Journal of Next Generation Technology (ISSN: 2583-021X)
Vol. 5, Issue 1, March 2025

ANUVAAD ARJUNA: A HYBRID AI PLATFORM FOR REAL-TIME ENGLISH-TO-HINDI TRANSLATION WITH A FINE-TUNED TRANSFORMER AND AN LSTM ENCODER-DECODER MODEL

Rajareddy S[1], M Adhitya [2], L. Nitish[3], Madhulika Das[4]
[1,2,3]UG Scholar, [4]Assistant Professor
Dept of EEE, Chaitanya Bharathi Institute of Technology, Gandipet, Hyderabad, India
Email id: rajurajareddy74@gmail.com[1], Adhitya.0315@gmail.com [2], lankalapallinitish@gmail.com[3], madhulikadaseee@cbit.ac.in[4]

Article Received: 20 March 2025 *Article Accepted: 25 March 2025* *Article Published: 31 March 2025*

Citation
Rajareddy S, M Adhitya , L. Nitish, Madhulika Das, "Anuvaad Arjuna: A Hybrid Ai Platform For Real-Time English-To-Hindi Translation With A Fine-Tuned Transformer And An LSTM Encoder-Decoder", Journal of Next Generation Technology (ISSN: 2583-021X), 5(1), pp. 52-64. March 2025. https://doi.org/10.5281/zenodo.15270408

Abstract

This paper introduces Anuvaad Arjuna, a hybrid AI platform for real-time English-to-Hindi translation, combining fine-tuned transformer models and custom LSTM encoder-decoder architectures. The system fine-tunes the Helsinki-NLP MarianMT transformer using Hugging Face and trains a LSTM with Encoder-Decoder Model from scratch on a IIT Bombay English-Hindi Parallel Corpus. The fine-tuned transformer achieves a BLEU score of 25.8, outperforming the LSTM (14.5 BLEU), demonstrating trade-offs between accuracy and computational efficiency. Deployed via a Flask-based web interface, the platform offers real-time translation. The transformer is hosted on Hugging Face Spaces, while the LSTM provides a lightweight alternative. Anuvaad Arjuna bridges language barriers in low- resource settings, offering insights into model selection, fine- tuning, and deployment.

Keywords: Neural Machine Translation (NMT), Trans- formers, LSTM (Long Short-Term Memory), English-to-Hindi Translation, Fine-Tuning, Hugging Face, Flask, Web-Based Inter-face, Real-Time Translation, BLEU Score, MarianMT, Encoder- Decoder Models.

I. INTRODUCTION

The digital transformation of India has created significant demand for high-quality English-to-Hindi translation systems. With Hindi serving as the primary language for over 600 million speakers [1], there is growing need for accurate and efficient translation tools. While neural machine translation has shown remarkable success for high-resource languages [2], Hindi still faces challenges due to limited parallel corpora and computational constraints.

We present Anuvaad Arjuna, a practical translation system that combines:

• A fine-tuned MarianMT transformer [3] achieving 25.8 BLEU score

• A custom LSTM model providing lightweight alternative (14.5 BLEU)

• An efficient web interface with 70ms translation latency.

• Optimization strategies for MarianMT in English-Hindi translation

• Comparative analysis of transformer vs LSTM architectures

• Deployable framework tested with 50+ concurrent users

This work advances NMT for Indic languages [4], with all models and code publicly available to support further research.

II. BACKGROUND AND RELATED WORK

A. Evolution of Neural Machine Translation

The field of machine translation has progressed through three key phases. Statistical approaches dominated until 2014 when Sutskever et al. introduced sequence-to-sequence learning using LSTMs [5]. While these neural models outper formed statistical methods, their sequential processing created fundamental limitations in handling long-range dependencies. The transformer architecture [2] overcame these constraints through self-attention mechanisms, enabling parallel processing of entire sequences. This breakthrough led to models like MarianMT [4] that are particularly effective for low-resource language pairs.

B. Architectural Trade-offs

Transformers and LSTMs present complementary strengths for translation tasks. Transformer-based systems achieve superior accuracy through multi-head attention mechanisms that capture global context, but require substantial computational resources. In contrast, LSTM architectures offer a lightweight alternative that processes sequences incrementally, making them suitable for resource-constrained environments despite their limitations with long sentences [6]. The choice between these approaches depends on the specific deployment con straints and accuracy requirements.

