LLM Response Classification Report

Project 2 - Machine Learning Project

Date: November 6, 2025

1. Dataset and Task

The dataset has 57,477 training samples, the goal of the competition is to determine which one is the best ai LLM answers to human prompt. So each row of the data is constituted of the prompt, the answer of the model a , the answer of the model b, and 3 colonne for 3 classes (0 or 1):

- a winner
- b winner
- winner tie

2. Methodology and Results

Step 1: Baseline Model

WE pulled out eight basic features comparing the two responses: differences in character count, word count, sentence count, average word length, uppercase ratio, digit count, punctuation count, and prompt length. Fed these into multinomial Logistic Regression, with an 80/20 stratified train-test split (random_state=42).
– Validation accuracy: 43.48%
– Kaggle Log Loss: 1.11616, ranked 167th
Basic length and lexical features catch things like verbosity and formality, but don't get at meaning. Still, not a bad starting point.

Step 2: Embedding-based Model

We used all-MiniLM-L6-v2 to turn prompts and responses into embeddings vectors.
Built a feature vector by stacking the prompt embedding, the difference between response A and B, and the absolute difference.
StandardScaler to normalize, then trained Logistic Regression (C=0.01).
– Validation accuracy: 43.69%
– Validation Log Loss: 1.0653
– Kaggle Log Loss: 1.07503, same rank (167)
That's a 3.65% improvement over the baseline. So, looking at what the responses actually say helps.

Step 3: Fine-tuned Transformer

For this part we fine-tuned DeBERTa-v3-extra-small (70.8M parameters) with inputs like “prompt [SEP] response_a [SEP] response_b”. Had to truncate the prompt to 200 chars and each response to 150 chars. Trained with AdamW (lr=2e-5), batch size 8, early stopping after one epoch, on two Tesla T4 GPUs.
– Validation accuracy: 30.90%
– Validation Log Loss: 1.0986
– Classification report: Precision and recall were zero for both “Model A” and “Model B”; the model just predicted “Tie” every time.
So, this one fell apart. It overfit, collapsed to one class, and was just too big for the data. Chopping up the input and a tiny batch size definitely didn’t help. The simple models actually worked better.

Performance Comparison :

Model	Val Acc	Val Log Loss	Kaggle Log Loss	Place
Baseline (Lexical)	43.48%	~1.087	1.11616	167
Embedding-based	43.69%	1.0653	1.07503	167
DeBERTa Fine-tuning	30.90%	1.0986	Not submitted	—

Model	Val Acc	Val Log Loss	Kaggle Log Loss	Place
Baseline	43.48%	~1.087	1.11616	167
Embedding-based	43.69%	1.0653	1.07503	167
DeBERTa Fine-tuning	30.90%	1.0986	Not submitted	-

Error and Bias Analysis :

Found a few biases:
– Verbosity bias: Longer responses usually win. Saw this in boxplots.
– Position bias: we didn’t model this directly, but the difference features pick up on it a bit.
– Class collapse (DeBERTa): The big model just learned the class distribution, not real preferences.

Error sources:
– Human preferences are subjective.
– Truncating inputs (DeBERTa: just 256 tokens total) tosses out context.
– 57K samples just isn’t enough to fine-tune a 70M parameter model.

Validation Strategy :

Used an 80/20 stratified split (random_state=42) to keep classes balanced. Log loss matches Kaggle's metric. Early stopping kept DeBERTa from overfitting (though in this case, it barely trained).

Reproducibility Notes :

Import the kaggle Notebook
Run with two Tesla T4 GPUs, no internet access.

Set random_state=42 for splits and sklearn models. All models loaded from local Kaggle datasets (/kaggle/input/all-minilm-l6-v2/).

Limitations and Future Directions :

Limitations:
1. Aggressive truncation chops out context
2. Didn't address position bias directly
3. Ignored model metadata
4. No ensembling or calibration

Ideas for next time:
1. Let models see up to 512 tokens
2. Train on (A, B) and (B, A) orderings
3. Try ensembling embedding and lexical models
4. Use LoRA fine-tuning for efficiency
5. Add temperature scaling for calibration