitting if I need to train it automatically.
Epoch 15/50 | Train Loss: 0.00037346 | Val Loss: 0.00034274 | Train Similarity: 64.87% | Val Similarity: 68.45%



Did more embedding methods and I don't see anything yet. Could be similar to early work, not training enough. I did not train much for these embeddings above. Raw embeddings don't look much better will try a longer training session and redo embeddings.

6-20-24
Embedding data was corrupted and I couldn't get it onto the server
Model training is in a good spot. Basic examples both working.

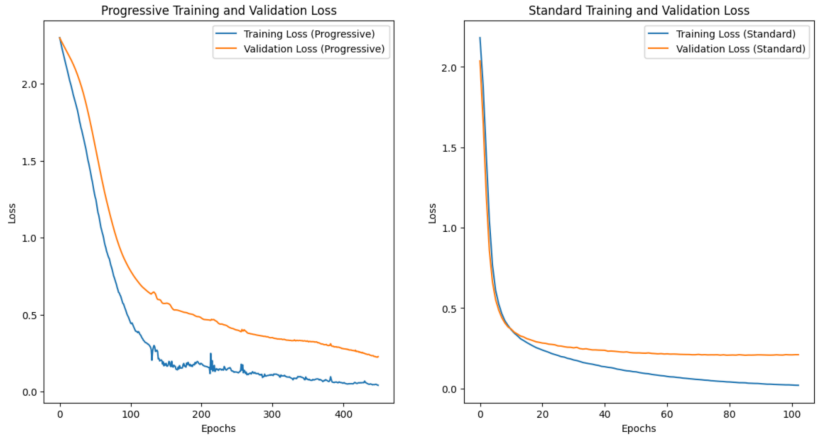
I'll build from the gpt neo version: Use shuffle every step (results below)

Progressive training no mixing per step:
# Configuration
INITIAL_SIZE = 80 # Initial dataset size
INCREMENT_RATIO = 0.1 # Increment ratio for each step
EPOCHS = 50
BATCH_SIZE = 8
LEARNING_RATE = 5e-5
MODEL_NAME = "EleutherAI/pythia-70m"

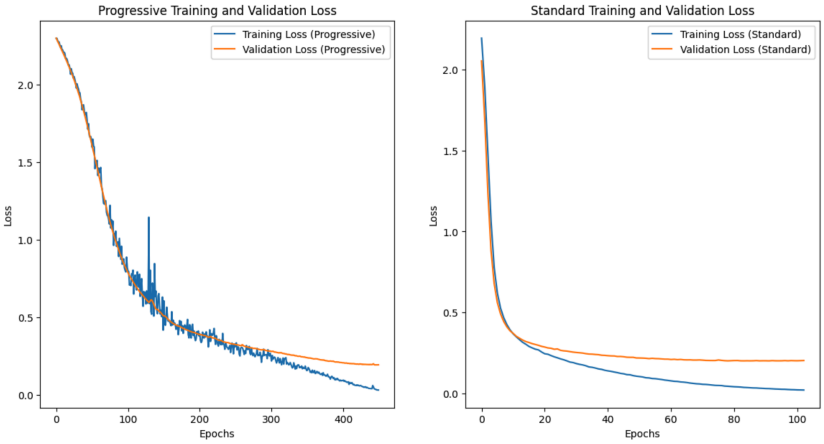
Ep 50/50 | Data size: 8085 | Trn Loss: 0.1780 | Tst Loss: 0.3965 | Trn Acc: 0.0035 | Tst Acc: 0.0036 | Pplx: 1.1949

Overall the training is working and I'm ready to scale it out while testing my training methods to ensure I can train each epoch on its own. Put training on hold as my 2nd embedding DS is ready.

Stupid checking my progressive train method once again:
Training basic mnist model on both methods and comparings keeping as similar as possible make sure to train model on exact same number of data samples regardless of method:
Results: its better, takes longer, I don't need to worry about it as much, no plateauing losses
Total data trained on (progressive): 516272 Total data trained on (standard): 515000
No random shuffling:
Meh on training fixed progressive
progressive:
Data size: 4994 | Trn Loss: 0.0410 | Val Loss: 0.2271 Total training time (progressive): 583.05 seconds
standard:
Epoch 103/103 | Trn Loss: 0.0199 | Val Loss: 0.2103 Total training time (standard): 179.72 seconds



Random shuffling:
Noisier but better results
Progressive:
Data size: 4994 | Trn Loss: 0.0328 | Val Loss: 0.1955 Total training time (progressive): 594.18 seconds
Standard:
Epoch 103/103 | Trn Loss: 0.0213 | Val Loss: 0.2035Total training time (standard): 192.81 seconds



Seems to be equivalent to finding optimizing LR/dropout of standard but is automatic. **USE IT**
Would be even better if I had an adjustable LR or data ratio metric

