💙 WECAMP

From Code to Cognition: Building Intelligent AI Agents with Python

Course Duration: 12 weeks × ~5 hours/week
Target Audience: Junior Python Developers
Primary Textbook: "Building Agentic AI Systems" by Biswas & Talukdar (Chapters 1-7)
Course Level: Intermediate

Course Overview

This hands-on course guides junior Python developers through the journey of building intelligent AI agents—from understanding foundational concepts to deploying multi-agent collaborative systems. Students will learn the principles of agentic AI, work with modern frameworks like CrewAI and LangGraph, and complete a capstone project demonstrating real-world application.

Prerequisites

- Proficiency in Python 3.8+ (functions, classes, decorators)
- Basic understanding of APIs and REST principles
- Familiarity with command-line tools
- Git for version control

Course Learning Outcomes

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

a. Explain the core principles of agency, autonomy, and intelligent behavior in AI systems
b. Design and implement single-agent and multi-agent systems using industry-standard frameworks
c. Integrate external tools and APIs into agentic workflows
d. Apply knowledge representation and reasoning techniques to agent design
e. Evaluate and debug agentic systems using appropriate methodologies
f. Build production-ready AI agents following best practices for code quality and documentation
g. Select and implement appropriate workflow patterns (prompt chaining, routing, parallelization, orchestrator-workers, evaluator-optimizer) for specific use cases
h. Design effective agent-computer interfaces (ACI) through thoughtful tool documentation and testing
i. Apply prompt engineering fundamentals and advanced techniques to agent development
j. Understand the generative AI application lifecycle from prototype to production

Required Tools & Setup

- Python 3.8 or higher
- API access to OpenAI or Anthropic (educational credits provided)
- CrewAI and LangGraph frameworks
- Git and GitHub account
- IDE (VS Code recommended)

Assessment Structure

- **Practical Exercises (30%)**: Six hands-on coding assignments
- **Module Projects (30%)**: Four progressive projects building agent capabilities
- **Midterm Project (15%)**: Multi-tool agent system (Module 5)
- **Capstone Project (25%)**: Complete multi-agent application (Module 7)

Week-by-Week Reading Assignments

Weeks 1-2 (Module 1):

- Chapter 1: Fundamentals of Generative AI (pages 3-25)
- Focus: Introduction to generative AI, types of models, LLM-powered agents, framework landscape overview

Weeks 3-4 (Module 2):

- Prompt Engineering Fundamentals
- Advanced Prompts
- Focus: Prompt construction, few-shot learning, chain-of-thought, prompt optimization

Weeks 5-6 (Module 3):

- Chapter 2: Principles of Agentic Systems (pages 27-49)
- Focus: Self-governance, agency, autonomy, agent architectures (deliberative, reactive, hybrid)

Weeks 7-8 (Module 4):

- Chapter 3: Essential Components of Intelligent Agents (pages 51-73)
- Focus: Knowledge representation, reasoning mechanisms, decision-making, planning algorithms

Weeks 9-10 (Module 5):

- Chapter 5: Enabling Tool Use and Planning in Agents (pages 107-135)
- Focus: Tool definition and use, planning algorithms, tool integration

Weeks 11-12 (Module 6):

- Chapter 6: Exploring the Coordinator, Worker, and Delegator Approach (pages 137-158)
- Focus: CWD model, orchestrator-worker pattern, workflow patterns, role assignments, multi-agent collaboration

Weeks 13-14 (Module 7):

- Chapter 4: Reflection and Introspection in Agents (pages 77-105)
- Chapter 7: Effective Agentic System Design Techniques (pages 161-185)
- Focus: Reflection, meta-reasoning, system design best practices, memory architecture

Week 15 (Module 8):

- The Generative AI Application Lifecycle
- Focus: Production deployment, framework selection for production, monitoring, maintenance

---

Module 1: Foundations of AI & Modern Language Models

Duration: 8 Hours | **Sessions:** 1-2

Module Objectives

- Articulate the professional role of AI systems in modern software development
- Distinguish between generative AI and traditional machine learning approaches
- Successfully interact with LLM APIs to perform basic text generation tasks
- Understand the distinction between workflows and agents
- Get familiar with the agent framework landscape

