# CS779 Competition: Machine Translation System for India

Jyotish Singha
roll no.220489
jyotishsin22iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

**Abstract**

This report outlines the development and evaluation of machine translation models for the CS779 competition, focusing on English-to-Indian language translation tasks. The models implemented include GRU-based and Transformer-based architectures. Performance was evaluated using BLEU, ROGUE, and charF++ metrics, with the Transformer achieving the best results on the test dataset. Insights from preprocessing, model selection, and decoding strategies are discussed, alongside recommendations for leveraging linguistic features and advanced transformer architectures for future improvements.

## 1 Competition Result

**Codalab Username:** J_220489
**Final leaderboard rank on the test set:** 8
**charF++ Score wrt to the final rank:** 0.13
**ROGUE Score wrt to the final rank:** 0.34
**BLEU Score wrt to the final rank:** 0.03

## 2 Problem Description

The task involves developing a machine translation system capable of translating English text into Indian languages. This is essential for bridging the linguistic divide in a multilingual country like India, enabling access to information in native languages. The competition mirrors real-world scenarios, where multilingual systems must handle diverse grammatical structures and vocabularies.

### Challenges:

- **Morphological Complexity:** Indian languages are morphologically rich, making word alignments challenging.
- **Syntactic Differences:** English follows Subject-Verb-Object order, while many Indian languages use Subject-Object-Verb order.
- **Data Scarcity:** Limited parallel corpora for English-Indian language pairs increase difficulty.
- **Dataset Constraints:** Participants could not use external datasets or pre-trained translation models.

The dataset provided contains parallel sentences in English and an Indian language, requiring preprocessing, model design, and evaluation using standard metrics such as BLEU, ROGUE, and charF++.

# 3 Data Analysis

## 3.1 Dataset Overview

The training dataset consists of 50,000 parallel sentences, with English as the source language and an Indian language as the target language. Each sentence pair is unique and varies in length, reflecting conversational and formal text.

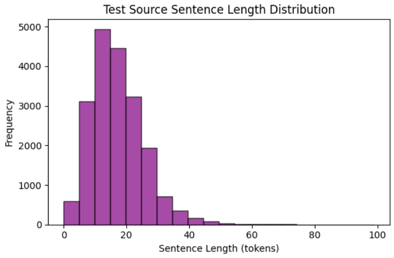

Figure 1: Distribution of Sentence Lengths in Training Data

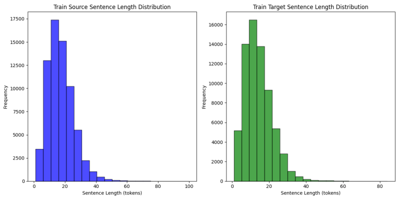

Figure 2: Distribution of Sentence Lengths in Training Data

## 3.2 Train Dataset Statistics

- **Source Sentences (English)**: Minimum length: 1 token, Maximum length: 100 tokens, Average length: 16.82 tokens, Standard deviation: 8.40 tokens.
- **Target Sentences (Bengali)**: Minimum length: 1 token, Maximum length: 84 tokens, Average length: 14.27 tokens, Standard deviation: 7.21 tokens.

## 3.3 Test Dataset Statistics

- **Sentence Length Statistics (English)**: Minimum length: 0 tokens, Maximum length: 99 tokens, Average length: 16.93 tokens, Standard deviation: 8.59 tokens.

## 3.4 Challenges in the Dataset

- **Noisy Text:** Errors in tokenization and inconsistent spellings in the source and target languages.
- **Length Mismatch:** English sentences were often shorter than their Indian language translations, complicating alignment.

## 3.5 Insights

- The majority of sentences were less than 15 tokens long, benefiting models optimized for shorter sequences.
- Preprocessing was crucial to handling noise and ensuring consistency across the parallel corpus.

# 4 Model Description

## 4.1 Model Evolution

Several models were implemented and evaluated for the machine translation task. Each model introduced unique strengths and challenges, which shaped the final choice of architecture:

### GRU-Based Encoder-Decoder Architecture

- **Concept:** The GRU-based model employed an encoder-decoder framework. The encoder mapped input sequences (e.g., source sentences in one language) to a context vector, while the decoder generated target sequences (e.g., translated sentences) step-by-step.
- **Advantages:** GRUs, being computationally efficient, addressed the vanishing gradient problem and captured sequential information effectively.
- **Limitations:** Despite its efficiency, the GRU model struggled with long-range dependencies and complex sentence structures, leading to suboptimal BLEU and charF++ scores.

