CS779 Competition: Machine Translation System for India

Arnab Datta
240185
arnabd24@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

November 14, 2025

Abstract

This competition was organized as a part of the course CS779 (Statistical Natural Language Processing). The goal of the competition was to develop robust neural machine translation systems, capable of translating English to Indian Languages. For the competition, the model was tested on two Indian Languages, namely Hindi/हिन्दी and Bengali/বাংলা . In this competition, I used SentencePiece style BPE subword tokenization algorithm for both English and the two Indian Langugages. I primarily used a vocabulary size of 15k tokens. Various encoder-decoder models, including seq-2-seq and transformer based models, were tried out during the Development Phase of the competition. The best performing model, among the ones that I have tried, is a Transformer with 333-dimensional token embeddings plus sinusoidal positional encodings, 4 encoder and 4 decoder layers using 9-head multi-head attention and ReLU feed-forward blocks, trained end-to-end via pad-aware token-level cross-entropy on shifted targets and decoded with beam search. With this model, I achieved **second** rank in the testing phase leaderboard and **fifth** rank in the testing phase leaderboard.

1 Competition Result

- **Codalab Username:** a_240185
- **Final Leaderboard Rank on the Test Set:** 5
- **chrF++ Score corresponding to the Final Rank:** 0.429
- **ROUGE Score corresponding to the Final Rank:** 0.474
- **BLEU Score corresponding to the Final Rank:** 0.209
- **Total Number of Submissions in the Training Phase:** 25
- **Total Number of Submissions in the Testing Phase:** 5
- **Table showing the Number of Submissions made each Week during the Training Phase:**

Table 1: Number of Submissions during Training

Week	Dates	Submissions
Week 1	6th–12th October	2
Week 2	13th–19th October	2
Week 3	20th–26th October	3
Week 4	27th October–2nd November	6
Week 5	3rd November	12

2 Problem Description

Nowadays, it is no longer necessary to use a bilingual dictionary when translating from one language to another – upon encountering words in a foreign language, one can instantly acquire the translation by using online translation services. An automatic method of translating one language to another, this is essentially what Machine Translation is. It is now so widespread to utilise machine translation that Google Translate claims to translate more than 100 billion words every day.

In this challenge, hosted on CodaBench, the goal is to translate sentences/phrases in source language (English) to the target languages (Bengali, Hindi). The models must be trained from scratch (no pretrained models allowed) to translate English sentences to their Indian language counterpart. All model implementations should be done in PyTorch. Transformers-based models are allowed provided that they are trained from scratch.

There are two phases in this competition—Development and Testing. In the first phase, you will train a model and validate it using a validation set. In the second phase, a test set will be provided for you to evaluate the model's performance on test data.

The model should be trained on the file "train_data1.json". The use of external datasets is forbidden. Your submission will be evaluated based on "test_data1.json" that will be provided later. During development phase, submissions would be made on "val_data1.json".

The submissions will be evaluated based on corpus-level BLEU-4 score, ROUGE-L score and chrF++ score.

3 Data Analysis

The training dataset consist of 68849 pairs of English-Bengali Sentences and 80797 pairs of English-Hindi Sentences. The testing data comprises 19672 English-Bengali pairs and 23085 English-Hindi pairs. All datasets follow a Zipfian distribution.