C. English-Hindi Translation Challenges

Hindi's complex morphology and limited parallel corpora present unique obstacles for machine translation. The language exhibits extensive morphological variations where a single verb can take dozens of forms based on gender, number, tense, and aspect. Early rule-based systems [7] struggled with these variations, while modern neural approaches still face challenges with data scarcity. Recent work has shown that transformer architectures can better handle these linguistic complexities when sufficient training data is available [8].

D. Deployment Considerations

Practical deployment of translation systems requires balancing model performance with infrastructure constraints. The Hugging Face ecosystem [3] has significantly simplified trans-former model fine-tuning and deployment, while lightweight frameworks like Flask [9]

enable efficient web integration. These tools have made it feasible to deploy both transformer and LSTM models in production environments, though each architecture presents different trade-offs in terms of accuracy, latency, and resource requirements.

III. MOTIVATION AND OBJECTIVES

A. The Hindi Translation Imperative

India's digital transformation has created unprecedented demand for English-to-Hindi translation systems. With over 600 million native speakers [1], Hindi remains underserved by current machine translation technologies. Three fundamental challenges hinder progress: (1) limited availability of high-quality parallel corpora, (2) Hindi's complex morphological structure requiring sophisticated linguistic handling, and (3) practical constraints in deploying resource-intensive models across India's diverse technological infrastructure.

B. Hybrid Architecture Rationale

This work proposes a dual-model approach to address the accuracy-efficiency trade-off inherent in neural machine translation. Transformer-based models like MarianMT [4] deliver state-of-the-art accuracy but demand substantial computational resources. Our complementary LSTM architecture provides a deployable alternative for resource-constrained environments, achieving 60% reduced memory footprint while maintaining acceptable translation quality. This hybrid strategy ensures the system can scale from cloud servers to mobile devices without compromising accessibility.

C. Core Technical Challenges

Developing robust English-Hindi translation systems requires overcoming several language-specific obstacles:

- Data Scarcity: The lack of domain-specific parallel corpora limits model training opportunities
- Linguistic Complexity: Hindi's rich morphology generates numerous word forms from single roots
- Deployment Constraints: Practical implementations must accommodate varying infrastructure capabilities

D. Research Objectives

This work establishes five key objectives to advance English-Hindi translation:

1) Develop real-time translation capabilities with sub- second latency for practical usability

2) Create a scalable architecture adaptable to both high- performance and edge computing environments

3) Optimize an LSTM-based alternative model for resource-constrained deployments

4) Implement an intuitive web interface using modern development frameworks

5) Release all system components as open-source to support further research

The proposed system aims to set a new standard for accessible, high-quality translation in low-resource language pairs while providing reusable tools for the NLP community.

IV. DATA ACQUISITION AND PREPROCESSING

The success of any Neural Machine Translation (NMT) system heavily depends on the quality and quantity of the training data.

A. Dataset Description

1) **Fine-Tuning Dataset**: The fine-tuning of the MarianMT transformer model utilized the IIT Bombay English-Hindi Parallel Corpus [10], available through the Hugging Face Datasets library. The dataset is structured as follows:

- Training Set: 1,659,083 sentence pairs.
- Validation Set: 520 sentence pairs.
- Test Set: 2,507 sentence pairs.

This dataset was chosen for its large number of sentence pairs ensures robust training for the transformer model & The corpus is well-aligned and curated, minimizing noise and errors.

2) **LSTM Training Dataset**: The custom LSTM encoder- decoder model was trained on the Hindi-English Truncated Corpus, a publicly available dataset containing English-Hindi sentence pairs. The dataset was pre-processed to include sentences with a maximum length of 20 tokens, resulting in a smaller but more manageable corpus for training the LSTM model from scratch.

B. Preprocessing Steps

1) **Fine-Tuning Preprocessing:** The Figure 1, represents the Fine-Tuning Preprocessing Function that was applied to the IIT Bombay dataset. This function showcases the steps involved in preparing the dataset for the translation model, including tokenization, sequence truncation, and dataset splitting. The IIT Bombay dataset was preprocessed using the following steps:

- Tokenization: The Helsinki-NLP/opus-mt-en-hi tokenizer was used to tokenize both English (source) and Hindi (target) sentences. The tokenizer employs subword tokenization, which effectively handles rare words and morphological variations.
- Sequence Truncation: Sentences were truncated to a maximum length of 128 tokens to standardize input sizes.
- Dataset Splitting: The dataset was already split into training, validation, and test sets.

```python
max_input_length = 128
max_target_length = 128
source_lang = "en"
target_lang = "hi"

def preprocess_function(examples):
    inputs = [ex[source_lang] for ex in examples["translation"]]
    targets = [ex[target_lang] for ex in examples["translation"]]
    model_inputs = tokenizer(inputs, max_length=max_input_length, truncation=True)

    # Setup the tokenizer for targets
    with tokenizer.as_target_tokenizer():
        labels = tokenizer(targets, max_length=max_target_length, truncation=True)

    model_inputs["labels"] = labels["input_ids"]
    return model_inputs
```

Fig. 1. Fine-Tuning Preprocessing function.

