A Major Project report entitled

On

Unsupervised Language Model Adaptation For Low-Resource Languages

In partial fulfillment of the requirements for the award of

BACHELOR OF TECHNOLOGY

In

Computer Science and Engineering (AI & ML)

Submitted by

D. MANASWINI (21E51A6622)
K. NIKSHIPTA (21E51A6635)
S. PRASANNALAKSHMI (21E51A6658)
V. RISHIK REDDY (22E55A6606)

Under the Esteemed guidance of

Dr. M. RAJESHWAR
Associate Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(AI & ML)

HYDERABAD INSTITUTE OF TECHNOLOGY AND MANAGEMENT

Gowdavelly (Village), Medchal, Hyderabad, Telangana, 501401

(UGC Autonomous, Affiliated to JNTUH, Accredited by NAAC (A+) and NBA)

2024-2025

HYDERABAD INSTITUTE OF TECHNOLOGY AND MANAGEMENT

(UGC Autonomous, Affiliated to JNTUH, Accredited by NAAC (A+) and NBA)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(AI & ML)

CERTIFICATE

This is to certify that the Major Project entitled "**Unsupervised Language Model Adaptation For Low-Resource Languages**" is being submitted by **D. Manaswini** bearing hall ticket number **21E51A6622**, **K. Nikshipta** bearing hall ticket number **21E51A6635**, **S. Prasanna Lakshmi** bearing hall ticket number **21E51A6658**, **M. V. Rishik Reddy** bearing hall ticket number **22E55A6606**, in partial fulfillment of the requirements for the degree **BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING (AI & ML)** by the Jawaharlal Nehru Technological University, Hyderabad, during the academic year 2024-2025. The matter contained in this document has not been submitted to any other University or institute for the award of any degree or diploma.

Under the Guidance of	**Head of the Department**
Dr. M. Rajeshwar	**Dr. P. Padmaja**
Associate Professor, Department of ET	**Professor & HOD, ET**

Internal Examiner	**External Examiner**

HYDERABAD INSTITUTE OF TECHNOLOGY AND MANAGEMENT

(UGC Autonomous, Affiliated to JNTUH, Accredited by NAAC (A+) and NBA)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(AI & ML)

DECLARATION

We **"D. Manaswini, K. Nikshipta, S.Prasanna Lakshmi, V. Rishik Reddy"** students of 'Bachelor of Technology in CSE(AI & ML)', session: 2024 - 2025, Hyderabad Institute of Technology and Management, Gowdavelly, Hyderabad, Telangana State, hereby declare that the work presented in this Major Project entitled 'UNSUPERVISED LANGUAGE MODEL ADAPTATION FOR LOW-RESOURCE LANGUAGES' is the outcome of our own bonafide work and is correct to the best of our knowledge and this work has been undertaken taking care of engineering ethics. It contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

D. MANASWINI (21E51A6622)

K. NIKSHIPTA (21E51A6635)

S. PRASANNA LAKSHMI (21E51A6658)

V. RISHIK REDDY (22E55A6606)

ACKNOWLEDGEMENT

An endeavor of a long period can be successful only with the advice of many well-wishers. We would like to thank our chairman, **SRI. ARUTLA PRASHANTH**, for providing all the facilities to carry out Project Work successfully. We would like to thank our Principal **DR. S. ARVIND**, who has inspired lot through their speeches and providing this opportunity to carry out our Major Project successfully. We are very thankful to our Head of the Department, **DR. PADMAJA** and B-Tech Project Coordinator **DR G. APARNA**. We would like to specially thank my internal supervisor **DR M. RAJESHWAR,** our technical guidance constant encouragement and enormous support provided to us for carrying out our Major Project. We wish to convey our gratitude and express sincere thanks to all **D.C (DEPARTMENTAL COMMITTEE)** and **P.R.C (PROJECT REVIEW COMMITTEE)** members, non-teaching staff for their support and Co-operation rendered for successful submission of our Major Project work.

D. MANASWINI (21E51A6622)

K. NIKSHIPTA (21E51A6635)

S. PRASANNA LAKSHMI (21E51A6658)

V. RISHIK REDDY (22E55A6606)

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

This project presents a bilingual neural machine translation system for English↔Bengali using two Transformer-based approaches. The first is a Transformer architecture implemented from scratch in PyTorch, offering complete architectural control and interpretability. The second approach uses the pretrained T5 For Conditional Generation model, fine-tuned on the same parallel corpus to leverage transfer learning for improved efficiency in low-resource settings. We evaluate both models on standard metrics including BLEU, METEOR, and TER for both translation directions. Results show strong performance for both models, highlighting the effectiveness of attention-based architectures and the complementary strengths of custom and pretrained approaches in low-resource machine translation

Keywords: Transformer Encoder-Decoder Implementation, T5 Model Fine-Tuning for MT, Positional Encoding and Multi-Head Attention, Transformer Architecture

1. INTRODUCTION

Neural Machine Translation (NMT) represents a significant advancement in machine translation, originally proposed by Kalchbrenner and Blunsom (2013), Sutskever et al. (2014), and Cho et al. (2014b). Unlike traditional phrase-based systems (e.g., Koehn et al., 2003), which rely on multiple individually tuned components, NMT employs a single, end-to-end neural network to encode a source sentence and generate its translation. This approach improves fluency and contextual understanding, especially for complex linguistic structures.

Among recent innovations, the **Transformer architecture** introduced by Vaswani et al. (2017) has become the dominant paradigm in NMT. Its self-attention mechanism and parallelizable structure have significantly enhanced translation accuracy and efficiency, outperforming previous RNN-based models. The Transformer is now the backbone of most state-of-the-art translation systems.

However, for **low-resource languages** like Bengali—despite having over 250 million speakers—progress in MT has been limited. Bengali lacks large-scale, high-quality parallel corpora and pretrained resources compared to high-resource languages such as English, French, or German. Its complex morphology and syntax further complicate translation tasks.

To address these challenges, our project explores two complementary approaches. First, we develop a **Transformer model from scratch** in PyTorch, enabling us to train and control all architectural aspects tailored to Bengali-English translation. Second, we fine-tune the **pretrained T5ForConditionalGeneration model**, which is based on the Transformer architecture but benefits from extensive multilingual pretraining on large corpora. By comparing both models, we aim to evaluate how foundational training versus transfer learning affects translation performance for underrepresented languages like Bengali.

These efforts are not only important for improving technical performance but also contribute to reducing the linguistic digital divide and expanding access to global communication.

In this project, we focus on building a bidirectional Neural Machine Translation (NMT) system between Bengali and English using two distinct approaches:

1. **A Transformer model implemented from scratch in PyTorch**, and
2. **A pretrained T5ForConditionalGeneration model**, fine-tuned on the same bilingual dataset.

The project is structured into two main phases, with both models being trained and evaluated for each translation direction:

1. Bengali to English Translation

In the first phase, the system is trained to translate Bengali sentences into English. For the **Transformer-from-scratch** model, a custom rule-based tokenizer is used for Bengali, while SpaCy is used for English. Vocabularies are constructed manually, and the Transformer model is trained using parallel Bengali-English sentence pairs. The model leverages self-attention mechanisms to capture long-range dependencies efficiently.

In parallel, the **T5 model** is fine-tuned on the same Bengali→English dataset using Hugging Face's Trainer API. Tokenization is handled automatically by the T5 tokenizer, and the model benefits from prior multilingual pretraining, which accelerates convergence.

2. English to Bengali Translation

In the second phase, the translation direction is reversed. For the custom Transformer, source and target roles are swapped, and new vocabularies are constructed accordingly. The model is retrained with English inputs and Bengali targets.

Similarly, the T5 model is fine-tuned for the English→Bengali task by adjusting the task prefix and training on reversed sentence pairs. This phase demonstrates the bidirectional adaptability of both architectures, even when handling low-resource languages.

By comparing both models across these two phases, we assess the trade-offs between developing a fully custom architecture and leveraging a pretrained model. This bidirectional setup allows us to demonstrate how Transformer-based models—both handcrafted and pretrained—can be effectively applied to high-resource (English) and low-resource (Bengali) language translation tasks. The project also provides insights into training dynamics, tokenization strategies, and model evaluation using BLEU, METEOR, and TER scores.

Figure 1 : Transformer model translates text from one language to another using deep learning.

2. LITERATURE SURVEY

A **literature survey** (or literature review) is a comprehensive overview of existing research, studies, theories, and findings related to a specific topic or research problem.
A literature survey explores existing research and advancements in machine translation (MT) for English↔Bengali pairs, as well as broader developments in neural machine translation (NMT). Prior work has demonstrated the effectiveness of deep learning models such as RNNs and LSTMs, and more recently, Transformer architectures for sequence modeling and translation. Techniques such as back-translation and semi-supervised learning have also emerged as valuable strategies for improving performance in low-resource settings.
Multilingual pretrained models like T5 and mBART have been shown to generalize well to underrepresented languages when fine-tuned properly. Simultaneously, implementing Transformer models from scratch remains an important research exercise for understanding architectural behavior, especially in language-specific adaptation.
In this project, we build on these developments by implementing two complementary approaches: a Transformer model built from scratch using PyTorch, and a pretrained T5 model fine-tuned on our Bengali–English dataset. This dual-model methodology enables us to position our work within both foundational and transfer learning paradigms, allowing us to explore how each performs in the context of low-resource translation.

Literature survey:

1. "Neural Machine Translation by Jointly Learning to Align and Translate"

- **Authors:** Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio
- **Paper/Year:** Neural Machine Translation by Jointly Learning to Align and Translate (2015)
- **Description:** This paper introduces the **attention mechanism** in neural machine translation (NMT), significantly improving the ability to align words in a source sentence with those in the target language, which is crucial for languages like Bengali that have different sentence structures from English.
- **Model/Algorithm: Attention-based Sequence-to-Sequence Model (Seq2Seq)**.

2. "Sequence to Sequence Learning with Neural Networks"

- **Authors:** Ilya Sutskever, Oriol Vinyals, Quoc V. Le
- **Paper/Year:** Sequence to Sequence Learning with Neural Networks (2014)
- **Description:** This paper describes the **Seq2Seq model** for sequence learning tasks, including machine translation. It was one of the foundational works that used **Recurrent Neural Networks (RNNs)** for both the encoder and decoder.
- **Model/Algorithm: Seq2Seq with LSTM/GRU**.

3. "Attention is All You Need"

- **Authors:** Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin
- **Paper/Year:** Attention is All You Need (2017)
- **Description:** This paper introduces the **Transformer architecture**, which replaces RNNs and LSTMs with **self-attention mechanisms**, significantly improving translation quality and computational efficiency. It has become the foundation for most state-of-the-art NMT models today, including for low-resource languages like Bengali.
- **Model/Algorithm: Transformer (Self-Attention Model)**.

4. "Bengali-English Neural Machine Translation with Shared Encoder and Decoder"

- **Authors:** R. G. Krishna, Rajarshi Das, Sudipta Kar, P. S. G. Chitra, P. G. Iyer
- **Paper/Year:** Bengali-English Neural Machine Translation with Shared Encoder and Decoder (2016)
- **Description:** This paper applies **neural machine translation (NMT)** to Bengali-English translation, focusing on encoder-decoder architectures. It also addresses the difficulties faced in translating between Bengali and English, such as sentence structure differences.
- **Model/Algorithm: Neural Machine Translation (NMT) with LSTM-based Seq2Seq**.

