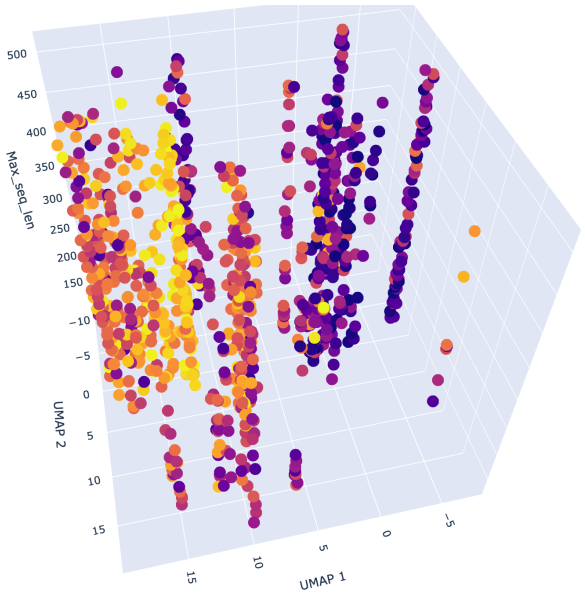
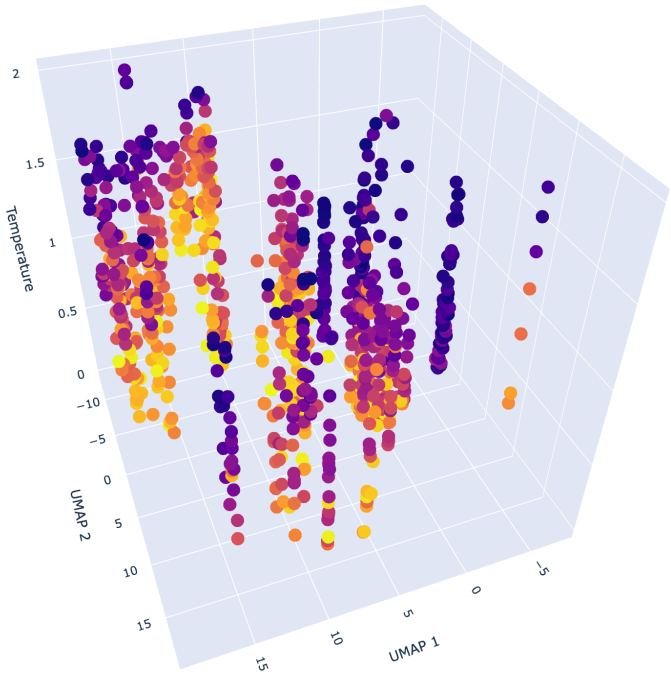
Objective: develop a new way of embedding and visualizing output responses that correlates with top_p and temperature allowing methods to guide generation.

Context:
- bigchat system worked. But the data was seemingly completely uncorrelated to temperature and top_p
- theory is that the data itself does have this correlation but my embedding and current methods are too weak to pull apart the data.
    - I need some metric to optimize toward differentiation. A model predicting top_p and temp may be a metric I can test different embedding types.