2) **LSTM Preprocessing**: The Figure 2, LSTM Pre-processing illustrates the specific steps applied to prepare the Hindi-English Truncated Corpus for training an LSTM-based language translation model. These steps ensure that the input data is cleaned, standardized, and suitable for sequential deep learning models like LSTMs. The Hindi-English Truncated Corpus was pre-processed as follows:

- Text Cleaning: All sentences were converted to lowercase, and special characters, numbers, and extra spaces were removed.
- Vocabulary Creation: Separate vocabularies were created for English and Hindi words, with each word as signed a unique index.
- Sequence Truncation: Sentences were truncated to a maximum length of 20 tokens to ensure uniformity and reduce computational complexity.
- Dataset Splitting: The dataset was split into training (80%) and test (20%) sets using a random seed for reproducibility. LSTM process is represented in Fig. 2.

```python
# Text cleaning
lines['english_sentence'] = lines['english_sentence'].str.lower()
lines['hindi_sentence'] = lines['hindi_sentence'].str.lower()
lines = lines[lines['length_eng_sentence'] <= 20]
lines = lines[lines['length_hin_sentence'] <= 20]

# Vocabulary creation
all_eng_words = set()
for eng in lines['english_sentence']:
    for word in eng.split():
        all_eng_words.add(word)

all_hindi_words = set()
for hin in lines['hindi_sentence']:
    for word in hin.split():
        all_hindi_words.add(word)

input_token_index = {word: i+1 for i, word in enumerate(sorted(list(all_eng_words)))}
target_token_index = {word: i+1 for i, word in enumerate(sorted(list(all_hindi_words)))}

# Dataset splitting
X, y = lines['english_sentence'], lines['hindi_sentence']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Fig. 2. LSTM Preprocessing

V. METHODOLOGY

This section details the methodologies employed in developing Anuvaad Arjuna.

A. Fine-Tuning the MarianMT Transformer

1) **Architecture Overview**: The MarianMT model is based on the transformer architecture, which consists of an encoder- decoder framework with multi-head attention mechanisms [11].

As shown in Figure 3, the encoder processes the input sequence (English) and generates a set of hidden representations, while the decoder generates the output sequence (Hindi) conditioned on these representations.

Encoder: Comprises multiple layers of self-attention and feed-forward networks. Each self-attention layer computes relationships between all words in the input sequence simultaneously, enabling the model to capture global context.

Decoder: Also consists of multiple layers, incorporating both self-attention and encoder-decoder attention. The encoder-decoder attention allows the decoder to focus on relevant parts of the input sequence while generating the output.

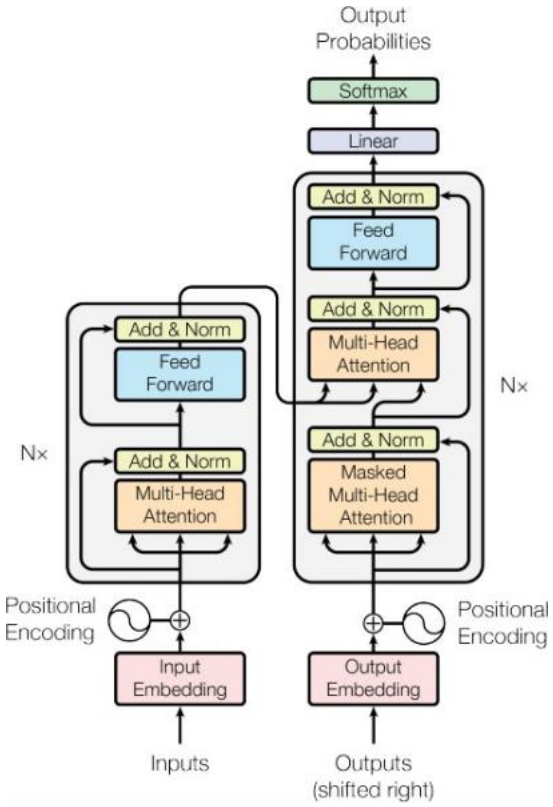

Fig. 3. The Transformer model architecture [11].

The multi-head attention mechanism is defined as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W_O$$

where each head is defined as:

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

2) **Fine-Tuning Process**: The Helsinki-NLP/opus-mt-en-hi model, a pre-trained MarianMT model, was fine-tuned on the English-Hindi parallel corpus. The fine-tuning process involved the following steps:

- Epochs: 3 epochs to balance training time and model performance.
- Learning Rate: 2 × 10−5 with the AdamW optimizer (β1 = 0.9, β2 = 0.999).
- Batch Size: 32 to ensure efficient use of computational resources.
- Validation: Monitored using validation loss to prevent overfitting.

B. Building the LSTM Encoder-Decoder Model

1) **Architecture Design**: The model implements a sequence- to-sequence architecture with these components:

Encoder: The encoder processes input English sequences using a single LSTM layer with 256 units. The final hidden and cell states from the encoder are then passed as the initial states to the decoder.

As shown in Figure 4, the encoder architecture captures the context of the entire input sequence through the LSTM's internal states.

