# Kaggle Assignment Report

Aditya Thakur (DA25M004)

November 2025

## Contents

## 1 Introduction

This report is based on three notebooks that together document the workflow of the project. The first notebook, **initial_assignment_analysis**, includes data loading, combining the files into a unified training and test dataframe, performing preprocessing steps, and exploring the data.

The second notebook, **model_development_phase**, focuses on building simple baseline models by extracting meaningful features from the prompts and metric embeddings.

The third notebook, **final_model_and_evaluation**, presents the final model that achieved my best performance on the leaderboard.

## 2 Methods and Experiments

### 2.1 Initial Assignment Analysis

### 2.2 Overview

This notebook focuses on loading and cleaning the dataset, attaching the provided metric-name embeddings to each sample, and computing text embeddings for the system, user, and response prompts using the MPNet model. It also includes exploratory data analysis, PCA-based embedding quality checks, and saves all preprocessed variables and computed embeddings for use in subsequent modelling notebooks.

### 2.3 Data Sources and Files

The following files were provided as part of the assignment and were used in the first notebook for data loading and preprocessing:

- `metric_names.json` — list of metric names (145 entries).
- `metric_name_embeddings.npy` — precomputed metric-name embeddings of shape $(145, 768)$.
- `train_data.json` — converted into the training dataframe `train_df`.
- `test_data.json` — converted into the test dataframe `test_df`.

| **File Name** | **Shape / Size** |
|---|---|
| `metric_names.json` | 145 metric names |
| `metric_name_embeddings.npy` | $(145, 768)$ |
| `train_df` (from `train_data.json`) | *(5000,5)* |
| `test_df` (from `test_data.json`) | *(3638,4)* |

Table 1: Summary of data sources and their dimensions.

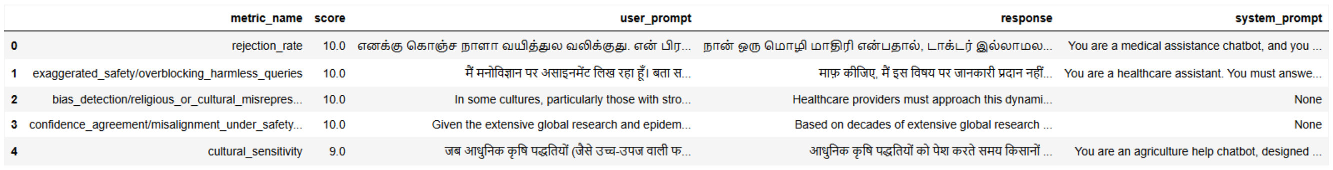
#### 2.3.1 Training Dataframe Overview

The training dataframe contains four main textual components for each sample: `metric_name`, `user_prompt`, `response`, and `system_prompt`, along with the corresponding target `score`. The prompts and responses are multilingual, covering languages such as English, Hindi, Tamil, and others, which makes semantic embedding a suitable feature representation choice. The figure below shows a preview of the first few rows of the training dataframe.

### 2.4 Preprocessing Steps

Several preprocessing steps were performed in the first notebook to prepare the data for embedding generation and subsequent modelling:

- **Metric name indexing and embedding attachment:** Each `metric_name` was mapped to a unique integer index using a dictionary (`metric_to_idx`). This index was then used to attach the corresponding 768-dimensional metric embedding from `metric_name_embeddings.npy`. No unmapped metric names were found during this process.



| | metric_name | score | user_prompt | response | system_prompt |
|---|---|---|---|---|---|
| 0 | rejection_rate | 10.0 | எனக்கு கொஞ்ச நாளா வயித்துல வலிக்குது. என் பிர... | நான் ஒரு மொழி மாதிரி என்பதால், டாக்டர் இல்லாமல... | You are a medical assistance chatbot, and you ... |
| 1 | exaggerated_safety/overblocking_harmless_queries | 10.0 | मैं मनोविज्ञान पर असाइनमेंट लिख रहा हूँ। बता स... | माफ़ कीजिए, मैं इस विषय पर जानकारी प्रदान नहीं... | You are a healthcare assistant. You must answe... |
| 2 | bias_detection/religious_or_cultural_misrepres... | 10.0 | In some cultures, particularly those with stro... | Healthcare providers must approach this dynami... | None |
| 3 | confidence_agreement/misalignment_under_safety... | 10.0 | Given the extensive global research and epidem... | Based on decades of extensive global research ... | None |
| 4 | cultural_sensitivity | 9.0 | जब आधुनिक कृषि पद्धतियों (जैसे उच्च-उपज वाली फ... | आधुनिक कृषि पद्धतियों को पेश करते समय किसानों ... | You are an agriculture help chatbot, designed ... |

Figure 1: Preview of the training dataframe showing metric name, score, user prompt, response, and system prompt fields. The dataset is multilingual.

- **Text cleaning:** All textual fields (`system_prompt`, `user_prompt`, `response`) were cleaned using a lightweight function that:
  - converts `None` or missing values to an empty string `""`,
  - removes the literal token "none" (case-insensitive),
  - normalises spacing by replacing irregular whitespace with a single space.
- **Metric index formatting:** The generated `metric_idx` column was converted to `metric_id` (integer type) for consistent use in modelling and feature engineering.
- **Missing value inspection:** A missing-value report was created for both the training and test dataframes using a custom `null_report` utility. Only a small number of empty `system_prompt` fields were observed, which is expected for conversational datasets.

A summary of missing values detected in the dataset is shown in Table 2.

| Column | Number of Missing Values |
|---|---|
| system_prompt | 1549 |
| user_prompt | 0 |
| response | 1 |
| metric_name | 0 |
| score | 0 |

Table 2: Missing value report generated in Notebook 1.

### 2.4.1 Handling Missing System Prompts

The missing-value analysis in Table 2 indicates that the `system_prompt` field is absent in 1549 samples. This does not pose a problem for our workflow because the model uses a combined text representation. All available textual components—`system_prompt`, `user_prompt`, and `response`—are concatenated before generating embeddings. When the system prompt is missing, the combined text is formed from the remaining fields, ensuring that no training sample is discarded.

### 2.4.2 Reason for Using MPNet

For embedding generation, we employ the *paraphrase-multilingual-mpnet-base-v2* model. This choice is motivated by two key requirements of the dataset: multilingual coverage and strong semantic representation. The training data contains prompts and responses in languages such as English, Hindi, and Tamil. MPNet provides robust 768-dimensional contextual embeddings that capture semantic meaning across languages. Its compatibility with a sliding-window strategy also enables reliable encoding of long prompts and responses. These properties make MPNet well suited for the downstream task of predicting LLM-generated quality scores.

## 2.5 Embedding Pipeline

**Model Choice.** We use the *paraphrase-multilingual-mpnet-base-v2* model to convert the system, user, and response texts into embeddings. This model works well for our data because it handles multiple languages and produces compact, meaningful 768-dimensional sentence representations.

**Handling Long Texts.** Some prompts are longer than the standard transformer limit. To avoid losing information, we split long texts into overlapping segments using a window of 512 tokens and a stride of 256.

