CS779 Competition: Machine Translation System for India

Name - Sunny Raja Prasad
Roll number - 218171078
Email id - `sunnyrp21@iitk.ac.in`
Indian Institute of Technology Kanpur (IIT Kanpur)

Abstract

I built NMT systems for EN→HI/BN under a train-from-scratch rule. After testing seq2seq and transformers, my final model is an 8-layer Convolutional Seq2Seq (ConvS2S) model. Decoding uses a batched greedy search. On Codabench (s_218171078) I ranked 31 on the dev board (chrF++ 0.385, ROUGE 0.404, BLEU 0.138). The final test submission ranked 46 with chrF++ 0.39 (tie), ROUGE 0.418, BLEU 0.143. Gains came from normalization, word-level tokenization, and hyperparameter tuning.

1 Competition Result

Codalab Username: s_218171078
Final Leaderboard Rank on the Test Set: 46
chrF++ Score corresponding to the Final Rank: 0.39
ROUGE Score corresponding to the Final Rank: 0.418
BLEU Score corresponding to the Final Rank: 0.143
Total Number of Submissions in the Training Phase: 18
Total Number of Submissions in the Testing Phase: 7
Table showing the Number of Submissions made each Week during the Training Phase:

Week (IST)	**# Submissions**	**Meets ≥4?**
Oct 6–Oct 12	2	No
Oct 13–Oct 19	4	Yes
Oct 20–Oct 26	4	Yes
Oct 27–Nov 2	4	Yes
Nov 3	4	Yes
Total	**18**	

2 Problem Description

The goal is to build a Neural Machine Translation (NMT) system that translates English sentences into Indian languages (Hindi and Bengali). The setting mirrors a real newsroom use–case: previously published English articles must be produced reliably in target Indian languages, and the reverse pipeline (Indian language authoring to English) can benefit from the same infrastructure. The task is framed as supervised sequence–to–sequence learning with parallel sentence pairs, trained and evaluated under a strictly controlled competition protocol.

Constraints and rules.

- **Train from scratch**: no pretrained language models; only non-contextual static embeddings (e.g., word2vec/GloVe) are allowed or embeddings trained from scratch.

- **Data usage**: only the provided train/dev/test splits; no external datasets.
- **Implementation**: PyTorch only (no higher-level NMT toolkits)
- **Evaluation**: official metrics are **chrF++**, **ROUGE**, and **BLEU**; leaderboard scoring is computed on Codabench.
- **Protocol**: two phases — a development (training) phase for iterative submissions and a separate test phase for final evaluation.

Objective. Given an English input sentence $x = (x_1, \ldots, x_T)$, learn a parameterized encoder–decoder model to generate a target sentence $y = (y_1, \ldots, y_{T'})$ in HI/BN that maximizes sequence likelihood while producing fluent and adequate translations under the competition metrics. The final deliverables are: (i) a trained model and inference code, (ii) leaderboard submissions in both phases, and (iii) a concise report analyzing design choices, experiments, results, and error patterns.

3 Data Analysis

1. **Train dataset.** The training corpus consists of parallel sentence pairs (x, y) with x in English (EN) and y in Hindi (HI) or Bengali (BN). Each instance is a single sentence-level alignment. The dataset is already split into `train`, `dev`, and a hidden `test` set used by Codabench. We treat EN→HI and EN→BN as two directions under a common preprocessing pipeline.

   Preprocessing steps (applied to both sides): Unicode NFC normalization; punctuation and digits removed; lowercasing; collapsing repeated whitespace; removal of control characters; tokenization using `nltk.word_tokenize`; and vocabulary building. A word-level vocabulary is built for each language pair, mapping all words that appear at least `MIN_COUNT=2` times to an index.

2. **Corpus statistics and noise.** Table 1 reports aggregate statistics. Typical issues I observed include stray HTML entities, duplicated danda "||" or repeated punctuation, mismatched parentheses/quotes, and occasional language-ID mismatches (EN on target side or mixed-script tokens).

Table 1: Corpus statistics for the provided training data (source/target measured after basic tokenization).

Direction	**# Sentences**	**Avg tokens (src/tgt)**	**Vocab size (src/tgt)**	**Duplicates (%)**
EN→HI	117632	17.10/17.36	87909/140182	0.51
EN→BN	31978	13.52/11.74	29846/38602	1.39

Length distribution and OOV. We bin sentence lengths in tokens to detect long-tail effects and truncate/pad accordingly at training time. Due to the word-level tokenization and `MIN_COUNT=2` filter, any word appearing only once in the training corpus is mapped to an $\langle UNK \rangle$ token, creating a significant OOV (out-of-vocabulary) challenge.