Trying to optimize autoencoder progressive training:

- It's hard to keep my gpu optimized and running at 100%

original
Data size: 12902 | Train Loss: 0.00796465 | Val Loss: 0.00433617 | Train Similarity: 1.14% | Val Similarity: -1.69% | Time per data point: **0.00040735 seconds**
Data size: 13547 | Train Loss: 0.00445149 | Val Loss: 0.00235238 | Train Similarity: -0.49% | Val Similarity: 1.54% | Time per data point: 0.00039000 seconds
Data size: 14224 | Train Loss: 0.00217701 | Val Loss: 0.00142907 | Train Similarity: 2.01% | Val Similarity: 3.93% | Time per data point: 0.00036768 seconds

New method:
Data size: 12902 | Train Loss: 0.00865330 | Val Loss: 0.00653226 | Train Similarity: -1.01% | Val Similarity: -0.73% | Time per data point: 0.00012675 seconds
Data size: 13547 | Train Loss: 0.00579376 | Val Loss: 0.00296037 | Train Similarity: -0.42% | Val Similarity: 1.76% | Time per data point: 0.00010967 seconds
Data size: 14224 | Train Loss: 0.00274119 | Val Loss: 0.00177281 | Train Similarity: 2.09% | Val Similarity: 3.41% | Time per data point: 0.00010861 seconds

4x increase in speed!

Training full embedding method:

```
initial_size = 12288  # Initial training data size and batch size
increment_ratio = 0.05  # Increment ratio for progressive training
max_epochs = 5000  # Maximum number of epochs
```

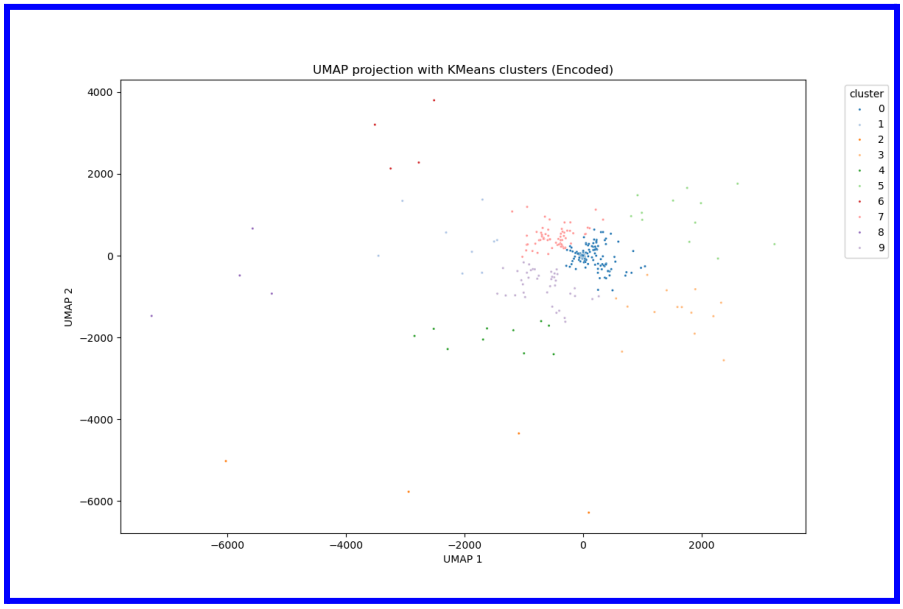
Trained pretty fast, decent accuracy, needs more training I think
Data size: 816132 | Train Loss: 0.00042585 | Val Loss: 0.00040956 | Train Similarity: 58.17% | Val Similarity: 60.27% | Time per data point: 0.00011988 seconds

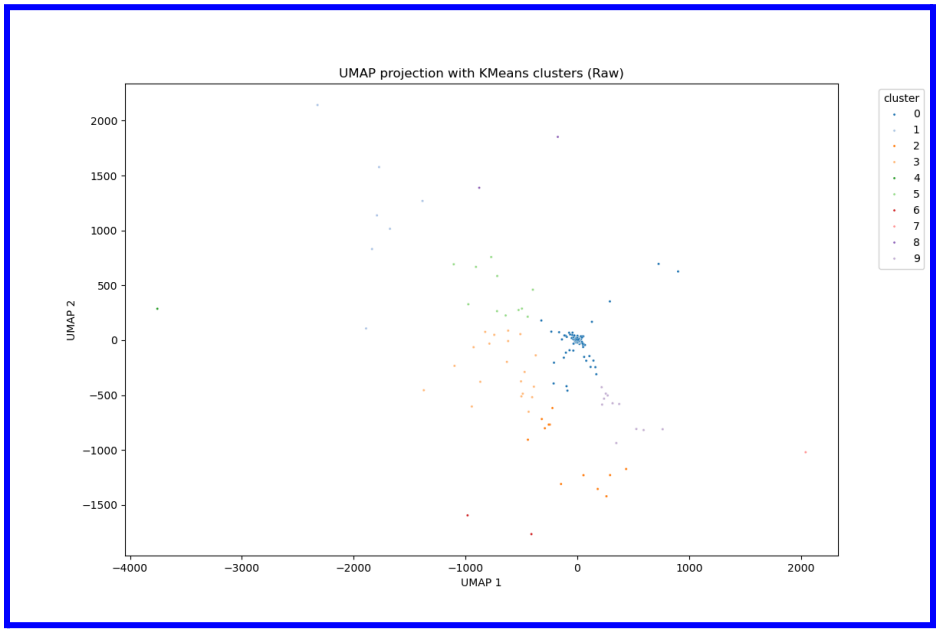
This was nowhere near trained (see below)

