COMPREHENSIVE CURRICULUM

Prompt Engineering & LLM Application Development

A Structured Learning Path from
Fundamentals to Production-Ready AI Systems

Phase 1
Core Skills

Phase 2
Applications

Phase 3
Production

Duration: 12–16 Weeks

Target Audience: Software Engineers & Developers

Version 1.0 | December 1, 2025

Contents

1 Executive Summary 4
1.1 Course Overview 4
1.2 Target Audience 4
1.3 Prerequisites 4
1.4 Program Structure 5
1.5 Learning Outcomes 5

2 Phase 1: Core Prompting Skills 6
2.1 Module 1.1: Foundations of Prompt Engineering 6
2.1.1 Overview 6
2.1.2 Reading Assignments 7
2.1.3 Topics Covered 7
2.1.4 Practical Exercises 8
2.2 Module 1.2: Advanced Prompting Patterns 9
2.2.1 Overview 9
2.2.2 Reading Assignments 9
2.2.3 Topics Covered 9
2.2.4 Practical Exercises 10
2.3 Module 1.3: Prompt Evaluation and Quality Assurance 11
2.3.1 Overview 11
2.3.2 Reading Assignments 12
2.3.3 Evaluation Methodologies 12
2.3.4 Practical Exercises 13
2.4 Module 1.4: Multi-Modal and Specialized Prompting 13
2.4.1 Overview 13
2.4.2 Reading Assignments 14
2.4.3 Code Generation Prompting 14
2.4.4 Practical Exercises 14
2.5 Phase 1 Capstone Project 15

3 Phase 2: From Prompts to Applications 16
3.1 Module 2.1: LLM Application Architecture 16
3.1.1 Overview 16
3.1.2 Reading Assignments 17
3.1.3 Architecture Patterns 17
3.1.4 Practical Exercises 18
3.2 Module 2.2: Tool Use and Function Calling 18
3.2.1 Overview 18
3.2.2 Reading Assignments 19
3.2.3 Implementation Patterns 19
3.2.4 Practical Exercises 19

3.3 Module 2.3: Retrieval-Augmented Generation (RAG) . . . 20
3.3.1 Overview . . . 20
3.3.2 Reading Assignments . . . 21
3.3.3 RAG Pipeline Components . . . 21
3.3.4 Practical Exercises . . . 22
3.4 Module 2.4: Conversational Agents and Memory . . . 22
3.4.1 Overview . . . 22
3.4.2 Reading Assignments . . . 23
3.4.3 Memory Architectures . . . 23
3.4.4 Practical Exercises . . . 23
3.5 Phase 2 Capstone Projects . . . 24

4 Phase 3: Full-Stack AI/LLM Engineering 25
4.1 Module 3.1: System Design for LLM Applications . . . 25
4.1.1 Overview . . . 25
4.1.2 Reading Assignments . . . 26
4.1.3 Decision Frameworks . . . 26
4.1.4 Practical Exercises . . . 27
4.2 Module 3.2: Evaluation and Quality Assurance at Scale . . . 27
4.2.1 Overview . . . 27
4.2.2 Reading Assignments . . . 28
4.2.3 Evaluation Strategies . . . 28
4.2.4 Practical Exercises . . . 28
4.3 Module 3.3: LLMOps and Production Operations . . . 28
4.3.1 Overview . . . 28
4.3.2 Reading Assignments . . . 29
4.3.3 Operational Practices . . . 29
4.3.4 Practical Exercises . . . 30
4.4 Module 3.4: Safety, Security, and Ethics . . . 30
4.4.1 Overview . . . 30
4.4.2 Reading Assignments . . . 30
4.4.3 Security Considerations . . . 31
4.4.4 Practical Exercises . . . 31
4.5 Phase 3 Capstone Project . . . 32

5 Resources and References 33
5.1 Core Textbooks . . . 33
5.1.1 Pure Prompting Focus . . . 33
5.1.2 Broader AI/LLM Engineering . . . 33
5.2 Official Documentation . . . 33
5.3 Frameworks and Tools . . . 34
5.4 Research Papers . . . 34

6 Assessment Framework 35
6.1 Assessment Philosophy . . . 35
6.2 Phase Assessments . . . 35
6.3 Competency Levels . . . 36

7 Schedule and Pacing Guide 37
7.1 Standard 12-Week Schedule . . . 37
7.2 Accelerated 8-Week Schedule . . . 37
7.3 Self-Paced Guidelines . . . 37

A Prompt Pattern Quick Reference **39**
A.1 Fundamental Patterns 39
A.2 Advanced Patterns 39

B Tool Setup Guides **40**
B.1 Python Environment 40
B.2 API Key Management 40

C Project Templates **41**
C.1 Standard Project Structure 41

Chapter 1

Executive Summary

1.1 Course Overview

This comprehensive curriculum provides a structured pathway for software engineers to master prompt engineering and build production-ready applications powered by Large Language Models (LLMs). The program is designed around three progressive phases that build upon each other, taking learners from fundamental prompting techniques through application development to full-stack AI engineering practices.

The curriculum synthesizes knowledge from the most authoritative resources in the field, including seminal texts such as *Prompt Engineering for Generative AI*, *AI Engineering: Building Applications with Foundation Models*, and the *LLM Engineer's Handbook*, complemented by continuously updated documentation from leading AI providers including OpenAI, Anthropic, and Google.

