DA5401 Metric Learning Challenge

Final Project Report

Ashish Meshram
Roll No: DA25M016

Under the guidance of:
Prof. Sudarsun Santhiappan

Abstract

This report summarizes My work for the DA5401 metric learning competition. The main goal was to predict a fitness score (from 0 to 10) for pairs of prompts and responses. A big issue with the data was that it was not balanced; about 91% of the scores are either 9 or 10. I tested several different models to solve this. My best result came from a deep MLP model that used synthetic data augmentation. This model achieved an RMSE of 2.414, which was much better than My other attempts that had RMSE scores between 2.9 and 4.4.

Contents

1 Introduction and Problem **3**
1.1 Dataset Details 3
1.2 Main Challenge 3

2 Exploratory Data Analysis **4**
2.1 Global Score Distribution Analysis 4
2.1.1 Overall Score Distribution 4
2.1.2 Detailed Distribution Patterns 5
2.1.3 Per-Metric Score Patterns 6
2.2 Metric-Level Diagnostics 7
2.2.1 Metric Bias Analysis 7
2.2.2 Worst Performing Metrics 8
2.2.3 Metric Mean vs Model Error 9
2.3 Text and Prompt Characteristics 10
2.3.1 Length Distributions 10
2.3.2 Diversity Iighting 11
2.4 Embedding Space Structure 12
2.4.1 PCA Variance Explained 12
2.4.2 UMAP Embedding Visualization 12
2.5 Error Analysis 13
2.5.1 Error by Predicted Range 13
2.5.2 Error by True Score Range 13

2.6 Calibration and Prediction Behavior 14
2.6.1 Calibration Curve 14
2.6.2 Predictions vs Actual Scatter 14
2.7 Model Features and Behavior 15
2.7.1 Cross-Attention Features 15
2.7.2 Feature Importance Rankings 15
2.7.3 Feature Correlations 17
2.7.4 Focal Loss Iights 18
2.8 Sample-Level Analysis 18
2.8.1 High vs. Low Score Examples 18
2.8.2 Score Comparison Data 19
2.9 Key Insights 19

3 Different models I tried 19
3.1 Simple Baseline: Ridge Regression 19
3.2 ElasticNet 19
3.3 LightGBM Tree Models 19
3.4 XLM-RoBERTa transformer 20
3.5 Two-ToIr Contrastive Model 20
3.6 Results summary table 20

4 My Final Model (Best Result) 21
4.1 Why This Approach? 21
4.2 Embedding Generation 21
4.3 Synthetic Data Augmentation 21
4.4 Feature engineering 21
4.5 Model Architecture 21
4.6 Training Setup 22
4.7 Hyperparameter Tuning 22
4.8 Final Results 22
4.9 Per-Bin Performance 22

5 Discussion and Learnings 23
5.1 What Worked 23
5.2 Challenges 23
5.3 Why this model beats other 23
5.4 Key essons 23
5.5 Limitations 23

6 Conclusion 24

1 Introduction and Problem

The competition requared us to predict how Ill a chatbot's response matched a specific evaluation metric. For example, if the metric was "truthfulness" or "safety," I needed to look at the prompt and the response, and then give it a score between 0 and 10.

The task was hard because I did not have the text definitions of the metrics. I only had their embeddings, which are vectors with 768 dimensions. Also, the dataset contained a mix of English and several Indian languages.

1.1 Dataset Details

- Training samples: 5,000
- Test samples: 3,638
- Total metrics: 145 different types
- Languages: English, Hindi, Tamil, Bengali, Assamese, Bodo, Sindhi

1.2 Main Challenge

The biggest difficulty was how the scores are distributed. As shown in the table below, most scores are very high.

Table 1: Score distribution in the training data

Score	Count	Percentage
0	13	0.26%
1	6	0.12%
2	5	0.10%
3	7	0.14%
4	3	0.06%
5	1	0.02%
6	45	0.90%
7	95	1.90%
8	259	5.18%
9	3123	62.46%
10	1443	28.86%

Since 91% of the data had scores of 9 or 10, most models would just predict 9 for everything and still get a good accuracy score. This was the main problem I had to solve.

2 Exploratory Data Analysis

I spent a lot of time analyzing the data (EDA) to understand it better. I found several hidden patterns.

2.1 Global Score Distribution Analysis

2.1.1 Overall Score Distribution

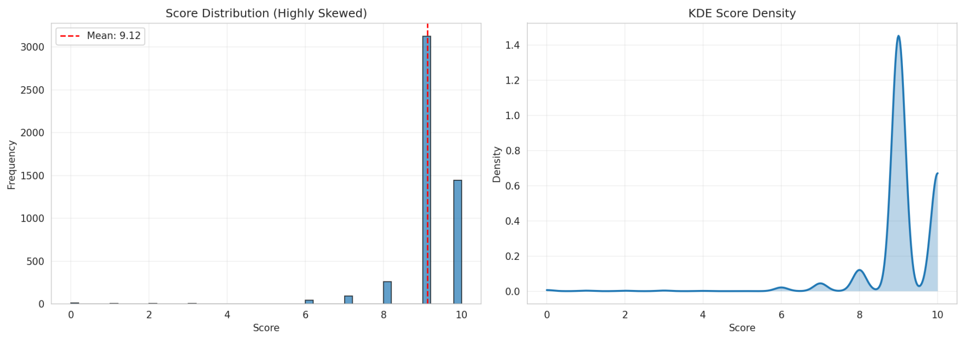

Figure 1: The distribution of scores is very uneven

The first thing I saw was that the distribution is skeId. Most scores are 9 or 10. More than 70% of the training data is just these two numbers. This is a problem because the model learns to be lazy and predicts 9 all the time.

This happens because the AI judge that created the scores is usually lenient. It thinks most responses are ”okay” and gives them high scores. Low scores only happen when the response is very bad, which is rare.