Table 2: Sentence-length histogram (token counts on the *source* side).

Bin (tokens)	$[1, 5]$	$(5, 10]$	$(10, 20]$	$(20, 40]$	> 40
Count (%)	6.30	21.23	45.41	25.56	1.49

Script and token-type checks. We run a light language-ID/script detector to flag off-language lines and mixed-script tokens. Common categories: digits, ASCII-only tokens, Devanagari/Bengali letters, punctuation, and named entities.

Table 3: Token-type shares on the target side (HI/BN).

Type	Letters-only	Alnum	Digits	Punct/Symbol
HI (%)	9.58	0.19	1.05	89.19
BN (%)	8.74	0.18	1.03	90.04

3. **Test vs. train differences.** Although the test references are hidden, we can analyze the *test sources* distributed via Codabench. Empirically we observed: (i) slightly longer sentences on average than train/dev; (ii) higher named-entity density (proper nouns, dates, numerals) leading to more copying/transliteration behavior; and (iii) broader topical variety (news/editorial blend). Table 4 summarizes the measurable shifts on sources only.

Table 4: Source-side comparison (train/dev vs. test).

Metric	Train (src)	Dev (src)	Test (src)
Avg tokens	16.33	16.38	16.41
Median tokens	15.00	15.00	15.00
% tokens that are digits	1.15	1.15	1.14
% capitalized (NE proxy)	13.59	13.51	13.52

4. **Insights (actionable for modeling).**
   - *Normalization helps decoding:* Unicode/spacing fixes reduce detokenization artefacts (e.g., duplicated danda) and yield small BLEU/ROUGE gains.
   - *Noise handling:* Removing exact/near duplicates and obvious misaligned pairs stabilizes training loss and improves dev chrF++.
   - *Vocabulary Size:* The word-level tokenization approach creates very large vocabularies (e.g., ¿140k for Hindi), which requires a large model embedding layer and makes handling rare words difficult.

4 Model Description

1. **Model evolution.** I iterated through three families, increasing capacity and alignment quality over time.
   - *Seq2Seq (baseline).* A 1-layer LSTM encoder–decoder without attention ([1, 2]). It established training/inference plumbing but under-translated long sentences.
   - *Seq2Seq + Attention.* Adding soft alignment (Bahdanau/additive) ([3]) improved adequacy and word-ordering.
   - *Transformer trial (small).* A compact encoder–decoder Transformer ([4]) trained from scratch.

   These observations, combined with runtime constraints, led me to select a **Convolutional Seq2Seq (ConvS2S)** model [**?**]. This architecture offered a good balance of performance and parallel training speed. The final system uses **greedy decoding** for simplicity and to meet time constraints (no beam/length penalty, no checkpoint averaging).
2. **Inspirations and citations.** The final model is an implementation of the **Convolutional Sequence to Sequence (ConvS2S)** architecture [**?**]. The encoder–decoder formulation follows this convolutional approach, differing from the RNN-based methods of [2] and [3]. The transformer [4] was also trialed. Indic-specific considerations (normalization) were informed by [5].
3. **Final model (used for the test set).**
   - *Embeddings:* **512-d**, randomly initialized and trained jointly.
   - *Tokenization & preprocessing:* **Word-level tokenization** using `nltk.word_tokenize`. Punctuation and digits were removed. A vocabulary was built for words with a **minimum count of 2**. Reserved tokens: <`pad`>, <`SOS`>, <`EOS`>, <`UNK`>.
   - *Encoder:* **8-layer Convolutional Network** (‘EncoderConvS2S‘), with hidden dimension H=512, embedding dimension E=512, and kernel size k=3. Uses Gated Linear Units (GLU) and weight normalization.
   - *Decoder:* **8-layer Convolutional Network** (‘DecoderConvS2S‘), with hidden dimension H=512, embedding dimension E=512, and kernel size k=3. Uses causal padding to prevent seeing future tokens.

