International Institute of Information Technology
Bangalore

NLP

AIM 829

Project Report

Team Members:

Abhinav Kumar (IMT2022079)
Abhinav Deshpande (IMT2022580)
Vaibhav Bajoriya (IMT2022574)
Shashank Devarmani (IMT2022107)

April 22, 2025

Problem Statement

A considerable number of non-native English speakers encounter difficulties comprehending spoken English educational content, particularly when delivered with unfamiliar accents.
Not everyone is comfortable reading captions along with an educational video (eg. elderly people). Especially, when the video is for demonstration purpose.

IIT Bombay English-Hindi Corpus

For our English-to-Hindi machine translation task, we utilize the IIT Bombay English-Hindi Corpus. This is a publicly available parallel corpus developed by the Center for Indian Language Technology (CFILT) at IIT Bombay. The corpus consists of aligned sentence pairs in English and Hindi, making it a valuable resource for supervised neural machine translation (NMT) models.

Data Preprocessing

After loading the IIT Bombay English-Hindi parallel corpus, we apply a series of preprocessing steps to clean and filter the dataset before training our translation model.

1. **Sentence Alignment and Loading:** The English and Hindi sentence pairs are loaded from the provided text files. Each line in the English file corresponds to a line in the Hindi file, resulting in a total of 788,099 aligned sentence pairs.

2. **Noise Removal:** Hindi sentences containing any Latin (English) characters are considered noisy and are filtered out. This helps ensure that the Hindi data is clean and purely in the target script.

3. **Text Normalization:** Both English and Hindi sentences undergo further cleaning using a custom `clean_text()` utility. This includes lowercasing (for English), punctuation removal, and script-specific normalization.

4. **Length Filtering:** Sentence pairs with either English or Hindi sentence lengths (measured in word tokens) exceeding 50 are removed. This helps in maintaining a manageable sequence length for training and improves overall model convergence.

5. **Train-Validation Split:**
   - A full dataset split is created with 90% of the data used for training and 10% for validation.
   - Additionally, a smaller subset of 150,000 sentence pairs is sampled to facilitate rapid experimentation. This smaller subset is split into training and validation sets with a 70-30 split.

These preprocessing steps help in improving data quality, ensuring script purity, and controlling sequence lengths, which are crucial for efficient and effective model training.

The dataset is available via the Hugging Face Datasets library[1] and includes:

- A parallel corpus of English-Hindi sentence pairs.

[1]https://huggingface.co/datasets/iitb

- Additional monolingual Hindi corpora for potential use in language modeling or back-translation.

We accessed the dataset using the Hugging Face `datasets` library and preprocessed the sentence pairs to standardize tokenization and remove any malformed or excessively long examples. This corpus served as the primary training and evaluation dataset for our sequence-to-sequence translation model with attention.

TED Talks – Hindi-English Truncated Corpus

The TED Talks Hindi-English Truncated Corpus is a high-quality parallel dataset derived from translated TED talk transcripts. Each English sentence is paired with its corresponding human-translated Hindi version, providing semantically rich and contextually coherent sentence pairs suitable for training machine translation models.

Dataset Overview:

- **Source:** Transcripts of TED talks, which are formal and well-articulated presentations covering a wide range of topics.
- **Alignment:** The dataset consists of aligned sentence pairs, ensuring that each English sentence corresponds accurately to its Hindi translation.
- **Availability:** The dataset is publicly available on: https://www.clarin.eu/resource-families/parallel-corpora

Seq2Seq Model based on LSTM

Dataset and Preprocessing

Source 1: TED Talks - Hindi-English Truncated Corpus

The first dataset used is obtained from clarin's parallel corpora *Hindi-English Truncated Corpus*.

Source 2: IIT Bombay English-Hindi Parallel Corpus

The second dataset used in this project is the IIT Bombay English-Hindi Parallel Corpus. After doing the general pre-processing which was done after downloading the dataset, the first 1,00,000 rows are used for training.

Preprocessing Steps

After the dataset is loaded, it is passed through a series of pre-processing steps given below.

- **Lowercasing Text**
  All text is converted to lowercase to maintain uniformity and reduce the vocabulary size. This ensures that words like "The" and "the" are treated as the same token.
- **Removing Punctuation and Special Characters**
  Punctuation marks are generally not useful in translation tasks unless explicitly modeled. Removing them reduces noise and focuses the model on meaningful words.

- **Removing Digits**
  Numeric digits are removed to simplify learning. Most numbers do not contribute to sentence semantics unless specially handled, which is beyond the scope of this basic model.
- **Trimming and Removing Extra Spaces**
  Leading/trailing spaces and multiple consecutive spaces are removed to maintain clean tokenization and ensure uniform sequence structure.
- **Handling Null and Duplicate Values**
  Sentence pairs with missing values or duplicates are removed. Null values can cause model failure, and duplicates may lead to biased training or overfitting.
- **Length Filtering**
  Sentences longer than 20 words are filtered out. Long sequences increase training complexity and are harder for Seq2Seq to handle effectively due to vanishing gradients.
- **Adding Start and End Tokens**
  Special tokens (e.g., `START_`, `_END`) are added to the target (Hindi) sentences to signal the beginning and end of translation during decoding.
- **Vocabulary Creation and Token Indexing**
  Unique words from both English and Hindi corpora are extracted to build vocabularies. Each word is assigned a unique index to convert text into numeric sequences suitable for model input.

Model Architecture

Sequence-to-Sequence model

Sequence-to-Sequence (Seq2Seq) modeling is a type of neural network architecture designed to transform one sequence into another. It is particularly useful in tasks such as machine translation, text summarization, and question answering, where the input and output sequences may differ in length.

Encoder-Decoder Model

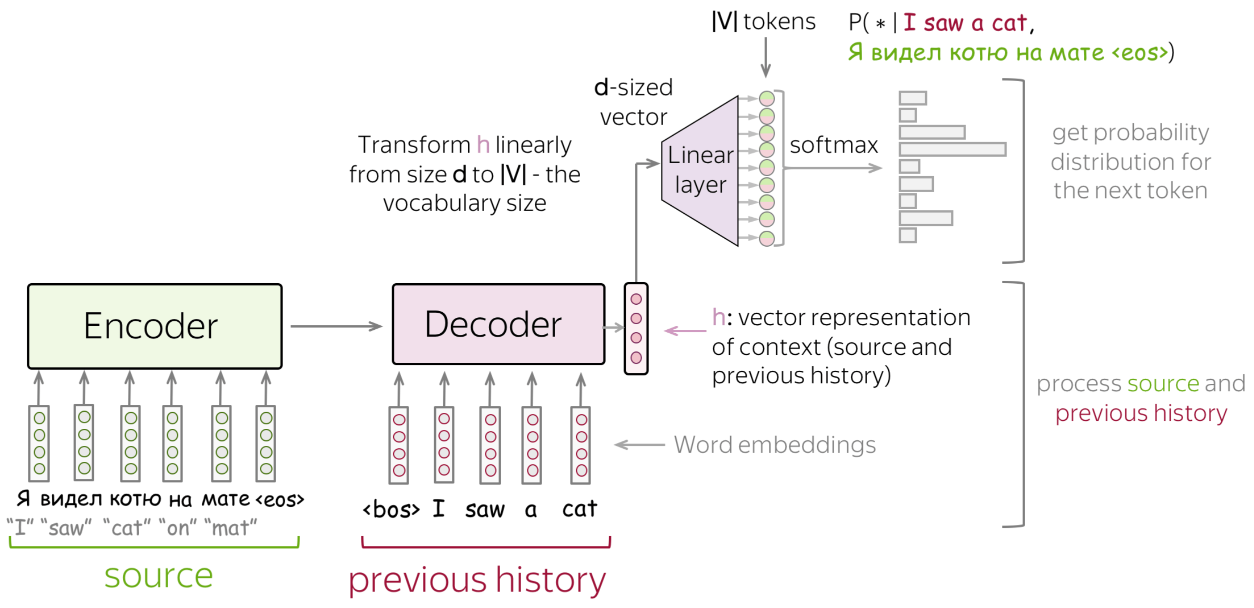

|V| tokens
P(∗ | I saw a cat,
Я видел котю на мате <eos>)
d-sized vector
Transform h linearly from size d to |V| - the vocabulary size
Linear layer
softmax
get probability distribution for the next token
Encoder
Decoder
h: vector representation of context (source and previous history)
process source and previous history
Word embeddings
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
source
<bos> I saw a cat
previous history