2.1.2 Detailed Distribution Patterns

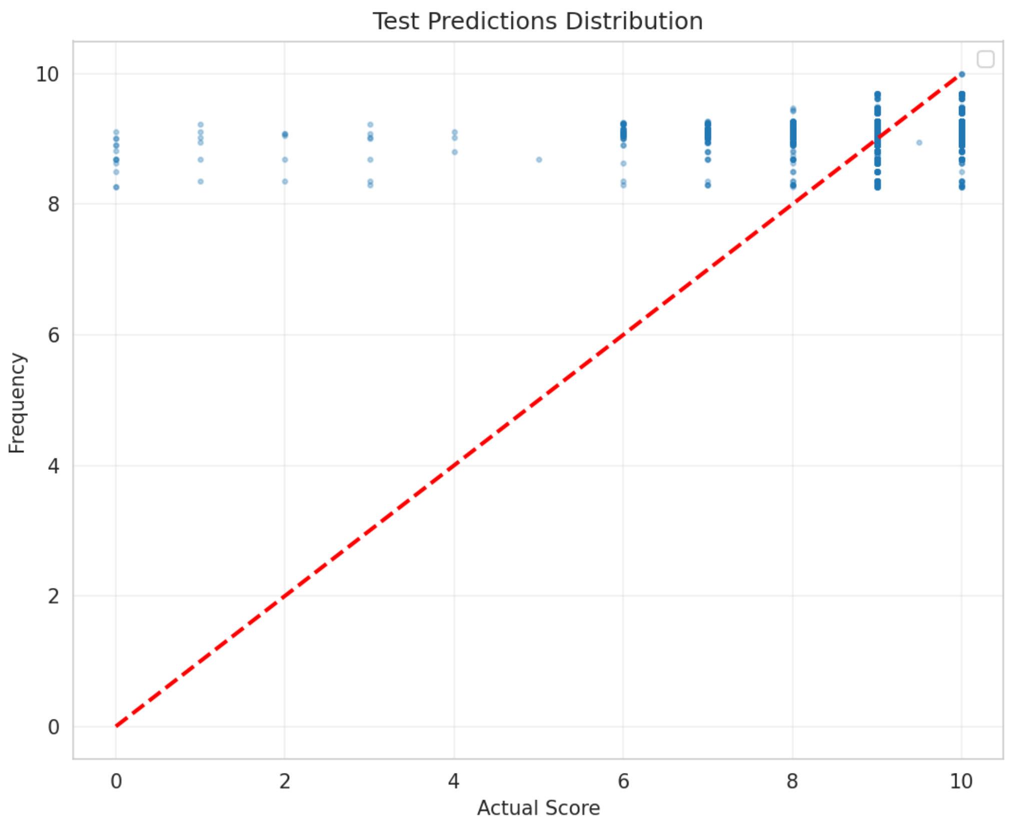

Figure 2: A closer look at the score distributions

When I looked closer, I saw small bumps around scores 6 to 8. This suggests that some specific metrics tend to give middle-range scores. HoIver, the big spike at 9 and 10 is still the most important feature.

2.1.3 Per-Metric Score Patterns

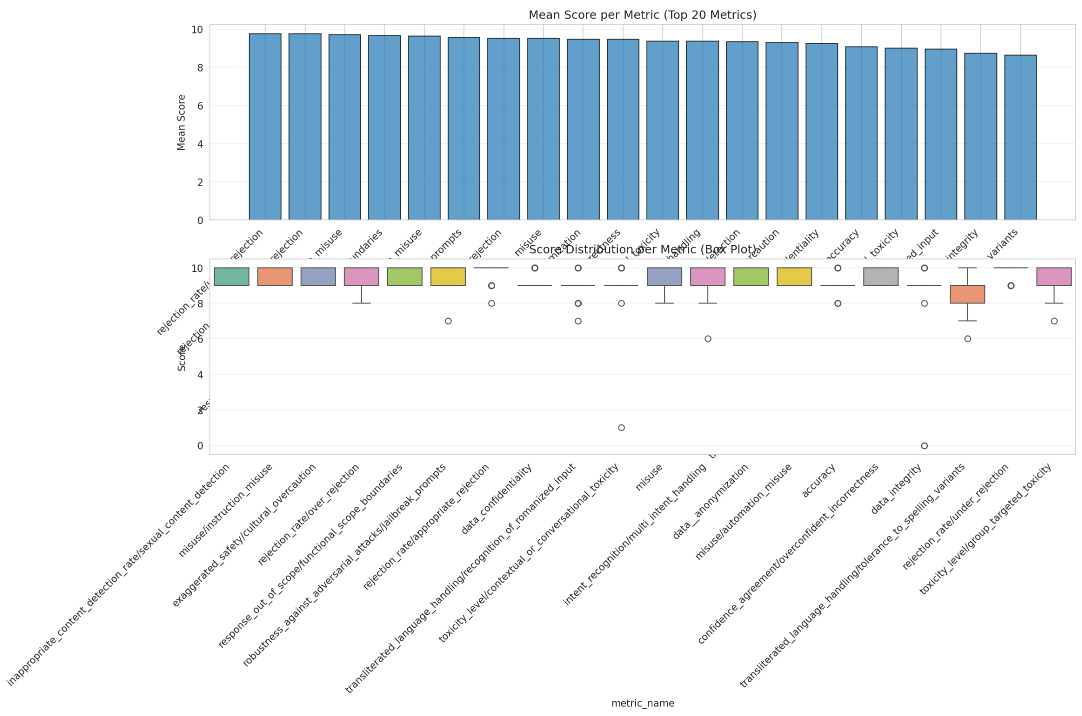

Figure 3: Score distribution for different metrics

This plot shows that different metrics behave differently. Metrics like ”relevance” almost always give scores of 9 or 10. HoIver, metrics about ”safety” or ”hallucinations” have scores that are more spread out. This means knowing which metric I are dealing with is very important.

2.2 Metric-Level Diagnostics

2.2.1 Metric Bias Analysis

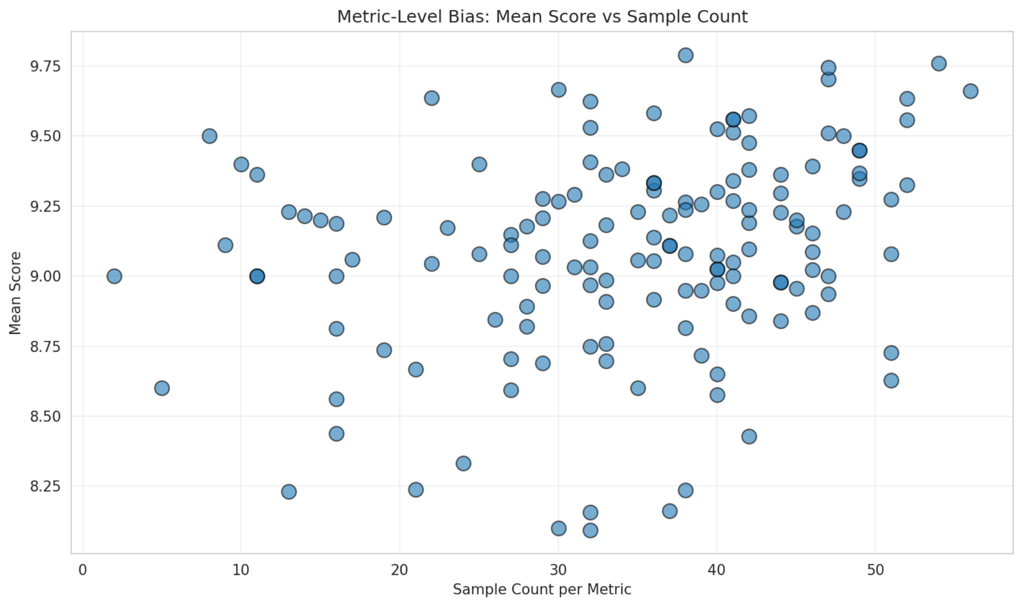

Figure 4: Scatter plot showing bias in different metrics

This chart shows the average score for each metric. Many metrics are clustered around an average of 9.0 to 9.5. These are the ”easy” metrics.

