

SmartSDLC – AI-Enhanced Software Development Lifecycle

(Naan Mudhalvan Project – SmartBridge)

INTRODUCTION:

Software development is evolving rapidly with the integration of Artificial Intelligence (AI). Traditional Software Development Lifecycle (**SDLC**) models often face challenges such as inaccurate requirement analysis, delays in testing, and inefficiencies in project management.

SmartSDLC is an AI-Enhanced Software Development Lifecycle framework that integrates machine learning, natural language processing, and automation tools into every stage of SDLC. This project under the **Naan Mudhalvan program by SmartBridge** aims to demonstrate how AI can optimize each stage of development, ensuring faster delivery, reduced costs, and improved quality.

The Software Development Lifecycle (SDLC) is a structured process used to design, develop, test, and deploy software systems. While traditional SDLC models such as Waterfall, Agile, and Spiral have been widely adopted, they often face challenges like delayed timelines, cost overruns, and human errors in requirement analysis, testing, or maintenance. With the growing demand for faster, more reliable, and high-quality software solutions, there is a pressing need to integrate advanced technologies into the development process.

SmartSDLC – AI-Enhanced Software Development Lifecycle addresses this challenge by combining the principles of **SDLC with Artificial Intelligence (AI)**. AI plays a transformative role in automating repetitive tasks, predicting project risks, improving code quality, and enhancing decision-making during each phase of the lifecycle. For example, AI-powered tools can analyze requirements more accurately, generate optimized code, automate testing, detect vulnerabilities, and provide predictive maintenance support. This not only accelerates software delivery but also ensures better efficiency and reliability.

As part of the **Naan Mudhalvan** initiative, **SmartSDLC** is designed to help students and aspiring professionals gain practical exposure to emerging technologies in software engineering. The initiative bridges the gap between academic knowledge and industry practices by equipping

learners with skills that are relevant to real-world software development. By adopting AI in the SDLC, the project encourages innovation, reduces manual effort, and sets the foundation for building intelligent, scalable, and adaptive software systems.

In summary, SmartSDLC demonstrates how AI can enhance the traditional software development process, making it faster, smarter, and more efficient. It reflects the vision of Naan Mudhalvan to prepare future-ready professionals capable of driving technological advancements in the IT industry.

PROJECT DESCRIPTION :

Our project “**SmartSDLC – AI-Enhanced Software Development Lifecycle**” is developed by our team under the Naan Mudhalvan with Smart Bridge program.

Team ID : NM2025TMID06277

Team Leader : VIKNESH S

Member : VIGNESH D

Member : THILAGAN C

This project, as part of the **Naan Mudhalvan** skill development initiative, aims to empower students with the knowledge and hands-on experience required to apply AI tools and techniques across various phases of the SDLC.

The project covers how AI can be utilized in:

- **Requirement Gathering:** Using NLP to analyze client needs and generate software requirement specifications.
- **Design Phase:** Employing AI to recommend system architectures and design patterns based on project needs.
- **Development:** Implementing AI-powered code generation, code completion, and error prediction tools to enhance developer productivity.
- **Testing:** Automating test case generation, bug detection, and performance analysis using AI algorithms.
- **Deployment & Maintenance:** Utilizing AI for predictive maintenance, anomaly detection, and continuous integration/delivery (CI/CD) enhancements.

By simulating real-world scenarios and building AI-integrated tools for each stage, **SmartSDLC** fosters a deep understanding of how modern AI techniques can reshape software engineering.

Key Objectives:

- Integrate AI tools and techniques across all SDLC phases.
- Improve software quality, reduce development time, and lower costs.
- Provide students with practical exposure to AI in real-world software projects.
- Promote innovation in the software development process through automation and intelligent decision-making.

Technologies Used:

- **Languages:** Python.
- **AI/ML Frameworks:** IBM Watsonx AI & Granite Models, FastAPI, Langchain, Streamlit.
- **DevOps Tools:** Git & GitHub.
- **Library:** Transformers, torch, gradio, PyPDF2.
- **Cloud:** Google Drive / Colab storage.

Expected Outcomes:

- A working prototype of an AI-powered SDLC framework.
- Enhanced student understanding of both software development and AI integration.
- A documented guide showcasing AI applications across each SDLC phase.
- Faster SDLC cycle with reduced human effort.
- Scalable solution for enterprises and startups.
- Cost-efficient and high-quality software development.

Applications:

- IT Companies for faster project delivery.
- Startups to reduce software development costs.
- E-Governance projects for efficiency.
- Educational institutions for training students in AI-driven SDLC.

