Project Report

Title:

Custom Transformer-Based NLP Pipeline for Bilingual Text Generation *(English ↔ Hindi)*

---

1. Introduction

This project implements a **Transformer architecture from scratch** in PyTorch for **English–Hindi machine translation** using the **IIT Bombay parallel corpus**.
Unlike approaches that rely on pre-built Transformer models from HuggingFace or OpenNMT, this implementation is **manual and modular**, covering the **entire pipeline** from data loading and preprocessing to training, evaluation, and inference.

Key motivations:

- Gain **deep understanding** of Transformer internals.
- Build **full forward and backward passes manually** using autograd.
- Avoid dependency on high-level NLP frameworks, ensuring flexibility and transparency.

---

2. System Architecture Overview

Modules and Their Roles

File	Purpose
config.py	Centralized configuration for hyperparameters, dataset paths, and training options.
dataset.py	Handles dataset loading, tokenization, padding, and batching for both source and target languages.
model.py	Defines the Transformer architecture including Encoder, Decoder, Multi-Head Attention, and Feed-Forward networks.
train.py	Implements the training loop, validation, checkpoint saving, and metric calculation.
translate.py	Loads trained model for inference on user-provided text or dataset samples.

---

3. Data Pipeline

Dataset

- **Source:** IIT Bombay English–Hindi Parallel Corpus
- **Structure:** Paired sentences (english, hindi)
- **Size:** Over 1.5M sentence pairs (subset used for training due to hardware limits)

Preprocessing Steps

1. **Tokenization**:
   - Custom WordLevel tokenizers for both languages.
   - Special tokens: [PAD], [SOS], [EOS], [UNK].
2. **Sequence Length Management**:
   - Fixed max length from config (max_seq_len).
   - Padding/truncation applied to both source and target sequences.
3. **Masking**:
   - **Padding mask** for ignoring [PAD] tokens in attention.
   - **Causal mask** for Decoder to prevent looking ahead during training.

---

4. Model Architecture

Core Components Implemented in model.py

1. **Positional Encoding**
   - Injects positional information using sine/cosine functions.
2. **Multi-Head Scaled Dot-Product Attention**
   - Splits queries, keys, values into num_heads.
   - Applies scaled dot-product attention in parallel.
   - Concatenates and projects back to d_model.
3. **Feed-Forward Network**
   - Position-wise dense layers with ReLU activation.
4. **Layer Normalization**
   - Applied before each sub-layer (Pre-LN architecture).
5. **Residual Connections**
   - Adds sub-layer output to its input before normalization.
6. **Encoder Block**
   - Multi-Head Attention + Feed Forward (stacked N times).
7. **Decoder Block**
   - Masked Multi-Head Self-Attention → Encoder–Decoder Attention → Feed Forward.
8. **Output Projection**
   - Linear layer mapping d_model to vocabulary size for target language.

---

5. Training Workflow

Script: train.py

Steps:

1. **Load Dataset**
   - Load IIT Bombay parallel corpus using dataset.py.
   - Create PyTorch DataLoader for batching.
2. **Initialize Model**
   - Encoder–Decoder Transformer with parameters from config.py.
3. **Loss & Optimizer**
   - Cross-Entropy Loss (ignoring PAD index).
   - Adam optimizer with learning rate scheduling (warm-up steps).
4. **Training Loop**
   - Teacher forcing: decoder receives previous correct token.
   - Track loss per batch.
5. **Validation**
   - Greedy decoding for validation sentences.
   - BLEU, CER, WER metrics calculated.
6. **Checkpointing**
   - Save model weights and optimizer state after each epoch.

---

6. Inference Pipeline

Script: translate.py

- Loads trained model from checkpoint.
- Tokenizes input text → passes through encoder → decodes step-by-step until [EOS].
- Supports:
  - **Custom text** entered by user.
  - **Dataset sample** translation by index.

Example usage:

python translate.py --text "How are you?"

python translate.py --idx 42

---

7. Key Achievements

1. **From-Scratch Implementation**
   - Fully manual Transformer construction in PyTorch.
   - Forward/backward passes without external model APIs.
2. **Core Module Development**
   - Attention mechanisms, positional encodings, masking, normalization.
3. **Performance**
   - Achieved meaningful bilingual translations after training subset.
   - BLEU score showing competitive results for custom model size.
4. **Research Understanding**
   - Direct mapping of "Attention Is All You Need" architecture into code.
   - Insights into hyperparameter tuning and sequence-to-sequence training.