SDLC

TEAM MEMBERS-NM2025TMID11737

GOPALA KRISHNAN

DHINESH KUMAR

JEEVA G

JEEVA V

RAJENDRAN

!pip install tranformers torch gradio pyPDF2

import gradio as gr

import torch

from transformers import AutoTokenizer, AutoModelForCausalLM

import PyPDF2

# Load model and tokenizer

model\_name = &quot;ibm-granite/granite-3.2-2b-instruct&quot;

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=&quot;auto&quot; if torch.cuda.is\_available() else None

)

# Ensure pad token exists

if tokenizer.pad\_token is None:

tokenizer.pad\_token = tokenizer.eos\_token

# Text generation

def generate\_response(prompt, max\_length=1024):

inputs = tokenizer(prompt, return\_tensors=&quot;pt&quot;, truncation=True, max\_length=512)

if torch.cuda.is\_available():

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=0.7,

do\_sample=True,

pad\_token\_id=tokenizer.eos\_token\_id

)

response = tokenizer.decode(outputs[0], skip\_special\_tokens=True)

response = response.replace(prompt, &quot;&quot;).strip()

return response

# Extract text from PDF

def extract\_text\_from\_pdf(pdf\_file):

if pdf\_file is None:

return &quot;&quot;

try:

file\_path = pdf\_file.name if hasattr(pdf\_file, &quot;name&quot;) else pdf\_file

pdf\_reader = PyPDF2.PdfReader(file\_path)

text = &quot;&quot;

for page in pdf\_reader.pages:

page\_text = page.extract\_text()

if page\_text:

text += page\_text + &quot;\n&quot;

return text.strip()

except Exception as e:

return f&quot;Error reading PDF: {str(e)}&quot;

# Requirement analysis

def requirement\_analysis(pdf\_file, prompt\_text):

if pdf\_file is not None:

content = extract\_text\_from\_pdf(pdf\_file)

analysis\_prompt = (

&quot;Analyze the following document and extract key software requirements. &quot;

&quot;Organize them into functional requirements, non-functional requirements, &quot;

&quot;and technical requirements.\n\n&quot; + content

)

else:

analysis\_prompt = (

&quot;Analyze the following requirement and organize them into functional requirements, &quot;

&quot;non-functional requirements, and technical requirements.\n\n&quot; + prompt\_text

)

return generate\_response(analysis\_prompt, max\_length=1200)

# Code generation

def code\_generation(prompt, language):

code\_prompt = f&quot;Generate {language} code for the following requirement:\n\n{prompt}\n\nCode:&quot;

return generate\_response(code\_prompt, max\_length=1200)

# Gradio app

with gr.Blocks() as app:

gr.Markdown(&quot;# ðŸ¤– AI Code Analysis &amp; Generator&quot;)

with gr.Tabs():

with gr.TabItem(&quot;Code Analysis&quot;):

with gr.Row():

with gr.Column():

pdf\_upload = gr.File(label=&quot;Upload PDF&quot;, file\_types=[&quot;.pdf&quot;])

prompt\_input = gr.Textbox(

label=&quot;Or write requirements here&quot;,

placeholder=&quot;Describe your software requirements...&quot;,

lines=5

)

analyze\_btn = gr.Button(&quot;Analyze&quot;)

with gr.Column():

analysis\_output = gr.Textbox(label=&quot;Requirements Analysis&quot;, lines=20)

analyze\_btn.click(requirement\_analysis, inputs=[pdf\_upload, prompt\_input],

outputs=analysis\_output)

with gr.TabItem(&quot;Code Generation&quot;):

with gr.Row():

with gr.Column():

code\_prompt\_input = gr.Textbox(label=&quot;Code Requirements&quot;)

language\_input = gr.Textbox(

label=&quot;Programming Language&quot;,

placeholder=&quot;e.g., Python, Java, JavaScript&quot;

)

generate\_btn = gr.Button(&quot;Generate Code&quot;)

with gr.Column():

code\_output = gr.Textbox(label=&quot;Generated Code&quot;, lines=20)

generate\_btn.click(code\_generation, inputs=[code\_prompt\_input, language\_input],

outputs=code\_output)

app.launch(share=True)

import gradio as gr

import torch

from transformers import AutoTokenizer, AutoModelForCausalLM

import PyPDF2

# Load model and tokenizer

model\_name = &quot;ibm-granite/granite-3.2-2b-instruct&quot;

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=&quot;auto&quot; if torch.cuda.is\_available() else None

)

# Ensure pad token exists

if tokenizer.pad\_token is None:

tokenizer.pad\_token = tokenizer.eos\_token

# Core text generation

def generate\_response(prompt, max\_length=1024):

if not prompt or prompt.strip() == &quot;&quot;:

return &quot;âš ï¸ Please provide valid input.&quot;

inputs = tokenizer(prompt, return\_tensors=&quot;pt&quot;, truncation=True, max\_length=512)

if torch.cuda.is\_available():

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=0.7,

do\_sample=True,

pad\_token\_id=tokenizer.eos\_token\_id

)

response = tokenizer.decode(outputs[0], skip\_special\_tokens=True)

response = response.replace(prompt, &quot;&quot;).strip()

return response if response else &quot;âš ï¸ No response generated.&quot;

# PDF text extraction

def extract\_text\_from\_pdf(pdf\_file):

if pdf\_file is None:

return &quot;&quot;

try:

file\_path = pdf\_file.name if hasattr(pdf\_file, &quot;name&quot;) else pdf\_file

pdf\_reader = PyPDF2.PdfReader(file\_path)

text = &quot;&quot;

for page in pdf\_reader.pages:

page\_text = page.extract\_text()

if page\_text:

text += page\_text + &quot;\n&quot;

return text.strip() if text else &quot;âš ï¸ No readable text found in PDF.&quot;

except Exception as e:

return f&quot;â Œ Error reading PDF: {str(e)}&quot;