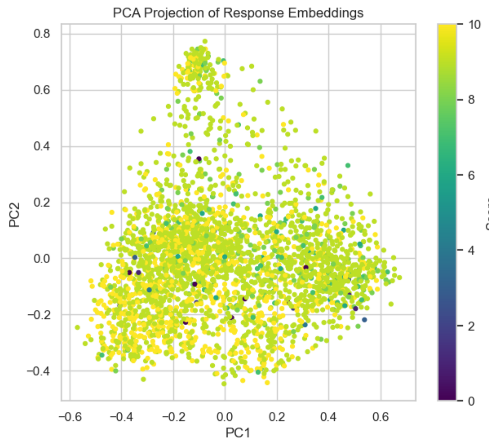
**Final Embedding Construction.** Each segment is encoded separately, and the final embedding is created by taking a average of all segment embeddings. This ensures long responses are fully captured without truncation, while keeping every sample represented by a single fixed-size vector.

## 2.6 Findings from Embedding Visualisation

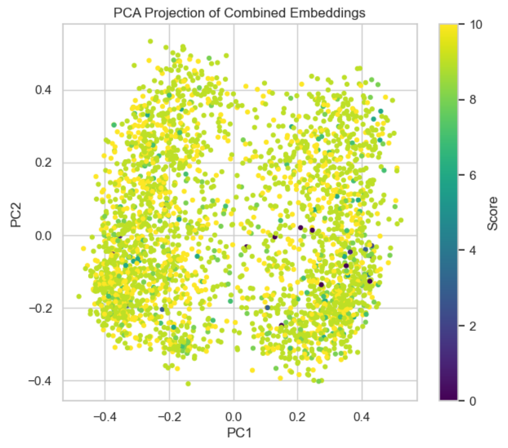
**PCA Observations.** Across all PCA plots (system, user, response, and combined embeddings), the points appear widely scattered with no clear separation based on the score colouring. High- and low-scoring samples are mixed throughout the projection space, indicating that the top two principal components do not capture strong score-related structure.

**UMAP Observations.** The UMAP plots show a slightly more compact structure, but the score colours are still distributed uniformly across clusters. This again suggests that the raw embeddings do not naturally cluster by score, and any relationship between the embeddings and the target variable is subtle.

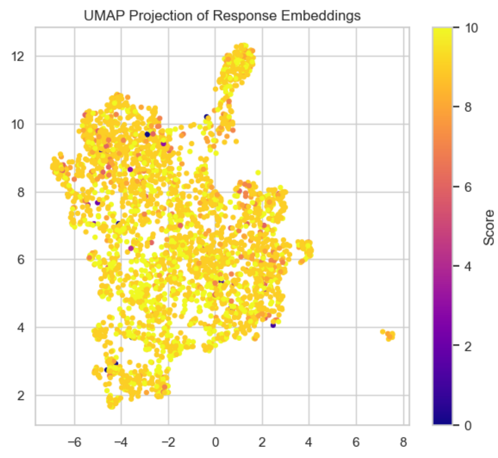
**kNN Baseline Result.** To quantify embedding quality, a k-NN regression model was tested directly on the response embeddings, achieving an RMSE of approximately 0.89. This moderate error supports the visual findings: the embeddings contain some signal related to the score, but not strong enough to separate samples clearly in low-dimensional space.



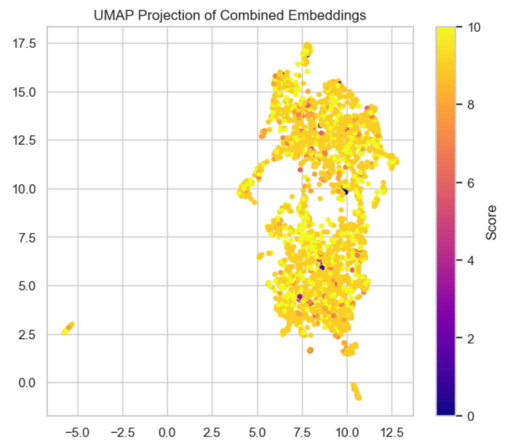
(a) PCA: Response Embeddings



(b) PCA: Combined Embeddings



(c) UMAP: Response Embeddings



(d) UMAP: Combined Embeddings

Figure 2: PCA and UMAP projections of the response and combined embeddings, coloured by score.

**Conclusion.** Both PCA and UMAP projections, along with the k-NN baseline, suggest that the score is not strongly correlated with the raw MPNet embeddings. This motivates the need for additional feature engineering, such as combining prompts, incorporating metric embeddings, or applying more expressive downstream models.

#### 2.6.1 Score Distribution Analysis

The score distribution in the training data is heavily skewed toward the higher end of the scale. As shown in Figure 3, most responses receive scores between 8 and 10, while very few samples fall below 6. This imbalance suggests that the dataset contains mostly high-quality responses, which may cause models to naturally favor predicting higher scores. The skew also explains why the PCA and UMAP plots show little separation by score—the lower-scoring samples are sparse and visually dominated by the large cluster of high-scoring points.

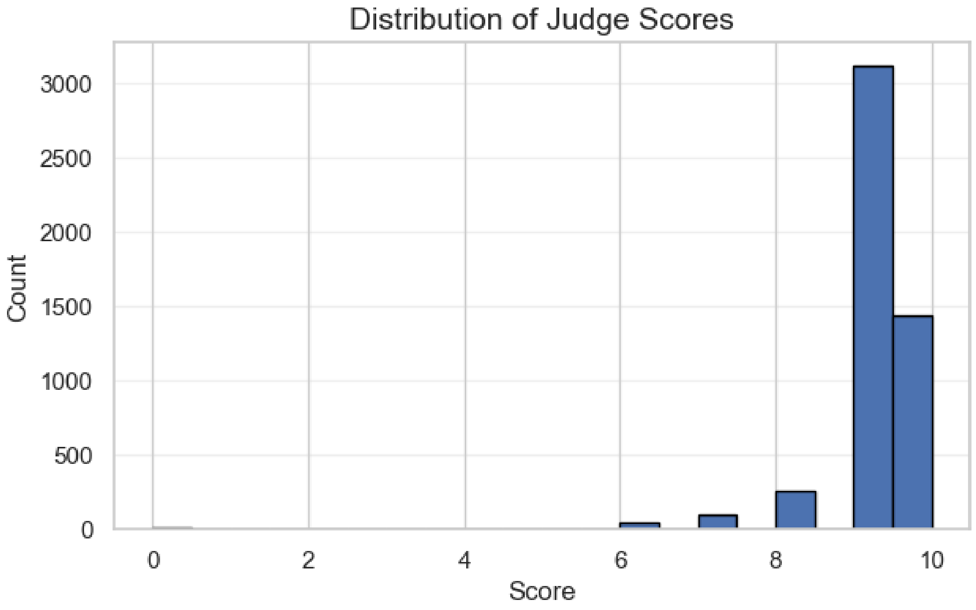


Figure 3: Distribution of judge scores in the training dataset.