- *Attention:* **Scaled dot-product attention** is applied at each decoder layer. The decoder state (combined with the previous word's embedding) forms the query, which attends to the encoder output.
- *Regularization:* Dropout 0.25, gradient clipping $g_2 \leq 1$. **No label smoothing** was used.
- *Optimization:* Adam with a fixed learning rate 0.0005; standard teacher forcing during training.
- *Checkpointing:* The model with the best validation loss was saved as `best.pt` and used for inference.
- *Inference:* **Greedy decoding** (no beam/length penalty); decoded indices are mapped back to words.
- *Implementation:* PyTorch-only, trained from scratch; no external datasets or pretrained LMs (competition rule).

4. **Objective (loss) functions.** Let x be the source sequence and $y = (y_1, \ldots, y_{T'})$ the target. The decoder parameterizes $p_\theta(y_t \mid y_{<t}, x)$ and the training objective is the standard **negative log-likelihood (Cross-Entropy Loss)**, which ignores the padding token index:

$$\mathcal{L}(\theta) \;=\; -\sum_{t=1}^{T'} \log p_\theta\big(y_t \mid y_{<t}, x\big) \tag{1}$$

Here y_t is the ground-truth token at timestep t. **Teacher forcing** is used during training (no scheduled sampling).

5. **Inference (decoding strategy).** I compared greedy decoding and beam search during development. **The final code uses batched greedy decoding** (no beam or length penalty) for speed. The model generates a sequence of word indices, which are stopped by an `<EOS>` token or the max length cap, and then mapped back to words using the vocabulary.

5 Experiments

1. **Data Pre-processing (source & target).** I applied a uniform pipeline to normalize the text for word-level tokenization:
   - *Normalization:* All text was lowercased.
   - *Cleaning:* All punctuation and digits were removed from the sentences.
   - *Tokenization:* Sentences were tokenized using `nltk.word_tokenize`.
   - *Vocabulary:* A word-level vocabulary was built for each language pair. Words were included if they appeared at least `MIN_COUNT = 2` times.
   - *Special Tokens:* `pad>`, `SOS>`, `EOS>`, `UNK>` were added to the vocabulary.
   - *Padding:* Sentences were truncated or padded to a fixed `seq_length = 64`.
   - *Detokenization rules (post-decoding):* Decoded tokens are simply joined with a space.

   Rationale: This word-level approach was chosen for simplicity. However, it creates a large vocabulary and a significant OOV (out-of-vocabulary) problem for any words not seen at least twice.

2. **Training procedure.** All models were trained in PyTorch from scratch (no external datasets or pretrained LMs). I used standard teacher forcing (ratio=1.0) and monitored **validation loss** for model selection.
   - *Optimizer:* Adam.
   - *Learning-rate schedule:* `ReduceLROnPlateau` (factor 0.5, patience 1) on validation loss.
   - *Regularization:* Dropout; gradient clipping ($\|g\|_2 \leq 1$). **No label smoothing** was used.
   - *Batching:* Standard `batch_size = 32` was used. **No gradient accumulation** was applied.
   - *Early stopping:* Patience of 5 epochs on validation loss; the single best checkpoint was saved and reloaded for inference.
   - *Inference-time stability:* **No checkpoint averaging** was used.

Table 5: Training configuration for the Final ConvS2S Model.

Parameter	**Value**
Model	Convolutional Seq2Seq (ConvS2S)
Optimizer	Adam
LR (initial)	5×10^{-4}
LR Schedule	ReduceLROnPlateau
Epochs	20
Batch Size	32
Teacher Forcing	1.0 (Constant)
Gradient Clip	1.0

Table 6: Model hyper-parameters for the Final ConvS2S Model.

Hyper-parameter	**Value**
Model	ConvS2S
Embedding Dim	512
Hidden Dim	512
Encoder Layers	8
Decoder Layers	8
Encoder Kernel Size	3
Decoder Kernel Size	3
Dropout	0.25
Label Smoothing ε	0.0
Tokenizer	NLTK (Word-level)
Decoding	Greedy Search
Checkpoint Averaging k	N/A

3. **Hyper-parameters (per model) and selection.** My early experiments included RNNs and Transformers, but the final submitted model is a Convolutional Seq2Seq (ConvS2S) model. The parameters were chosen based on common settings for this architecture.

   How hyper-parameters were chosen.

   - *Capacity vs. stability:* A deep stack of 8 convolutional layers was used for both the encoder and decoder, with a 512-dimensional hidden state.
   - *Regularization:* A fixed dropout of 0.25 was applied throughout the network. No label smoothing was used.
   - *Tokenization:* `nltk.word_tokenize` was used for simplicity, despite creating OOV issues.
   - *Decoding:* Batched greedy search was chosen for inference speed over the slower, more complex beam search.
   - *Stability:* A single best checkpoint was saved based on validation loss, rather than averaging checkpoints.