6-21-24
Looking at embedding method with encoder, may need to retrain slower for higher accuracy, also forgot to save the output.

Output of embedding comparisons seems to validate that autoencoder expands the embeddings space and gives more structure.

Also seems to keep a similar global structure which is reassuring that I am not just losing all my information.

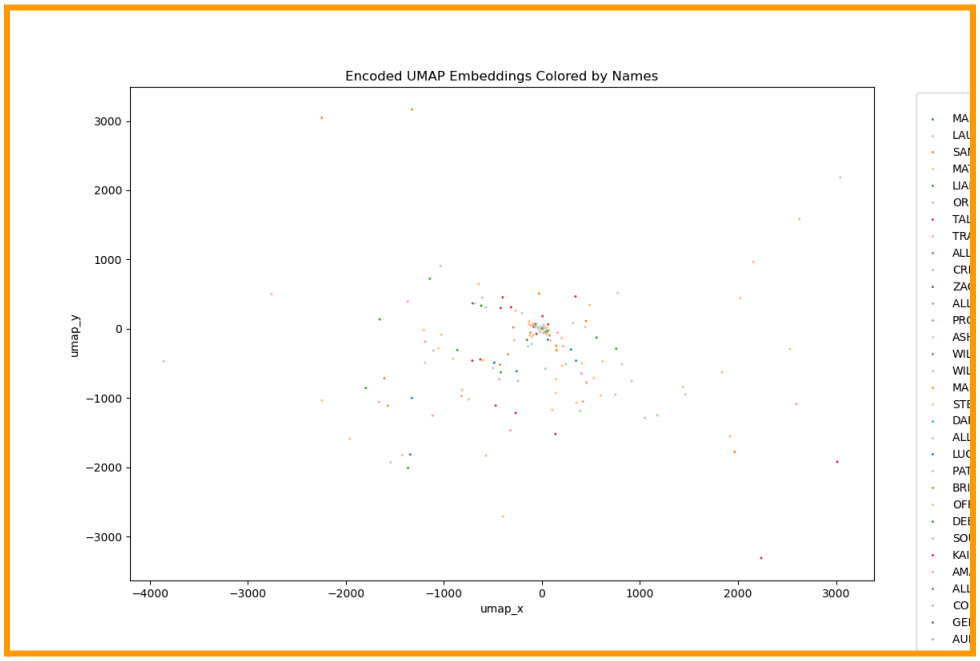




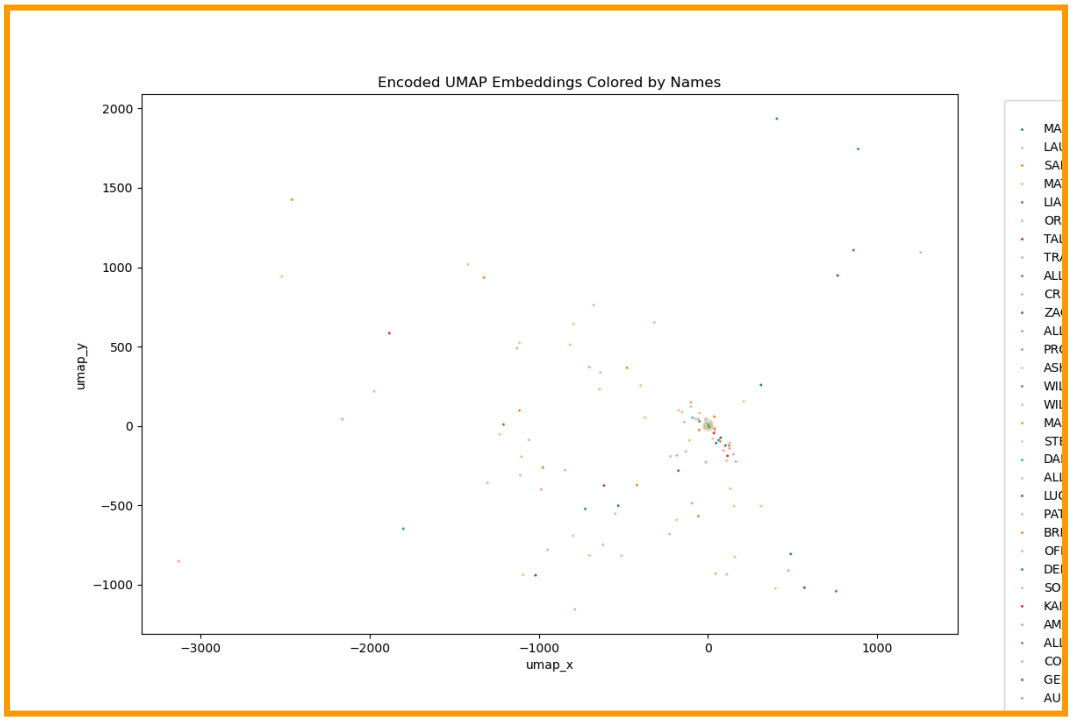
Can I get the same kinds of outputs by just doing PCA and visualizing? Perhaps an autoencoder is not needed. >> No, pca is terrible.