Figure 1: seq2seq model

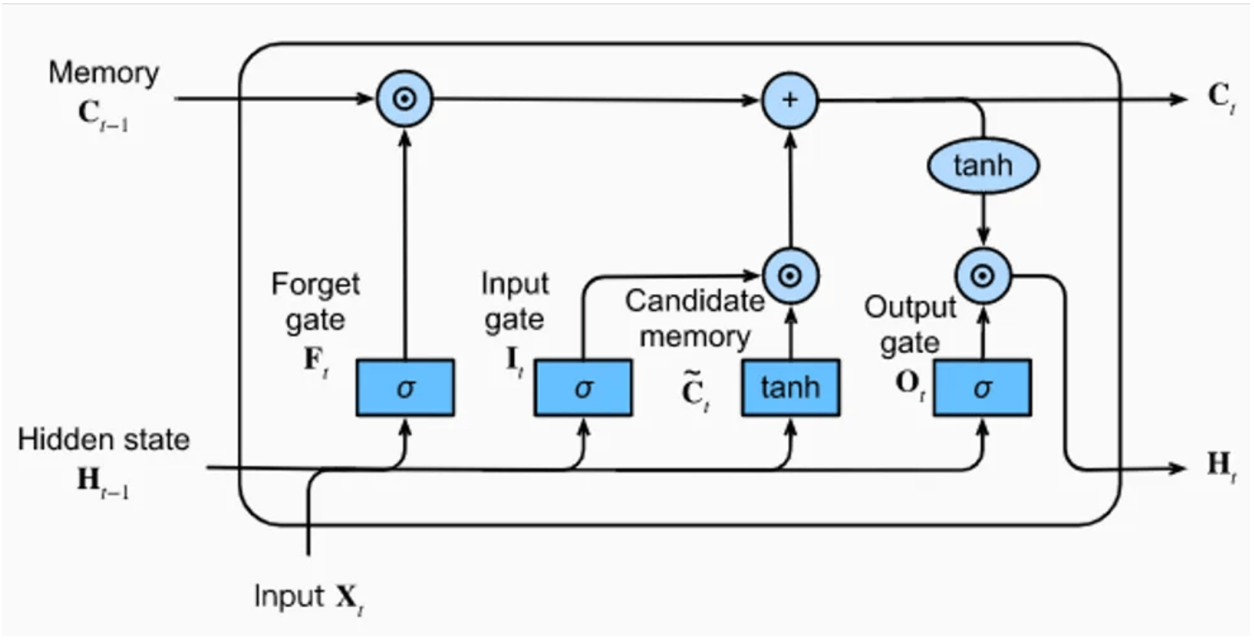

Memory $\mathbf{C}_{t-1}$
$\mathbf{C}_t$
tanh
Forget gate $\mathbf{F}_t$
Input gate $\mathbf{I}_t$
Candidate memory $\tilde{\mathbf{C}}_t$
Output gate $\mathbf{O}_t$
σ
σ
tanh
σ
Hidden state $\mathbf{H}_{t-1}$
$\mathbf{H}_t$
Input $\mathbf{X}_t$

Figure 2: LSTM architecture

The translation model follows a sequence-to-sequence encoder-decoder architecture with the following key components:

Word Embeddings

In this project, trainable embedding layers are used for both English and Hindi vocabularies. These embeddings are initialized randomly and learned during model training. As the model is exposed to bilingual sentence pairs, the embedding layers adjust their weights to better represent the contextual usage of words, making them highly task-specific. This approach allows the model to build its own understanding of word meanings based on the translation task at hand, without relying on pre-trained vectors.

Encoder

The encoder is responsible for reading the input English sentence and compressing its information into a fixed-size context vector. This vector is then passed to the decoder.

- **Input Layer:** Takes the tokenized English sentence (a sequence of word indices).
- **Embedding Layer:** Converts each word index into a dense vector representation, capturing semantic relationships between words.
- **LSTM Layer:** Processes the sequence of embeddings and outputs a final hidden state and cell state. These states summarize the entire input sequence and are passed to the decoder.

Decoder

The decoder generates the Hindi translation, one word at a time, using the context vector from the encoder.

- **Input Layer:** Receives the target sequence (Hindi) shifted by one position during training, starting with the special `START_` token.
- **Embedding Layer:** Similar to the encoder, it transforms word indices into dense vectors.
- **LSTM Layer:** Initialized with the encoder's final hidden and cell states. It uses these states to begin generating the translation.
- **Dense Output Layer:** A fully connected layer with softmax activation, producing a probability distribution over the Hindi vocabulary. The word with the highest probability is selected at each step.

Sequence Generation

During inference (translation of unseen sentences):

- The encoder processes the input sentence and generates initial states.
- The decoder uses these states along with the `START_` token to begin generation.
- At each time step, the predicted word is fed back into the decoder to generate the next word.
- The process continues until the `_END` token is produced or a maximum sentence length is reached.

Summary of Dimensions and Settings

- **Embedding Dimension:** 300 units for both encoder and decoder LSTM layers.
- **Vocabulary Sizes:** Based on unique words in the dataset after preprocessing.
- **Sequence Lengths:** Padded to a maximum of 20 tokens.

LSTM Cell Equations

An LSTM cell at time step t computes:

$$\begin{aligned}
f_t &= \sigma\big(W_f x_t + U_f h_{t-1} + b_f\big) && \text{(forget gate)}\\
i_t &= \sigma\big(W_i x_t + U_i h_{t-1} + b_i\big) && \text{(input gate)}\\
\tilde{c}_t &= \tanh\big(W_c x_t + U_c h_{t-1} + b_c\big) && \text{(cell candidate)}\\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t && \text{(cell state update)}\\
o_t &= \sigma\big(W_o x_t + U_o h_{t-1} + b_o\big) && \text{(output gate)}\\
h_t &= o_t \odot \tanh(c_t) && \text{(hidden state)}
\end{aligned}$$

where $\sigma(\cdot)$ is the logistic sigmoid, $\odot$ denotes element-wise multiplication, x_t is the input vector, h_{t-1} and c_{t-1} are the previous hidden and cell states, and W_*, U_*, b_* are the learned weight matrices and biases.

Categorical Cross-Entropy Loss

For a single training example with true one-hot target $\mathbf{y} = (y_1, \ldots, y_K)$ and predicted probability distribution $\hat{\mathbf{y}} = (\hat{y}_1, \ldots, \hat{y}_K)$, the categorical cross-entropy loss is

$$\mathcal{L}_{\text{CE}}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{k=1}^{K} y_k \log\big(\hat{y}_k\big).$$

For a dataset of N examples, we take the average:

$$\mathcal{L}_{\text{CE}} = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} y_{n,k} \log\big(\hat{y}_{n,k}\big).$$

Training Details

Parameter	TED Talks Corpus	IIT Bombay Corpus
Loss Function	Categorical Cross-Entropy	Categorical Cross-Entropy
Optimizer	Adam	Adam
Batch Size	128	128
Epochs	100 (Early Stopping applied)	15 (Early Stopping applied)
Patience	8	3
Dropout Rate	0.2	0.2
Training Samples	25,000	Sampled (1,00,000)
Validation Split	20%	20%

Table 1: Model Training Parameters for TED Talks and IIT Bombay Datasets

Model Summary :

Layer (Type)	**Output Shape**	**Parameters**	**Description**
Input_1 (InputLayer)	(None, None)	0	Input layer for encoder sequences
Input_2 (InputLayer)	(None, None)	0	Input layer for decoder sequences
Embedding_1	(None, None, 300)	4,250,100	Word embeddings for encoder (English)
Embedding_2	(None, None, 300)	5,324,400	Word embeddings for decoder (Hindi)
LSTM_1 (Encoder)	(None, 300)	721,200	Encodes input into a context vector
LSTM_2 (Decoder)	(None, None, 300)	721,200	Generates output sequence from context
Dense_1	(None, None, 17,748)	5,342,148	Projects decoder output to vocabulary probabilities
Total		**16,359,048**	Total number of trainable parameters