1.2 Target Audience

This curriculum is designed for:

- **Software Engineers** seeking to integrate LLM capabilities into applications
- **Backend/Full-Stack Developers** building AI-powered products
- **Technical Product Managers** requiring deep understanding of LLM systems
- **ML Engineers** transitioning to LLM-focused roles
- **Technical Writers and Content Creators** working with generative AI

1.3 Prerequisites

Category	**Requirements**
Programming	Proficiency in Python (intermediate level); familiarity with REST APIs
Development	Basic understanding of version control (Git); experience with package managers
Concepts	Fundamental understanding of machine learning concepts (helpful but not required)
Tools	Access to OpenAI API, Anthropic API, or similar LLM providers
Mindset	Willingness to experiment iteratively and embrace empirical approaches

1.4 Program Structure

Phase	Duration	Focus Area	Outcome
Phase 1	2–4 weeks	Core prompting skills, patterns, and evaluation	Personal prompt cookbook
Phase 2	4–6 weeks	Building LLM-powered applications and agents	2–3 portfolio projects
Phase 3	4–6 weeks	Production systems, MLOps, and engineering practices	Production playbook

1.5 Learning Outcomes

Upon successful completion of this curriculum, learners will be able to:

1. Design and implement effective prompts across text, code, and multimodal applications
2. Apply systematic prompt patterns including chain-of-thought, few-shot learning, and critique-and-revise
3. Build functional LLM applications incorporating retrieval-augmented generation (RAG)
4. Implement tool use and function calling within LLM systems
5. Evaluate prompt and system quality using appropriate metrics and methodologies
6. Deploy LLM applications with proper monitoring, feedback loops, and cost management
7. Make informed architectural decisions between prompting, fine-tuning, and RAG approaches
8. Navigate the trade-offs between different model providers and deployment strategies

Chapter 2

Phase 1: Core Prompting Skills

🕓 Estimated Time

Duration: 2–4 weeks
Study Hours: 20–40 hours total
Recommended Pace: 8–12 hours per week

◎ Learning Objectives

By the end of Phase 1, learners will:

- Reliably obtain high-quality outputs from GPT, Claude, Gemini, and similar models
- Apply 20–30 distinct prompting patterns with confidence
- Understand the principles underlying effective prompt construction
- Evaluate and iteratively improve prompt performance
- Construct prompts for text, code, and image generation tasks

2.1 Module 1.1: Foundations of Prompt Engineering

2.1.1 Overview

This foundational module establishes the theoretical and practical groundwork for all subsequent learning. Students will develop an intuitive understanding of how language models process and respond to prompts, along with the fundamental techniques that form the basis of effective prompting.

Key Concepts

- How LLMs process text and generate responses (tokenization, attention, sampling)
- The prompt-completion paradigm and its implications
- Zero-shot vs. few-shot prompting fundamentals
- The role of context windows and token limits
- Temperature, top-p, and other generation parameters

2.1.2 Reading Assignments

Resources

Primary Reading:

1. *The Art of Prompt Engineering with ChatGPT* – Chapters 1–3 (Core Techniques)
2. OpenAI Prompt Engineering Guide – Complete reading
3. Anthropic Prompt Engineering Documentation – Overview and Best Practices sections

Supplementary Reading:

1. DAIR.AI Prompt Engineering Guide – Introduction section
2. Google Cloud Prompt Engineering Guide – Fundamentals

2.1.3 Topics Covered

Understanding Language Model Behavior

Students will explore the fundamental mechanisms that govern how language models interpret and respond to prompts:

- **Tokenization:** How text is converted to tokens and why token boundaries matter for prompt design
- **Context Windows:** Working within token limits; strategies for context management
- **Probabilistic Generation:** Understanding that outputs are sampled from probability distributions
- **Instruction Following:** How models are trained to follow instructions (RLHF, Constitutional AI)
- **Model Capabilities and Limitations:** Realistic expectations for different model sizes and types

Core Prompting Techniques

Technique	Description and Application
Role Prompting	Assigning a persona or expert role to the model (e.g., "You are a senior software architect...") to influence response style and depth
Instruction Clarity	Writing unambiguous, specific instructions; avoiding vague language; being explicit about format and scope
Constraint Specification	Defining boundaries, length limits, style requirements, and output formats
Context Setting	Providing relevant background information; establishing scope; defining terminology
Output Formatting	Specifying JSON, XML, Markdown, or custom formats; using delimiters and structure
Positive/Negative Constraints	Explicitly stating what to include and what to avoid
Task Decomposition	Breaking complex requests into clear, sequential steps

2.1.4 Practical Exercises

Practical Exercise

Exercise 1.1.1: Vague to Precise Transformation
Take the following vague prompts and transform them into precise, well-structured prompts. Document your reasoning for each transformation.

1. "Write something about climate change"
2. "Help me with my code"
3. "Explain machine learning"
4. "Write a marketing email"

Deliverable: For each prompt, provide the improved version and a 2–3 sentence explanation of the changes made and why.

Practical Exercise

Exercise 1.1.2: Role Prompting Comparison
Select a technical topic you're familiar with. Write three versions of a prompt asking for an explanation:

1. No role specified
2. Role: "You are a college professor"
3. Role: "You are an experienced practitioner teaching a junior colleague"

Compare the outputs systematically: tone, depth, examples used, accessibility.
Deliverable: Written analysis (500–700 words) comparing the outputs and identifying when each role framing would be most appropriate.

Practical Exercise

Exercise 1.1.3: Generation Parameter Exploration
Using the same prompt, generate outputs with different temperature settings (0.0, 0.5, 0.7, 1.0) and document the differences. Repeat with different top-p values.
Deliverable: Table of results with observations about creativity vs. consistency trade-offs.