6-10-24
- Generated text data: 6-11-24-data
- Umap visuals: 6-11-24-visuals

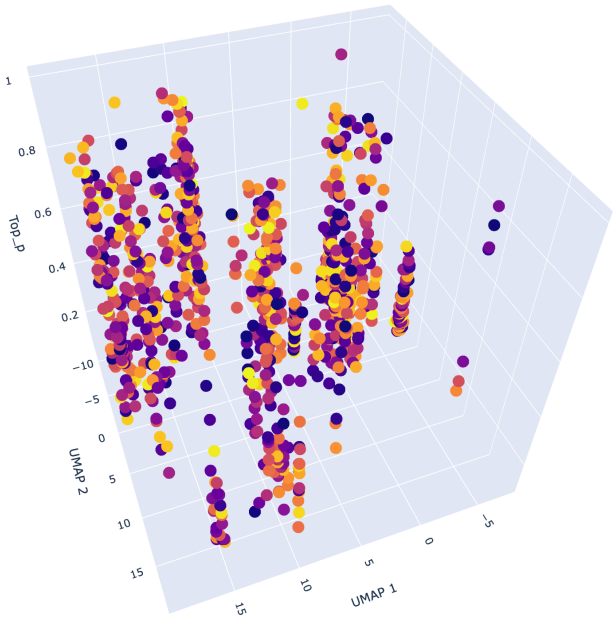


Seq vs temp

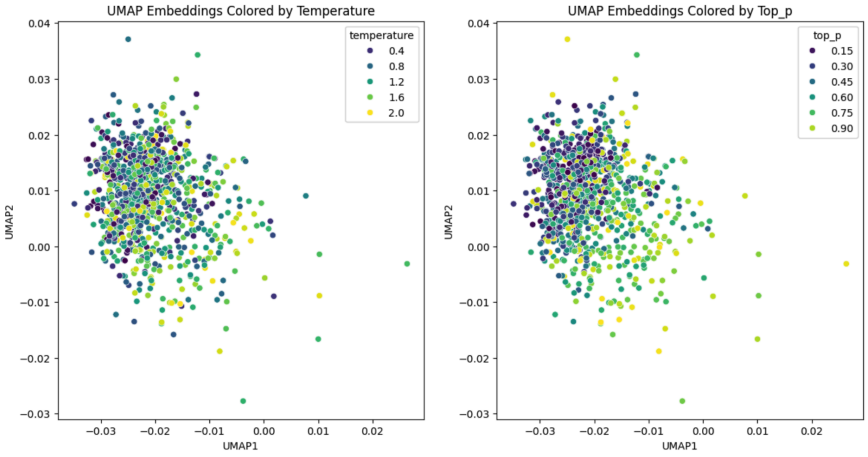


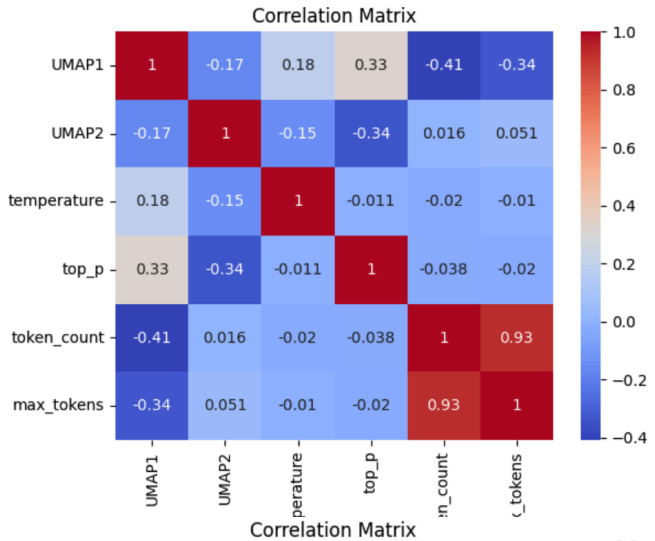
Temp vs top_p

Seq vs Top_p

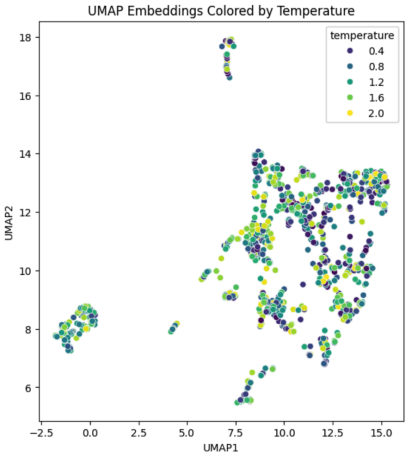


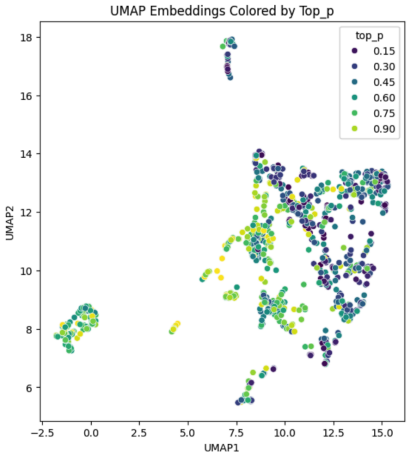
- Initial analysis of embeddings: “openai_confirmation”

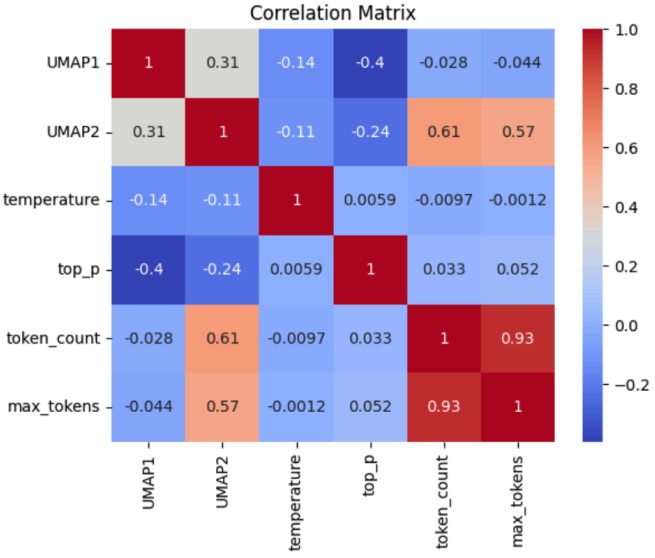




test6_all_timeprompt_gpt3.5_1000

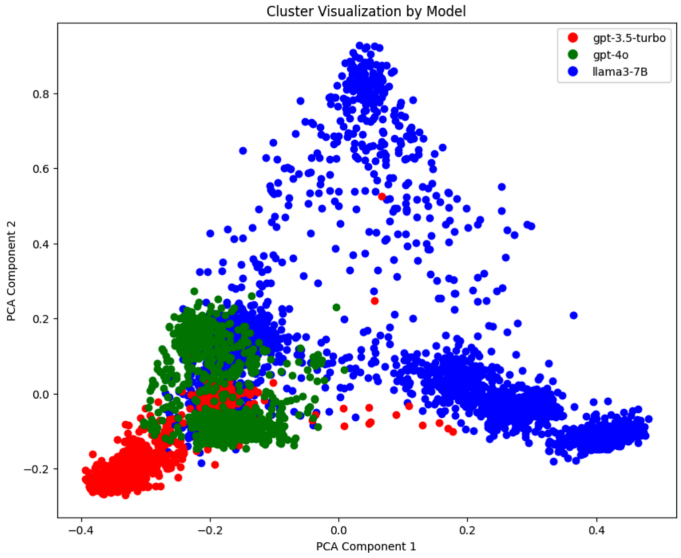


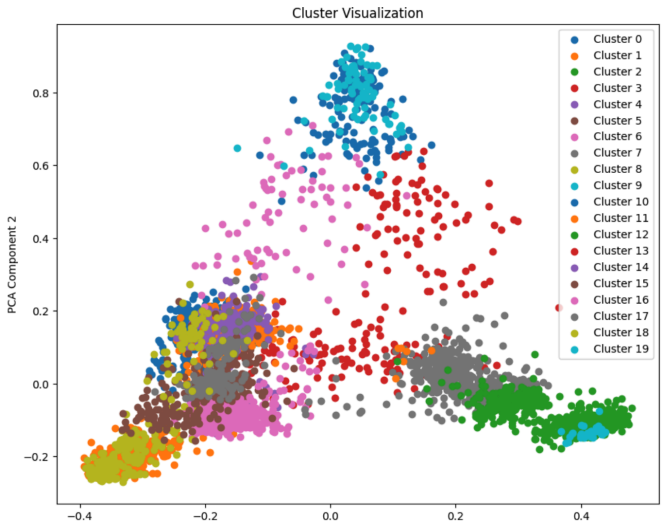




Basically I was under impression temperature and top_p would generate obvious cluster differences. This is not the case in the slightest.

[cluster results]





