**Milestone 1 – Code Explainer Project**

**Objective:**

The goal of Milestone 1 is to design a pipeline that parses code snippets, extracts structural features, tokenizes code, and forwards the processed data to pretrained NLP models (MiniLM, DistilRoBERTa, MPNet). The outputs and embeddings are compared to analyze representational differences.

**Methodology:**

1. Prepared 10+ code snippets in Python covering functions, classes, and imports.

2. Parsed code using Abstract Syntax Tree (AST) to extract functions, classes, imports, and code patterns.

3. Tokenized code with Hugging Face tokenizer for each pretrained model.

4. Forwarded code tokens to MiniLM, DistilRoBERTa, and MPNet to generate embeddings.

5. Compared embeddings using cosine similarity and visualized differences via dimensionality reduction (t-SNE/PCA).

**Results:**

- MiniLM produced compact embeddings with strong contextual capture.

- DistilRoBERTa showed higher sensitivity to variable naming.

- MPNet balanced semantic and syntactic understanding effectively.

- Visualizations highlighted clustering of similar code patterns.

**Observations:**

* AST parsing ensures structural clarity in code before model input.-
* Pretrained NLP models differ in their treatment of syntax vs. semantics.
* Embedding comparisons provide insights for future fine-tuning on code-specific datasets.
* MPNet embeddings demonstrated the most stable clustering across snippets

**Conclusion:**

The Code Explainer pipeline successfully integrates AST parsing, tokenization, and embedding generation using multiple pretrained NLP models. Comparative analysis shows that each model offers unique strengths, with MPNet proving most consistent for code representation tasks. This milestone lays the foundation for building robust AI-based code explanation and generation systems.  
  
  
  
  
  
  
Detailed Explanation   
  
**Infosys\_Springboard — Milestone 1: Code Explainer**

**Project Overview**

This repository contains the implementation for Milestone 1 of the Infosys CodeGenie AI Explainer and Code Generator Internship Project. The objective of this milestone is to develop a Code Explainer pipeline that parses, analyzes, and represents Python code snippets using Natural Language Processing (NLP) and Transformer-based models for semantic understanding.

**Objectives**

1. Parse at least 10 Python code snippets to extract key structural elements.
2. Perform AST (Abstract Syntax Tree) parsing to identify functions, classes, imports, and syntax patterns.
3. Tokenize and analyze the snippets for keywords, operators, and token distribution.
4. Generate embeddings using three pretrained NLP models:

-> MiniLM

-> DistilRoBERTa

-> MPNet

1. Compare model embeddings to study semantic differences in code representation.
2. Visualize and summarize all results through charts and tables.

**Pipeline Architecture**

🔹 **Section 1: Importing Dependencies**

Imported all required libraries for:

-> Text preprocessing → re, tokenize, ast

-> NLP & embeddings → sentence\_transformers, transformers

-> Visualization → matplotlib, seaborn

-> Data handling → pandas, numpy

🔹 **Section 2: Dataset Preparation**

Prepared a dataset of 10 Python code snippets, covering:

-> Arithmetic operations

-> Conditional logic

-> Loops and recursion

-> String and list manipulations

-> Class and function definitions

🔹 **Section 3: AST Parsing**

Used Python’s built-in ast module to extract:

-> Function definitions

-> Class structures

-> Import statements

-> Code-level patterns

This step helps in structural understanding before semantic modeling.

🔹 **Section 4: Tokenization & Lexical Analysis**

Generated:

-> Token counts

-> Keyword and operator frequencies

-> Token distribution histogram

-> Line length analysis

These insights provide the syntactic richness of each snippet.

🔹 **Section 5: Model Embeddings**

Used SentenceTransformer to generate embeddings for each snippet via:

-> all-MiniLM-L12-v2

-> all-distilroberta-v1

-> all-mpnet-base-v2

Each model captures code semantics in vector form.

🔹 **Section 6: Embedding Comparison & Visualization**

Performed cosine similarity across snippets and plotted:

-> Token Distribution

-> Line Length Distribution

-> Pairwise Similarity Heatmaps for all three models

These visualizations reveal semantic closeness between code snippets.

🔹 **Section 7: Summary Report**

Created a structured summary table containing:

-> Total Functions, Classes, Imports

-> Token Counts & Uniqueness

-> Keywords & Operators Used

Displayed using pandas DataFrame for easy interpretation.

**Key Visualizations**

-> Token Distribution Plot

-> Line Length Distribution

-> Pairwise Similarity Heatmaps for MiniLM, DistilRoBERTa & MPNet

-> Summary Report Table

Each visualization highlights a different layer of code analysis — from syntax to semantics.

**Observations & Insights**

1. Embedding similarities were consistent across models for functionally related snippets.
2. MPNet provided more stable contextual embeddings compared to MiniLM and DistilRoBERTa.
3. The AST-based structural breakdown enhanced the interpretability of code semantics.
4. The combination of AST parsing + Transformer embeddings gives both syntactic and semantic understanding of code.

**Technologies Used**

Category Tools / Libraries Language Python Notebook Google Colaboratory NLP Models MiniLM, DistilRoBERTa, MPNet Visualization Matplotlib, Seaborn Data Handling Pandas, NumPy Parsing AST, Tokenize Embedding SentenceTransformers

**Future Scope**

1. Integrate CodeT5 or StarCoder for more advanced code understanding.
2. Extend pipeline for code-to-text summarization (AI Explainer).
3. Develop an interactive web app for real-time code explanation.
4. Incorporate attention heatmaps for interpretability analysis.