2.2 Module 1.2: Advanced Prompting Patterns

2.2.1 Overview

Building on foundational techniques, this module introduces sophisticated patterns that enable more complex reasoning and higher-quality outputs from language models.

Key Concepts

- Chain-of-thought (CoT) prompting and its variants
- Few-shot learning and example selection strategies
- Self-consistency and ensemble approaches
- Critique-and-revise patterns
- Tree-of-thought and advanced reasoning structures
- Prompt chaining and decomposition

2.2.2 Reading Assignments

Resources

Primary Reading:

1. *Prompt Engineering for Generative AI* – Chapters on Five Principles of Prompting
2. *Prompt Engineering for Generative AI* – Chapters on evaluation and multi-modal prompts
3. DAIR.AI Guide – Chain-of-Thought Prompting section

Supplementary Reading:

1. Original CoT paper: "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models"
2. "Self-Consistency Improves Chain of Thought Reasoning in Language Models"

2.2.3 Topics Covered

Chain-of-Thought Prompting

Chain-of-thought (CoT) prompting represents one of the most significant advances in prompt engineering, enabling language models to solve complex reasoning problems by explicitly generating intermediate steps.

CoT Variant	Description
Zero-shot CoT	Adding “Let’s think step by step” or similar phrases to trigger reasoning without examples
Few-shot CoT	Providing examples that demonstrate the reasoning process
Self-consistency	Generating multiple reasoning paths and selecting the most consistent answer
Tree-of-Thought	Exploring multiple reasoning branches and evaluating them
Program-aided CoT	Using code execution to verify or assist reasoning steps

Few-Shot Learning Strategies

- **Example Selection:** Choosing diverse, representative examples; avoiding bias
- **Example Ordering:** Impact of example sequence on model performance
- **Example Format:** Structuring input-output pairs consistently
- **Negative Examples:** When and how to include counterexamples
- **Dynamic Few-Shot:** Selecting examples based on input similarity

Iterative Refinement Patterns

Pattern	Application
Critique-and-Revise	Model generates output, then critiques and improves it
Multi-turn Refinement	Progressive improvement through conversation
Self-Evaluation	Model assesses its own output against criteria
Adversarial Prompting	Testing outputs for weaknesses and edge cases
Constitutional AI Patterns	Applying principles-based evaluation and revision

2.2.4 Practical Exercises

Practical Exercise

Exercise 1.2.1: Chain-of-Thought Implementation
Solve the following problem types using both zero-shot and few-shot CoT:

1. Multi-step arithmetic word problems
2. Logical reasoning puzzles
3. Code debugging scenarios
4. Strategic decision analysis

Compare accuracy and reasoning quality between approaches.
Deliverable: Documented prompts and outputs with comparative analysis.

Practical Exercise

Exercise 1.2.2: Few-Shot Example Engineering
Create a few-shot prompt for a text classification task (e.g., sentiment analysis, intent detection). Systematically vary:

1. Number of examples (1, 3, 5, 10)
2. Example diversity
3. Example ordering

Deliverable: Performance analysis across variations with recommendations.

Practical Exercise

Exercise 1.2.3: Critique-and-Revise Pipeline
Build a three-stage prompt pipeline:

1. Initial generation of a technical document (API documentation, tutorial, etc.)
2. Self-critique against specified criteria (clarity, completeness, accuracy)
3. Revision incorporating critique feedback

Deliverable: Complete pipeline with prompts at each stage; before/after comparison.

2.3 Module 1.3: Prompt Evaluation and Quality Assurance

2.3.1 Overview

Systematic evaluation is critical for developing reliable prompts. This module covers methodologies for assessing prompt quality, establishing baselines, and implementing continuous improvement processes.

Key Concepts

- Evaluation metrics for different task types
- Creating golden test sets
- A/B testing prompts
- Automated evaluation pipelines
- Human evaluation protocols
- Regression testing for prompts

2.3.2 Reading Assignments

Resources

Primary Reading:

1. *Prompt Engineering for LLMs* – Evaluation chapters
2. Anthropic Documentation – Prompt evaluation best practices

Supplementary Reading:

1. "Evaluating Large Language Models: A Comprehensive Survey"
2. OpenAI Evals framework documentation

2.3.3 Evaluation Methodologies

Quantitative Metrics

Metric Type	**Examples**	**Best Used For**
Exact Match	Accuracy, F1	Classification, factual QA
Text Similarity	BLEU, ROUGE, BERTScore	Summarization, translation
Semantic Similarity	Embedding cosine similarity	Open-ended generation
Task-Specific	Code execution pass rate, SQL validity	Code generation, structured output
LLM-as-Judge	GPT-4 evaluation scores	Complex, nuanced outputs

Qualitative Assessment

- **Rubric Development:** Creating consistent scoring criteria
- **Inter-rater Reliability:** Ensuring evaluation consistency
- **Error Taxonomy:** Categorizing failure modes
- **Edge Case Identification:** Systematic boundary testing

2.3.4 Practical Exercises

Practical Exercise

Exercise 1.3.1: Golden Test Set Creation
For a prompt you've developed in previous exercises:

1. Create a test set of 20+ input-expected output pairs
2. Include edge cases and adversarial examples
3. Document expected behavior for each case
4. Implement automated evaluation script

Deliverable: Test set JSON/YAML file plus evaluation script.

