AI Coding Mentor

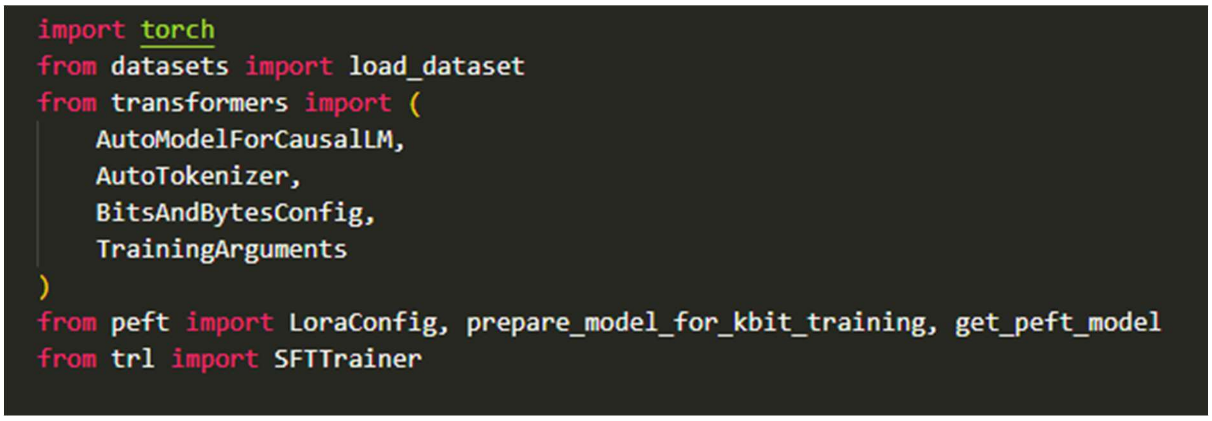
1.Model-Fine tuning

This script demonstrates the process of fine tuning CodeLlama model using the Hugging Face Transformers library. It highlights efficient configurations and techniques for fine-tuning, including the use of 4-bit quantization and LoRA (Low-Rank Adaptation).
This script runs on Google Colab to utilize 16GB VRAM of T4 GPU which significantly reduces the time required to fine tune the model.

1.1 Key Components

1.1.1 Import Statements

The script uses several libraries for dataset preparation, model configuration, and training.



```python
import torch
from datasets import load_dataset
from transformers import (
    AutoModelForCausalLM,
    AutoTokenizer,
    BitsAndBytesConfig,
    TrainingArguments
)
from peft import LoraConfig, prepare_model_for_kbit_training, get_peft_model
from trl import SFTTrainer
```

1.1.2 Dataset Preparation

CodeSearchNet corpus is a dataset of 2 milllion (comment, code) pairs from opensource libraries hosted on GitHub. It contains code and documentation for several programming languages.

The dataset is formatted for training by mapping documentation and code strings into a structured format.

The data is preprocessed by combining function documentation and code snippets into a unified text format for the model to learn meaningful patterns.

https://huggingface.co/datasets/code-search-net/code_search_net

1.1.3 Model and Configuration Details

Here, I use the CodeLlama 7b model and employ 4-bit quantization to reduce memory usage. LoRA is configured to fine-tune only specific layers, which greatly speeds up training.

- Model: CodeLlama 7b model.
- Quantization: Applied 4-bit quantization for efficient training.
- LoRA Configuration: Enables parameter-efficient fine-tuning by focusing on specific trainable components.

1.1.4 Training

The training pipeline uses the SFTTrainer (Supervised Fine-Tuning Trainer) from the Hugging Face library. After fine-tuning, the model is saved for further use.

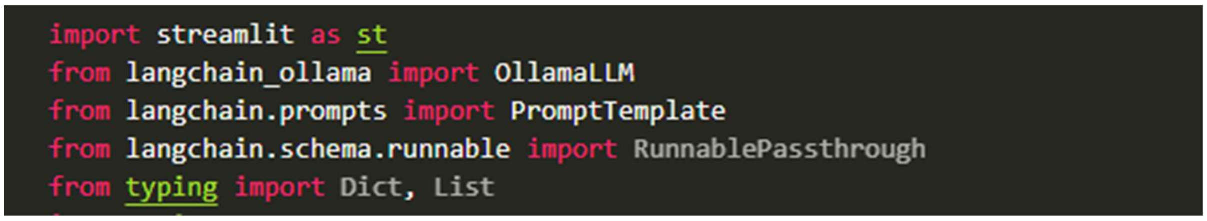
2.Coding Mentor

The CodingMentor program is an AI-powered assistant designed to help users with various coding tasks, such as debugging, code conversion, complexity analysis, and explaining code functionality. It utilizes Streamlit for the user interface and integrates with the langchain_ollama library and CodeLlama for processing.