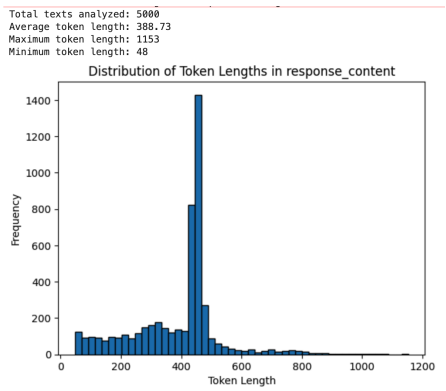
6-11-24

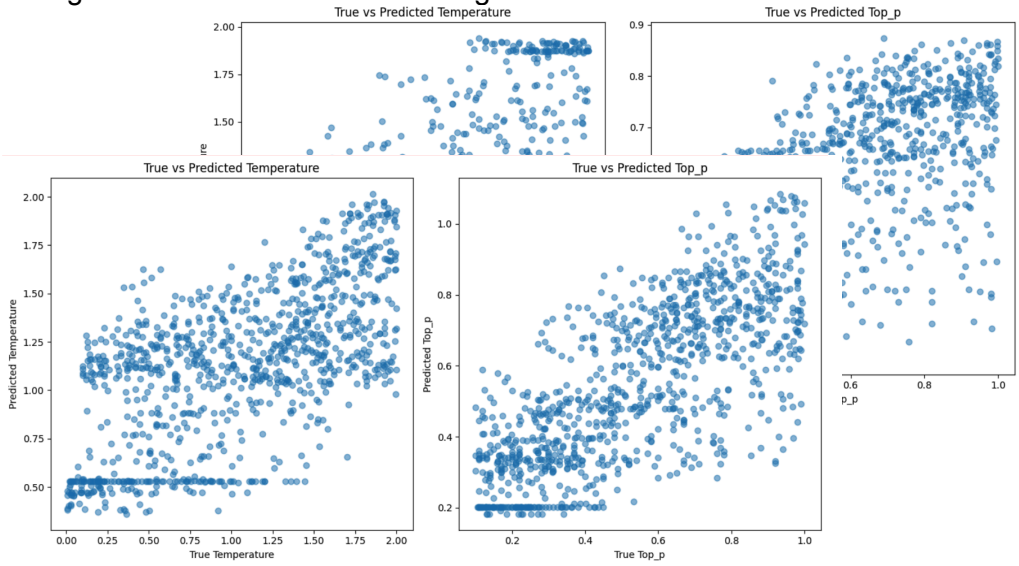
predicting temp, top_p from text

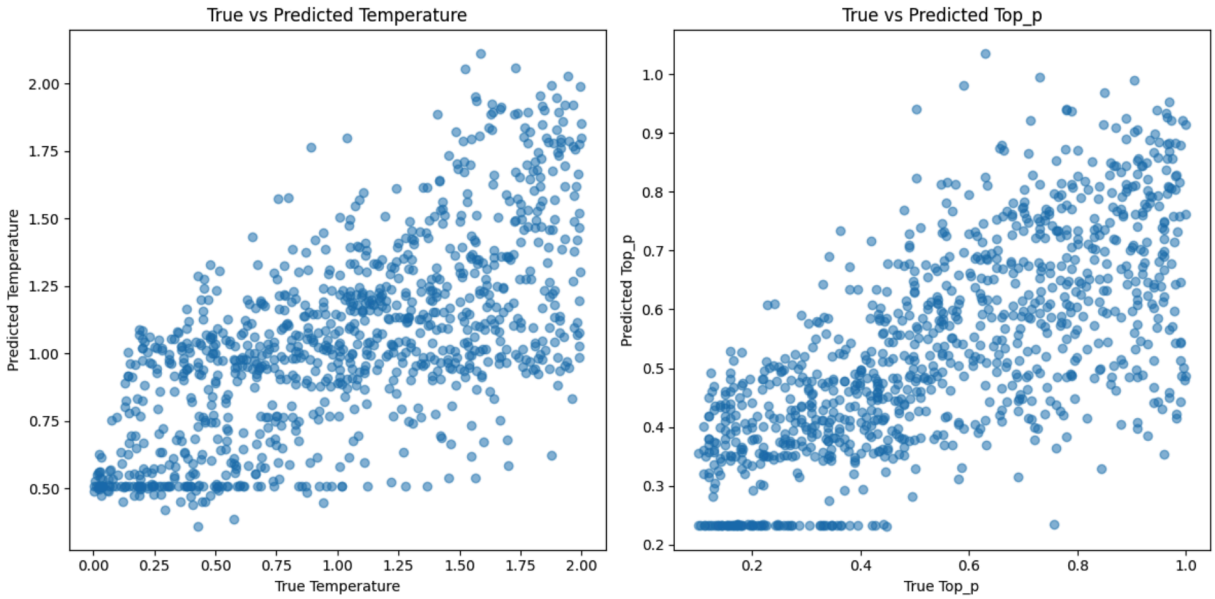
- https://chatgpt.com/c/92238c0d-c70a-4112-ab60-bfcede96633a
- It can softly predict the values given text. There are “columns” of data that are not predicted at all. Perhaps similar to previous results? There are distinctive columns of data that are completely unchanged from temperature or top_p.

I’m using a very small length updating to use full length. Also it is much happier to use batch size around 1 or 2. Even with proper size, still have lots of uncorrelated data validating other results. There are some that are clearly correlated but many are not. Columns still present

Trying again but using the embeddings instead of raw text for training. Same results.



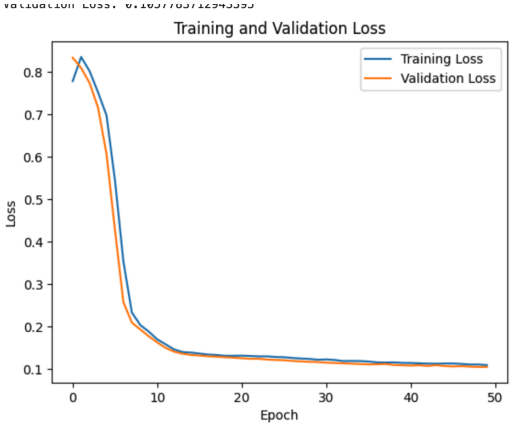


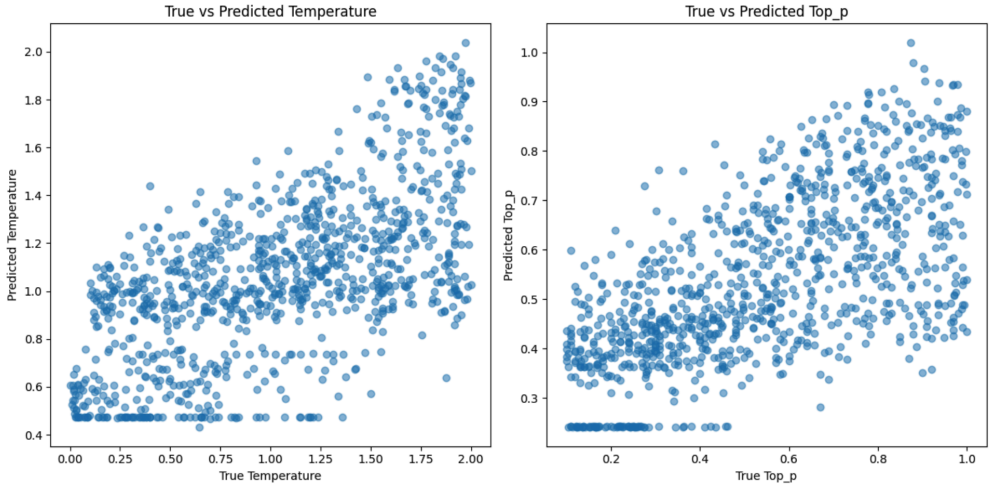


Epoch data size increases per epoch.
- divide data into epoch chunks, per epoch add 1/epochs data to the system. No overfitting ever.

