Technical Report: Udhgam 2.0 ML Challenge

Predicting Prompt Quality from Obfuscated Token Sequences

Team Name: Encoder-Decoder

Members: Himanshu Maurya & Sampriti Mohanty

Repository: https://github.com/HimanshuMaurya12/Udhgam-ML-Challenge-Encoder-Decoder

Best Score (MAE): 0.14689

---

1. Abstract

This report details the methodology used by Team *Encoder-Decoder* in the Udhgam 2.0 ML Challenge. The task required predicting a continuous quality score for LLM prompts based solely on obfuscated token sequences, preventing the use of pre-trained language models (PLMs) like BERT or GPT. Our winning solution employs a custom **CRNN-Attention Hybrid architecture** trained from scratch. Key contributions include the injection of statistical meta-features, the use of Iterative Pseudo-Labeling to refine decision boundaries, and Monte Carlo Dropout for robust Test-Time Augmentation (TTA).

2. Problem Statement & Analysis

The challenge presented a supervised regression problem where the input features were sequences of scrambled integer token IDs.

- **Constraint:** The vocabulary was obfuscated, neutralizing semantic transfer learning from standard PLMs.
- **Insight:** While semantic meaning was hidden, **structural patterns** (syntax trees, repetition, constraint formatting) and **local n-gram statistics** remained intact. Our model was designed to extract these structural signals explicitly.

3. Methodology

3.1 Feature Engineering (Meta-Features)

Since the model could not infer "meaning" from token IDs alone, we engineered a set of statistical features to provide a "global view" of the prompt's complexity and structure. These were concatenated with the deep learning features at the final dense layer:

- **Sequence Length (L):** Proxy for prompt detail and specificity.
- **Unique Token Count (U):** Measure of vocabulary richness.
- **Lexical Diversity (U/L):** Ratio indicating repetition versus variety.
- **Token Variance (σ):** Standard deviation of IDs to capture spread.
- **Boundary Tokens:** First and last token IDs, capturing potential framing patterns (e.g., specific start/end tokens used in high-quality templates).

3.2 Model Architecture: CRNN-Attention

We designed a hybrid neural network that combines the strengths of Convolutional and Recurrent architectures:

1. **Learned Embeddings:** A trainable embedding layer ($d=256$) initialized from scratch to map obfuscated IDs to dense vectors.
2. **1D-Convolution (CNN):** A 1D convolutional layer with a kernel size of 3 acts as a feature extractor for local **n-gram patterns** (texture), identifying specific token combinations that correlate with quality.
3. **Bidirectional GRU (Bi-GRU):** A 2-layer Bi-GRU processes the sequence to capture **long-range dependencies** and the global flow of the prompt.
4. **Attention Mechanism:** A custom attention layer computes a weighted sum of the GRU hidden states, allowing the model to dynamically focus on critical segments of the input while suppressing noise/padding.
5. **Dense Head:** The context vector from the Attention layer is concatenated with the normalized Meta-Features and passed through a final regressor with Sigmoid activation.

3.3 Training Strategy

- **Loss Function:** We employed **SmoothL1Loss (Huber Loss)** with $\beta=0.01$. This provided stability by being less sensitive to outliers than MSE, while offering smoother convergence near zero error compared to standard L1 Loss.
- **Optimization:** The model was optimized using AdamW with a One-Cycle Learning Rate scheduler to ensure rapid convergence and regularization.
- **Pseudo-Labeling (Self-Training):** To bridge the gap between the training and test distributions, we employed iterative pseudo-labeling. We generated soft labels for the test set using our best-performing model ensemble and retrained the architecture on this combined dataset. This significantly smoothed the decision boundary.

4. Inference: Test-Time Augmentation (TTA)

A single model prediction can be noisy due to the stochastic nature of neural networks. To mitigate this, we implemented **Monte Carlo Dropout** during inference.

- **Technique:** Instead of disabling Dropout at test time, we kept it active and performed **10 forward passes** for each test example.
- **Result:** The final prediction is the average of these 10 passes. This simulates an ensemble of 10 slightly different sub-networks, reducing model variance and improving the Mean Absolute Error (MAE) by approximately 0.001.

5. Results

Our final submission achieved a Mean Absolute Error (MAE) of **0.14689** on the Public Leaderboard. The combination of structural meta-features and the CRNN-Attention architecture proved robust against the obfuscation barrier, while TTA provided the necessary precision for the final ranking.

6. Resources

- **Code Repository:** https://github.com/HimanshuMaurya12/Udhgam-ML-Challenge-Encoder-Decoder
- **License:** MIT License

References

- *Srivastava, N., et al. "Dropout: A simple way to prevent neural networks from overfitting." JMLR 2014.*
- *Bahdanau, D., et al. "Neural Machine Translation by Jointly Learning to Align and Translate." ICLR 2015.*