Comparative Study of Encoder-Decoder Architectures with Attention Mechanisms for Praphrase Generation

Group Members
Jayesh Deshmukh - 202201040203
Kaustubh Wagh - 202201070021
Alvin Abraham - 202201070132

Project Summary

Problem Statement : Implementation and comparison of neural architectures for Pharaphrase Generation, requires complex contextual understanding between the question and the passage.

Aim and Objectives : The project aims to implement and compare different neural architectures for Pharaphrase Generation.
Specifically, the objectives include implementing and evaluating:

- Encoder-Decoder without Attention (LSTM/GRU)
- Encoder-Decoder with Bahdanau Attention
- Transformer with Self-Attention

Methodology

- **Dataset Preparation**

  A parallel corpus of sentence pairs was preprocessed by tokenization, lowercasing, and padding to ensure uniform input length for all models.

- **Model Architectures**
- **Without Attention:** Basic encoder-decoder using LSTM/GRU layers.
- **With Attention**: Enhanced with Bahdanau and Luong attention mechanisms to focus on relevant input parts during decoding.
- **Transformer:** Utilized multi-head self-attention and positional encoding for parallelized sequence learning.

- **Training and Optimization**

  All models were trained using categorical cross-entropy loss with Adam optimizer. Early stopping and validation-based checkpointing were applied.

- **Inference Strategy**

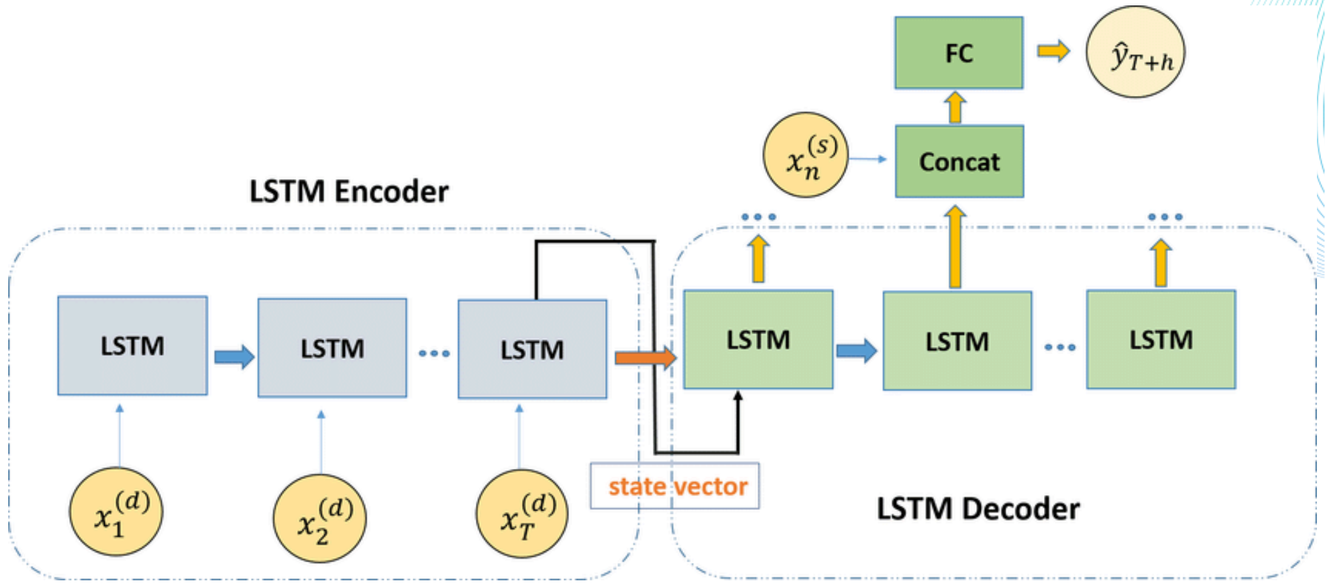
  Greedy decoding was used to generate paraphrases from test sentences, with start and end tokens guiding the sequence.