Topics Covered

Session 1.1: Understanding AI in Professional Context (2 hours)

- **Demystifying AI**
- Professional Reality: AI as powerful tools for problem-solving, pattern recognition, and automation with clear limitations and specific use cases
- Introduction to Generative AI: How it differs from traditional machine learning

Session 1.2: Large Language Models as Components (3 hours)

- **The Engine of Modern AI: An Overview of LLMs**

- What is an LLM? Understanding it as a powerful pattern-matching and text-generation engine
- Key takeaway: An LLM is a *component*, not the entire AI system itself
- The Augmented LLM: Building block with retrieval, tools, and memory
- **Framework Landscape Overview**

- Introduction to agent development frameworks
- **CrewAI**: Role-based multi-agent collaboration, rapid prototyping
- **LangGraph**: State management and complex workflows
- **Pydantic AI**: Type-safe agent development

- Why frameworks exist and when to use them
- Understanding the trade-off: abstraction vs. control

Session 1.3: First Practical Interaction (3 hours)

- **Hands-On Development**
- Setting up API access and making basic API calls for text generation
- Understanding prompts and the concept of "prompt engineering"
- When to use simple solutions vs. agentic systems: The simplicity-first principle
- Basic prompt construction for the practical exercise

Practical Exercise

Project: "Creative Assistant"
Estimated Time: 3 hours
Deliverables: Python script demonstrating three LLM capabilities

Write a Python script that uses an LLM API to perform three distinct tasks: summarize a block of text, generate a marketing headline, and answer a simple factual question. This demonstrates the versatility of a base LLM.

Note: *Students will be provided with basic prompt templates and guidance from Session 1.3 to complete this exercise. Full prompt engineering skills will be developed in Module 2.*

Success Criteria:

- Successful API integration with error handling
- Three distinct, working functions using provided prompt templates
- Clear documentation and code comments

Module 2: Prompt Engineering Fundamentals

Duration: 8 Hours | **Sessions:** 3-4

Module Objectives

- Construct effective prompts using established techniques
- Apply basic and advanced prompt patterns to solve problems
- Understand prompt optimization strategies
- Evaluate prompt quality and iterate improvements

Topics Covered

Session 2.1: Basic Prompt Construction (3 hours)

- **Core Prompting Techniques**
- Prompt anatomy: instructions, context, input data, output indicators
- Zero-shot vs. few-shot prompting
- Role-based prompting and persona assignment
- Clear instruction formulation

Session 2.2: Advanced Prompt Patterns (3 hours)

- **Sophisticated Prompting Strategies**
- Chain-of-thought prompting for reasoning tasks
- ReAct pattern (Reasoning + Acting)
- Self-consistency and multiple reasoning paths
- Prompt templating and variables

Session 2.3: Prompt Optimization & Evaluation (2 hours)

- **Iterative Improvement**
- Testing and measuring prompt effectiveness
- Common pitfalls and how to avoid them
- Prompt versioning and documentation
- Handling edge cases and failure modes

Practical Exercise

Assignment: "Prompt Engineering Lab"
Estimated Time: 4 hours
Deliverables: Prompt library with documentation

Create a collection of optimized prompts for five different tasks (classification, summarization, extraction, generation, reasoning). For each, provide:

- Initial naive prompt
- Improved version using techniques from sessions
- Test results showing improvement
- Documentation of what changed and why

Success Criteria:

- Demonstrable improvement from initial to final prompts
- Application of at least 3 different prompting techniques
- Clear documentation of rationale
- Test cases showing effectiveness

Module 3: The Principles of Agentic Systems

Duration: 8 Hours | **Sessions:** 5-6

Module Objectives

- Define and differentiate between agency, autonomy, and self-governance
- Identify the key characteristics that make a system "agentic"
- Select appropriate agent architectures for different problem domains
- Distinguish between workflows (predefined paths) and agents (dynamic decision-making)
- Recognize when agentic complexity is warranted vs. simpler solutions

Topics Covered

Session 3.1: Defining Agency and Autonomy (3 hours)

