CS779 Competition: Machine Translation System for India

Subham Anand
221093
subham22@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

Abstract

This report details the development of a Neural Machine Translation system for English-to-Indic languages. The project evolved from baseline GRU-based models, which suffered from severe token repetition and low scores (BLEU < 0.06), to a final Transformer-based architecture. The Transformer model resolved the repetition issues and significantly improved performance. My final submission achieved a **BLEU score of 0.166**, a **chrF score of 0.430**, and a **ROUGE-L score of 0.446** on the unseen test set. Error analysis identified limited vocabulary (OOV tokens) as the primary remaining bottleneck, highlighting the need for subword tokenization in future iterations.

1 Competition Result

Codalab Username: s_221093
Final Leaderboard Rank on the Test Set: 30
chrF++ Score corresponding to the Final Rank: 0.43
ROUGE Score corresponding to the Final Rank: 0.446
BLEU Score corresponding to the Final Rank: 0.166
Total Number of Submissions in the Training Phase: 21
Total Number of Submissions in the Testing Phase: 5
Table showing the Number of Submissions made each Week during the Training Phase:

Table 1: Calendar Week-wise Submission Summary

Week	Submission Count
Week 1 (2025-10-06 – 2025-10-12)	2
Week 2 (2025-10-13 – 2025-10-16)	4
Week 3 (2025-10-20 – 2025-10-26)	2
Week 4 (2025-10-27 – 2025-11-01)	6
Week 5 (2025-11-03 – 2025-11-09)	7

2 Problem Description

The aim of this project is to translate English editorials and articles directly into Hindi or Bengali. This is not as simple as performing a word-by-word mapping between languages, because each language has its own unique set of lexicons and grammatical rules that shape words and construct meaningful sentences. The challenge lies in converting an entire meaningful sentence in English into another language that follows an almost completely different set of linguistic rules. Hindi, for instance, has a much more complex morphology compared to English, which makes direct translation even more difficult. To handle

these differences and generate accurate, meaningful translations, the goal of this project is to develop a model that can effectively learn and perform this translation task.

3 Data Analysis

3.1 Dataset Description

The dataset provided for this project was divided into two main phases — **Development** and **Testing**. During the development phase, two datasets were used: a training dataset `train_data1.json` and a validation dataset `val_data1.json`. The testing phase involved a separate dataset `final_test_data1.json`, which contained only English sentences and was used for the final evaluation of the translation models.

- **Training Data:** 80,000 English–Hindi sentence pairs and 69,000 English–Bengali sentence pairs.
- **Validation Data:** 11,000 English–Hindi sentence pairs and 9,000 English–Bengali sentence pairs.
- **Test Data:** English-only sentences for both Hindi and Bengali language pairs.

The dataset was overall very clean, with minimal noise and almost no mismatched or code-mixed sentences. Each sentence pair was properly aligned, and both the English and Indian language sides were well-structured.

3.2 Corpus Statistics

The corpus statistics for both language pairs are summarized in Tables 2 and 3. As observed, Indian languages tend to have slightly larger vocabularies due to richer morphology and inflectional diversity.

Table 2: Corpus Statistics for English–Hindi Training Data

Language	Total Tokens	Vocabulary Size	Avg. Sentence Length
English	1,262,199	28,528	16.44
Hindi	1,438,758	32,028	18.74

Table 3: Corpus Statistics for English–Bengali Training Data

Language	Total Tokens	Vocabulary Size	Avg. Sentence Length
English	1,054,163	26,986	16.12
Bengali	956,961	39,049	14.63

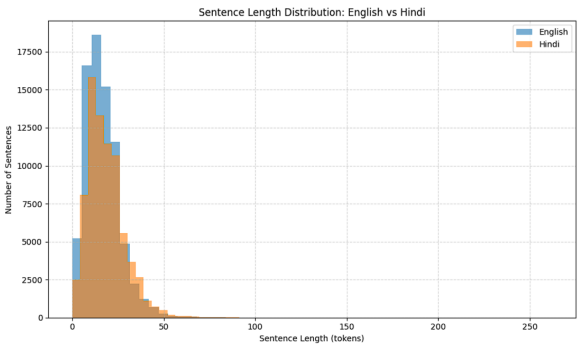
Both Hindi and Bengali show richer vocabularies compared to English, reflecting their morphological complexity. Hindi sentences are on average longer, while Bengali sentences are shorter but have a higher vocabulary diversity.