I should do this for the autoencoders or any data I am not Sure what the answer should be to prevent issues.

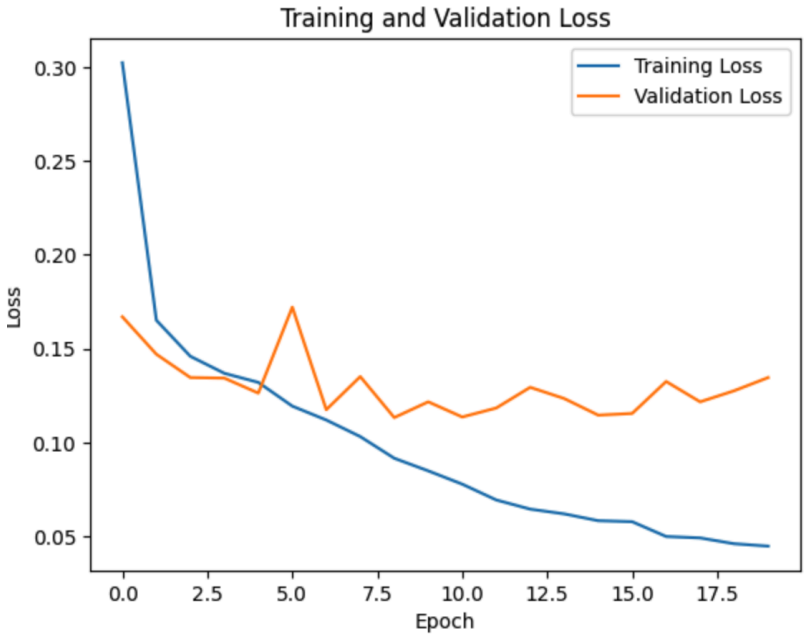
Best result so far.





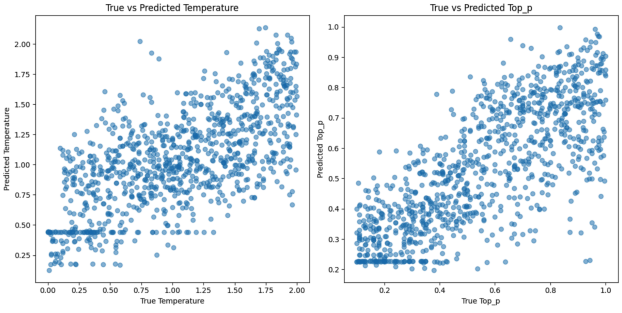
This data increasing method also seems to directly translate and work on the previous text results as well. Again it gives very stable values for decreasing training loss and val loss.

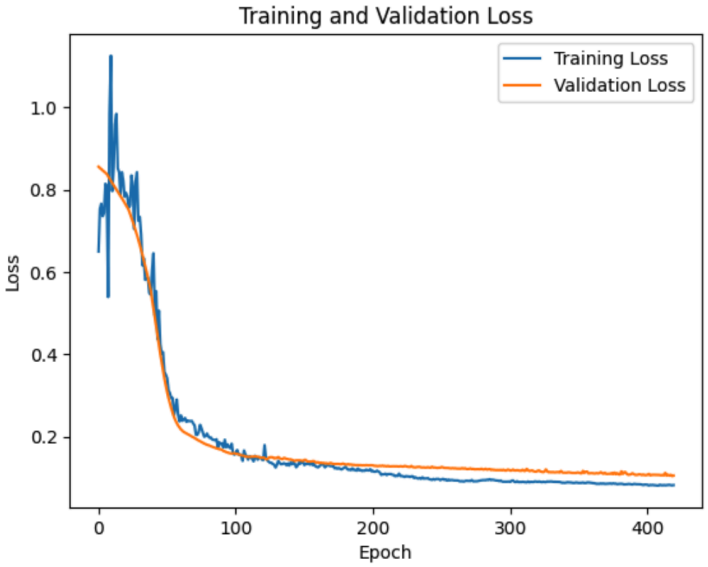
It doesn't seem to overfit?? But memorization continues. Keeps a level of generalization but still memorizes information?



Trying on the embeddings method. Constant ratio instead of diminishing like the above.

Training a model on a 1% increasing training no need to set # epochs let it train itself. Until it runs out of data. No overfitting at all? Memorization without overfitting.





Prompt to get another method like this:

Progressive Data Increment Method for ML Training

Concept Overview:

The Progressive Data Increment Method is designed to train a machine learning model with a gradually increasing dataset. The primary objective is to start with a small subset of the data and incrementally add more data in each subsequent epoch, allowing the model to learn progressively from a growing dataset. This approach can help in scenarios where training on the full dataset from the beginning might be overwhelming or inefficient.

Key Parameters:

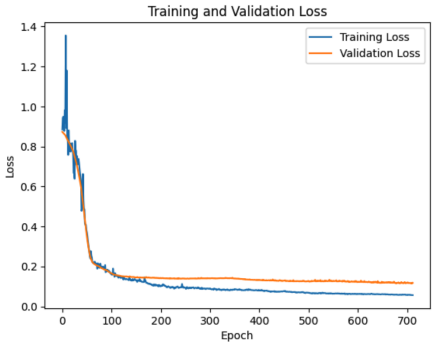
1. **Initial Size**: The number of data samples to start with in the first epoch.
2. **Increment Ratio**: The percentage of the current data size to be added to the training set in each subsequent epoch.
3. **Max Epochs**: The maximum number of epochs for which the training should run.

Procedure:

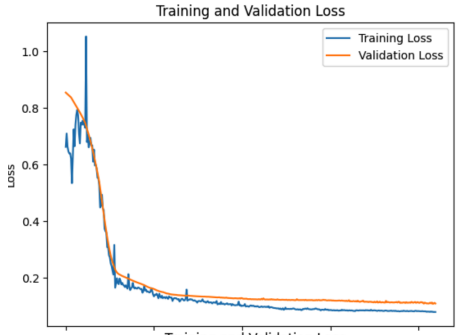
1. **Initialization**:
   - Define the initial size of the training dataset (e.g., 10 samples).
   - Set the increment ratio (e.g., 10%, or 0.1).
   - Determine the maximum number of epochs for the training process.
2. **Epoch Iteration**:
   - Begin training with the initial subset of the dataset.
   - After each epoch, increase the size of the training dataset by a fixed ratio. For instance, if the initial size is 10 samples and the increment ratio is 10%, then in the second epoch, the training dataset will have 11 samples (10 + 1).
   - Continue this incremental increase:
     - Epoch 1: 10 samples
     - Epoch 2: 11 samples
     - Epoch 3: 12 samples
     - And so on...
3. **Data Management**:
   - The training subset for each epoch is a slice of the original dataset, starting from the beginning up to the current data size.
   - This ensures that the model is exposed to an increasing amount of data with each epoch, allowing it to learn from a progressively larger dataset.
   - The data increment continues until either the maximum data size (i.e., the full dataset) is reached or the maximum number of epochs is completed.
4. **Stopping Condition**:
   - If the current data size exceeds the total available dataset size, the training process should be capped at the total data size.
   - Optionally, print messages or logs to indicate the current size of the training dataset and the performance metrics (e.g., training loss, validation loss) at each epoch.