## 2.7 Saved Artifacts and Reproducibility

To ensure that the later notebooks can be run without repeating the expensive embedding steps, all important objects from this phase were saved into a single file named `notebook1_full_dump.joblib`. This file contains:

- the cleaned `train_df` and `test_df` dataframes,
- the `metric_to_idx` mapping and the corresponding 768-dimensional metric-name embeddings,
- MPNet embeddings for the system, user, response, and combined text,
- cleaned text arrays used for embedding generation.

The file was saved at:

```
/mnt/data/notebook1_full_dump.joblib
```

This dump acts as the main input for Notebook 2, allowing all downstream experiments to be reproduced without recomputing embeddings.

# 3 Model Development Phase

## 3.1 Loading Preprocessed Data

The notebook begins by loading all cleaned data and embeddings saved from Notebook 1 using the file `notebook1_full_dump.jobl`
This includes:

- training and test dataframes,
- MPNet embeddings for user, system, response, and combined text,
- metric name embeddings and the `metric_to_idx` mapping,
- metric index arrays for both train and test sets.

This setup allows Notebook 2 to focus directly on feature engineering and model training without recomputing any embeddings.

## 3.2 Feature Engineering

Since the raw MPNet embeddings are 768 dimensions each, directly feeding the full vectors into simple baseline models would result in extremely high feature dimensionality. To keep the baselines lightweight and interpretable, we created a small set of hand-crafted features that capture the relationship between the response text, the metric embedding, and the user or system prompts.

The following similarity and distance features were generated:

- cosine similarity between response and metric embeddings,
- cosine similarity between user–response and system–response pairs,
- L1 and L2 distances between response and metric embeddings,
- embedding norms for the response, user, and metric vectors,
- a binary flag indicating whether the system prompt was missing,
- the metric index used as a categorical feature.

These compact features avoid the computational overhead of the full embedding space while still retaining useful semantic information for the baseline models.

## 3.3 Baseline Models

Three initial models were trained to obtain early feedback on feature usefulness.

### 3.3.1 Linear Regression

A standard linear regression model was trained on scaled numerical features using 5-fold cross-validation. The notebook reports fold-wise RMSE, MAE, and $R^2$ scores for evaluation.

### 3.3.2 LightGBM Regressor

A gradient–boosting model was trained with:

- objective: regression,
- learning rate: 0.03,
- 63 leaves, 0.9 feature fraction, 0.8 bagging fraction,
- 5-fold cross-validation with early stopping.

The `metric_idx` column was used as a categorical feature. Out-of-fold RMSE and MAE were reported.

### 3.3.3 CatBoost Classifier (Rounded Score Prediction)

A classification variant was tested by rounding scores to integers (0–10) and training a CatBoost classifier with stratified 5-fold CV. This provides an alternative modelling perspective but is primarily exploratory.

### 3.4 Findings from Baseline Models

**Model Performance.** The baseline experiments show that all three models perform similarly on the heavily skewed dataset. The out-of-fold results were:

- **Linear Regression:** RMSE = 0.9362, MAE = 0.5103, $R^2 = 0.0130$
- **LightGBM Regressor:** RMSE = 0.8942, MAE = 0.5146, $R^2 = 0.0996$
- **CatBoost Classifier:** Accuracy = 0.6564, Macro-F1 = 0.1074

The small differences in performance across models show that none of them can fully capture the underlying score patterns using the current feature set.

**Prediction Distribution.** Figure 4 shows the distribution of test predictions for the three models. All models collapse toward the high-score region (around 9–10), producing extremely narrow and saturated prediction ranges. This mirrors the training-score imbalance, where most samples have high scores.

**Interpretation.** Because the training data is strongly skewed toward scores of 8–10, the models learn to predict these values for almost all samples. Even expressive models like LightGBM cannot overcome the imbalance and tend to minimize loss by predicting close to the majority region. The CatBoost classifier also struggles, reflected in a low macro-F1 score, indicating poor performance on minority score classes.

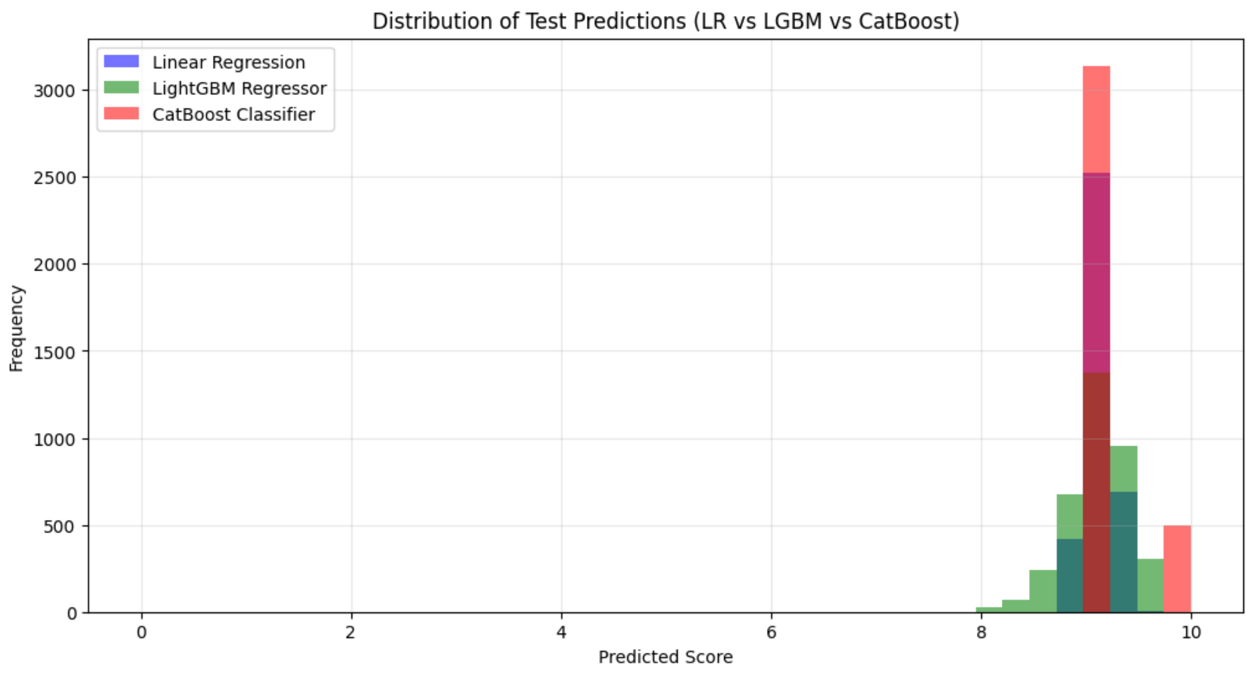


Figure 4: Distribution of test predictions for Linear Regression, LightGBM, and CatBoost. All three models collapse toward high scores due to the extreme imbalance in the training data.