Practical Exercise

Exercise 1.2.2: Evaluation Dashboard
Build a simple evaluation dashboard that:

1. Runs prompts against test cases
2. Calculates relevant metrics
3. Visualizes results
4. Tracks changes over prompt iterations

Deliverable: Working dashboard (Streamlit, Jupyter, or similar).

2.4 Module 1.4: Multi-Modal and Specialized Prompting

2.4.1 Overview

This module extends core prompting skills to specialized domains including code generation, image understanding, and multi-modal applications.

Key Concepts

- Code generation and completion prompting
- Image understanding and visual prompting
- Multi-modal prompt construction (text + image)
- Domain-specific adaptations (legal, medical, technical)
- Structured output generation (JSON, XML, YAML)

2.4.2 Reading Assignments

Resources

Primary Reading:

1. *Prompt Engineering for Generative AI* – Multi-modal chapters
2. *The Art of Prompt Engineering with ChatGPT* – Code generation sections

Supplementary Reading:

1. OpenAI Vision API documentation
2. Anthropic Claude vision capabilities guide

2.4.3 Code Generation Prompting

Effective Patterns for Code

Pattern	**Application**
Specification-First	Provide detailed requirements before requesting implementation
Pseudo-code Scaffolding	Outline structure before requesting full implementation
Test-Driven Prompting	Provide test cases; ask for implementation that passes them
Incremental Development	Build functionality step-by-step through conversation
Code Review Prompting	Ask for analysis and improvement of existing code
Documentation Generation	Generate docstrings, comments, and README content

2.4.4 Practical Exercises

Practical Exercise

Exercise 1.4.1: Code Generation Pipeline
Create a prompt workflow that:

1. Generates code from natural language specification
2. Produces corresponding unit tests
3. Generates documentation
4. Reviews for security issues

Deliverable: Complete prompt chain with example outputs.

Practical Exercise

Exercise 1.4.2: Multi-Modal Analysis
Using a vision-capable model:

1. Analyze technical diagrams and generate descriptions
2. Extract data from charts and tables in images
3. Generate code from UI mockups
4. Create test cases from screenshot of application state

Deliverable: Portfolio of multi-modal prompt examples.

2.5 Phase 1 Capstone Project

Deliverable

Personal Prompt Cookbook
Compile a comprehensive prompt cookbook containing:

1. **20–30 documented prompt patterns** with:
   - Pattern name and description
   - Use cases and best applications
   - Template with placeholders
   - 2–3 concrete examples
   - Known limitations and failure modes
2. **Evaluation framework** including:
   - Test cases for each pattern
   - Automated evaluation scripts
   - Quality metrics and thresholds
3. **Personal best practices** derived from experimentation
4. **Provider comparison notes** (GPT vs. Claude vs. Gemini)

Format: GitHub repository with Markdown documentation and supporting code.

Chapter 3

Phase 2: From Prompts to Applications

🕓 **Estimated Time**

Duration: 4–6 weeks
Study Hours: 40–60 hours total
Recommended Pace: 10–12 hours per week

◎ **Learning Objectives**

By the end of Phase 2, learners will:

- Build functional LLM-powered applications
- Implement tool use and function calling
- Design and deploy RAG (Retrieval-Augmented Generation) systems
- Construct conversational agents with memory
- Integrate LLMs with external data sources and APIs
- Apply proper architecture patterns for LLM applications

3.1 Module 2.1: LLM Application Architecture

3.1.1 Overview

Transitioning from prompts to applications requires understanding how LLMs fit into broader system architectures. This module covers design patterns, component interactions, and architectural decisions.

Key Concepts

- LLM application architecture patterns
- API design for LLM services
- State management and conversation handling
- Caching and performance optimization
- Error handling and graceful degradation
- Security considerations in LLM applications

3.1.2 Reading Assignments

Resources

Primary Reading:

1. *Prompt Engineering for LLMs* – Application architecture chapters
2. *Building LLM Powered Applications* – Chapters 1–4
3. *AI Engineering* – System design chapters

Supplementary Reading:

1. LangChain documentation – Architecture section
2. LlamaIndex documentation – Core concepts

3.1.3 Architecture Patterns

Common Application Architectures

Pattern	**Description**	**Use Cases**
Direct API	Simple prompt-response through API	Single-turn tasks, simple chatbots
Chain of Prompts	Sequential prompt execution	Multi-step workflows, complex reasoning
Router Pattern	Classifier directing to specialized prompts	Multi-intent systems, varied task types
RAG Pipeline	Retrieval + generation combination	Knowledge-based QA, document analysis
Agent Loop	Autonomous action-observation cycle	Complex task completion, tool use
Human-in-the-Loop	LLM assistance with human verification	High-stakes decisions, content moderation

Component Design

Key components in LLM applications include:

- **Prompt Templates:** Parameterized, versioned prompt management
- **Context Managers:** Conversation history and context window optimization
- **Response Parsers:** Structured output extraction and validation
- **Retry Logic:** Handling rate limits, timeouts, and transient failures
- **Caching Layer:** Response caching for cost and latency optimization
- **Logging/Telemetry:** Request/response logging for debugging and analytics

3.1.4 Practical Exercises

Practical Exercise

Exercise 2.1.1: Architecture Design Document
Choose a hypothetical LLM application (e.g., customer support bot, code review assistant, research summarizer). Create an architecture design document including:

1. System context diagram
2. Component diagram
3. Data flow diagram
4. API specifications
5. Error handling strategy
6. Scaling considerations

Deliverable: Architecture document (Markdown or PDF).