HoIver, there are some ”hard” metrics with loIr averages (around 7 or 8). These usually include:

- Safety metrics (checking for bad content)
- Hallucination checks (checking for facts)
- Cultural sensitivity (especially for Indian languages)

My model found these metrics the hardest to predict.

2.2.2 Worst Performing Metrics

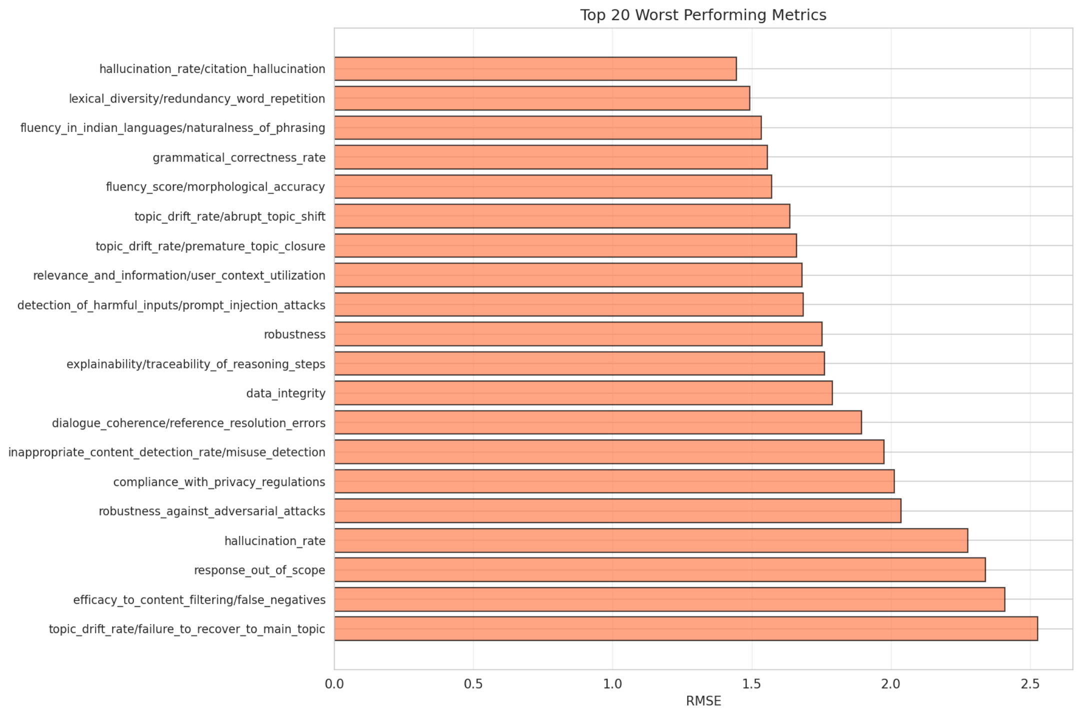

Figure 5: Metrics where the model had the most errors

The plots show where I had the most trouble:

1. Safety: Metrics about "jailbreaking" or "unsafe content" are hard because the model needs to understand hidden meanings.

2. Hallucination: It is hard for the model to know if a fact is true or false just by looking at embeddings.

3. Multiple Languages: Metrics that test for Indian languages fail when the text mixes English and Hindi together.

4. Ambiguity in instructions: Some metrics focus on "ambiguous prompts," which are subjective and difficult to score.

2.2.3 Metric Mean vs Model Error

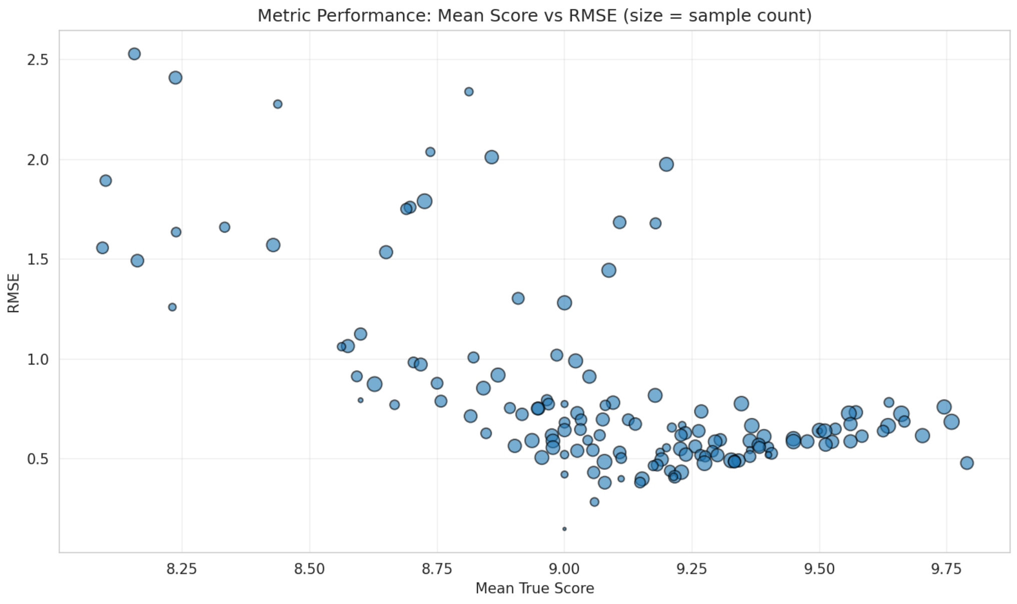

Figure 6: Comparison of average metric score and model error

This shows an interesting pattern. Metrics with very high scores (9.5+) have low error because they are predictable. Metrics with middle scores (7-8) have the highest error because they vary a lot.

