Jashna Futurica Private Limited

GEN AI & AGENTIC AI

TRAINING PROGRAM

From LLMs to Autonomous Enterprise AI Systems

Elaborated Curriculum | Week 1 – Week 10

10	30+	10
Weeks	Topics	Labs

WEEK 01

Foundations of Agentic AI

Understanding what makes AI systems truly autonomous

Jashna Futurica Private Limited

Evolution: LLM Apps → Agentic AI

W01

Traditional LLM Apps

- Single-turn question-answer
- Stateless interactions
- No memory between sessions
- Manual orchestration required
- Fixed, pre-defined workflows
- Human decides every next step

Intermediate Systems

- Multi-turn conversations
- Basic context retention
- Simple tool calls (search, calc)
- Semi-automated pipelines
- Retrieval-Augmented Generation
- Prompt chaining patterns

Agentic AI Systems

- Autonomous goal pursuit
- Self-directed multi-step plans
- Dynamic tool selection
- Persistent memory & learning
- Self-reflection & correction
- Minimal human intervention

What Makes AI 'Agentic'?

W01

Core Property: Autonomy

- Acts without step-by-step instructions
- Decides own sequence of actions
- Adapts plan when obstacles arise
- Self-monitors progress toward goal
- Requests help only when truly stuck

Core Property: Goal-Driven

- Receives high-level objective
- Decomposes into sub-goals
- Tracks completion state
- Re-prioritizes dynamically
- Knows when goal is achieved

Core Property: Environment

- Perceives context via inputs
- Maintains world model internally
- Takes actions that affect outputs
- Reads feedback from environment
- Iterates based on observations

Components of an AI Agent

W01

🧠 Reasoning & Planning

- Breaks complex tasks into steps
- Chain-of-Thought processing
- Evaluates multiple approaches
- Selects optimal action sequence
- Handles ambiguity gracefully

🔧 Tool Usage & Memory

- Calls APIs, search, databases
- Executes code & reads files
- Short-term: current context
- Long-term: vector DB storage
- Episodic: event history recall

🔄 Reflection & Learning

- Evaluates own output quality
- Detects errors and self-corrects
- Improves prompts iteratively
- Learns from past task failures
- Updates strategies over time

Single-Agent vs Multi-Agent Systems

Single-Agent Architecture

- One LLM handles entire task
- Simpler to build and debug
- Lower latency for simple tasks
- Context window is the limit
- Good for focused, bounded tasks
- Examples: Customer support bot, code reviewer
- Bottleneck: one model, one context window

Multi-Agent Architecture

- Multiple specialized agents collaborate
- Parallel execution of sub-tasks
- Each agent has focused expertise
- Overcomes single-model limitations
- Coordinator manages agent routing
- Examples: Research → Write → Review pipeline
- Challenge: coordination & communication overhead

W01 | 🛠️ Hands-On Lab

Foundations of Agentic AI

Lab 1 **Build a Simple Reasoning-Based Agent**

Create an agent that accepts a high-level goal, decomposes it into steps using chain-of-thought, and executes each step sequentially with basic tool access (web search + calculator)

Lab 2 **Agent Anatomy Exploration**

Instrument your agent with print statements to trace: goal intake → planning → tool call → reflection → output. Visualize the decision loop in a flowchart

WEEK 02

LLM Reasoning & Planning Mechanisms

Teaching AI to think step-by-step and plan ahead

Jashna Futurica Private Limited

Chain-of-Thought & ReAct Framework

W02

Chain-of-Thought (CoT)

- Prompts model to 'think aloud'
- Intermediate reasoning steps visible
- Reduces errors on complex tasks
- Zero-shot: 'Let's think step by step'
- Few-shot: show worked examples
- Self-consistency: sample & vote on best answer

ReAct Framework

- Reason + Act interleaved loop
- Thought → Action → Observation cycle
- Grounded in real environment feedback
- Reduces hallucination significantly
- Supports tool use naturally
- Combines reasoning with external APIs

Tree-of-Thought (ToT)

- Explores multiple reasoning paths
- Branches at key decision points
- Evaluates and prunes weak paths
- Backtracking when path fails
- Best for: puzzles, multi-step math
- Higher compute but better accuracy