3.2 Module 2.2: Tool Use and Function Calling

3.2.1 Overview

Tool use enables LLMs to interact with external systems, execute code, and access real-time information. This module covers implementation patterns and best practices.

Key Concepts

- Function calling APIs (OpenAI, Anthropic, Google)
- Tool schema design and documentation
- Parallel and sequential tool execution
- Error handling in tool chains
- Security considerations for tool access
- Building custom tools and integrations

3.2.2 Reading Assignments

Resources

Primary Reading:

1. *Prompt Engineering for LLMs* – Tool use chapters
2. *Building LLM Powered Applications* – Tool integration sections
3. OpenAI Function Calling documentation
4. Anthropic Tool Use documentation

3.2.3 Implementation Patterns

Tool Definition Best Practices

Aspect	**Best Practice**
Naming	Clear, action-oriented names (e.g., `search_database`, `send_email`)
Descriptions	Detailed descriptions of when and how to use each tool
Parameters	Well-typed parameters with clear descriptions and examples
Return Values	Consistent, parseable return formats
Error Messages	Informative error messages the LLM can interpret
Scope Limitation	Minimal necessary permissions; principle of least privilege

3.2.4 Practical Exercises

Practical Exercise

Exercise 2.2.1: Multi-Tool Agent
Build an agent with access to multiple tools:

1. Web search tool
2. Calculator tool
3. File read/write tool
4. API call tool (weather, stocks, etc.)

Implement proper tool selection, execution, and result integration.
Deliverable: Working multi-tool agent with documentation.

Practical Exercise

Exercise 2.2.2: SQL Query Agent
Create an agent that:

1. Accepts natural language questions about a database
2. Generates appropriate SQL queries
3. Executes queries safely
4. Formats and explains results

Include proper SQL injection prevention and query validation.
Deliverable: Working SQL agent with test database.

3.3 Module 2.3: Retrieval-Augmented Generation (RAG)

3.3.1 Overview

RAG combines the power of retrieval systems with generative models, enabling LLMs to access and reason over large document collections while reducing hallucinations.

Key Concepts

- RAG architecture and components
- Document chunking strategies
- Embedding models and vector stores
- Retrieval algorithms and reranking
- Context window management
- Hybrid search approaches
- RAG evaluation methodologies

3.3.2 Reading Assignments

Resources

Primary Reading:

1. *Building LLM Powered Applications* – RAG chapters
2. *LLM Engineer's Handbook* – RAG architecture sections
3. LlamaIndex documentation – RAG pipelines

Supplementary Reading:

1. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" (original paper)
2. Pinecone RAG best practices guide

3.3.3 RAG Pipeline Components

Document Processing

Stage	Considerations	Tools/Techniques
Ingestion	Format handling, metadata extraction	PDF parsers, HTML extractors
Chunking	Size, overlap, semantic boundaries	Recursive splitting, semantic chunking
Embedding	Model selection, dimensionality	OpenAI, Cohere, open-source models
Storage	Vector database selection	Pinecone, Weaviate, Chroma, pgvector
Indexing	Index type, HNSW parameters	Approximate nearest neighbor configs

Retrieval Strategies

- **Dense Retrieval:** Embedding-based semantic search
- **Sparse Retrieval:** BM25 and keyword-based approaches
- **Hybrid Search:** Combining dense and sparse methods
- **Reranking:** Cross-encoder models for relevance refinement
- **Query Expansion:** Generating multiple query variants
- **Hypothetical Document Embedding (HyDE):** Generating hypothetical answers for retrieval

3.3.4 Practical Exercises

Practical Exercise

Exercise 2.3.1: Document Q&A System
Build a RAG-based Q&A system:

1. Ingest a collection of technical documents (PDFs, markdown)
2. Implement chunking with multiple strategies
3. Set up vector storage and retrieval
4. Create a conversational interface
5. Add source citation to responses

Deliverable: Working RAG application with evaluation on test questions.

Practical Exercise

Exercise 2.3.2: RAG Optimization Study
Using the system from Exercise 2.3.1:

1. Compare different chunking strategies (fixed size, semantic, recursive)
2. Compare embedding models
3. Test hybrid retrieval vs. dense-only
4. Implement and evaluate reranking

Deliverable: Comparative analysis report with metrics.

3.4 Module 2.4: Conversational Agents and Memory

3.4.1 Overview

Building effective conversational agents requires sophisticated memory management and context handling across multi-turn interactions.

Key Concepts

- Conversation state management
- Short-term and long-term memory patterns
- Context summarization and compression
- Entity extraction and tracking
- Persona consistency
- Multi-turn dialogue flow

3.4.2 Reading Assignments

Resources

Primary Reading:

1. *Building LLM Powered Applications* – Conversational AI chapters
2. *Prompt Engineering for LLMs* – Memory and state sections
3. LangChain Memory documentation

3.4.3 Memory Architectures

Memory Type	**Implementation**	**Use Cases**
Buffer Memory	Full conversation history	Short conversations
Window Memory	Last N turns	Medium-length chats
Summary Memory	Summarized history	Long conversations
Entity Memory	Tracked entities and attributes	Customer service, personal assistants
Vector Memory	Semantically retrieved history	Knowledge workers, research assistants
Hybrid	Combined approaches	Production systems

3.4.4 Practical Exercises

Practical Exercise

Exercise 2.4.1: Customer Support Agent
Build a customer support chatbot with:

1. Product knowledge base (RAG)
2. Conversation memory
3. Ticket creation capability (tool use)
4. Escalation logic
5. Session persistence

Deliverable: Deployed chatbot with conversation logs analysis.

