ONE FLAGSHIP SYSTEM. PRODUCTION ENGINEERING END TO END.

Agentic AI Enterprise Mastery Bootcamp

Build one AI system. Own every decision.

The engineering around the model: harness, context and memory, governed tools, reliable execution, evaluation, security, deployment and the reasoning behind every layer.

Next cohort: Third week of January 2027 · tentative

Sat & Sun, 7-10 AM IST | US Eastern: Fri & Sat, 8:30-11:30 PM EST

The January start window is tentative. The exact opening date and joining details will be confirmed in the learner community.

Six-week core build + two-week Advanced Production Mastery bonus

Start with recorded depth. Apply it in the live build.

- Bonus recordings from earlier completed cohorts
- Asynchronous access to the November 2026 cohort
- Two free future live cohort attendances
- Python and LangChain foundation bonuses
- Advanced RAG Implementation Vault after full payment

Early-learner benefit

Complimentary Agentic Ops Bootcamp for the first 100 confirmed learners. Opening allocation: 17 claimed, supplied by Manifold. Bonus reference value US$89. The live course page shows current availability.

Full payment unlocks the bootcamp. The team activates the Advanced RAG Implementation Vault after confirming full payment.

WEEK 01 / CORE FLAGSHIP BUILD

Production Agent Architecture & Harness Engineering

Build layer: The Agent Harness - architecture, repository, and the controlled execution spine your flagship system runs on

You already know what an agent is. This week is about what surrounds it. The model is one component; production behaviour is decided by the harness around it - instructions, runtime state, orchestration, tools, context, permissions, validation, budgets and recovery. You convert a business use case into functional and non-functional requirements, decide honestly what belongs inside a graph versus plain deterministic code, and design the architecture before writing it. Then you build the runtime: typed state contracts, structured execution, loop protection, iteration and token budgets, checkpointing and resumable workflows - deterministic control wrapped around a probabilistic model.

Concepts and engineering practice

- Use case / system requirements
- Agent vs deterministic workflow
- The agent harness
- Config & environment management
- LangGraph runtime
- Structured execution
- Token & timeout budgets
- Failure-state modelling
- Architecture Decision Records
- Functional & non-functional requirements
- Architecture-first design
- Production repo structure
- Model-provider abstraction
- Typed state contracts
- Loop protection & iteration limits
- Checkpointing & resumable workflows
- Deterministic control around probabilistic models

By the end of Week 1 you can explain: what the harness is and why it decides production behaviour, what belongs inside the graph and what stays deterministic, how runaway execution is bounded, and how the system recovers from interrupted work.

WEEK 02 / CORE FLAGSHIP BUILD

Context Engineering, Memory & Durable Agent Runtime

Build layer: Context & Memory - what the model sees, what it remembers, and a runtime that survives interruption

Prompt engineering is one small part of a larger discipline. Context engineering is deciding what enters the model's window at all - system instructions, conversation history, agent state, tool results, retrieved knowledge, memory, and output from other agents - and what gets budgeted, compressed, compacted or left out. You then separate working state from memory, decide what should persist and what must never persist, and build an agent runtime with durable execution: checkpointing, pause and resume, long-running work and human approval pauses. Retrieval appears here as one source of context among several.

Concepts and engineering practice

- Context selection & assembly
- Context compression & compaction
- Progressive disclosure
- Context freshness & provenance
- Prompt caching concepts
- Retrieval as a context source
- Short-term vs long-term memory
- Memory persistence (Redis / PostgreSQL)
- TTL & deletion policies
- What should never become memory
- Pause / resume & interrupted workflows
- Context budgeting
- Summarization
- Context isolation
- Long-context failure modes
- Tool-result / context handling
- Working state vs memory
- Episodic vs semantic memory
- Memory summarisation
- Memory pollution & poisoning
- Agent runtime & durable execution
- Human approval pauses

On retrieval: production retrieval remains an important subsystem of the flagship architecture - here you engineer how it participates: retrieval interfaces, access control, context assembly, grounding, citations, confidence and abstention, and how it differs from memory. Deep implementation - ingestion, parsers, chunking, embeddings, vector databases, metadata engineering, hybrid retrieval, reranking and retrieval evaluation - is provided in full through the self-paced Advanced RAG Implementation Vault , included with your seat and activated after full bootcamp payment, which keeps the live sessions focused on the whole production system.

By the end of Week 2 you can explain: how context is selected and budgeted rather than merely written, how retrieval and memory differ architecturally, what should never be persisted, and how a long-running agent resumes after interruption.

WEEK 03 / CORE FLAGSHIP BUILD

Tool, MCP & Agent Gateway Engineering

Build layer: The Gateway - reliable tool contracts, MCP integration, agent identity and sandboxed execution

A tool that works in a notebook is not a tool that survives production. You build reliable tool contracts with structured schemas and Pydantic validation, then wrap them in execution policy: allowlists, read versus write separation, idempotency, retries with backoff, timeouts, circuit breakers, rate limits and execution budgets. You cover where MCP genuinely belongs - and where a direct function call is the better engineering choice. Then the concept widens: Tool Gateway / MCP Gateway / Agent Gateway , a policy and control plane between agents and the tools, models, services and other agents they reach for. Finally, identity: user identity versus agent identity, delegated authorization, least privilege, and sandboxed execution boundaries.