Planning Loops & Self-Reflection

W02

Planning Loops

- Goal decomposition into sub-tasks
- Dependency mapping between steps
- Sequential vs parallel execution
- Replanning when subtask fails
- Plan-and-Execute vs ReAct pattern
- MRKL: Modular Reasoning & Knowledge
- Task completion tracking & checkpoints

Self-Reflection & Correction

- Agent critiques its own output
- Reflexion framework: verbal RL
- Identifies errors before finalizing
- Re-tries with improved strategy
- Critic agent evaluates actor agent
- Constitutional AI-style self-checking
- Confidence scoring on outputs

Prompt-to-Plan Transformation

W02

Prompt Engineering for Plans

- Structured system prompts
- Role definition: 'You are a planner'
- Output format specification (JSON)
- Step count & constraint setting
- Example plans in few-shot prompt
- Iterative plan refinement via dialogue

Plan Representation

- JSON structured task lists
- DAG (Directed Acyclic Graph) format
- Numbered step sequences
- Conditional branch notation
- Dependency declarations
- Human-readable + machine-parseable

Plan Execution Engine

- Parse plan into executable steps
- Execute each step in order/parallel
- Capture outputs as context
- Pass results to next step
- Handle failures gracefully
- Log each step for observability

W02 | 🛠️ Hands-On Lab

LLM Reasoning & Planning Mechanisms

Lab 1 — **Build a Task-Planning Agent**

Given a complex goal (e.g. 'Write a market research report on EVs'), the agent must generate a structured JSON plan with steps, execute them via tool calls, and compile results

Lab 2 — **Implement Reasoning Trace Logging**

Add structured logging to capture every Thought → Action → Observation cycle. Visualize the trace as a timeline and analyze where the agent spent most tokens

WEEK 03

Tool Usage & Function Calling

Extending AI capabilities with real-world integrations

Jashna Futurica Private Limited

Tool Integration Architecture

W03

Tool Types

- Information retrieval (web search, DB)
- Computation (calculator, code exec)
- Communication (email, Slack APIs)
- File I/O (read/write documents)
- External APIs (weather, finance)
- Browser automation (Playwright)

Tool Registry Design

- Centralized tool catalog
- Schema-based tool definitions
- Tool capability descriptions
- Input/output type specifications
- Version management for tools
- Access control per tool

Tool Selection Logic

- LLM reads tool descriptions
- Matches task to best tool
- Single vs multi-tool per step
- Fallback tool chains
- Confidence threshold before calling
- Human approval for sensitive tools

Function Calling & Schema-Driven Outputs

W03

Function Calling in LLMs

- Define functions as JSON schemas
- LLM decides when & how to call
- Structured arguments auto-extracted
- Response parsed back into context
- Parallel function calls supported
- Error handling on bad arguments
- Native support: OpenAI, Anthropic, Gemini

Schema-Driven Outputs

- Force structured JSON responses
- Pydantic/dataclass validation
- Type checking on LLM output
- Retry on schema violation
- Enum constraints for choices
- Required vs optional fields
- Nested object schemas for complex data

API Orchestration & Safe Execution

W03

API Orchestration

- Chain multiple API calls together
- Pass outputs between API steps
- Rate limit management per API
- API key rotation and secrets mgmt
- Request batching for efficiency
- Response caching strategies

Error Handling in Tool Calls

- Catch & classify tool errors
- Timeout handling with retries
- Exponential backoff strategy
- Fallback to alternative tool
- Inform agent of failure reason
- Log all tool errors for debugging

Safe Execution Patterns

- Sandbox code execution (Docker)
- Input sanitization before tool call
- Output validation after tool call
- Budget limits on tool usage
- Irreversible action confirmation
- Human-in-loop for destructive ops

W03 | 🛠️ Hands-On Lab

Tool Usage & Function Calling

Lab 1 — **Build Tool-Aware Financial Calculator Agent**

Agent with tools: stock price API, currency converter, compound interest calculator, and news search. Given a query like 'What is AAPL worth in EUR today?', agent chains tools to answer

Lab 2 — **Multi-Tool Execution Pipeline**

Build a pipeline where 3+ tools are called in sequence: search → extract data → calculate → format report. Add retry logic and structured error logging for each tool

WEEK 04

Memory Systems for Agents

Giving agents the ability to remember and recall

Jashna Futurica Private Limited

Memory Types: Short-Term vs Long-Term

Short-Term (Working) Memory

- Lives in the LLM context window
- Temporary per-conversation state
- Current task + recent exchanges
- Tool call history in this session
- Limited by token window size
- Lost when conversation ends

