TEXT-TO-PYTHON CODE GENERATION USING SEQ2SEQ MODELS EXPERIMENTAL RESULTS AND ANALYSIS

ABSTRACT

This report presents a comprehensive analysis of three sequence-to-sequence (Seq2Seq) neural network architectures for automatic Python code generation from natural language docstrings. We implemented and evaluated:

1. Vanilla RNN-based Seq2Seq
2. LSTM-based Seq2Seq
3. LSTM with Bahdanau Attention mechanism

Our experiments on the CodeSearchNet Python dataset demonstrate the effectiveness of attention mechanisms in improving code generation quality, while revealing significant challenges in producing syntactically correct code.

Key Results:
- Attention model achieved best token-level accuracy: 15.85%
- Vanilla RNN achieved highest BLEU score: 18.04
- All models struggled with syntax validity (0% AST parse rate)
- Zero exact match accuracy across all architectures

1. INTRODUCTION

Automatic code generation from natural language descriptions is a challenging task that requires understanding semantic intent, preserving long-range dependencies, and producing syntactically correct structured output. This work explores how different recurrent neural network architectures perform on the text-to-code generation task, specifically translating function docstrings to Python source code.

1.1 Objectives

- Understand limitations of vanilla RNNs in modeling long sequences
- Observe how LSTMs improve long-term dependency handling
- Learn how attention mechanisms overcome the fixed-length context bottleneck
- Analyze generated code using quantitative and qualitative metrics

2. DATASET

2.1 CodeSearchNet Python Dataset

We utilized the CodeSearchNet Python dataset (Nan-Do/code-search-net-python) from Hugging Face, which contains pairs of English docstrings and corresponding Python functions extracted from real-world GitHub repositories.

2.2 Dataset Configuration

Training examples: 8,000 pairs (within 5,000-10,000 range)
Max source length (docstring): 50 tokens
Max target length (code): 80 tokens
Vocabulary size: 30,000 tokens
Splits: Automatic train/val/test using partition column

The dataset was preprocessed using a custom tokenizer that handles Python operators, punctuation, and whitespace appropriately. Sequences exceeding the maximum length was truncated to ensure feasible training times.

3. MODEL ARCHITECTURES

3.1 Model 1: Vanilla RNN Seq2Seq

Architecture:

- Encoder: RNN with embedding dimension 256
- Decoder: RNN with hidden dimension 256
- Fixed-length context vector (encoder final hidden state)
- No attention mechanism

Goal: Establish a baseline and observe performance degradation for longer sequences.

3.2 Model 2: LSTM Seq2Seq

Architecture:

- Encoder: LSTM with embedding dimension 256
- Decoder: LSTM with hidden dimension 256
- Fixed-length context vector (encoder final hidden and cell states)
- No attention mechanism

Goal: Improve modeling of long-range dependencies compared to vanilla RNN.

3.3 Model 3: LSTM with Bahdanau Attention

Architecture:

- Encoder: Bidirectional LSTM ($2 \times 256 = 512$ dimensional outputs)
- Decoder: LSTM with hidden dimension 256
- Bahdanau (additive) attention mechanism
- Bridge network to map bidirectional encoder states to decoder initial state

Goal: Remove the fixed-context bottleneck and improve code generation quality through dynamic attention over encoder outputs.

4. TRAINING CONFIGURATION

All models were trained with identical hyperparameters to ensure fair comparison:

Parameter	Value
Embedding dimension	256
Hidden dimension	256
Number of layers	1
Dropout	0.1
Batch size	64
Epochs	12
Learning rate	3×10^{-4}
Optimizer	Adam
Loss function	Cross-entropy
Teacher forcing ratio	0.5
Random seed	42

5. EVALUATION METRICS

We evaluated all models using the following metrics:

- Token-level Accuracy: Percentage of correctly predicted tokens
- BLEU Score: N-gram overlap between generated and reference code
- Exact Match: Percentage of completely correct outputs
- AST Parse Rate: Percentage of syntactically valid Python code

6. EXPERIMENTAL RESULTS

6.1 Overall Performance Comparison

Model	Token Accuracy	BLEU	Exact Match	AST Parse
Vanilla RNN	15.34%	18.04	0.00%	0.00%
LSTM	11.96%	5.37	0.00%	0.00%
LSTM+ Attention	15.85%	10.69	0.00%	0.00%