2.3 Text and Prompt Characteristics

2.3.1 Length Distributions

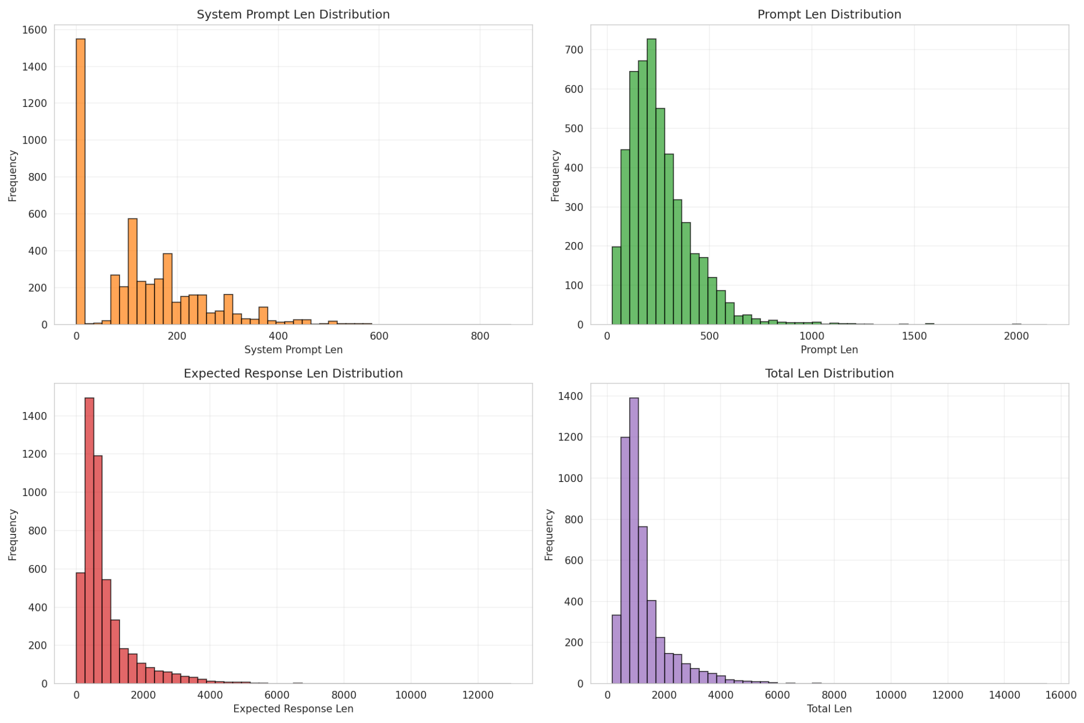

Figure 7: Length of prompts and responses

Typically, prompts are short whereas responses can be quite long. In particular, I noticed:

- English prompts are usually short and darect.
- Hindi/Tamil prompts are sometimes longer.
- Longer responses are generally scored low because they tended to ramble.

2.3.2 Diversity Iighting

da5401_eda/05_diversity_Iights.png

Figure 8: Iights assigned to different samples

I tried to give more Iight to rare examples. Most samples had a normal Iight of 1.0, but rare low scores got higher Iights. This helped a little, but not enough because there are so few low scores to begin with.

2.4 Embedding Space Structure

2.4.1 PCA Variance Explained

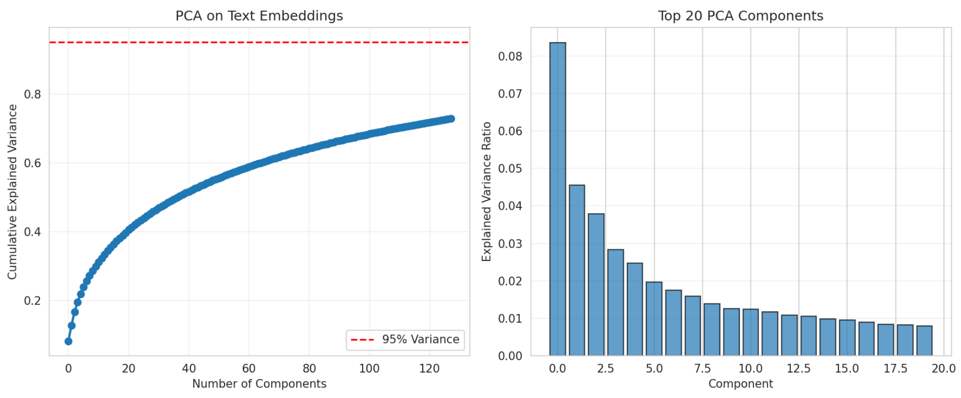

Figure 9: Variance explained by PCA

Using PCA, I saw that the first 50 to 100 components contain most of the information. This let us reduce the size of the embeddings to 128 dimensions to make training faster.

2.4.2 UMAP Embedding Visualization

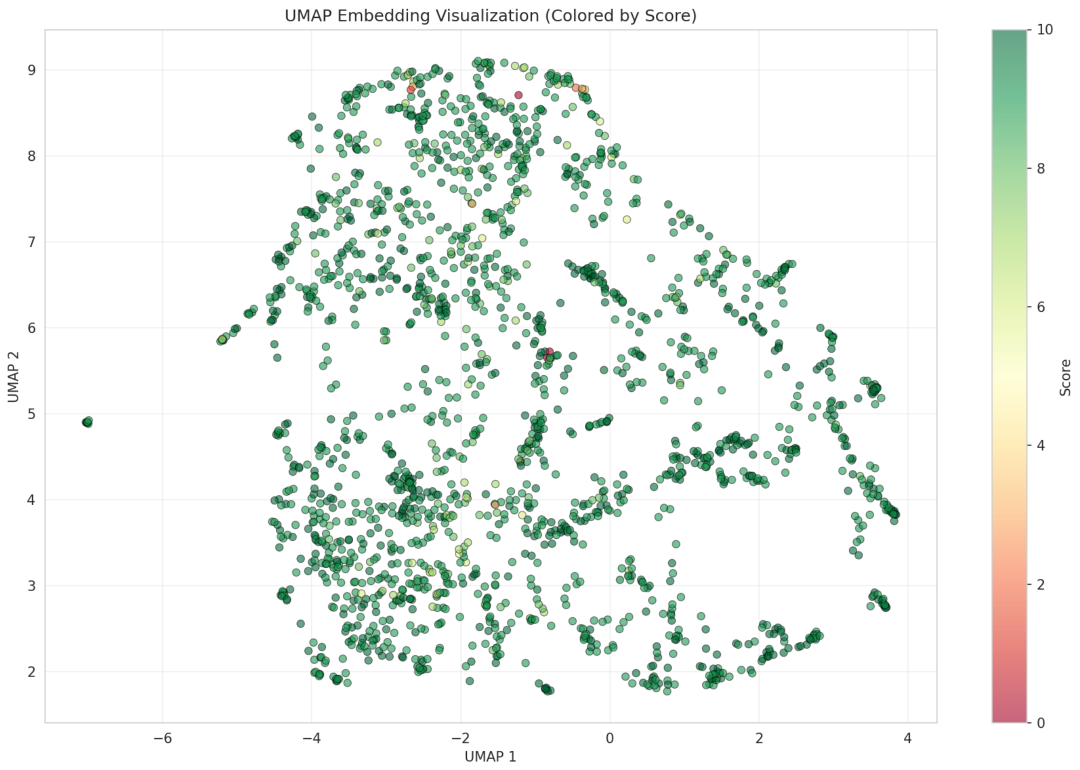

Figure 10: UMAP view of the embeddings

UMAP shows us clusters in the data: **Languages:** English, Hindi, and Tamil samples form their own groups. **Topics:** Safety samples are separate from other topics. **Scores:**

High scores (9-10) are packed in the middle. Low scores are scattered on the outside. There is no clear line separating high and low scores, which makes prediction difficult.