Values from experiments:

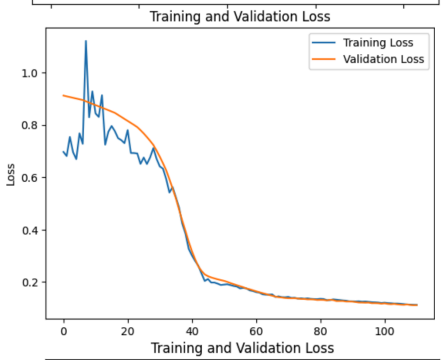
0.5% - Epoch: 712/5000, Train: 0.0564, Val: 0.1176, Epoch size: 3995



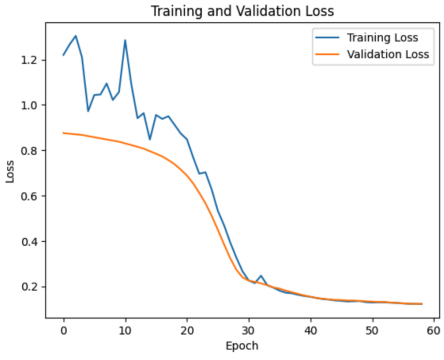
1% - Epoch: 420/5000, Train: 0.0802, Val: 0.1102, Epoch size: 3962



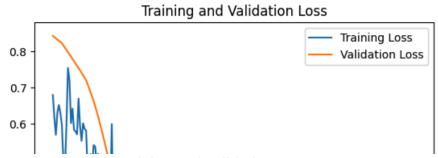
5% - Epoch: 111/5000, Train: 0.1128, Val: 0.1117, Epoch size: 3923



10% - Epoch: 59/5000, Train: 0.1219, Val: 0.1232, Epoch size: 3681

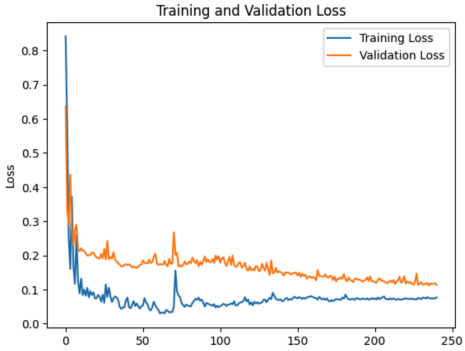


2% - Epoch: 241/5000, Train: 0.0953, Val: 0.1057, Epoch size: 3942



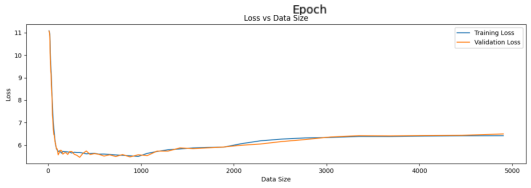
ALL OF THE ABOVE: lr=2e-5

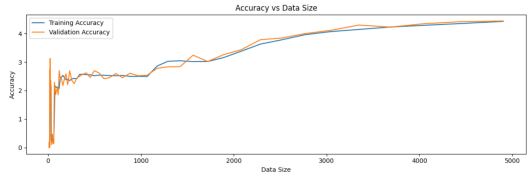
LR - 2e-3
2% - Epoch: 241/5000, Train: 0.0769, Val: 0.1131, Epoch size: 3942
10% - Epoch: 59/5000, Train: 0.1069, Val: 0.1148, Epoch size: 3681



Now trying this method with an autoencoder.

LR 1e–3
10% - Epoch 66/500 | Data size: 5394.077978276343 | Train Loss: 6
4.43% | Val Acc: 4.44%

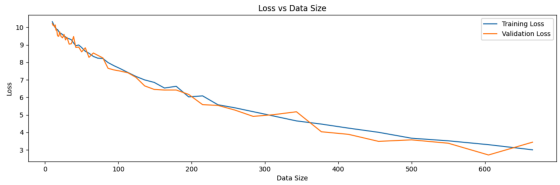


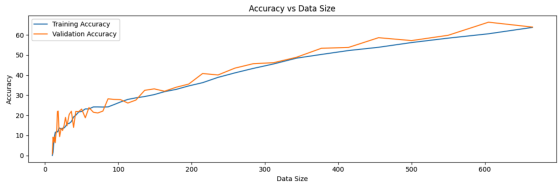


Method works but nowhere near the power to accomplish it.

Updated method to use bert pretrained, giant change:

LR - 1e-4
Bert-base-uncased - BS 32
10% - Epoch 50/50 | Data size: 731 | Train Loss: 2.9996 | Val Loss:
63.91%

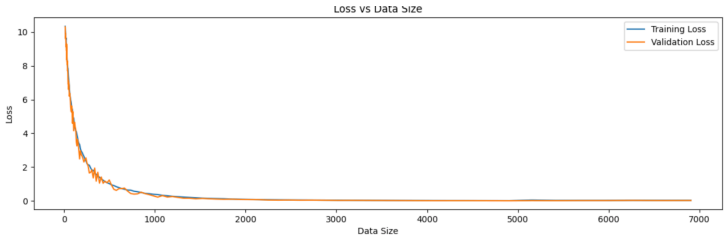


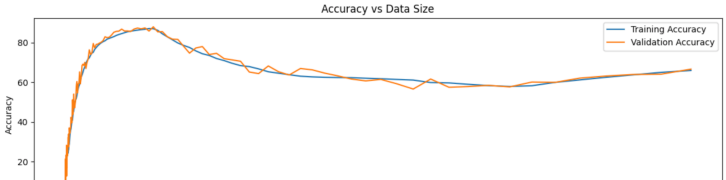


Same but large BS 16
Epoch 50/50 | Data size: 731 | Train Loss: 1.3575 | Val Loss: 1.6153 | Train Acc: 80.07% | Val Acc:
80.89%