- **Core Concepts**
- **Self-Governance:** An agent's ability to operate by its own internal rules
- **Agency:** The capacity to act independently and make choices on behalf of a user
- **Autonomy:** The degree of independence an agent has in performing tasks without human intervention
- Discussion with examples from the book
- **Workflows vs. Agents: A critical distinction**
- Workflows: Systems where LLMs and tools are orchestrated through predefined code paths
- Agents: Systems where LLMs dynamically direct their own processes and tool usage
- When each approach is appropriate

Session 3.2: Characteristics of an Intelligent Agent (2 hours)

- **Behavioral Traits**
- **Reactivity:** Responding to environmental changes
- **Proactiveness:** Taking initiative to achieve goals
- **Social Ability:** Interacting and cooperating with other agents
- **Ground truth feedback:** How agents learn from environmental responses

Session 3.3: Agent Architectures (3 hours)

- **Design Patterns**
- **Deliberative (Sense-Plan-Act):** Agents that use explicit knowledge and reasoning to make plans
- **Reactive (Stimulus-Response):** Agents that map perceptions directly to actions for rapid response
- **Hybrid:** Systems that combine both deliberative and reactive layers to get the best of both worlds
- Selecting the right architecture for your use case

Practical Exercise

Assignment: "Architecture Design Document"

Estimated Time: 5 hours
Deliverables: Three design documents

Given three scenarios (a real-time stock trading bot, a long-term vacation planner, and a smart home assistant), write a short design document for each, justifying whether a deliberative, reactive, or hybrid architecture would be most appropriate.

Success Criteria:

- Clear architectural choice with justification
- Identification of key components for each agent
- Discussion of trade-offs
- Consideration of when to use workflows vs. agents

Module 4: The Agent's Mind: Essential Components

Duration: 8 Hours | **Sessions:** 7-8

Module Objectives

- Implement different knowledge representation structures in Python
- Apply deductive, inductive, and abductive reasoning to agent scenarios
- Design utility functions for agent decision-making
- Understand how knowledge representation enables sophisticated agent behavior

Topics Covered

Session 4.1: Knowledge Representation for Agents (3 hours)

- **Structuring Agent Knowledge**
- Why agents need structured knowledge: Moving beyond simple text generation
- **Semantic Networks:** Representing concepts and relationships as graphs
- **Frames:** Structured objects with attributes and values (Python dictionaries/classes)
- **Logic-Based Representations:** Using formal logic for precision
- **Practical application:** How knowledge structures enable agent planning and reasoning
- Connection to modern agent architectures (memory, context management)

Session 4.2: Reasoning in Intelligent Agents (2 hours)

- **Cognitive Processes**
- **Deductive Reasoning:** From general rules to specific conclusions
- **Inductive Reasoning:** From specific examples to general rules (basis of learning)
- **Abductive Reasoning:** Inferring the most likely explanation
- **Real-world application:** How agents use reasoning to solve problems autonomously

Session 4.3: Decision-Making and Planning (3 hours)

- **Action Selection**
- **Utility Functions:** Quantifying preferences to guide agent choices
- **Planning Algorithms Overview:** State-space search and hierarchical planning
- **From theory to practice:** How these concepts manifest in modern agent frameworks
- Connecting knowledge representation to tool selection and task execution

Practical Exercise

Project: "The Smart Recommender"
Estimated Time: 6 hours
Deliverables: Working recommendation system with documentation

Build a movie recommender agent that uses frame-based knowledge structures (Python dictionaries/classes) to store movie data (genre, rating, length, themes). Implement a utility function that scores movies based on user preferences. The agent should explain its reasoning process, demonstrating how structured knowledge enables intelligent decision-making.

Key Learning Goal: Understand how the knowledge representation techniques in this module provide the foundation for agents that can reason about complex domains, make informed decisions, and explain their choices—skills essential for building production agent systems.