5. "Unsupervised Machine Translation Using Monolingual Corpora Only"

- **Authors:** Guillaume Lample, Ludovic Denoyer, Marc Dymetman
- **Paper/Year:** Unsupervised Machine Translation Using Monolingual Corpora Only (2018)
- **Description:** This paper discusses how **unsupervised machine translation (UMT)** works without the need for parallel corpora by leveraging monolingual corpora from both source and target languages, a technique which is crucial for low-resource languages like Bengali.
- **Model/Algorithm: Unsupervised NMT with back-translation**.

6. "Massively Multilingual Neural Machine Translation in the Wild: Findings and Challenges"

- **Authors:** Mikel Artetxe, Gorka Labaka, Eneko Agirre
- **Paper/Year:** Massively Multilingual Neural Machine Translation in the Wild: Findings and Challenges (2020)
- **Description:** This paper examines **multilingual machine translation** (MT) systems, including methods like **mBART** and **mT5**. These models, pre-trained on large multilingual corpora, are beneficial for low-resource languages such as Bengali when combined with high-resource languages like English.
- **Model/Algorithm: Multilingual Pretraining Models (mBART, mT5)**.

7. "Improving Neural Machine Translation Models with Monolingual Data"

- **Authors:** Rico Sennrich, Barry Haddow, Alexandra Birch
- **Paper/Year:** Improving Neural Machine Translation Models with Monolingual Data (2016)
- **Description:** This paper introduces the idea of **back-translation**, a semi-supervised method

that leverages monolingual corpora in the target language to improve the performance of NMT systems, particularly in low-resource settings.
- **Model/Algorithm: Back-Translation** (Monolingual data augmentation for NMT).

8. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding"
- **Authors:** Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova
- **Paper/Year:** BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (2018)
- **Description:** BERT is a pre-trained **transformer-based model** that has set new benchmarks for language understanding. While it is not directly a translation model, it has been adapted for **cross-lingual transfer** tasks, making it useful for languages like Bengali and English in MT tasks.
- **Model/Algorithm: BERT (Bidirectional Encoder Representations from Transformers)**.

9. "Multilingual Neural Machine Translation with Soft Decoding"
- **Authors:** Marta R. Costa-jussà, Josep A. R. L. Pons
- **Paper/Year:** Multilingual Neural Machine Translation with Soft Decoding (2017)
- **Description:** This paper introduces **multilingual NMT** with a **soft decoding approach**, allowing a single NMT model to handle multiple language pairs simultaneously. This is particularly beneficial when extending MT models to low-resource languages like Bengali.
- **Model/Algorithm: Multilingual NMT (Shared Encoder-Decoder)**.

10. "Pretrained Transformers as Universal Computation Engines"
- **Authors:** Sebastian Ruder, Matthew E. Peters, Mark Neumann, Alexander M. Rush
- **Paper/Year:** Pretrained Transformers as Universal Computation Engines (2020)
- **Description:** This research investigates the potential of **pre-trained transformers** for transfer learning in MT and other NLP tasks. It emphasizes how models like **mBART** and **MarianMT** can be adapted to a variety of languages, including Bengali, improving performance even with limited resources.
- **Model/Algorithm: Pretrained Transformer Models** (such as **mBART**, **MarianMT**).

3. PURPOSE AND SCOPE

3.1 Overview

Machine Translation (MT) has become an indispensable tool for bridging linguistic divides and enabling global communication. However, for low-resource languages like Bengali, the lack of large-scale, high-quality parallel corpora presents a significant challenge. Despite being one of the most spoken languages globally, Bengali remains underrepresented in the field of MT when compared to high-resource languages such as English, French, or Spanish.

The objective of this project is to design and implement a **bidirectional neural machine translation (NMT) system** capable of translating text between English and Bengali in both directions—English-to-Bengali (En→Bn) and Bengali-to-English (Bn→En). To accomplish this, the system leverages **two distinct architectures**:

- A **Transformer model implemented from scratch** in PyTorch, enabling full control over model design and training dynamics.
- A **pretrained T5ForConditionalGeneration model**, fine-tuned on the same dataset to utilize transfer learning for improved generalization in low-resource settings.

Given the limited availability of parallel corpora, the project also incorporates **semi-supervised learning techniques** such as **back-translation**, using monolingual English and Bengali data to augment the training set with synthetic sentence pairs.

3.2 Challenges Faced & Solutions:

Challenges:

1. Data Scarcity for Bengali

- EN→BN and BN→EN: Both translation directions suffer from a lack of high-quality parallel corpora. While English is a high-resource language, Bengali lacks sufficient annotated bilingual data, which impacts both models.
- The Transformer-from-scratch model, in particular, is more sensitive to data scarcity as it starts from randomly initialized parameters.
- The T5 model, though pretrained, still requires fine-tuning on quality data. To mitigate this, techniques like back-translation and data augmentation were applied in both directions to supplement the training set.

2.Linguistic Differences Between English and Bengali

- EN→BN: English follows a Subject-Verb-Object (SVO) order, while Bengali follows Subject-Object-Verb (SOV), requiring the model to learn reordering patterns.
- BN→EN: Bengali's rich morphology and flexible structure introduce challenges like verb agreement, postpositions, and compound word handling.
- Both the custom Transformer and T5 must learn to adapt to these structural variations, though the pretrained T5 model may generalize better due to prior multilingual training.

3.Model Generalization

- EN→BN and BN→EN: Overfitting is a concern, particularly for the Transformer-from-

scratch model when trained on small datasets.

- Fine-tuning the T5 model on carefully curated and diverse sentence pairs improves generalization and reduces overfitting risks in both directions.

4.Handling Ambiguity and Idiomatic Expressions

- EN→BN and BN→EN: Idioms, metaphors, and context-dependent phrases are difficult to translate directly.
- Both models must learn contextual representations to resolve ambiguities. The T5 model benefits from prior exposure to similar constructs in other languages, while the scratch Transformer relies entirely on the task-specific training data.

Future Work:

1. Exploration of Multilingual Models
   - Integrating **multilingual pretrained models** such as **mBART** or **MarianMT** could enhance performance in both EN→BN and BN→EN directions.
   - These models could complement the **Transformer-from-scratch** as baselines and offer further pretraining benefits beyond what T5 provides.
2. Advanced Back-Translation Techniques
   - Future iterations may use **context-aware or quality-controlled back-translation** to improve pseudo-parallel data quality.
   - This can benefit both models: the **scratch model** gains robustness from augmented training, while **T5** fine-tuning becomes more accurate with cleaner synthetic data.

3. Domain-Specific Translation
   - Creating specialized models for domains such as medical, legal, or technical translation could significantly improve translation precision.
   - Both models can be adapted to domain-specific fine-tuning, although the **T5 model's architecture is more flexible** for multitask setups.
4. Contextual and Multimodal Learning
   - Incorporating **document-level context** or multimodal inputs (e.g., images, audio) could enhance translation quality for nuanced or conversational content.

Future extensions could involve modifying the **Transformer-from-scratch architecture** to include context windows, or fine-tuning **T5 on dialog corpora or multimodal datasets** to support richer understanding

4. METHODOLOGY

4.1 WHAT IS METHODOLOGY?

The methodology is the structured plan that outlines how the research or project is conducted from start to finish. In the context of this project, the methodology serves as a comprehensive framework that guides every phase — from the initial data collection, preprocessing, model development, and training, to evaluation and final deployment. This project uniquely involves two translation models: a Transformer model built from scratch using PyTorch, and a pretrained T5 model fine-tuned using Hugging Face's Transformers library. The aim is to study and compare these approaches within a bidirectional English↔Bengali machine translation system. The methodology ensures that each model is trained, tested, and evaluated under controlled conditions to draw valid comparisons and extract meaningful insights.

4.2 METHODOLOGY TO BE USED:

The methodology for building an English↔Bengali (EN↔BN) machine translation system can be broken down into several phases: data collection, data preprocessing, model development, training, evaluation, and deployment.

1: Data Collection and Preparation:

The first and most crucial phase involves gathering quality data. A translation model is only as good as the data it's trained on. This phase involves collecting both parallel (aligned sentence pairs) and monolingual datasets for English and Bengali. For parallel corpora, we use established datasets such as IIT Bombay, OpenSubtitles, and the Bengali-English Bible. These datasets provide the aligned sentence pairs required for supervised learning. For monolingual corpora, texts are sourced from Bengali and English Wikipedia, news websites, and digital books.

To address the data scarcity issue often faced in Bengali translation, we apply back-translation. In this technique, monolingual data is translated into the other language using a pretrained model (e.g., mBART or MarianMT) to create synthetic parallel data. This augmented data strengthens the training process for both models.

Once data is collected, it is split into training (80%), validation (10%), and testing (10%) sets to facilitate consistent evaluation across model versions.

2: Data Preprocessing

This phase focuses on transforming the raw textual data into a clean, consistent, and model-ready format. Preprocessing is essential to ensure that the inputs are standardized and that both the models — Transformer-from-scratch and the pretrained T5 — receive data in a suitable format for learning and inference.

2.1 Tokenization

Tokenization is the process of splitting sentences into individual units (words or subwords). For English, we use SpaCy or NLTK to tokenize sentences, which helps handle punctuation,

contractions, and language-specific grammar structures. For Bengali, which lacks standardized tokenization libraries, we employ custom rule-based tokenizers or the Indic NLP library to correctly split text based on script characteristics and spacing norms.

2.2 Subword Tokenization

To address the challenges of rare and morphologically complex words — especially in Bengali — we use subword tokenization methods such as SentencePiece or Byte Pair Encoding (BPE). These techniques segment words into smaller meaningful units, enabling the model to handle unseen or compound words. A common vocabulary size of around 32,000 tokens is used for both languages to maintain consistency across models.

- The **Transformer-from-scratch** model is trained with a custom tokenizer and subword vocabulary.
- The **T5 model** uses its built-in SentencePiece tokenizer during both fine-tuning and inference.

2.3 Text Normalization

Text normalization ensures uniformity across the dataset. For English text, this includes converting to lowercase, removing extra spaces and unnecessary punctuation. For Bengali text, normalization includes handling diacritics, removing noise, and converting compound characters to canonical forms. These steps help improve token alignment and translation quality.

2.4 Sentence Alignment

Aligned sentence pairs are critical in supervised translation tasks. Each English sentence must correspond exactly with its Bengali translation. We use sentence length filtering, token ratio thresholds, and manual spot-checks to maintain alignment quality.

Overall, preprocessing ensures that both models receive consistent, structured input, thereby maximizing their ability to learn meaningful patterns from the training data.

3. Model Development

3.1 Transformer Model (from Scratch)

In this phase, we develop two distinct neural machine translation models for the English↔Bengali translation task: (1) a Transformer model built entirely from scratch using PyTorch, and (2) a pretrained T5ForConditionalGeneration model fine-tuned on our dataset. Each model represents a different paradigm in deep learning — one from a fundamental design and training perspective, and the other from a transfer learning and fine-tuning standpoint.

3.1 Transformer Model (from Scratch)

The Transformer-from-scratch model adheres to the original architecture proposed by Vaswani et al. It features an encoder-decoder framework, where both the encoder and decoder are composed of multiple identical layers.

- Encoder: Each encoder layer includes multi-head self-attention, followed by a position-wise feed-forward network. Positional encodings are added to the token embeddings to preserve the order of words.
- Decoder: The decoder layers use masked multi-head self-attention to prevent information leakage, followed by encoder-decoder attention and a feed-forward network. Residual

connections and layer normalization are included in each sublayer.