3.1 Training Dataset

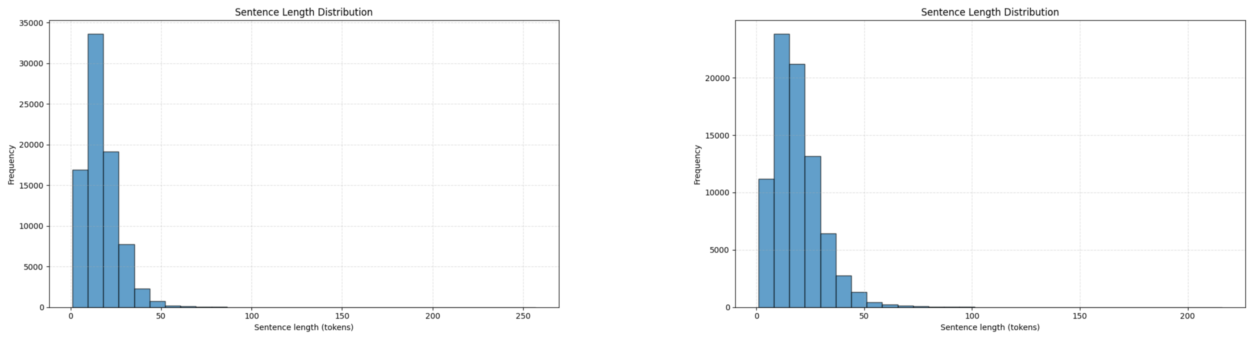
In the training dataset, the Bengali dataset had 82.7% characters in the Bengali script Unicode range, whereas only 78.1% of the characters in the Hindi dataset were in the Devanagari scipt Unicode range. This indicates that the Hindi dataset has more English Words as is, rather than being transliterated to Hindi. This is also supported by the observation that mostly Latin numerals (0-9) are used in the Hindi dataset instead of Devanagari numerals (०-९). The entropy of Bengali dataset (~12.7) is much higher than that of the Hindi dataset (~11.0), while the entropy of English (~11.2) is similar across both sentences. There are ~11.6k unique words in Bengali dataset, but there are only ~9.6k unique words in the Hindi dataset. This reflects the richer vocabulary and morphological richness of Bengali Language. This reflects the greater lexical and morphological richness of the Bengali language. This can be partly attributed to the fact that, in Bengali, the form of a noun often changes according to its grammatical function, whereas in Hindi, the original noun form typically remains unaltered. For example, the English phrase "in the house" is expressed in Bengali as বাড়িতে (baṛite), where the noun বাড়ি (baṛi, "house") takes the locative suffix –তে (-te). In contrast, in Hindi, the same meaning is conveyed as घर में (ghar mein), where the noun घर (ghar, "house") retains its base form, and the postposition में (mein) indicates location. The average length of English sentences is nearly 17 words for both Language datasets. However on an average Bengali sentences (~14 words) are much shorter compared to Hindi (~19 words). This is expected due to the above mentioned fact. The Hebrew punctuation "׀" appears many places in the Bengali dataset instead of the Bengali fullstop "৷". A preprocessing function was used to normalise unicode and fix LATEXstyle quotations.

3.2 Training and Validation dataset

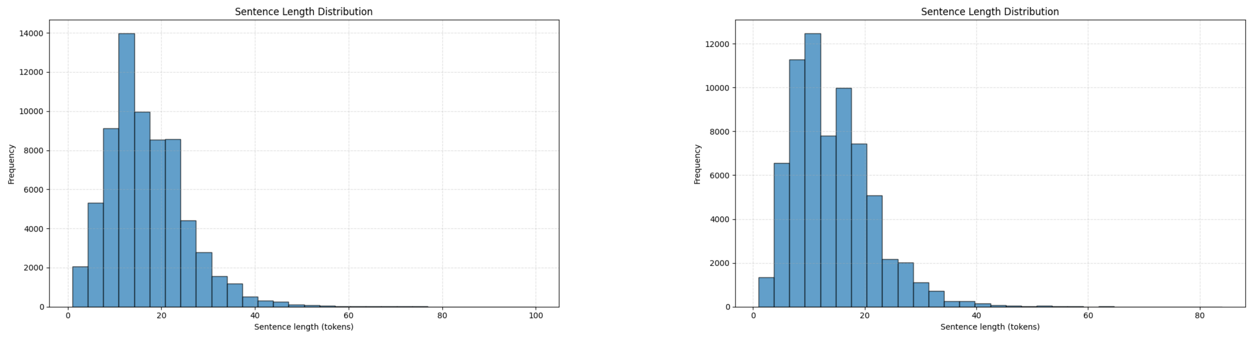
The validation and testing dataset is very similar to the training dataset, in terms of entropy and average sentence length.