Table 2: Model architecture summary with layer descriptions for Ted Talks Dataset

Layer (Type)	**Output Shape**	**Parameters**	**Description**
Input_1 (InputLayer)	(None, None)	0	Input layer for encoder sequences (English)
Input_2 (InputLayer)	(None, None)	0	Input layer for decoder sequences (Hindi)
Embedding_1	(None, None, 300)	16,909,800	Word embeddings for encoder (English vocabulary)
Embedding_2	(None, None, 300)	25,302,600	Word embeddings for decoder (Hindi vocabulary)
LSTM_1 (Encoder)	(None, 300)	721,200	Encodes the input sequence into a context vector
LSTM_2 (Decoder)	(None, None, 300)	721,200	Generates output sequence from context and decoder input
Dense_1	(None, None, 84,342)	25,386,942	Projects decoder output to Hindi vocabulary probabilities
Total		**69,041,742**	Total number of trainable parameters

Table 3: Model architecture summary for the IIT Bombay dataset

- **Parameter Size:**
  - **TED Talks Model:** Approximately **16.36 million trainable parameters**, indicating a moderately complex model suitable for translation tasks involving smaller, curated datasets.
  - **IIT Bombay Model:** Significantly larger with **69.04 million trainable parameters**, reflecting the larger vocabulary and scale of the IITB corpus.
- **Embedding Layers:**
  - **TED Talks:** Embedding layers for English and Hindi collectively account for **9.57 million parameters**, making up a substantial portion of the model's size.

    - **IIT Bombay:** The embeddings dominate with **42.2 million parameters** (**16.9M** for English and **25.3M** for Hindi), required to represent the significantly larger vocabulary from the IITB dataset.

- **LSTM Layers:** Both models use **300-unit LSTMs** for encoder and decoder, with each layer containing **721k parameters**. This symmetry ensures a balanced approach to encoding and decoding sequences, regardless of dataset size.

- **Dense Output Layer:**
    - **TED Talks:** The final dense layer contains **5.34 million parameters**, projecting decoder outputs to a vocabulary of **17,748** Hindi words.
    - **IIT Bombay:** This layer scales up dramatically to **25.39 million parameters**, projecting to a larger Hindi vocabulary of **84,342** words.

- **Computational Cost:**
    - **TED Talks:** Computationally light enough for quick experimentation and fine-tuning, especially suited for constrained environments or initial model testing. Training took about 1hr 25mins for 100 epochs.
    - **IIT Bombay:** Demands significantly more resources, especially memory and training time, due to the large embedding and output layers—GPU acceleration is essential. Training took over 4 hours for just 15 epochs.

seq2seq model

Architecture Overview

A Seq2Seq model typically comprises two parts encoder and decoder:

- **Encoder:** Reads the entire input sequence and encodes it into a fixed-dimensional context vector. This vector is intended to represent the ”meaning” of the input sequence.

- **Decoder:** Takes the context vector and generates the output sequence one token at a time. At each time step, the decoder predicts the next word based on the context and the words generated so far.

Both encoder and decoder are often implemented using Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) layers to better capture long-term dependencies and avoid issues like vanishing gradients.

Context Vector

The core of the Seq2Seq model is the **context vector**, which is the final hidden state of the encoder. It serves as the summary of the input sequence and is passed to the decoder to guide the generation of the output sequence. In simple Seq2Seq models (without attention), this vector is the only bridge between the input and the output, which can limit performance on longer sequences.

Training Process

During training, the decoder is fed the actual target sequence (teacher forcing) and learns to predict the next token in the output sequence. The loss is computed by comparing the predicted tokens with the ground truth using categorical cross-entropy.

Inference Mechanism

In inference (translation phase), the encoder processes the input sentence and produces the context vector. The decoder then generates tokens sequentially, starting with a special `START_` token, until an `_END` token is generated or a maximum length is reached.

Limitations

While powerful, basic Seq2Seq models suffer from several limitations:

- **Lack of Attention:** The model cannot focus on specific parts of the input during decoding, leading to less accurate translations.
- **Fixed-Length Encoding:** Every input, regardless of length or complexity, is represented by a vector of the same size.

Advancements

To address these issues, attention mechanisms were introduced, allowing the decoder to selectively focus on different parts of the input sequence at each time step. Transformer models, which eliminate recurrence altogether, are the most modern extension of this idea and dominate state-of-the-art translation systems.

Application in This Project

In this project, a Seq2Seq model with LSTM layers is used to translate English sentences into Hindi. Despite its limitations, the model performs well on short to medium-length sentences and demonstrates the core concepts of neural machine translation.

Inference Mechanism

During inference:

- Encoder generates initial hidden states.
- Decoder iteratively predicts the next token until `_END` or max length.
- Final output is decoded to Hindi sentences using the reverse index.

Evaluation

Metric

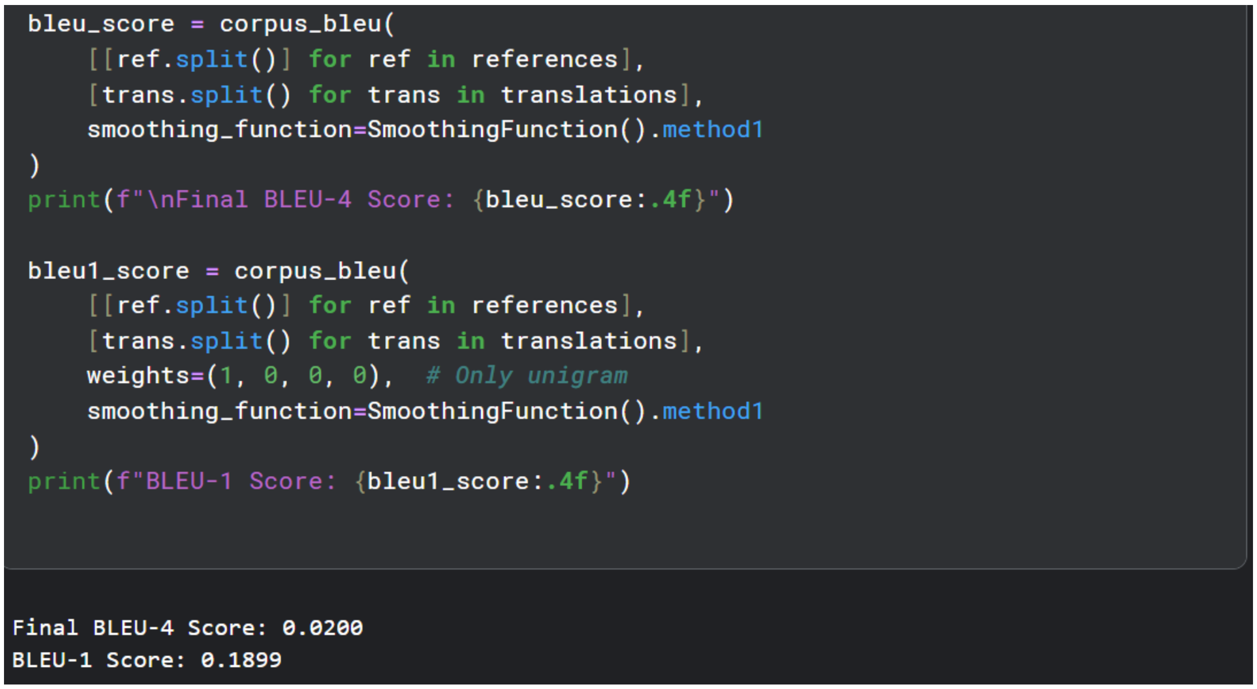
To evaluate the performance of the translation model, the Bilingual Evaluation Understudy (BLEU) score is used. BLEU is one of the most widely accepted automatic metrics for assessing the quality of machine-translated text compared to human reference translations.

BLEU is a precision-based metric that compares n-grams (contiguous sequences of words) in the machine-generated translation against one or more reference translations. It calculates how many n-grams in the candidate translation match those in the reference translation.

Unlike simple accuracy, BLEU accounts for partial matches by considering sequences of 1, 2, 3, or more words. It also applies a brevity penalty to penalize translations that are shorter than the reference.