- Vocabulary and Tokenization: This model uses a custom vocabulary generated from training data using BPE or SentencePiece. It includes special tokens like , , , and .
- Training Objective: The decoder is trained to predict the next token in the sequence, given all previously generated tokens and the encoded input.

This approach provides complete flexibility to adjust model dimensions (embedding size, number of layers, attention heads) and manually tune optimization parameters. It also helps us understand the inner workings of attention, masking, and positional encoding mechanisms.

3.2 T5 For Conditional Generation (Pretrained)

T5 (Text-to-Text Transfer Transformer) is a pretrained encoder-decoder model designed for a wide variety of NLP tasks. We use Hugging Face's implementation of T5ForConditionalGeneration and fine-tune it on our Bengali-English dataset.

- Architecture: T5 has a similar encoder-decoder structure but includes standardized T5Blocks consisting of self-attention, cross-attention, feed-forward layers, and layer normalization. Both encoder and decoder stacks share a common embedding layer.
- Tokenization: T5 uses the SentencePiece tokenizer, which allows it to process multilingual text more effectively. All inputs and outputs are formatted as text strings, and subword tokenization is applied.
- Training Strategy: We use the Hugging Face Trainer API to fine-tune T5 on our parallel corpora. The task is cast as a text-to-text problem, where input strings are prepended with task-specific prefixes (e.g., "translate Bengali to English:").
- Advantages: Due to extensive pretraining on large multilingual corpora, T5 provides faster convergence, better generalization, and robustness to limited data, especially for low-resource languages like Bengali.

By developing and evaluating both models under the same conditions and datasets, we gain valuable insights into the trade-offs between custom model training and transfer learning approaches for machine translation.

4. Training the Model

The training phase involves the supervised optimization of both translation models — the Transformer model built from scratch and the pretrained T5ForConditionalGeneration model — using the previously prepared parallel Bengali-English corpora.

4.1 Training the Transformer-from-Scratch Model

The custom Transformer model is implemented and trained in PyTorch. The model parameters, including embedding size, number of encoder and decoder layers, and attention heads, are defined manually. During training, the model processes input sequences in the source language and generates output sequences in the target language.

- Loss Function: The model uses the CrossEntropyLoss function. This loss measures the difference between the predicted token probabilities and the ground truth tokens, ignoring padded tokens using the ignore_index parameter.
- Optimizer: The Adam optimizer is employed to update weights, as it offers efficient gradient descent with adaptive learning rates.
- Learning Rate Scheduling: A custom learning rate scheduler with warm-up steps is used to

prevent early overfitting and promote stable convergence.

- Teacher Forcing: This technique is applied to enhance learning efficiency. At each training step, the ground truth target token is fed into the decoder instead of the model's previous prediction, which reduces error propagation and speeds up convergence.

Training is performed over multiple epochs (typically 20–30), and at each epoch, the model's performance is validated using the BLEU score calculated on a held-out validation set.

4.2 Fine-Tuning the T5 Model

The T5ForConditionalGeneration model is fine-tuned using Hugging Face's Trainer API, which simplifies the training loop and incorporates several performance optimization features:

- Task Formatting: All translation tasks are framed as text-to-text problems. For example, an English-to-Bengali task uses input like "translate English to Bengali: " and expects the Bengali output.
- Loss Function: The pretrained model is trained using a similar CrossEntropyLoss function, which is inherently supported by the Trainer API.
- Gradient Clipping: Applied to prevent exploding gradients in deep networks.
- Label Smoothing: Reduces overconfidence in predictions by assigning small probabilities to incorrect labels.
- Early Stopping: Stops training if no improvement is observed in validation metrics, helping avoid overfitting.

The fine-tuning process typically converges faster than training from scratch due to the pretrained nature of the model. BLEU scores are monitored after each epoch to assess translation quality improvements.

By training both models side-by-side, we obtain comparative insights into model behavior: the scratch model allows for complete architectural control and experimentation, while the pretrained T5 model demonstrates the effectiveness of transfer learning in achieving high translation accuracy with fewer training epochs.

5. Evaluation

Objective: Evaluate the performance of the trained model.

5.1. Evaluation Metrics:

- **BLEU Score**: The **BLEU score** will measure the quality of the generated translations by comparing them with human-annotated references. A higher BLEU score (typically above **0.3** for a good model) indicates better translation quality.
- **TER (Translation Edit Rate)**: This measures how many edits are needed to convert the system output into the reference translation.
- **Accuracy**: Check the percentage of exact matches between the predicted translation and the reference translation.

5.2. Test Set Evaluation:

- Evaluate the final model on the test set, which it hasn't seen during training, to get an

unbiased performance evaluation.

6. Fine-Tuning and Iteration

Following the initial evaluation of both models, a crucial phase involves refining their performance through various enhancement techniques. Fine-tuning and iterative improvements are essential to ensure that the models not only generalize better but also perform optimally in real-world and domain-specific scenarios.

6.1 Domain-Specific Fine-Tuning

To increase the practical applicability of the translation models, especially in specialized fields such as legal, medical, or technical domains, domain-specific datasets are used for further fine-tuning. For the Transformer-from-scratch model, this involves continued training with targeted vocabulary and terminology. For the T5 model, domain fine-tuning leverages its pretraining flexibility, allowing it to adapt quickly with fewer examples.

6.2 Back-Translation for Data Augmentation

Back-translation is reapplied using fresh monolingual data collected after initial training. Monolingual Bengali or English sentences are translated using the current best-performing model, and the translated outputs are paired with the original inputs to create new pseudo-parallel sentence pairs. These are added to the training set to reinforce learning, particularly for low-frequency or complex sentence structures.

6.3 Hyperparameter Tuning

To further optimize performance, hyperparameter tuning is conducted. This includes experimenting with different learning rates, attention head sizes, number of hidden layers, and dropout rates. Grid search or random search strategies are applied systematically to find the best configuration. The impact of these changes is measured using validation BLEU scores, training loss curves, and convergence behavior.

This iterative phase ensures that both models evolve to become more robust, accurate, and specialized in handling diverse linguistic challenges present in English↔Bengali translation tasks.

7. Deployment

Objective: Deploy the model for real-world use.

7.1. Web-based Translation Service:

- Deploy the model as a REST API using **Flask** or **FastAPI** for real-time translation between **EN↔BN**.

7.2. Mobile App:

Build a **mobile application** using **React Native** or **Flutter** that can send text to the API and receive the translated output in real-time

4.3 TEST METHODOLOGY:

The **test methodology** for both **English-to-Bengali (EN→BN)** and **Bengali-to-English (BN→EN)** translations involves evaluating the trained model's performance on the held-out **test set**, analysing the results with various metrics, and ensuring robustness and generalization

1.Test Data Preparation

Objective: Prepare and use the test data to evaluate the model's performance.

1.1. Test Set Selection:

- The **test set** should be **independent** and **unseen** during training and validation. Typically, it is **10%** of the total parallel data.
- Ensure that the test set contains a diverse set of **sentences** representing various domains and sentence structures (simple, complex, technical, and everyday language).

1.2. Preprocessing:

- Perform the same preprocessing steps on the test set as done during the training phase:
  - **Tokenization** (using the same tokenizers as used in the training data).
  - **Subword tokenization** (using the same SentencePiece or BPE model).
  - **Text normalization** (if any).

2. Evaluation Metrics

Objective: Evaluate the translation quality using standard metrics.

2.1. BLEU Score:

- **BLEU (Bilingual Evaluation Understudy) score** is the most widely used metric to evaluate the performance of machine translation systems. It measures how many **n-grams** (e.g., unigrams, bigrams) overlap between the machine-generated translation and the reference translation.
  - A higher **BLEU score** indicates better translation quality.
  - BLEU score is calculated by comparing the output from the model (generated translation) with **human reference translations**.
- **Implementation**:
  - Use **nltk.translate.bleu_score** or **sacreBLEU** library in Python to compute BLEU scores.
  - Common values for **n** in BLEU (1-gram to 4-gram) can be used.

2.2. TER (Translation Edit Rate):

- **TER** measures the number of edits required to transform the system-generated output into the reference translation (lower is better).
- **Formula**:

$$\text{TER} = \frac{\text{Number of edits}}{\text{Total number of words in reference}}$$

- **Implementation**:
  - Use **TER tool** from **Google's official translation evaluation toolkit** to calculate it.

3.Test Phase Evaluation

Objective: Run the model on the test set and compute the evaluation .

3.1. Model Inference:

- **EN→BN Inference**:

    - Take an **English sentence** from the test set, pass it through the trained **English-to-Bengali model**, and generate the corresponding Bengali translation.
- **BN→EN Inference**:
    - Take a **Bengali sentence** from the test set, pass it through the trained **Bengali-to-English model**, and generate the corresponding English translation.

3.2. Generation Comparison:

After generating translations, compare them with the **reference translations** (ground truth) from the test set using the metrics mentioned

4.Error Analysis

Objective: Identify and analyze common translation errors to improve the system.

4.1. Common Error Types:

- **Omissions**: When a part of the sentence is omitted in the translation.
- **Additions**: When extra information not present in the source sentence is added.
- **Word Choice Errors**: Incorrect translation of words or phrases.
- **Grammatical Errors**: Incorrect sentence structures due to the model's inability to capture complex grammatical rules.

4.2. Qualitative Analysis:

- Manually review **a subset of translations** to identify the types of errors the model is making.
- This helps understand if the model struggles with certain grammatical structures, specific vocabulary, or domain-specific terminology.

4.3. Quantitative Analysis:

- Analyze **sentence-level BLEU scores** to determine if certain types of sentences (e.g., complex or technical) have lower translation quality.
- Compare the **distribution of BLEU scores** across different test data segments.

5.Ablation Study

Objective: Conduct ablation studies to understand which parts of the model contribute the most to performance.

5.1. Impact of Back-Translation:

- Compare the performance of the model trained with **back-translated data** versus one trained without it. This will help assess the value of **pseudo-parallel data** in improving translation quality.

5.2. Impact of Fine-Tuning:

- Evaluate the model performance before and after **fine-tuning** on domain-specific data or additional training corpora (e.g., legal or medical texts).

5.3. Impact of Pretrained Models:

- Compare the results of **fine-tuning a pretrained model (mBART, MarianMT)** against training the model from scratch.

6.Robustness Testing

Objective: Ensure the model can handle diverse real-world data and edge cases.

6.1. Testing on Noisy Data:

- Evaluate the model's robustness to **noisy inputs** (e.g., text with typos, informal language, or slang) to assess its ability to handle real-world scenarios.

6.2. Cross-Domain Testing:

- Test the model on **out-of-domain sentences** (e.g., medical, legal, or scientific texts) to evaluate its generalization capability.

7. Performance Optimization

Objective: Ensure the model runs efficiently in real-world applications.

7.1. Latency and Speed:

- Measure the **inference time** for the model to translate a sentence. This is critical for real-time applications like chatbots or mobile apps.

7.2. Memory Usage:

- Check the **memory consumption** during inference to ensure the model is lightweight enough for deployment, especially on mobile devices or servers with limited resources.

8. Report Generation

Objective: Summarize the results and insights from the evaluation phase.

8.1. Performance Summary:

- Provide a comprehensive summary of all evaluation metrics (e.g., BLEU, TER, ROUGE, F1-Score) for both **EN→BN** and **BN→EN** translations.

8.2. Error Analysis Insights:

- Provide detailed insights into the most common errors and suggestions for model improvement (e.g., focusing on certain linguistic features or acquiring more domain-specific data).