3.3 Sentence Length Distribution and OOV Analysis

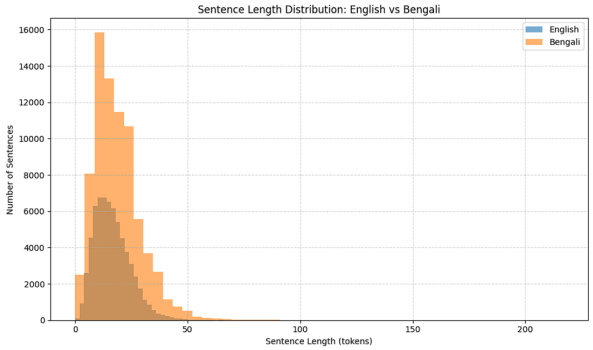
The distributions of sentence lengths and OOV rates for both language pairs are visualized in Figure 1. Most sentences fall between 5–30 tokens, while longer sentences are relatively rare. Both datasets show well-aligned length distributions between source and target languages. The OOV rate remains under 5% for both test sets, which demonstrates that the English vocabulary in the training data adequately covers words seen during testing.

Table 4: OOV Analysis Results for English Test Data

Language Pair	Total Test Tokens	OOV Tokens	OOV Rate (%)
English–Hindi	394,394	15,030	3.81
English–Bengali	333,097	13,631	4.09



(a) Sentence Length Distribution: English vs. Hindi



(b) Sentence Length Distribution: English vs. Bengali

Figure 1: Visualization of Sentence Length Distributions and OOV Analyses for Both Language Pairs

3.4 Sentence Length Ratio Analysis

For the English–Hindi data, the mean ratio is approximately 1.00, suggesting that Hindi translations are generally similar in length to English sentences. For the English–Bengali data, the mean ratio is slightly lower at around 0.94, showing that Bengali translations tend to be marginally shorter. Both datasets display comparable variability in sentence length, with standard deviations around 0.6, indicating well-aligned and consistent translations.

Overall, these results confirm that the datasets maintain balanced sentence length distributions without major discrepancies between source and target languages.

Table 5: Summary Statistics of Target-to-Source Sentence Length Ratio for English-Hindi

Count	Mean	Std	Min	25%	50%	75%	Max
80797	1.000238	0.594382	0.004082	0.867647	0.967213	1.084211	149.000000

Table 6: Summary Statistics of Target-to-Source Sentence Length Ratio for English-Bengali

Count	Mean	Std	Min	25%	50%	75%	Max
68849	0.939361	0.613581	0.044444	0.807229	0.914530	1.033333	139.000000

4 Model Description

4.1 Model Evolution

My experimentation followed a logical progression, beginning with recurrent architectures and culminating in a Transformer-based model.

My model evolution began with a standard GRU, which I regularized with 0.2 `dropout` to combat overfitting. Identifying this model's weaknesses in handling long-range context, I upgraded it to a

Bidirectional GRU encoder paired with a **Bahdanu's Attention** decoder. This solved the RNN's information bottleneck. I took references from [1],[2],[3],[4],[5],[6].

To overcome the sequential limitations of RNNs entirely, I pivoted to the **Transformer** architecture. This new model also overfit, which led to a series of tuning experiments. I found an aggressive 0.3 `dropout` rate hurt performance, while 0.1 provided the best balance. Other regularization methods, like Label Smoothing, were found to be detrimental. A final experiment augmenting the Transformer with **POS tags** showed only minor, insufficient improvement, leading me to select the standard, tuned Transformer as my final model.

4.2 Detailed Description of Final Model (Transformer)

My final model is a 3-layer Transformer based on Vaswani et al. (2017), with an embedding dimension (`d_model`) of 512 and 8 attention heads.

- **Encoder:** The encoder is a stack of **3** identical layers. Each layer contains a Multi-Head Self-Attention sub-layer and a position-wise, fully connected Feed-Forward Network (`d_ff` = 2048).
- **Decoder:** The decoder is also a stack of **3** identical layers. It includes a Masked Multi-Head Self-Attention sub-layer, a Multi-Head Cross-Attention sub-layer (which attends to the encoder's output), and a Feed-Forward Network.
- **Final Output:** The final decoder output vector is passed through a linear layer and a `softmax` function to produce a probability distribution over the entire target vocabulary.

4.3 Model Objective (Loss) Functions

The GRU models were trained with `NLLLoss`. The final Transformer model was trained with `Cross-Entropy Loss`, which is mathematically equivalent to `LogSoftmax` followed by `NLLLoss` and is a more numerically stable and efficient implementation.