- **Evaluation Metrics**

  BLEU, ROUGE, METEOR, and qualitative assessment of syntactic variation and semantic preservation were used for performance comparison.

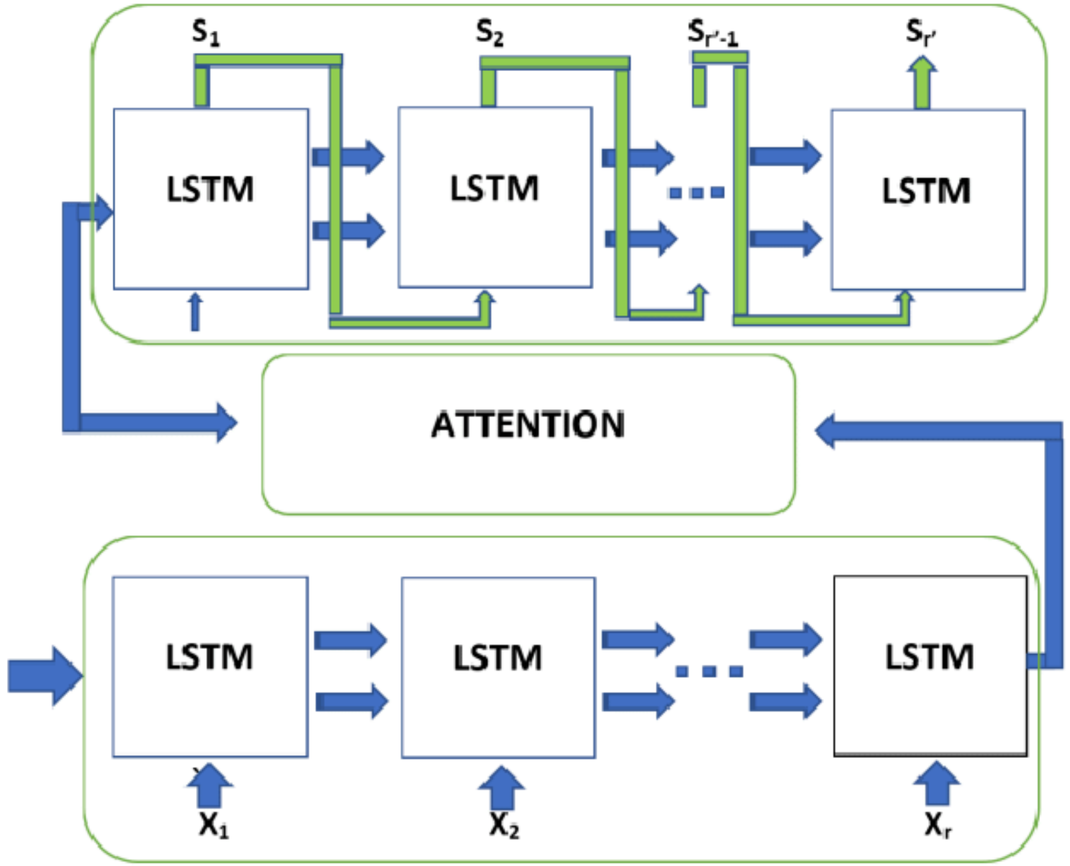
Basic Encoder-Decoder Without Attention

- Encoder processes the input sequence: The encoder takes a variable-length input sequence (e.g., a sentence) and processes it using an RNN, LSTM, or GRU to produce a fixed-size context vector (also called the hidden state).Context vector captures input meaning.
- The final hidden state of the encoder summarizes the entire input sequence — this acts as a compressed representation of the input's meaning. Decoder generates output sequence.
- The decoder receives the context vector and starts generating the target sequence one token at a time, predicting each word based on the context and previously generated tokens.