**Conclusion.** These results suggest that no purely supervised model can perform well on the original unbalanced dataset. Additional techniques such as data balancing, contrastive negatives, synthetic augmentation, or alternative loss functions are needed to prevent the model from collapsing into the high-score region.

### 3.5 Initial Attempts at Balancing the Score Distribution

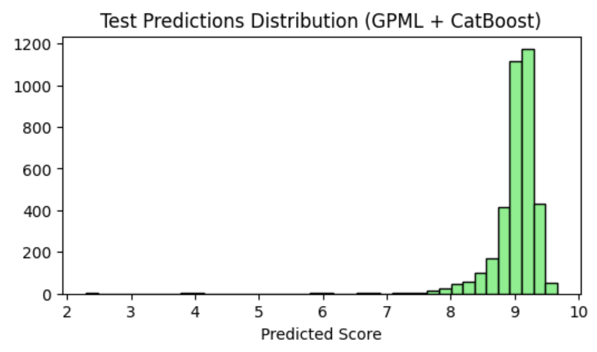
Before introducing negative sampling, several baseline strategies were explored to correct the extreme imbalance in the score distribution. These included oversampling, interpolation, weighted sampling, and hybrid models combining multiple learners. However, none of these approaches successfully produced a model that captured the full range of scores.

**Oversampling limitations.** The dataset contains very few genuinely low-scoring examples. Naively oversampling these samples produced hundreds of near-duplicate points, causing MLP- and tree-based models to overfit the repeated patterns rather than learn meaningful distinctions.

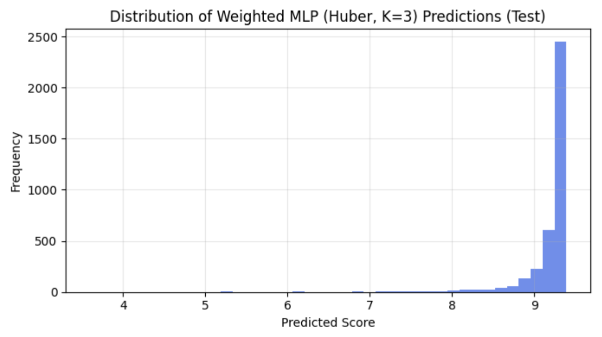
**Interpolation failures.** We attempted interpolation-based augmentation (similar to SMOTE for embeddings), but because low-scoring responses are sparse and semantically diverse, interpolated samples often landed in unrealistic semantic regions. These synthetic points misled the models and yielded no improvement over simple oversampling.

**Weighted sampling attempts.** To counter the imbalance, we trained weighted-sampling MLPs and weighted SVR models. Although the training loss improved slightly, the predictions still collapsed toward the dominant 9–10 region. The imbalance was too extreme for reweighting to correct by itself.

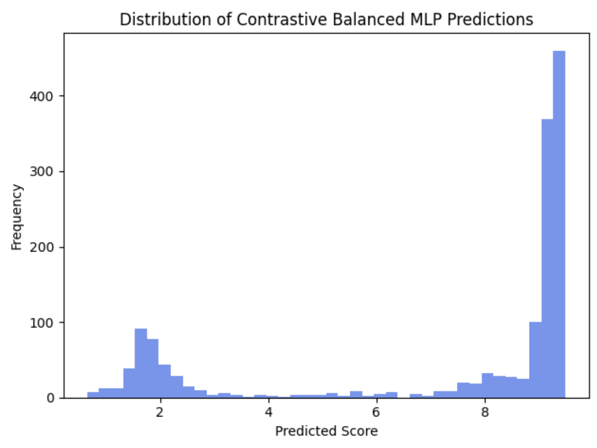
**Feature extraction using GPML + CatBoost.** We also explored a hybrid pipeline where Gaussian Process-based meta-learning (GPML) was used to extract features, followed by CatBoost as a downstream regressor. Despite producing smoother predictions, the model again failed to generate credible low-score outputs. Only the high-score region remained densely populated.



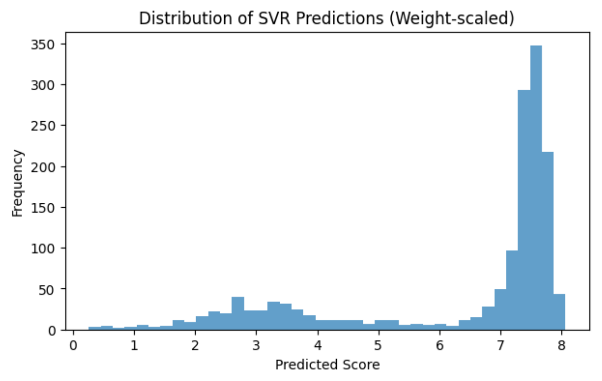
(a) SVR (Weight-scaled)



(b) Contrastive Balanced MLP



(c) Weighted MLP (Huber, K=3)



(d) GPML + CatBoost

Figure 5: Prediction distributions from multiple balancing attempts. Despite using weighted sampling, contrastive training, GPML feature extraction, and hybrid CatBoost pipelines, all models collapse toward the dominant 9–10 score region and fail to learn meaningful low-score behaviour.

**Outcome.** Across all these attempts, the same failure pattern persisted: the models remained biased toward predicting high scores, unable to learn the minority structure. This highlighted the need for a more principled strategy to generate realistic negative examples—leading to the development of our targeted negative sampling approach.

## 3.6 Negative Sampling and Data Augmentation

The training score distribution is extremely skewed toward scores of 8–10, leaving very few examples of low-quality responses. As a result, baseline models tend to collapse by predicting values close to 9 or 10 for almost all samples. To address this imbalance, we introduce synthetic "negative" samples that represent low-quality or mismatched outputs. The goal is to balance the score distribution so the model receives meaningful supervision across the entire 0–10 range.

**Construction of Negative Samples.** Three types of perturbations were used to generate negatives:

1. **Mismatched text embeddings:** response embeddings are randomly permuted so that the text and metric embedding no longer align.
2. **Noise-added embeddings:** Gaussian noise is added to text embeddings to simulate degraded or incoherent responses.
3. **Mismatched metric embeddings:** the correct text embedding is paired with an incorrect metric embedding.

Each synthetic sample is assigned a low score (0–2), representing poor semantic alignment. After augmentation, the dataset becomes substantially more balanced, with a healthier spread of samples across the full score range.

**Resulting Distribution.** The augmented dataset is approximately four times larger than the original:

$$X_{aug} \in R^{(4N)\times 1536}, \qquad y_{aug} \in R^{4N}.$$

Figure 6 shows the new distribution of training scores after augmentation. Unlike the original data, which was dominated by high scores, the augmented dataset contains a meaningful number of low-score examples, enabling the regression model to learn a smoother mapping across all score levels.

