Claude for Agentic AI & AI Architect + RAG + MCP

Claude API • Claude SDK • RAG • Agentic Orchestration • MCP • Claude Code

10 Days | 3 Hours/Day | 30 Hours Total | Hands-On, Project-Based Curriculum

Aligned to the Claude Certified Architect

Day	Topic	Detailed Content	Hands-On / Practical Session
Day 1 *3 hrs*	**Foundations: AI, ML, DL, GenAI, Agents & Agentic AI**	• Introduction to Artificial Intelligence • Machine Learning Fundamentals: supervised, unsupervised, reinforcement learning • Deep Learning Fundamentals • Generative AI: How LLMs generate text — tokens, embeddings, transformer architecture (attention, self-attention) at a conceptual level • Evolution: ML → DL → GenAI → LLMs → AI Agents → Agentic AI • What makes an 'Agent': perception, reasoning, memory, action, tool • Agentic AI Defined: autonomy, planning, multi-step reasoning, orchestration of tools/agents • Landscape of LLM providers (OpenAI, Anthropic, Google, Meta) and where Claude fits • Course roadmap and outcomes: from prompting to building a production AI Architect solution	• Explore Claude.ai interface — chat, Projects, Artifacts • Compare a plain LLM response vs an agentic multi-step response on the same task • Group discussion: Identify Real business processes that could be automated by an agent • Set up dev environment: Python, VS Code, Git, Anthropic account & API key
Day 2 *3 hrs*	**Claude 101 — Model Family & Platform Deep Dive**	• Claude Introduction • Claude model family: Haiku, Sonnet, Opus • Claude's core capabilities: reasoning, coding, vision, long context window, multilingual support • Claude.ai vs Claude Console vs Claude API vs Claude Code vs Claude Cowork — when to use which • Understanding context window, tokens, and pricing model (input/output tokens) • Claude's constitutional AI approach & safety design — why it matters for enterprise adoption • Extended thinking / reasoning mode overview • Introduction to Claude Extensions • Setting up Anthropic Console: API keys, workspaces, billing, rate limits	• Create Anthropic Console account, generate API key, explore usage dashboard • Run first API call using cURL / Postman to the Messages endpoint • Compare outputs of Haiku vs Sonnet vs Opus on the same prompt — latency & quality trade-off exercise • Explore token counting using Anthropic's tokenizer

Day	Topic	Detailed Content	Hands-On / Practical Session
Day 3 *3 hrs*	**Prompt Engineering & Context Management** ***CCA-F Domains — Prompt Engineering & Structured Output (20%) / Context Management & Reliability (15%)***	• Anatomy of an effective prompt: role, task, context, constraints, output format • System prompts vs user prompts vs assistant turns — message roles in the API • Techniques: zero-shot, few-shot, chain-of-thought, self-consistency, structured output (XML/JSON) prompting • Prompt engineering fundamentals • XML/tag-based prompt structuring • Prompt chaining and task decomposition • Prefilling responses, stop sequences, temperature/top-p/top-k tuning • JSON-Schema constrained output: forcing schema-compliant tool/response objects • Validation-retry loops: detecting malformed structured output and re-prompting for a corrected response • Message Batches API: asynchronous bulk processing, status polling, and when batch beats real-time for cost/throughput • Context management: context window limits and summarization strategies for long conversations • 'Lost-in-the-middle': why instructions/facts placed mid-context get under-weighted, and placement strategies to counter it • Scratchpads: preserving intermediate reasoning/state across context boundaries and multi-turn/multi-agent handoffs • Context provenance: tagging and tracking where injected context (retrieved docs, tool output, user input) came from, for trust and debugging • Prompt caching internals: cache breakpoints, TTL, and their effect on cost/latency • Token budgeting: allocating a fixed context window across system prompt, history, retrieved context, and tool output • Common pitfalls: prompt injection basics, ambiguity, over-constraining vs under-constraining • Evaluating prompts: rubric-based grading	• Rewrite a weak prompt into a strong structured prompt using XML tags • Build a few-shot classification prompt (e.g., support ticket triage) • Implement prompt caching with explicit cache breakpoints and measure cost/latency savings • Build a chain-of-thought prompt for a multi-step reasoning task and evaluate accuracy • Design a JSON-Schema output contract and implement a validation-retry loop for malformed responses • Submit a bulk classification job through the Message Batches API and poll for completion • SCENARIO: A 10k-token prompt buries the one critical instruction in the middle and Claude ignores it — diagnose the lost-in-the-middle failure and restructure the prompt to fix it
Day 4 *3 hrs*	**Claude API Deep Dive** ***Feeds CCA-F Domains 2 & 4 — stop_reason / tool_choice / Batches API mechanics***	• Messages API structure: request/response schema, roles, content blocks • Multi-modal input: text, images (vision), PDFs/documents — use cases and limits • Streaming responses (SSE) for real-time UX vs standard synchronous calls	• Build a Python script calling the Messages API with streaming enabled • Build a simple tool-calling flow: weather/lookup tool with tool_use/tool_result loop • Implement retry logic with exponential backoff for rate-limited requests