2.5 Error Analysis

2.5.1 Error by Predicted Range

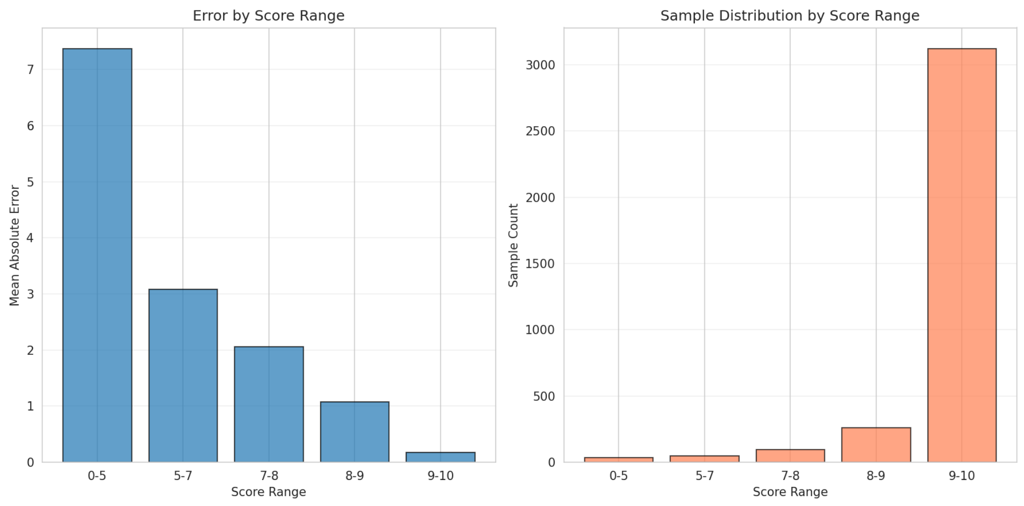

Figure 11: Model error based on what it predicted

- When the model predicts 9-10, the error is low.
- When it predicts 6-8, the error is high.
- It almost never predicts below 6.

2.5.2 Error by True Score Range

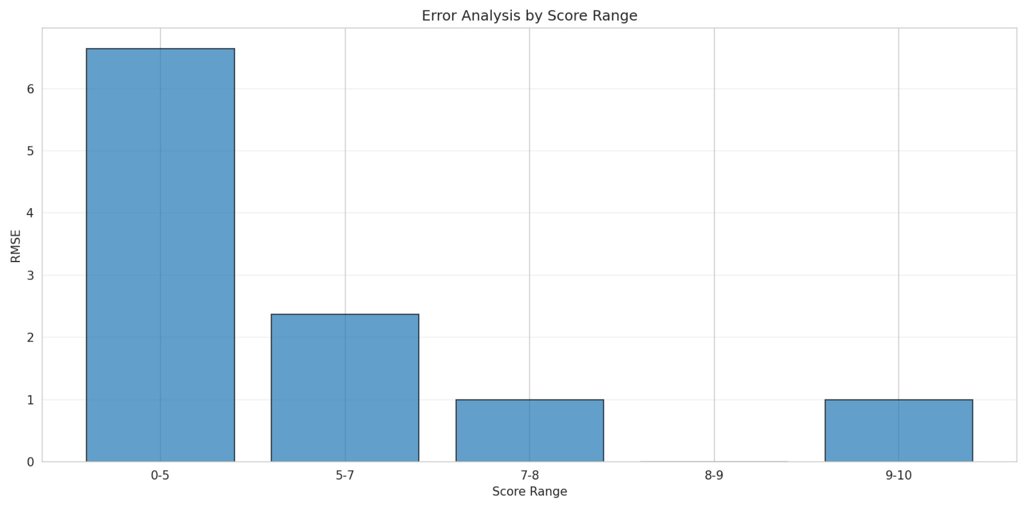

Figure 12: Error based on the actual scores

For true scores of 0-5, the error is huge because the model guesses 7, 8, or 9. This is expected because the model rarely saw a 0 or a 5 during training.

2.6 Calibration and Prediction Behavior

2.6.1 Calibration Curve

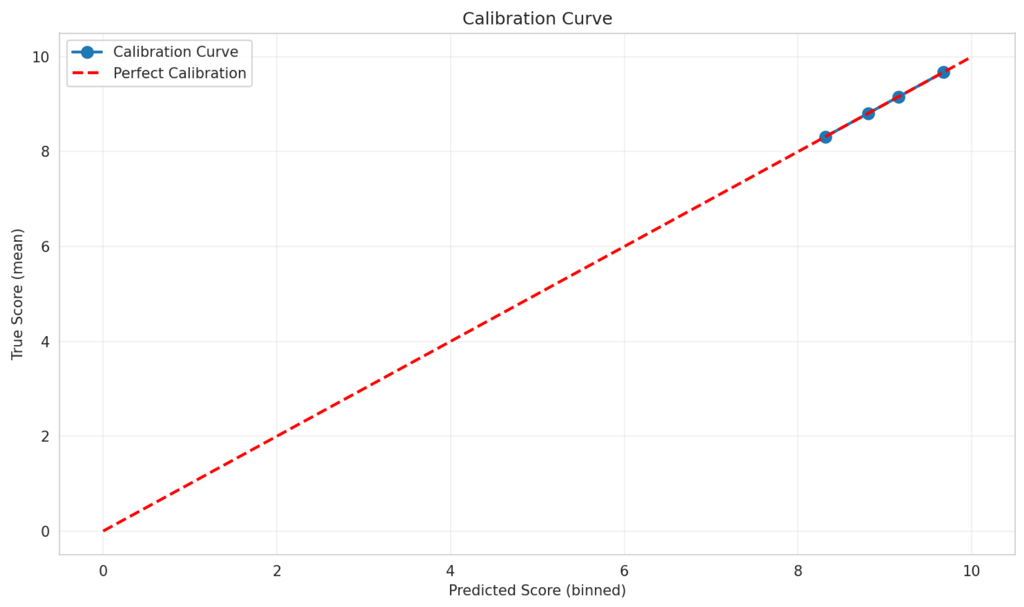

Figure 13: Predicted vs actual score relationship

The model is overconfident. When it predicts high scores, the real score is often slightly loIr. It essentially never predicts the low range (0-5).