With Attention (Bahdanau)

- Encoder processes the input sequence - The encoder (typically a bi-directional RNN/LSTM/GRU) reads the input sequence and produces a sequence of hidden states — one for each input token (not just a single context vector).
- Attention computes alignment scores - Instead of compressing all input into one fixed vector, the attention mechanism (Bahdanau or Luong) computes alignment scores between the decoder's current hidden state and each encoder hidden state, to decide which input parts to focus on.
- Context vector is dynamically generated - A context vector is computed as a weighted sum of all encoder hidden states, where weights come from the attention scores. This vector changes at each decoding step.Decoder uses context + previous outputs to predict next token:
- At every step, the decoder receives the context vector and the previous hidden state (and output token) to predict the next word in the target sequence.



Transformer (Self Attention)

- **Encoder uses self-attention to process the input sequence:**

Each input token attends to all others via multi-head self-attention. This helps the encoder build contextual representations of each token by considering the entire input sequence.

- **Decoder uses masked self-attention:**

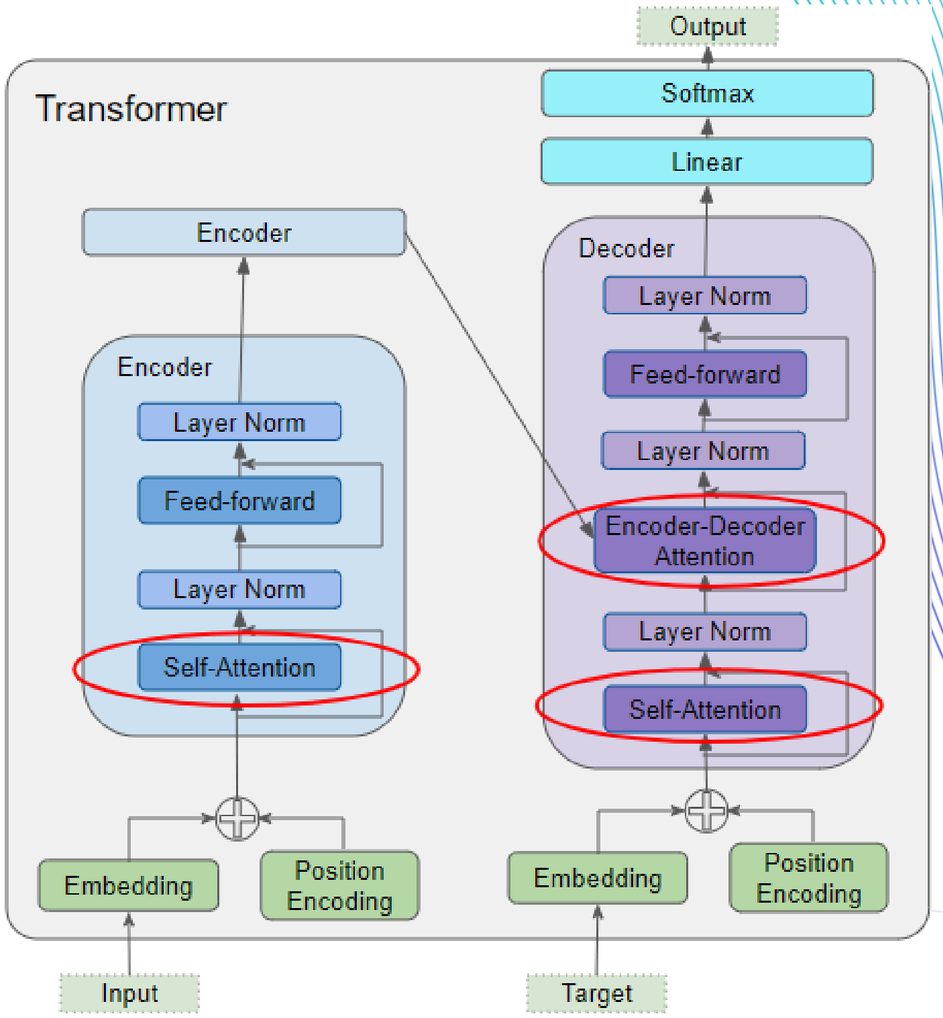
The decoder also uses self-attention, but with masking to prevent attending to future tokens (during training), ensuring autoregressive generation.