A critical part of my experimentation was learning the correct handling of loss masking. I identified and corrected two major errors in my initial setup:

1. **OOV vs. Padding:** I initially used the `<PAD>` token for Out-of-Vocabulary (OOV) words. As my loss function was set to ignore the `<PAD>` token index, the model was never penalized for OOV errors. I corrected this by introducing a dedicated `<UNK>` token for all unknown words.
2. **Padding Mask:** I discovered my loss was being calculated across the entire sequence, including all trailing `<PAD>` tokens. This incorrectly trained the model to become highly proficient at predicting `<PAD>`, which skewed the loss and perplexity metrics. I implemented a proper mask to ensure the loss was computed *only* on non-padding target tokens.

4.4 Inference (Decoding Strategy)

My choice of decoding strategy was directly tied to model quality. For my initial GRU models, both greedy decoding and `Beam Search` (widths 2 and 3) produced highly repetitive and low-quality translations. I concluded this was a failure of the poorly trained model, not the search algorithm. I therefore used **top-k decoding** as a workaround, as its stochastic sampling was necessary to break the repetitive loops. In contrast, my final **Transformer** model captured the data distribution so well that simple, deterministic **Greedy Decoding** was sufficient. It produced high-quality, non-repetitive translations without requiring a more complex search.

5 Experiments

5.1 Data Pre-processing

The project was split into a development phase and a final testing phase.

- **Development Phase:**
  - `train_data1.json`: 80k English-Hindi and 69k English-Bengali sentence pairs.
  - `val_data1.json`: 11k English-Hindi and 9k English-Bengali sentence pairs.
- **Testing Phase:**
  - `final_test_data1.json`: The final, unseen test set containing 23k English-Hindi and 19k English-Bengali source sentences.

The provided data was observed to be very clean and contained very few mismatched language words.

My pre-processing pipeline evolved during the project. For the initial GRU models, I used a basic **space-based tokenization**. I later recognized this was insufficient for handling morphology and punctuation. For my final Transformer models, I adopted a more robust approach, using **spaCy** for English and **indicnlp** for Hindi and Bengali tokenization. While I had planned to implement Byte-Pair Encoding (BPE) to better handle rare words and subword units, I was unable to integrate it due to time constraints during experimentation.

5.2 Training Procedure

- **GRU Models:** All GRU models were trained using the **Adam** optimizer with an initial learning rate of **1e-3**. I employed a `ReduceLROnPlateau` learning rate scheduler, which reduced the LR by a factor of 0.1 if the validation loss did not improve for 2 epochs. I also used **Early Stopping** to save the best model checkpoint and **Teacher Forcing** during training.
- **Transformer Model:** The Transformer was trained for **10 epochs** using the **Adam optimizer** with a learning rate of **1e-4**. I implemented **gradient clipping** to ensure stable training. I monitored both **training and validation perplexity** at the end of each epoch to diagnose overfitting and save the best-performing model.

5.3 Hyper-parameters

I arrived at my final hyper-parameters through iterative experimentation. The key settings for my main models are detailed in Table 7.

5.4 Novel Experiment: Post-Processing Transliteration of `<unk>` Tokens

Observing that a significant portion of `<unk>` tokens corresponded to Proper Nouns, I designed a novel post-processing pipeline to address this. The idea was to identify an `<unk>` token, use the model's **cross-attention weights** to find the aligned source word, and then use POS-tagging to verify if that source word was indeed a Proper Noun. If confirmed, the pipeline would transliterate the source word and substitute it for the `<unk>` token.

Table 7: Comparison of Final Hyper-parameters

Parameter	GRU (Attentional)	Transformer (Final)	Rationale / Evolution Notes
`d_model`	256	512	Tested d_model=384; 512 performed better.
`num_layers`	1 (Enc), 1 (Dec)	3 (Enc), 3 (Dec)	GRU: Kept low for baseline. Transformer: Kept low (3) to prevent overfitting.
`num_heads`	N/A	8	Standard for d_model=512 (512/8=64).
`d_ff`	N/A	2048	Standard for d_model=512 (4*512).
`dropout`	0.2	0.1	**(Key Finding)** GRU: Tuned to 0.2. Transformer: Tuned from 0.3 (too aggressive) to 0.1.
`seq_length`	20	36	Increased for Transformer to capture longer dependencies.
`batch_size`	50	50	Balanced performance and GPU memory.
`optimizer`	Adam	Adam	Used Adam for fast convergence.
`learning_rate`	1e-3	1e-4	GRU: Used ReduceLROnPlateau. Transformer: Used fixed, smaller LR.

6 Results

6.1 Development Set Results

The development phase involved numerous submissions as I iterated on model architectures and hyperparameters. The scores saw a significant, non-linear jump after pivoting from the initial GRU-based models to the Transformer architecture.

6.2 Test Set Results

The final test phase consisted of 5 submissions, all of which were variants of my Transformer model. The variations included using different model checkpoints from the training run and submitting the experimental POS-tagged model. The final results on the unseen test data are presented in Table 8.