Success Criteria:

- Implementation of frame-based knowledge structure
- Working utility function with configurable preferences
- Agent reasoning explanation capability
- Demonstration with test cases showing how structure enables better decisions
- Clear code documentation connecting theory to implementation

Module 5: Practical Agent Development: Tools & Frameworks

Duration: 10 Hours | **Sessions:** 9-10

Module Objectives

- Define and implement custom tools for agents
- Compare CrewAI and LangGraph frameworks
- Apply framework selection criteria to choose appropriate tools for different use cases
- Build agents that can orchestrate multiple tools to solve complex tasks
- Master the principles of effective tool design (Agent-Computer Interface)

Topics Covered

Session 5.1: Tool Integration & Agent-Computer Interface (4 hours)

- **Extending Agent Capabilities**

- The practical necessity of tool use for agents
- Defining tools in Python for frameworks like CrewAI and LangGraph
- Tool Composition: Chaining tools together to solve multi-step problems
- **Designing Effective Tools: The Agent-Computer Interface (ACI)**

- **Critical principle:** Tool design deserves as much attention as Human-Computer Interface (HCI)
- **Tool format considerations:**
- Give models enough tokens to "think" before writing themselves into a corner
- Keep formats close to what models have seen in training data
- Minimize formatting "overhead" (line counting, string escaping, etc.)
- **Best practices for tool documentation:**
- Write like you're documenting for a junior developer
- Include example usage, edge cases, input format requirements
- Make parameter names and descriptions obvious
- Test extensively and iterate based on model mistakes
- "Poka-yoke" your tools—design them so mistakes are harder to make
- **Real-world example:** Using absolute filepaths instead of relative paths to prevent agent confusion

Session 5.2: Framework Comparison & Implementation (6 hours)

- **Framework Fundamentals**

- Why use a framework? Abstraction, state management, and simplifying complexity
- When frameworks help and when they add unnecessary complexity
- Understanding what's "under the hood"—avoiding incorrect assumptions
- **Understanding Framework Trade-offs**

CrewAI: Role-Based Multi-Agent Systems

- **Strengths:** Intuitive team-based design, fastest time to productivity, excellent for specialized agent collaboration
- **Best for:** Content pipelines, business process automation, research teams, scenarios that map naturally to human team structures
- **Considerations:** Higher token usage with multiple agents, not yet v1.0, less sophisticated state management

LangGraph: Stateful Graph-Based Workflows

- **Strengths:** Production-grade reliability, powerful state management, supports cycles and iterative workflows, excellent debugging
- **Best for:** Long-running workflows, human-in-the-loop systems, complex conditional branching, mission-critical applications
- **Considerations:** Steeper learning curve, requires predefined workflow structure, more complex for simple tasks

Pydantic AI: Type-Safe Agent Development

- **Strengths:** Excellent type safety and validation, familiar FastAPI-like developer experience, ideal for structured outputs
- **Best for:** Applications requiring validated outputs, teams already using Pydantic/FastAPI, enterprise systems needing consistent data formats
- **Considerations:** Newest framework, limited multi-agent orchestration, smaller ecosystem

Quick Decision Framework:

- Specialized team collaboration needed? → **CrewAI**
- Complex stateful workflows or production-critical? → **LangGraph**
- Type safety and validated outputs paramount? → **Pydantic AI**
- Simple linear workflows? → Consider simpler alternatives or basic workflow patterns

- **Live coding demonstrations:** Implementing a simple workflow in both CrewAI and LangGraph

Practical Exercise

Midterm Project: "The Daily Briefing Bot"
Estimated Time: 10 hours
Deliverables: Two implementations (CrewAI + LangGraph) with brief comparison + tool documentation

Build an agent that uses three well-designed tools: `get_weather(location)` , `get_top_news_headlines(topic)` , and `get_stock_price(ticker)` . When the user asks for their "daily briefing," the agent must plan to use all three tools and synthesize the information into a single, coherent summary.

Part 1: Tool Design Before implementation, create comprehensive tool documentation for each of the three tools, including:

- Clear, unambiguous descriptions
- Example usage
- Edge cases and error conditions
- Parameter format requirements

Part 2: Implementation Implement this project first in **CrewAI**, then in **LangGraph** to compare the different design philosophies.

Part 3: Comparison Report (1 page maximum)

- Which implementation was easier to build and why?
- Which would you choose for a production system handling 10,000 users? Justify your answer
- Key difference you noticed in how each framework handles tool orchestration
- How did thoughtful tool design impact your development experience?