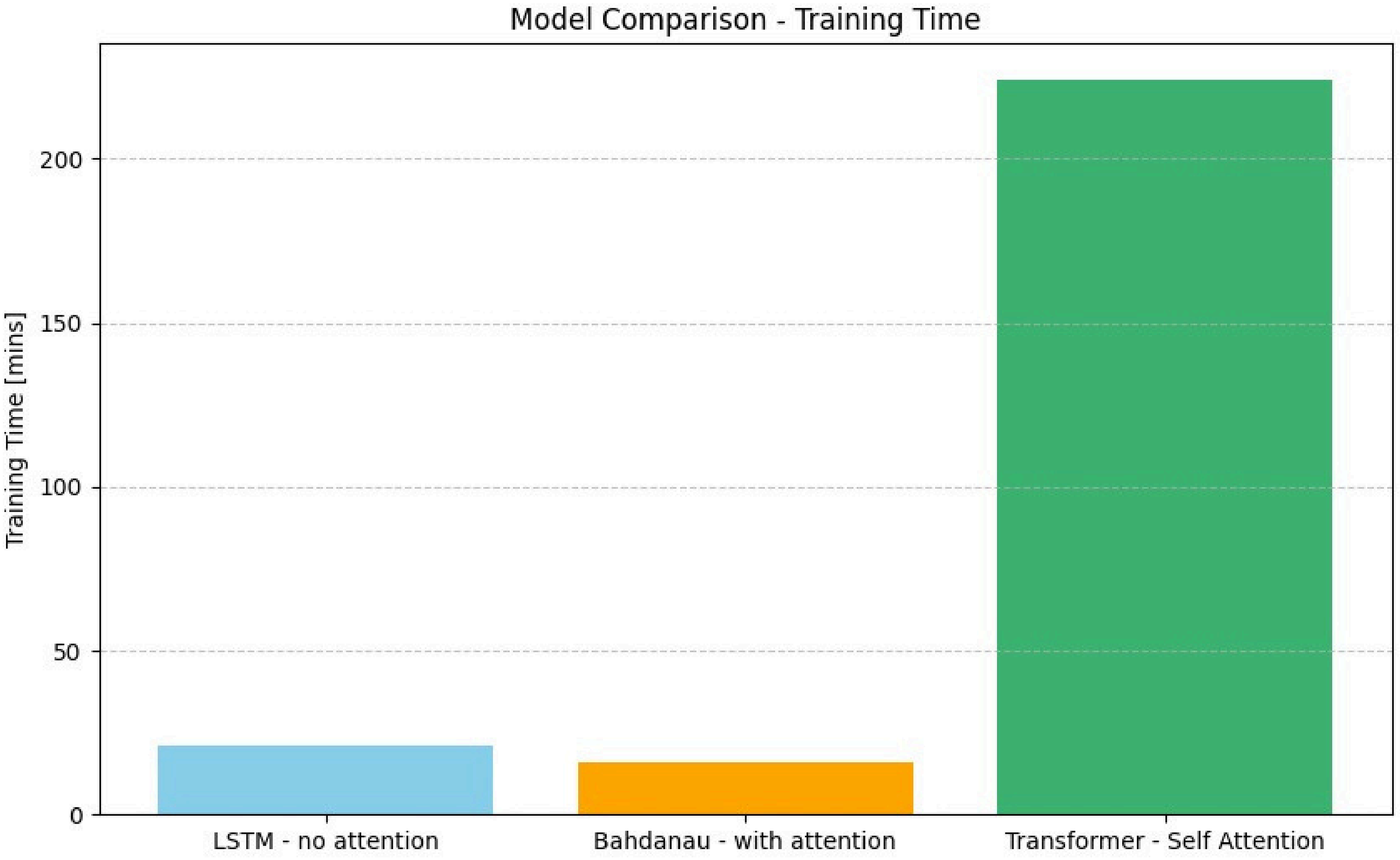
- **Encoder-decoder attention links the two:**