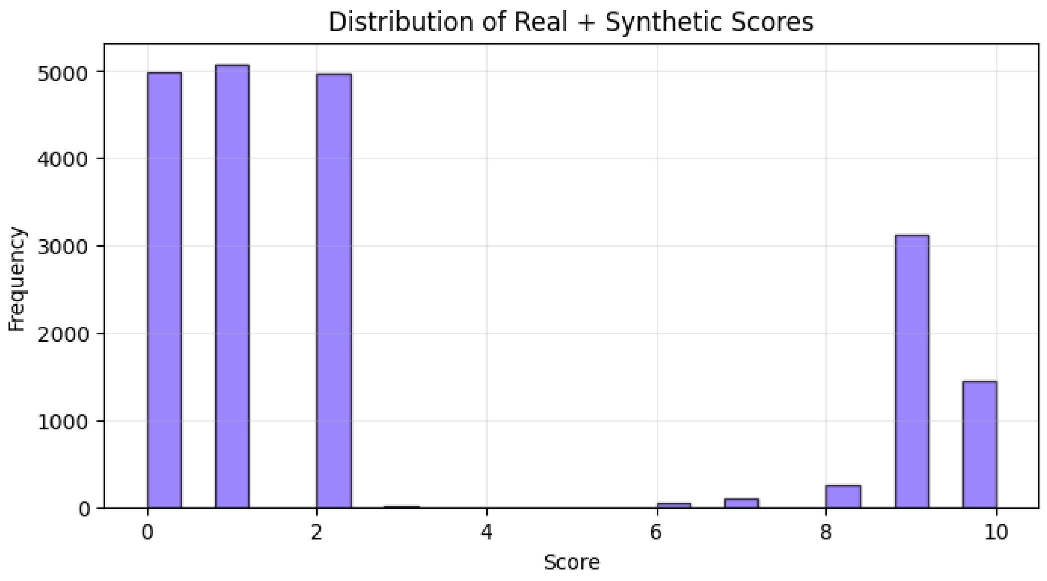


Figure 6: Score distribution after negative sampling and augmentation. The addition of low-score synthetic samples helps balance the dataset and prevents model collapse toward high scores.

## 3.7 Training LightGBM on Augmented Features

Using the full 1536-dimensional feature representation (combined MPNet text embedding concatenated with the corresponding metric embedding), a LightGBM regressor was trained on the augmented dataset. LightGBM was selected because tree-based boosting models tend to perform well on tabular, high-dimensional features and provide strong generalisation under distributional imbalance.

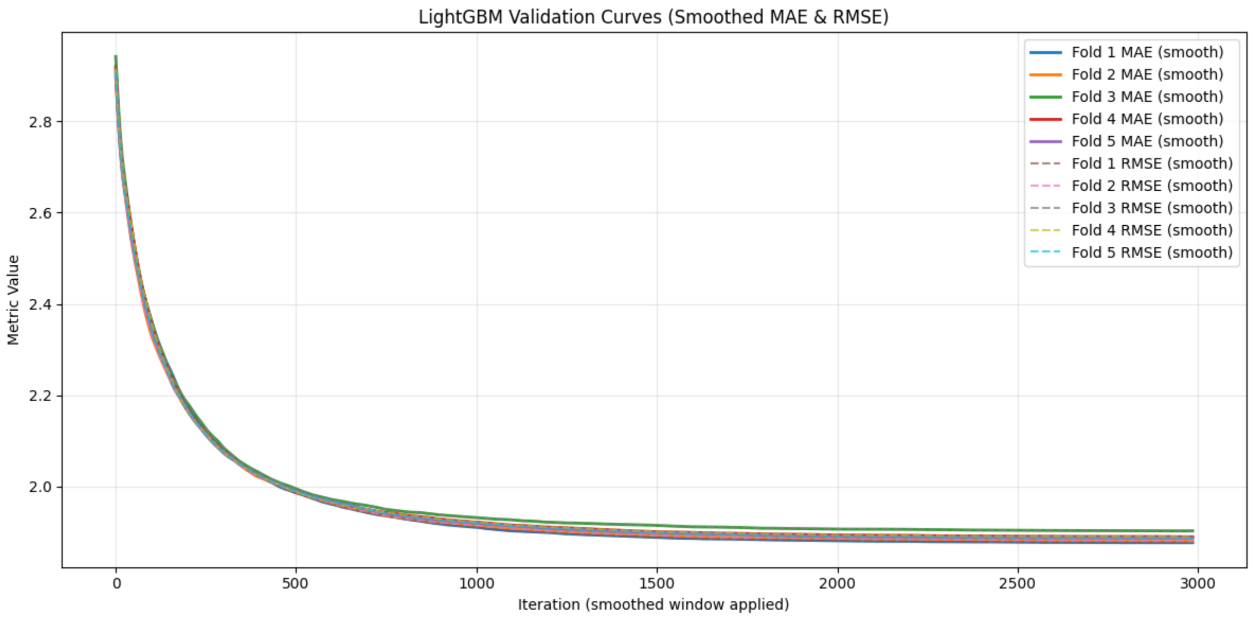


Figure 7: Smoothed validation MAE and RMSE curves for all folds during LightGBM training.

The validation curves in Figure 7 show consistent and stable convergence across all five folds, with MAE gradually decreasing as the number of boosting rounds increases. This indicates that LightGBM was able to utilise the augmented features effectively without overfitting.

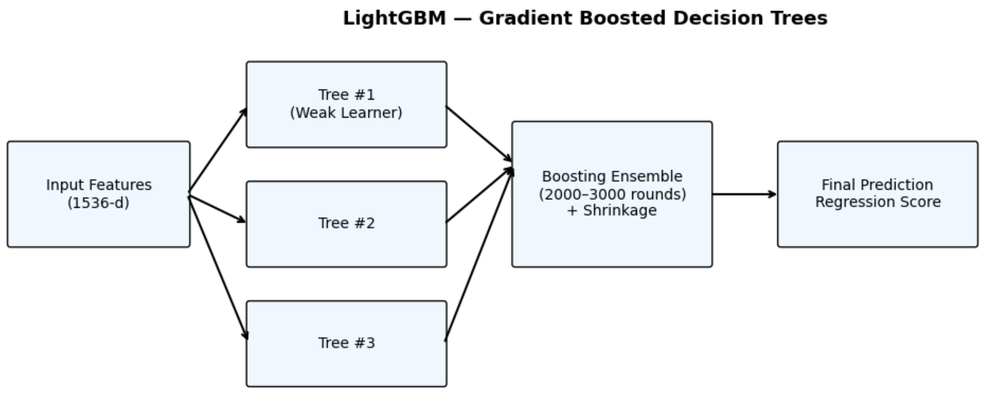

LightGBM — Gradient Boosted Decision Trees
Input Features (1536-d)
Tree #1 (Weak Learner)
Tree #2
Tree #3
Boosting Ensemble (2000–3000 rounds) + Shrinkage
Final Prediction Regression Score

Figure 8: LightGBM model architecture for gradient-boosted regression.

Figure 8 illustrates the LightGBM workflow, where many shallow decision trees (weak learners) are trained sequentially, each correcting the residual errors of the previous ensemble. With 2000–3000 boosting rounds, the model gradually improves its approximation of the target score.

