# AI-PROJECT REPORT

**Project Title:** 

AI CODE EXPLAINER

**Submitted by:** 

**UM-E-HABIBA** 

SU92-BSDSM-F23-040

BSDS-3A

Submitted to:

Prof. RASIKH ALI

**Program:** 

BS DATA SCIENCE

### PROJECT OVERVIEW

#### INTRODUCTION

#### Overview of the AI Code Explainer

The AI Code Explainer is a tool that uses machine learning to generate human-readable explanations for programming code. It simplifies understanding complex code by providing step-by-step descriptions, making it useful for both beginners and experienced developers.

#### **Importance and Motivation**

As programming becomes more complex, understanding code can be challenging. This tool automates the process of explaining code, helping developers and learners quickly grasp its functionality, saving time and improving productivity.

### **Objectives**

### **Simplify Complex Code Explanations**

• Automatically generate clear, step-by-step explanations for complex code, making it easier to understand.

#### **Enhance Learning for Developers**

 Help developers, especially beginners, quickly learn programming concepts by providing automated explanations.

## **Technologies and Tools Used**

## **Programming Language:**

 Python: The primary language used for developing the AI Code Explainer, chosen for its rich ecosystem of libraries and frameworks in machine learning and natural language processing.

#### **Machine Learning Libraries:**

- Transformers: Utilized for loading pre-trained models like GPT-2 to generate code explanations. The library enables easy integration of state-of-the-art natural language models.
- Torch: PyTorch is used for managing model execution, ensuring efficient performance

across CPU and GPU for faster processing.

#### **Development Tools:**

• **Jupyter Notebook**: Used for prototyping, experimenting, and developing the AI Code Explainer, providing an interactive environment for coding and testing the system.

#### **Pip Install Commands:**

• !pip install transformers: Installs the Transformers library, which provides access to pre-trained models and tokenizers.

#### **Libraries Used**

- **transformers:** For working with pre-trained models like GPT-2. It's used to load models and tokenizers.
- from transformers import AutoModelForCausalLM, AutoTokenizer: Imports the model and tokenizer to generate and process text.
- **from transformers import pipeline:** A simple interface for using the models in the transformers library.
- **torch:** The PyTorch library for managing tensors and performing model operations on CPU or GPU.

#### **Basic Functions Used**

#### **Key Libraries:**

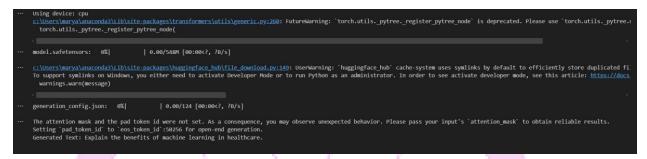
- **Hugging Face Transformers:** Provides pre-trained models and tokenizers for code explanation generation.
- Torch (PyTorch): Manages model execution and tensor operations on CPU/GPU.

from transformers import AutoModelForCausalLM, AutoTokenizer import torch

### **Core Algorithms and Models:**

#### Superior university

- **GPT-2 Model**: A pre-trained language model used to generate natural language explanations of code.
  - o Example: model.generate(...).
- Tokenizers: Converts code into numerical data for the model to process.



## **Implementation**

#### **Backend Logic for Code Explanation:**

• The backend uses the **GPT-2** model from Hugging Face to generate explanations. The input code is tokenized using AutoTokenizer, processed by the model, and then decoded to produce a human-readable explanation.

#### **User Interaction Flow:**

• The user inputs code into the system. The backend processes the code, generates an explanation using the trained model, and returns the result to the user.

```
Using device: cpu
Welcome to the Code Explainer!
Enter your Python code (end with an empty line):
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass your input's 'attention_mask' to obtain reliable results.

Setting 'pad_token_id' to 'eos_token_id':50256 for open-end generation.

Explanation:

Explanation:

Explanation:

Explanation:

Explanation:

The following Python code step by step:

def add_numbers(a, b): return a + b result = add_numbers(5, 10) print(f"The sum is {result}")

Explanation: the above code is a simple one-liner, but it adds a number to a number that can be used to add numbers.

Let's say you want to add a few numbers to a list of strings. You can define the following Python code step:

def add_numbers(a, b): return a + b result = add_numbers(5, 10) print(f"The sum is {result}")

Here, the python code step is executed, and the result is added to the list.

The Python code step is executed, and the result is added to the list.
```

## **Challenges and Solutions**

### **Handling Ambiguous Code Inputs:**

• Challenge: Ambiguous or poorly formatted code may lead to inaccurate or unclear

Superior university

explanations.

• **Solution:** Use context-aware NLP techniques to improve understanding and clarification of ambiguous code.

#### **Managing Resource Limitations During Training:**

- **Challenge**: Training large models like GPT-2 requires significant computational resources.
- **Solution**: Use cloud platforms like Google Colab for free GPU access and optimize model fine-tuning to reduce resource usage.

#### **Machine Learning in the Project:**

 Model: A pre-trained GPT-2 model from Hugging Face is fine-tuned to generate code explanations.

#### Process:

- 1. **Data Tokenization**: Code is tokenized using AutoTokenizer to convert text into a format suitable for the model.
- 2. **Model Inference**: The tokenized code is passed through the GPT-2 model using model.generate() to produce a natural language explanation.
- 3. **Explanation Generation**: The output from the model is decoded back into a human-readable explanation.

#### Conclusion

Developed an AI-powered tool that generates clear, human-readable explanations for complex code, enhancing learning and productivity for developers.