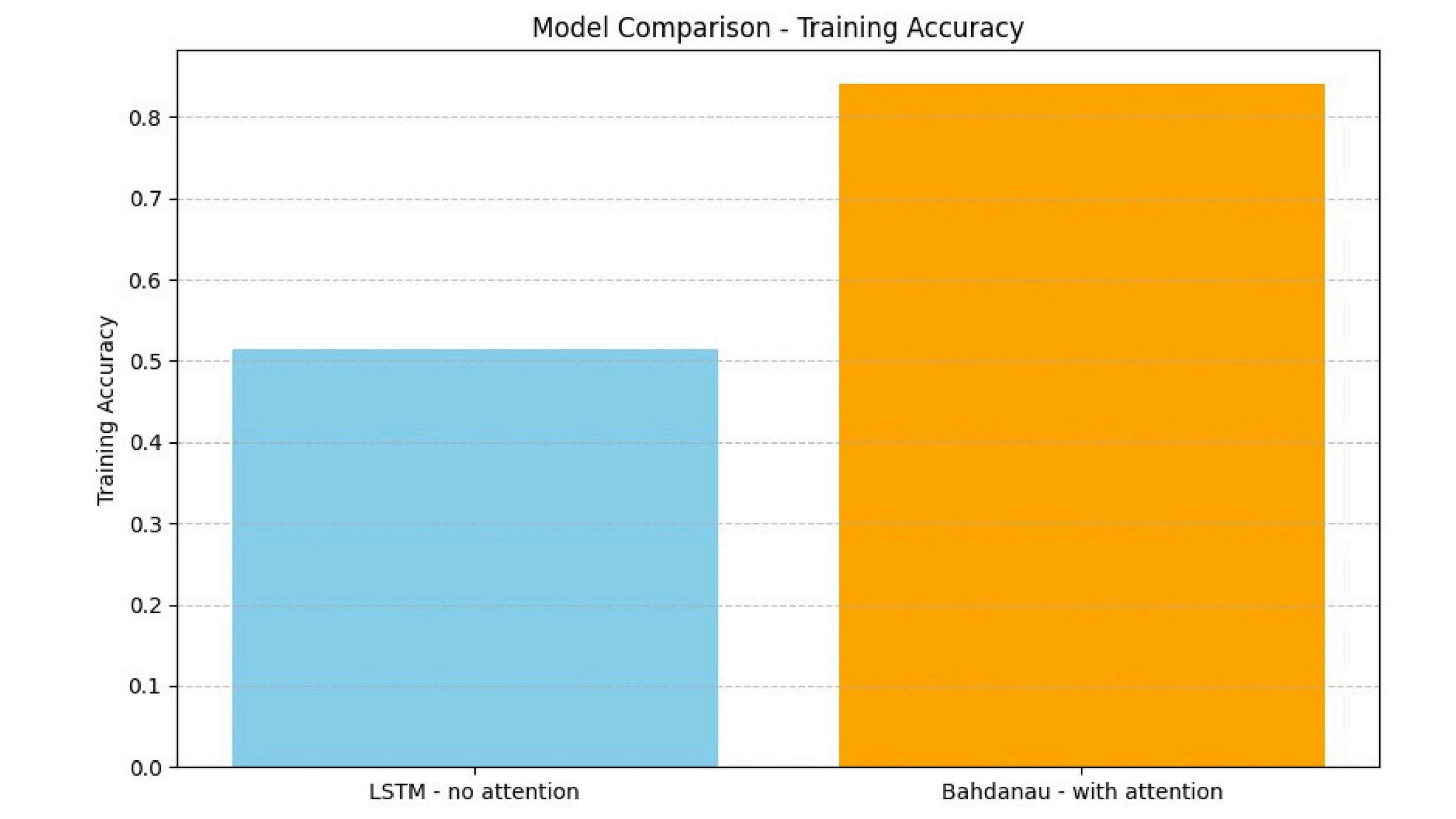
After masked self-attention, the decoder uses encoder-decoder attention. Here, the decoder attends to the encoder's output representations to incorporate source information when generating each output token.