Proposed Solution:

The proposed solution is to develop an **AI-Integrated Software Development Framework** that enhances each phase of the Software Development Lifecycle (SDLC) using modern Artificial Intelligence technologies. This framework, called **SmartSDLC**, aims to reduce manual effort, increase development speed, improve software quality, and enable intelligent decision-making throughout the software lifecycle.

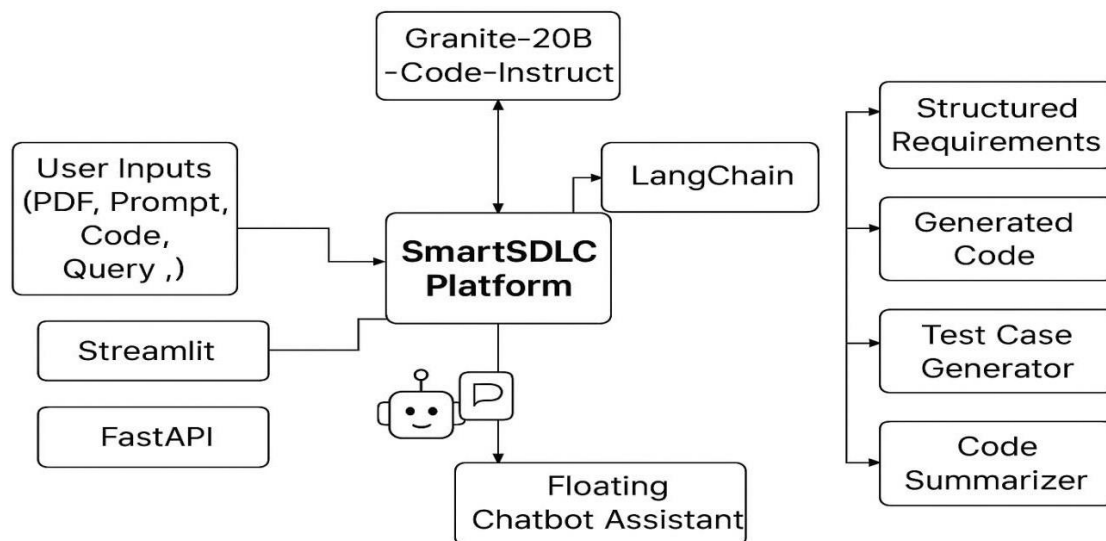
Intelligent Testing Automation:

- Use AI to **auto-generate test cases** based on code and requirements.
- Apply machine learning for **predictive bug detection**, performance testing, and identifying test coverage gaps.

Benefits of the Proposed Solution:

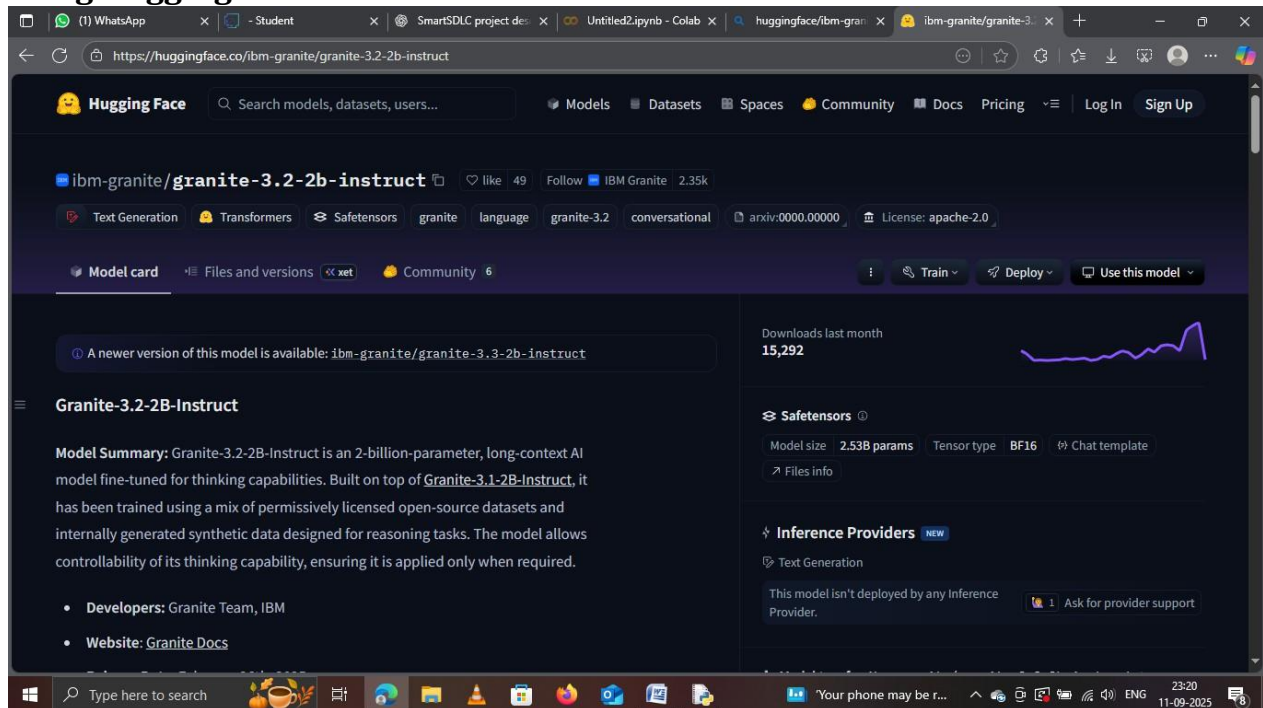
- **Reduced Time-to-Market:** Automation accelerates development and testing phases.
- **Improved Accuracy:** AI reduces human errors in code and requirement interpretation.
- **Skill Development:** Equips students with practical AI and SDLC integration skills.