# SDLC Phase Analysis

def sdlc\_analysis(phase, pdf\_file, user\_input):

content = &quot;&quot;

if pdf\_file is not None:

content = extract\_text\_from\_pdf(pdf\_file)

else:

content = user\_input

if not content:

return &quot;âš ï¸ Please upload a PDF or enter requirements.&quot;

prompt = f&quot;Perform {phase} phase analysis for the following requirements:\n\n{content}&quot;

return generate\_response(prompt, max\_length=1200)

# Code Generation

def code\_generation(prompt, language):

if not language:

return &quot;âš ï¸ Please specify a programming language.&quot;

code\_prompt = f&quot;Generate {language} code for the following requirement:\n\n{prompt}\n\nCode:&quot;

return generate\_response(code\_prompt, max\_length=1200)

# Gradio App

with gr.Blocks() as app:

gr.Markdown(&quot;# ðŸš€ AI-Enhanced SDLC Assistant&quot;)

with gr.Tabs():

# Requirement Analysis

with gr.TabItem(&quot;Requirement Analysis&quot;):

with gr.Row():

with gr.Column():

pdf\_upload = gr.File(label=&quot;Upload Requirement PDF&quot;, file\_types=[&quot;.pdf&quot;])

req\_input = gr.Textbox(label=&quot;Or enter requirements&quot;, lines=5)

analyze\_btn = gr.Button(&quot;Analyze Requirements&quot;)

with gr.Column():

req\_output = gr.Textbox(label=&quot;Requirements Analysis&quot;, lines=20)

analyze\_btn.click(lambda pdf, text: sdlc\_analysis(&quot;Requirement Analysis&quot;, pdf, text),

inputs=[pdf\_upload, req\_input], outputs=req\_output)

# System Design

with gr.TabItem(&quot;System Design&quot;):

with gr.Row():

with gr.Column():

design\_input = gr.Textbox(label=&quot;Enter system design details&quot;, lines=5)

design\_btn = gr.Button(&quot;Generate Design&quot;)

with gr.Column():

design\_output = gr.Textbox(label=&quot;System Design Output&quot;, lines=20)

design\_btn.click(lambda text: generate\_response(f&quot;Generate a system design for:\n\n{text}&quot;,

max\_length=1200),

inputs=design\_input, outputs=design\_output)

# Implementation

with gr.TabItem(&quot;Implementation&quot;):

with gr.Row():

with gr.Column():

impl\_input = gr.Textbox(label=&quot;Enter implementation details&quot;, lines=5)

impl\_btn = gr.Button(&quot;Generate Implementation Plan&quot;)

with gr.Column():

impl\_output = gr.Textbox(label=&quot;Implementation Output&quot;, lines=20)

impl\_btn.click(lambda text: generate\_response(f&quot;Create an implementation strategy

for:\n\n{text}&quot;, max\_length=1200),

inputs=impl\_input, outputs=impl\_output)

# Testing

with gr.TabItem(&quot;Testing&quot;):

with gr.Row():

with gr.Column():

test\_input = gr.Textbox(label=&quot;Enter testing requirements&quot;, lines=5)

test\_btn = gr.Button(&quot;Generate Test Cases&quot;)

with gr.Column():

test\_output = gr.Textbox(label=&quot;Test Plan Output&quot;, lines=20)

test\_btn.click(lambda text: generate\_response(f&quot;Generate detailed test cases and scenarios

for:\n\n{text}&quot;, max\_length=1200),

inputs=test\_input, outputs=test\_output)

# Deployment

with gr.TabItem(&quot;Deployment&quot;):

with gr.Row():

with gr.Column():

dep\_input = gr.Textbox(label=&quot;Enter deployment requirements&quot;, lines=5)

dep\_btn = gr.Button(&quot;Generate Deployment Plan&quot;)

with gr.Column():

dep\_output = gr.Textbox(label=&quot;Deployment Output&quot;, lines=20)

dep\_btn.click(lambda text: generate\_response(f&quot;Create a deployment plan for:\n\n{text}&quot;,

max\_length=1200),

inputs=dep\_input, outputs=dep\_output)

# Maintenance

with gr.TabItem(&quot;Maintenance&quot;):

with gr.Row():

with gr.Column():

maint\_input = gr.Textbox(label=&quot;Enter maintenance requirements&quot;, lines=5)

maint\_btn = gr.Button(&quot;Generate Maintenance Plan&quot;)

with gr.Column():

maint\_output = gr.Textbox(label=&quot;Maintenance Output&quot;, lines=20)

maint\_btn.click(lambda text: generate\_response(f&quot;Generate a maintenance strategy

for:\n\n{text}&quot;, max\_length=1200),

inputs=maint\_input, outputs=maint\_output)

# Code Generator

with gr.TabItem(&quot;Code Generator&quot;):

with gr.Row():

with gr.Column():

code\_req = gr.Textbox(label=&quot;Code Requirements&quot;, lines=5)

lang\_input = gr.Textbox(label=&quot;Programming Language&quot;, placeholder=&quot;e.g., Python, Java,

JavaScript&quot;)

code\_btn = gr.Button(&quot;Generate Code&quot;)

with gr.Column():

code\_out = gr.Textbox(label=&quot;Generated Code&quot;, lines=20)

code\_btn.click(code\_generation, inputs=[code\_req, lang\_input], outputs=code\_out)

app.launch(share=True)