Day	Topic	Detailed Content	Hands-On / Practical Session
		• Tool use / function calling: defining tool schemas, tool_use & tool_result blocks, multi-tool workflows • Claude API terminology : stop_reason, tool_use, end_turn, max_tokens, stop_sequence, and branching application logic correctly on each • tool_choice parameter • tool_result • Structured outputs: forcing JSON schema-compliant responses for downstream systems • Error handling: rate limits, retries with exponential backoff, timeouts, overloaded errors • Authentication, security best practices (never expose keys client-side), usage monitoring	• Force a specific tool call using tool_choice='tool' inside a deterministic workflow step • Build a structured tool-error response and verify Claude retries with corrected parameters • An agent loop never terminates because the app only checks for end_turn — trace stop_reason values across three sample transcripts (tool_use, max_tokens, stop_sequence) and fix the loop-exit condition
Day 5 *3 hrs*	**Claude SDK — Building Applications**	• Anthropic Python & TypeScript SDK setup, client configuration, environment variables • SDK abstractions over raw API: client.messages.create, async clients, helper utilities • Claude Agent SDK overview: agent loop primitives (gather context → take action → verify → repeat) • Managing conversation history and message threading using the SDK • File & document handling via SDK (PDFs, code files, datasets) • Building reusable client wrappers/services for production apps (config, logging, observability hooks) • Testing strategies for LLM-powered code: mocking API calls, snapshot testing prompts • SDK-based cost tracking and token usage logging	• Initialize a project using the Python/TS SDK with a config-driven client wrapper • Build a multi-turn chatbot service class with conversation memory using the SDK • Add structured logging for token usage and latency per request
Day 6 *3 hrs*	**Retrieval-Augmented Generation (RAG) on Claude**	• Why RAG: grounding responses in private/enterprise data, reducing hallucination, staying current • RAG pipeline anatomy: ingestion → chunking → embedding → vector store → retrieval → augmented prompt → generation • Chunking strategies: fixed-size, semantic, recursive chunking; overlap trade-offs • Embeddings and vector databases (e.g., FAISS, Pinecone, Chroma, pgvector) — selection criteria • Building the retrieval layer: similarity search, hybrid search (keyword + vector), re-ranking • Prompt design for RAG: injecting retrieved context, citation/source attribution formatting	• Build a document ingestion + chunking pipeline for a sample PDF/knowledge base • Generate embeddings and store them in a local vector DB (Chroma) • Implement retrieval + augmented prompt construction feeding into Claude • Add citation formatting so answers reference source chunks