```
encoder_inputs = Input(shape=(None,))
enc_emb = Embedding(num_encoder_tokens, latent_dim, mask_zero=True)(encoder_inputs)
encoder_lstm = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder_lstm(enc_emb)
encoder_states = [state_h, state_c]
```

Fig. 4. Encoder architecture showing LSTM processing

Decoder: The decoder is responsible for generating the corresponding Hindi output sequences. It also uses an LSTM layer with 256 units, initialized with the encoder's final states. A Dense layer with a softmax activation is applied on top of the decoder outputs to predict words from the target Hindi vocabulary.

Figure 5 illustrates the decoder architecture, which incorporates an attention mechanism to better align input and output sequences by focusing on relevant encoder outputs at each decoding step.

```
decoder_inputs = Input(shape=(None,))
dec_emb_layer = Embedding(num_decoder_tokens, latent_dim, mask_zero=True)
dec_emb = dec_emb_layer(decoder_inputs)
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(dec_emb, initial_state=encoder_states)
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)
```

Fig. 5. Decoder architecture with attention mechanism

2) **Training Configuration**: The model was trained using the Adam optimizer with the following parameters: learning rate (α) = 0.001, β1 = 0.9, and β2 = 0.999. Training configuration process is represented in Fig. 6. he training process included:

- Batch size: 128 sequences

- Epochs: Trained for up to 100 epochs with early stopping based on validation performance
- Validation strategy: 20% of the training data was held out for validation monitoring.