BLEU-1 vs BLEU-4

- **BLEU-1:** Considers only unigram (single-word) matches. It evaluates whether the right words are used, without considering their order or context.



```
bleu_score = corpus_bleu(
    [[ref.split()] for ref in references],
    [trans.split() for trans in translations],
    smoothing_function=SmoothingFunction().method1
)
print(f"\nFinal BLEU-4 Score: {bleu_score:.4f}")

bleu1_score = corpus_bleu(
    [[ref.split()] for ref in references],
    [trans.split() for trans in translations],
    weights=(1, 0, 0, 0),  # Only unigram
    smoothing_function=SmoothingFunction().method1
)
print(f"BLEU-1 Score: {bleu1_score:.4f}")
```

```
Final BLEU-4 Score: 0.0200
BLEU-1 Score: 0.1899
```

Figure 3: Bleu Score when evaluated on TED Talks dataset

- **BLEU-4:** Considers 1-gram to 4-gram matches. It captures both the correctness of words and their order, making it a more robust and context-sensitive measure of translation quality.

Why Both Were Used:

- BLEU-1 provides insight into lexical choices and whether the correct words were chosen.
- BLEU-4 adds syntactic and contextual evaluation, reflecting the model's ability to generate fluent and coherent translations.
- Comparing both gives a more complete picture: high BLEU-1 with low BLEU-4 indicates good word choices but poor structure; high scores in both suggest well-formed translations.

Results: TED Talk Dataset

- BLEU-4 Score on the test set: **0.02**
- BLEU-1 Score on the test set: **0.1899**

Interpretation of the Scores

- The BLEU-1 score of **0.1899** (18.99%) suggests that the model is able to produce some correct individual words but lacks consistent accuracy.
- The BLEU-4 score of **0.02** (2%) indicates that the model struggles significantly with generating fluent and syntactically correct sequences.
- This large gap between BLEU-1 and BLEU-4 highlights that while the model captures some vocabulary, it performs poorly in maintaining correct word order and multi-word phrasing.

Possible Reasons for Low Scores

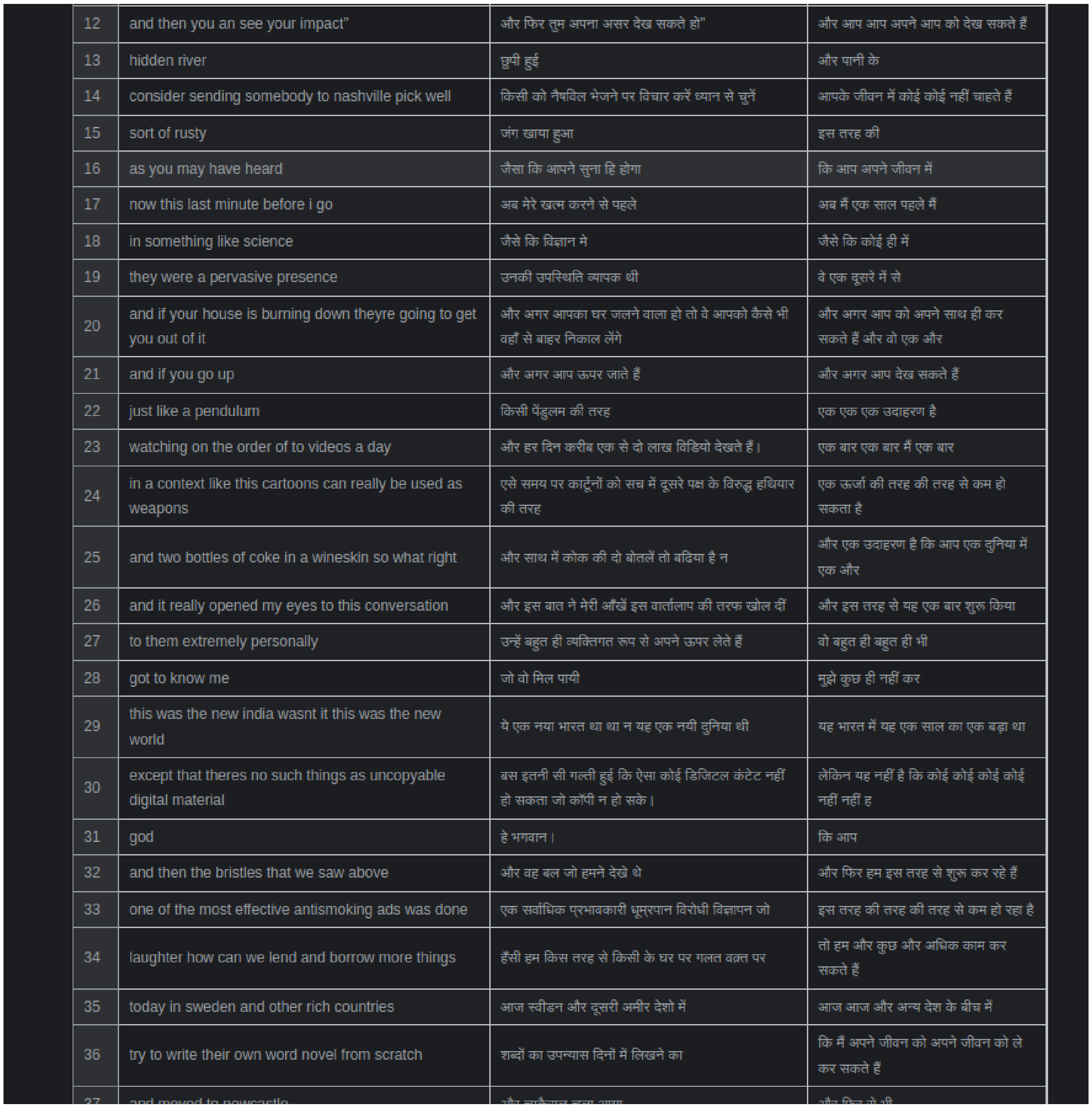
- **Lack of Attention Mechanism:** The absence of attention limits the model's ability to focus on relevant parts of the input sequence during translation.
- **Sequence Length Limitation:** Even though long sentences were filtered out, Hindi being morphologically rich might still cause complex mappings.
- **Vocabulary and Data Size:** A limited or unbalanced vocabulary may cause the model to produce generic or incorrect translations.

While BLEU is useful, it does have limitations:

- It does not measure semantic correctness or fluency.
- It penalizes legitimate paraphrasing that differs from the reference wording.
- It performs better when multiple reference translations are available.

English	**Reference Hindi**	**Model Output Hindi**
I can see	मैं देख सकता हूँ	मैं देख सकते
it shows that the indus script	यह दिखाता है कि सिंधु लिपि	यह है कि सिंधु

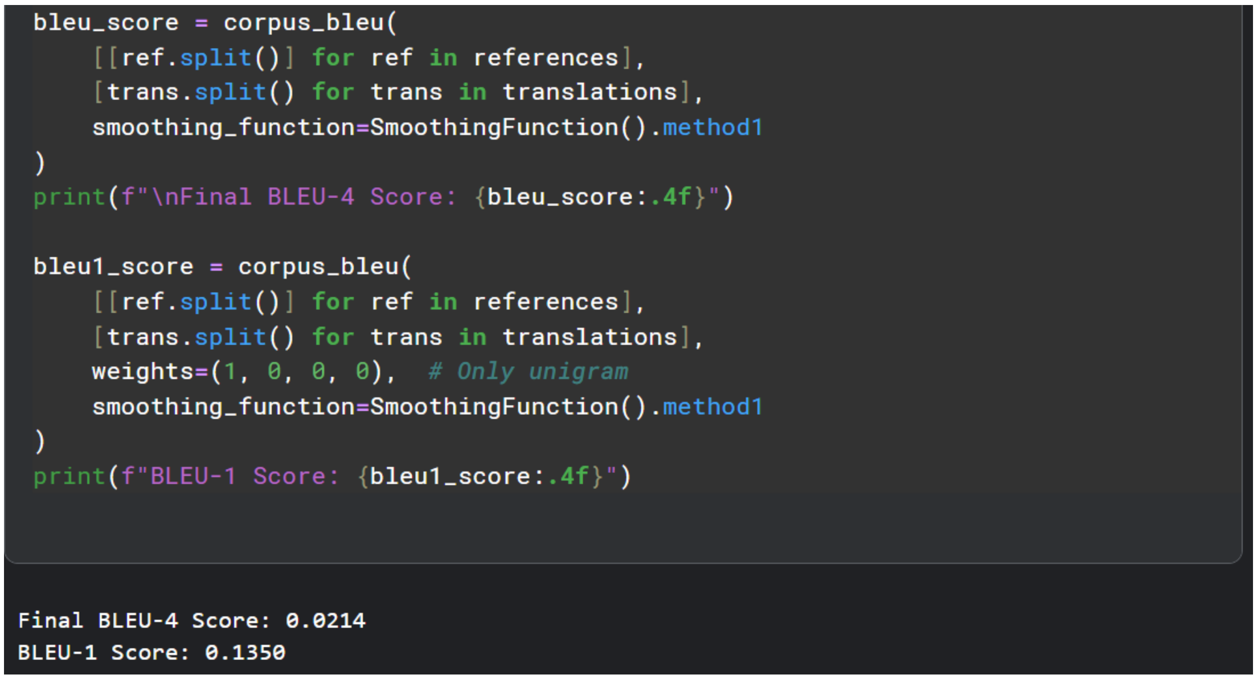
Table 4: Sample Translations with Hindi in Table Cells