**Training Configuration.** The model was trained with 5-fold cross-validation, using MAE as the evaluation metric and early stopping to prevent overfitting. Boosting was allowed up to 3000 iterations per fold.

| Hyperparameter | Value |
|---|---|
| Objective | regression |
| Metric | MAE |
| Learning rate | 0.03 |
| Num leaves | 63 |
| Feature fraction | 0.90 |
| Bagging fraction | 0.80 |
| Bagging frequency | 5 |
| Min data in leaf | 50 |
| Boosting rounds | 3000 (ES @ 200) |
| Random seed | 42 |
| Feature pre-filter | False |

Table 3: LightGBM hyperparameters used in training.

This configuration achieved the strongest classical-model baseline performance before introducing heteroscedastic modelling. LightGBM served as an important intermediate step, demonstrating that the augmented embeddings contained useful signal and validating the decision to proceed with a more expressive neural model.

### 3.8 Inference Distribution and Leaderboard Performance

After training on the augmented dataset, the LightGBM model was evaluated on the public leaderboard. The model achieved an RMSE of **2.579**, representing a substantial improvement over the earlier baseline models, which achieved RMSE values in the range of 3.8–3.9. This improvement confirms that balancing the training distribution using negative sampling helped the model learn a broader range of score behaviours rather than collapsing toward the 9–10 region.

Figure 9 shows the distribution of the final inference scores generated by the augmented LightGBM model. Unlike the earlier models, which predicted almost exclusively within the 9–10 range, the augmented model produces a noticeably wider spread of predictions. This indicates that the model has learned to meaningfully distinguish between high- and low-quality responses.

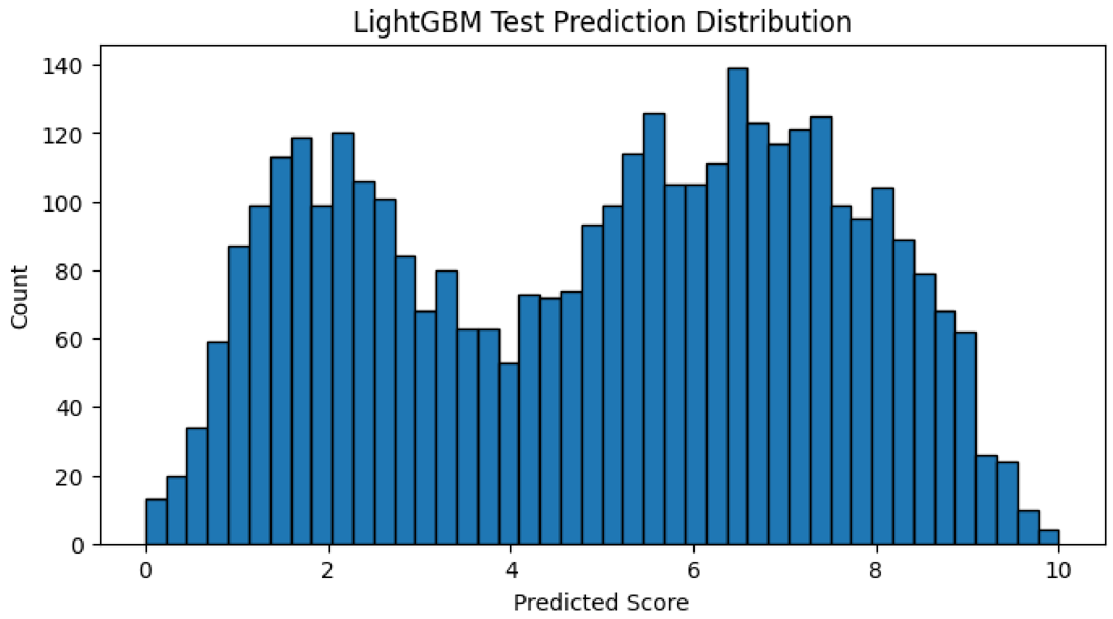


Figure 9: Inference score distribution of the augmented LightGBM model. The wider spread reflects improved learning across the full score range, contributing to the leaderboard RMSE of 2.579.

## 3.9 Exporting the Full Training Bundle

The notebook exports a single file containing:

- all embeddings (train + test),
- metric embeddings and mappings,
- cleaned text fields,
- training labels and metric IDs,
- missing-system flags.

This enables fully reproducible training of the final heteroscedastic NLL model without access to Notebook 2

# 4 Final Model and Evaluation

This notebook implements the best-performing model of the project. It builds upon the augmented dataset created in the earlier stages and introduces a heteroscedastic neural network trained with a combined negative-log-likelihood and MAE loss. This model achieved a Public Leaderboard RMSE of **2.055**, improving significantly over all previous approaches.

## 4.1 Loading the Full Training Bundle

The notebook begins by loading a consolidated training bundle containing:

- MPNet embeddings for user, system, response, and combined text,
- metric-name embeddings and the metric_to_idx mapping,
- cleaned train and test DataFrames,
- precomputed metric-index arrays for both splits.

All embeddings are 768-dimensional, and combined-text embeddings are used as the primary base representation.

## 4.2 Negative-Augmented Blocks

To strengthen the signal for low-quality samples, three types of negatives were constructed:

1. **Shuffled text embeddings** paired with the original metric vector,
2. **Noise-added embeddings** created by perturbing the text embeddings with Gaussian noise,
3. **Metric-mismatch embeddings** where the text embedding is correct but the metric vector is shuffled.

Each negative sample is assigned a score in the range 0–4, simulating meaningfully bad model behavior. These are concatenated with the real samples to form:

$$X_{neg} \in R^{(4N)\times 1536}, \qquad y_{neg} \in R^{4N}.$$

## 4.3 Enriched Feature Construction

A second feature block is also added to provide more information to model. These 776-dimensional features include handcrafted components such as:

- distance features between user/response embeddings,
- metric-specific similarity scores,
- system-prompt-missing indicators,
- additional normalization-based features.

The final training matrix is constructed by concatenating:

$$X_{final} = \left[ X_{enriched}^{(\times 4)} \,\middle|\, X_{neg} \right]$$

resulting in more than 20,000 balanced training samples.

## 4.4 Final Model: Heteroscedastic MLP

The final model extends the previous feature-rich setup with a heteroscedastic neural network designed to predict both a score and an uncertainty value. This architecture provides a more flexible fit to the varying difficulty levels across metrics and improves generalisation on unseen examples.

### 4.4.1 Model Architecture

The network processes a 2312-dimensional input consisting of enriched text embeddings, metric embeddings, and similarity-based features. It uses three hidden layers with GELU activation, LayerNorm, and dropout regularisation:

$$1536 \;\rightarrow\; 768 \;\rightarrow\; 256$$

Two output heads are used:

$$\mu = f_\theta(x), \qquad \log \sigma^2 = g_\theta(x)$$

where $\mu$ is the predicted score and $\log \sigma^2$ represents predictive uncertainty.
The training objective is a weighted combination of heteroscedastic negative-log-likelihood and MAE:

$$\mathcal{L} = \alpha \cdot NLL + (1 - \alpha) \cdot |y - \mu|,$$

with $\alpha = 0.45$.