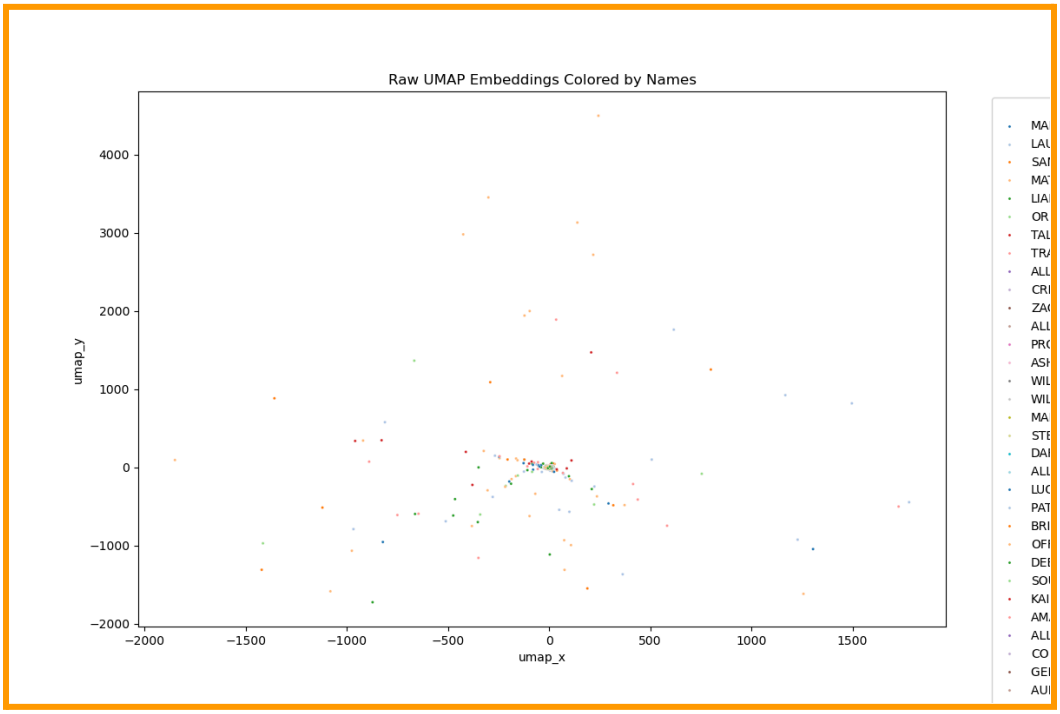
- **I have generated 2 sets of embeddings from utterances**

The below visuals are the same as above but the model used for encoding was trained on set 2 of the embeddings and this was applied to set 1. So it should not work as well but it is interesting to note that it still seems to bring out information.
Experiment ID: 60%AE_DS2_Model_DS1_Inference



This above example was weird, could not reproduce it? Below is another example run like above. 60%AE_DS2_Model_DS1_Inference_2

