## PROJECT OVERVIEW

This project implements and compares three neural sequence-to-sequence (Seq2Seq) architectures in PyTorch that learn to generate Python code from natural language docstring descriptions. It demonstrates the progression from a basic RNN baseline to an attention-enhanced LSTM, showing how each architectural improvement affects code generation quality.

## DATASET

| Field | Details |
|---|---|
| Source | CodeSearchNet - Python (Hugging Face: Nan-Do/code-search-net-python) |
| Splits | 10,000 training, 1,500 validation, 1,500 testing |
| Token Limits | Max 50 source tokens (docstring), Max 80 target tokens (code) |

## FOLDER STRUCTURE

```
Final_submission/
├── src/                    Shared Python modules
│   ├── __init__.py         Package marker
│   ├── config.py           All hyperparameters, paths, constants
│   ├── data.py             Dataset loading, tokenization, vocabulary
│   ├── models.py           All three model architectures
│   ├── train_utils.py      Training and validation loops
│   └── eval_utils.py       BLEU, accuracy, generation, error analysis
├── Notebook/               Jupyter training notebooks (one per model)
├── Checkpoints/            Saved model weights (.pt files)
├── Attention_Visualizations/   Attention heatmap analysis
├── Evaluation/             Model comparison and results
├── Combined.ipynb          All models in one notebook
├── main.py                 Full pipeline orchestrator (CLI)
├── Dockerfile              Docker image definition
├── run.sh / run.bat        Linux / Windows run scripts
├── requirements.txt        Python dependencies
└── README.md               Full project documentation
```

## MODEL ARCHITECTURES

| MODEL 1: Vanilla RNN Seq2Seq (Baseline) | |
|---|---|
| ENCODER | Standard RNN - 2 layers |
| DECODER | Standard RNN - 2 layers |
| ATTENTION | None, fixed-length context vector |
| PURPOSE | Establishes baseline; expected to degrade on longer inputs due to the information bottleneck of compressing the entire input into a single vector. |

| MODEL 2: LSTM Seq2Seq | |
|---|---|
| ENCODER | LSTM - 2 layers |
| DECODER | LSTM - 2 layers |
| ATTENTION | None, still uses a fixed-length context vector |
| PURPOSE | Demonstrates improvement from LSTM gating (forget, input, output gates) for better long-range dependency modeling and gradient flow via the cell state highway. |

| MODEL 3 LSTM with Bahdanau Attention | |
|---|---|
| ENCODER | Bidirectional LSTM - 2 layers |
| DECODER | LSTM - 2 layers + Bahdanau (additive) attention |
| ATTENTION | Dynamic context computed at each decoding step |
| PURPOSE | Removes the information bottleneck by allowing the decoder to attend to all encoder hidden states. Enables interpretability through attention weight visualization (heatmaps). |

## TRAINING CONFIGURATION

| Hyperparameter | Value |
|---|---|
| Embedding Dimension | 256 |
| Hidden Dimension | 256 |
| Number of Layers | 2 |
| Dropout | 0.3 |
| Optimizer | Adam (learning rate = 0.001) |

| | |
|---|---|
| Loss Function | Cross-entropy (PAD token ignored) |
| Teacher Forcing Ratio | 0.5 |
| Gradient Clipping | 1.0 |
| Batch Size | 64 |
| Epochs | 15 |
| Random Seed | 42 |

## TOKENIZATION & VOCABULARY

- **Method:** Whitespace + regex tokenization
- **Structural Tokens:** NEWLINE and INDENT replace \n and \t in code
- **Frequency Threshold:** Words appearing fewer than 2 times become <UNK>
- **Vocabulary Source:** Built on the training set only

◆ **Special Token Indices:**

| Index | Token | Role |
|---|---|---|
| 0 | <PAD> | Padding - fills sequences to uniform length |
| 1 | <SOS> | Start of Sequence - decoder initialization token |
| 2 | <EOS> | End of Sequence - signals completion of output |
| 3 | <UNK> | Unknown / rare token replacement |

## EVALUATION METRICS

| No | Metric | Description |
|---|---|---|
| 1 | Training / Validation Loss | Cross-entropy loss to monitor model convergence |
| 2 | Token Accuracy | Percentage of correctly predicted tokens |
| 3 | BLEU Score | N-gram overlap (1- 4 grams) with smoothing and brevity penalty |
| 4 | Exact Match Accuracy | Percentage of outputs that perfectly match the reference |
| 6 | BLEU vs. Docstring Length | Performance degradation analysis with longer input descriptions |

## PIPELINE OUTPUTS

After running the pipeline, the following files are generated:

◆ **Checkpoints/**

- Vanilla_RNN_best.pt: Best model weights for the Vanilla RNN
- LSTM_best.pt: Best model weights for the LSTM
- LSTM_Attention_best.pt: Best model weights for LSTM + Attention
- Transformer_best.pt: Serialized source and target vocabulary objects

◆ **Attention_Visualizations/**

- example_1.png / _2.png / _3.png: Heatmaps for test examples

◆ **Result:**

| Model | BLEU | Token acc | Exact Match |
|---|---|---|---|
| Vanilla_RNN | 0.0816 | 0.1542 | 0.0000 |
| LSTM | 0.0609 | 0.1588 | 0.0000 |
| LSTM + Attention | 0.0860 | 0.1582 | 0.0000 |
| Transformer | 0.1469 | 0.0919 | 0.0000 |

## BONUS WORK

**[1] Syntax Validation using Python AST**

Used Python's built-in ast.parse() to validate whether generated code is syntactically correct. Provides a stricter evaluation metric beyond BLEU and token accuracy, measuring how often the model produces code that can actually be parsed and executed.

**[2] Extend to Longer Docstrings**

The current pipeline truncates docstrings at 50 tokens. Increasing this limit (like 100-200 tokens) would test the models on more complex, multi-sentence descriptions and further highlight the advantage of the attention mechanism over fixed-context architectures.

**[3] Compare with a Transformer-based Model**

Added a fourth model using the Transformer architecture (self-attention encoder-decoder) to benchmark the RNN/LSTM models against the current state-of-the-art. This demonstrates the performance gap and motivates the shift toward attention-only models used in modern code generation systems.

Github Link: https://github.com/Trina-SE/Text_to_Python