3.5 Phase 2 Capstone Projects

🏁 Deliverable

Build 2–3 Portfolio-Quality Applications
Complete at least two of the following:
Project A: Conversational Application

- Multi-turn chatbot with memory
- Persona consistency
- Conversation analytics dashboard

Project B: RAG/Search Application

- Document ingestion pipeline
- Semantic search interface
- Answer with citations
- Evaluation metrics

Project C: Structured Data Application

- Natural language to SQL/API
- Result visualization
- Query explanation

Project D: Agentic Application

- Multi-tool agent
- Planning and execution
- Human-in-the-loop for critical actions

Requirements for each project:

- Prompts stored in templates/configuration
- Evaluation scripts with golden test cases
- Documentation (README, architecture)
- Multi-provider support (test with OpenAI, Anthropic, etc.)

Chapter 4

Phase 3: Full-Stack AI/LLM Engineering

Estimated Time

Duration: 4–6 weeks (ongoing thereafter)
Study Hours: 40–60 hours total
Recommended Pace: 10–12 hours per week

Learning Objectives

By the end of Phase 3, learners will:

- Make informed decisions between prompting, fine-tuning, and RAG
- Design comprehensive evaluation and monitoring systems
- Deploy LLM applications with proper observability
- Manage costs, latency, and reliability in production
- Implement feedback loops for continuous improvement
- Address safety, security, and policy considerations

4.1 Module 3.1: System Design for LLM Applications

4.1.1 Overview

Production LLM systems require careful consideration of trade-offs between cost, latency, quality, and reliability. This module covers system design principles specific to AI applications.

Key Concepts

- Prompting vs. fine-tuning vs. RAG decision framework
- Model selection criteria
- Latency optimization strategies
- Cost management and budgeting
- Scalability patterns
- Reliability and redundancy

4.1.2 Reading Assignments

Resources

Primary Reading:

1. *AI Engineering: Building Applications with Foundation Models* – Complete book
2. *LLM Engineer's Handbook* – System design chapters

Supplementary Reading:

1. "The Shift from Models to Compound AI Systems" (Berkeley AI Research)
2. Papers on model distillation and efficiency

4.1.3 Decision Frameworks

When to Use Each Approach

Approach	Best For	Limitations	Cost Profile
Prompting	Rapid iteration, general tasks	Token limits, consistency	Per-token, predictable
Fine-tuning	Consistent style/format, domain adaptation	Training cost, data requirements	Upfront + inference
RAG	Dynamic knowledge, attribution	Retrieval quality, latency	Storage + retrieval + inference
Hybrid	Complex requirements	System complexity	Combined costs

Model Selection Criteria

- **Capability:** Task performance, reasoning ability, instruction following
- **Latency:** Time to first token, throughput requirements
- **Cost:** Input/output token pricing, volume discounts
- **Context Length:** Maximum context window, effective context utilization

- **Reliability:** API availability, rate limits, SLAs
- **Features:** Tool use support, vision, streaming, structured outputs

4.1.4 Practical Exercises

Practical Exercise

Exercise 3.1.1: System Design Review
Take one of your Phase 2 projects and conduct a production readiness review:

1. Cost analysis at various usage levels
2. Latency profiling and optimization opportunities
3. Failure mode analysis
4. Scaling plan
5. Provider redundancy strategy

Deliverable: Production readiness document.

4.2 Module 3.2: Evaluation and Quality Assurance at Scale

4.2.1 Overview

Production systems require comprehensive evaluation beyond simple accuracy metrics. This module covers evaluation strategies for complex, open-ended LLM outputs.

Key Concepts

- Evaluation taxonomy (automatic, human, LLM-as-judge)
- Building evaluation datasets
- Continuous evaluation pipelines
- A/B testing for LLM applications
- Detecting regression and drift
- Safety and alignment evaluation

4.2.2 Reading Assignments

Resources

Primary Reading:

1. *AI Engineering* – Evaluation chapters
2. *LLM Engineer's Handbook* – Quality assurance sections

Supplementary Reading:

1. OpenAI Evals framework and methodology
2. LangSmith/LangFuse evaluation documentation

4.2.3 Evaluation Strategies

Multi-Level Evaluation

Level	What to Measure	How to Measure
Component	Individual prompt quality	Unit tests, golden sets
Pipeline	End-to-end correctness	Integration tests, trace analysis
System	User-facing quality	A/B tests, user feedback
Business	Impact on metrics	Conversion, satisfaction, efficiency

4.2.4 Practical Exercises

Practical Exercise

Exercise 3.2.1: Comprehensive Evaluation Pipeline
Build an evaluation system that:

1. Runs nightly against test sets
2. Includes automatic metrics (BLEU, semantic similarity)
3. Incorporates LLM-as-judge evaluations
4. Generates reports and alerts
5. Tracks metrics over time

Deliverable: Working evaluation pipeline with dashboard.

4.3 Module 3.3: LLMOps and Production Operations

4.3.1 Overview

Operating LLM systems in production requires specialized practices for monitoring, observability, and continuous improvement.