Near identical training dynamics
5% - Epoch 135/50000 | Data size: 7253.629591379206 | Train Loss: 0.0319 | Val Loss: 0.0268 | Train
Acc: 65.90% | Val Acc: 66.58%

Decent ability, should try a changing LR





6-12-24 - 6-12-24-data

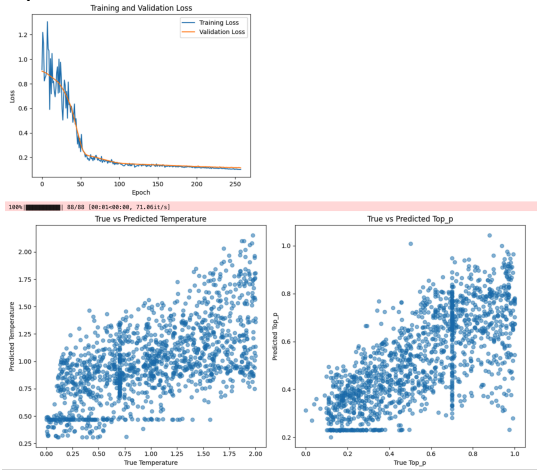
Accuracy decreases and loss is decreasing? Probably val and training dataset are too similar. Increase val datasize and see if same effect exists.

Also could be that it simply hasn't had enough time to train on the increasing size of the data coming in. The above graph shows it is starting to increase again with the larger dataset.

Validation data was mixed with training data every epoch.

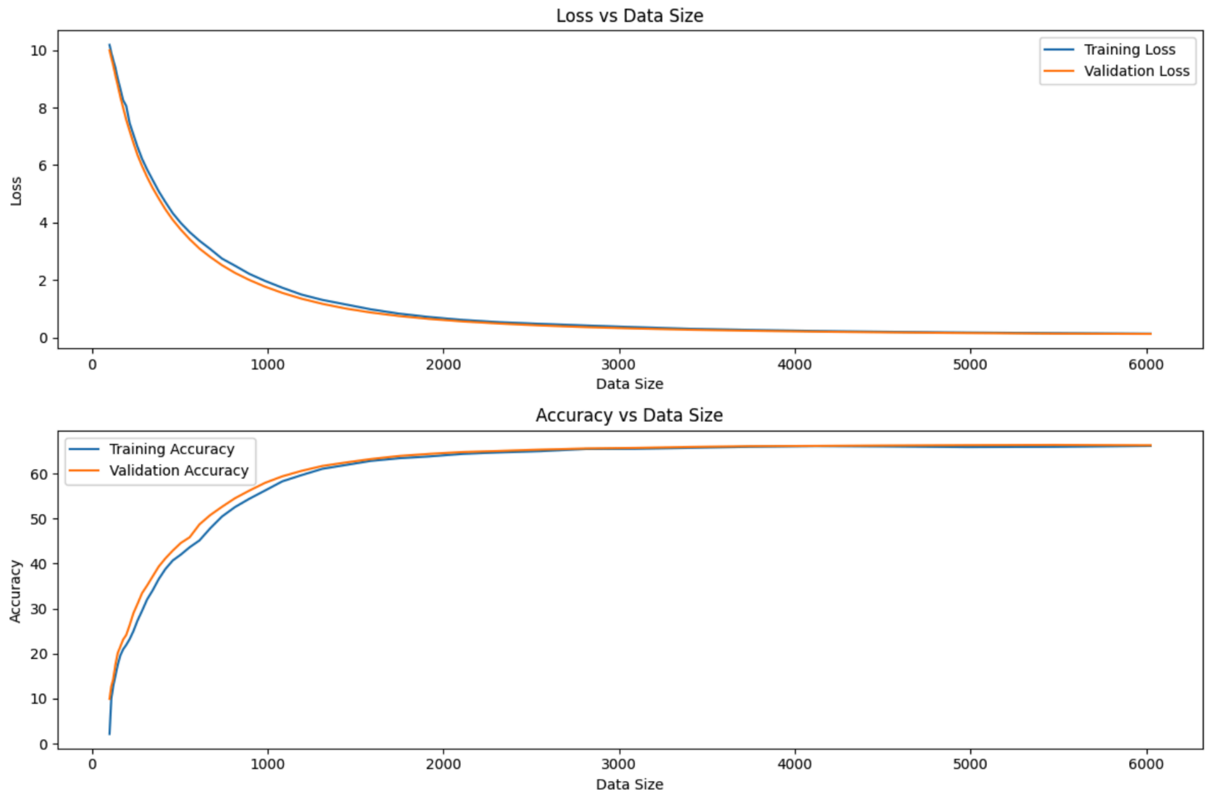
Data was mixed per epoch over all data. Mixing training data per epoch seems to have massive bonus effect for training data. Trying it on the original training method also results in good data.

Mixing per step: Epoch: 258/500, Train: 0.1013, Val: 0.1142, Epoch size: 5532 so mixing per step is equivalent.



Previous method for reference:
Epoch: 258/5000, Train: 0.1013, Val: 0.1099

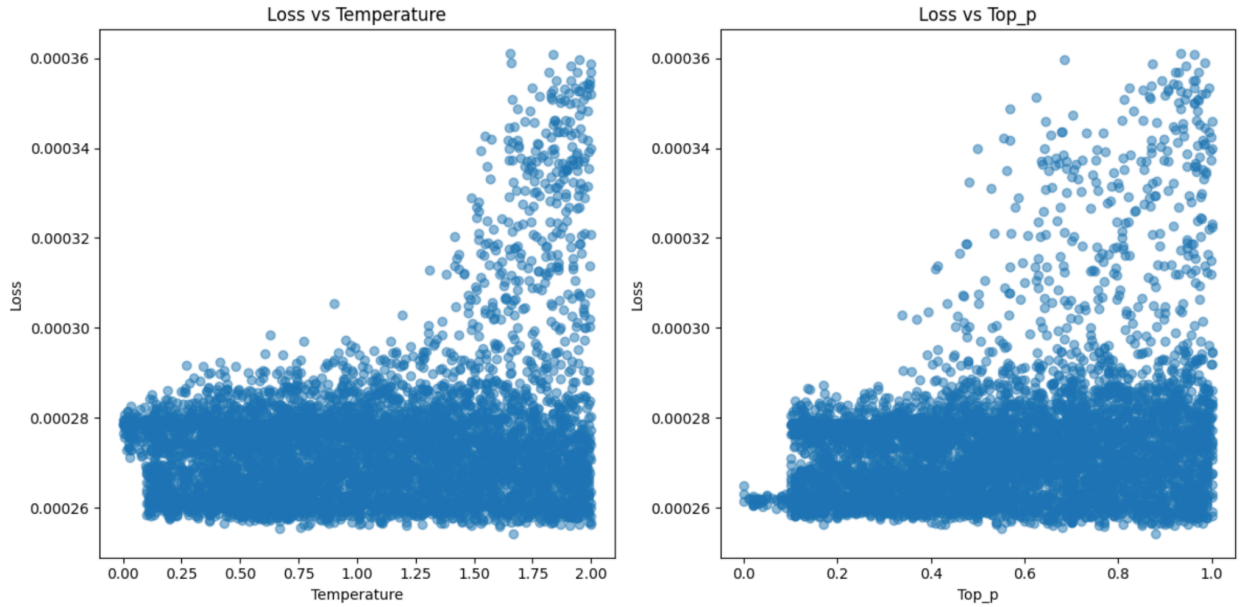
Fixed mixing, seems to behave like previous methods. No weird decrease mid way. Good responses.
Epoch 44/50000 | Data size: 6626.40760773664 | Train Loss: 0.1381 | Val Loss: 0.1300 | Train Acc: 66.16% | Val Acc: 66.30%