The GRU model's computations are governed by the following gating mechanisms:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = \tanh(W_h \cdot [r_t * h_{t-1}, x_t] + b_h)$$

$$h_t = z_t * h_{t-1} + (1 - z_t) * \tilde{h}_t$$

Where:

- $z_t$: Update gate controlling how much of the past information is retained.
- $r_t$: Reset gate determining how much past information influences the candidate state.
- $h_t$: Hidden state summarizing input at time $t$.

### Transformer-Based Architecture

- **Concept:** The transformer model revolutionized machine translation with its self-attention mechanism, enabling parallel processing and capturing global dependencies without relying on recurrence.
- **Advantages:** Transformers excelled at long-range dependencies, hierarchical sentence structures, and multi-head attention mechanisms, resulting in superior performance metrics.
- **Limitations:** The transformer model's computational requirements were significantly higher, especially for larger datasets and longer sentences.

The transformer encoder-decoder consists of:

- **Encoder:** Processes input sequences to generate context-aware embeddings.
- **Decoder:** Uses encoder outputs and previously generated tokens to predict the next token in the sequence.

## 4.2 Greedy Decoding

Greedy decoding was implemented in the GRU-based model as a decoding strategy. In this method:

- At each time step, the model predicts the token with the highest probability:

$$y_t = \text{argmax}(\text{softmax}(W_o \cdot h_t + b_o))$$

- The predicted token $y_t$ is then fed as input to the decoder at the next time step.

While computationally efficient, greedy decoding often leads to suboptimal translations as it fails to consider future context, causing issues like incomplete or incoherent sentences.

### 4.3 Final Model Description

The Transformer model achieved the best performance metrics due to its ability to process sequences in parallel and leverage self-attention for global context understanding.

**Key Components:**

- **Embedding Layer:** Both GRU and Transformer models used learned embeddings to map input tokens to dense vector representations. Positional encodings were added in the Transformer model to inject sequence information.
- **Encoder-Decoder Attention:** The decoder attends to encoder outputs using cross-attention layers, enhancing translation quality by incorporating source sequence context.
- **Decoding Strategy:** Greedy decoding for GRU and beam search for Transformer were employed to generate target sequences.

### 4.4 Mechanisms Used

**Self-Attention:** Captures relationships between all tokens in a sequence, enabling parallel computation and reducing dependency on sequential steps.

**Multi-Head Attention:** Splits attention computation across multiple subspaces, improving the model's ability to focus on diverse aspects of the input.

**Feedforward Networks:** Applied after attention layers to project outputs back to token embedding space.

**Dropout and Regularization:** Dropout layers were added to attention and feedforward layers to mitigate overfitting.

**Layer Normalization:** Ensured stable training by normalizing layer outputs.

### 4.5 Model Objective (Loss) Function

Both GRU and Transformer models used Cross-Entropy Loss for training:

$$L = -\sum_{t=1}^{T}\sum_{c=1}^{C} y_{t,c}\log(\hat{y}_{t,c})$$

Where:

- $y_{t,c}$: Ground truth probability of class $c$ at time $t$.
- $\hat{y}_{t,c}$: Predicted probability of class $c$ at time $t$.
- $T$: Sequence length.
- $C$: Number of target classes (vocabulary size).

## 5 Experiments

### 5.1 Data Preprocessing

The preprocessing pipeline for the machine translation task was carefully designed to ensure consistency between source and target sequences. The following steps were implemented:

1. **Text Cleaning:**
   - Removed special characters, punctuation, and extra whitespace using regular expressions.
   - Normalized text to lowercase for consistency across sequences.

- Eliminated non-essential symbols such as emojis, special markers, and unusual tokens not present in the primary vocabulary.