2.1Key Components

2.1.1 Import Statements

The following libraries are used for building the interface and integrating with CodeLlama:

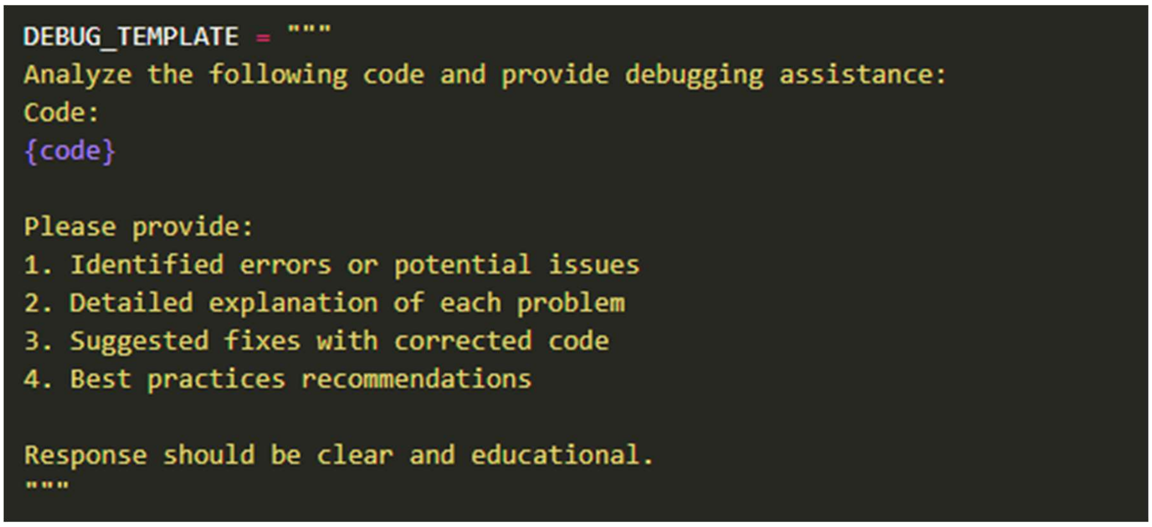


```
import streamlit as st
from langchain_ollama import OllamaLLM
from langchain.prompts import PromptTemplate
from langchain.schema.runnable import RunnablePassthrough
from typing import Dict, List
```

2.1.2Prompt Templates

Prompt templates are predefined instructions provided to the AI model. These templates define how the AI should interpret and respond to user input for specific tasks. Each task has a unique template tailored to its purpose, ensuring clear and consistent interactions.

- Debug Template: Identifies errors and provides solutions.

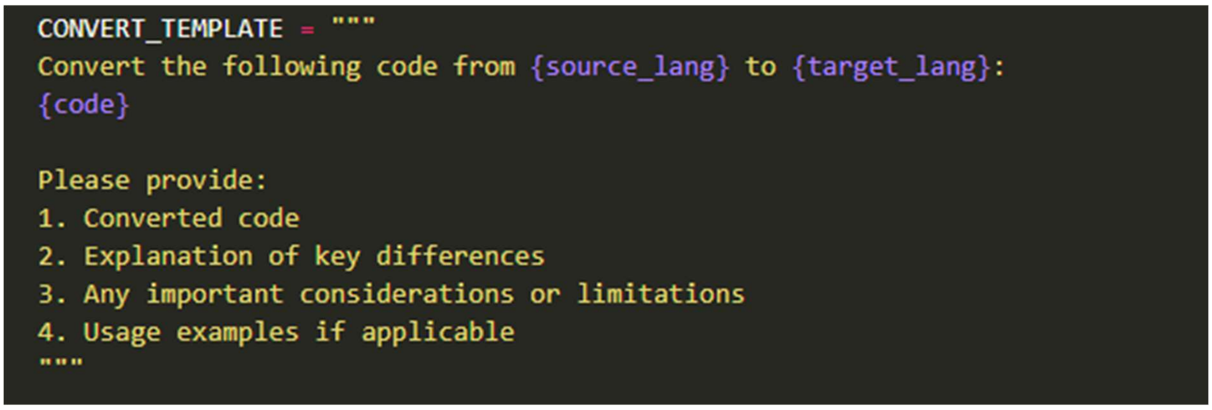


```
DEBUG_TEMPLATE = """
Analyze the following code and provide debugging assistance:
Code:
{code}

Please provide:
1. Identified errors or potential issues
2. Detailed explanation of each problem
3. Suggested fixes with corrected code
4. Best practices recommendations

Response should be clear and educational.
"""
```