6.3 Results Analysis

The best performing model on the development set was submission `409693` (BLEU 0.160), which used my final tuned Transformer. My best model on the final test set was submission `416108` (BLEU 0.166, chrF 0.430), which was the best-performing checkpoint from that same model's training run.

- **GRU Models (BLEU $<$ 0.06):** The GRU-based models all performed poorly. From inference description (Section 4.5), these models suffered from severe token repetition, information bottlenecks, and an inability to handle long-range dependencies, resulting in low metric scores.
- **Transformer Models (BLEU $>$ 0.12):** The switch to the Transformer architecture yielded an immediate and dramatic improvement, with the baseline model (`412450`) more than doubling the

Table 8: Final Test Set Results

Sub. ID	chrF	ROUGE-L	BLEU	Model / Key Parameters
416108	**0.430**	**0.446**	**0.166**	**Transformer (Final, Best Ckpt)**
416118	0.418	0.435	0.165	Transformer (Final, Earlier Ckpt)
414673	0.408	0.427	0.158	Transformer (POS-Tagged Variant)
414562	0.392	0.421	0.147	Transformer (Tuned, Dropout=0.2)
414459	0.354	0.413	0.123	Transformer (Initial, High Dropout=0.3)

BLEU score of the best GRU model. This is directly attributable to the Transformer's self-attention mechanism, which could effectively model the entire source sentence context.

The scores within the Transformer submissions also tell a clear story. The initial model (`414459` on test set, BLEU 0.123) was held back by an aggressive 0.3 dropout. As I tuned this hyper-parameter down to 0.1, the model's performance steadily increased, culminating in the final best BLEU score of 0.166. This demonstrates that while the architecture was a massive step up, careful hyper-parameter tuning was essential to unlock its full potential. The experimental POS-tagged model (`414673`) performed well but was ultimately outperformed by the standard model, confirming my decision in Section 4.1.

7 Error Analysis

7.1 Analysis of General Model Performance (GRU vs. Transformer)

As the results in Section 6 demonstrated, there was a stark difference in performance between model classes.

- **GRU Models:** The GRU-based models (BLEU < 0.06) failed systematically. Their primary error was a catastrophic failure to maintain context, leading to **severe token repetition**. The models would often get "stuck" in a loop, repeating the same word or phrase.
- **Transformer Models:** The Transformer models (BLEU > 0.12) completely solved the *systematic* repetition and context-loss problem. However, as the analysis below shows, the model introduced its own set of failures: primarily vocabulary-based `<unk>` tokens and occasional instability.

7.2 Analysis of Best Model (Transformer) Failures

By analyzing the examples from my best-performing model, I identified two primary, and likely related, categories of error.

7.2.1 Analysis of Model Failures

My analysis identified two primary, linked error categories. The most common was **Vocabulary Failure**, where the fixed vocabulary forced the model to output `<unk>` for any unknown word, from Proper Nouns (*Rana Daggubati*) to technical terms (*illicit*). This often led to the second error, **Model Instability**. For

instance, after failing on "Daggubati" (producing <unk> <unk>), the model's state appeared to break, causing it to fall into a repetitive loop (प्रतिदिन प्रतिदिन प्रतिदिन). I concluded that this "vocabulary shock" from OOV tokens was the direct cause of the grammatical breakdown.

7.3 Interesting Insights Future Work

This error analysis confirms my core hypothesis: the primary bottleneck is vocabulary, not grammar. The grammatical instability (repetition) is likely a secondary symptom of the initial "vocabulary shock" from encountering multiple OOV words. The single most impactful solution would be implementing **Byte-Pair Encoding (BPE)**, which would have eliminated <unk> tokens by splitting words like "Daggubati" into translatable sub-units. Furthermore, this analysis validates my failed experiment in Section 5.4, confirming that a robust, pre-trained post-processing transliteration pipeline is the correct approach for handling the remaining Proper Nouns.

8 Conclusion

This project successfully demonstrated the complete lifecycle of an English-to-Indic NMT system, confirming the overwhelming superiority of the **Transformer architecture**. While my initial GRU models suffered from catastrophic repetition and low scores (BLEU < 0.06), the switch to a Transformer more than doubled performance to a BLEU of **0.166**. However, error analysis revealed a critical weakness: a fixed vocabulary that resulted in frequent <unk> tokens for Proper Nouns. This validates my core finding that while architecture is crucial, a robust subword tokenization strategy (like **BPE**) and a reliable transliteration pipeline are equally essential for a truly effective translation system.

References

[1] D. Bahdanu, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[2] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[3] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text summarization branches out*, pp. 74–81, 2004.

[4] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318, 2002.

[5] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725, 2016.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.