Table 2: Comparison of Hindi and Bengali Datasets on Key Metrics

Dataset Split	**Bengali**		**Hindi**	
	Avg. Sent. Length	**Entropy**	**Avg. Sent. Length**	**Entropy**
Train	16.82	11.17	17.07	11.20
Validation	16.88	10.88	17.08	10.92
Test	16.93	11.01	17.08	11.06



(a) Increase in average sentence length in Hindi compared to English.



(b) Decrease in average sentence length in Bengali compared to English.

Figure 1: Overall comparison of sentence lengths in Hindi and Bengali datasets.

4 Model Description

4.1 Model Evolution and Key Learnings

Our model evolved through several stages, each addressing limitations uncovered in prior experiments. We began with a toy GRU encoder–decoder (single layer, small hidden size) to rapidly validate the data pipeline and serialization process. This baseline produced extremely low BLEU and chrF++ scores (BLEU ≤ 0.05) and frequent truncation, revealing three critical issues: inadequate handling of long-distance dependencies (word-order divergences from English to Indic), vocabulary sparsity due to word-level tokenization, and vanishing gradients for longer sentences.

Subsequent multilayer and bidirectional GRU configurations improved source context capture and reduced truncation slightly. Bidirectionality aided initial alignment, but deeper recurrent stacks increased training time without proportional performance gains. Interestingly, GRUs tended to outperform LSTMs on single-sentence translation tasks, benefiting from simpler gating and faster convergence, though both remained limited by sequential processing and difficulty modeling long-range reordering.

Integrating subword tokenization (byte-level BPE, $\approx$15k tokens per language) following Sennrich et al. (2016) drastically reduced out-of-vocabulary rates and stabilized training. Replacing GRUs with LSTMs (and bidirectional encoders) yielded marginal adequacy gains—LSTM gating alleviated some gradient vanishing but still failed to handle complex reorderings and sometimes hallucinated under noisy conditions.

Introducing Bahdanau-style additive attention significantly improved coverage and reduced omissions, confirming that attention is essential for effective sequence modeling. However, single-head additive attention displayed capacity limits with diffuse distributions and weak handling of concurrent reordering events. Refining teacher forcing (predicting the next token conditioned on gold prefixes) with padding-aware loss stabilized training further; scheduled sampling was rejected after inducing instability for minimal potential gain.

Transitioning to the Transformer architecture (Vaswani et al., 2017) marked a major leap. Multi-head self-attention captured multiple alignment patterns simultaneously—named-entity copying, verb/object reordering, and numeral placement—leading to immediate gains (dev BLEU ≈ 0.17). The medium configuration (333-dimensional, 9-head, 4 encoder / 4 decoder layers) achieved the best trade-off between quality and efficiency. Larger and deeper variants (384d/12-head/5/5 and 360d/10-head/5/5) underperformed within the 12-hour 2$\times$T4 time budget, suggesting over-regularization and suboptimal warmup scheduling. Beam search decoding (width 5–8) with mild length penalties (0.7–0.8) and no-repeat n-gram constraints outperformed greedy decoding, improving adequacy and reducing repetitions.

Overall, the evolution demonstrated clear lessons: recurrent models struggle with long dependencies and morphological richness, while Transformers excel at parallel alignment and context integration. For our dataset, moderate width, moderate depth, and constant learning rate produced the most efficient and robust translation quality.

4.2 Inspirations and Citations

Our architecture and training strategy were informed by prior research: Seq2Seq with RNN encoder–decoders [1, 2]; attention mechanisms [3, 4]; subword segmentation [5]; and the Transformer architecture [6]. Regularization and optimization choices were influenced by the original Transformer schedule (inverse square root), though simplified in our final configuration due to computational constraints.