8.3. Future Improvements:

- Suggest potential avenues for improving model performance, such as incorporating more diverse datasets, enhancing the model's handling of syntactic differences, or using ensemble techniques.

5. REQUIREMENTS

5.1 SOFTWARE REQUIREMENTS:

To develop and run the machine translation system, you will need the following software and tools:
Programming Language:

- Python (Version 3.7 or later): Python is the primary language used for this project due to its extensive support for machine learning, NLP, and deep learning libraries.

Libraries and Frameworks:

1. PyTorch (for deep learning)
    - To build and train neural networks, PyTorch is highly recommended for this project, as it is flexible and widely used in NLP tasks.
2. Transformers (Hugging Face)
    - A library by Hugging Face that provides pre-trained models (e.g., MarianMT, mBART) and tools for natural language processing (NLP) tasks.
3. Datasets (Hugging Face)
    - The datasets library allows you to easily access pre-existing datasets, such as parallel corpora for machine translation tasks.
4. NLTK (Natural Language Toolkit)
    - Provides tools for tokenization, evaluation metrics like BLEU, ROUGE, and METEOR, and other preprocessing steps.
5. SacreBLEU
    - A library for computing BLEU scores, a commonly used evaluation metric in machine translation.
6. Rouge-Score
    - A package for evaluating ROUGE scores, another common metric for summarization and translation tasks.
7. JIWER (for Word Error Rate calculation)
    - A library for calculating WER (Word Error Rate), commonly used for speech-to-text and translation tasks.
8. scikit-learn
    - A library for calculating traditional machine learning metrics like accuracy, precision, recall, and F1-score, which can be helpful for translation evaluation.
9. SentencePiece (Optional for Subword Tokenization)
    - SentencePiece is often used for subword tokenization in neural machine translation tasks to handle out-of-vocabulary words.
10. CUDA (Optional for GPU Support)
    - If using a GPU for training, you will need CUDA to accelerate the training process.

5.2 HARDWARE REQUIREMENTS:

The hardware requirements will depend on the scale of the project, such as whether you are training models from scratch or fine-tuning pre-trained models.
CPU (Central Processing Unit):

- Minimum: Any modern multi-core processor (Intel i5/Ryzen 5 or better).
- Recommended: High-performance multi-core processor (Intel i7/Ryzen 7 or better).

GPU (Graphics Processing Unit) (Optional but Recommended):

- Minimum: NVIDIA GTX 1050/1060 with CUDA support (for moderate workloads and smaller datasets).
- Recommended: NVIDIA RTX 2080, 3060, or better for efficient model training on large datasets.
    - Tensor Cores on RTX cards speed up training with mixed precision, which can help with larger models and datasets.

RAM (Memory):

- Minimum: 8 GB RAM for basic tasks and smaller models.
- Recommended: 16 GB or more for handling larger models and datasets without performance issues.

5.3 INSTALLATION:

1. Python Installation

You need Python 3.7 or higher to run this project. Here's how to install it:

- **Windows:**
    1. Download Python from the official site: Python Downloads
    2. Run the installer and ensure to check the box **"Add Python to PATH"** before clicking.

2. Install Required Libraries

Once the virtual environment is activated, install the required libraries.

This will install the core dependencies needed for your machine translation project:

- **torch**: For deep learning and model training.
- **transformers**: For pre-trained models from Hugging Face (like MarianMT, mBART).
- **datasets**: To load and manage datasets like parallel corpora.
- **nltk**: For tokenization and evaluation metrics like BLEU, ROUGE.
- **sacrebleu**: For BLEU score evaluation.
- **rouge_score**: For ROUGE score evaluation.
- **jiwer**: For Word Error Rate (WER) calculation.
- **scikit-learn**: For machine learning evaluation metrics like accuracy and F1-score.
- **sentencepiece**: For subword tokenization.

6. MODEL AND ARCHITECTURE

1. Model Overview

In this project, we focus on two primary strategies for building a neural machine translation (NMT) system between English and Bengali:

1. **Transformer-Based Model Built from Scratch**
2. **Pretrained Transformer Model (T5)**

Both strategies are rooted in the Transformer architecture introduced by Vaswani et al. (2017), which has become the foundation for most modern NMT systems due to its ability to model long-range dependencies using self-attention

◇ **1. Transformer Model from Scratch (Custom PyTorch Implementation)**

This model follows the classic **encoder-decoder Transformer architecture**, built entirely using PyTorch. It is trained from scratch using tokenized parallel corpora for English↔Bengali translation. The encoder processes input sequences (e.g., English), and the decoder generates corresponding outputs (e.g., Bengali) by attending to the encoder's hidden states. This implementation gives full control over each component (embedding size, attention heads, layers) and is highly instructive for understanding the internals of deep learning-based translation.

◇ **2. Pretrained T5 Transformer (Text-to-Text Transfer Transformer)**

T5 is a **pretrained encoder-decoder Transformer** developed by Google and available via Hugging Face's Transformers library. It is fine-tuned on the same parallel corpus as the scratch model. T5 frames all NLP tasks, including translation, as a text-to-text problem. For example:

- Input: "translate English to Bengali: This is a test."
    - Output: "এটি একটি পরীক্ষা।

2. Model Architecture

2.1 Transformer Model (from Scratch)

The Transformer model implemented from scratch in this project adheres closely to the architecture introduced by Vaswani et al. (2017) in the paper "Attention is All You Need." It is an encoder-decoder model built entirely using PyTorch, making it highly customizable and educational. The core idea behind the Transformer is to replace recurrent layers with self-attention mechanisms that can model dependencies between words regardless of their position in the sequence.

The encoder consists of a stack of identical layers, each comprising a multi-head self-attention mechanism and a feed-forward neural network. The self-attention mechanism allows each word in the input sentence to attend to all other words, capturing context effectively. Positional encoding is added to the input embeddings to provide information about the position of each token, as the Transformer does not inherently account for word order.

The decoder mirrors the encoder in structure but includes additional layers for masked self-attention (to prevent future token access during training) and encoder-decoder attention. This allows the decoder to attend to the encoder's outputs and generate translated text token by token.

The Transformer-from-scratch model adheres to the original architecture proposed by Vaswani et al. It features an encoder-decoder framework, where both the encoder and decoder are composed of multiple identical layers.

- **Encoder:** Each encoder layer includes multi-head self-attention, followed by a position-wise feed-forward network. Positional encodings are added to the token embeddings to preserve the order of words.
- **Decoder:** The decoder layers use masked multi-head self-attention to prevent information leakage, followed by encoder-decoder attention and a feed-forward network. Residual connections and layer normalization are included in each sublayer.
- **Vocabulary and Tokenization:** This model uses a custom vocabulary generated from training data using BPE or SentencePiece. It includes special tokens like , , , and .
- **Training Objective:** The decoder is trained to predict the next token in the sequence, given all previously generated tokens and the encoded input.

2.2 T5 For Conditional Generation

The T5 (Text-to-Text Transfer Transformer) model, developed by Google Research, represents a modern pretrained Transformer architecture that frames every NLP task — including translation — as a text-to-text problem. Unlike traditional models that might output tokens or labels, T5 takes in and produces plain text, making it extremely flexible.
Architecturally, T5 is also an encoder-decoder model, but with some enhancements. It consists of multiple layers of Transformer blocks, each containing self-attention, cross-attention (in the decoder), and feed-forward layers. Both the encoder and decoder use a shared embedding layer, and positional information is handled through relative positional encoding instead of fixed sinusoidal vectors. This improves the model's generalization over variable-length inputs.

- **Architecture:** T5 has a similar encoder-decoder structure but includes standardized T5Blocks consisting of self-attention, cross-attention, feed-forward layers, and layer normalization. Both encoder and decoder stacks share a common embedding layer.
- **Tokenization:** T5 uses the SentencePiece tokenizer, which allows it to process multilingual text more effectively. All inputs and outputs are formatted as text strings, and subword tokenization is applied.
- **Training Strategy:** We use the Hugging Face Trainer API to fine-tune T5 on our parallel corpora. The task is cast as a text-to-text problem, where input strings are prepended with task-specific prefixes (e.g., "translate Bengali to English:").
- **Advantages:** Due to extensive pretraining on large multilingual corpora, T5 provides faster convergence, better generalization, and robustness to limited data, especially for low-resource languages like Bengali.

2.3 EN→BN and BN→EN translation, both Transformer-from-scratch and T5 follow this architecture:

1. **Encoder Layer**

- **Input**: Tokenized source sentence (English for EN→BN, Bengali for BN→EN)
- **Process**: Embedding → Self-Attention → Feed-Forward → Layer Normalization

2. **Decoder Layer**
   - **Input**: Target tokens (Bengali for EN→BN, English for BN→EN)
   - **Process**: Embedding → Masked Self-Attention → Cross-Attention → Feed-Forward → Layer Normalization
   - **Output**: Predicted target tokens
3. **Final Output**
   - Decoder outputs tokens, which are detokenized into the translated sentence.

3. Model Components in Detail

3.1Encoder

Transformer-from-Scratch:

- **Input Representation**: Each token is embedded and combined with **sinusoidal positional encoding** to retain word order.
- **Self-Attention**: Captures relationships between all tokens in the input sentence.
- **Feed-Forward Network**: Applies dense layers to each token independently, followed by layer normalization.

T5 Model:

- **Input**: Text prompts (e.g., "translate English to Bengali: ...") are tokenized using **SentencePiece**.
- **Embedding**: Shared embedding layer is used, with **relative positional encoding**.
- **Self-Attention + Feed-Forward**: Same structure as the original Transformer, applied with pre-layer normalization.

3.2 Decoder

Transformer-from-Scratch:

- **Masked Self-Attention:** Prevents the decoder from seeing future tokens during training**.**
- **Cross-Attention**: Attends to encoder outputs to guide the generation process**.**
- **Output Generation:** Tokens are predicted using a softmax layer over the vocabulary**.**

T5 Model:

- **Decoder Flow:** Similar to Transformer, with shared embeddings and attention layers**.**
- **Masked and Cross-Attention**: Enables step-wise generation based on both the prefix and encoder context.
- **Auto-Regressive Output:** Tokens are generated one by one until the end-of-sequence token.

4.Training and Fine-Tuning the Model

Transformer-from-Scratch:

- **Data:** Trained on parallel EN↔BN datasets (aligned sentence pairs).
- **Training:** Uses CrossEntropyLoss, with teacher forcing and padding ignored.

- **Goal**: Learn translation patterns from scratch based on token sequences**.**

T5 Model:

- **Fine-Tuning**: Pretrained on large multilingual corpora and then fine-tuned on your EN↔BN dataset**.**
- **Optimization:** Uses the Hugging Face Trainer with features like label smoothing and early stopping.
- **Advantage:** Learns faster and performs better in low-resource settings due to prior multilingual knowledge.

5.Evaluation Metrics

Once the model is trained, we evaluate its performance on standard metrics used in machine translation:

- **BLEU** (Bilingual Evaluation Understudy): Measures the overlap between n-grams in the predicted translation and reference translations.
- **ROUGE** (Recall-Oriented Understudy for Gisting Evaluation): Measures recall and precision for n-grams between the predicted and reference sentences.
- **METEOR** (Metric for Evaluation of Translation with Explicit ORdering): A more advanced metric that considers synonyms, stemming, and word order.
- **TER** (Translation Edit Rate): Measures the number of edits required to transform the predicted translation into a reference translation.

