English–Urdu Neural Machine Translation with LSTM and Transformer Architectures on the UMC005 Corpus

Moiz Sajjad
Dept. of Computer Science
FAST NUCES
Islamabad, Pakistan
i212691@nu.edu.pk

***Abstract*—This paper presents a comparative study of neural machine translation (NMT) systems for English-to-Urdu translation using the UMC005 parallel corpus. We implement and evaluate two architectures: a sequence-to-sequence LSTM model with Bahdanau attention and a Transformer encoder–decoder. Both models are trained from scratch on a dataset of 4,793 sentence pairs derived from religious texts (Quran). We evaluate performance using BLEU, ROUGE-L, and perplexity metrics. The Transformer achieves a BLEU score of 2.16 and ROUGE-L of 6.76, outperforming the LSTM baseline (BLEU: 0.42, ROUGE-L: 4.29). However, both models exhibit limited translation quality due to small model size, limited training epochs, word-level tokenization, and domain-specific training data. We also develop an interactive Gradio-based interface for real-time translation and visualize cross-attention patterns in the Transformer to understand source–target alignment. Our findings demonstrate that while Transformers show promise for low-resource language pairs, significant improvements are needed through larger architectures, subword tokenization, and domain adaptation.**

***Index Terms*—Neural Machine Translation, Transformer, LSTM, English–Urdu, UMC005 Corpus, Generative AI**

I. Introduction

Machine translation (MT) has evolved from rule-based and statistical approaches to modern neural architectures that learn end-to-end mappings between languages. English–Urdu translation presents unique challenges due to differences in script (Latin vs. Arabic-based), word order, and limited parallel corpora compared to high-resource language pairs. This work explores two prominent neural architectures for this task: the sequence-to-sequence LSTM with attention mechanism and the Transformer model based on self-attention.

The sequence-to-sequence LSTM with attention, introduced by Bahdanau et al., uses recurrent networks to encode source sentences and decode target sequences while attending to relevant source positions. The Transformer architecture, proposed by Vaswani et al., replaces recurrence entirely with multi-head self-attention and feed-forward layers, enabling parallel processing and often superior performance on large datasets.

This assignment implements both architectures from scratch using PyTorch, trains them on the UMC005 English–Urdu corpus, and compares their performance. We contribute: (1) a detailed comparison of LSTM and Transformer models on a low-resource language pair, (2) an interactive translation interface using Gradio, (3) attention visualization to understand model behavior, and (4) analysis of limitations and future directions. The code is designed to run in Google Colab with GPU acceleration.

II. Related Work

Classical machine translation relied on phrase-based statistical methods that aligned source and target phrases using parallel corpora. These systems required extensive feature engineering and linguistic resources. Neural machine translation (NMT) emerged as a paradigm shift, learning continuous representations that capture semantic and syntactic relationships automatically.

Sutskever et al. introduced the encoder–decoder framework using LSTM networks, where an encoder processes the source sentence into a fixed-size vector, and a decoder generates the target sequence. However, this bottleneck limited performance on long sequences. Bahdanau et al. addressed this by introducing attention mechanisms that allow the decoder to focus on different parts of the source sentence at each generation step, significantly improving translation quality.

The Transformer architecture, proposed by Vaswani et al., revolutionized NMT by replacing recurrent layers with self-attention. The model uses multi-head attention to capture relationships between all positions in parallel, enabling faster training and better handling of long-range dependencies. Transformers have become the foundation for state-of-the-art models like BERT, GPT, and modern MT systems.

For English–Urdu translation, prior work has explored various approaches, but limited parallel data remains a challenge. The UMC005 corpus provides a small but valuable resource for training and evaluation.

III. Dataset and Preprocessing

A. UMC005 Corpus

The UMC005 corpus is a parallel English–Urdu dataset containing aligned sentences from religious texts, specifically the Quran and Bible. For this experiment, we use the Quran subset,

which provides 6,000 sentence pairs. The corpus exhibits characteristics typical of religious texts: formal language, archaic expressions, and domain-specific terminology. While this domain specificity limits generalizability to conversational English, it provides a controlled environment for comparing architectures.

B. Preprocessing Pipeline

We apply several preprocessing steps to prepare the data for training. First, we perform basic cleaning: removing zero-width characters (Unicode U+200C, U+200D), normalizing whitespace, and trimming sentences. English sentences are lowercased, while Urdu text remains in its original form since case is not applicable.