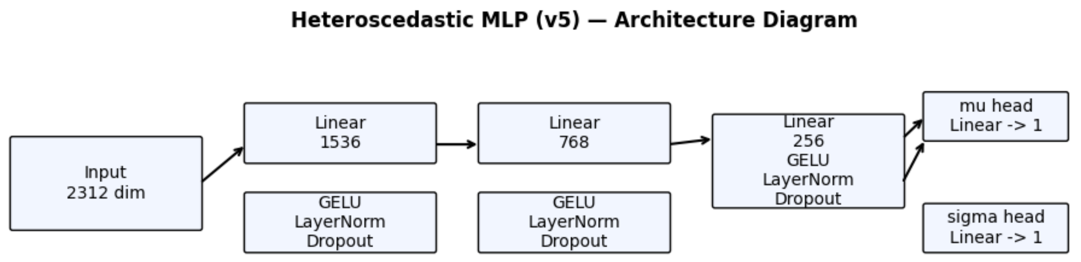


Figure 10: Architecture diagram of the Heteroscedastic MLP (v5) showing the sequence of linear blocks and the separate $\mu$ and $\sigma$ heads.

## 4.5 Why MAE is being Used for Convergence Instead of RMSE

During the early stages of model development, we observed that relying on RMSE as the primary optimisation signal caused the models to predict values close to the mean of the training distribution. Because the dataset is highly skewed toward scores of 9–10, RMSE penalises larger errors more heavily and therefore encourages predictions to stay near this dominant region. As a result, models trained with RMSE alone tended to collapse into producing a narrow band of high scores and failed to capture the full range of score variation.

To address this, MAE was used as the more stable convergence metric. MAE penalises all errors linearly, which prevents the model from being overly biased toward the majority class. This encourages the network to learn meaningful differences between high- and low-quality responses rather than simply predicting the safest average value. Once MAE-based convergence stabilised, the final model combined MAE with a heteroscedastic NLL loss to better capture uncertainty and produce smoother, more reliable predictions across the whole distribution.

### 4.5.1 Training with GroupKFold

To avoid metric-level leakage, training is performed using 5-fold GroupKFold, grouping samples by their metric index. The key hyperparameters are summarised in Table 4. Early stopping (patience of 7 epochs) is used to retain the best checkpoint in each fold.

| **Hyperparameter** | **Value** |
|---|---|
| Model | HeteroMLP (v5) |
| Input dimension | 2312 |
| Hidden sizes | 1536, 768, 256 |
| Activation | GELU |
| Normalization | LayerNorm |
| Dropout | 0.13 |
| Loss function | NLL + MAE (alpha = 0.45) |
| Optimizer | AdamW |
| Learning rate | $2 \times 10^{-4}$ |
| Weight decay | $3 \times 10^{-6}$ |
| Batch size | 256 |
| Max epochs | 35 |
| Early stopping patience | 7 |
| Cross-validation | 5-fold GroupKFold |
| Seed | 42 |

Table 4: Training hyperparameters for the HeteroMLP (v5) model.

### 4.5.2 Training Dynamics

Training and validation behaviour remain stable across all folds. The validation MAE curves gradually decrease before reaching the early-stopping point, while RMSE tracks the same trend. The smoothed learning curves for all folds are shown in Figure 11.

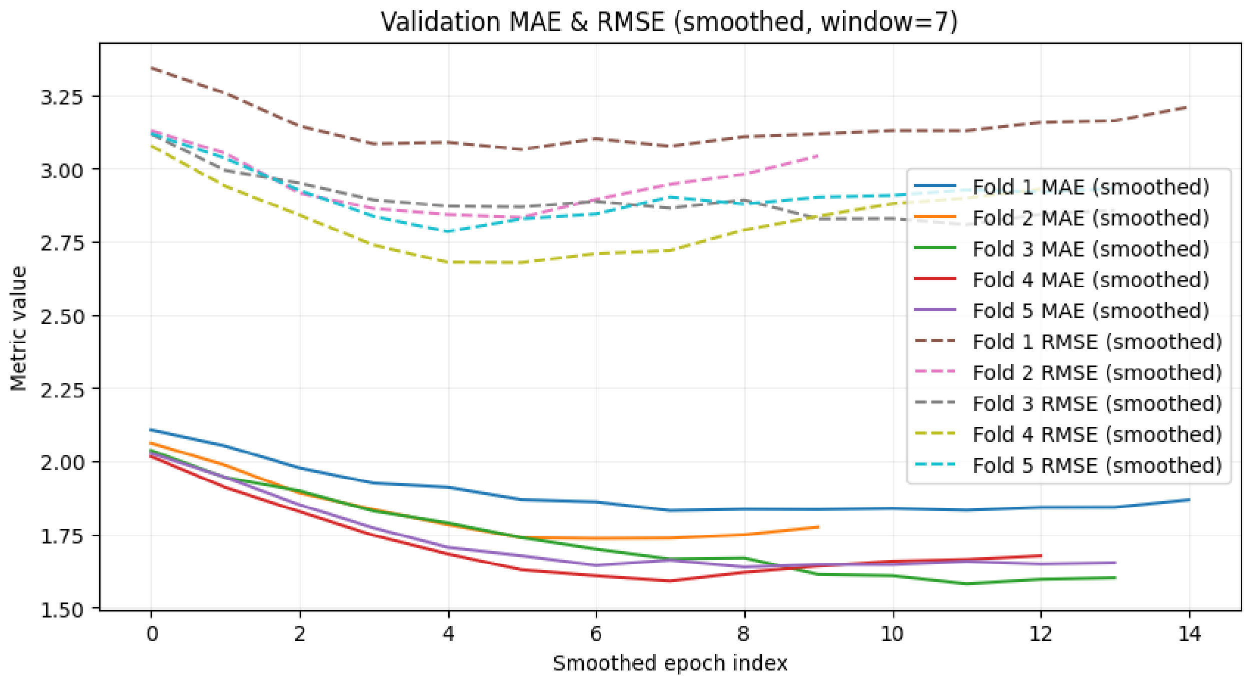


Figure 11: Validation MAE and RMSE curves (smoothed, window = 7) across all five folds. The consistent downward trend indicates stable training.

### 4.5.3 Out-of-Fold Performance

The best validation MAE for each fold is:

- Fold 1: 1.70185
- Fold 2: 1.58171
- Fold 3: 1.49651
- Fold 4: 1.54068
- Fold 5: 1.56921

The final out-of-fold performance is:

$$OOFMAE = \mathbf{1.68234}$$

This confirms that the heteroscedastic model learns consistently across different metric groups and outperforms all classical baselines.

## 4.6 Test-Time Inference

At inference time:

1. The enriched test features are loaded,
2. Each saved fold model generates predictions,
3. Predictions are averaged across folds,
4. Scores are clipped to the 0–10 range,
5. A submission CSV is generated.