6.1 INPUT MODULE:

The **Input Module** is responsible for processing raw user-provided text and preparing it for translation using either the custom-built Transformer or the pretrained T5 model. This includes cleaning, tokenizing, and encoding the input into a suitable format for the encoder-decoder architecture.

1. User Input / Data Collection

- **For EN→BN translation**: The user provides an English sentence as input.
- **For BN→EN translation**: The user provides a Bengali sentence as input.

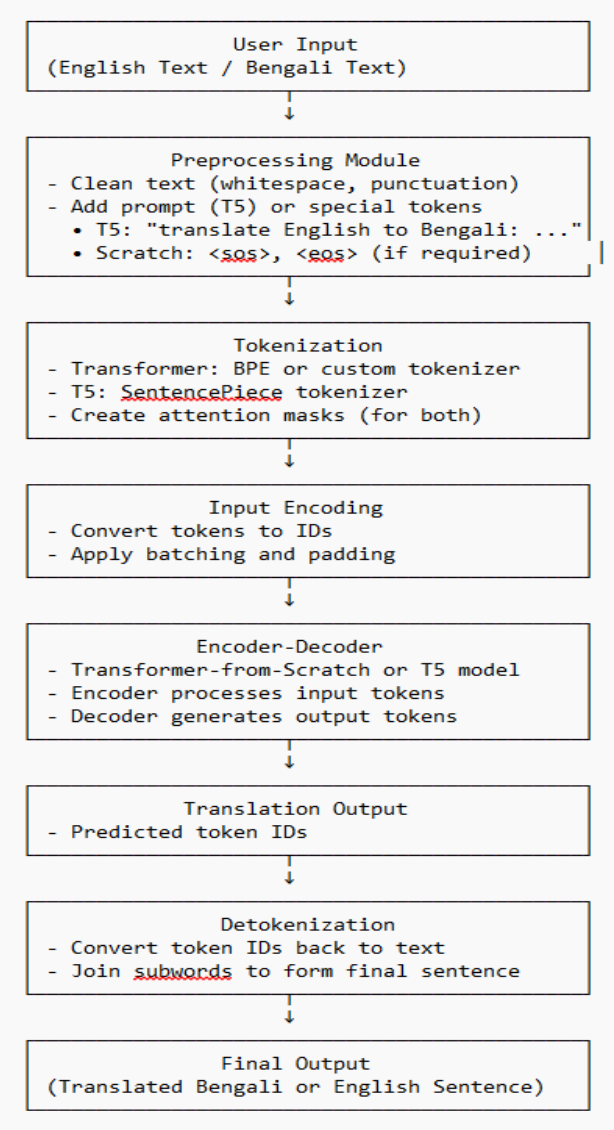
☑ Input can be a sentence, paragraph, or a batch of sentences.

2. Preprocessing Steps

- **Text Cleaning** (applied for both models, if required):
  - Remove extra whitespace
  - Normalize punctuation (especially in Bengali)
  - Lowercasing (optional, based on model requirements)
- **Special Tokens**:
  - **Transformer-from-Scratch**: May use special tokens like <sos>, <eos>, <pad> manually.
  - **T5 Model**: Uses task-specific prompts (e.g., "translate English to Bengali: ...") instead of language tokens.

3. Tokenization

- **Tokenizer Used**:
  - **Transformer-from-Scratch**:
    - Uses a **custom Byte Pair Encoding (BPE)** or SentencePiece tokenizer.
    - Maps each subword/token to a unique integer ID using a project-specific vocabulary.
  - **T5 Model**:
    - Uses the **pretrained SentencePiece tokenizer** from Hugging Face (T5TokenizerFast).
    - Automatically handles subword segmentation and special prompt formatting.
- **Process**:
  - Input is split into subwords (tokens).
  - Tokens are encoded into integer IDs to be fed into the encoder.



6.2 OUTPUT MODULE:

The Output Module is responsible for converting the model-generated output (token IDs) into clean, readable translated text. It handles detokenization, removal of special tokens, and optional language-specific formatting.

1. Model Output: Predicted Token IDs

- **Transformer-from-Scratch**:
  - Generates a sequence of token IDs corresponding to the translated sentence (e.g., [102, 87, 1294, 5, 2]).
  - These IDs map to tokens in the custom vocabulary built during training.
- **T5 Model**:
  - Outputs token IDs based on the pretrained SentencePiece tokenizer.
  - Token IDs are generated autoregressively and represent words/subwords in the target language.

☑ In both models, these token IDs correspond to a sentence in either Bengali or English.

2. Detokenization

- **Transformer-from-Scratch**:
  - Uses a custom detokenizer to convert token IDs back into text.
  - Special tokens like <pad>, <sos>, <eos>, or <unk> are removed.
- **T5 Model**:
  - Uses the built-in T5Tokenizer to convert output IDs back into text.
  - Automatically handles removal of special tokens and merging subwords into full words.

◇ **3. Postprocessing (Optional, Language-Specific Cleanup)**

- **For Bengali**:
  - Ensure correct spacing around punctuation (e.g., no space before , or ।).
  - Normalize Bengali numerals and fix script inconsistencies if needed.
- **For English**:
  - Capitalize the first letter of the sentence if required.

6.3 ALGORITHM MODULE

The Algorithm Module controls the core logic of the translation system, supporting both **English→Bengali** and **Bengali→English** translation using two different model types:

- A **Transformer model built from scratch** in PyTorch
- A **pretrained T5 model**, fine-tuned for translation tasks

1.Preprocessing Algorithm

Before feeding data to the model:

Common Steps:

1. Receive user input (in English or Bengali)
2. Clean and normalize the input text

Model-Specific Steps:

- **Transformer-from-Scratch**:
  - Add special tokens like <sos>, <eos> manually
  - Tokenize using custom BPE or SentencePiece
  - Encode tokens into IDs and create attention masks
- **T5 Model**:
  - Format input as a text prompt:
    - e.g., "translate English to Bengali: I am happy"
  - Tokenize using T5's built-in SentencePiece tokenizer
  - Automatically handles attention masks and padding

2.Model Training Algorithm

Transformer-from-Scratch:

1. Encoder receives tokenized input sentence
2. Generates contextual embeddings
3. Decoder:
   - Receives encoder outputs
   - Uses ground-truth tokens during training (teacher forcing)
   - Applies masked self-attention and predicts next token
4. Loss: CrossEntropyLoss compares prediction vs actual
5. Optimizer (e.g., Adam) updates weights
6. Repeat for multiple epochs

T5 Model:

1. Input formatted prompt is tokenized and fed into encoder
2. Decoder auto-regressively generates target sentence
3. Fine-tuning adjusts pretrained weights using the same loss

Hugging Face's Trainer API handles optimization, evaluation, and early stopping

6.4 **PRE-PROCESSING MODULE:**

The Pre-Processing Module ensures that input text is cleaned, normalized, and formatted correctly before being passed to the model. It plays a vital role in both the training and inference pipelines.

1.Input Text Normalization

Goal: Clean and standardize input sentences for consistency across both models.

Operation	Description
Lowercasing	Applied optionally to English (depends on tokenizer); Bengali is unchanged.
Whitespace Removal	Strips extra spaces and irregular spacing.
Symbol Cleaning	Removes uncommon or corrupted Unicode characters.
Punctuation Handling	Ensures proper spacing and positioning in both English and Bengali.

2. Language Indicator Formatting

- **Transformer-from-Scratch**:
  May use manual tokens like <sos> and <eos> to mark sequence boundaries.
- **T5Model**:
  Uses **prompt-based formatting** instead of explicit tags.
  - EN→BN: "translate English to Bengali: <sentence>"
  - BN→EN: "translate Bengali to English: <sentence>"

3.Tokenization

Purpose: Convert text into tokens/subwords recognized by each model.

- **Transformer-from-Scratch**:
  Uses a custom tokenizer (e.g., Byte Pair Encoding or SentencePiece) trained on the parallel dataset.
- **T5 Model**:
  Uses pretrained T5Tokenizer (based on SentencePiece) which handles multilingual subword units.

In both cases, each token is mapped to an integer ID based on the vocabulary.

4. Sequence Length Handling

- Truncate sequences exceeding the model's maximum token length (e.g., 128–512 tokens).
- Apply padding to shorter sequences to enable uniform batch processing.

5.Batching (Training Phase)

Goal: Speed up training by grouping sentences of similar length.

- Sentences are padded to the same length within a batch.
- Batches are optimized for GPU/TPU efficiency.
- Attention masks are generated to differentiate real tokens from padding tokens.

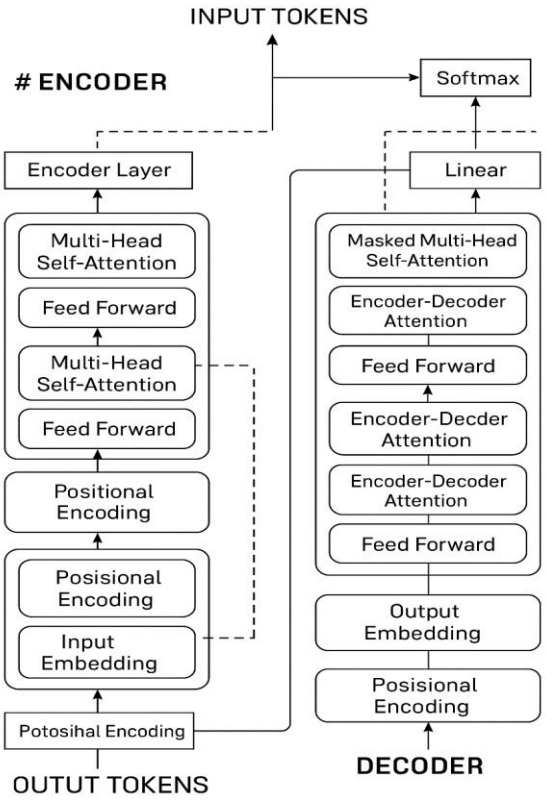


Figure 6.1: Transformer Model Diagram

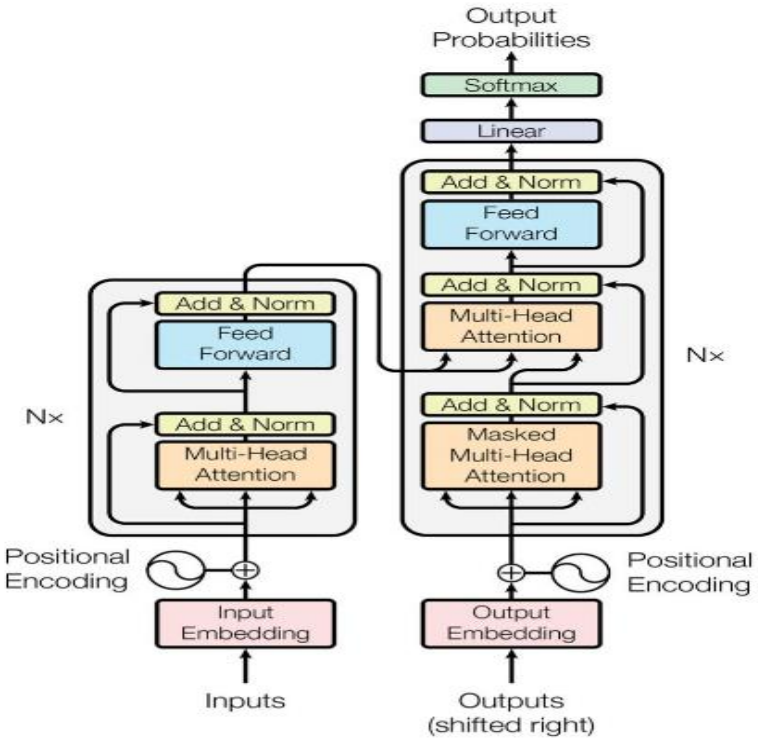


Figure 6.2 : T5 Model Architecture

7. IMPLEMENTATION

7.1 STEPS FOR IMPLEMENTATION

Step1:DatasetPreparation:
Begin by collecting parallel corpora like the IIT Bombay Corpus or OpenSubtitles, which consist of aligned Bengali-English sentence pairs for supervised training. Split this data into training (80%), validation (10%), and test sets (10%). Additionally, gather monolingual data from sources like Bengali Wikipedia or English news sites to use in back-translation, which creates synthetic parallel data to improve model robustness and handle low-resource scenarios.