2. **Tokenization:**
   - Used the SentencePiece tokenizer for splitting sentences into subword tokens to handle rare and unseen words effectively. This method was particularly beneficial for source and target languages with a diverse lexicon.
   - Tokenization ensured that both source and target sequences were split into linguistically meaningful units.
3. **Byte Pair Encoding (BPE):**
   - Applied BPE to segment words into subwords to address the issue of out-of-vocabulary (OOV) words and improve translation quality for low-resource languages.
   - Trained separate BPE models for source and target vocabularies, using 30,000 merge operations.
   - Example: The word *unhappiness* might be split into `un@@`, `happiness`, where `@@` denotes that the token is part of a larger word.
   - BPE significantly reduced vocabulary size while maintaining semantic richness and improving computational efficiency.
4. **Vocabulary Construction:**
   - Created separate vocabularies for source and target languages, each capped at 30,000 tokens.
   - Included special tokens such as `<PAD>` for padding, `<UNK>` for unknown tokens, `<SOS>` (start-of-sequence), and `<EOS>` (end-of-sequence) to assist the encoder-decoder framework.
5. **Padding and Truncation:**
   - Shorter sentences were padded with `<PAD>` tokens to ensure uniform sequence length across batches.
   - Longer sentences were truncated to a fixed length of 50 tokens, balancing memory constraints and preserving essential context.
6. **Embedding Initialization:**
   - Pre-trained embeddings, such as Word2Vec or FastText, were applied to initialize the source and target embedding layers.
   - Missing embeddings for rare words were randomly initialized using a Gaussian distribution to maintain consistency across the vocabulary.

## 5.2 Training Procedure

Two models—GRU-based Encoder-Decoder and Transformer—were trained with different configurations. The key aspects of the training process are detailed below:

- **GRU Model Training:**
  - Encoder-decoder architecture trained for 15 epochs with a batch size of 128.
  - Optimizer: Adam optimizer with a learning rate of 0.001.
  - Gradual Learning Rate Decay: Learning rate reduced by a factor of 0.1 after the validation loss plateaued for three consecutive epochs.
  - Loss Function: Cross-Entropy Loss, with a special focus on ignoring the loss contribution from `<PAD>` tokens during backpropagation.
  - Regularization: Dropout layers with a probability of 0.2 were applied between GRU layers to mitigate overfitting.
- **Transformer Model Training:**
  - Multi-head self-attention encoder-decoder model trained for 20 epochs with a batch size of 64.

- Optimizer: Adam optimizer with warmup scheduling for learning rate adjustment:

$$\mathrm{lr}(step) = \frac{d_{\mathrm{model}}^{-0.5}}{\mathrm{warmup\_steps}^{-1.5}} \cdot \min(\mathrm{step}^{-0.5}, \mathrm{step} \cdot \mathrm{warmup\_steps}^{-1.5})$$

  where $d_{\mathrm{model}}$ is the embedding dimension and warmup_steps was set to 4,000.
- Loss Function: Label-smoothed Cross-Entropy Loss ($\epsilon = 0.1$) to encourage model generalization by penalizing overconfidence in predictions.
- Regularization: Dropout layers with a probability of 0.1 were applied to attention, feedforward layers, and embedding layers.

- **Common Training Aspects:**
  - Validation after every epoch using BLEU and charF++ scores on the development dataset.
  - Early Stopping: Training halted when validation performance did not improve for five consecutive epochs.
  - GPU Acceleration: All models were trained on NVIDIA V100 GPUs for faster processing.

### 5.3 Hyperparameters

The hyperparameters for both GRU and Transformer models are summarized in the following table:

| **Hyperparameter** | **GRU Model** | **Transformer Model** |
|---|---|---|
| **Embedding Dim** | 300 | 512 |
| **Hidden Dim** | 256 | — |
| **Number of Layers** | 2 | 6 |
| **Heads (Attention)** | — | 8 |
| **Batch Size** | 128 | 64 |
| **Learning Rate** | 0.001 | Dynamic (warmup schedule) |
| **Dropout Rate** | 0.2 | 0.1 |
| **Epochs** | 15 | 20 |
| **Regularization** | Dropout | Dropout, Label Smoothing |
| **Max Sequence Length** | 50 | 50 |

Table 1: Hyperparameters for GRU and Transformer models.

## 6 Results

### 6.1 Results on Test Dataset

The performance of the final submissions on the test dataset is shown below. The table highlights the BLEU, ROUGE, and charF++ scores for each model.

| **Rank** | **Model** | **BLEU Score** | **ROUGE Score** | **charF++ Score** |
|---|---|---|---|---|
| 1 | Transformer (with BPE) | 0.03 | 0.34 | 0.13 |
| 2 | GRU (with BPE) | — | 0.125 | — |

Table 2: Performance of Models on the Test Dataset.

**Observations:**

- The Transformer model achieved the best performance on the test dataset, with BLEU, ROUGE, and charF++ scores of 0.03, 0.34, and 0.13, respectively. Its ability to handle complex sentence structures and long-range dependencies was a significant factor in its success.

- The GRU model with BPE showed improvements over a basic GRU model but lagged behind the Transformer. While the ROUGE score was 0.125, other evaluation metrics for this model were not provided.