Success Criteria:

- Comprehensive tool documentation following ACI principles
- Successful implementation in both frameworks
- All three tools integrated and working
- Coherent information synthesis
- One-page written comparison addressing the required questions
- Error handling and edge cases addressed

Module 6: Multi-Agent Collaboration

Duration: 8 Hours | **Sessions:** 11-12

Module Objectives

- Design multi-agent systems with clear role separation
- Implement the Coordinator-Worker-Delegator pattern
- Understand the Orchestrator-Worker pattern for parallel task execution
- Apply workflow patterns in multi-agent contexts
- Manage inter-agent communication and workflow

Topics Covered

Session 6.1: Introduction to Multi-Agent Systems (2 hours)

- **Beyond Single Agents**
- The power of specialization and division of labor
- When to use multi-agent vs. single-agent systems

- **The economics of multi-agent systems:**
- Token usage considerations: Multi-agent systems use ~15x more tokens than simple chat
- When the value justifies the cost
- Performance gains with orchestrator patterns (can outperform single agents by 90%+ on complex tasks)
- **When multi-agent systems excel:**
- Tasks requiring heavy parallelization
- Information exceeding single context windows
- Interfacing with numerous complex tools
- Breadth-first queries with multiple independent directions

Session 6.2: Multi-Agent Patterns (3 hours)

- **The Coordinator-Worker-Delegator (CWD) Model**

- **Coordinator:** Receives tasks, develops plans, assigns work, synthesizes results
- **Workers:** Execute specialized tasks (research, writing, analysis, etc.)
- **Delegator:** Routes complex tasks to appropriate workers
- Communication protocols and state management between agents
- When to use: Content pipelines, research tasks, business process automation
- **The Orchestrator-Worker Pattern**

- **Orchestrator:** Lead agent that coordinates specialized subagents
- **Workers:** Domain-specific agents working in parallel
- Key advantages: parallel processing, separation of concerns, reduced path dependency
- When to use: Complex research, multi-aspect analysis, tasks requiring diverse expertise
- **Common Workflow Patterns**

- **Prompt Chaining:** Sequential agent collaboration for step-by-step workflows
- **Routing:** Directing requests to specialized agents based on classification
- **Parallelization:** Multiple agents working simultaneously on independent subtasks
- **Evaluator-Optimizer:** Quality assurance agents providing feedback for iterative refinement
- Selecting and combining patterns for sophisticated multi-agent workflows

Session 6.3: Designing Multi-Agent Workflows (3 hours)

- **Practical Implementation**
- Assigning roles, backstories, and goals
- Structuring communication between agents
- Managing shared state and context
- Common failure modes and solutions

Practical Exercise

Project: "The Collaborative Content Team"
Estimated Time: 8 hours
Deliverables: Working multi-agent system using CWD pattern

Build a content creation system with three agents working together:

- **Coordinator Agent:** Receives content request, plans approach, assigns tasks
- **Research Worker:** Gathers information and facts on the topic
- **Writer Worker:** Creates content based on research findings

Framework: Implement using **CrewAI** with clear role separation.

Specific Requirements:

- Clear role definition for each agent
- Sequential workflow with information passing (prompt chaining pattern)
- Tool usage by workers (web search for researcher, writing tools for writer)
- Final content synthesis by coordinator

Success Criteria:

- Clear role separation and communication protocols
- Successful task delegation and completion
- Quality content output with proper research integration
- Documentation showing workflow decisions and pattern application
- Error handling for failed subtasks

Module 7: Advanced Capabilities & Capstone

Project

> **Duration:** 8 Hours | **Sessions:** 13-14

Module Objectives

- Implement reflection and self-correction in agents
- Apply best practices for system design and prompt engineering
- Design memory architecture for agent systems
- Deliver a complete, production-ready multi-agent application

Topics Covered

Session 7.1: Reflection & Introspection (3 hours)

- **Meta-Reasoning**
- Revisiting the concept of meta-reasoning and self-correction
- How agents can "think about their thinking"
- Implementing feedback loops and iterative improvement
- Evaluator-optimizer patterns for self-improvement

Session 7.2: Advanced System Design (3 hours)