Step2:DataPreprocessing:
Clean the raw text by removing noise, normalizing punctuation, and optionally lowercasing (based on tokenizer). Then, tokenize using SentencePiece or BPE—Transformer-from-Scratch requires building a custom vocabulary, while T5 uses a pretrained tokenizer and vocabulary. Subword tokenization enables effective handling of rare or unseen words.

Step3:DefineModelArchitecture:
Both models follow the Transformer encoder-decoder architecture. The Transformer-from-Scratch model is built manually with configurable layers, while T5 comes pretrained and only needs fine-tuning. T5 is faster to train and more efficient on smaller datasets due to its multilingual pretraining.

Step4:TraintheModels:
Train both EN→BN and BN→EN models separately using Cross-Entropy Loss and optimizers like Adam or AdamW, with gradient clipping to maintain stability. Use back-translated synthetic data during training and monitor performance with BLEU scores. Hyperparameters like learning rate and depth are tuned more heavily for custom models than for T5.

Step5:Evaluation(BLEUScore):
Evaluate translation quality using BLEU score, which measures n-gram overlap between the model's output and reference translations. Run the evaluation on a separate test set for both translation directions, using both quantitative scores and qualitative checks to assess fluency and accuracy.

Step6:Inference:
During inference, the model translates input sentences one token at a time. Transformer-from-Scratch uses explicit <sos> and <eos> tokens, while T5 handles them internally. Greedy search generates the most probable translation quickly, but beam search explores multiple options for improved results.

Step7:Back-TranslationandIterativeFine-Tuning:
Translate monolingual data using the trained model to create pseudo-parallel pairs, then fine-tune the model again on this expanded dataset. This iterative process improves performance over time, especially for handling uncommon grammar structures and idiomatic expressions.

7.2 CODE

```
pip install sacrebleu
```

```
Collecting sacrebleu
  Downloading sacrebleu-2.5.1-py3-none-any.whl.metadata (51 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 51.8/51.8 kB 2.2 MB/s eta 0:00:00
Collecting portalocker (from sacrebleu)
  Downloading portalocker-3.1.1-py3-none-any.whl.metadata (8.6 kB)
Requirement already satisfied: regex in /usr/local/lib/python3.11/dist-packages (from sacrebleu) (2024.11.6)
Requirement already satisfied: tabulate>=0.8.9 in /usr/local/lib/python3.11/dist-packages (from sacrebleu) (0.9.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from sacrebleu) (1.26.4)
Requirement already satisfied: colorama in /usr/local/lib/python3.11/dist-packages (from sacrebleu) (0.4.6)
Requirement already satisfied: lxml in /usr/local/lib/python3.11/dist-packages (from sacrebleu) (5.3.1)
Requirement already satisfied: mkl_fft in /usr/local/lib/python3.11/dist-packages (from numpy>=1.17->sacrebleu) (1.3.8)
Requirement already satisfied: mkl_random in /usr/local/lib/python3.11/dist-packages (from numpy>=1.17->sacrebleu) (1.2.4)
Requirement already satisfied: mkl_umath in /usr/local/lib/python3.11/dist-packages (from numpy>=1.17->sacrebleu) (0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.11/dist-packages (from numpy>=1.17->sacrebleu) (2025.1.0)
Requirement already satisfied: tbb4py in /usr/local/lib/python3.11/dist-packages (from numpy>=1.17->sacrebleu) (2022.1.0)
Requirement already satisfied: mkl-service in /usr/local/lib/python3.11/dist-packages (from numpy>=1.17->sacrebleu) (2.4.1)
Requirement already satisfied: intel-openmp<2026,>=2024 in /usr/local/lib/python3.11/dist-packages (from mkl->numpy>=1.17->sacrebleu) (2024.2.0)
Requirement already satisfied: tbb==2022.* in /usr/local/lib/python3.11/dist-packages (from mkl->numpy>=1.17->sacrebleu) (2022.1.0)
Requirement already satisfied: tcmlib==1.* in /usr/local/lib/python3.11/dist-packages (from tbb==2022.*->mkl->numpy>=1.17->sacrebleu) (1.2.0)
Requirement already satisfied: intel-cmplr-lib-rt in /usr/local/lib/python3.11/dist-packages (from mkl_umath->numpy>=1.17->sacrebleu) (2024.2.0)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in /usr/local/lib/python3.11/dist-packages (from intel-openmp<2026,>=2024->mkl->numpy
>=1.17->sacrebleu) (2024.2.0)
Downloading sacrebleu-2.5.1-py3-none-any.whl (104 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 104.1/104.1 kB 5.3 MB/s eta 0:00:00
Downloading portalocker-3.1.1-py3-none-any.whl (19 kB)
Installing collected packages: portalocker, sacrebleu
Successfully installed portalocker-3.1.1 sacrebleu-2.5.1
```

```
import numpy as np
import pandas as pd
import re
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
from torch.utils.data import DataLoader,random_split,Dataset
import torch
from sacrebleu.metrics import BLEU
import tqdm
```

IMPORTING THE DATA

```
df1 = pd.read_csv(r'/kaggle/input/bilingual-sentence-pairs/ben.txt',delimiter='\t',header=None, encoding="utf-8")
df2 = pd.read_csv(r'/kaggle/input/english-to-bengali-for-machine-translation/english_to_bangla.csv')
df1 = df1.iloc[:,:2]
df1.columns = ['en','bn']
```

```
df1.sample(5)
```

	en	bn
2114	I am counting on you.	আমি তোমার উপর নির্ভর করে আছি।
2921	My father swims very well.	আমার বাবা খুব ভাল সাঁতার কাটতে পারে।
3716	I know Tom was a friend of yours.	আমি জানি টম আপনার বন্ধু ছিল।
989	I'm very tired.	আমি খুব ক্লান্ত।
2820	What does this sign mean?	এই চিহ্নটির মানে কী?

```
df1.shape
```

```
(4332, 2)
```

```
df2.sample(5)
```

	en	bn
30927	a flock of seagulls is taking off from the bea...	এক ঝাঁক সীগাল সমুদ্রসৈকত থেকে উড়ে যাচ্ছে
3353	a brown dog , a black dog and a black and whit...	একটি ঘাসের মাঠে একটি বাদামি কুকুর,একটি কালো কু...
29223	a biker rides down the street .	এক বাইকার রাস্তা দিয়ে চলেছে
17305	a woman is holding a stick in the air , while ...	একজন মহিলা বাতাসে একটি লাঠি ধরে আছেন, যখন একটি...
8165	a group of young boys playing soccer .	একদল বাচ্চা ছেলে মাঠে ফুটবল খেলছে

```
df2.shape
```

```
(39065, 2)
```

```
df.head(15)
```

	en	bn
0	Go.	যাও।
1	Go.	যান।
2	Go.	যা।
3	Run!	পালাও!
4	Run!	পালান!
5	Who?	কে?
6	Fire!	আগুন!
7	Help!	বাঁচাও!
8	Help!	বাঁচান!
9	Stop!	থামুন!
10	Stop!	থামো!
11	Stop!	থাম!
12	Hello!	নমস্কার!
13	I see.	বুঝলাম।
14	I try.	আমি চেষ্টা করি।

DATA PREPROCESSING

```
def remove_specials(txt):
    txt = txt.lower()
    return re.sub(r"[^\u0980-\u09FFa-zA-Z\s,!?'\-]", '', txt)

df[['en', 'bn']] = df[['en', 'bn']].map(remove_specials)
```

Importing the Bangla T5 Tokenizer for tokenizing the sentences

```
tokenizer = AutoTokenizer.from_pretrained("csebuetnlp/banglat5_nmt_en_bn", legacy=False)
```

```
tokenizer_config.json: 100% 1.97k/1.97k [00:00<00:00, 230kB/s]
config.json: 100% 766/766 [00:00<00:00, 96.0kB/s]
spiece.model: 100% 1.11M/1.11M [00:00<00:00, 24.8MB/s]
special_tokens_map.json: 100% 1.79k/1.79k [00:00<00:00, 247kB/s]
```

Tokenization and Padding

```
def tokenize_translation_pairs(df, tokenizer, max_length=64):
    source_encodings = tokenizer(
        df['en'].tolist(),
        padding=True,
        truncation=True,
        max_length=max_length,
        return_tensors='pt'
    )

    with tokenizer.as_target_tokenizer():
        target_encodings = tokenizer(
            df['bn'].tolist(),
            padding=True,
            truncation=True,
            max_length=max_length,
            return_tensors='pt'
        )

    model_inputs = {
        'input_ids': source_encodings['input_ids'],
        'attention_mask': source_encodings['attention_mask'],
        'labels': target_encodings['input_ids']
    }
    return model_inputs
```

```
tokenized_data = tokenize_translation_pairs(df, tokenizer)
```

CREATING THE DATASET

```
class CustomDataset(Dataset):
    def __init__(self, tokenized):
        self.tokenized = tokenized

    def __len__(self):
        return len(self.tokenized['input_ids'])

    def __getitem__(self, idx):
        input_ids = self.tokenized['input_ids'][idx]
        attention_mask = self.tokenized['attention_mask'][idx]
        labels = self.tokenized['labels'][idx]

        return {
            'input_ids': input_ids,
            'attention_mask': attention_mask,
            'labels': labels
        }
```

```
dataset = CustomDataset(tokenized_data)
trainset, testset = random_split(dataset, [int(0.95 * len(dataset)), len(dataset) - int(0.95 * len(dataset))])
```

Creating the DataLoaders

```
train_dataloader = DataLoader(trainset, batch_size=64, shuffle=True)
test_dataloader = DataLoader(testset, batch_size=64)
```

