Project Title

Software Development Life Cycle (SDLC)

Project Documentation

1. Introduction

• Project title : Software Development Life Cycle (SDLC)
• Team member : Harini.M
• Team member : Lubna lutfiyya.I
• Team member : Jothi.E
• Team member : Harini.C

The Software Development Life Cycle (SDLC) is a systematic process for building software that ensures quality and correctness. In this project, AI has been applied to automate two core phases of SDLC: Requirements Analysis and Code Generation. By leveraging Large Language Models (LLMs), our system reads requirements, organizes them into categories, and generates code snippets.

2. Project Overview

Purpose

The purpose of this project is to develop an AI-driven platform that simplifies requirement management, policy summarization, forecasting, and anomaly detection while promoting eco-friendly practices. The system is designed with a conversational interface and multimodal input support, making it easy for both technical and non-technical users to interact with data and generate actionable insights.

Features

- Conversational Interface: AI-powered chatbot for natural language interaction.

- Policy Summarization: Generates concise summaries of lengthy policy documents.
- Resource Forecasting: Predicts demand and utilization of resources.
- Eco-Tip Generator: Provides personalized eco-friendly suggestions.
- Citizen Feedback Loop: Collects feedback and suggestions from users/citizens.
- KPI Forecasting: Forecasts key performance indicators for proactive decision-making.
- Anomaly Detection: Identifies unusual or irregular data patterns.
- Multimodal Input Support: Accepts text, speech, images, and PDF inputs.
- Requirement Analysis: Categorizes requirements into functional, non-functional, and technical specifications.
- Code Generation: Auto-generates code in multiple languages (Python, Java, JavaScript, C++, C#, PHP, Go, Rust).
- PDF Support: Extracts requirements directly from uploaded PDF documents.
- User Interface (Streamlit / Gradio): Clean two-tab interface for easy navigation.

Requirement Analysis

- Functional Requirements:
  - Conversational queries
  - Policy summarization
  - Forecasting (resources, KPIs)
  - Code generation
  - Eco-tip generation

- Non-Functional Requirements:
  - Usability
  - Performance
  - Security
  - Scalability
- Technical Requirements:
  - AI/ML model integration
  - PDF parsing libraries
  - Streamlit/Gradio-based UI
  - Multi-language code support

Code Generation

- Supports multiple languages:
  - Python, Java, JavaScript, C++, C#, PHP, Go, Rust
- Generates boilerplate/sample code based on extracted requirements.

PDF Support

- Direct upload of requirement documents in PDF format.
- Extracts and categorizes requirements automatically.
- Displays results in a structured format.

User Interface

- Tab 1: Requirement Analysis
  - Displays extracted and categorized requirements.
- Tab 2: Project Tools
  - Forecasting dashboards

- Anomaly detection insights
- Eco-tip suggestions
- Code generation options

3. Architecture

The architecture of this project is divided into multiple layers:

Frontend (Gradio):
• An interactive UI with tabs for Requirement Analysis and Code Generation.

Backend (Python + Transformers):
• Handles requirement analysis and code generation using IBM Granite LLM.

LLM Integration (IBM Granite):
• Granite LLM model is used for natural language understanding, text classification, and generating code.

PDF Handling (PyPDF2):
• Extracts and processes text from uploaded requirement documents.

Core Functions:
• generate_response(): Generates responses from the LLM.
• extract_text_from_pdf(): Extracts content from PDFs.
• requirement_analysis(): Analyzes and classifies requirements.
• code_generation(): Generates code in selected programming language.

4. Setup Instructions

Prerequisites:
o Python 3.9 or later
o Libraries: torch, transformers, gradio, PyPDF2
o Internet access for downloading model weights

Installation Process:

1. Install dependencies using pip:

```
pip install torch transformers gradio PyPDF2
```

2. Run the script in Google Colab or local environment.
3. Launch the Gradio app with app.launch(share=True).
4. Access the public link generated by Gradio to use the interface.

5. Folder Structure

app/ – Contains backend logic (analysis, code generation)
ui/ – Contains Gradio interface design
sdlc_app.py – Main program script
models/ – Placeholder for model handling scripts

6. Running the Application

➢ Step 1: Launch the Python script in Colab or locally.

➢ Step 2: Gradio dashboard will open in browser.

➢ Step 3: Navigate between Code Analysis and Code Generation tabs.

