Here's a **full, updated example** of a **local AI coding assistant** using:

- **LangChain agent framework**
- **Ollama LLaMA 3** as the local LLM
- **FAISS vectorstore** over your codebase + uploaded PDFs
- **Conversational memory** for context
- **Gradio UI** for chat interface
- Retrieval + code generation + error solving in one agent via tools

---

Full `coding_assistant.py`:-

```
import os
import gradio as gr
from langchain_community.llms import Ollama
from langchain.document_loaders import RecursiveDirectoryLoader, PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter,
CharacterTextSplitter
from langchain.vectorstores import FAISS
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.chains import RetrievalQA
from langchain.memory import ConversationBufferMemory
from langchain.agents import initialize_agent, Tool

# === Config ===
CODEBASE_PATH = "codebase"
PDF_DIR = "uploaded_pdfs"
FAISS_INDEX_PATH = "coding_assistant_faiss_index"

os.makedirs(CODEBASE_PATH, exist_ok=True)
os.makedirs(PDF_DIR, exist_ok=True)

# === Initialize LLM and embeddings ===
llm = Ollama(model="llama3")
embedding_model = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")

# === Load codebase documents ===
code_loader = RecursiveDirectoryLoader(
    CODEBASE_PATH,
    ignore_hidden=True,

glob="**/*.{py,js,ts,java,cpp,c,cs,go,rb,php,sh,bash,md,rst,txt,json,yaml,yml
,ini,toml,sql}"
)
print("Loading codebase files...")
code_docs_raw = code_loader.load()

code_splitter = RecursiveCharacterTextSplitter(chunk_size=1000,
chunk_overlap=200)
```

```
code_docs = code_splitter.split_documents(code_docs_raw)

# === Load or create FAISS index ===
if os.path.exists(FAISS_INDEX_PATH):
    print("Loading existing FAISS index...")
    vectorstore = FAISS.load_local(FAISS_INDEX_PATH,
embeddings=embedding_model)
else:
    print("Creating FAISS index from codebase...")
    vectorstore = FAISS.from_documents(code_docs, embedding_model)
    vectorstore.save_local(FAISS_INDEX_PATH)

retriever = vectorstore.as_retriever(search_type="similarity", k=5)

# === Memory ===
memory = ConversationBufferMemory(memory_key="chat_history",
return_messages=True)

# === Define QA tool for retrieval ===
qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever)

def codebase_qa_tool(query: str) -> str:
    return qa_chain.run(query)

# === Tool to add PDFs dynamically ===
def add_pdfs(files):
    global vectorstore, retriever, qa_chain, agent

    new_docs = []
    for file in files:
        filepath = os.path.join(PDF_DIR, file.name)
        file.save(filepath)

        loader = PyPDFLoader(filepath)
        pdf_docs = loader.load()

        pdf_splitter = CharacterTextSplitter(chunk_size=1000,
chunk_overlap=200)
        pdf_chunks = pdf_splitter.split_documents(pdf_docs)
        new_docs.extend(pdf_chunks)

    if new_docs:
        vectorstore.add_documents(new_docs)
        vectorstore.save_local(FAISS_INDEX_PATH)
        retriever = vectorstore.as_retriever(search_type="similarity", k=5)
        qa_chain.retriever = retriever
        # Update agent's tool retriever as well
        for t in agent.tools:
            if t.name == "Codebase QA":
                t.func = codebase_qa_tool
        return f"□ Added {len(new_docs)} PDF chunks!"
    else:
        return "□ No documents added."

# === Initialize LangChain agent with tools ===
tools = [
    Tool(
```

```
        name="Codebase QA",
        func=codebase_qa_tool,
        description="Use this tool to answer questions about the codebase and
uploaded documents."
    ),
    # You can add more tools here like a code generator, error solver, etc.
]

agent = initialize_agent(
    tools=tools,
    llm=llm,
    agent="conversational-react-description",
    memory=memory,
    verbose=True
)

# === Gradio chat function ===
def chat_fn(message, chat_history):
    response = agent.run(message)
    chat_history = chat_history or []
    chat_history.append((message, response))
    return chat_history, chat_history

# === Gradio UI ===
with gr.Blocks() as demo:
    gr.Markdown("#  AI Coding Assistant")
    gr.Markdown(
        """
        Ask questions about your codebase or uploaded PDFs, get code
generation help, or error solving.
        Upload PDFs to add more documents dynamically.
        """
    )

    with gr.Row():
        chatbot = gr.Chatbot()
        with gr.Column():
            msg = gr.Textbox(placeholder="Ask me anything about your code or
upload PDFs...")
            upload = gr.File(label="Upload PDF(s)", file_types=[".pdf"],
file_count="multiple")
            upload_status = gr.Textbox(label="Upload Status",
interactive=False)
            clear = gr.Button("Clear Chat")

    state = gr.State([])

    msg.submit(chat_fn, [msg, state], [chatbot, state])
    clear.click(lambda: ([], []), outputs=[chatbot, state])
    upload.upload(fn=add_pdfs, inputs=[upload], outputs=[upload_status])

demo.launch()
```

How to run:

1. Put your code files in `codebase/`
2. Run:

```
pip install langchain langchain-community faiss-cpu sentence-transformers
ollama gradio pypdf
python coding_assistant.py
```

3. Open the Gradio URL, upload PDFs if you want, and start chatting!

What this does:

- Loads your codebase + PDFs into a FAISS index for retrieval
- Uses Ollama LLaMA 3 as LLM to answer questions and generate code
- Uses LangChain agent framework to combine tools + memory + chat
- Lets you upload PDFs anytime and adds them to searchable context
- Supports multi-turn chat with memory in the session

Next steps (optional features you can add):

- Add a code generation tool (e.g., prompt LLM to generate functions)
- Add an error solver tool (parse error logs and suggest fixes)
- Add file/line metadata for better source traceability
- Integrate with IDE for live coding assistance
- Add sandboxed code execution to test snippets dynamically

Here are a few simple example files you can put inside your `codebase/` folder to test your Codebase QA Agent. These cover different languages and a README file for docs:

---

1. `codebase/main.py`

```
def greet(name: str) -> str:
```

```
    """Return a greeting message."""
    return f"Hello, {name}!"

def main():
    user_name = input("Enter your name: ")
    print(greet(user_name))

if __name__ == "__main__":
    main()
```

---

2. codebase/utils.js

```
// Utility function to add two numbers
function add(a, b) {
  return a + b;
}

// Export the function
module.exports = { add };
```

---

3. codebase/config.yaml

```
app:
  name: SampleApp
  version: 1.0.0
  debug: true

database:
  host: localhost
  port: 5432
  user: admin
  password: secret
```

---

4. codebase/README.md

```
# Sample Codebase

This repository contains sample files in Python and JavaScript.

- `main.py`: A simple greeting script.
- `utils.js`: Basic utility functions.
- `config.yaml`: Configuration settings.
```

---

5. codebase/script.sh

```
#!/bin/bash
echo "Running setup script..."
# Simple setup commands
mkdir -p logs
touch logs/app.log
echo "Setup complete."
```

Put these in your `codebase/` folder and run the app — then try asking questions like:

- *"What does greet do?"*
- *"How do I add two numbers?"*
- *"What's the database host?"*
- *"What commands are in the setup script?"*