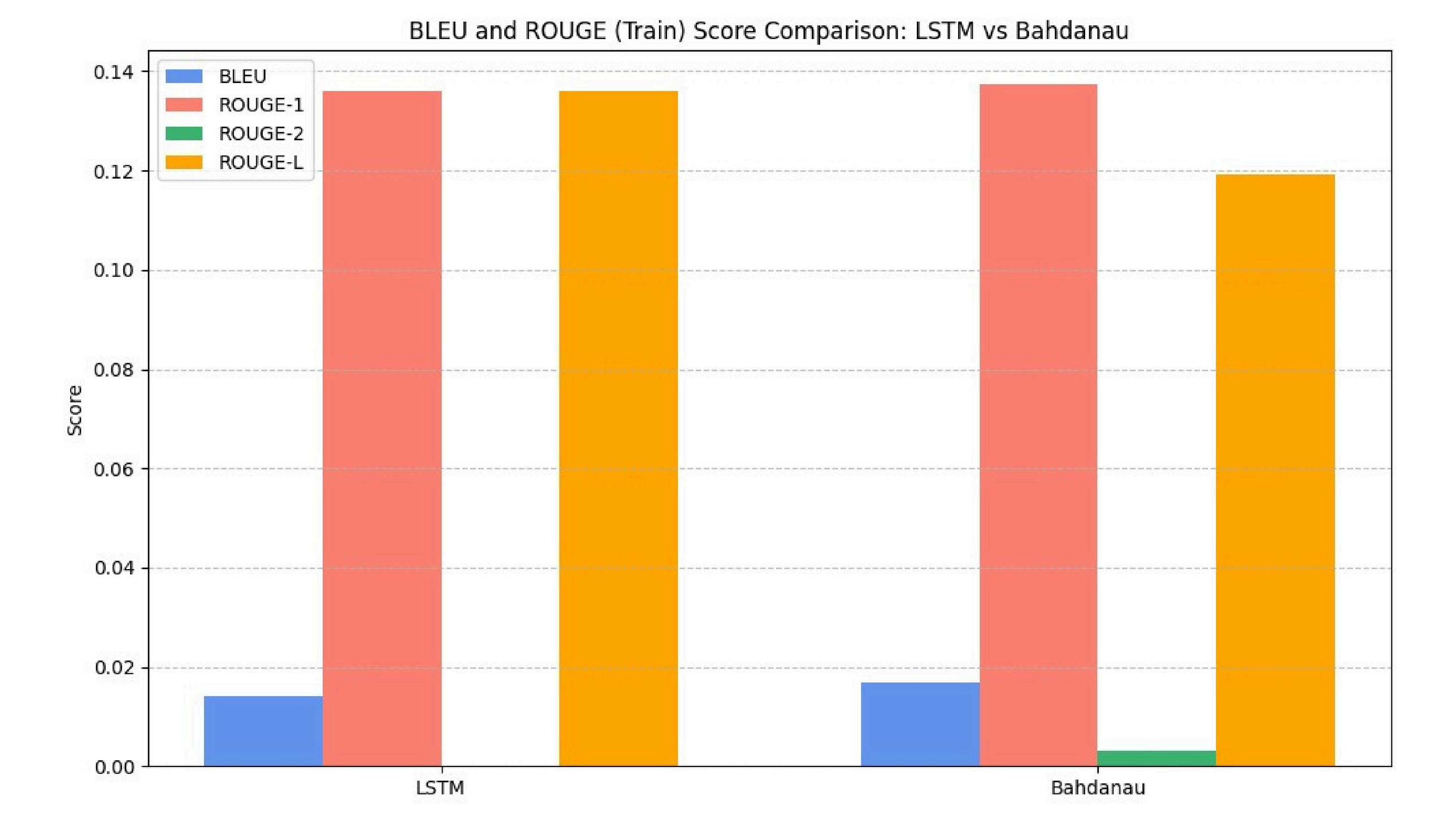
- **Final linear + softmax layer produces predictions:**

The output of the decoder is passed through a linear layer followed by softmax to generate the next word in the sequence.