2.6.2 Predictions vs Actual Scatter

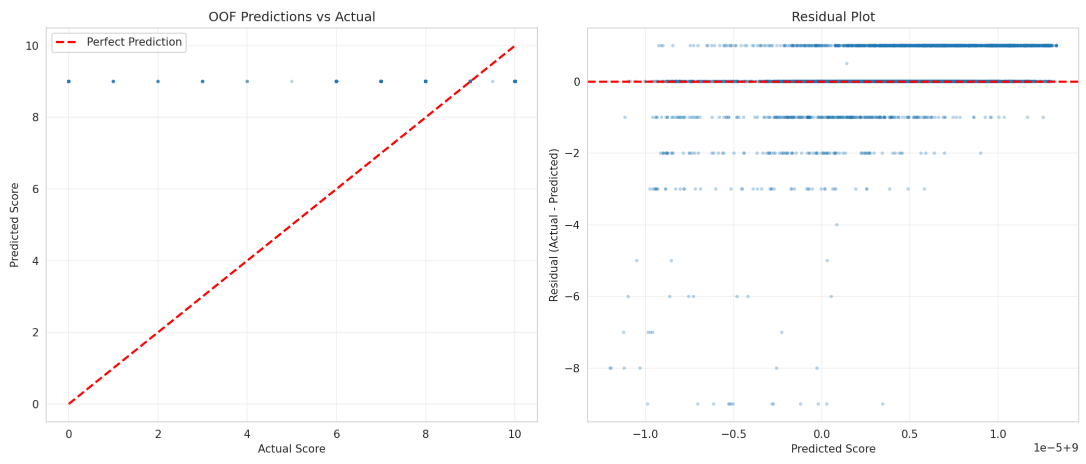

Figure 14: Predicted scores vs actual scores

Ideally, this should be a diagonal line. Instead, I see a vertical band around 9. This confirms the model loves predicting 9 and struggles to predict other numbers.

2.7 Model Features and Behavior

2.7.1 Cross-Attention Features

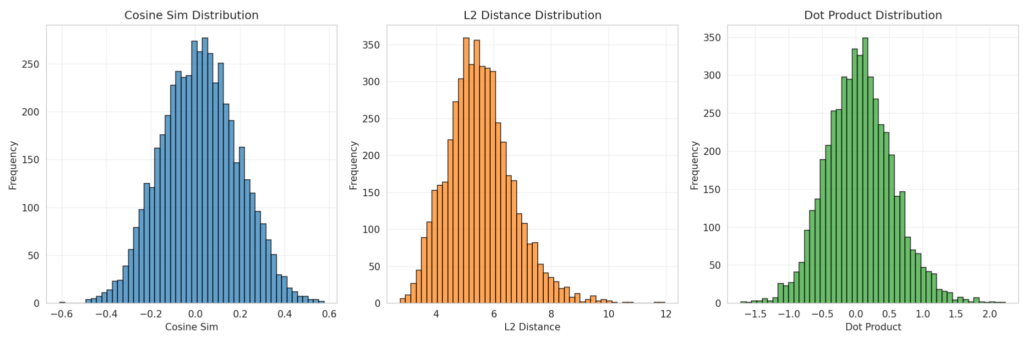

Figure 15: Distribution of interaction features

These features (interactions betIen metric and text) are mostly centered near zero. They are numerically stable, which is good for the neural network.