4.3 Final Model Architecture

The final production model, used for test submissions, consists of four Transformer encoder layers and four decoder layers. Each encoder layer includes multi-head self-attention (9 heads) and a position-wise feedforward block (ReLU activation; hidden dimension $1332 = 4 \times d$). Each decoder layer includes masked self-attention, encoder–decoder cross-attention, and feedforward modules of the same size. Separate source (English) and target (Bengali/Hindi) embedding matrices (dimension 333) are combined with sinusoidal positional encodings. Special tokens `<s>`, `</s>`, `<pad>`, and `<unk>` are reserved in each vocabulary, and byte-level storage ensures lossless reconstruction.

Dropout of 0.15 is applied to embeddings, attention outputs, and residual connections, with post-residual LayerNorm following PyTorch's default Transformer design.

4.4 Objective (Loss) Function

The training objective is token-level categorical cross-entropy:

$$\mathcal{L} = \frac{1}{N_{\text{nonpad}}} \sum_t \text{CE}(\text{logits}_t, y_t)$$

where pad positions are masked (using `ignore_index`). The loss is normalized by the count of non-pad tokens to maintain consistency across variable-length batches. Weight decay (5×10^{-6}) provides mild L2 regularization, and gradient clipping ($\|g\|_2 \leq 1.0$) prevents exploding gradients early in training. Label smoothing ($\epsilon = 0.1$) was tested but not retained due to limited improvement. A constant learning rate (1×10^{-4}) outperformed warmup–decay schedules within the time budget.

4.5 Inference and Decoding Strategy

Inference evolved from greedy decoding—used initially for correctness checks—to beam search (width 5–8), which improved fluency and adequacy. A length penalty of 0.7–0.8 counteracted beam's preference for shorter sequences. Minimum length (5 tokens), no-repeat n-gram constraints (2–3), and a repetition penalty of 1.2 mitigated degeneracy and over-copying of entities and numerals. Temperature remained fixed at 1.0 to preserve deterministic outputs. Beam search with mild length encouragement proved especially effective for Indic targets, where post-positions and inflectional endings require longer, context-aware sequences.

4.6 Summary of Architectural Rationale

The final Transformer achieves strong adequacy and fluency under tight resource constraints. Its 9-head attention configuration balances alignment diversity and parameter efficiency; moderate depth (4/4) avoids over-regularization and excessive padding; feedforward width ($4 \times d$) captures morphological structure without memory overhead; and separate vocabularies minimize cross-script interference.

5 Experiments

5.1 Data Pre-processing

In our machine translation system, we have a preprocessing pipeline that is identical at training and inference to avoid train–serve skew. We first standardize the source text with Unicode NFKC normalization and casefolding to collapse compatibility variants and English casing, reducing sparsity and normalizing punctuation forms. We then collapse repeated whitespace. Now we apply SentencePiece Byte Pair Encoding Subword Tokenization, as developed in Assignment 2, to each language separately. For this, data is prepared by marking word boundaries with the dedicated symbol "_", which ensures clean detokenization. The tokenization is script-aware: we use separate subword vocabularies for English and each target language, store tokens as bytes for round-trip fidelity, and define reserved tokens <pad>, <unk>, <s>, </s>. We deliberately preserve punctuation and numerals—including native Indic digits (০–৯, ०–९) – because they carry meaning and are often copied or transliterated. For the Transformer, we wrap sequences with <s> and </s>, pad or truncate to a fixed maximum length of 150 subwords for efficient batching on Kaggle T4 GPUs, and construct key-padding and causal masks. The entire dataset is preprocessed once (takes ~30 mins), and the saved .pkl files are uploaded on Hugging Face. With this setup, the trained can be started directly after loading these preprocessed files.

5.2 Training procedure

We trained our NMT systems end to end in vanilla PyTorch on Kaggle Tesla T4 GPUs, using a fixed subword sequence length of 150 and a batch size of 64 to balance coverage and memory.