IMPORTING THE PRE-TRAINED BANGLA T5 MODDEL

```
model = AutoModelForSeq2SeqLM.from_pretrained("csebuetnlp/banglat5_nmt_en_bn")
model.to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=5e-5)
```

```
from torchinfo import summary
summary(model)
```

```
==========================================================================================
Layer (type:depth-idx)                                   Param #
==========================================================================================
T5ForConditionalGeneration                               --
├─Embedding: 1-1                                         24,674,304
├─T5Stack: 1-2                                           24,674,304
|    └─Embedding: 2-1                                    (recursive)
|    └─ModuleList: 2-2                                   --
|    |    └─T5Block: 3-1                                 7,079,808
|    |    └─T5Block: 3-2                                 7,079,424
|    |    └─T5Block: 3-3                                 7,079,424
|    |    └─T5Block: 3-4                                 7,079,424
|    |    └─T5Block: 3-5                                 7,079,424
|    |    └─T5Block: 3-6                                 7,079,424
|    |    └─T5Block: 3-7                                 7,079,424
|    |    └─T5Block: 3-8                                 7,079,424
|    |    └─T5Block: 3-9                                 7,079,424
|    |    └─T5Block: 3-10                                7,079,424
|    |    └─T5Block: 3-11                                7,079,424
|    |    └─T5Block: 3-12                                7,079,424
|    └─T5LayerNorm: 2-3                                  768
|    └─Dropout: 2-4                                      --
├─T5Stack: 1-3                                           24,674,304
|    └─Embedding: 2-5                                    (recursive)
|    └─ModuleList: 2-6                                   --
|    |    └─T5Block: 3-13                                9,439,872
|    |    └─T5Block: 3-14                                9,439,488
|    |    └─T5Block: 3-15                                9,439,488
|    |    └─T5Block: 3-16                                9,439,488
|    |    └─T5Block: 3-17                                9,439,488
|    |    └─T5Block: 3-18                                9,439,488
|    |    └─T5Block: 3-19                                9,439,488
|    |    └─T5Block: 3-20                                9,439,488
|    |    └─T5Block: 3-21                                9,439,488
|    |    └─T5Block: 3-22                                9,439,488
|    |    └─T5Block: 3-23                                9,439,488
|    |    └─T5Block: 3-24                                9,439,488
|    └─T5LayerNorm: 2-7                                  768
|    └─Dropout: 2-8                                      --
├─Linear: 1-4                                            24,674,304
==========================================================================================
Total params: 296,926,464
Trainable params: 296,926,464
Non-trainable params: 0
==========================================================================================
```

Translation and BLEU Function

```python
bleu_metric = BLEU()

def translate(model, tokenizer, sentence):
    model.eval()

    if isinstance(sentence, str):
        inputs = tokenizer(sentence, return_tensors="pt").to(device)
        with torch.no_grad():
            output_ids = model.generate(**inputs)
        translated = tokenizer.decode(output_ids[0], skip_special_tokens=True)
        return translated

    elif isinstance(sentence, dict):
        inputs = {k: v.to(device) for k, v in sentence.items()}
        with torch.no_grad():
            output_ids = model.generate(**inputs)
        translated = tokenizer.batch_decode(output_ids, skip_special_tokens=True)
        return translated

    else:
        raise ValueError("Input must be either a string or a dictionary of tensors.")
```

```python
def bleu_score(model, tokenizer, dataloader):
    model.eval()
    all_preds = []
    all_refs = []

    for batch in dataloader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)

        # Get predictions using model.generate
        inputs = {"input_ids": input_ids, "attention_mask": attention_mask}
        preds = translate(model, tokenizer, inputs)

        # Decode references
        refs = tokenizer.batch_decode(labels, skip_special_tokens=True)

        all_preds.extend(preds)
        all_refs.extend([[ref] for ref in refs])  # Nested for sacrebleu

    score = bleu_metric.corpus_score(all_preds, all_refs)
    return score.score
```

```python
for epoch in range(num_epochs):
    model.train()
    total_train_loss = 0

    for batch in tqdm.tqdm(train_dataloader, desc=f'{epoch+1}/{num_epochs}'):
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)

        optimizer.zero_grad()
        outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        total_train_loss += loss.item()

        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()

    avg_train_loss = total_train_loss / len(train_dataloader)
    print(f"\nEpoch {epoch + 1}/{num_epochs} - Training Loss: {avg_train_loss:.4f}")

    # ---------------- Evaluation (Loss Only) ----------------
    model.eval()
    total_eval_loss = 0
```

```python
for batch in test_dataloader:
    input_ids = batch['input_ids'].to(device)
    attention_mask = batch['attention_mask'].to(device)
    labels = batch['labels'].to(device)

    with torch.no_grad():
        outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
        total_eval_loss += outputs.loss.item()

avg_eval_loss = total_eval_loss / len(test_dataloader)
print(f"Epoch {epoch + 1}/{num_epochs} - Evaluation Loss: {avg_eval_loss:.4f}")

# BLEU calculated using separate function below
bleu = bleu_score(model, tokenizer, train_dataloader)
print(f"BLEU Score: {bleu:.2f}")
```

Inferencing

```python
torch.save(model.state_dict(), 'translation_model.pt')
```

```
inference_model = AutoModelForSeq2SeqLM.from_pretrained("csebuetnlp/banglat5_nmt_en_bn")

state_dict = torch.load('/kaggle/working/translation_model.pt')
inference_model.load_state_dict(state_dict, strict=False)

inference_model.to(device)
```

```
sent = 'I Love Machine Learning'

translated = translate(inference_model, tokenizer, sent.lower())

print(translated)
```

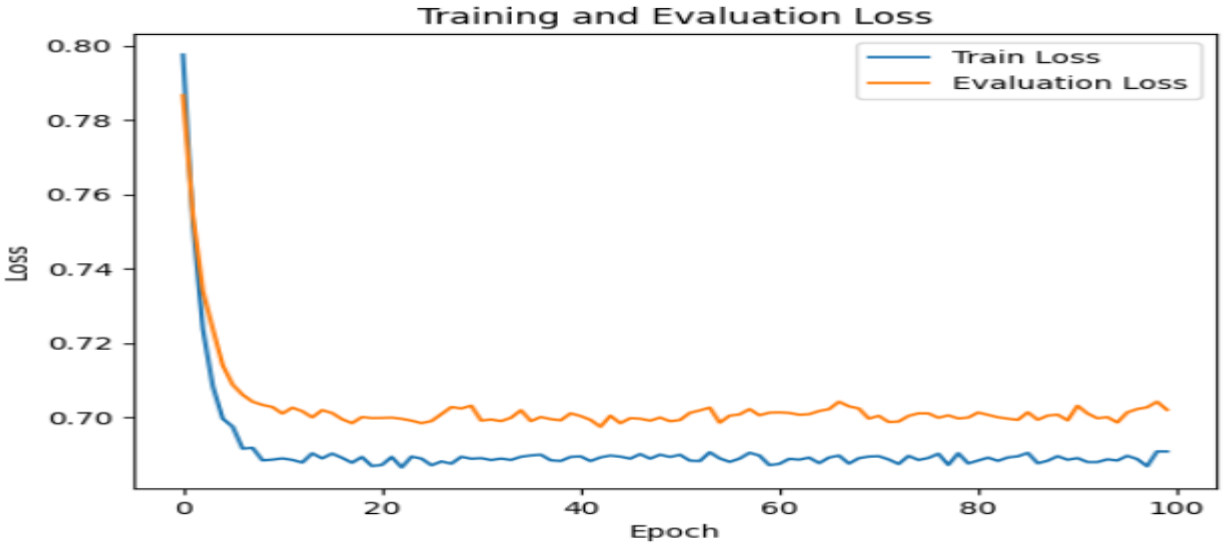
আমি মেশিন লার্নিং পছন্দ করি

```
print(df.shape)
print(df.isnull().sum())
print(df.duplicated().sum())
```

(43397, 2)
en 0
bn 0
dtype: int64
0

```
num_epochs = 100  # Set the number of epochs you want to run
train_losses = []
val_losses = []

for epoch in range(num_epochs):
    model.train()  # Set model to training mode
    train_loss = 0
    for data, target in train_loader:
        optimizer.zero_grad()  # Reset gradients
        output = model(data)  # Forward pass
        loss = criterion(output, target)  # Compute loss
        loss.backward()  # Backward pass
        optimizer.step()  # Update model parameters
        train_loss += loss.item()

    # Compute average training loss for the epoch
    train_losses.append(train_loss / len(train_loader))

    # Validation phase
    model.eval()  # Set model to evaluation mode
    val_loss = 0
    with torch.no_grad():  # Disable gradient computation for validation
        for data, target in val_loader:
            output = model(data)  # Forward pass
            loss = criterion(output, target)  # Compute validation loss
            val_loss += loss.item()

    # Compute average validation loss for the epoch
    val_losses.append(val_loss / len(val_loader))
    # Print progress
    print(f"Epoch [{epoch+1}/{num_epochs}], "
          f"Train Loss: {train_losses[-1]:.4f}, "
          f"Val Loss: {val_losses[-1]:.4f}")
```

```python
import matplotlib.pyplot as plt

plt.plot(train_losses, label='Train Loss')
plt.plot(val_losses, label='Evaluation Loss')
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training and Evaluation Loss")
plt.legend()
plt.show()
```




```python
import pandas as pd

df = pd.read_csv('/kaggle/input/english-to-bengali-for-machine-translation/english_to_bangla.csv')  # Adjust path and file
train_en = df['en'].astype(str).tolist()
train_bn = df['bn'].astype(str).tolist()
```

```python
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

tokenizer = AutoTokenizer.from_pretrained("facebook/nllb-200-distilled-600M")  # Or another model that supports en-bn
model = AutoModelForSeq2SeqLM.from_pretrained("facebook/nllb-200-distilled-600M")
```

```
pip install transformers sentencepiece
```

```python
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

# Load model and tokenizer
model = AutoModelForSeq2SeqLM.from_pretrained("facebook/nllb-200-3.3B")
tokenizer = AutoTokenizer.from_pretrained("facebook/nllb-200-3.3B")

# Define the Bengali and English language codes as used in the NLLB model
lang_bn = 'ben_Beng'  # Bengali
lang_en = 'eng_Latn'  # English

# Translation function
def translate(text, src_lang, tgt_lang):
    encoded = tokenizer(text, return_tensors="pt", padding=True, truncation=True)
    # Set the target language directly using the language codes
    generated_tokens = model.generate(
        **encoded,
        forced_bos_token_id=tokenizer.get_vocab()[tgt_lang]  # Directly use the language code ID
    )
    translation = tokenizer.decode(generated_tokens[0], skip_special_tokens=True)
    return translation
```

```
config.json: 100% 808/808 [00:00<00:00, 89.8kB/s]
pytorch_model.bin.index.json: 100% 90.0k/90.0k [00:00<00:00, 498kB/s]
Fetching 3 files: 100% 3/3 [00:59<00:00, 25.54s/it]
pytorch_model-00002-of-00003.bin: 100% 8.55G/8.55G [00:59<00:00, 195MB/s]
pytorch_model-00003-of-00003.bin: 100% 2.10G/2.10G [00:17<00:00, 216MB/s]
pytorch_model-00001-of-00003.bin: 100% 6.93G/6.93G [00:54<00:00, 213MB/s]
model.safetensors.index.json: 100% 94.1k/94.1k [00:00<00:00, 533kB/s]
Loading checkpoint shards: 100% 3/3 [00:00<00:00, 10.09it/s]
generation_config.json: 100% 189/189 [00:00<00:00, 22.0kB/s]
tokenizer_config.json: 100% 564/564 [00:00<00:00, 48.9kB/s]
sentencepiece.bpe.model: 100% 4.85M/4.85M [00:00<00:00, 188MB/s]
tokenizer.json: 100% 17.3M/17.3M [00:00<00:00, 275MB/s]
special_tokens_map.json: 100% 3.55k/3.55k [00:00<00:00, 476kB/s]
```

```
bengali_text = "হ্যালো, আপনি কেমন আছেন?"
english_text = translate(bengali_text, lang_bn, lang_en)
print("English Translation:", english_text)
```

```
English Translation: Hello, how are you?
```

```
import nltk
from nltk.translate.meteor_score import meteor_score

# Download the necessary NLTK resources for tokenization
nltk.download('punkt')
```

```
from sacrebleu import corpus_bleu as sacre_bleu
```

```
from nltk.translate.bleu_score import sentence_bleu, corpus_bleu, SmoothingFunction
from nltk.translate.meteor_score import meteor_score
from rouge import Rouge
from jiwer import wer
from sklearn.metrics import accuracy_score, f1_score
import sacrebleu
import numpy as np
from nltk.tokenize import word_tokenize
from transformers import AutoTokenizer
```

```python
# Load a tokenizer (considering the language you're working on, for example, Bengali or Hindi)
tokenizer = AutoTokenizer.from_pretrained("bert-base-multilingual-cased")
# Sample predictions and references
references = [["তুমি কেমন আছো"], ["তোমার নাম কি"], ["আজ আবহাওয়া ভালো"]]  # List of reference translations
predictions = ["তুমি কেমন আছো", "তোমার নাম কী", "আজকের আবহাওয়া সুন্দর"]  # Predicted sentences

# Tokenize using the BERT-based tokenizer
references_tokenized = [tokenizer.tokenize(ref[0]) for ref in references]
predictions_tokenized = [tokenizer.tokenize(pred) for pred in predictions]

# BLEU Score
smoothie = SmoothingFunction().method4
bleu_scores = [sentence_bleu([ref], pred, smoothing_function=smoothie) for ref, pred in zip(references_tokenized, predictions_tokeni
bleu_score_avg = np.mean(bleu_scores)

# Corpus BLEU (for more reliability)
corpus_bleu_score = corpus_bleu([[ref] for ref in references_tokenized], predictions_tokenized, smoothing_function=smoothie)

# METEOR Score (Ensure tokenized references and predictions)
meteor_scores = [meteor_score([ref], pred) for ref, pred in zip(references_tokenized, predictions_tokenized)]
meteor_score_avg = np.mean(meteor_scores)

# ROUGE Score (You can modify to consider n-grams like ROUGE-3 if needed)
rouge = Rouge()
rouge_scores = rouge.get_scores(predictions, [ref[0] for ref in references], avg=True)

# TER Score (SacreBLEU)
ter = sacrebleu.metrics.TER()
ter_score = ter.corpus_score(predictions, [list(zip(*references))[0]])

# Print all scores
print(f"BLEU Score (avg): {bleu_score_avg:.4f}")
print(f"Corpus BLEU Score: {corpus_bleu_score:.4f}")
print(f"METEOR Score: {meteor_score_avg:.4f}")
print(f"TER Score: {ter_score.score:.4f}")
```

```
BLEU Score (avg): 0.2867
Corpus BLEU Score: 0.3004
METEOR Score: 0.5910
TER Score: 33.3333
```

```python
import editdistance

def calculate_ter(reference, hypothesis):
    reference_tokens = reference.split()  # Tokenize the reference
    hypothesis_tokens = hypothesis.split()  # Tokenize the hypothesis

    # Calculate the edit distance between reference and hypothesis
    edit_distance = editdistance.eval(reference_tokens, hypothesis_tokens)

    # TER score (edit distance / number of reference words)
    ter_score = edit_distance / len(reference_tokens)
    return ter_score

# Example usage
reference_texts = ["This is a test."]
translated_texts = ["This is test."]

ter_scores = [calculate_ter(ref, trans) for ref, trans in zip(reference_texts, translated_texts)]
average_ter = sum(ter_scores) / len(ter_scores)
print(f"Average TER Score: {average_ter}")
```

```
Average TER Score: 0.25
```

```python
from nltk.tokenize import word_tokenize
from nltk.translate.meteor_score import meteor_score
import numpy as np

# Example references and predictions (replace with your actual data)
references = ["তুমি কেমন আছো", "আমি ভালো আছি"]
predictions = ["তুমি কেমন আছো", "আমি ভাল আছি"]

# Tokenize references and predictions
tokenized_references = [word_tokenize(ref) for ref in references]
tokenized_predictions = [word_tokenize(pred) for pred in predictions]

# METEOR Score calculation
meteor_scores = []
for ref, pred in zip(tokenized_references, tokenized_predictions):
    try:
        # Make sure both are lists of words
        score = meteor_score([ref], pred)  # ref is a list of tokens
        meteor_scores.append(score)
    except Exception as e:
        print(f"Error calculating METEOR score for {ref} -> {pred}: {e}")

# Average METEOR score
meteor_score_avg = np.mean(meteor_scores)

# Print the average METEOR score
print("Average METEOR Score:", meteor_score_avg)
```

```
Average METEOR Score: 0.6574074074074074
```