Model Comparison - Training Time
Training Time [mins]
200
150
100
50
0
LSTM - no attention
Bahdanau - with attention
Transformer - Self Attention


Model Comparison - Training Accuracy
Training Accuracy
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
LSTM - no attention
Bahdanau - with attention


BLEU and ROUGE (Train) Score Comparison: LSTM vs Bahdanau
BLEU
ROUGE-1
ROUGE-2
ROUGE-L
Score
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
LSTM
Bahdanau

Final Analysis Table

Comparison Table

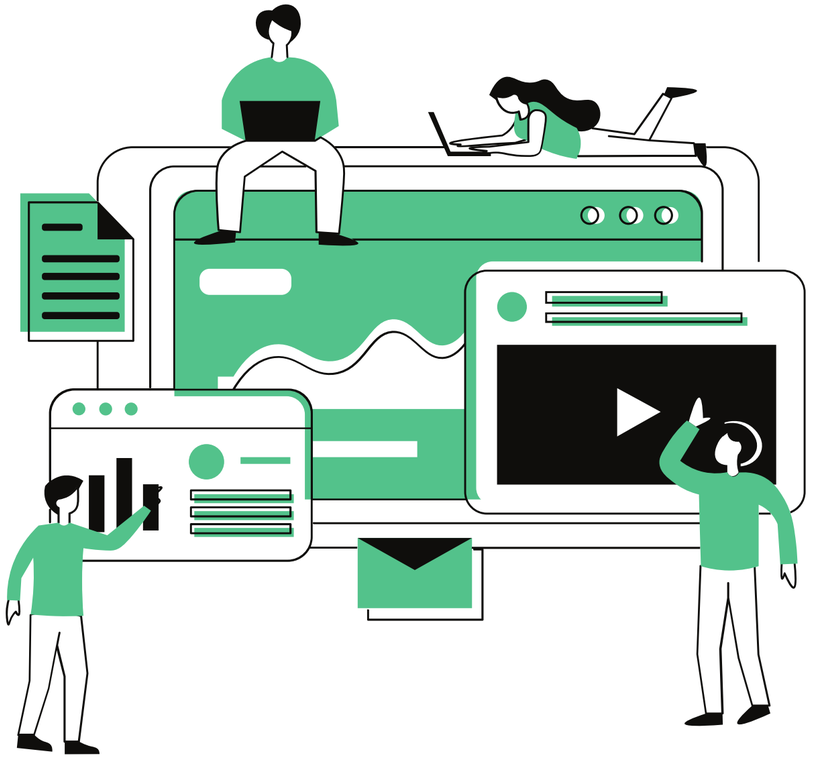
Criteria	LSTM/GRU (No Attention)	Attention (Bahdanau)	Transformer (Self-Attention)
Model Complexity	Low (263,810 parameters)	Medium (272,066 parameters)	High (1,821,954 parameters)
Memory Usage	Low (1.0x)	Medium (1.2x)	High (6.7x)
Interpretability	Low	✓ High (Attention Maps)	✓ Medium (Attention Heads)
Parallelization	Poor (sequential)	Poor (sequential)	Excellent (parallel)
Scalability	Low	Medium	High
Ease of Implementation	High	Medium	Low
Convergence Speed	Medium	Medium-High	High
Context Length Handling	Poor	Medium	Good

Conclusion

- The comparative study of the three encoder-decoder models—without attention, with attention, and Transformer—highlights the evolving effectiveness of neural architectures in paraphrase generation.

- The basic model performs adequately but lacks contextual sensitivity. Adding attention significantly improves sequence alignment and meaning retention, while the Transformer surpasses both in fluency and accuracy due to its parallelized self-attention mechanism.

- Overall, as architectural complexity increases, so does the model's ability to generate more coherent and contextually accurate paraphrases.



Thank You