We filter sentence pairs based on length constraints: both source and target must contain between 1 and 60 tokens after tokenization. This filtering removes extremely short or long sentences that could cause training instability or memory issues. After filtering, we retain 4,793 sentence pairs.

C. Tokenization and Vocabulary

Tokenization strategies differ for each language. For English, we use NLTK's word tokenizer, which handles punctuation and contractions appropriately. For Urdu, we employ simple whitespace tokenization, splitting on spaces. While subword tokenization (e.g., Byte Pair Encoding) would better handle out-of-vocabulary words and reduce vocabulary size, we use word-level tokenization for simplicity and to match the baseline implementation.

We build separate vocabularies for source (English) and target (Urdu) languages. Each vocabulary includes special tokens: <pad> for padding, <sos> for start-of-sequence, <eos> for end-of-sequence, and <unk> for unknown words. We set a maximum vocabulary size of 30,000 and a minimum token frequency of 2, meaning tokens appearing only once are replaced with <unk>. The final source vocabulary contains 3,895 tokens, and the target vocabulary contains 3,398 tokens.

D. Data Splitting

We randomly shuffle the 4,793 filtered pairs and split them into training (80%), validation (10%), and test (10%) sets, resulting in 3,834 training examples, 479 validation examples, and 480 test examples. This split ensures sufficient training data while maintaining reasonable validation and test sets for evaluation.

IV. Methodology

A. Seq2Seq LSTM Baseline

1) Architecture: The baseline model consists of three main components: an encoder LSTM, a Bahdanau attention mechanism, and a decoder LSTM. The encoder processes the source sentence using a bidirectional LSTM, producing hidden states at each position. The final encoder hidden state is transformed through a linear layer to initialize the decoder's hidden and cell states.

The attention mechanism computes alignment scores between the decoder's current hidden state and all encoder outputs. These scores are normalized via softmax to produce attention weights, which are used to compute a weighted context vector. This context vector, combined with the current decoder input, is fed into the decoder LSTM to generate the next token.

The decoder operates autoregressively: at each step, it receives the previous token (or ground truth during training via teacher forcing), the previous hidden state, and the attention-weighted context. The output is passed through a linear layer to produce logits over the target vocabulary.

2) Hyperparameters: We set embedding dimensions to 256 for both encoder and decoder. The hidden dimension is 256, and we use a single-layer bidirectional LSTM in the encoder and a single-layer unidirectional LSTM in the decoder. Dropout is set to 0.2 to regularize training. The model is trained using the Adam optimizer with a learning rate of 10^{-3} and gradient clipping at 1.0 to prevent exploding gradients.

B. Transformer MT Model

1) Architecture: The Transformer model follows the standard encoder–decoder architecture. The encoder consists of a stack of identical layers, each containing multi-head self-attention and a position-wise feed-forward network, with residual connections and layer normalization. The decoder has a similar structure but includes an additional cross-attention layer that attends to encoder outputs.

Input embeddings are combined with sinusoidal positional encodings to inject sequence order information. Multi-head attention splits the model dimension into multiple heads, allowing the model to attend to different representation subspaces simultaneously. The feed-forward network applies two linear transformations with ReLU activation, expanding the dimension from d_{model} to d_{ff} and back.

During decoding, we apply two types of masks: a padding mask to ignore padding tokens and a causal mask to prevent attending to future positions. The model generates tokens greedily by selecting the token with the highest probability at each step.

2) Hyperparameters: We set $d_{model} = 256$, the number of attention heads to 4, and the number of encoder/decoder layers to 3. The feed-forward dimension d_{ff} is 512, and dropout is 0.1. We use the Adam optimizer with a learning rate of 10^{-4} and gradient clipping at 1.0. The lower learning rate compared to the LSTM helps stabilize Transformer training.

C. Implementation Details

Both models are implemented in PyTorch and trained on Google Colab with GPU acceleration. We use a batch size of 64 and pad sequences within each batch to the maximum length. The loss function is cross-entropy, and we ignore padding tokens when computing loss. During training, we use teacher forcing: the decoder receives ground truth tokens as input. During inference, we use greedy decoding, selecting the most likely token at each step.

TABLE I
EVALUATION RESULTS COMPARING LSTM AND TRANSFORMER MODELS.