Key Concepts

- LLM observability and tracing
- Prompt versioning and management
- Cost monitoring and optimization
- Feedback collection and processing
- Incident response for LLM systems
- Continuous improvement loops

4.3.2 Reading Assignments

Resources

Primary Reading:

1. *LLM Engineer's Handbook* – Operations chapters
2. *AI Engineering* – Deployment and monitoring sections

Supplementary Reading:

1. LangSmith documentation
2. Helicone, Portkey observability guides

4.3.3 Operational Practices

Observability Stack

Component	Purpose	Tools
Request Logging	Audit trail, debugging	Custom logging, LangSmith
Tracing	Multi-step execution visibility	LangSmith, Phoenix, Jaeger
Metrics	Performance monitoring	Prometheus, DataDog, custom
Cost Tracking	Budget management	Provider dashboards, Helicone
Quality Monitoring	Output quality over time	LLM-based evaluation, human review

4.3.4 Practical Exercises

Practical Exercise

Exercise 3.3.1: Observability Implementation
Add comprehensive observability to a Phase 2 project:

1. Request/response logging
2. Latency and token tracking
3. Cost dashboards
4. Error alerting
5. Quality score tracking

Deliverable: Instrumented application with observability dashboard.

4.4 Module 3.4: Safety, Security, and Ethics

4.4.1 Overview

Responsible deployment of LLM applications requires attention to safety, security vulnerabilities, and ethical considerations.

Key Concepts

- Prompt injection and jailbreaking
- Output filtering and safety layers
- Data privacy in LLM applications
- Bias detection and mitigation
- Regulatory considerations (GDPR, AI Act)
- Responsible AI practices

4.4.2 Reading Assignments

Resources

Primary Reading:

1. *AI Engineering* – Safety and ethics chapters
2. OWASP LLM Top 10
3. Anthropic responsible scaling policy

Supplementary Reading:

1. NeMo Guardrails documentation
2. Constitutional AI paper

4.4.3 Security Considerations

Common Vulnerabilities

Vulnerability	Description	Mitigation
Prompt Injection	Malicious input overriding instructions	Input validation, separate contexts
Data Leakage	Exposing training or user data	Output filtering, access controls
Excessive Agency	Unintended autonomous actions	Permission boundaries, confirmations
Model Theft	Extracting model via API	Rate limiting, watermarking
Insecure Output	XSS, SQL injection via output	Output sanitization, type checking

4.4.4 Practical Exercises

Practical Exercise

Exercise 3.4.1: Security Hardening
Take a Phase 2 project and implement:

1. Input validation layer
2. Output safety filtering
3. Prompt injection testing suite
4. Rate limiting
5. Audit logging

Deliverable: Hardened application with security documentation.

4.5 Phase 3 Capstone Project

🏁 Deliverable

Production Playbook
Create a comprehensive playbook that answers, for any use case:
Part 1: Decision Framework

- When to use better prompts vs. RAG vs. fine-tuning
- Model selection decision tree
- Architecture pattern selection guide

Part 2: Quality Management

- Evaluation strategy templates
- Testing checklists
- Quality metrics definitions

Part 3: Operations

- Monitoring setup guides
- Incident response procedures
- Cost management strategies

Part 4: Safety

- Security checklists
- Compliance considerations
- Responsible AI guidelines

Format: Notion, Confluence, or GitHub wiki with templates and checklists.

Chapter 5

Resources and References

5.1 Core Textbooks

5.1.1 Pure Prompting Focus

Title	Authors	Best For
Prompt Engineering for Generative AI	Phoenix & Taylor	Structured prompting playbook for developers
The Art of Prompt Engineering with ChatGPT	—	Beginners seeking practical skills quickly
Prompt Engineering for LLMs: The Art and Science of Building LLM-Based Applications	Berryman & Ziegler	Building robust LLM systems

5.1.2 Broader AI/LLM Engineering

Title	Authors	Best For
AI Engineering: Building Applications with Foundation Models	Chip Huyen	Full lifecycle from idea to production
LLM Engineer's Handbook	Iusztin & Labonne	Production-grade infrastructure and ops
Building LLM Powered Applications	Valentina Alto	Hands-on projects with LangChain

5.2 Official Documentation

Provider	Resource	URL
OpenAI	Prompt Engineering Guide	platform.openai.com/docs
Anthropic	Prompt Engineering Documentation	docs.anthropic.com

Provider	Resource	URL
Google	Vertex AI Prompt Design	cloud.google.com/vertex-ai
DAIR.AI	Prompt Engineering Guide	promptingguide.ai

5.3 Frameworks and Tools

Tool	Purpose	Documentation
LangChain	LLM application framework	langchain.com/docs
LlamaIndex	Data framework for LLM apps	docs.llamaindex.ai
LangSmith	LLM observability	docs.smith.langchain.com
Pinecone	Vector database	docs.pinecone.io
Chroma	Open-source vector store	docs.trychroma.com
NeMo Guardrails	Safety guardrails	github.com/NVIDIA/NeMo-Guardrails

5.4 Research Papers

Key papers to understand for advanced topics:

1. “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models” (Wei et al., 2022)
2. “Self-Consistency Improves Chain of Thought Reasoning” (Wang et al., 2022)
3. “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks” (Lewis et al., 2020)
4. “Constitutional AI: Harmlessness from AI Feedback” (Bai et al., 2022)
5. “Tree of Thoughts: Deliberate Problem Solving with LLMs” (Yao et al., 2023)
6. “ReAct: Synergizing Reasoning and Acting in Language Models” (Yao et al., 2022)

Chapter 6

Assessment Framework

6.1 Assessment Philosophy

This curriculum emphasizes practical, project-based assessment over traditional examinations. Learners demonstrate competency through building, documenting, and presenting working systems.