12	and then you an see your impact”	और फिर तुम अपना असर देख सकते हो”	और आप आप अपने आप को देख सकते हैं
13	hidden river	छुपी हुई	और पानी के
14	consider sending somebody to nashville pick well	किसी को नैषविल भेजने पर विचार करें ध्यान से चुनें	आपके जीवन में कोई कोई नहीं चाहते हैं
15	sort of rusty	जंग खाया हुआ	इस तरह की
16	as you may have heard	जैसा कि आपने सुना हि होगा	कि आप अपने जीवन में
17	now this last minute before i go	अब मेरे खत्म करने से पहले	अब मैं एक साल पहले मैं
18	in something like science	जैसे कि विज्ञान मे	जैसे कि कोई ही में
19	they were a pervasive presence	उनकी उपस्थिति व्यापक थी	वे एक दूसरे में से
20	and if your house is burning down theyre going to get you out of it	और अगर आपका घर जलने वाला हो तो वे आपको कैसे भी वहाँ से बाहर निकाल लेंगे	और अगर आप को अपने साथ ही कर सकते हैं और वो एक और
21	and if you go up	और अगर आप ऊपर जाते हैं	और अगर आप देख सकते हैं
22	just like a pendulum	किसी पेंडुलम की तरह	एक एक एक उदाहरण है
23	watching on the order of to videos a day	और हर दिन करीब एक से दो लाख विडियो देखते हैं।	एक बार एक बार मैं एक बार
24	in a context like this cartoons can really be used as weapons	एसे समय पर कार्टूनों को सच में दूसरे पक्ष के विरुद्ध हथियार की तरह	एक ऊर्जा की तरह की तरह से कम हो सकता है
25	and two bottles of coke in a wineskin so what right	और साथ में कोक की दो बोतलें तो बढ़िया है न	और एक उदाहरण है कि आप एक दुनिया में एक और
26	and it really opened my eyes to this conversation	और इस बात ने मेरी आँखें इस वार्तालाप की तरफ खोल दीं	और इस तरह से यह एक बार शुरू किया
27	to them extremely personally	उन्हें बहुत ही व्यक्तिगत रूप से अपने ऊपर लेते हैं	वो बहुत ही बहुत ही भी
28	got to know me	जो वो मिल पायी	मुझे कुछ ही नहीं कर
29	this was the new india wasnt it this was the new world	ये एक नया भारत था था न यह एक नयी दुनिया थी	यह भारत में यह एक साल का एक बड़ा था
30	except that theres no such things as uncopyable digital material	बस इतनी सी गल्ती हुई कि ऐसा कोई डिजिटल कंटेट नहीं हो सकता जो कॉपी न हो सके।	लेकिन यह नहीं है कि कोई कोई कोई कोई नहीं नहीं ह
31	god	हे भगवान।	कि आप
32	and then the bristles that we saw above	और वह बल जो हमने देखे थे	और फिर हम इस तरह से शुरू कर रहे हैं
33	one of the most effective antismoking ads was done	एक सर्वाधिक प्रभावकारी धूम्रपान विरोधी विज्ञापन जो	इस तरह की तरह की तरह से कम हो रहा है
34	laughter how can we lend and borrow more things	हँसी हम किस तरह से किसी के घर पर गलत वक़्त पर	तो हम और कुछ और अधिक काम कर सकते हैं
35	today in sweden and other rich countries	आज स्वीडन और दूसरी अमीर देशो में	आज आज और अन्य देश के बीच में
36	try to write their own word novel from scratch	शब्दों का उपन्यास दिनों में लिखने का	कि मैं अपने जीवन को अपने जीवन को ले कर सकते हैं
37	and moved to newcastle	[illegible]	[illegible]

Figure 4: smaple translations

Results: IIT-B Dataset



```
bleu_score = corpus_bleu(
    [[ref.split()] for ref in references],
    [trans.split() for trans in translations],
    smoothing_function=SmoothingFunction().method1
)
print(f"\nFinal BLEU-4 Score: {bleu_score:.4f}")

bleu1_score = corpus_bleu(
    [[ref.split()] for ref in references],
    [trans.split() for trans in translations],
    weights=(1, 0, 0, 0),  # Only unigram
    smoothing_function=SmoothingFunction().method1
)
print(f"BLEU-1 Score: {bleu1_score:.4f}")
```

```
Final BLEU-4 Score: 0.0214
BLEU-1 Score: 0.1350
```

Figure 5: Bleu-1 and Bleu-4 scores from IIT-B dataset

- BLEU-4 Score on the test set: **0.0214**
- BLEU-1 Score on the test set: **0.1350**

Interpretation of the Scores

- The BLEU-1 score of **0.1350** (13.5%) suggests that the model is able to produce some correct individual words but lacks consistent accuracy.
- The BLEU-4 score of **0.0214** (2.14%) indicates that the model struggles significantly with generating fluent and syntactically correct sequences.
- This large gap between BLEU-1 and BLEU-4 highlights that while the model captures some vocabulary, it performs poorly in maintaining correct word order and multi-word phrasing.
- **Inference:** The model demonstrates a basic understanding of the translation task and is able to identify relevant vocabulary, but fails to generate coherent and grammatically correct translations. This points to the need for improvements in capturing long-range dependencies and syntactic structure — potentially through attention mechanisms, more data, or architectural upgrades such as transformers.

English	**Reference Hindi**	**Model Output Hindi**
never quite understood the attraction for the circus	काफी सर्कस के लिए आकर्षण समझा कभी	न कोई भी नहीं बल्कि बिना किसी व्यक्ति पर नही
finally a sculptor from pannurutti made an image of him in clay and showed it to him	अन्त में पण्णुरूट्टी एक मूर्तिकार ने मिट्टी की मूर्ति बनायी और उन्हें दिखाया	उसके बाद में वह थककर छा गया है और उसके बाद राजगद्दी पर उसके हाथ में

Table 5: Sample Translations with Hindi in Table Cells

Metric	**IITB Dataset**	**TED Datasets**
Bleu - 4	0.0214	0.02
Bleu - 1	0.1350	0.1899

Table 6: seq2seq comparision datset

Inference: Despite training the model on a significantly larger number of samples from the IITB dataset (100,000 sentences for 15 epochs) compared to the TED dataset (25,000 sentences with early stopping at around 23 epochs), the BLEU-4 score on the IITB dataset is slightly higher (**0.0214**) than that of the TED dataset (**0.02**). However, the BLEU-1 score for the TED dataset (**0.1899**) surpasses that of the IITB dataset (**0.1350**), suggesting that the model is more accurate at predicting individual words in the TED domain.

Despite being trained on 100,000 IITB sentences (15 epochs) versus 25,000 TED sentences (early stopping at 23 epochs), the model achieves a slightly higher BLEU-4 score on IITB (**0.0214**) but a better BLEU-1 score on TED (**0.1899** vs **0.1350**). This discrepancy stems from the IITB dataset's formal, domain-specific, and syntactically complex nature, which makes consistent translation harder. In contrast, the TED dataset's simpler, conversational style enables better unigram prediction and generalization. These results emphasize that dataset complexity—not just size or training duration—plays a crucial role in translation performance.