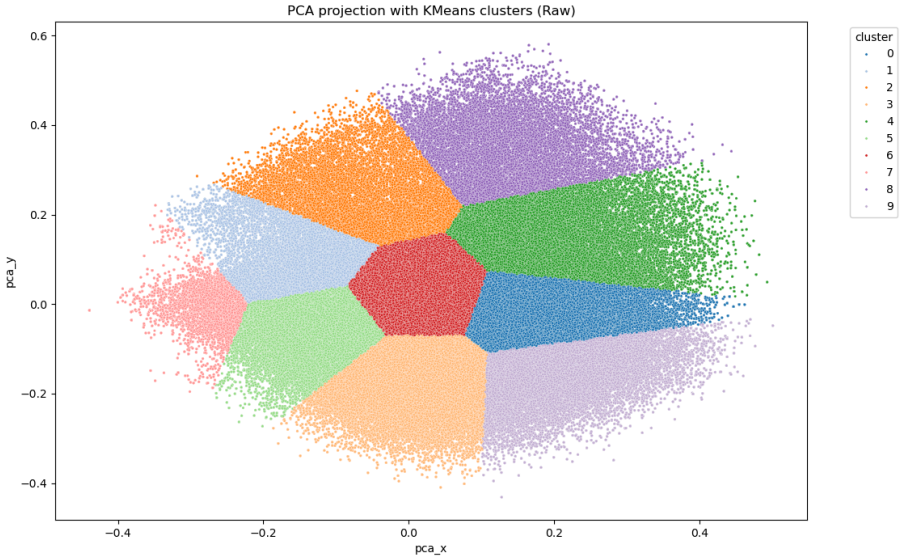


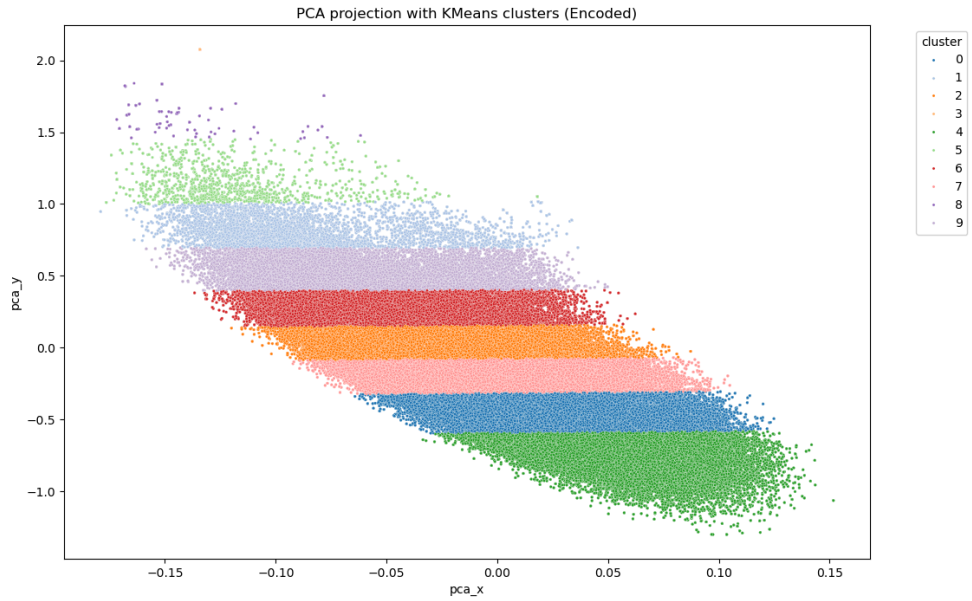


It looks really similar which I suppose is a good thing. Since the encoder was under trained and trained on an different embedding set. Weird it did not reproduce the result above though. I'm going to do one more run on a fully well trained autoencoder. In the >90% accuracy range.

This is repeatable, running it more than once gives the same umap results.

PCA is weird: **looks the same in every experiment So won't use it anymore**
It clearly works to encode it but at the same time I am now worried my embeddings are in fgeneral junk and I am increasing resolution onto noise.





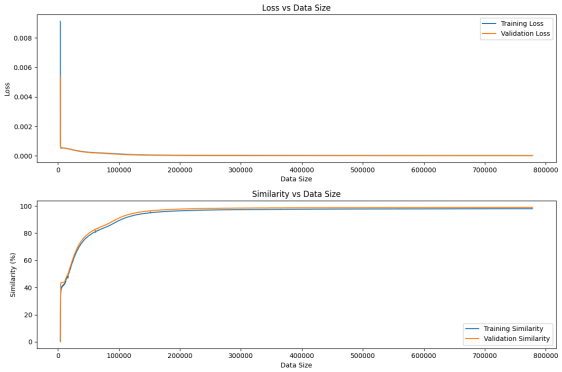
Training 2 autoencoders one on each repeated embedding data.
Running two full analysis of raw vs autoencoder types.
- one of them runs the analysis on the original model but different data
- other runs on original data it was trained

I will repeat this when both new high accuracy models are trained.
- two >95% accuracy models trained on repeated embedding data.