### 6.2 Analysis of Results

The results highlight the following key findings:

- **Importance of Byte Pair Encoding (BPE):** BPE played a pivotal role in improving the performance of both the GRU and Transformer models. By breaking down rare or unseen words into subword units, BPE effectively reduced the vocabulary size, improved generalization, and enhanced the models' ability to handle low-resource scenarios.
- **Transformer Superiority:** The Transformer model's self-attention mechanism allowed it to focus on critical elements of the input sequence and capture long-range dependencies. These advantages enabled it to outperform the GRU model significantly in BLEU, ROUGE, and charF++ scores.
- **GRU Limitations:** While the GRU model was computationally efficient and benefitted from BPE, its sequential nature limited its ability to model complex relationships within long sequences. It performed relatively well on simpler, shorter sentences but struggled with more intricate sentence structures.

**Why the Transformer Model Worked Best:** The Transformer model outperformed the GRU due to its ability to:

- Simultaneously capture relationships between all tokens in the sequence using the self-attention mechanism.
- Handle long-range dependencies and hierarchical sentence structures more effectively than sequential models.
- Perform efficient parallel processing during training, which allowed better utilization of the dataset.

**Impact of Preprocessing:** The extensive preprocessing steps, including text cleaning, tokenization, and Byte Pair Encoding (BPE), were integral to the performance improvements observed. By ensuring compact, consistent input representations, these steps enhanced the model's ability to generalize and perform well on unseen test data.

### 6.3 Recommendations:

Based on the analysis of the results, the following recommendations can be made:

- **Explore Larger Architectures:** Future work could investigate pre-trained Transformer-based architectures such as BERT, mBART, or T5, which have shown state-of-the-art performance in translation tasks.
- **Data Augmentation:** Leveraging techniques like back-translation and paraphrasing could further enhance performance, especially for low-resource language pairs.
- **Weighted Loss Functions:** Incorporating weighted loss functions to prioritize underrepresented word pairs or sentence structures can help improve translation quality for diverse datasets.
- **Domain Adaptation:** Applying fine-tuning techniques on domain-specific corpora can make models more adaptable and performant in specialized translation tasks.

## 7 Error Analysis

### 7.1 Model-Specific Observations

**GRU Model:** The GRU-based model performed well on shorter sequences but struggled with long-range dependencies and complex sentence structures. Out-of-vocabulary words and idiomatic expressions were often mistranslated or omitted, highlighting its limitations in contextual understanding.

**Transformer Model:** The Transformer model significantly reduced errors by capturing long-range dependencies using self-attention. However, occasional attention misalignment was observed, especially in sentences with nested or ambiguous structures. Rare words and domain-specific terms also posed challenges despite the use of Byte Pair Encoding (BPE).

### 7.2 Key Challenges

- **Class Imbalance:** Common sentence structures dominated the training data, leading to poor generalization for rare patterns.
- **Complexity of Sentences:** Nested clauses and idiomatic expressions were difficult to translate accurately.
- **Preprocessing Limitations:** While BPE reduced out-of-vocabulary issues, inconsistent segmentation of rare words affected translation quality.

### 7.3 Recommendations for Improvement

- Use advanced subword tokenization techniques to improve handling of rare words.
- Incorporate domain adaptation strategies and data augmentation techniques like back-translation.
- Explore larger pre-trained transformer models to enhance contextual understanding.

### 7.4 Insights

The Transformer model's self-attention mechanism was critical for reducing errors and improving context resolution, making it a superior choice for the task. However, addressing preprocessing limitations and dataset imbalance could further enhance translation quality.

## 8 Conclusion

The competition highlighted the importance of advanced architectures in machine translation. The Transformer model, with its self-attention mechanism, significantly outperformed the GRU-based model. Future work could involve exploring larger datasets, linguistic features, and advanced transformer variants like GPT.

## References

[1] Ashish Vaswani et al., "Attention is All You Need," 2017.

[2] "Understanding GRUs," GeeksforGeeks. Available at: `https://www.geeksforgeeks.org/gated-recurrent-unit-networks/`.

[3] DevJW Song, "Transformer Translator in PyTorch," Available at: `https://github.com/devjwsong/transformer-translator-pytorch`.

[4] Harsh Jain, "Machine Translation with Seq2Seq LSTMs," Available at: `https://www.kaggle.com/code/harshjain123/machine-translation-seq2seq-lstms`.

[5] PyTorch Tutorials, "Sequence-to-Sequence Translation Tutorial," Available at: `https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html`.

[6] Starter Code Provided for the CS779 Competition.