LangGraph RAG Python Code Assistant

AI-Powered Coding Companion with Memory

Doaa Said

December 7, 2025

Key Libraries

- **Streamlit:** Interactive Web UI
- **LangGraph:** State-machine workflow for AI tasks
- **LangChain + ChatOllama:** LLM integration
- **Chroma:** Vector database for RAG
- **HuggingFaceEmbeddings:** Sentence embeddings
- **datasets:** HumanEval dataset for code examples

LLM Setup

ChatOllama Setup

```
llm = ChatOllama(
    model="llama3.2",
    temperature=0.6,
    num_ctx=2048,
    base_url="http://localhost:11434",
)
```

- **temperature=0.6** → controls creativity
- **num_ctx=2048** → context window
- **base_url** → local LLaMA server

Dataset Vector Store

Code Example

```
dataset = load_dataset("openai/openai_humaneval", split="test")
documents = [
    Document(page_content=f"Problem:\n{item['prompt']}\n\nSolution:\n{item['canonical_solution']}")
    for item in dataset
]

embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
vector_store = Chroma(collection_name="py_examples", embedding_function=embeddings)
vector_store.add_documents(documents)
```

Memory Module

Conversation Memory

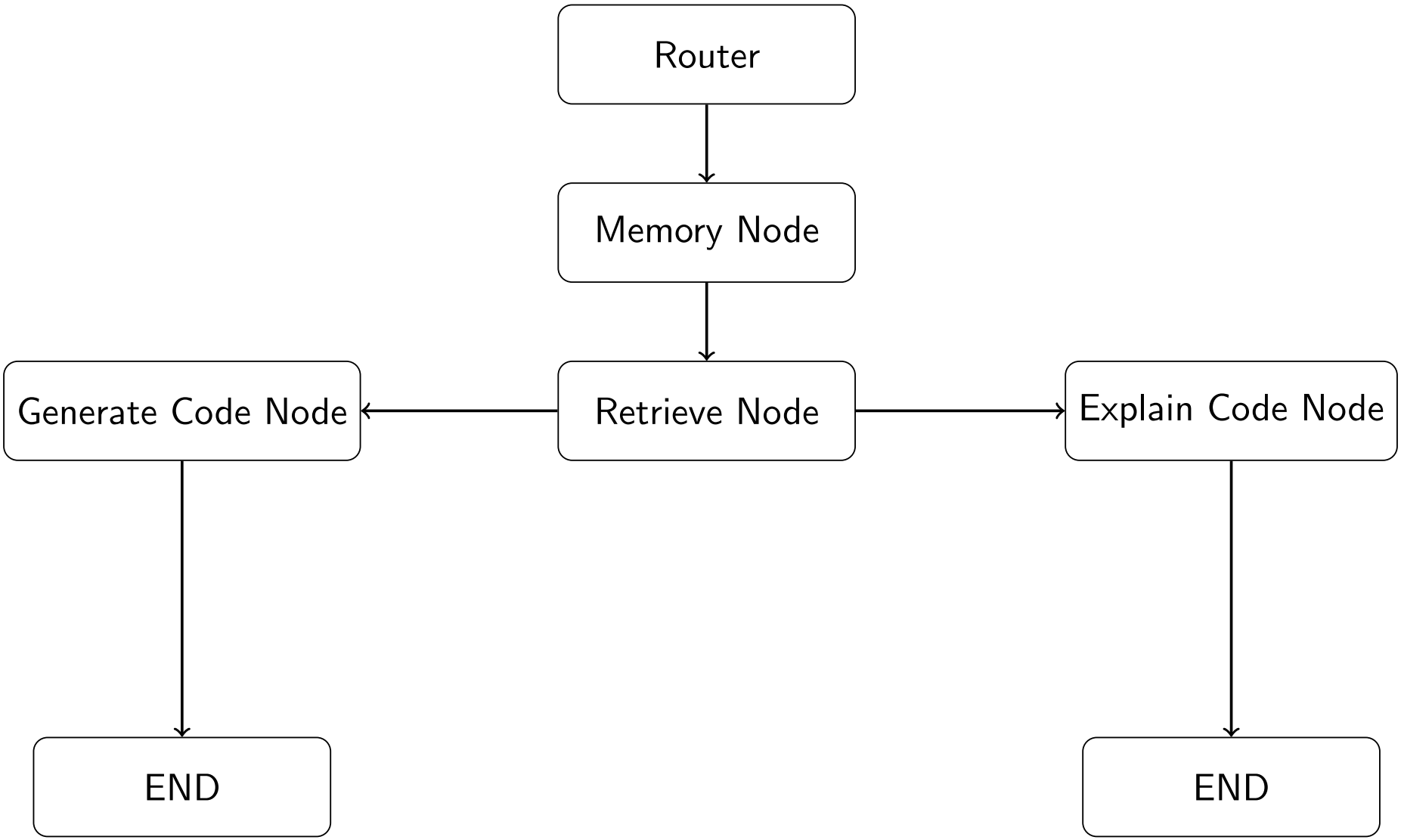
```
memory_buffer = deque(maxlen=5)

def add_to_memory(role, content):
    memory_buffer.append({"role": role, "content": content})

def generate_memory_prompt():
    return "\n".join([f"{m['role']}: {m['content']}" for m in memory_buffer])
```

- Maintains last 5 conversation turns
- Provides context to LLM

LangGraph Workflow



Key Features

- Integrates RAG + LLM for code generation
- Conversational memory (last 5 turns)
- Intent detection: Generate or Explain
- Interactive web interface via Streamlit
- Modular extensible workflow

Conclusion

- Combines RAG + LLM + Memory for interactive coding assistant
- Modular workflow using LangGraph
- Real-time Streamlit UI
- Scalable for future datasets LLMs