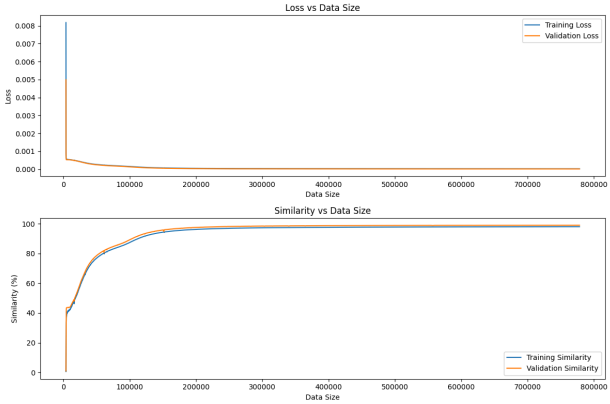
This will allow me to determine the quality of the original embedding method. See noise between the methods and ensure that the openai embedding is suitable for my analysis.

```
initial_size = 4096  # Initial training data size and batch size
  increment_ratio = 0.005  # Increment ratio for progressive training
  max_epochs = 50000  # Maximum number of epochs
```

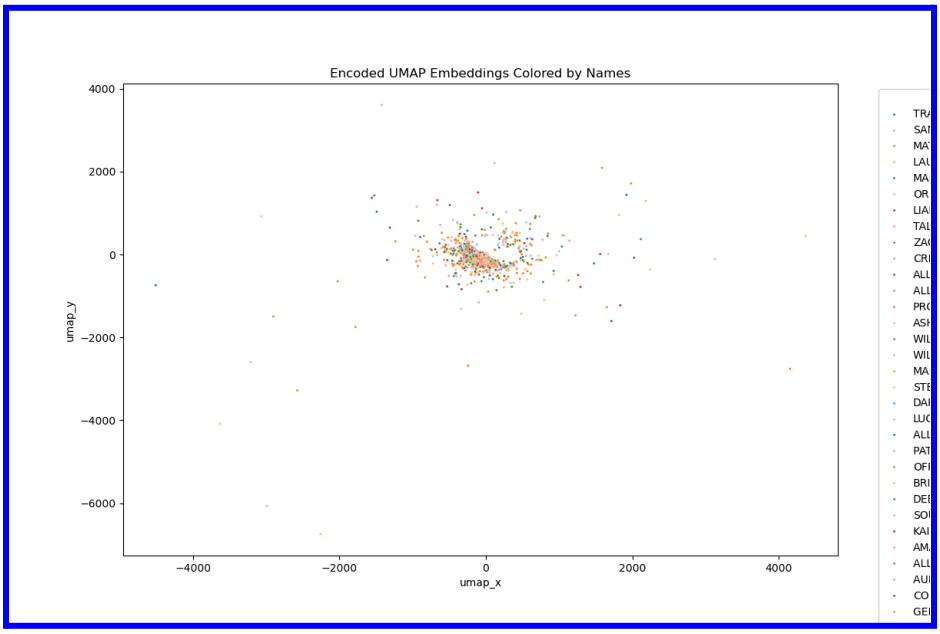
Trained on DS #2
Data size: 782029 | Train Loss: 0.00002599 | Val Loss: 0.00001428 | Train Similarity: 97.99% | Val Similarity: 98.98%



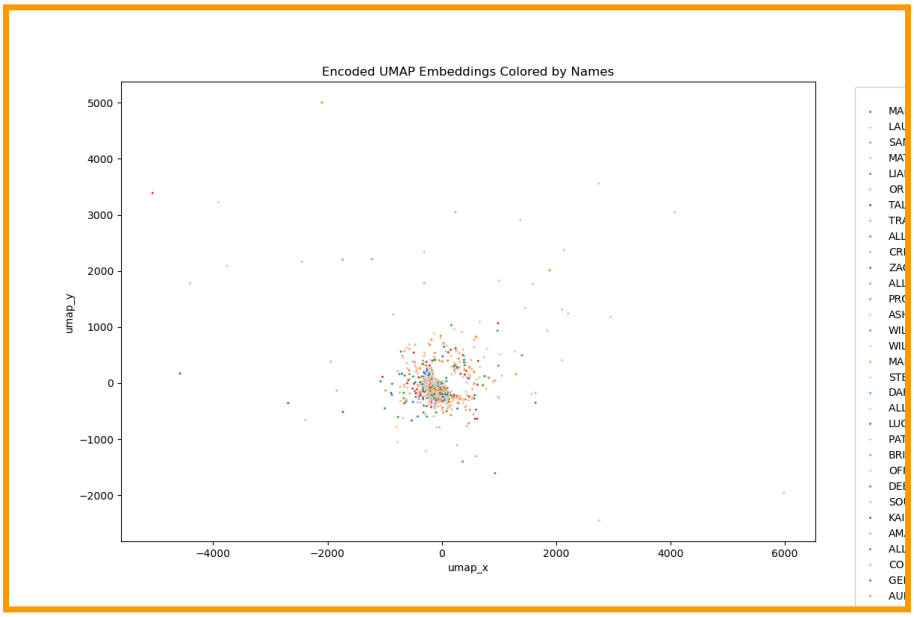
Trained on DS1:
Data size: 782029 | Train Loss: 0.00002593 | Val Loss: 0.00001414 | Train Similarity: 97.99% | Val Similarity: 98.99%



Running analysis of embeddings with new model. Not happy?