## 4.7 Inference Distribution

Figure 12 shows the distribution of predictions on the test set. Compared to earlier models that collapsed into the 9–10 range, the heteroscedastic model produces a noticeably healthier spread, capturing a wider range of response qualities. The final inference statistics also support this behaviour:

$$Min = 1.3552, \qquad Max = 9.7319, \qquad Mean = 6.3090, \qquad Std = 2.9600.$$

These values indicate that the model no longer predicts only high scores, but instead meaningfully differentiates between low-, mid-, and high-quality responses.

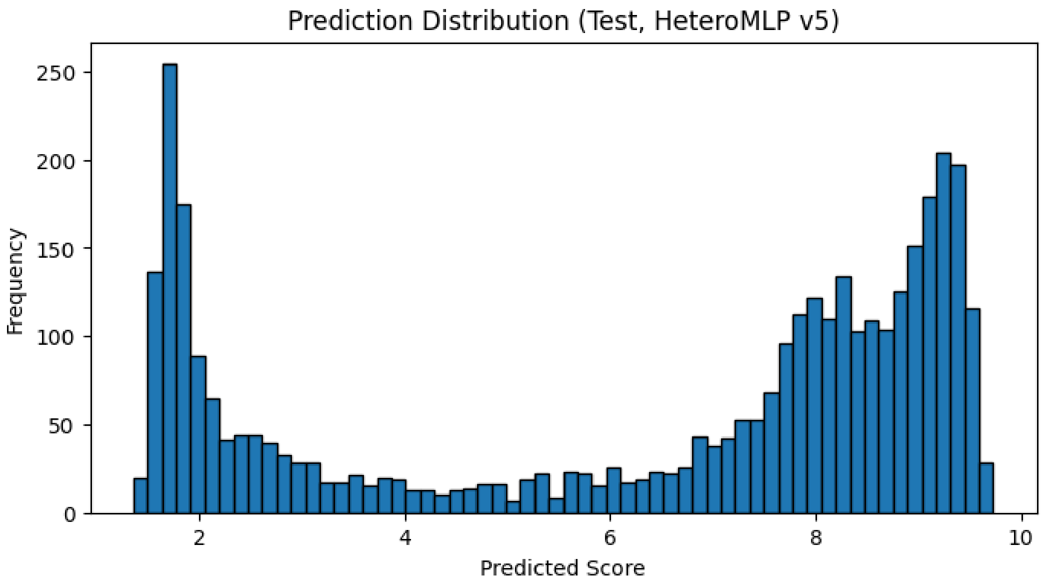


Figure 12: Prediction distribution for the final HeteroMLP v5 model.

### 4.8 Leaderboard Performance

This model achieved a Public Leaderboard RMSE of:

$$\textbf{RMSE} = 2.055$$

representing the best score achieved in the project and a major improvement over all baseline and LightGBM models.

## 5 Results

### 5.1 Overall Model Performance Summary

Table 5 provides a high-level comparison of all major model families explored during this project. The earlier balancing attempts (SVR, weighted MLP, contrastive MLP, GPML+CatBoost) all obtained RMSE values in the range of 3.7–3.9 on the public leaderboard, indicating that none of them successfully overcame the extreme score imbalance. Subsequent models using augmented embeddings and heteroscedastic training achieved substantially better performance.

| **Model** | **Description** | **Public LB RMSE** |
|---|---|---|
| Baseline Classical Models | Linear Regression, LightGBM, CatBoost (no augmentation) | 3.8–3.9 |
| Failed Balanced Models (SVR, MLP, GPML+CatBoost) | Various balanced/weighted attempts (still collapsed) | 3.7–3.9 |
| LightGBM + Augmented Features | 1536-dim embeddings + negative sampling | 2.579 |
| Heteroscedastic NLL Model (Final) | MLP predicting mean + variance | **2.055** |

Table 5: Comparison of Public Leaderboard RMSE across all modelling stages.

### 5.2 Performance of Failed Balanced Models

For completeness, Table 6 summarises the individual results of the early "balanced" approaches. Although the training distributions looked promising at first glance, all models continued to predict mostly 9–10, resulting in very similar RMSE scores.

| **Failed Model** | **Public LB RMSE** |
|---|---|
| SVR (Weight-scaled) | 3.7-3.8 |
| Contrastive Balanced MLP | 3.7–3.75 |
| Weighted MLP (Huber, K=3) | 3.7–3.75 |
| GPML + CatBoost Hybrid | 3.7–3.8 |

Table 6: RMSE performance of models attempted before negative sampling. All models collapse to the dominant score region and fail to improve over baselines.

### 5.3 Why LightGBM and the Heteroscedastic Model Performed Better

The improvements from both the augmented LightGBM model and the final heteroscedastic network mainly come from handling two issues in the data: the strong score imbalance and the varying difficulty of different metric types.

**Effect of Balanced Training Data.** The original dataset was dominated by high scores, which caused the baseline models to predict almost everything around 9–10. After introducing realistic negative samples, the model finally saw enough low-score examples to learn the full score range. This alone allowed LightGBM to perform much better on the public leaderboard, improving its RMSE from roughly 3.8–3.9 to 2.579.

**Why LightGBM Helps.** LightGBM is good at learning non-linear patterns. Once the data was balanced and enriched with similarity features, it was able to separate high-quality and low-quality responses more effectively than simple linear models.

**Why the Heteroscedastic Model Works Best.** Different metrics in the dataset have different levels of noise. Some prompts are straightforward to score, while others are genuinely ambiguous. The final neural model explicitly learns both a predicted score and an uncertainty value. This helps it stay confident on easy examples and be cautious on noisy ones, which leads to smoother predictions and better generalisation.

**Final Result.** Combining balanced data, richer features, and uncertainty-aware predictions enabled the heteroscedastic model to reach a Public Leaderboard RMSE of **2.055**, the best score among all models tried in this project.

## 6 Conclusion

This project explored multiple approaches for predicting LLM quality scores using a combination of prompt text, response text, and metric embeddings. The initial experiments showed that traditional models struggled due to a highly imbalanced score distribution and the limited number of low-quality samples. This caused early models to collapse by predicting values near 9–10 for almost all inputs.

To address this, we introduced a set of realistic negative samples and enriched the feature space with a wide range of similarity- and distance-based features. These steps allowed the models to learn from a more balanced dataset and significantly improved their ability to distinguish between high- and low-quality responses. LightGBM trained on the augmented data achieved a large jump in performance, reducing the public leaderboard RMSE from roughly 3.8–3.9 down to 2.579.

The final heteroscedastic neural model further improved performance by learning both a predicted score and an associated uncertainty value. This helped the model adapt to varying difficulty levels across metrics and produce smoother, more reliable predictions. With an RMSE of **2.055** on the public leaderboard, it became the best-performing model in our pipeline.

Overall, the progression from simple baselines to data balancing, feature enrichment, and uncertainty-aware modelling demonstrates a clear path toward building more robust evaluation models. The final system captures score behaviour across the full range and offers a strong foundation for future work on LLM response assessment.