LEARNING OBJECTIVES

By the end of this course, learners will be able to:

1. Explain what Retrieval Augmented Generation (RAG) is and why it's valuable
2. Set up a local large language model using Ollama
3. Configure RAG implementation using Python programming
4. Apply RAG solutions to improve LLM accuracy for domain-specific tasks

SECTION 1: ATTENTION-GRABBING OPENER *(30 seconds)*

[Visual: Split screen showing ChatGPT giving incorrect legal advice vs. accurate RAG-enhanced response]

Narrator: "What if I told you that the AI you're relying on for critical work might be giving you outdated or completely wrong information? And what if you could fix this problem yourself, without waiting for big tech companies to solve it for you? Today, you'll learn how to supercharge any large language model with your own data using Retrieval Augmented Generation, served completely locally for maximum security."

SECTION 2: COURSE OVERVIEW *(60-90 seconds)*

[Visual: Clean title slide with course outline]

Narrator: "Welcome to Local Retrieval Augmented Generation in Python. I'm Alexander Harris, and in the next 8 minutes, you'll master three critical skills:

First, we'll explore what RAG is and why it's a game-changer for AI accuracy. You'll understand real-world use cases where RAG transforms unreliable AI into a precision tool.

Second, we'll dive into a practical scenario following Linus, a legal researcher who needs ChatGPT's intelligence but with enhanced security and domain-specific accuracy.

Finally, we'll get hands-on with code, setting up Ollama for local AI serving and building a complete RAG solution in Python.

Prerequisites: You should have basic Python knowledge, command line familiarity, and understand client-server interactions."

SECTION 3: CONCEPT EXPLANATION *(2-3 minutes)*

[Visual: Simple diagram showing traditional LLM vs RAG-enhanced LLM]

Narrator: "So what exactly is Retrieval Augmented Generation? Think of RAG as giving your AI a specialized library card.

Traditional large language models are like brilliant students who memorized everything up to their training cutoff date. Ask them about recent events or highly specific domain knowledge, and they'll either make something up or give you outdated information.

RAG solves this by adding a retrieval step. Before generating an answer, the AI searches through your specific documents, finds relevant information, and uses that as context for its response.

[Visual: Callout boxes highlighting benefits]

Why use RAG? Three critical advantages:

- **Accuracy** : Your AI now references YOUR authoritative sources
- **Currency** : Information stays up-to-date as you add new documents
- **Control** : You decide what knowledge the AI can access

Real-world applications include legal research, medical documentation, technical support, and any field where precision matters more than creativity."

SECTION 4: LINUS SCENARIO *(1-2 minutes)*

[Visual: Character illustration of Linus at his desk with legal documents]

Narrator: "Meet Linus, a software developer on a legal research team. Linus loves ChatGPT's capabilities, but he's facing two critical problems:

First, security. His legal documents are confidential - he cannot send sensitive case information over the internet to OpenAI's servers.

Second, accuracy. ChatGPT doesn't know about recent case law, specific regulations in his jurisdiction, or the nuanced legal precedents his team relies on.

[Visual: Split screen showing problems vs solutions]

Traditional solution? Wait for ChatGPT to update, compromise security, or manually research everything.

Linus's solution? Build his own system using Ollama for local AI serving, Python for the application logic, and RAG to inject his legal database directly into the AI's knowledge base.

The result? ChatGPT-level intelligence with bank-level security and laser-focused accuracy on his specific legal domain."

SECTION 5: HANDS-ON DEMO *(3-4 minutes)*

[Visual: Clean code editor with callouts, no cursor pointing]

Narrator: "Let's build Linus's solution step by step. I've prepared a complete GitHub repository - link in the description.

Step 1: Environment Setup [Callout highlighting terminal commands] First, we install Ollama and pull our local model. This runs entirely on your machine - no internet required after setup.

Step 2: Generate Embeddings

[Callout on embedding code section]

We convert Linus's legal documents into mathematical vectors that our AI can search through:

Step 3: Retrieval Function [Callout on retrieval logic] When Linus asks a question, we find the most relevant documents first:

Step 4: RAG Pipeline

[Callout on main function]

Finally, we combine retrieval with generation:

[Visual: Live execution showing before/after comparison]

Watch the difference: Without RAG, the AI gives generic legal advice. With RAG, it references specific case law from Linus's database and provides precise, current information."

SECTION 6: CONCLUSION *(60 seconds)*

[Visual: Summary slide with key takeaways]

Narrator: "Congratulations! You've just learned how to transform any large language model into a domain-specific expert using Retrieval Augmented Generation.

Key takeaways:

- RAG bridges the gap between general AI and specialized knowledge
- Ollama enables secure, local AI deployment
- Python makes RAG implementation accessible and customizable
- Real-world applications span any field requiring accuracy and security

Linus can now query his legal database with natural language, maintain complete data privacy, and get responses based on authoritative sources rather than AI hallucinations.

What's next? Explore advanced embedding techniques, experiment with different local models, and consider implementing semantic chunking for larger document collections.

The complete code repository is linked below. Start building your own RAG solution today, and transform how you interact with your specialized knowledge base."

[End screen with GitHub link and related course suggestions]