Long-Term (Persistent) Memory

- Survives across sessions
- Stored in vector databases
- User preferences & profile
- Past task outcomes & decisions
- Entity relationships over time
- Indexed for semantic retrieval

Episodic Memory

- Remembers specific past events
- Timestamped experience records
- Contextual episode storage
- Retrieve by similarity to now
- Summarized episode compression
- Cross-session continuity

Vector Databases & Retrieval Strategies

W04

Vector Database Options

- Pinecone: managed, scalable
- Weaviate: schema-based, hybrid
- ChromaDB: lightweight, local-first
- Qdrant: high-performance Rust-based
- pgvector: Postgres extension
- FAISS: Facebook's in-memory index

Embedding & Indexing

- Convert text to dense vectors
- OpenAI/Cohere embedding models
- Chunking strategy affects recall
- Metadata filtering alongside vectors
- Cosine vs dot-product similarity
- Approximate nearest neighbor (ANN) search

Retrieval Strategies

- Top-K semantic search
- MMR: Maximum Marginal Relevance
- Hybrid: keyword + vector search
- HyDE: hypothetical doc embeddings
- Contextual compression of results
- Re-ranking with cross-encoders

Context Compression & Memory Pruning

W04

Context Compression

- Summarize old conversation turns
- Extract key facts, discard filler
- Progressive summarization technique
- LLMLingua token compression
- Dynamic context window management
- Store summaries, not raw text
- Balance detail vs token budget

Memory Pruning

- TTL (Time-to-Live) for memories
- Relevance scoring for retention
- Frequency-based importance weighting
- User feedback updates memory score
- Conflict resolution: newer wins
- Archival vs active memory tiers
- Privacy-compliant memory deletion

W04 | 🛠️ Hands-On Lab

Memory Systems for Agents

Lab 1 — **Add Persistent Memory to Agent**

Extend your Week 1 agent with ChromaDB or Pinecone. Agent should recall user name, preferences, and prior task outcomes across separate Python script runs

Lab 2 — **Build Contextual Memory Retrieval System**

Implement hybrid retrieval: BM25 keyword search + vector similarity. Add MMR to diversify results. Test with 100+ stored facts and measure recall accuracy vs baseline

WEEK 05

Multi-Agent Architectures

Designing systems where multiple AI agents collaborate

Jashna Futurica Private Limited

Multi-Agent Design Patterns

W05

Coordinator Pattern

- Central orchestrator agent
- Delegates tasks to worker agents
- Aggregates and synthesizes results
- Manages inter-agent communication
- Resolves task dependencies
- Single point of control & visibility

Supervisor–Worker Model

- Supervisor evaluates worker output
- Workers specialize in one domain
- Supervisor requests revisions
- Quality gate before final output
- Parallel workers, sequential review
- Human replaces supervisor optionally

Debate Pattern

- Multiple agents argue positions
- Adversarial quality checking
- Red team / Blue team dynamic
- Consensus emerges from debate
- Reduces individual agent bias
- Best for: fact-checking, analysis

Agent Communication & Conflict Resolution

W05

Distributed Agent Communication

- Message-passing via shared queue
- Publish-subscribe event model
- Direct agent-to-agent API calls
- Shared state via database
- Structured message schemas (JSON)
- Async communication for efficiency
- Message ordering & deduplication

Conflict Resolution

- Priority ranking among agents
- Voting on conflicting outputs
- Arbitrator agent as tiebreaker
- Timestamp-based precedence rules
- Human escalation on deadlock
- Confidence score comparison
- Domain expertise weighting

Collaborative Reasoning Patterns

W05

Role-Based Collaboration

- Researcher: gathers information
- Analyst: processes & interprets
- Writer: generates output content
- Reviewer: validates and edits
- Coordinator: manages workflow
- Each role has focused system prompt

Shared Memory Model

- Agents read/write shared context
- Blackboard architecture pattern
- Ordered contribution history
- Agents subscribe to relevant updates
- Prevents duplicate work
- Enables true collaboration

Evaluation & Quality

- Each agent output scored
- Cross-agent fact verification
- Consistency checks across agents
- Final aggregation strategy
- Confidence-weighted merging
- Human final review checkpoint