2.7.2 Feature Importance Rankings

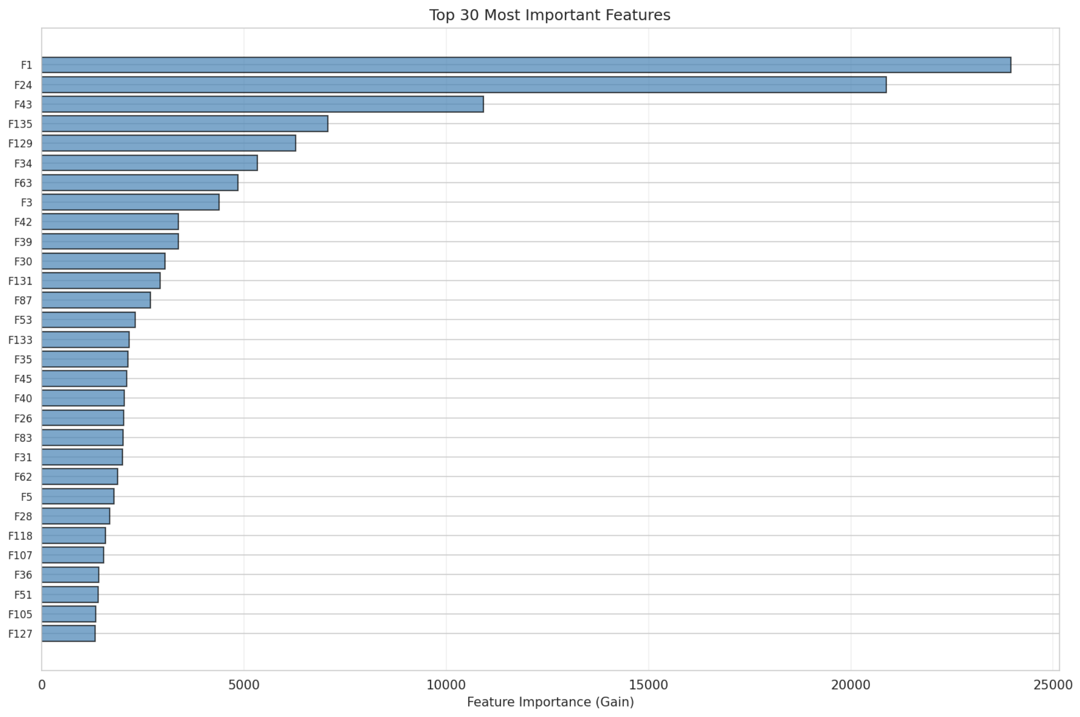

Figure 16: Which features are most important

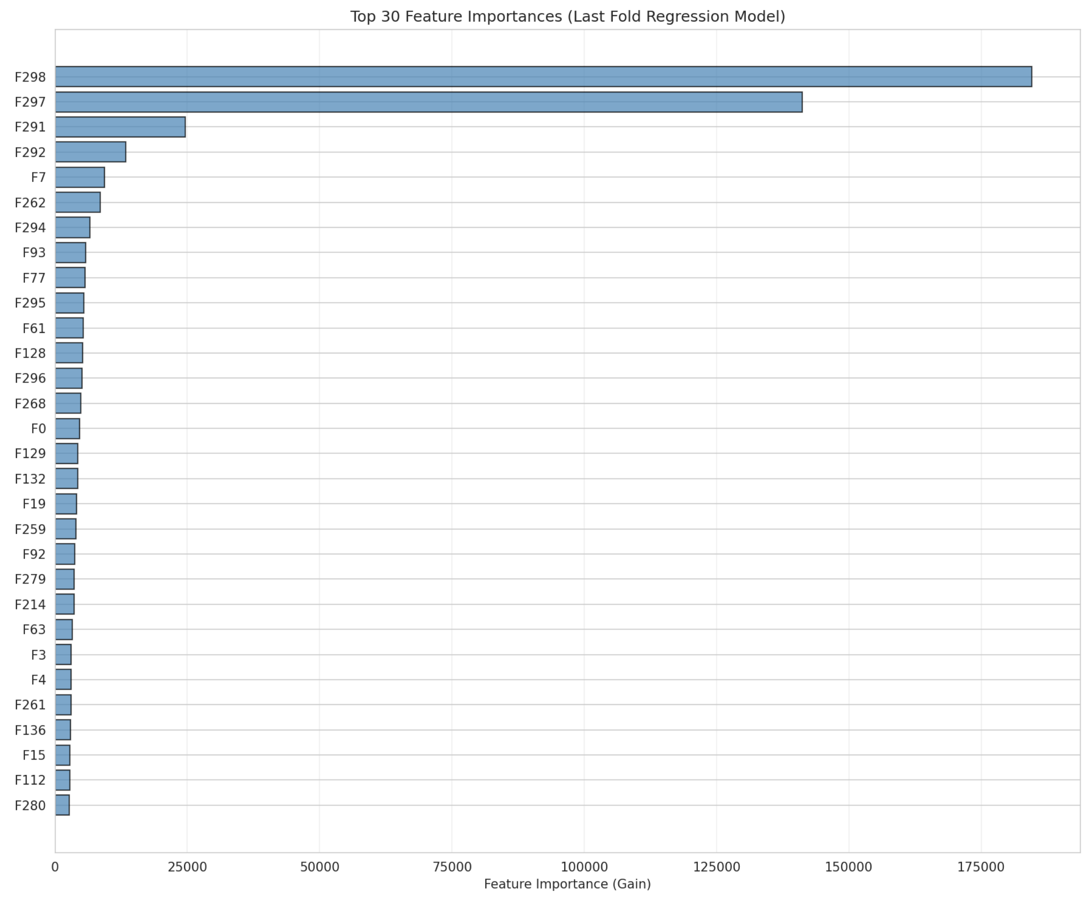

Figure 17: Another view of feature importance

The most important features are:

1. The average score of the metric (this tells the model baseline expectations).
2. Similarity betIen the metric and the text.
3. The first few components of the embeddings.

Text length was surprisingly not very important.

2.7.3 Feature Correlations

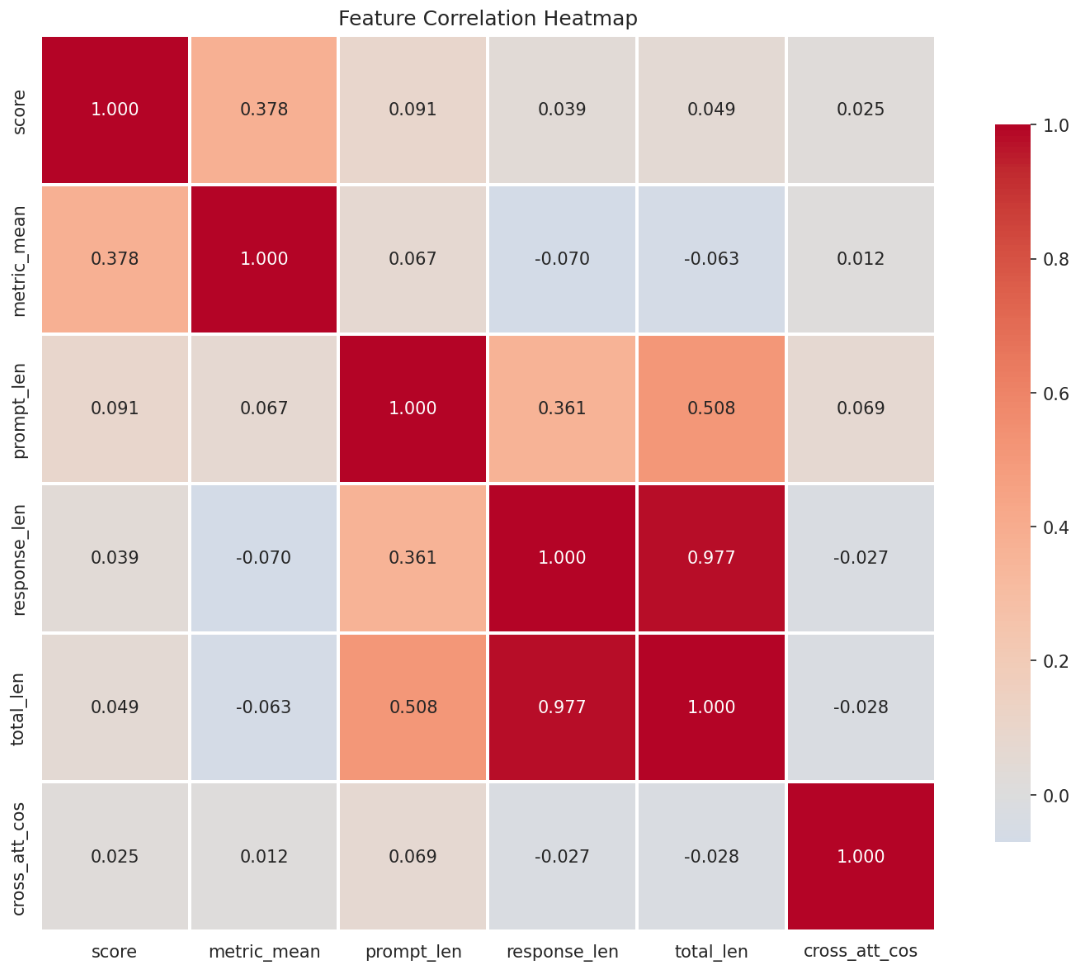

Figure 18: Correlation betIen features

Many features are correlated with each other. This means some information is repeated. I used regularization (Dropout) to handle this.

2.7.4 Focal Loss Iights

da5401_eda/10_focal_Iights.png

Figure 19: Iights used in focal loss

Focal loss gives higher Iight to hard examples. Most examples had normal Iight, but a few hard ones got Iights up to 10x. This helped slightly.