Concepts and engineering practice

- Tool contracts & structured schemas
- Pydantic validation
- Allowlists & denylists
- Read vs write operations
- Idempotency
- Retries & exponential backoff
- Timeout policies
- Circuit breakers
- Rate limiting
- Tool execution budgets
- Failure-aware tool responses
- Testing external tools
- Tool audit logging
- MCP architecture
- MCP server & client
- When MCP - and when a direct call
- MCP gateway architecture
- MCP trust boundaries
- Secure secret handling
- Tool Gateway / MCP Gateway / Agent Gateway
- Agent identity vs user identity
- Workload identity
- Delegated authorization (OAuth / OIDC concepts)
- Least privilege & scoped access
- Short-lived credentials
- Permission boundaries
- Sandboxed execution
- Filesystem & network isolation
- Ephemeral environments & resource limits

By the end of Week 3 you can explain: why not every function should become a tool, what happens when a tool times out, where MCP belongs versus a direct API call, what an Agent Gateway is for, and which identity is actually accessing the data.

WEEK 04 / CORE FLAGSHIP BUILD

Agent Reliability Engineering & Asynchronous Execution

Build layer: Asynchronous Agent Service - an API-backed system that runs long jobs without blocking anyone

Production agent workflows run for minutes, not milliseconds. The flagship system becomes a FastAPI service with Celery workers, queue design, job state, progress streaming, cancellation and resumption. Then the distributed-systems reality: one user request fans out into multiple agents, multiple model calls and multiple tool calls, so agent workloads amplify infrastructure pressure in ways ordinary CRUD services do not. You cover backpressure, load shedding, admission control, retry storms and cascading failure - and set SLIs, SLOs and error budgets for things that actually matter: task success rate, p95/p99 latency, tool success rate, cost per successful workflow. If you have run production services before, this is the week your existing experience pays off directly.

Concepts and engineering practice

- FastAPI service architecture
- Sync vs async execution
- Celery & Redis workers
- Cancellation & resumption
- Duplicate-request handling & idempotency
- SSE streaming
- Graceful degradation
- Backpressure
- Admission control
- Cascading failures
- SLI / SLO design
- Task success rate & p95/p99 latency
- API & worker testing
- Request / response contracts
- Queue design
- Job state & progress tracking
- Retries & dead-letter handling
- Concurrency control
- Webhook patterns
- Health & readiness checks
- Load shedding
- Retry storms
- Fan-out amplification
- Error budgets
- Cost per successful workflow

By the end of Week 4 you can explain: why a queue is required, what happens when a worker dies mid-run, how one request becomes twenty downstream calls, which SLOs this system is held to, and how it degrades instead of collapsing.

WEEK 05 / CORE FLAGSHIP BUILD

Continuous Evaluation, Agentic Observability & Agentic Security

Build layer: Evaluated, Observable and Defended System - eval suites, OpenTelemetry traces, and a real threat model

Three disciplines that decide whether anyone trusts the system. Evaluation starts with golden datasets, output quality, retrieval, tool-selection and trajectory evaluation, deterministic checks, LLM-as-judge design and CI eval gates - then becomes continuous : production traces feed categorised failures, failures become new eval cases, changes are regression-tested before redeployment. Agentic observability means a trace that spans request / agent / model / tool / MCP / retrieval / sub-agent, instrumented with OpenTelemetry for vendor-neutral traces alongside Langfuse and LangSmith. Agentic security goes well past generic prompt injection into goal hijacking, excessive agency, context and memory poisoning, privilege abuse, exfiltration and unsafe code execution - taught as threat modelling and blast-radius reduction, not a cybersecurity specialisation.

Concepts and engineering practice

- Golden evaluation datasets
- Output-quality evaluation
- Retrieval evaluation
- Tool-selection evaluation
- Trajectory evaluation
- Deterministic checks
- LLM-as-judge design
- Failure taxonomy
- Regression testing
- CI evaluation gates
- Offline & CI evals
- Production sampling
- Continuous / online evaluation
- Trace / eval conversion
- Incident-derived evals
- Human feedback loops
- Agentic observability
- OpenTelemetry (OTel)
- Span design across agent / model / tool / MCP
- Correlation & trace IDs
- Structured logging
- Latency attribution
- Token & cost visibility
- Langfuse / LangSmith
- Direct & indirect prompt injection
- Agent goal hijacking
- Tool misuse & excessive agency
- Context poisoning
- Memory poisoning
- Identity & privilege abuse
- Credential leakage & data exfiltration
- Insecure inter-agent communication
- Agentic supply-chain risk
- Unsafe code execution
- Threat modelling
- Blast-radius reduction
- Trust boundaries & policy enforcement
- Human-in-the-loop for high-risk actions
- Agent red teaming

By the end of Week 5 you can answer: How do you know the agent got better rather than merely different? Which span introduced the latency? Which tool failed and how many times did it retry? What did this request cost? What stops the agent taking an unsafe action?