PROJECT ARCHITECTURE:



IMPLEMENTATION :

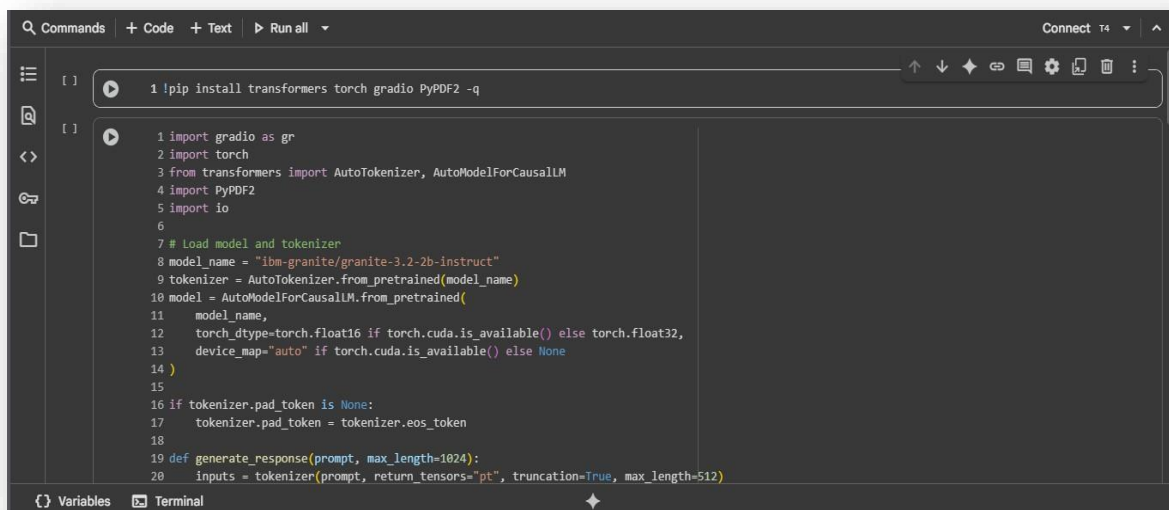
Using Hugging Face for AI Models :



- Here for this project ([Hugging Face](https://huggingface.co/ibm-granite/granite-3.2-2b-instruct)), we are using “ibm-granite/granite-3.2-2b-instruct” which is compatible fast and light weight.

Using Google Colab for Execution :