➢ Step 4: Upload PDF or input requirements → Results appear as structured categories.

➢ Step 5: Enter requirement description and select programming language → Generated code snippet appears.

➢ Step 6: Copy results for documentation or prototype development.

7. API Documentation

The current version runs via Gradio and does not expose REST APIs.
However, the backend functions can be extended into endpoints such as:

- POST /analyze – Analyze requirements
- POST /generate – Generate code in selected language
- POST /upload – Upload and process PDF documents

8. Authentication

The system is open for demonstration. Future deployments can include:

- Token-based authentication (JWT)

• Role-based access (admin, user)
• OAuth2 authentication
• User session management and history tracking

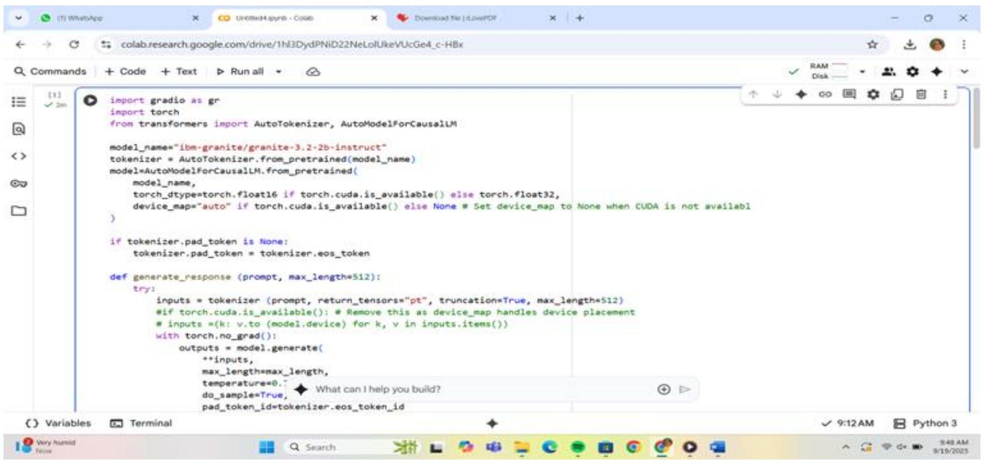
9. User Interface

The user interface is built with Gradio and contains:
• Tab 1: Requirement Analysis – Upload PDF or type requirements, view structured analysis.
• Tab 2: Code Generation – Input requirements, select programming language, generate code.
The design is minimalistic, user-friendly, and suitable for students and professionals.

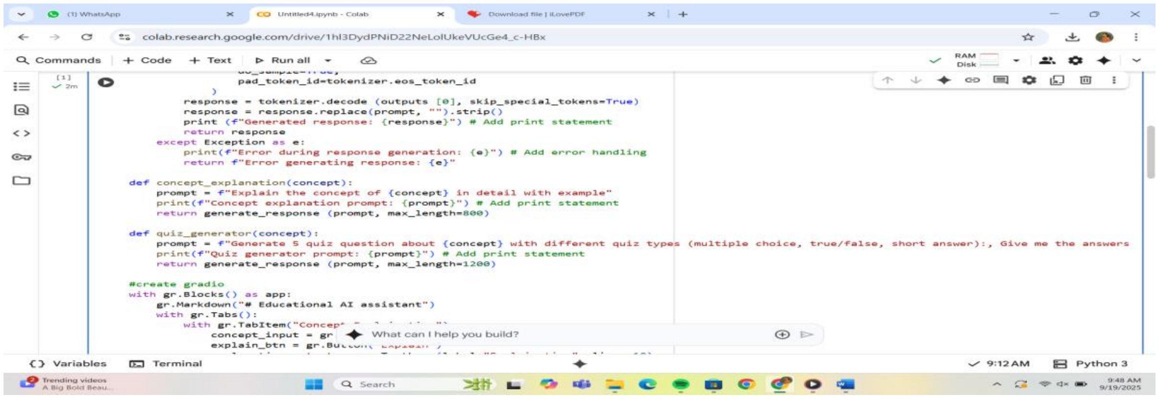
10. Testing

Testing strategy included:
• Unit Testing – Verifying individual functions like generate_response and extract_text_from_pdf.
• Functional Testing – Validating requirement analysis and code generation.
• Manual Testing – Uploading PDFs, typing inputs, checking responses.
• Edge Case Handling – Testing empty inputs, very large files, unsupported formats.