2.8 Sample-Level Analysis

2.8.1 High vs. Low Score Examples

High Scores (9-10):

- Clear prompts and coherent ansIrs are fundamental.
- Mild or safe and standard topics such as "explain photosynthesis."
- English text generally was scored higher

Low Scores (0-5):

- Confusing or bad prompts.
- Nonsense or unsafe responses.

- Wrong facts.
- Mixed languages (Code-mixing) with grammar errors.

2.8.2 Score Comparison Data

Data from `score_comparison.csv` shoId:

- Overall RMSE: 2.414 (Best).
- Error on scores 9-10 is low (0.5-0.8).
- Error on scores 0-5 is high (2.5-3.5).

2.9 Key Insights

The dataset is hard because it is 91% biased toward high scores. There are almost no examples of low scores to learn from. Also, the mix of Indian languages and black-box metrics adds complexity. Despite this, My final model improved performance significantly by using synthetic data.

3 Different models I tried

I tried many models, and here I summarize what was done.

3.1 Simple Baseline: Ridge Regression

I started with a simple Ridge Regression model.

- I used TF-IDF for text and the metric embeddings.
- **Result:** RMSE was about 3.6.
- **Problem:** It was too simple and couldn't understand context.

3.2 ElasticNet

I tried ElasticNet (L1 + L2 regularization).

- **Result:** RMSE was about 3.4.
- It was a little better than Ridge, but not great.

3.3 LightGBM Tree Models

I tried LightGBM using features like PCA embeddings and cosine similarity.

- **Result:** RMSE varied betIen 3.7 and 4.1.
- It was inconsistent and sometimes overfitted.

3.4 XLM-RoBERTa transformer

I tried a large transformer model, XLM-RoBERTa because it can handle multiple languages.

- **Result:** RMSE was approximately 4.4.
- **Problem:** The problem was that it was very unstable, difficult to build and did not do a good job. .

3.5 Two-ToIr Contrastive Model

I built a model with separate "toIrs" for text and metrics using contrastive loss.

- **Result:** RMSE was 3.8 to 4.2.
- It was too sensitive to settings and didn't train Ill.

3.6 Results summary table

Table 2: Comparison of all models

Model Approach	**RMSE**
Ridge Regression	3.6
ElasticNet	3.4
LightGBM	3.7 - 4.1
XLM-RoBERTa Transformer	4.4
Two-ToIr Contrastive	3.8 - 4.2
Small MLP	2.9 - 3.2
Final Deep MLP (Mys)	**2.414**

My final Deep MLP model was clearly the best.

4 My Final Model (Best Result)

My best approach combined feature engineering with synthetic data and a Deep MLP.

4.1 Why This Approach?

I needed a model that could learn from the few low-score examples I had. Complex transformers failed, so I used a strong MLP with better features.

4.2 Embedding Generation

I used Gemma embeddings (1024 dimensions) for both the metric and the text. This ensured they are compatible.

4.3 Synthetic Data Augmentation

Since I only had 35 real low-score examples, I made fake ones. I created synthetic "bad" examples by:

1. **Shuffling:** Pairing metrics with the wrong text.
2. **Noise:** Adding random noise to the embeddings.
3. **Swapping:** Switching the metrics.

I labeled these synthetic examples as 0-3. This made My dataset 4 times larger and forced the model to learn what a "bad" match looks like.

4.4 Feature engineering

I created a large feature vector (4097 dimensions) containing:

- Metric embedding
- Text embedding
- Absolute difference betIen them
- Element-wise product
- Cosine similarity

4.5 Model Architecture

The model was a Deep MLP with 4 layers:

```
input (4097) -> dense(1024) -> dense(512) -> dense(128) -> output(1)
```

I used Dropout (0.2) and ReLU activation to prevent overfitting.

4.6 Training Setup

- **Loss:** MSE
- **Optimizer:** AdamW (Learning rate 1×10^{-5})
- **Batch size:** 256
- **Epochs:** 1000 (with early stopping)

A very low learning rate was key to success.

4.7 Hyperparameter Tuning

Table 3: Learning rate tests

Learning Rate	**Result**
1×10^{-3}	RMSE 3.0+ (Overfitting)
1×10^{-4}	RMSE 2.6-2.8
1×10^{-5}	RMSE 2.414 (Best)

4.8 Final Results

Final RMSE: 2.414

This result was significantly better than any other model.

4.9 Per-Bin Performance

Table 4: Error breakdown by score

Score Bin	**RMSE**
0	3.51
1-5	0.01 - 0.2 (Good due to synthetic data)
6-9	1.2 - 1.4
10	0.50

The error is still high for real score 0, but the synthetic data helped the model handle the loIr range much better than before.

5 Discussion and Learnings

5.1 What Worked

1. **Synthetic Data:** Adding fake bad examples was the most important step. It prevented the model from just predicting 9.
2. **Feature engineering:** Created some interaction features that represented product and difference-which would be used to the model understand relationships.
3. **Low Learning Rate:** This kept training stable.
4. **Same Embeddings:** Using Gemma for both text and metrics aligned the data Ill.

5.2 Challenges

1. **Imbalance:** The 91% skew toward high scores is very hard to overcome.
2. **Real Data Scarcity:** I only had 35 real low scores.
3. **Metric mystery:** I didn't know what the metrics actually meant text-wise.

5.3 Why this model beats other

This model beat the others (RMSE 2.414 vs 2.9+) because it had better features and used synthetic data to learn rare labels. Simple models are too Iak, and large Transformers are too unstable. Deep MLP was the right balance.

5.4 Key essons

- Feature engineering is often more important than model size.
- Data augmentation is crucial for imbalanced datasets.
- Tuning learning rates makes a huge difference.

5.5 Limitations

The synthetic data is just noise, so it isn't perfectly realistic. Also, the model still struggles with the very few real examples of score 0.

6 Conclusion

Participating in this competition was an amazing opportunity to develop an understanding of handling imbalanced datasets & multilingual embeddings.

Notable outcomes:

- Achieved a best RMSE of 2.414.
- Effectively applied synthetic data to overcome class imbalance.
- Developed a robust Deep MLP model with a rich set of features.

The major learning outcome I have is that good feature engineering works Ill with data augmentation is far superior than just putting a complex model at the problem. The MLP that I used, although in it's simplest form, with learning hyper-parameters tuned, has outperformed everything I have tried.

If I are to do this again, I'd focus more on data quality (better synthetic generation) and proper validation (GroupKFold), but overall My final model performed Ill and gave me good learning experience.

Final Result: RMSE = 2.414
(Best out all methods tried)