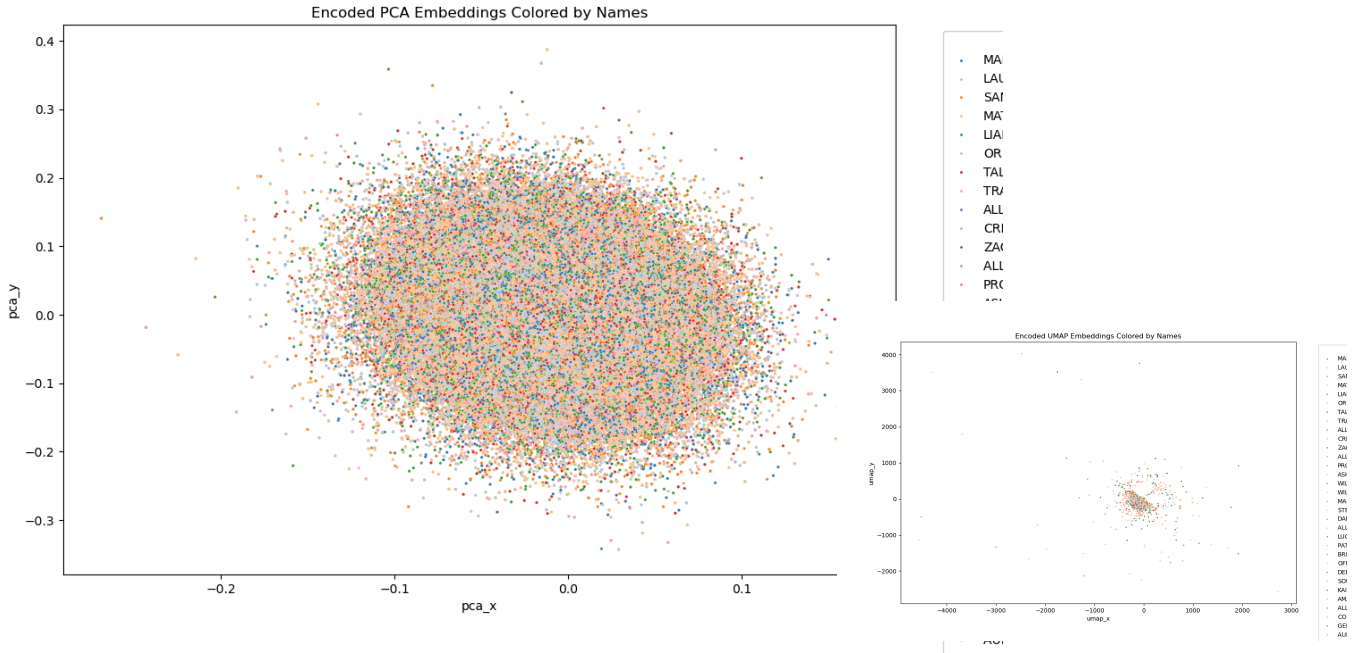
It clearly expanded it and there is a lot more structure but still not great I feel. I may have to abandon this embedding method for more traditional text analysis methods.
As a second thought though looking back at previous data it does massively expand the central blob of data. I have to realize these are 800k points of data so it's very possible this is a significant amount of resolution into the output. I would say I should still use it but do an integration confirmation with some standard text analysis methods as well.
6-22-24