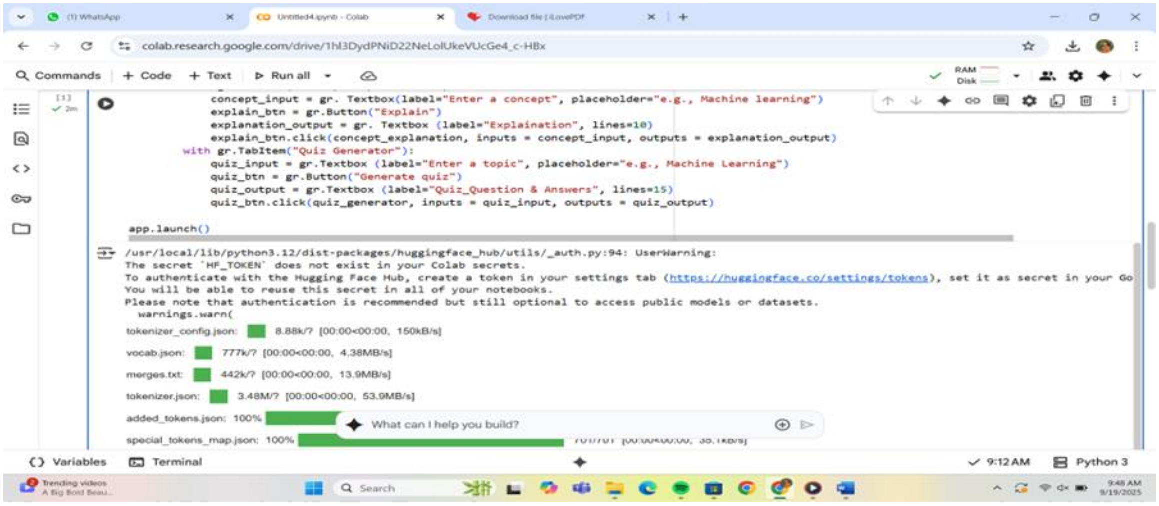
11. Screenshots


colab.research.google.com/drive/1hl3DydPNiD22NeLolUkeVUcGe4_c-HBx
Commands
+ Code
+ Text
Run all
What can I help you build?
Variables
Terminal
9:12 AM
Python 3

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

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 # Set device_map to None when CUDA is not availabl
)

if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

def generate_response (prompt, max_length=512):
    try:
        inputs = tokenizer (prompt, return_tensors="pt", truncation=True, max_length=512)
        #if torch.cuda.is_available(): # Remove this as device_map handles device placement
        # inputs ={k: v.to (model.device) for k, v in inputs.items()}
        with torch.no_grad():
            outputs = model.generate(
                **inputs,
                max_length=max_length,
                temperature=0.
                do_sample=True,
                pad_token_id=tokenizer.eos_token_id
```

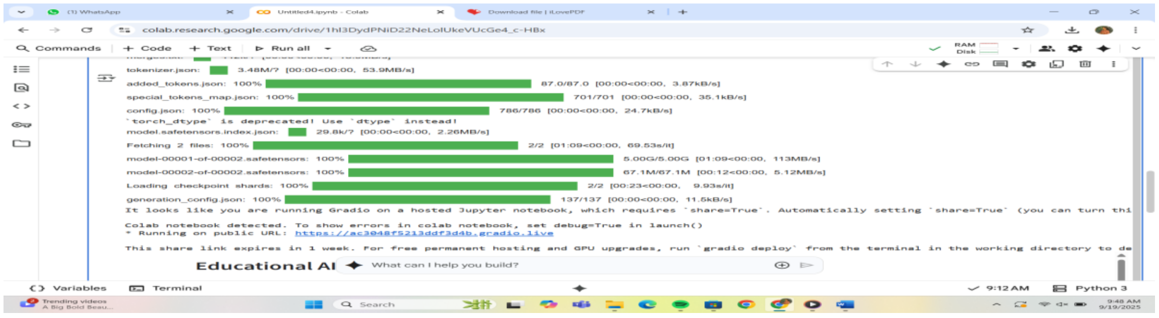