W05 | 🛠️ Hands-On Lab

Multi-Agent Architectures

Lab 1 — **Build Multi-Agent Research System**

3-agent pipeline: Researcher (web search) → Analyst (summarize & extract insights) → Writer (format report). Each agent is a separate LLM call with its own system prompt and tool access

Lab 2 — **Role-Based Agent Orchestration**

Build a Coordinator that dynamically routes tasks to 4 specialized agents based on task type. Implement conflict detection and resolution when agents produce contradictory outputs

WEEK 06

State Graph Orchestration

Using graph-based workflows for complex agent control flow

State Machines & Graph-Based Workflows

W06

State Machine Design

- Define discrete states (nodes)
- Transitions triggered by conditions
- Each state has entry/exit actions
- Explicit error/fallback states
- Deterministic behavior per state
- Finite State Machine (FSM) theory

Graph-Based Workflows

- Nodes = agents or action functions
- Edges = conditional transitions
- LangGraph-style implementation
- DAG (no cycles) vs cyclic graphs
- Subgraph composition
- Visual graph representation

Conditional Execution

- Branch based on LLM output
- If/else routing at graph edges
- Dynamic next-node selection
- Routing functions evaluate state
- Combine conditions with AND/OR
- Default fallback edges

Parallel Branches & Human-in-the-Loop

W06

Parallel Branch Execution

- Fan-out from single node
- Multiple branches run concurrently
- Fan-in aggregates results
- Python asyncio / threading
- Wait for all or first-complete
- Branch-specific error isolation
- Reduces total execution time

Human-in-the-Loop (HITL)

- Pause workflow at checkpoints
- Present state to human reviewer
- Accept: continue to next node
- Reject: route to revision node
- Modify: inject human corrections
- Approval gates for sensitive actions
- Audit trail of human decisions

Recovery Mechanisms & State Persistence

W06

Recovery Mechanisms

- Catch exception → recovery state
- Retry with exponential backoff
- Alternative path on repeated fail
- Checkpoint saves before risky steps
- Rollback to last good state
- Dead-letter queue for stuck flows

State Persistence

- Serialize state to JSON/DB
- Resume from any checkpoint
- Cross-session workflow continuation
- Durable execution (Temporal-style)
- State versioning & migration
- Audit log of all state transitions

Workflow Visualization

- Mermaid.js graph diagrams
- LangGraph built-in viz support
- Real-time execution highlighting
- State transition animation
- Bottleneck identification
- Shareable workflow documentation

W06 | 🛠️ Hands-On Lab

State Graph Orchestration (LangGraph-Based Thinking)

Lab 1 — **Build State-Driven Autonomous Workflow**

Implement a document processing pipeline as a state graph: Ingest → Classify → Extract → Validate → (if invalid: Correct → Validate) → Store. Include HITL at the Validate node

Lab 2 — **Visualize State Transitions**

Instrument your graph to log every node entry/exit with timestamp. Generate a Mermaid.js diagram from execution logs. Add color coding: green=success, red=error, yellow=pending

WEEK 07

Agent Observability & Debugging

Making AI agent behavior transparent and debuggable

Jashna Futurica Private Limited

Logging Agent Reasoning & Execution Tracing

W07

Logging Agent Reasoning

- Capture every CoT thought
- Log tool calls with input/output
- Record agent decision rationale
- Structured JSON logging (Loguru)
- Log levels: DEBUG/INFO/WARN/ERROR
- Correlate logs by session/run ID

Execution Tracing

- Distributed trace per agent run
- Span per: LLM call, tool call, node
- LangSmith / Langfuse / Phoenix
- Trace context propagation
- Parent-child span relationships
- End-to-end latency breakdown

Monitoring Token Usage

- Track prompt + completion tokens
- Per-agent token consumption
- Cost estimation in real-time
- Token budget enforcement
- Alert on budget overrun
- Usage dashboards (Grafana)

Latency Optimization & Failure Diagnostics

W07

Latency Optimization

- Identify slowest spans via traces
- Reduce prompt verbosity
- Parallel tool calls where possible
- Stream responses incrementally
- Cache repeated LLM calls
- Use smaller models for simple steps
- Async agent execution patterns