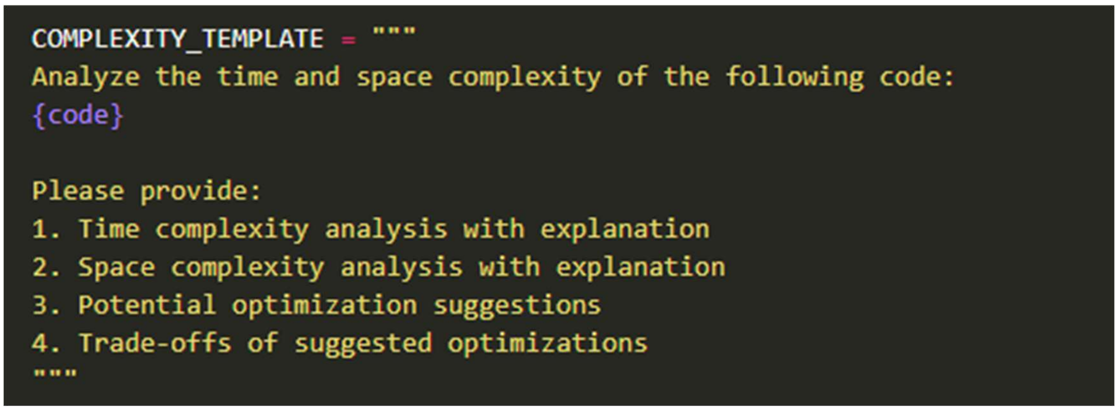
- Convert Template: Converts code between programming languages.



```
CONVERT_TEMPLATE = """
Convert the following code from {source_lang} to {target_lang}:
{code}

Please provide:
1. Converted code
2. Explanation of key differences
3. Any important considerations or limitations
4. Usage examples if applicable
"""
```

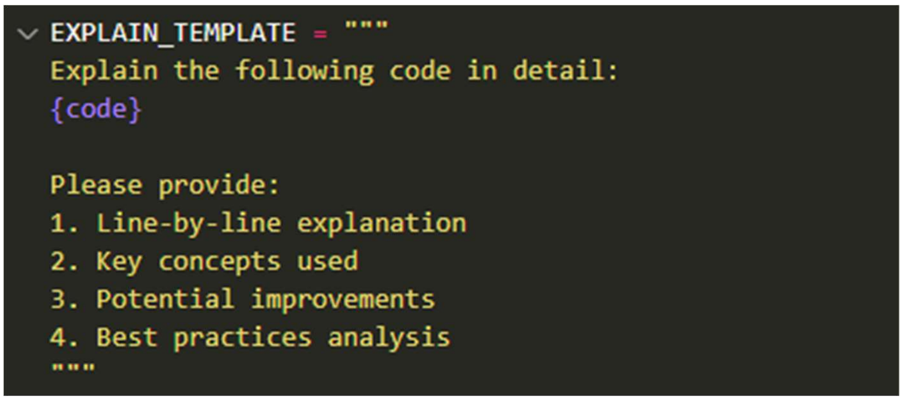
- Complexity Template: Analyzes time and space complexity.



```
COMPLEXITY_TEMPLATE = """
Analyze the time and space complexity of the following code:
{code}

Please provide:
1. Time complexity analysis with explanation
2. Space complexity analysis with explanation
3. Potential optimization suggestions
4. Trade-offs of suggested optimizations
"""
```

- Explain Template: Explains the purpose and functionality of code.



```
EXPLAIN_TEMPLATE = """
Explain the following code in detail:
{code}

Please provide:
1. Line-by-line explanation
2. Key concepts used
3. Potential improvements
4. Best practices analysis
"""
```

2.1.3 The CodingMentor Class

This class encapsulates the core functionalities, initializing different prompt templates and connecting them to the model pipeline.

2.1.4 Streamlit UI

The interactive user interface is built using Streamlit. Users can select tasks, input code, and view results. Users interact with dropdowns and text fields to submit tasks. The application processes inputs and displays results dynamically.