```
                pad_token_id=tokenizer.eos_token_id
            )
        response = tokenizer.decode (outputs [0], skip_special_tokens=True)
        response = response.replace(prompt, "").strip()
        print (f"Generated response: {response}") # Add print statement
        return response
    except Exception as e:
        print(f"Error during response generation: {e}") # Add error handling
        return f"Error generating response: {e}"

def concept_explanation(concept):
    prompt = f"Explain the concept of {concept} in detail with example"
    print(f"Concept explanation prompt: {prompt}") # Add print statement
    return generate_response (prompt, max_length=800)

def quiz_generator(concept):
    prompt = f"Generate 5 quiz question about {concept} with different quiz types (multiple choice, true/false, short answer):, Give me the answers
    print(f"Quiz generator prompt: {prompt}") # Add print statement
    return generate_response (prompt, max_length=1200)

#create gradio
with gr.Blocks() as app:
    gr.Markdown("# Educational AI assistant")
    with gr.Tabs():
        with gr.TabItem("Concep
            concept_input = gr
            explain_btn = gr.Bu
```



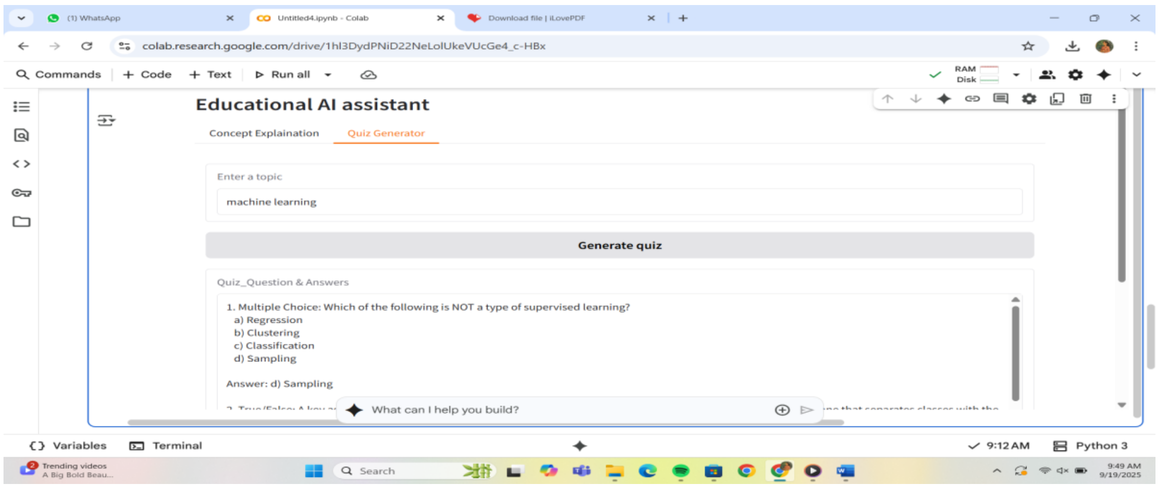
```
            concept_input = gr. Textbox(label="Enter a concept", placeholder="e.g., Machine learning")
            explain_btn = gr.Button("Explain")
            explanation_output = gr. Textbox (label="Explaination", lines=10)
            explain_btn.click(concept_explanation, inputs = concept_input, outputs = explanation_output)
        with gr.TabItem("Quiz Generator"):
            quiz_input = gr.Textbox (label="Enter a topic", placeholder="e.g., Machine Learning")
            quiz_btn = gr.Button("Generate quiz")
            quiz_output = gr.Textbox (label="Quiz_Question & Answers", lines=15)
            quiz_btn.click(quiz_generator, inputs = quiz_input, outputs = quiz_output)

app.launch()
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Go
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json:  8.88k/? [00:00<00:00, 150kB/s]
vocab.json:  777k/? [00:00<00:00, 4.38MB/s]
merges.txt:  442k/? [00:00<00:00, 13.9MB/s]
tokenizer.json:  3.48M/? [00:00<00:00, 53.9MB/s]
added_tokens.json: 100%
special_tokens_map.json: 100%
```