After validating a small baseline (d_model 300, 8 heads, 4/4 layers) for pipeline sanity, we converged on a medium Transformer as the primary model: d_model 333 with 9 attention heads, 4 encoder and 4 decoder layers, feedforward dimension 1332, and dropout 0.15. This configuration consistently improved adequacy and fluency for Indic targets without the compute overhead of a larger variant. Explorations with a larger feedforward stack (d_model 360, 10 heads, 5/5 layers) yielded only a slightly inferior model at roughly 22% higher time cost, while a high-dropout setting (0.25) slowed convergence and plateaued higher. The odd head count triggers a nested tensor warning in PyTorch but has no impact.

Optimization used separate Adam optimizers for encoder and decoder with a constant learning rate of 1e-4, betas at 0.9/0.999, and light weight decay at 5e-6 to steady embeddings. We trained with token-average cross-entropy (ignore_index on pad), gradient clipping at 1.0 each step for stability, and no scheduler in production; warmup and cosine decay were tested but did not lead to better scores. Although the notebook defines gradient accumulation, effective training ran without relying on it given our batch and length constraints. We configured it to run for 50 epochs. On dual T4s, the total notebook run time on Kaggle was ~10hrs, with each model training on its separate T4 GPU. Runs are seeded (42) for reproducibility.

For decoding, we use beam search with width 5 in batch validation and width 8 interactively, a length penalty of 0.7–0.8 to counter shortness, minimum target length 5, no-repeat-ngram size 2–3, a mild repetition penalty of 1.2, and temperature 1.0. This setup reduces trivial loops, preserves numeric and punctuation fidelity, and yields stable outputs across domains.

6 Results

Table 3: Model performance during training and testing

	Model	**chrF++**	**ROUGE**	**BLEU**
TRAIN	*Toy Model(GRU)*	0.178	0.283	0.032
	Multilayered BiGRU	0.092	0.205	0.012
	GRU with SP BPE	0.160	0.288	0.040
	LSTM with SP BPE	0.104	0.243	0.023
	BiLSTM with SP BPE	0.086	0.146	0.012
	LSTM with Teacher Forcing	0.160	0.288	0.040
	BiLSTM with Attentiion	0.192	0.322	0.049
	300-dim Transformer, 8-head, 4/4 (Greedy)	0.443	0.443	0.174
	Final Transformer	**0.435**	**0.464**	**0.196**
	Larger 384-dim, 12-head, 5/5 model	0.345	0.381	0.119
TEST	Larger 360-dim, 10-head, 5/5 model	0.435	0.462	0.191
	Final Transformer	**0.429**	**0.474**	**0.209**

I trained the models on Kaggle (2×T4 GPUs, session limit 12 hours). Our best model was a medium Transformer: 333 hidden size, 9 attention heads, 4 encoder and 4 decoder layers, feed-forward 4×d, dropout 0.15. We used Adam with a constant learning rate of 1e-4, light weight decay (5e-6), and gradient clipping at 1.0. This setup beat a larger 360-dim, 10-head, 5/5 model and 384-dim, 12-head,5/5 model that used warmup/decay on both the dev and test sets. The win comes from a better match between model size and data, faster learning without a warmup delay in a short training window, enough heads to handle long-range reordering, and a dropout level that prevents overfitting without slowing learning. We trained for 50 epochs to fit the time budget.

7 Error Analysis

Our error analysis shows a clear progression across architectures: RNN/GRU/LSTM baselines performed poorly overall (BLEU ≤ 0.049), yet within this group, GRUs tended to outperform LSTMs on single-sentence translation tasks. This likely stems from GRUs' simpler gating mechanism and faster convergence, which make them more efficient for shorter sequences where long-term dependencies are limited. The first Transformer (300-dimensional, 8 heads, 4/4) with greedy decoding achieved a large improvement (dev BLEU ≈ 0.174), demonstrating the benefit of self-attention for capturing broader contextual relationships. The final 333-dimensional, 9-head, 4/4 Transformer with beam search performed best on both development and test sets (dev BLEU ≈ 0.196, chrF++ ≈ 0.435; test BLEU ≈ 0.209, chrF++ ≈ 0.429, ROUGE 0.474). Larger and deeper variants with warmup (384-dimensional, 12-head, 5/5; 360-dimensional, 10-head, 5/5) underperformed (e.g., test BLEU 0.119 and 0.191), indicating capacity overshoot and schedule inefficiency within the 12-hour 2×T4 limit. The best model still makes systematic errors: inconsistent transliteration of named entities, digit and unit mismatches, occasional punctuation imbalance, and agreement issues in long sentences—often reinforced by noisy Bengali pairs that are not literal translations/transliterations.