Model	Val Loss	Perplexity	BLEU	ROUGE-L
LSTM	4.6030	99.78	0.42	4.29
Transformer	4.7077	110.80	2.16	6.76

We train both models for 5 epochs, which is sufficient to observe convergence trends but limited by computational constraints. Longer training would likely improve results but requires more time and resources.

V. EXPERIMENTAL SETUP

A. Dataset Configuration

The training set contains 3,834 sentence pairs, the validation set 479 pairs, and the test set 480 pairs. We evaluate on a subset of 50 batches from the test set (approximately 3,200 sentences, depending on batch composition) to balance evaluation thoroughness with computational efficiency.

B. Evaluation Metrics

We report three metrics: BLEU, ROUGE-L, and perplexity. BLEU measures n-gram precision between generated and reference translations, with a brevity penalty for overly short outputs. ROUGE-L computes the longest common subsequence between hypotheses and references, capturing semantic similarity. Perplexity, computed as $\exp(\text{validation loss})$, measures how well the model predicts the validation set, with lower values indicating better performance.

C. Training Configuration

Both models are trained for 5 epochs. The LSTM baseline uses a learning rate of 10^{-3}, while the Transformer uses 10^{-4}. We apply gradient clipping at 1.0 for both models. Training is performed on GPU (CUDA) when available, falling back to CPU otherwise. We monitor training and validation loss after each epoch to track convergence.

VI. RESULTS AND DISCUSSION

A. Quantitative Results

Table I summarizes the evaluation metrics for both models. The Transformer achieves higher BLEU (2.16 vs. 0.42) and ROUGE-L (6.76 vs. 4.29) scores, indicating better alignment with reference translations. However, the LSTM achieves a lower perplexity (99.78 vs. 110.80), suggesting it may be more calibrated in its probability estimates, though this could also reflect overfitting or differences in loss scaling.

The Transformer's superior BLEU and ROUGE scores align with expectations, as self-attention mechanisms better capture long-range dependencies and parallel processing enables more effective learning. However, the absolute scores remain low, reflecting limitations discussed below.

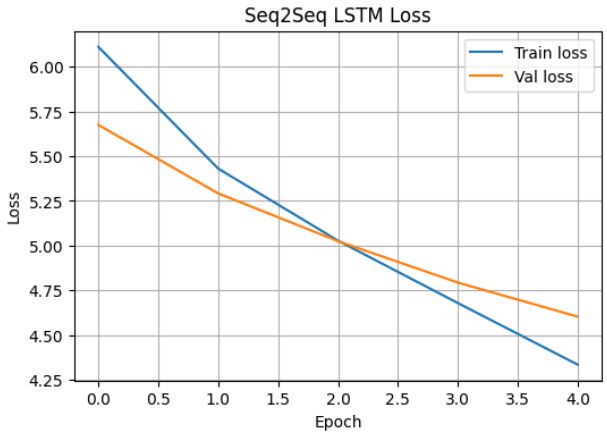

Fig. 1. Training and validation loss for the Seq2Seq LSTM baseline.

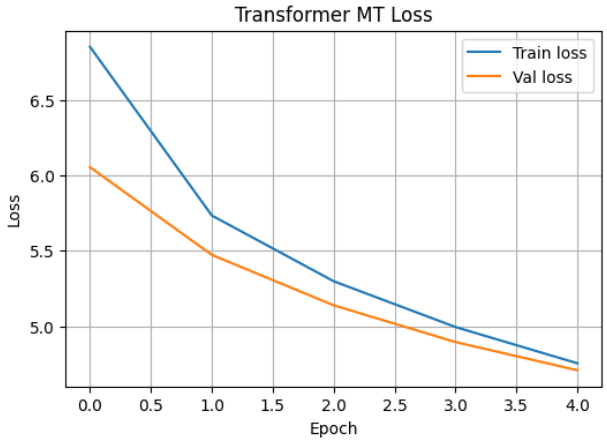

Fig. 2. Training and validation loss for the Transformer MT model.

B. Loss Curves

Figures 1 and 2 show training and validation loss curves for both models. The LSTM exhibits steady convergence, with training loss decreasing from 6.11 to 4.34 and validation loss from 5.67 to 4.60 over 5 epochs. The Transformer shows similar trends, with training loss decreasing from 6.85 to 4.75 and validation loss from 6.06 to 4.71. Both models show signs of continued improvement potential, suggesting that more epochs would be beneficial.