```
tokenizer.json:  3.48M/? [00:00<00:00, 53.9MB/s]
added_tokens.json: 100%  87.0/87.0 [00:00<00:00, 3.87kB/s]
special_tokens_map.json: 100%  701/701 [00:00<00:00, 35.1kB/s]
config.json: 100%  786/786 [00:00<00:00, 24.7kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json:  29.8k/? [00:00<00:00, 2.26MB/s]
Fetching 2 files: 100%  2/2 [01:09<00:00, 69.53s/it]
model-00001-of-00002.safetensors: 100%  5.00G/5.00G [01:09<00:00, 113MB/s]
model-00002-of-00002.safetensors: 100%  67.1M/67.1M [00:12<00:00, 5.12MB/s]
Loading checkpoint shards: 100%  2/2 [00:23<00:00, 9.93s/it]
generation_config.json: 100%  137/137 [00:00<00:00, 11.5kB/s]
It looks like you are running Gradio on a hosted Jupyter notebook, which requires `share=True`. Automatically setting `share=True` (you can turn thi

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://ac3048f5213ddf3d4b.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to de
```

Educational AI



Educational AI assistant

Concept Explaination | Quiz Generator

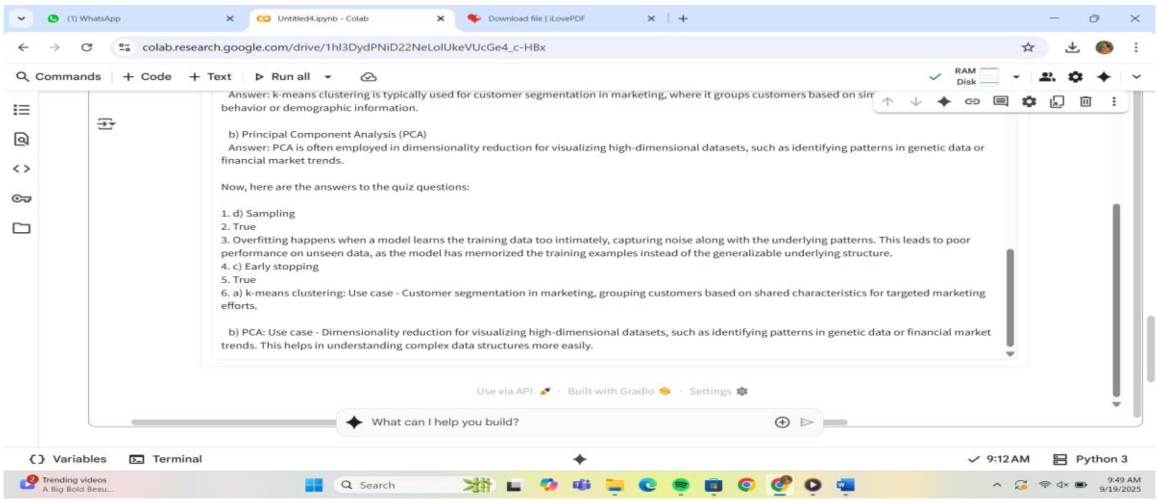
Enter a topic

machine learning

Generate quiz

Quiz_Question & Answers

```
1. Multiple Choice: Which of the following is NOT a type of supervised learning?
  a) Regression
  b) Clustering
  c) Classification
  d) Sampling

Answer: d) Sampling
```



```
  Answer: k-means clustering is typically used for customer segmentation in marketing, where it groups customers based on sim
behavior or demographic information.

  b) Principal Component Analysis (PCA)
  Answer: PCA is often employed in dimensionality reduction for visualizing high-dimensional datasets, such as identifying patterns in genetic data or
financial market trends.

Now, here are the answers to the quiz questions:

1. d) Sampling
2. True
3. Overfitting happens when a model learns the training data too intimately, capturing noise along with the underlying patterns. This leads to poor
performance on unseen data, as the model has memorized the training examples instead of the generalizable underlying structure.
4. c) Early stopping
5. True
6. a) k-means clustering: Use case - Customer segmentation in marketing, grouping customers based on shared characteristics for targeted marketing
efforts.

  b) PCA: Use case - Dimensionality reduction for visualizing high-dimensional datasets, such as identifying patterns in genetic data or financial market
trends. This helps in understanding complex data structures more easily.
```

12. Known Issues

• Generated code may not always be production-ready.
• Large models require significant computational power.
• Output quality depends on LLM capabilities and prompt design.

13. Future Enhancement

• Add syntax highlighting and formatting for generated code.
• Enable export of results into Word/PDF reports.
• Provide REST APIs for integration with other platforms.
• Add authentication and session-based history.
• Enhance visualization of requirement analysis results.