6 Results

1. **Development (dev) results.** The table reports all official metrics from the evaluation scripts (higher is better). A leaderboard *rank* column is included for convenience.
2. **All metrics shown.** chrF++, ROUGE, and BLEU are reported for each model, along with an optional leaderboard rank column.
3. **Test results.** The table above summarizes the final test-phase outcomes (official leaderboard metrics).

Table 7: Dev set results (official metrics; higher is better).

Model	chrF++	ROUGE	BLEU	LB Rank
Seq2Seq (no attention)	_____	_____	_____	____
Seq2Seq + Additive Attn	_____	_____	_____	____
Transformer (small; scratch)	_____	_____	_____	____
Final: ConvS2S (8-Layer)	**0.385**	**0.404**	**0.138**	**31**

Table 8: Test set results (official leaderboard; higher is better).

Model	chrF++	ROUGE	BLEU	LB Rank
Final: ConvS2S (8-Layer)	**0.39**	**0.418**	**0.143**	**46**

4. **Brief discussion.** On the dev set, the strongest model was the **8-layer Convolutional Seq2Seq (ConvS2S)** model. This configuration was selected over the non-attention baseline and a scratch-trained small Transformer due to its stable convergence and performance on the dev metrics. Regularization (dropout) and training stability (gradient clipping, `ReduceLROnPlateau` LR schedule) contributed to the final result.

   At test time, **batched greedy decoding** was used for inference speed, yielding the final leaderboard scores shown above. The model likely performed well because (i) the deep stack of 8 convolutional layers captured local context and word-order information effectively, (ii) the Gated Linear Units (GLU) controlled the information flow through the network, and (iii) the multi-step attention in the decoder helped align the source and target sequences.

7 Error Analysis

1. **Model-wise error profile (dev vs. test).**
   - *Seq2Seq (no attention).* Under-translated long sentences; frequent content drops and reordering errors.
   - *Seq2Seq + Additive Attention.* Large reduction in omissions; better verb–argument alignment. Residual issues on rare named entities.
   - *Transformer (small, scratch).* Competitive on short/medium sentences but unstable convergence given budget.
   - *Final: ConvS2S (8-Layer).* Best balance of performance and training speed among experiments. Remaining errors are concentrated in: (i) **Out-of-Vocabulary (OOV)** words, (ii) **missing punctuation/digits**, (iii) **repetitive loops** or short sentences from greedy decoding, and (iv) handling of very long-distance dependencies.
2. **Analysis with model instrumentation (qualitative).** We inspected attention weights and decoder token probabilities to understand failure modes.
   - *Attention heat maps.* The ConvS2S model applies attention at each decoder layer. In correct translations, the attention (averaged across layers) focuses on the relevant source tokens. In failure cases (like repetitions), the attention appears to get "stuck" on the same source words.
   - *Token-level confidence.* Over-confident spikes correlate with brevity (early `</s>`) or repetitive loops (e.g., "the the the"). This is a common failure mode for greedy decoding, which cannot recover from an early bad decision.
3. **Why models are not perfect (typology of errors) and remedies.**
   - *Named entities & OOV.* This is the **largest error source**. The NLTK word-level vocabulary and `MIN_COUNT=2` filter mean any rare name or word is mapped to `<UNK>`.
     Mitigation: Use **subword tokenization** (like BPE or SentencePiece) to handle rare words and name-entity transliteration.
   - *Missing punctuation/spacing.* The model cannot generate punctuation or digits because they were **removed during preprocessing**. This leads to a loss of information.

Mitigation: Retain punctuation and digits in the preprocessing pipeline so the model can learn to generate them.

- *Repetitions & Brevity.* The model often gets stuck in repetitive loops or stops translating too early. This is a characteristic failure of **greedy search**.
  Mitigation: Implement **beam search**, which explores multiple hypotheses and would almost certainly yield large quality gains.
- *Long sentences & clause reordering.* The fixed kernel size of the CNN (k=3) may struggle to capture very long-range dependencies in complex sentences.
  Mitigation: Experiment with larger kernel sizes, a deeper model, or a model with recurrent properties (like BiLSTM).