C. Qualitative Analysis

Qualitative inspection of generated translations reveals several issues. Both models produce repetitive outputs, sometimes generating the same token multiple times or getting stuck in loops. The translations often lack grammatical correctness in Urdu, and many words are replaced with `<unk>` tokens due to limited vocabulary coverage. The models struggle with domain mismatch: when tested on conversational English (e.g., "How are you today?"), outputs are poor because training data consists of religious text with different vocabulary and style.

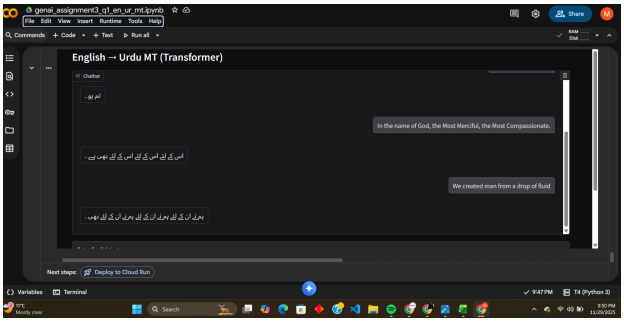

Fig. 3. Gradio-based chat-style UI for English–Urdu translation.

The Transformer's attention visualization (implemented but not shown in figures) reveals that the model learns reasonable alignments between source and target tokens. For example, when generating Urdu tokens, the model attends to semantically related English words, though the attention patterns are sometimes diffuse rather than sharply focused.

D. Limitations and Challenges

Several factors contribute to the limited translation quality. First, the model architectures are relatively small: the LSTM uses 256-dimensional embeddings and hidden states, and the Transformer uses $d_{model} = 256$ with only 3 layers. Larger models with more parameters would likely improve performance.

Second, training is limited to 5 epochs due to computational constraints. Modern NMT systems typically train for dozens or hundreds of epochs, allowing models to better learn language patterns.

Third, word-level tokenization creates large vocabularies and frequent `<unk>` tokens. Subword tokenization (e.g., BPE, SentencePiece) would reduce vocabulary size, handle rare words better, and improve generalization.

Fourth, greedy decoding limits exploration of the output space. Beam search, which maintains multiple hypotheses, would likely produce better translations.

Fifth, the UMC005 corpus is domain-specific (religious texts) and relatively small. Models trained on this data struggle with general conversational English, highlighting the need for domain adaptation or larger, more diverse corpora.

E. Interactive Translation Interface

We develop a Gradio-based chat interface that allows users to input English text and receive Urdu translations in real time. Figure 3 shows a screenshot of the interface. The system uses the trained Transformer model for inference, providing an accessible way to test translation quality interactively. Users can input sentences, view translations, and clear the conversation history. This interface demonstrates practical deployment of the model, though the translation quality limitations discussed above are evident in user interactions.

VII. Conclusion and Future Work

This work implements and compares LSTM and Transformer architectures for English–Urdu neural machine translation on the UMC005 corpus. The Transformer achieves better BLEU and ROUGE scores, demonstrating the effectiveness of self-attention mechanisms for this task. However, both models exhibit limited translation quality due to small architectures, limited training, word-level tokenization, and domain-specific data.

Future work should explore several directions. First, scaling up model size (e.g., $d_{model} = 512$ or 1024, more layers) would likely improve performance. Second, implementing subword tokenization (BPE or SentencePiece) would better handle vocabulary and reduce unknown tokens. Third, training for more epochs with learning rate scheduling would allow models to converge further. Fourth, beam search decoding would explore the output space more effectively than greedy decoding.

Fifth, leveraging pretrained models (e.g., Helsinki-NLP's opus-mt-en-ur) and fine-tuning on the UMC005 corpus could provide a strong baseline with minimal training. Sixth, domain adaptation techniques could help bridge the gap between religious text training data and conversational English test cases. Finally, collecting or incorporating larger parallel corpora would provide more diverse training examples.

Despite current limitations, this work provides a foundation for understanding NMT challenges in low-resource settings and demonstrates the practical implementation of modern architectures for a linguistically diverse language pair.

References

[1] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. ICLR*, 2015.
[2] A. Vaswani et al., "Attention is all you need," in *Proc. NeurIPS*, 2017, pp. 5998–6008.
[3] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. NeurIPS*, 2014, pp. 3104–3112.
[4] K. Cho et al., "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. EMNLP*, 2014, pp. 1724–1734.