- **Professional Design Practices**
- Writing focused system prompts and instructions
- The importance of agent memory and context management
- Memory architecture patterns:
- Short-term (conversation context)
- Long-term (persistent knowledge)
- Working memory (task-specific state)
- Workflow Optimization: Choosing between sequential and parallel agent execution
- Design patterns for complex agent behaviors

Session 7.3: Capstone Project Workshop (2 hours)

- **Final Project Development**
- Design and build the final project with instructor guidance
- Code review sessions
- Troubleshooting and optimization
- Applying all learned concepts

Capstone Project

> **Project: "The Automated Support Assistant"**
> **Estimated Time:** 18 hours (includes independent work)
> **Deliverables:** Complete production-ready multi-agent system, documentation, evaluation results, and presentation
>
> Students will design a multi-agent customer support system that handles incoming requests, routes them appropriately, and provides solutions.
>
> **Required Architecture:**

- **Coordinator Agent:** Receives and analyzes customer requests, routes to appropriate worker
- **Knowledge Base Worker:** Searches documentation and FAQs for solutions
- **Technical Worker:** Handles technical troubleshooting tasks
- **Escalation Worker:** Identifies when human intervention is needed

> **Advanced Requirements:**

- Comprehensive tool documentation following ACI principles
- Memory management (maintaining conversation context)
- Error handling with graceful recovery
- Application of appropriate workflow patterns (routing, prompt chaining, etc.)
- Clear role separation and communication protocols

> **Bonus Challenge:** Add a `Feedback Agent` (evaluator-optimizer pattern) that reviews responses, provides critiques, and triggers refinement iterations.
>
> **Success Criteria:**

- Minimum four specialized agents with clear roles
- Complete workflow from input to resolution
- Appropriate workflow pattern application
- Memory and context management implementation
- Robust error handling
- Comprehensive documentation (README, API docs, architecture diagram, tool specifications)
- 10-minute presentation demonstrating the system
- Clean, maintainable, well-documented code following best practices
- Bonus: Working evaluator-optimizer feedback loop

Module 8: Production Systems & Deployment

Duration: 4 Hours | **Session:** 15

Module Objectives

- Understand production reliability and engineering challenges
- Apply the generative AI application lifecycle to agent systems
- Select appropriate frameworks for production deployments
- Address security, monitoring, and maintenance concerns
- Evaluate agentic systems effectively

Topics Covered

Session 8.1: The Generative AI Application Lifecycle (1.5 hours)

- **From Prototype to Production**
- **Ideation and Prototyping:**
- Moving from idea to proof-of-concept
- Rapid iteration and testing
- Validating core assumptions
- **Building and Augmenting:**
- Integrating tools, memory, and knowledge bases
- Scaling from prototype to production-ready system
- Adding robustness and error handling
- **Evaluation and Deployment:**
- Testing strategies for AI applications
- Deployment patterns and monitoring
- Continuous improvement cycles
- **Maintenance and Evolution:**
- Monitoring performance and costs
- Updating models and prompts
- Managing technical debt

Session 8.2: Framework Selection for Production (1.5 hours)

- **Production-Grade Framework Comparison**
  CrewAI in Production:

- **When to choose:** Rapid development cycles, content generation pipelines, business process automation
- **Production considerations:** AWS integration, token cost management, role-based scaling
- **Limitations:** Not yet v1.0, requires careful state management, higher token costs
- **Best practices:** Clear agent boundaries, efficient tool design, monitoring agent interactions
  LangGraph in Production:

- **When to choose:** Mission-critical applications, complex stateful workflows, enterprise deployments
- **Production advantages:** 400+ companies in production, comprehensive observability, debugging tools
- **Trade-offs:** Steeper learning curve, more complex initial setup
- **Best practices:** LangGraph Studio for debugging, checkpoint strategies, human-in-the-loop patterns
  Pydantic AI in Production:

- **When to choose:** Type-safe applications, teams using FastAPI, structured output requirements
- **Production advantages:** V1 stability, 100% test coverage, familiar developer experience
- **Limitations:** Smaller ecosystem, limited multi-agent orchestration
- **Best practices:** Leverage Pydantic validation, structured outputs, type safety