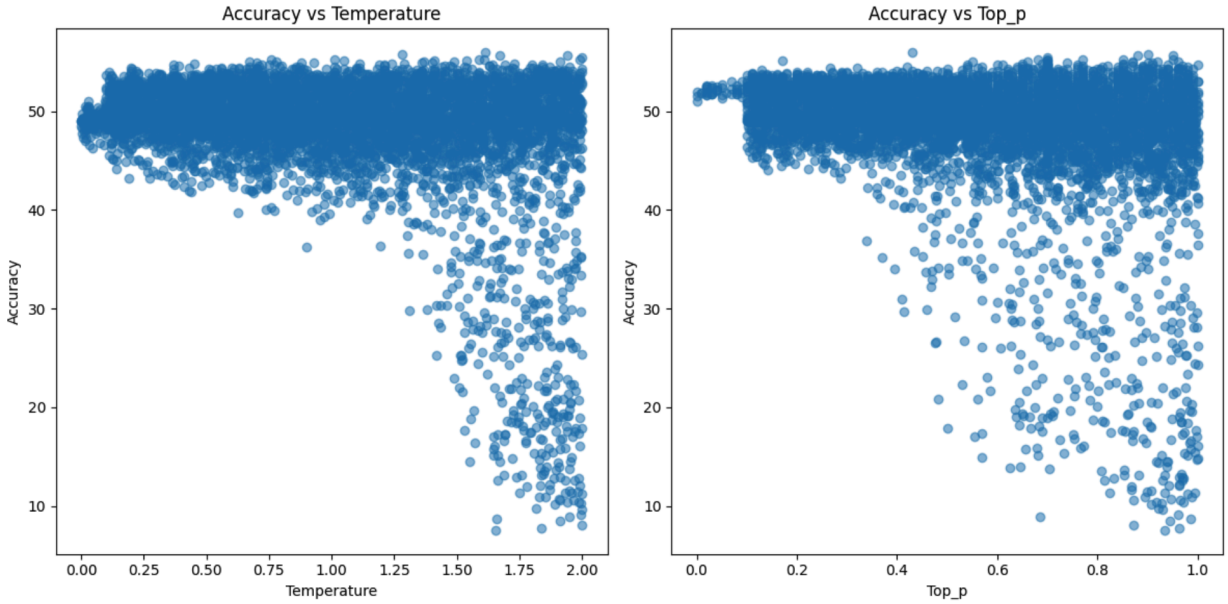
Original: in the year 3000, the most surprising change i notice is in the realm of technology, specifically the advent and widespread adoption of neural interfacing and brain - computer interfaces (bcis). these interfaces allow humans to seamlessly interact with digital systems and each other using thought alone, bypassing traditional input
Predicted: in the year 3000, the most surprising change i notice is in the realm of technology, specifically the advent and widespread adoption of neural interfacing and brain - computer interfaces (bcis). these interfaces allow humans to seamlessly interact with digital systems and each other using thought alone, bypassing traditional input ’ se se ’ ’ ’ se se ’ ’ se se ’ ’ ’ ’ ’ ’ ’ ’ se se ’ ’ㄴ se — ’ ’ ’ ’ ’ ’ ’ ’ ’ ’ ’ ’ ’ se ’ ’ ’ ’ ’ ’ se se se ’ seㄴ ㄴ se se ’ ’ — se se — ’ec ’ se — — se se ’ ’ ’ ’ — — ’ — — — seㄴ ’ ’ se se se — ’ㄴ — — ’ ’ — ’ — ’ se ’ se ’ se se ’ ’ ’ ’ ’ ’ ’ ’ ’ ’ ’ ’’ec ’ ’ㄴ ’ ’ — ’ ’ ’ se — ’ — — se — ’ ’ ’ se ’ ’ ’ ’ ’ ’ se ’ ’ ’ ’ ’ — ’ ’ ’ ’ ’ ’ se se ’ ’ se se — — ’ — se — ’ ’ ” ’ — se ’ ’ec se ’ — — ’ ’ se ’ ’ ’ ’ ’ — ’ ’ se ’ ’ ’ ’ ’ ’ ’ ’ se — — — se — ’ ’ ’ ’ — —ec se se ’ — — se — — — se — — ’ ’ — — — — — — — — — — ’ — ’ — — ’ — — ’ ’ ’ ’ — — — — — ”のㄴ se — ’ seㄴ ’ ’ ’ — — se ’ ’ — ’ ’ ’ ’ — se — — se ’ se ’ — — ’ — — — ’ — — — — — — ’ — — — ’ — —の ’ ’ — — ’ ’ ’ ’ ’ ’ ’ — ’ — ’ — ’ ’ — ’ ’ㄴ ’ ’ ’ ’ ’ ’ — — ’ ’ ’ se ’ — ’ ’.

Using embedding for the same methods above instead of text:
10% - 1e-4 - Epoch 50/50 | Data size: 731 | Train Loss: 0.0002 | Val Loss: 0.0002

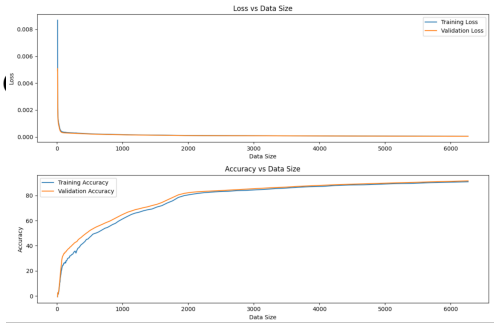
Intial run with warning. Unrelated for majority of data but there are some that are, and they seem to have the higher values. Is that the junk data generation?