Failure Diagnostics

- Classify failure types: timeout/hallucination/tool error
- Trace to root cause node/span
- Reproduce failures with logged inputs
- Error rate dashboards by agent type
- Regression testing agent behavior
- A/B test prompt changes
- Alert on anomalous failure spikes

Guardrails & Safety Layers

W07

Input Guardrails

- Detect harmful/off-topic inputs
- PII detection and redaction
- Prompt injection detection
- Content category filtering
- Topic boundary enforcement
- Rate limiting per user

Output Guardrails

- Validate output against schema
- Hallucination detection checks
- Toxicity & bias scoring
- Fact grounding verification
- Confidence threshold filtering
- Human review escalation path

Runtime Safety

- Sandboxed code execution
- Tool call authorization checks
- Resource usage limits (CPU/mem)
- Data access permission model
- Irreversible action confirmation
- Kill switch for runaway agents

W07 | 🛠️ Hands-On Lab

Agent Observability & Debugging

Lab 1 — **Add Structured Logging with Loguru**

Instrument your multi-agent system with Loguru. Log every LLM call (prompt, response, tokens, latency) and tool call. Output structured JSON logs parseable by Grafana/ELK stack

Lab 2 — **Implement Execution Tracing with Langfuse**

Integrate Langfuse (or LangSmith) to create full traces. Visualize the trace waterfall. Identify which agent node consumes most tokens/time and optimize it

WEEK 08

Enterprise Integration & MCP Concepts

Connecting agentic AI to enterprise systems at scale

Jashna Futurica Private Limited

Agent + Backend APIs & RAG Integration

W08

Agent + Backend APIs

- RESTful API tool wrappers
- GraphQL query tool integration
- OAuth 2.0 auth for secure APIs
- API versioning awareness
- Pagination handling in tool calls
- Webhook-triggered agent runs

Agent + RAG Integration

- Retrieval-Augmented Generation
- Agent retrieves before generating
- Knowledge base as a tool
- Dynamic index updates
- Citation tracking from retrieval
- RAG evaluation: RAGAS metrics

Multi-Model Orchestration

- Route tasks to best-fit model
- GPT-4 for reasoning, Haiku for speed
- Specialized models: vision, code
- Cost-quality optimization routing
- Model fallback on failure
- Ensemble outputs from multiple models

Model Context Protocol (MCP) Concepts

MCP Basics

- Standardized tool/resource protocol
- Server exposes tools + prompts + resources
- Client (agent) discovers capabilities
- JSON-RPC 2.0 communication
- Tool schemas auto-described to LLM
- Anthropic open standard (2024)
- Growing ecosystem of MCP servers

MCP Architecture

- MCP Host (Claude Desktop, agent)
- MCP Client (in-process connector)
- MCP Server (exposes capabilities)
- stdio or HTTP/SSE transport
- Dynamic capability discovery
- Secure sandboxed execution
- Build custom MCP servers in Python/JS

Tool Chaining & Security Concerns

W08

Tool Chaining Architecture

- Output of Tool A feeds Tool B
- Intermediate result transformation
- Schema mapping between tools
- Conditional chain branching
- Parallel chain execution
- Chain-level error recovery

Security Concerns

- Prompt injection via tool outputs
- Data exfiltration via tool calls
- Credential exposure in logs
- OWASP LLM Top 10 awareness
- Least-privilege tool access
- Audit all tool invocations

Compliance Concerns

- GDPR: personal data in prompts
- SOC 2 data handling requirements
- Retention policy for LLM logs
- Right to erasure for memory
- Regulated industry guardrails
- Explainability requirements

W08 | 🛠️ Hands-On Lab

Enterprise Integration & MCP Concepts

Lab 1 — **Build MCP-Inspired Tool Orchestration Layer**

Create a tool registry server with dynamic tool discovery. Agent queries the registry at runtime, retrieves tool schemas, and selects appropriate tools for each task step. Implement 5+ tools.