- Click on the first link ([Google Colab](https://colab.research.google.com/)), then click on “Files” and then “Open Notebook”.



```
Untitled2.ipynb
File Edit View Insert Runtime Tools Help
Q Commands + Code + Text ▶ Run all
Connect T4

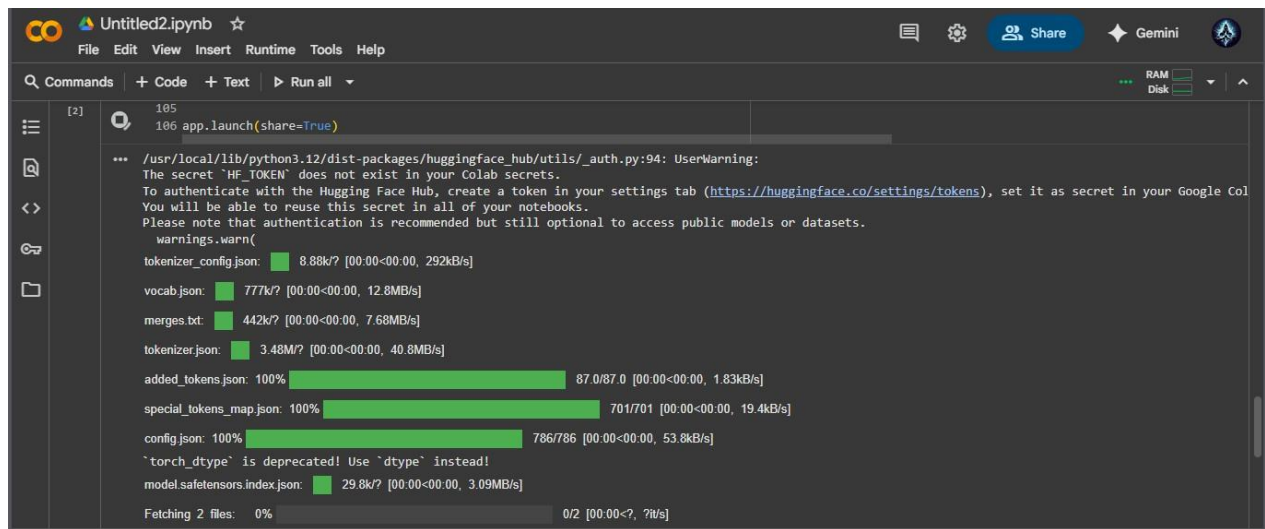
18
19 def generate_response(prompt, max_length=1024):
20     inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)
21
22     if torch.cuda.is_available():
23         inputs = {k: v.to(model.device) for k, v in inputs.items()}
24
25     with torch.no_grad():
26         outputs = model.generate(
27             **inputs,
28             max_length=max_length,
29             temperature=0.7,
30             do_sample=True,
31             pad_token_id=tokenizer.eos_token_id
32         )
33
34     response = tokenizer.decode(outputs[0], skip_special_tokens=True)
35     response = response.replace(prompt, "").strip()
36     return response
37
38 def extract_text_from_pdf(pdf_file):
39     if pdf_file is None:
40         return ""
41
42     try:
```

```
Untitled2.ipynb
File Edit View Insert Runtime Tools Help
Q Commands + Code + Text ▶ Run all
Connect T4

42     try:
43         pdf_reader = PyPDF2.PdfReader(pdf_file)
44         text = ""
45         for page in pdf_reader.pages:
46             text += page.extract_text() + "\n"
47         return text
48     except Exception as e:
49         return f"Error reading PDF: {str(e)}"
50
51 def requirement_analysis(pdf_file, prompt_text):
52     # Get text from PDF or prompt
53     if pdf_file is not None:
54         content = extract_text_from_pdf(pdf_file)
55         analysis_prompt = f"Analyze the following document and extract key software requirements. Organize them into functional requirements, non-functional requirements, and technical requirements."
56     else:
57         analysis_prompt = f"Analyze the following requirements and organize them into functional requirements, non-functional requirements, and technical requirements."
58
59     return generate_response(analysis_prompt, max_length=1200)
60
61 def code_generation(prompt, language):
62     code_prompt = f"Generate {language} code for the following requirement:\n\n{prompt}\n\nCode:"
63     return generate_response(code_prompt, max_length=1200)
64
65 # Create Gradio interface
66 with gr.Blocks() as app:
```

```
65 # Create Gradio interface
66 with gr.Blocks() as app:
67     gr.Markdown("# AI Code Analysis & Generator")
68
69     with gr.Tabs():
70         with gr.TabItem("Code Analysis"):
71             with gr.Row():
72                 with gr.Column():
73                     pdf_upload = gr.File(label="Upload PDF", file_types=[".pdf"])
74                     prompt_input = gr.Textbox(
75                         label="Or write requirements here",
76                         placeholder="Describe your software requirements...",
77                         lines=5
78                     )
79                     analyze_btn = gr.Button("Analyze")
80
81                 with gr.Column():
82                     analysis_output = gr.Textbox(label="Requirements Analysis", lines=20)
83
84             analyze_btn.click(requirement_analysis, inputs=[pdf_upload, prompt_input], outputs=analysis_output)
85
86         with gr.TabItem("Code Generation"):
87             with gr.Row():
88                 with gr.Column():
```