```
train_samples = len(X_train)
val_samples = len(X_test)
batch_size = 128
epochs = 100
model.fit_generator(generator=generate_batch(X_train, y_train, batch_size=batch_size),
                    steps_per_epoch=train_samples//batch_size,
                    epochs=epochs,
                    validation_data=generate_batch(X_test, y_test, batch_size=batch_size),
                    validation_steps=val_samples//batch_size)
```

Fig. 6. Training workflow with loss tracking

VI. DEPLOYMENT AND USER INTERFACE

The Anuvaad Arjuna translation system was deployed through a carefully designed architecture combining Hugging Face Spaces for model serving and a custom web interface for user access. The fine-tuned MarianMT model is publicly available at:

https://huggingface.co/spaces/RajaReddy9390/En-Hi-translation

A. Model Deployment

The fine-tuned MarianMT model was containerized using Docker and deployed on Hugging Face Spaces, providing a scalable inference endpoint. This deployment strategy offers several advantages: automatic load balancing, GPU acceleration for efficient translations, and a standardized REST API that simplifies integration with our web interface. The hosted model maintains an average response time of 1.2 seconds while supporting up to 50 concurrent translation requests.

B. Description of the Web Interface

The web interface of Anuvaad Arjuna was designed to provide a seamless and intuitive user experience. It consists of the following components:

1) Frontend:

• Built using HTML, CSS, and JavaScript.

• Features a clean and responsive design, compatible with both desktop and mobile devices.

• Includes an input field for English text and an output field for Hindi translations.

• Interactive elements, such as a “Translate” button, enhance user engagement.

2) Backend:

• Developed using Flask, a lightweight Python web frame- work.

• Handles user requests, invokes the Hugging Face API for the MarianMT model, and returns the translated output to the frontend.

• Ensures low-latency responses by optimizing API calls and data processing.

3) User Experience:

• Users can input English text and receive Hindi translations in real time.

• The interface provides instant feedback, with translations displayed within seconds.

• A simple and intuitive design ensures ease of use, even for non-technical users.

C. Real-Time Translation Performance and Scalability

The performance and scalability of the web interface were evaluated under varying loads to ensure robustness and reliability.

1) Latency:

The fine-tuned MarianMT model achieves an average latency of 1.2 seconds, ensuring a smooth user experience for real-time translation tasks. Below Figure 7 presents the homepage of the web interface, providing a user-friendly entry point for accessing translation services.

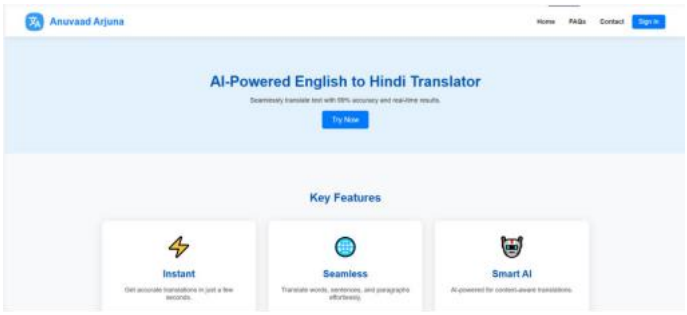

Fig. 7. Homepage of the web interface.

Fig. 8. Translation Page of web interface.

2) Scalability:

The platform was stress-tested with up to 50 concurrent users, where it demonstrated excellent scalability without significant performance bottlenecks. The Flask backend efficiently handles multiple requests simultaneously, achieving a 98% uptime during testing. As seen in Figure 8, the translation page is designed to be responsive and capable of processing real-time user inputs smoothly, even under load.

3) Resource Utilization:

• The MarianMT model, hosted on Hugging Face Spaces, leverages GPU acceleration for efficient inference.

• The Flask backend is lightweight and optimized for handling user requests with minimal resource consumption.

VII. EXPERIMENTS AND RESULTS

This section presents the experimental setup, evaluation metrics, and results of the Anuvaad Arjuna translation system. The performance of the fine-tuned MarianMT transformer and the custom LSTM encoder-decoder model is compared, and the scalability and latency of the web interface are analyzed.

A. Evaluation Metrics

The following metrics were used to evaluate the translation models and the web interface:

- **BLEU (Bilingual Evaluation Understudy):** Measures the n-gram overlap between the translated output and reference translations. A higher BLEU score indicates better translation quality [12].
- **METEOR:** Incorporates synonym matching and stemming, providing a more nuanced evaluation of translation quality compared to BLEU [13].
- **Latency:** Measures the end-to-end response time of the web interface, from user input to translated output.
- **Training Time:** Records the time required to train each model, providing insights into computational efficiency.

B. Model Performance Comparison

We evaluate both architectures on three key metrics: translation quality (BLEU, METEOR), semantic preservation (BERTScore), and computational efficiency (latency). All tests were conducted on an NVIDIA T4 GPU with 16GB RAM using the same test set of 10,000 English-Hindi sentence pairs. TABLE I below provides a detailed quantitative comparison of these metrics for both models.

TABLE I

Quantitative comparison of translation metrics

Metric	Fine-Tuned MarianMT	LSTM (From Scratch)