8 Conclusion

Table 4: Parameters of best model

Adam lr=1e-4	333 hidden size	9 attention heads
4 encoder layers	4 decoder layers	dropout 0.15

I built a reliable, end-to-end English→Bengali/Hindi translation system in plain PyTorch that fits within a single Kaggle session (2×T4, 12 hours). The Unicode-aware, script-preserving preprocessing and separate tokenizers per language helped the model handle Indic morphology and numerals cleanly. Larger or deeper variants and warmup/decay schedules did not outperform this setup for our data scale and time budget.

The Hindi model produces better responses than its Bengali counterpart when tested on custom sample sentences. This may be due to the fact that the Bengali training dataset was smaller. It might be that the model cannot capture its morphological and lexical diversity from the limited dataset.

Our key finding is that although Transformer architecture is far superior to RNN models, vast experimentation is required to stumble upon the best architecture. Transformers require precise tuning of the optimization landscape, hence LR Schleduling and selection of optimal hyperparameters become crucial.

To build a better model, we first need more and cleaner data, especially for Bengali. Many Bengali–English pairs are not true translations: the Bengali side often adds or drops information, and proper names are sometimes changed instead of transliterated (for example, New York becomes Dhaka). This bad data hurts quality more than the model design. So we should clean and grow the corpus: remove mismatched or duplicate pairs, enforce the right script, and spot-check samples. Keep names and numbers consistent by copying or transliterating them. Add trusted parallel data and use back-translation to expand coverage. After the data is solid, we can try a slightly larger model and standard training tweaks (better learning-rate schedules, label smoothing, mixed precision, length-based batching). We can also try other architectures, like fine-tuning a T5-style encoder–decoder, implemented directly in PyTorch. With better data and these steps, we can surpass the current system.

References

[1] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in Neural Information Processing Systems (NIPS)*, pp. 3104–3112, 2014.

[2] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using RNN encoder–decoder for statistical machine translation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1724–1734, Association for Computational Linguistics, Oct. 2014.

[3] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv preprint arXiv:1409.0473*, 2014.

[4] M.-T. Luong, H. Pham, and C. D. Manning, “Effective approaches to attention-based neural machine translation,” in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Lisbon, Portugal), pp. 1412–1421, Association for Computational Linguistics, Sept. 2015.

[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)*, (Berlin, Germany), pp. 1715–1725, Association for Computational Linguistics, Aug. 2016.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pp. 5998–6008, Curran Associates, Inc., 2017.

[7] D. Britz, A. Goldie, M.-T. Luong, and Q. Le, “Massive exploration of neural machine translation architectures,” in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, (Copenhagen, Denmark), pp. 1442–1451, Association for Computational Linguistics, 2017.

[8] T. Gowda and J. May, “Finding the optimal vocabulary size for neural machine translation,” in *Findings of the Association for Computational Linguistics: EMNLP 2020*, (Online), pp. 3955–3964, Association for Computational Linguistics, 2020.

[9] T. Mihaylova and A. F. T. Martins, “Scheduled sampling for transformers,” in *Proceedings of the ACL 2019 Student Research Workshop*, (Florence, Italy), pp. 279–285, Association for Computational Linguistics, 2019.

[10] O. Press and L. Wolf, “Using the output embedding to improve language models,” in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2017)*, (Valencia, Spain), pp. 117–126, Association for Computational Linguistics, 2017.