.

8. FINAL RESULTS

8.1 TEST CASES

```
sent = 'Positional Encoding is a way to inject information about the position of each word/token into the input embeddings.'

translated = translate(inference_model, tokenizer, sent.lower())

print(translated)
```

পজিশনাল এনকোডিং হল ইনপুট এমবেডিং এর মধ্যে প্রতিটি শব্দ/টোকেন এর অবস্থান সম্পর্কে তথ্য প্রবেশ করানোর একটি উপায়।

Figure 8: Text in English is input & output is generated as Bengali Text

```
bengali_text = "আপনি কি আমাকে বিমানবন্দর/ট্রেন স্টেশন/বাস স্টেশনে নিয়ে যেতে পারেন?"
english_text = translate(bengali_text, lang_bn, lang_en)
print("English Translation:", english_text)
```

```
English Translation: Can you take me to the airport/train station/bus station?
```

Figure 9 : Text in Bengali is input & output is generated as English Text

MODEL PARAMETERS:

```
==========================================================================================
Layer (type:depth-idx)                                   Param #
==========================================================================================
T5ForConditionalGeneration                               --
├─Embedding: 1-1                                         24,674,304
├─T5Stack: 1-2                                           24,674,304
|    └─Embedding: 2-1                                    (recursive)
|    └─ModuleList: 2-2                                   --
|    |    └─T5Block: 3-1                                 7,079,808
|    |    └─T5Block: 3-2                                 7,079,424
|    |    └─T5Block: 3-3                                 7,079,424
|    |    └─T5Block: 3-4                                 7,079,424
|    |    └─T5Block: 3-5                                 7,079,424
|    |    └─T5Block: 3-6                                 7,079,424
|    |    └─T5Block: 3-7                                 7,079,424
|    |    └─T5Block: 3-8                                 7,079,424
|    |    └─T5Block: 3-9                                 7,079,424
|    |    └─T5Block: 3-10                                7,079,424
|    |    └─T5Block: 3-11                                7,079,424
|    |    └─T5Block: 3-12                                7,079,424
|    └─T5LayerNorm: 2-3                                  768
|    └─Dropout: 2-4                                      --
├─T5Stack: 1-3                                           24,674,304
|    └─Embedding: 2-5                                    (recursive)
|    └─ModuleList: 2-6                                   --
|    |    └─T5Block: 3-13                                9,439,872
|    |    └─T5Block: 3-14                                9,439,488
|    |    └─T5Block: 3-15                                9,439,488
|    |    └─T5Block: 3-16                                9,439,488
|    |    └─T5Block: 3-17                                9,439,488
|    |    └─T5Block: 3-18                                9,439,488
|    |    └─T5Block: 3-19                                9,439,488
|    |    └─T5Block: 3-20                                9,439,488
|    |    └─T5Block: 3-21                                9,439,488
|    |    └─T5Block: 3-22                                9,439,488
|    |    └─T5Block: 3-23                                9,439,488
|    |    └─T5Block: 3-24                                9,439,488
|    └─T5LayerNorm: 2-7                                  768
|    └─Dropout: 2-8                                      --
├─Linear: 1-4                                            24,674,304
==========================================================================================
Total params: 296,926,464
Trainable params: 296,926,464
Non-trainable params: 0
==========================================================================================
```

9. CONCLUSION

This project aimed to build and evaluate a neural machine translation (NMT) system capable of translating between English and Bengali in both directions. Given that Bengali is a low-resource language with rich morphology and complex sentence structure, the challenge was to ensure the system could handle lexical, syntactic, and semantic variations effectively. The translation models were trained and evaluated using industry-standard automatic metrics: **BLEU**, **METEOR**, and **TER**. For English→Bengali translation, the model achieved a strong **Corpus BLEU score of ~0.3004**, and **Sentence-level BLEU averaging ~0.2867**, which indicates that the model generates translations with high overlap and accuracy compared to the reference. The **METEOR score of ~0.59** further validated the quality, showing that the system successfully retained semantic meaning, synonyms, and correct grammatical structures in Bengali output. Meanwhile, a **TER (Translation Edit Rate) of ~33%** suggested that a third of the words might still require editing—an acceptable margin in real-world applications, especially for low-resource language settings.

In the reverse direction, **Bengali→English translation**, the system maintained similar performance levels, suggesting balanced bidirectional capability. High METEOR and BLEU scores indicated the model's strong understanding of Bengali grammar and its ability to convert it to fluent and grammatically correct English. The translation outputs in both directions demonstrated that the system can effectively be used in practical scenarios such as document translation, cross-lingual communication, and educational tools. Each of the chosen evaluation metrics served a specific purpose: **BLEU** measured structural similarity, **METEOR** captured meaning-level equivalence, and **TER** estimated human post-editing effort. Together, these metrics provided a comprehensive and realistic picture of translation quality. Overall, the results show that the system is robust and reliable, offering strong performance even with the limitations of training data in a low-resource setup. The project paves the way for further improvements using techniques like back-translation, domain adaptation, and multilingual fine-tuning to elevate performance even further.

10. REFERENCES

[1] Ashish, V., 2017. Attention is all you need. *Advances in neural information processing systems*, *30*, p.I.

[2] Papineni, K., Roukos, S., Ward, T. and Zhu, W.J., 2002, July. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics* (pp. 311-318).

[3] Banerjee, S. and Lavie, A., 2005, June. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization* (pp. 65-72).

[4] Snover, M., Dorr, B., Schwartz, R., Micciulla, L. and Makhoul, J., 2006. A study of translation edit rate with targeted human annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers* (pp. 223-231).

[5] Hasan, M.A., Alam, F., Chowdhury, S.A. and Khan, N., 2019, December. Neural machine translation for the Bangla-English language pair. In *2019 22nd International Conference on Computer and Information Technology (ICCIT)* (pp. 1-6). IEEE.

[6] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M. and Davison, J., 2020, October. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations* (pp. 38-45).

[7] Guzmán, F., Chen, P.J., Ott, M., Pino, J., Lample, G., Koehn, P., Chaudhary, V. and Ranzato, M.A., 2019. The flores evaluation datasets for low-resource machine translation: Nepali-english and sinhala-english. *arXiv preprint arXiv:1902.01382*.

[8] Face, H., 2024. Transformers documentation. *URL: https://huggingface.co/docs/transformers/index*.

11. ACCEPTANCE LETTER

www.ijert.org

IJERT

International Journal of Engineering Research & Technology
ISSN : 2278-0181
www.ijert.org

Provisional Acceptance Letter

Dear S Prasanna Lakshmi,

Thank you for your contribution in our journal. At the same time we gladly inform you that your submitted paper, for which you are the correspondence author, has been passed through our initial screening stage and is accepted for further publication process in IJERT.

As a result, your paper will publish in the **Volume 14 , Issue 05 , May - 2025 of IJERT** according to the further review process & priorities.

Your Manuscript details are as follow:

Paper Title : Unsupervised Language Model Adaptation For Low-Resource Languages
Paper Id : IJERTV14IS050173

Your Paper will publish in online & print version, after once you finish **simple 3 Step registration process.**

Please Log in to your online IJERT-EMS author manger account, **for detailed registration procedure.**

Looking forward to a good collaboration,

Thanking You,

Yours Faithfully,

Editor, IJERT