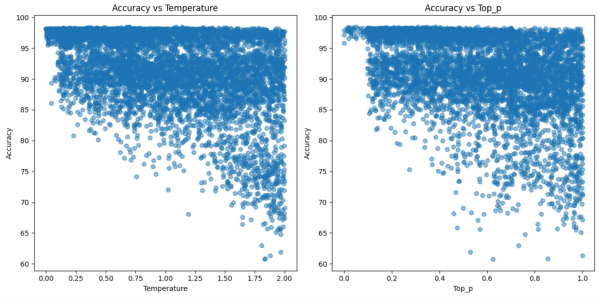


Loss seems very different from accuracy in this method:
Epoch 50/50 | Data size: 731 | Train Loss: 0.0003 | Val Loss: 0.0002 | Train Acc: 41.38% | Val Acc: 48.42%
Accuracy paints the same picture as befor a subset of data that is not trainable that is high in temp and top_p again most likely the failed data.

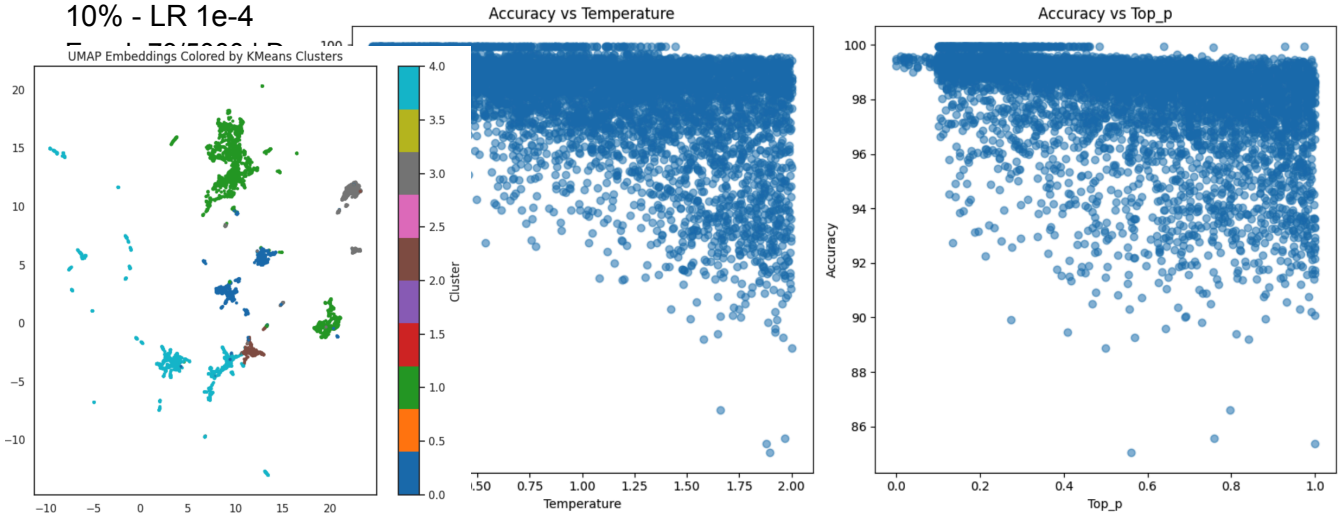


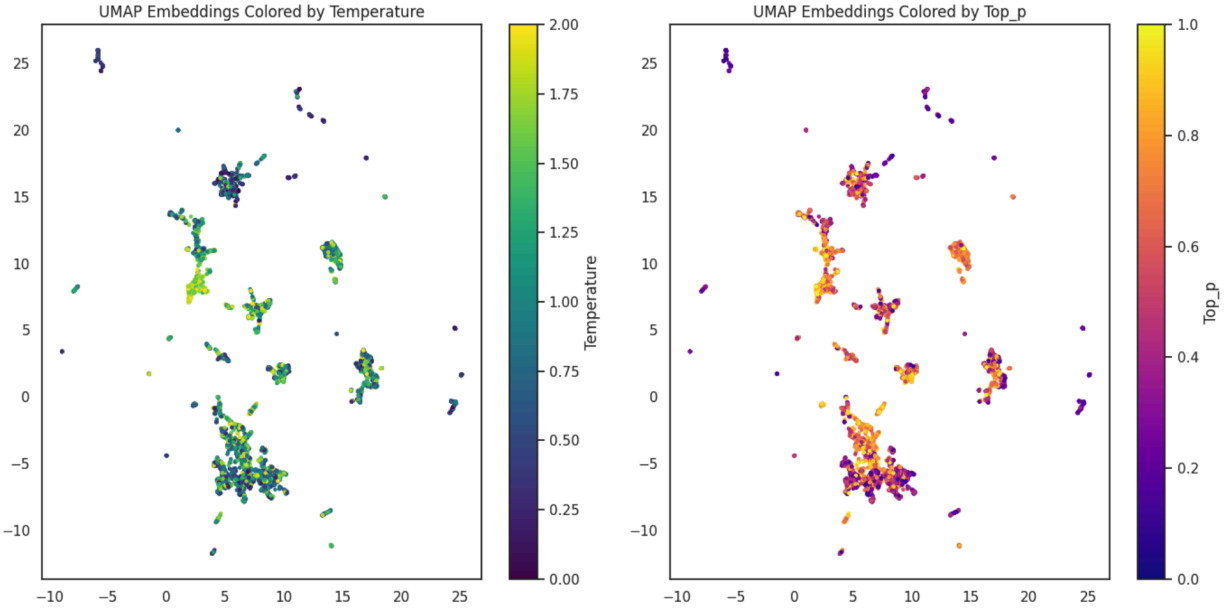
Trains very fast and has very high accuracy
Epoch 133/5000 | Data size: 6579.255865196559 | Train Loss: 5.017081925221386e-05 | Val Loss: 4.629895110238894e-05 | Train Acc: 90.97% | Val Acc: 91.70%





10% - LR 1e-4





Conclusions:
Data is still confusing. Make the analysis easier, add more steps and perform more analysis.

- do the same analysis on a multi step problem
- Add the best analysis to bigchat
- Prompt with testable output (code without error, formatted text)
- Add new autoencoder system

7-17-24: Thoughts on the above

This work foundational to the concepts of filtering and analyzing text data that is very similar. This worked continued in greater depth in the AE embedding document. The results were that I was properly able to pull apart the data into clusters that I have increased resolution in.

However the issue of text not clustering with temp or top_p was never solved. This seems to be a fundamental issue that they are seemingly uncorrelated. Only time they do correlate is under complete model collapsing output (junk output) at very high values.

I wonder how insanely good of a model I could train if I combined the mixed data training with LR searcher?