- **Decision Framework:**

- Prototyping → CrewAI (fastest to value)
- Enterprise-critical → LangGraph (proven reliability)
- Type-safe systems → Pydantic AI (validation first)
- Simple workflows → Consider vanilla Python implementation

Session 8.3: Production Reliability & Challenges (1 hour)

- **Production-Specific Concerns**

- Error handling, logging, and debugging strategies
- Security considerations: API key management, prompt injection prevention
- **Stateful execution and error compounding:**
- Agents maintain state across many tool calls
- Minor errors cascade into large behavioral changes
- Need for durable execution and graceful error handling
- Resume-from-checkpoint strategies
- **Debugging agentic systems:**
- Non-deterministic behavior between runs
- Full production tracing and observability
- Monitoring decision patterns
- Diagnosing root causes systematically
- **Deployment considerations:**
- Deployment strategies for stateful agent systems
- Avoiding disruption to running agents during updates
- Gradual traffic shifts between versions
- Cost monitoring and optimization
- Performance metrics and SLAs
- **Evaluating Agentic Systems**

- **Start small and iterate:**
- Begin with representative test cases
- Scale up as system matures
- **LLM-as-judge evaluation:**
- Criteria: factual accuracy, completeness, source quality, tool efficiency
- Most effective for tasks with clear correct answers
- **Human evaluation catches edge cases:**
- Unusual query handling
- Subtle biases
- System failures not caught by automated evals
- **End-state evaluation for stateful agents:**
- Focus on final outcomes
- Evaluate whether correct final state achieved

Final Assessment

Students will present their capstone projects, demonstrating:

- System architecture and design decisions
- Live demonstration of the working system
- Discussion of production readiness
- Lessons learned and future improvements

Success Criteria:

- Understanding of complete application lifecycle
- Appropriate framework selection with justification
- Production considerations addressed
- Evaluation strategy implemented
- Professional presentation of capstone project

Additional Resources

Recommended Reading

- Anthropic Claude API Documentation
- OpenAI API Documentation
- CrewAI Official Documentation
- LangGraph Documentation

- Anthropic Engineering Blog: "Building Effective Agents"
- Anthropic Engineering Blog: "Multi-Agent Research System"
- Anthropic Cookbook: Agent Patterns & Prompts
- Microsoft: Generative AI for Beginners (GitHub Repository)
- "Prompt Engineering Guide" (online resource)
- Model Context Protocol Documentation

Support Resources

- Weekly office hours: [Schedule TBD]
- Course discussion forum
- Peer programming sessions (optional)
- Industry guest speakers (bi-weekly)
- Agent simulation environment for testing

Ethics & Responsible AI

Throughout the course, students are expected to:

- Consider the ethical implications of autonomous agent behavior
- Implement appropriate safeguards and monitoring
- Follow responsible AI development practices
- Respect data privacy and API usage policies
- Understand token usage economics and environmental impact
- Design agents with appropriate autonomy limits and human oversight

---

Course Schedule Summary

Week	Module	Topics	Deliverables
1-2	Module 1	AI Foundations, LLMs, Framework Overview	Creative Assistant
3-4	Module 2	Prompt Engineering (Fundamentals & Advanced)	Prompt Engineering Lab
5-6	Module 3	Agentic Principles, Workflows vs. Agents, Architectures	Architecture Design Documents
7-8	Module 4	Agent Components, Knowledge Representation	Smart Recommender
9-10	Module 5	Tool Integration (ACI), Framework Comparison	Daily Briefing Bot (Midterm)
11-12	Module 6	Multi-Agent Systems, CWD, Orchestrator, Workflow Patterns	Collaborative Content Team
13-14	Module 7	Advanced Capabilities, Reflection, System Design	Support Assistant (Capstone)
15	Module 8	Production Systems, Lifecycle, Framework Selection, Deployment	Capstone Presentations

---

This curriculum is designed to be flexible and may be adjusted based on class progress and student needs. The course reflects both academic foundations and real-world production practices from leading AI companies.

TYPESET WITH MARKDOWNER (HTTP://MARKDOWNER.GITHUB.IO)