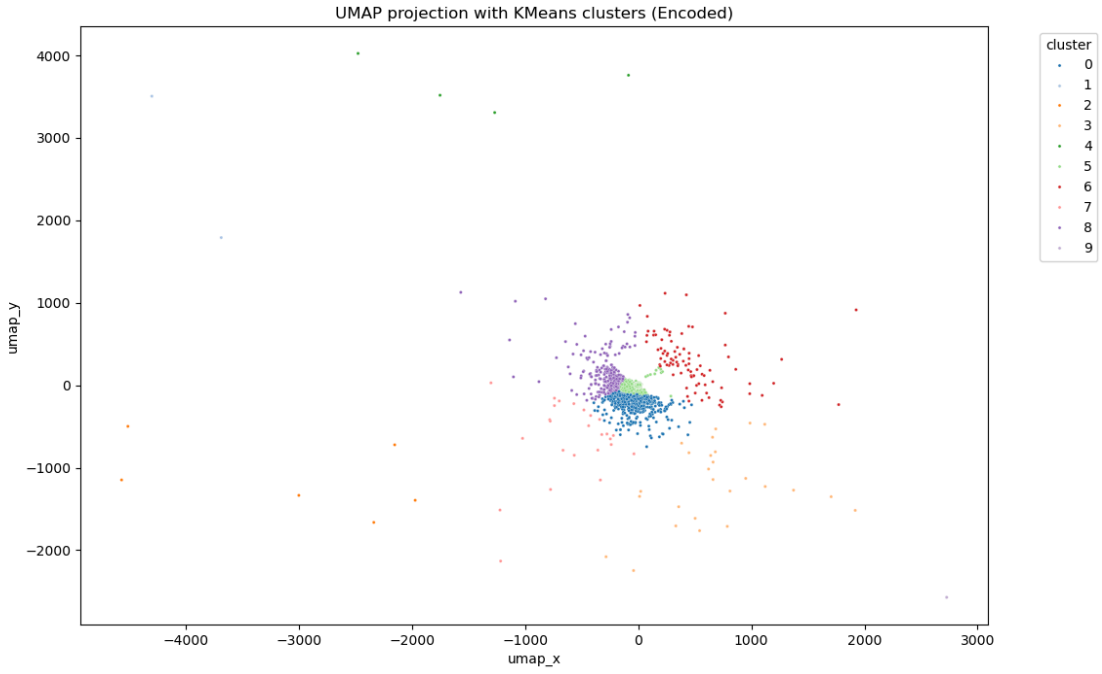


Repeatability seems high Both of these above are trained on their respective datasets and then inference. Seems to imply that the original embedding model even if it is "wrong" it is at least it's repeatably wrong which is a good sign for future work.

V1 analysis code on high accuracy test Experiment ID:
90%AE_DS1_MODEL_DS1_CodeV2/V1

- Slightly different results bur still on track. PCA is quite different that 60%, Is that better?
- Following 4 are DS1 data on DS1 trained model
- The same umap structure repeats in all examples for high acccuarcy and low. More similar in high accuracy.

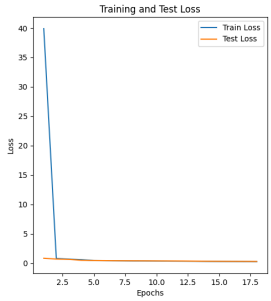




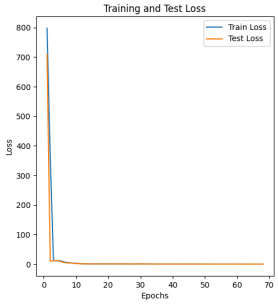
Overall the embedding results are interesting and promising. I believe that I have enough understanding to start creating the full system with multisteps to start analysis. Next stage will be to build an inference of a single step and analyze the results along with the correct response.

6-24-24
First trying to get model training and inference on the sets of utterance data.

Training: Progressive training LLM model's behave the same as the other models. Train and val follow very closely. Good sign probably will train well. This is on dolly 15k data only, progressive growth no shuffle. (PGNS)



Training Progressive growth shuffle (PGS):
# Configuration
INITIAL_SIZE = 24 # Initial dataset size
INCREMENT_RATIO = 0.1 # Increment ratio for each step
EPOCHS = 100
BATCH_SIZE = INITIAL_SIZE # Batch size is equal to the initial size
LEARNING_RATE = 1e-4
MODEL_NAME = "EleutherAI/pythia-70m"
Ep 68/100 | Data size: 11536 | Trn Loss: 0.1538 | Tst Loss: 0.3688 | Pplx: 1.1662



Debugging issues, seems the data isn't being loaded in correctly or formatted correctly. Also found that the PGS training may have issues because each loop the Dataloader actually keeps or initializes the entire dataset even if I chose a really small number of items. Meaning that on small data training epochs the overhead is equivalent to the very large ones.

Results above are basically useless. Need to fix before can tell the new data is trainable. Training is going well though. The results are pretty stable and decrease nicely.

Fixed it by reverting to a standard training method.

CR3 training decently, even with high loss and perplexiy it seems to perform well. Highly trained model and slightly trained model seem very similar.

Dolly15k model is having a very hard time giving any real responses. This may be due to some of the data in the set have very long context length.

Overall the CR3 model is training well and I can start integration.

I may just want to switch to the smallest dolly model or a larger pythia model to fix dolly issues. Overall I should be able to continue onto pipeline design.

7-17-24: Thoughts

The above show the fundamentals of solving the embedding resolution problem. These tests show that hyper similar text is able to be embedded and analyzed however you cannot do it on a pretrained method.

There are results that seem to show I can expand data and learn from it even if they are extremely close together.

OpenAI embeddings seems to actually divide the data but you must train on them to find the more detailed parts.

The output embeddings from openai were more expanded results of the same data. This implies that I increased the resolution of the embedding space without destroying it.

I need to reproduce these results one more time in a more convincing way.

My worry is that the expansion of the data space is very similar to what I saw in fine tuning my models in the “no learning” method. Those models space massively expanded into nothing, completely collapsed into nothing. This would imply that there is such a thing as data collapse as well as model collapse.

In a way, aren’t models just data? If they can collapse why can’t data representations?