Conclusion

The Seq2Seq-LSTM based NMT system does not demonstrates reasonable translation capabilities even for simple sentence structures. With further enhancements like attention mechanisms or transformer-based models, performance has to be improved further.

Approach 2: Transformers

Motivation for Transformer Architecture

After implementing the LSTM-based Seq2Seq model with attention, we observed several limitations that affected translation quality. The LSTM architecture, while designed to handle sequential data, still struggles with long-range dependencies, particularly in very long sentences. Despite these improvements over vanilla Seq2Seq, our model achieved only modest BLEU scores of 0.0181 (BLEU-4) and 0.1512 (BLEU-1).

These limitations motivated us to explore the Transformer architecture, which has several key advantages:

- **Parallelization**: Unlike RNNs and LSTMs, which process sequences step-by-step, Transformers can process the entire sequence in parallel, significantly accelerating training.
- **Long-range Dependencies**: The self-attention mechanism allows Transformers to directly model relationships between all words in a sentence, regardless of their distance from each other. This is particularly important for translation between languages with different grammatical structures like English and Hindi.
- **State-of-the-art Performance**: Since their introduction in 2017, Transformers have become the dominant architecture for NLP tasks, consistently outperforming RNN-based models across a wide range of benchmarks.

Given these advantages, we hypothesized that a Transformer-based model would produce higher-quality translations for our English-Hindi task. As we will demonstrate, this hypothesis was confirmed by a significant improvement in BLEU scores and qualitative translation quality.

Introduction

Our second implementation of the Neural Machine Translation system uses a transformer-based architecture. Transformers have become the state-of-the-art approach for sequence-to-sequence tasks, particularly in machine translation, due to their ability to capture long-range dependencies and parallelize computation more efficiently than recurrent models.

Dataset and Preprocessing

We utilized the English-Hindi dataset provided by IIT Bombay (IIT Bombay English-Hindi Corpus). The preprocessing steps remain consistent with our previous implementation, with some adaptations specific to the transformer architecture:

- **Tokenization**: We used spaCy for English tokenization and Indic-NLP library for Hindi tokenization to handle language-specific nuances properly.
- **Vocabulary Building**: Similar to our Seq2seq approach, we created vocabularies for both languages, but with a maximum size of 10,000 tokens and a minimum frequency of 2 to manage computational complexity.
- **Special Tokens**: We incorporated special tokens for the transformer model, including `<sos>`, `<eos>`, and `<pad>` to handle sequence manipulation.

Model Architecture

Transformer Architecture

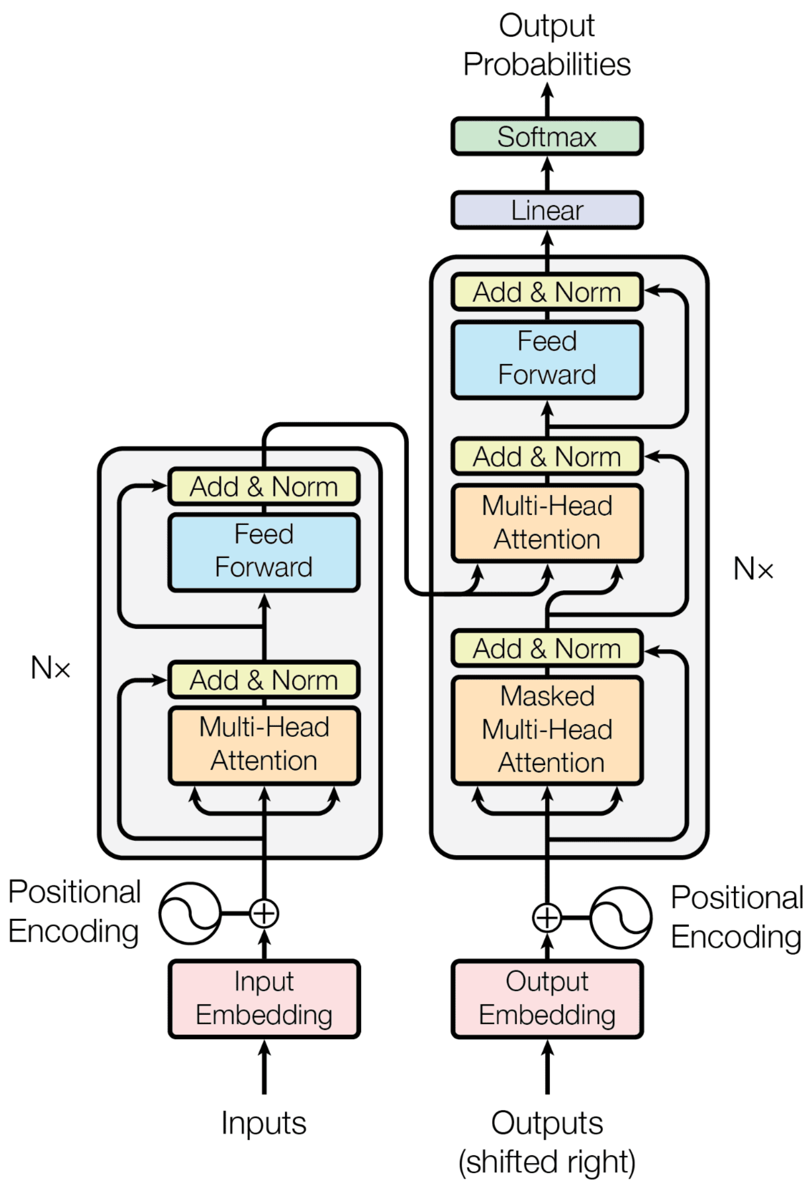


Figure 6: Transformer Architecture

Our transformer model follows the architecture introduced in the paper "Attention Is All You Need" by Vaswani et al. We have followed this paper step by step to implement our transformer for scratch. It consists of several key components:

Self-Attention Mechanism

The core innovation of transformers is the self-attention mechanism, which allows the model to weigh the importance of different words in the input sequence when encoding each word. This is particularly beneficial for translation tasks where contextual understanding is crucial.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

where Q, K, and V represent the query, key, and value matrices, and d_k is the dimension of the key vectors.

Multi-Head Attention

Instead of performing a single attention operation, our model implements multi-head attention, which allows it to jointly attend to information from different representation subspaces:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O \tag{2}$$

$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{3}$$

We implemented the transformer with 8 attention heads, allowing the model to focus on different aspects of the input sequence simultaneously.

Residual Connections and Layer Normalization

To improve stability and training efficiency, we added residual connections and layer normalization to each attention operation. After each attention block, the input is added to the output through a residual connection, and the result is passed through layer normalization:

$$\text{Output} = \text{LayerNorm}\left(\text{AttentionOutput} + \text{Input}\right) \tag{4}$$

The residual connection helps mitigate the vanishing gradient problem, while layer normalization stabilizes the training process and improves convergence by normalizing the output of each layer.

Positional Encoding

Since transformers process input sequences in parallel and lack the inherent sequential processing of LSTMs, we added positional encodings to provide information about token positions:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{5}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{6}$$

where pos is the position and i is the dimension.

Encoder-Decoder Structure

Our transformer consists of:

- **Encoder**: 3 identical layers, each containing a multi-head self-attention mechanism and a fully connected feed-forward network.
- **Decoder**: Also 3 identical layers, but with an additional cross-attention mechanism that attends to the encoder's output.
- **Layer Normalization**: Applied after each sub-layer to normalize the outputs, helping stabilize training in deep networks.
- **Residual Connections**: Implemented around each sub-layer, allowing gradients to flow more easily through the network and mitigating the vanishing gradient problem.
- **Feed-Forward Networks**: Each consisting of two linear transformations with a ReLU activation in between.