Day	Topic	Detailed Content	Hands-On / Practical Session
		• Claude's long context window as an alternative/complement to RAG — when to use context-stuffing vs retrieval • Evaluating RAG systems: faithfulness, relevance, answer correctness metrics	
Day 7 *3 hrs*	**Agentic Architecture & Orchestration** ***CCA-F Domain 1 — Agentic Architecture & Orchestration***	• Introduction to orchestration frameworks (LangGraph, custom orchestration with Claude SDK) • Core agent loop: gather context → take action → verify results → repeat (the formal 'agentic loop') • Agent design patterns: ReAct (reason+act), planner-executor, reflection/self-critique loops • workflows vs. model-directed agents • task decomposition strategies and sequential vs. parallel execution patterns • dynamic planning, replanning, and ambiguity handling mid-task • orchestrator-subagent model: defining subagent roles, scope, and context isolation from the orchestrator's full history • multi-agent topology patterns: hub-and-spoke, pipeline, and peer-to-peer — trade-offs of each for coordination, cost, and failure isolation • Agent communication and handoff patterns between orchestrator and subagents • Memory systems for agents: short-term (context window), long-term (vector store/DB), episodic memory • iteration/step limits as a formal reliability control (not just a safety afterthought), and where to place human-in-the-loop approval checkpoints based on risk	• Design and diagram an agent architecture for a real use case (e.g., research assistant, ops agent) • Implement a basic ReAct-style loop in code using the Claude SDK (reasoning + tool call + observation) • Add a formal step-limit and human-approval checkpoint to prevent runaway agent loops • Build a 2-agent orchestrator-worker flow (planner agent delegates to executor agent) with context isolation between them
Day 8 *3 hrs*	**Tools & MCP (Model Context Protocol) Integration** ***CCA-F Domain 2 — Tool Design & MCP Integration***	• Function/tool calling recap: schema design best practices, parallel tool calls • Exam mechanic — tool descriptions as the primary selection mechanism: how Claude chooses between available tools based on description quality, and how ambiguous/overlapping descriptions cause wrong-tool selection • tool_choice in an MCP context: auto/any/tool/none applied to MCP-exposed tools • Structured error design for tool_result blocks: making failures recoverable so Claude can self-correct rather than derail the task • Introduction to MCP: purpose, architecture (hosts, clients, servers), why it standardizes tool/context sharing	• Connect Claude to a sample/public MCP server and query it end-to-end • Build a minimal custom MCP server (e.g., exposing a CSV/database query tool) • Wire the custom MCP server into a Claude-powered agent and validate the tool_use → tool_result → stop_reason flow • Add logging/auditing around tool execution for traceability • Force a specific MCP tool call via tool_choice and confirm the correct tool_result is returned • SCENARIO: An MCP tool call fails with a raw stack trace as the error — redesign the tool_result error payload so Claude can interpret the failure and retry with corrected input

Day	Topic	Detailed Content	Hands-On / Practical Session
		• MCP servers vs traditional API integrations — resources, tools, and prompts exposed via MCP • Connecting Claude to existing MCP servers (e.g., file systems, databases, SaaS connectors) • Building a custom MCP server: exposing a tool/data source to Claude via MCP • Claude Tag and Claude Cowork as MCP-consuming surfaces — enterprise integration patterns • Debugging and testing MCP integrations, including diagnosing tool-selection failures	
Day 9 *3 hrs*	**Claude Code Configuration & Agentic Coding Workflows**	• Claude Code overview: terminal, VS Code, JetBrains, and desktop variants • Project configuration via CLAUDE.md: conventions, coding standards, architecture notes for the agent — and the CLAUDE.md configuration hierarchy (project vs user vs enterprise level) • Slash commands, custom commands, and Agent Skills for workflow automation within Claude Code • Subagents: delegating specialized tasks (testing, review, docs) to focused subagents • Hooks: pre/post-action automation (lint on save, run tests before commit, etc.) • Using Claude Code for end-to-end feature development: plan → code → test → review → PR	• Install and configure Claude Code in a sample repository with a CLAUDE.md file • Use Claude Code to implement a small feature end-to-end and open a pull request • Configure an explicit permission model (allowed-tools list + approval mode) restricting what Claude Code can do unattended
Day 10 *3 hrs*	**AI Architect**	• Role of an AI Architect: translating business requirements into agentic system design • Architecture patterns for production agentic systems: API gateway, orchestration layer, tool/MCP layer, RAG layer, observability layer • Security & governance: data privacy, PII handling, prompt-injection defense, access control, audit trails • Cost optimization: model selection strategy, prompt caching, batch processing, token budgeting • Change management: versioning prompts/agents, rollback strategy, human-in-the-loop review	• Draw a full architecture diagram for an end-to-end agentic system combining RAG + tools + MCP + orchestration

Program Outcome: By the end of this program, participants will be able to design, build, and deploy production-grade agentic AI systems on Claude — spanning prompt engineering, the Claude API/SDK, RAG pipelines, multi-agent orchestration, MCP-based tool integration, and Claude Code-driven engineering workflows — culminating in an end-to-end capstone application.