WEEK 06 / CORE FLAGSHIP BUILD

AgentOps, Deployment & Production Readiness

Build layer: Deployed System - containerised, gated, released, monitored and documented

AgentOps is the operational discipline around shipping and evolving agents. You containerise the API, workers and supporting services, wire CI/CD with automated tests and evaluation gates before release, and handle the versioning problem that ordinary software does not have: prompts, models, workflows and tools all version independently and all can regress. You ship a reference AWS deployment with Azure and GCP portability explained, then add monitoring, autoscaling, load and concurrency testing, rollback, incident response and a runbook. The week also covers model gateway and model routing as architecture patterns - provider abstraction, capability-, cost- and latency-aware routing, fallback models and provider failover. The core ends with a system that is actually deployed, not merely deployment-aware.

Concepts and engineering practice

- Docker & Docker Compose
- Containerising API + workers
- Environment-specific configuration
- Secrets management
- CI/CD pipeline
- Automated testing
- Evaluation gates before release
- Environment promotion
- Prompt versioning
- Model versioning
- Workflow versioning
- Tool versioning
- Rollback strategy
- Reference AWS deployment
- Azure / GCP portability
- Monitoring after deployment
- Autoscaling concepts
- Load & concurrency testing
- Incident response
- Runbooks
- Release management
- Model gateway
- Model routing (capability / cost / latency aware)
- Fallback models & provider failover
- Tiered inference
- Cost estimation
- Production-readiness review
- Final ADRs

You leave Week 6 with: a deployed Agentic AI system with API and async workers, a governed tool and MCP layer, engineered context and memory, an evaluation and regression suite, OpenTelemetry traces and cost visibility, security controls, a CI/CD pipeline with eval gates, an architecture diagram, ADRs, an operational runbook and a production-readiness report.

WEEK 07 / ADVANCED PRODUCTION MASTERY BONUS

Advanced Bonus: Enterprise Agent Architecture

Build layer: Enterprise patterns - human-in-the-loop, constrained multi-agent, A2A, and the agent control plane

The orchestration patterns serious enterprise systems actually use. Human-in-the-loop approval workflows, escalation and exception handling, supervisor patterns, and constrained multi-agent coordination - including an honest treatment of when not to use multi-agent architecture at all. You cover A2A and event-driven agent workflows, failure recovery across agents and services, and identity boundaries between them. Then the governance layer enterprises eventually need: an Agent Registry for discovery and control of agents, tools, MCP servers, capabilities, owners, versions and dependencies - the beginnings of an enterprise agent control plane. Closing with the vocabulary distinction that keeps architectures clean: a prompt is an immediate instruction, a skill is a reusable procedural capability, MCP is an interface to tools and context, and A2A is how agents talk to each other.

Concepts and engineering practice

- Human-in-the-loop workflows
- Escalation & exception handling
- Constrained multi-agent architectures
- A2A patterns
- Failure recovery across agents & services
- Agent Gateway as control plane
- Capability, owner & version governance
- Agent Skills vs prompts vs MCP vs A2A
- Approval patterns
- Supervisor patterns
- When NOT to use multi-agent
- Event-driven agent workflows
- Identity boundaries between agents
- Agent Registry
- Enterprise agent control plane
- Progressive disclosure in skills & context

By the end of Week 7 you can explain: when a second agent genuinely earns its place in an architecture, how approval and escalation are designed rather than bolted on, and why enterprises end up needing a registry and a control plane.

WEEK 08 / ADVANCED PRODUCTION MASTERY BONUS

Advanced Bonus: Production Hardening & Senior Architecture Review

Build layer: Architecture review, production-readiness report, portfolio case study and senior-level defence

Your flagship system goes in front of a review the way a senior architecture panel would run it: threat model, reliability, evaluation coverage, observability gaps, latency attribution, cost analysis, scalability bottlenecks, failure modes, identity and permission review, and the architecture decisions behind all of it. You then turn the implementation into the artefacts that outlast it - architecture diagram, ADRs, production-readiness report, portfolio case study and a senior system-design walkthrough. The goal is a specific and durable capability: being able to answer why was the system built this way , not merely which libraries did I use .

Concepts and engineering practice

- Threat-model review
- Reliability review
- Evaluation-coverage review
- Observability-gap review
- Latency analysis
- Cost analysis
- Scalability bottleneck analysis
- Failure-mode review
- Security review
- Identity & permission review
- Architecture decision review
- Production readiness
- Operational readiness
- Runbook review
- Architecture diagram
- ADR pack
- Production-readiness report
- Portfolio case study
- Senior system-design walkthrough
- Defending design trade-offs

You leave Week 8 able to: walk a senior engineer, an architect or an interview panel through the system end to end and defend every significant decision in it - including the ones you would make differently next time.

Review the live programme, current price and bonus allocation