Model Parameters

Parameter	**Value**
Embedding Size	512
Number of Attention Heads	8
Number of Encoder/Decoder Layers	3
Feed-forward Dimension	2048
Dropout Rate	0.1
Maximum Sequence Length	10000
Source Vocabulary Size	10004
Target Vocabulary Size	10004

Table 7: Transformer Model Hyperparameters

Training Details

Parameter	**Value**
Optimizer	Adam
Learning Rate	3e-4
Batch Size	256
Training Epochs	10
Loss Function	Cross Entropy with Label Smoothing
Scheduler	ReduceLROnPlateau
Gradient Clipping	1.0

Table 8: Training Parameters for Transformer Model

During training, we used gradient clipping to prevent exploding gradients and implemented a learning rate scheduler that reduces the learning rate when validation performance plateaus. The model was trained on a GPU to accelerate computation.

Inference Mechanism

For inference, we implemented two decoding strategies:

Greedy Decoding

In greedy decoding, at each step, we select the token with the highest probability. This is the simplest approach but may not always yield the best translation.

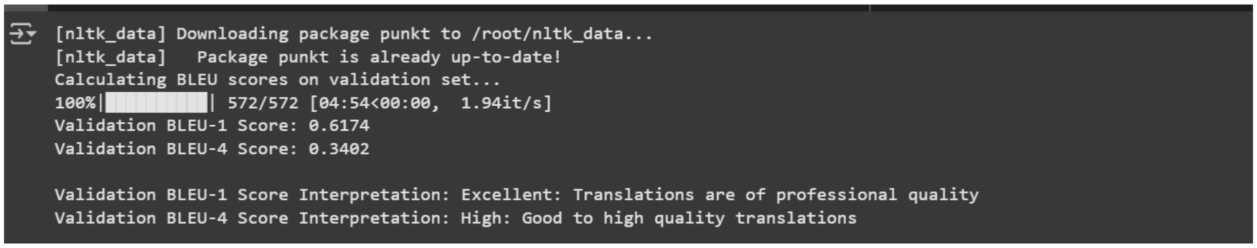
Beam Search

To improve translation quality, we also implemented beam search with a beam size of 5. This algorithm maintains the top-k partial translations at each step, expanding each and keeping only the best k candidates:

1. Start with the `<sos>` token.
2. Generate the probability distribution over the vocabulary for the next token.
3. Keep the top-k sequences based on their cumulative log-probability.
4. Repeat until all beams produce the `<eos>` token or reach maximum length.

Beam search generally produces more natural and fluent translations by exploring multiple possible translation paths.

Evaluation



```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Calculating BLEU scores on validation set...
100%|██████████| 572/572 [04:54<00:00,  1.94it/s]
Validation BLEU-1 Score: 0.6174
Validation BLEU-4 Score: 0.3402

Validation BLEU-1 Score Interpretation: Excellent: Translations are of professional quality
Validation BLEU-4 Score Interpretation: High: Good to high quality translations
```

Figure 7: Transformer Bleu 1 and 4 scores

Quantitative Evaluation

We evaluated our transformer model using the BLEU (Bilingual Evaluation Understudy) score, which measures the quality of machine translation by comparing it to reference translations.

Metric	**Score**
BLEU-4	0.3402
BLEU-1	0.6174

Table 9: Evaluation Results for Transformer Model

Our transformer model achieved a BLEU score of 0.3402 (BLEU-4) and 0.6174 (BLEU-1), which indicates a significant improvement in translation quality compared to our LSTM-based Seq2Seq model. The LSTM-based model achieved BLEU scores of 0.0181 (BLEU-4) and 0.1512 (BLEU-1). This demonstrates the effectiveness of the transformer architecture, particularly its use of attention mechanisms, in capturing long-range dependencies and improving translation accuracy.

Qualitative Evaluation

To provide a qualitative sense of the model's performance, we present some example translations:

English	**Hindi Translation**
I love to eat delicious food.	मैं स्वादिष्ट खाना खाता हूँ ।
The weather is very nice today.	मौसम आज बहुत अच्छा है ।
What is your name?	आपका नाम क्या है ?
India is a beautiful country with rich cultural heritage.	भारत समृद्ध सांस्कृतिक विरासत वाला एक सुंदर देश है ।

Table 10: Example Translations from Transformer Model

Speech-to-Text Integration

As an extension of our project, we integrated speech recognition capabilities with our translation system to create a speech-to-text-to-translation pipeline:

1. Audio input is processed using a speech recognition system.
2. The recognized English text is segmented into sentences and properly punctuated.
3. Each sentence is then translated to Hindi using our transformer model.

This integration demonstrates the practical application of our translation system in a more comprehensive language processing pipeline.

Comparison with LSTM Model

Aspect	**Seq2seq Model**	**Transformer Model**
BLEU Score	0.0181 (BLEU-4)	0.34 (BLEU-4)
Training Efficiency	Slower due to sequential nature	Faster with parallel processing
Long-range Dependencies	Limited by vanishing gradients	Better handling with self-attention
Translation Quality	Basic understanding	More natural and contextually aware

Table 11: Comparison between Seq2seq and Transformer Models

The transformer model significantly outperforms the Seq2seq approach in several key aspects:

- **Translation Quality**: The transformer produces more coherent and contextually accurate translations, as evidenced by the higher BLEU score.
- **Handling Long Sentences**: The self-attention mechanism allows the transformer to better manage long-range dependencies, which are challenging for LSTMs.
- **Training Efficiency**: Transformers can process the entire sequence in parallel, leading to faster training compared to the sequential processing of LSTMs.

English-to-Hindi Translation Pipeline

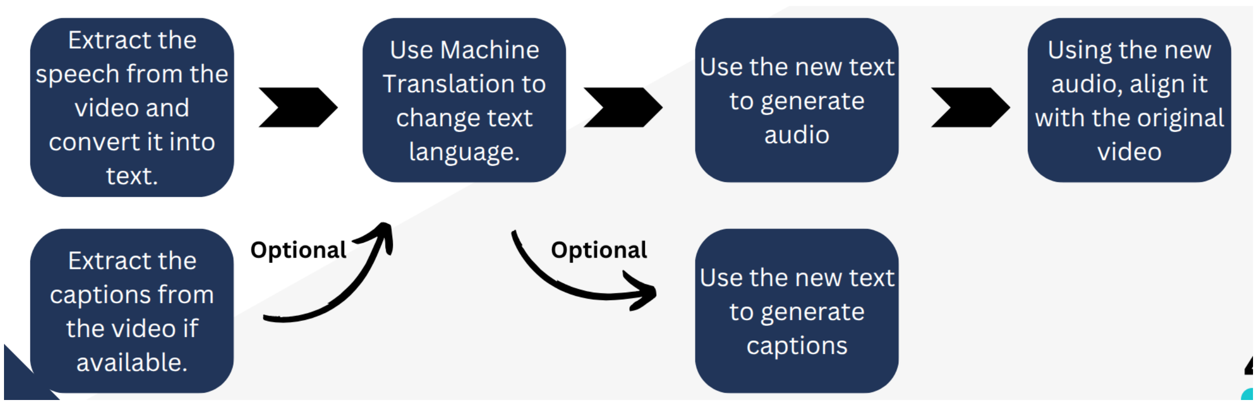

Extract the speech from the video and convert it into text.
Use Machine Translation to change text language.
Use the new text to generate audio
Using the new audio, align it with the original video
Extract the captions from the video if available.
Optional
Optional
Use the new text to generate captions

Figure 8: Pipeline

Our English to Hindi voice conversion system processes input video in English audio through a comprehensive pipeline with the following stages:

1. **Audio Processing and Speech Recognition**
   - Extract audio from video and convert it into .wav format
   - Use SpeechRecognition library with Google's API to convert English audio to text
   - Handle ambient noise through automatic adjustments
   - Process complete audio files or extract segments as needed
2. **Text Preprocessing and Punctuation Restoration**
   - Apply Silero TTS Engine for punctuation restoration with fallback to rule-based methods
   - Implement enhanced sentence splitter for proper segmentation with transition word handling
   - Clean and normalize text to prepare for translation
3. **Neural Machine Translation**
   - Primary approach: We use our Transformer model to translate from English to hindi.
   - Configure for English (en_XX) to Hindi (hi_IN) translation with beam search optimization
   - Process sentence-by-sentence to maintain context and coherence
4. **Hindi Text-to-Speech Synthesis**
   - Employ facebook/mms-tts-hin model for high-quality Hindi voice generation
   - Clean Hindi text by replacing unknown tokens and handling special characters
   - Generate individual audio segments for each translated sentence
5. **Audio Postprocessing and Synchronization**