6.2 Phase Assessments

✓ Assessment Criteria

Phase 1 Assessment (25% of total)

- **Prompt Cookbook (70%):** Quality, completeness, and originality of documented patterns
- **Exercise Completion (20%):** Timely completion with thoughtful analysis
- **Peer Review Participation (10%):** Providing constructive feedback on others' prompts

Rubric Criteria:

- Pattern variety and coverage
- Quality of examples and documentation
- Evaluation methodology rigor
- Personal insights and discoveries

Assessment Criteria

Phase 2 Assessment (40% of total)

- **Project Quality (60%):** Functionality, code quality, and architecture
- **Documentation (20%):** README, architecture diagrams, and setup instructions
- **Evaluation Implementation (20%):** Test coverage and evaluation scripts

Rubric Criteria:

- Working functionality
- Clean, maintainable code
- Comprehensive documentation
- Thoughtful evaluation approach
- Multi-provider compatibility

Assessment Criteria

Phase 3 Assessment (35% of total)

- **Playbook Quality (50%):** Completeness, practicality, and clarity
- **Production Enhancement (30%):** Observability, security, and operations improvements
- **Presentation (20%):** Ability to explain decisions and trade-offs

Rubric Criteria:

- Decision framework clarity
- Operational readiness
- Security awareness
- Business context understanding

6.3 Competency Levels

Level	Description
Foundational	Can write effective prompts for common tasks; understands basic patterns
Intermediate	Can build LLM applications; implements RAG and tool use; evaluates systematically
Advanced	Makes production-ready architectural decisions; manages operations; addresses safety
Expert	Designs novel systems; optimizes at scale; contributes to organizational practices

Chapter 7

Schedule and Pacing Guide

7.1 Standard 12-Week Schedule

Week	Phase	Focus
1	Phase 1	Module 1.1: Foundations
2		Module 1.2: Advanced Patterns
3		Module 1.3–1.4: Evaluation & Specialized Prompting
4	Phase 2	Module 2.1: Application Architecture
5		Module 2.2: Tool Use
6		Module 2.3: RAG Systems
7		Module 2.4: Conversational Agents
8		Project Work
9	Phase 3	Module 3.1: System Design
10		Module 3.2: Evaluation at Scale
11		Module 3.3–3.4: Operations & Safety
12		Capstone Completion

7.2 Accelerated 8-Week Schedule

For experienced developers who can commit 15–20 hours per week:

Week	Phase	Focus
1	Phase 1	Modules 1.1–1.2
2		Modules 1.3–1.4 + Cookbook
3	Phase 2	Modules 2.1–2.2
4		Module 2.3
5		Module 2.4 + Project 1
6		Project 2
7	Phase 3	Modules 3.1–3.2
8		Modules 3.3–3.4 + Playbook

7.3 Self-Paced Guidelines

For those learning independently:

- **Minimum commitment:** 5 hours per week
- **Recommended:** Complete Phase 1 before moving to Phase 2
- **Project-first approach:** Consider starting projects early and learning as needed
- **Community:** Join Discord/Slack communities for peer support
- **Accountability:** Set weekly goals and track progress

Appendix A

Prompt Pattern Quick Reference

A.1 Fundamental Patterns

Pattern	**Trigger**	**Template**
Role Prompting	Need expert perspective	"You are a [role] with expertise in [domain]..."
Few-Shot	Need consistent format	"Here are examples: [Ex1] [Ex2] Now do: [Task]"
Chain-of-Thought	Complex reasoning	"Let's think through this step by step..."
Output Format	Structured output needed	"Respond in the following JSON format: {...}"
Constraint List	Specific requirements	"Requirements: 1. ... 2. ... 3. ..."

A.2 Advanced Patterns

Pattern	**Trigger**	**Template**
Self-Consistency	High-stakes decisions	Generate N responses, select most consistent
Critique-Revise	Quality improvement	"Critique the above, then provide improved version"
Decomposition	Complex multi-part task	"Break this into subtasks: [Task]"
Hypothetical	Creative exploration	"Imagine you are [scenario]. How would you..."
Meta-Prompting	Prompt improvement	"How could this prompt be improved?"

Appendix B

Tool Setup Guides

B.1 Python Environment

```
# Create virtual environment
python -m venv llm-curriculum
source llm-curriculum/bin/activate  # Unix
llm-curriculum\Scripts\activate     # Windows

# Install core packages
pip install openai anthropic google-generativeai
pip install langchain langchain-openai langchain-anthropic
pip install chromadb pinecone-client
pip install pandas numpy jupyter
pip install pytest black mypy
```

B.2 API Key Management

```
# .env file (never commit!)
OPENAI_API_KEY=sk-...
ANTHROPIC_API_KEY=sk-ant-...
GOOGLE_API_KEY=...

# Python usage
from dotenv import load_dotenv
import os

load_dotenv()
api_key = os.getenv("OPENAI_API_KEY")
```

Appendix C

Project Templates

C.1 Standard Project Structure

```
project-name/
|-- README.md
|-- requirements.txt
|-- .env.example
|-- src/
|   |-- __init__.py
|   |-- prompts/
|   |   |-- __init__.py
|   |   +-- templates.py
|   |-- chains/
|   |-- tools/
|   +-- utils/
|-- tests/
|   |-- __init__.py
|   |-- test_prompts.py
|   +-- golden_tests/
|-- evaluation/
|   |-- eval_scripts.py
|   +-- test_cases.json
|-- notebooks/
|   +-- exploration.ipynb
+-- docs/
    |-- architecture.md
    +-- api.md
```