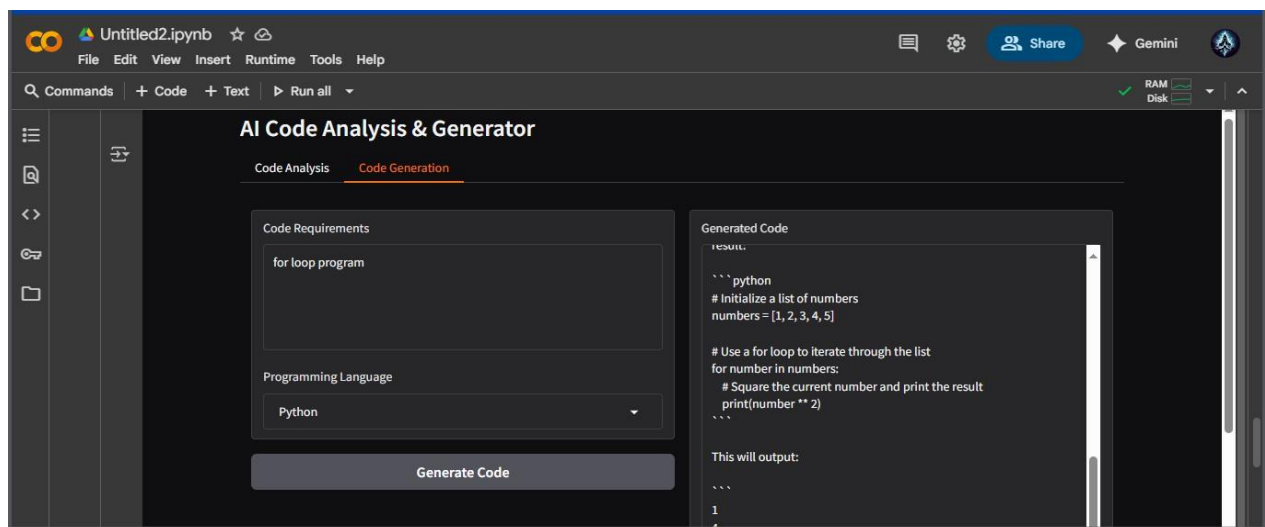
```
85
86         with gr.TabItem("Code Generation"):
87             with gr.Row():
88                 with gr.Column():
89                     code_prompt = gr.Textbox(
90                         label="Code Requirements",
91                         placeholder="Describe what code you want to generate...",
92                         lines=5
93                     )
94                     language_dropdown = gr.Dropdown(
95                         choices=["Python", "JavaScript", "Java", "C++", "C#", "PHP", "Go", "Rust"],
96                         label="Programming Language",
97                         value="Python"
98                     )
99                     generate_btn = gr.Button("Generate Code")
100
101                 with gr.Column():
102                     code_output = gr.Textbox(label="Generated Code", lines=20)
103
104             generate_btn.click(code_generation, inputs=[code_prompt, language_dropdown], outputs=code_output)
105
106 app.launch(share=True)
```



```
105
106 app.launch(share=True)

... /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 Google Colab. 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, 292kB/s]
vocab.json: 777k? [00:00<00:00, 12.8MB/s]
merges.txt: 442k? [00:00<00:00, 7.68MB/s]
tokenizer.json: 3.48M? [00:00<00:00, 40.8MB/s]
added_tokens.json: 100% [87.0/87.0] [00:00<00:00, 1.83kB/s]
special_tokens_map.json: 100% [701/701] [00:00<00:00, 19.4kB/s]
config.json: 100% [786/786] [00:00<00:00, 53.8kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json: 29.8k? [00:00<00:00, 3.09MB/s]
Fetching 2 files: 0% [0/2] [00:00<?, ?B/s]
```

OUTPUT:



- Now you can see our project output.
- Click on the URL to open the **Project Application** link.

[Click in](#)

Conclusion:

The **SmartSDLC** project demonstrates how Artificial Intelligence can be effectively integrated into every phase of the Software Development Lifecycle to create smarter, faster, and more reliable software systems. By leveraging AI tools such as Natural Language Processing, Machine Learning, and Intelligent Automation, the project enhances traditional development workflows—from requirement gathering to deployment and maintenance.

Through this AI-driven approach, **SmartSDLC** not only improves productivity and code quality but also reduces development time and operational costs. Most importantly, it equips students with real-world, future-ready skills by combining software engineering fundamentals with cutting-edge AI technologies.

This project aligns with the goals of the **Naan Mudhalvan** initiative by fostering innovation, practical skill development, and industry-relevant expertise among learners. As the software industry continues to evolve, SmartSDLC sets a strong foundation for the next generation of AI-enhanced software development practices.