4. **Illustrative examples (dev).**

Table 9: Qualitative examples (EN→HI/BN). Common error categories and notes.

Source (EN)	**Reference (HI/BN)**	**Hypothesis (Final Model) & Notes**
The committee met on Friday to finalize the budget.	(HI)	Correct content; minor article/style difference (acceptable).
He pushed the deadline by two weeks.	(HI)	<u>Error:</u> literal translation, greedy search error. *Mitigation:* Use subword tokenization and beam search.
India recorded higher rainfall in the northern states this year.	(BN)	<u>Error:</u> NE handling ("northern states") → `<UNK>`; *Mitigation:* Use subword tokenization to handle OOV named entities.
The Prime Minister addressed the nation at 8 pm.	(HI) 8	<u>Error:</u> Missing "8 pm". Model cannot generate "8" because digits were removed. *Mitigation:* Keep digits during preprocessing.

5. **Interesting insights.**
   - *Word-level tokenization is the main bottleneck.* The massive OOV problem created by the `MIN_COUNT=2` filter and NLTK word tokenizer is the single biggest limiting factor on performance, especially for named entities.
   - *Greedy decoding is fast but costly.* The speed of batched greedy search comes at a high price in translation quality. Failures like repetition and early stopping are common and could be significantly reduced by implementing beam search.
   - *Preprocessing decisions are critical.* The choice to remove all punctuation and digits, while simplifying tokenization, results in a critical loss of information, making it impossible for the model to produce fully correct translations.

8 Conclusion

This project built EN→HI/BN neural MT systems under strict "train-from-scratch" constraints. Across baselines (including RNNs and Transformers), the final submitted model was an **8-layer Convolutional Seq2Seq (ConvS2S)** model, trained with dropout, gradient clipping, and an Adam optimizer with a `ReduceLROnPlateau` scheduler. The preprocessing pipeline was based on **word-level tokenization** (`nltk.word_tokenize`) after removing all punctuation and digits. Inference was performed with **batched greedy decoding** for speed. This approach culminated in the final leaderboard performance (dev rank 31; test rank 46).

Key findings.

- **Preprocessing decisions are critical.** The choice to remove all punctuation and digits, while simplifying tokenization, resulted in a critical loss of information and made it impossible for the model to generate fully correct translations.
- **Word-level tokenization is a major bottleneck.** Using a `MIN_COUNT=2` filter on word-level tokens created a massive Out-of-Vocabulary (OOV) problem, especially for named entities, which severely limited performance.
- **Greedy decoding limits quality.** The speed of batched greedy search comes at a high price in translation quality. Failures like repetition and early stopping are common and are a direct result of this inference strategy.

Recommendations.

- **Use subword vocabularies.** The single most important improvement would be to replace NLTK word tokenization with a **SentencePiece (BPE/unigram)** vocabulary to effectively handle OOV words and morphology.
- **Implement Beam Search.** The second most critical improvement would be to replace greedy decoding with **beam search** (with length normalization) to significantly reduce repetition and brevity errors.
- **Preserve information.** Do not remove punctuation and digits during preprocessing. A subword tokenizer would handle these naturally, allowing the model to learn to generate them.

Future directions.

- **Implement recommendations:** The first steps must be to implement subword tokenization and beam search.
- **Experiment with architectures:** A deeper analysis comparing the optimized ConvS2S model against an optimized BiLSTM+Attention model (with subwords and beam search) would be needed to truly determine the best architecture.
- **Copy/transliteration constraints:** for named entities and numerals, integrated into decoding (once subwords are used).
- **Coverage/consistency penalties:** (applied during beam search) to reduce omissions and repetitions on long sentences.

Overall, a Convolutional Seq2Seq (ConvS2S) model was able to achieve a competitive rank. However, its performance was severely constrained by foundational decisions in preprocessing (word-level tokenization, punctuation removal) and decoding (greedy search). The proposed extensions—primarily adopting subwords and beam search—target the largest error classes and provide clear paths to significant further improvements.

References

[1] G. Neubig, "Neural machine translation and sequence-to-sequence models: A tutorial," *arXiv preprint arXiv:1703.01619*, 2017.

[2] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *arXiv preprint arXiv:1409.3215*, 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] 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 (NeurIPS)*, 2017.

[5] A. Kunchukuttan and P. Bhattacharyya, "Utilizing language relatedness to improve machine translation: A case study on languages of the indian subcontinent," *arXiv preprint arXiv:2003.08925*, 2020.