Lab 2 — **RAG + Agent Integration**

Build an agent that queries a vector store (ChromaDB) of 500+ company documents, retrieves relevant chunks, and generates grounded answers with source citations in structured JSON output

WEEK 09

Scaling Agentic AI Systems

Deploying production-grade agents at enterprise scale

Jashna Futurica Private Limited

Async Execution & Concurrency

W09

Async Execution

- Python asyncio for non-blocking I/O
- Async LLM calls with aiohttp
- Event loop management
- async/await patterns in agents
- Concurrent tool calls per step
- Background task execution

Concurrency Handling

- Thread pool for CPU-bound tasks
- Process pool for isolation
- Semaphore limits on parallelism
- Deadlock prevention strategies
- Race condition handling
- Shared state synchronization

Rate Limiting & Caching

- Token-per-minute (TPM) limits
- Requests-per-minute (RPM) limits
- Token bucket algorithm
- LLM response caching (exact match)
- Semantic caching (similar queries)
- Redis for distributed cache

Cost Optimization & Cloud Deployment

W09

Cost Optimization

- Model tiering: use cheapest viable
- Prompt compression reduces tokens
- Batch API for non-urgent tasks
- Cache warm vs cold call costs
- Monitor cost per task/session
- Set hard budget limits via code

Cloud Deployment Patterns

- Serverless: Azure Functions / AWS Lambda
- Container: AKS / ECS / Cloud Run
- Kubernetes for orchestration
- Auto-scaling based on queue depth
- Multi-region for latency reduction
- Blue-green deployment for agents

Containerization (Docker)

- Dockerfile for agent application
- docker-compose for local testing
- Multi-stage builds for small images
- Environment variables for secrets
- Health checks & restart policies
- Volume mounts for persistent state

Production Readiness Checklist

W09

Infrastructure

- Horizontal pod autoscaling (HPA)
- Database connection pooling
- Message queue (Azure Service Bus / SQS)
- Secrets in Key Vault / AWS Secrets
- CI/CD pipeline for agent updates
- Infrastructure as Code (Terraform)
- Staging environment validation

Operations

- Centralized log aggregation (ELK)
- Distributed tracing (Jaeger/Tempo)
- Metrics dashboards (Grafana)
- Alerting on error rate spikes
- Runbook for common failures
- On-call rotation for prod issues
- SLA monitoring & reporting

W09 | 🛠️ Hands-On Lab

Scaling Agentic AI Systems

Lab 1 — **Deploy Agent System to Cloud (Azure/AWS)**

Containerize your Week 5 multi-agent system with Docker. Deploy to Azure Container Apps or AWS ECS with auto-scaling rules. Set up a queue-based trigger system for incoming tasks

Lab 2 — **Add Cost Monitoring Dashboard**

Instrument all LLM calls to report token usage to a SQLite/Postgres table. Build a Grafana dashboard showing: cost per run, cost per agent, cumulative daily spend, and model breakdown

WEEK 10

Capstone: Autonomous AI System

Apply all 9 weeks — build one production-grade system

Capstone Project Options

W10

01 **Autonomous Financial Risk Analyzer**

Multi-agent system: data fetcher → risk calculator → report writer. Integrates real-time market APIs, vector memory of historical risk events, cloud deployed.

02 **AI Research Copilot**

Agents: literature searcher → summarizer → gap analyzer → recommendation writer. Persistent memory of research history. MCP tool integration.

03 **Multi-Agent Resume Screening System**

Agents: parser → scorer → ranker → report generator. State graph with HITL approval gate. Deployed as REST API on Azure/AWS.

04 **Enterprise Document Intelligence Agent**

Ingest → classify → extract → validate → store pipeline. RAG over 1000+ docs. Structured output with audit trail. Compliance-aware guardrails.

05 **AI Startup MVP Agent Platform**

Build a mini agentic platform: tool registry, agent templates, state graph runner, observability dashboard. Full-stack with FastAPI backend.

Capstone Requirements & Evaluation

W10

Technical Requirements

- Multi-agent design (3+ agents)
- Minimum 3 tool integrations
- Persistent memory system
- State graph orchestration
- Observability & logging layer
- Cloud deployment (Azure/AWS)

Architecture Documentation

- System architecture diagram
- Agent interaction flowchart
- Data flow & storage design
- API & tool integration map
- Deployment architecture diagram
- Runbook & operational guide

Final Presentation

- Live demo of running system
- Architecture walkthrough (10 min)
- Key design decisions & tradeoffs
- Challenges faced & solutions
- Cost & performance metrics
- Q&A with panel (10 min)

Ready to Build

Autonomous AI Systems?

10 Weeks • 30+ Topics • 10 Hands-On Labs • 1 Production Capstone

Jashna Futurica Private Limited | Agentic AI Training Program