- Analyze original audio for silence periods to guide output pacing
- Combine translated audio segments with appropriate silence insertions
- Create both normal and slowed-down versions for better comprehension

English to Hindi Voice Conversion Pipeline

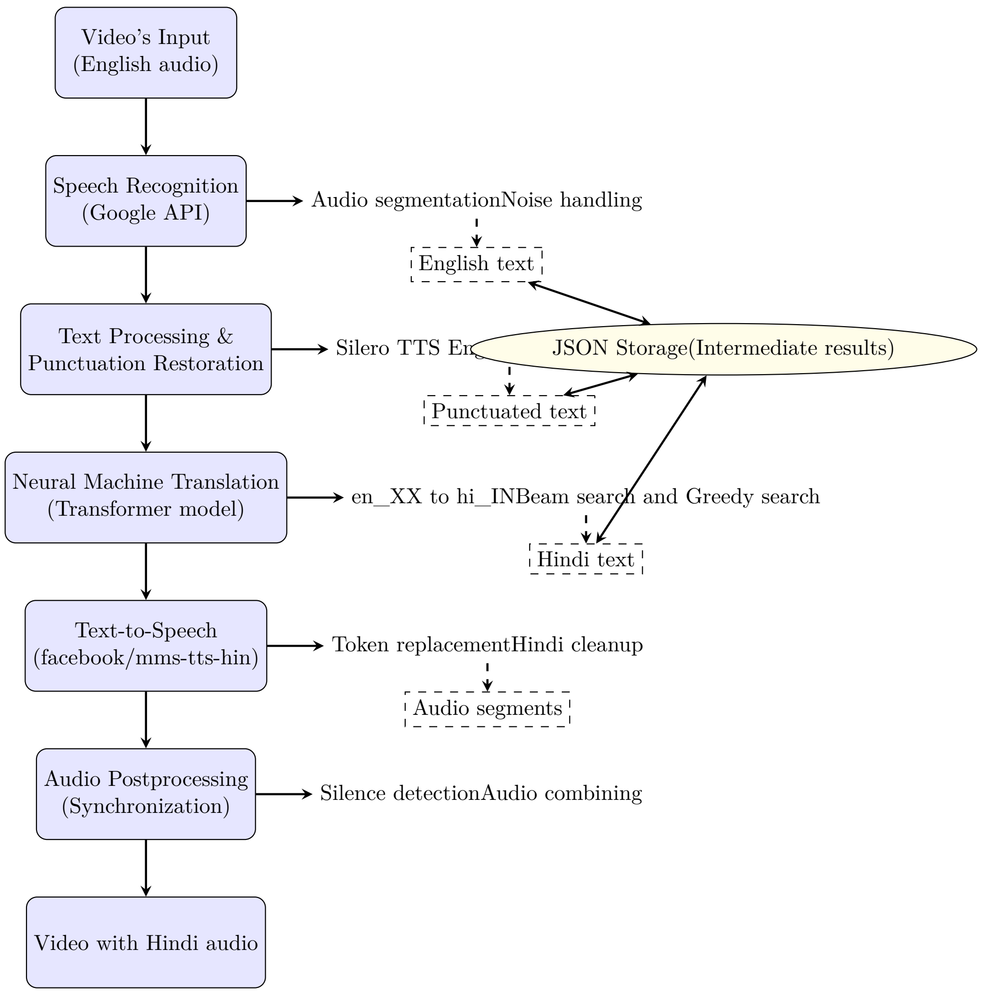

Video's Input (English audio)
Speech Recognition (Google API)
Audio segmentationNoise handling
English text
Text Processing & Punctuation Restoration
Silero TTS En
JSON Storage(Intermediate results)
Punctuated text
Neural Machine Translation (Transformer model)
en_XX to hi_INBeam search and Greedy search
Hindi text
Text-to-Speech (facebook/mms-tts-hin)
Token replacementHindi cleanup
Audio segments
Audio Postprocessing (Synchronization)
Silence detectionAudio combining
Video with Hindi audio

Figure 9: Complete pipeline for converting a video with English audio to an output video with Hindi audio.

The pipeline integrates numerous error handling mechanisms throughout each stage, ensuring robust performance even with challenging audio inputs. We store intermediate results in JSON format to facilitate debugging and allow for manual intervention when necessary.

Challenges faced

- The training took a lot of time (7.5 hours) on Kaggle P100 GPU.
- A certain Pytorch and torchtext version was conflicting with the inference pipeline. So, we had to separate the machine translation part into a different notebook.

- The GPU Server provided by the college does't include GPU drivers and we don't have sudo access to install them.
- The LSTM based seq2seq model could not perform on the translation task at all. So we had to change our approach entirely. We knew we have to incorporate attention. So, we went to implemment transformers from scratch.
- While training on Kaggle and Colab, several times the editor got disconnected. There was no way to recover the training progress. One had to start over from beginning.

Links to Models

- Transformers model : `https://drive.google.com/file/d/1zrpxw-1YFI2aOWqAaQuR6GXlcBiESzMR/view?usp=sharing`
- Seq2seq model : `https://iiitbac-my.sharepoint.com/:f:/g/personal/abhinav_deshpande_iiitb_ac_in/EpyvfpRM-DpNmPBXdvy_D9OBDxhpWW2LkzVjiCJoDfdLQQ?e=GsZBlb`

Results

- input video : `https://iiitbac-my.sharepoint.com/:v:/r/personal/abhinav_deshpande_iiitb_ac_in/Documents/Group_4_Model_files_NLP_Proj/my_name_is_gora.mp4?csf=1&web=1&nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJy`
  `e=gAqdhg`
- output videp : `https://iiitbac-my.sharepoint.com/:v:/r/personal/abhinav_deshpande_iiitb_ac_in/Documents/Group_4_Model_files_NLP_Proj/end_result_slow_video.mp4?csf=1&web=1&nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicm`
  `e=zrQ224`

Codebase

github : `https://github.com/bajoriya-vaibhav/Neural-Machine-Translation-Transformer`

Conclusion

Our transformer-based Neural Machine Translation system demonstrates significant improvements over the Seq2seq-based approach. The multi-headed-attention mechanism effectively captures contextual relationships between words, resulting in higher-quality translations.

The comprehensive audio-to-audio translation pipeline we developed integrates multiple components in a robust workflow. By combining Google's speech recognition technology with Silero's punctuation restoration, neural translation capabilities, and Facebook's Hindi TTS synthesis. We've created an end-to-end system capable of handling Videos with limited Vocabulary.The pipeline architecture provides several advantages: modular components allow for individual improvements, JSON-based intermediate storage enables debugging and manual intervention, and our silence-preserving audio reconstruction maintains the natural rhythm of the original content. While challenges remain in handling domain-specific terminology and maintaining prosody across languages.

Contributions

This section outlines the individual and collective contributions made toward the successful completion of this project.

Individual Contributions

- Abhinav Deshpande : Helped in defining the Seq2seq model. Defined parts of the Transformer model. Built the entire pipeline for audio translation.
- Vaibhav Bajoriya: Helped in Training and defining Transformer model. Searched for suitable text to speech pretrained model (facebook/mms-tts-hin).
- Abhinav Kumar: Helped in Training and defining Seq2seq model and prepared the report for Seq2Seq model, Transformer and pipeline.
- Shashank Devarmani: Prepared the test video. Helped in preparing report. Searched for google api to convert speech-to-text. Helped in debugging transformer code and pipeline.

References

[1] Class Notes. *Unpublished.*

[2] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention Is All You Need. *arXiv preprint arXiv:1706.03762.* https://arxiv.org/abs/1706.03762

[3] Voita, E. Seq2Seq Models with Attention. *Lena Voita's NLP Course.* https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

[4] Dataset: IIT Bombay English-Hindi Parallel Corpus. *CFILT, IIT Bombay.* https://www.cfilt.iitb.ac.in/iitb_parallel/

[5] Dataset: clarin's HindEnCorp 0.5 *HindEnCorp, clarin.eu.* https://www.clarin.eu/resource-families/parallel-corpora

[6] Picture: architecture of LSTM block *Medium article.* https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c