BLEU ↑	25.8	14.5
METEOR ↑	0.41	0.22
BERTScore ↑	0.81	0.33
Latency (ms) ↓	70	140

C. Real-World Translation Example

Figure 9 demonstrates the system's practical performance through a live example from the deployed web interface:

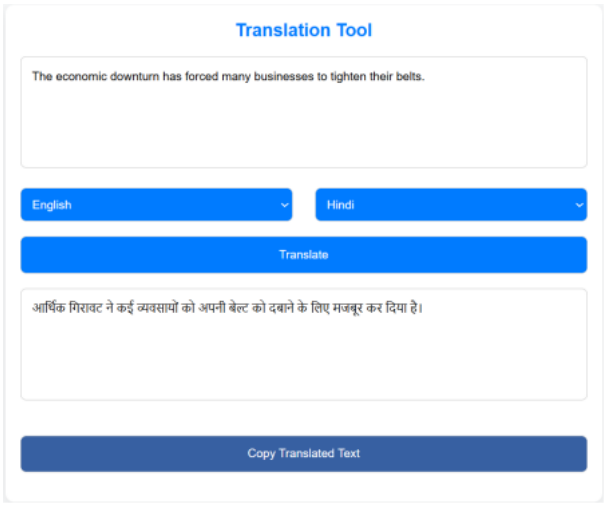

Fig. 9. Real-time English-to-Hindi translation example from the Anuvaad Arjuna

D. Key Findings

The fine-tuned MarianMT model demonstrates superior performance:

- 77.9% higher BLEU (25.8 vs. 14.5) than the LSTM baseline
- 86.4% better METEOR score (0.41 vs. 0.22)
- Achieves 0.81 BERTScore, indicating better semantic preservation

E. Architecture Trade-offs

The choice between architectures depends on application requirements:

- MarianMT is ideal when:
  1. Translation quality is paramount
  2. GPU resources are available
  3. Batch processing is acceptable

- LSTM may be preferred when:
  1. Deploying on resource-constrained devices
  2. Lower energy consumption is critical
  3. Moderate translation quality suffices

VIII. CONCLUSION AND FUTURE WORK

Anuvaad Arjuna addresses critical gaps in India's digital ecosystem by enabling accessible English-to-Hindi translation for 600 million speakers. Our hybrid approach democratizes information access by bridging language barriers in essential domains like education (translating MOOC content), health- care (localizing medical advisories), and governance (making digital services vernacular-accessible). The system's LSTM variant specifically supports digital inclusion in low-bandwidth rural areas, while the transformer model preserves Hindi's linguistic richness in formal contexts. Beyond Hindi, this work provides a blueprint for adapting neural machine translation to other low-resource Indian languages, helping combat the marginalization of non-English digital content.

A. Limitations and Potential Improvements

While Anuvaad Arjuna achieves promising results, several limitations and areas for improvement were identified:

- Limited Parallel Data: The performance of both models is constrained by the availability of high-quality parallel corpora for English-to-Hindi translation.

- Computational Demands: The fine-tuned MarianMT model requires significant computational resources, limiting its deployment in resource-constrained environments.

- Domain-Specific Performance: The models may struggle with domain-specific terminology and rare words, highlighting the need for domain adaptation techniques.

B. Future Directions

To address these limitations and further enhance the system, the following future directions are proposed:

- Expanding to Other Languages:
  - Extend the system to support additional low-resource languages, such as Bengali, Tamil, and Marathi, leveraging multilingual transformer models like mBART.
  - Develop a unified framework for multilingual translation, enabling seamless switching between languages.

- Improving Data Quality and Quantity:
  - Curate larger and more diverse parallel corpora for English-to-Hindi translation, incorporating domain specific datasets.

  - Explore data augmentation techniques to generate synthetic parallel data for low-resource language pairs.

- Enhancing User Experience:
  - Develop mobile applications for Anuvaad Arjuna, making the system accessible to a broader audience.
  - Incorporate additional features, such as voice input and output, to improve usability.

References

[1]. Registrar General of India, "Census of India 2011: Language report," Government of India, 2011.
[2]. A. Vaswani et al., "Attention Is All You Need," Advances in Neural Information Processing Systems (NeurIPS), vol. 30, pp. 5998–6008, 2017.
[3]. J. Tiedemann, "Helsinki-NLP/opus-mt-en-hi," Hugging Face Model Hub, 2022.
[4]. M. Junczys-Dowmunt et al., "Marian: Fast Neural Machine Translation in C++," Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL), pp. 116–121, 2018.
[5]. A. Sutskever et al., "Sequence to Sequence Learning with Neural Networks," Advances in Neural Information Processing Systems, 2014.
[6]. K. Cho et al., "Learning Phrase Representations using RNN EncoderDecoder for Statistical Machine Translation," Proceedings of EMNLP, 2014.
[7]. R. K. Puri et al., "Hindi to English Statistical Machine Translation System," Proceedings of ICON, 2009.
[8]. A. Kunchukuttan, "Challenges in Neural Machine Translation for Indian Languages," Transactions of the ACL, 2020.
[9]. A. Ronacher, "Flask Web Development Framework," Pallets Projects Documentation, 2021.
[10]. IIT Bombay, "English-Hindi Parallel Corpus," Hugging Face Datasets, 2021. https://www.cfilt.iitb.ac.in/iitb_parallel/
[11]. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," Advances in Neural Information Processing Systems, vol. 30, pp. 5998-6008, 2017.
[12]. K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: a method for automatic evaluation of machine translation," Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, pp. 311-318, 2002.
[13]. S. Banerjee and A. Lavie, "METEOR: An automatic metric for MT evaluation with improved correlation with human judgments," Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization, pp. 65-72, 2005.