

SMART SDLC- AI ENHANCED SOFTWARE DEVELOPMENT LIFE CYCLE

Team Members

- Abirami G
- Pavithra P
- Shanmugapriya S
- Vadivukkarasi D

Introduction

The AI Code Analysis & Generator project is designed to simplify software development by automating the requirement analysis and code generation process. It is an academic and practical project showcasing the power of Generative AI in software engineering.

Project Description

This project is a Generative AI-powered tool that can analyze software requirement documents and generate relevant code snippets in different programming languages. It leverages the IBM Granite model to perform requirement analysis and automatic code generation. The tool also integrates with Gradio for an interactive web interface.

Features

- Requirement Analysis: Extracts and organizes functional, non-functional, and technical requirements from PDF or text input.
- Code Generation: Generates code in various programming languages including Python, JavaScript, Java, C++, C#, PHP, Go, and Rust.
- Interactive UI: Built with Gradio, allowing easy interaction for uploading documents, analyzing requirements, and generating code.
- Model Integration: Uses Hugging Face Transformers with IBM Granite for advanced natural language understanding and generation.
- PDF Support: Reads and extracts text from uploaded PDFs using PyPDF2.

Technology Stack

- Python
- Gradio

- Hugging Face Transformers
- IBM Granite 3.2 2B Instruct
- PyTorch
- PyPDF2
- ReportLab (for documentation)

System Architecture

The system follows a modular architecture consisting of the following layers:

- Input Layer: Accepts requirements via text or PDF uploads.
- Processing Layer: Extracts data, analyzes requirements, and processes text using IBM Granite.
- Code Generation Layer: Produces code snippets in the requested programming language.
- Presentation Layer: Provides an interactive UI through Gradio for user interaction.

Workflow

1. User uploads a requirements document (PDF) or enters text manually.
2. The system extracts and analyzes the requirements into functional, non-functional, and technical categories.
3. Based on user input, the model generates code in the chosen programming language.
4. Results are displayed interactively in the Gradio interface.

Use Cases

- Academic Projects: Helping students understand requirement analysis and automated coding.
- Software Development: Speeding up requirement documentation and code prototyping.
- Research: Demonstrating the integration of AI in software engineering tasks.
- Training & Learning: Aiding beginners to learn how requirements translate into code.

Limitations

- Limited context understanding in large and complex documents.
- Generated code may need refinement and optimization.
- Dependency on internet for model downloads and updates.
- Currently supports a predefined set of programming languages.

Future Enhancements

- Add support for more programming languages.
- Improve contextual understanding for better requirement-to-code mapping.
- Integrate with IDEs like VS Code for seamless development.
- Enable real-time collaboration features.
- Expand to mobile-friendly applications.


Screenshots

```
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# Load model and tokenizer
model_name = "ibm-granite/granite-3.2-2b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto" if torch.cuda.is_available() else None
)

# Fix padding token
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token
```

```
merges.txt 442k/? [00:00<00:00. 9.00MB/s]
tokenizer.json 3.48M/? [00:00<00:00. 6.59MB/s]
added-tokenizer 100% ██████████ 87.0/87.0 [00:00<00:00. 1.93kB/s]
special-tokens-map.json 100% ██████████ 701/701 [00:00<00:00. 17.3kB/s]
config.json 100% ██████████ 786/786 [00:00<00:00. 23kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json 29.8k/? [00:00<00:00. 2.41MB/s]
Fetching 2 files: 100% ██████████ 2/2 [00:28<00:00. 88.32kB/s]
model-000002-of-000002.safetensors 100% ██████████ 67.3M/67.1M [00:04<00:00. 13.4MB/s]
```



Educational AI Assistant

Concept Explanation

Quiz Generator

Enter a concept

What is machine learning

Generate Quiz

Quiz Questions and Answers

Educational AI Assistant

Concept Explanation

Quiz Generator

Enter a concept

What is mechine learning

Generate Quiz

Quiz Questions and Answers

Quiz Answers:

1. c) Biological

2. True

3. Supervised learning uses labeled data (i.e., input-output pairs) for training, where the correct output is known. The algorithm learns to map inputs to outputs based on this guidance. In contrast, unsupervised learning identifies patterns or structures in unlabeled data. The algorithm must find hidden relationships or groupings on its own, without any pre-existing knowledge of correct outputs.

4. b) It can process and learn from vast amounts of unstructured data, such as images or text. Deep learning is a subset of machine learning that uses artificial neural networks with many layers (hence "deep") to learn and make decisions on data. This architecture allows deep learning models to automatically learn and represent complex features from raw input, such as images, audio, or text, making them highly effective for tasks involving unstructured data.

5. True

Ensemble methods in machine learning combine multiple models to make predictions. This is done by training several models on the same dataset and then combining their outputs. The rationale behind ensemble methods is to leverage the strengths of individual models and mitigate their individual weaknesses, leading to improved overall performance and reduced risk of overfitting. By averaging or voting the predictions of diverse models, ensemble methods provide a more robust and accurate prediction.

Conclusion

The AI Code Analysis & Generator project highlights the importance of Generative AI in modern software engineering. By streamlining requirement analysis and automating code generation, it reduces development time, improves accuracy, and enhances productivity. This tool lays the foundation for future innovations in AI-driven software development workflows.