Agentic AI Framework Comparison Matrix

A practical worksheet to compare leading agentic AI frameworks based on capabilities, use cases, governance requirements, and technical fit.

1. Introduction

Agentic AI frameworks help teams design systems that can plan, reason, call tools, use enterprise data, coordinate multiple agents, and complete multi-step workflows. Unlike a simple chatbot that responds once to a prompt, an agentic system may perform a sequence of actions such as retrieving documents, validating information, updating a record, escalating an exception, or requesting human approval before execution.

This document is designed as a practical comparison worksheet. It helps business, technology, risk, compliance, and architecture teams evaluate frameworks such as LangGraph, CrewAI, AutoGen, Semantic Kernel, LlamaIndex, and other agent development platforms against the needs of a real project rather than selecting a tool based only on popularity or vendor messaging.

1.1 Why Framework Selection Matters

Choosing an agentic AI framework is an architectural decision, not just a developer preference. The framework influences how agents are designed, how workflows are controlled, how state is managed, how tools are called, how humans approve actions, and how failures are logged and recovered.

- **Reliability:** A framework with explicit state management and controllable workflow paths is better suited for high-impact processes where failures must be predictable and recoverable.

- **Governance:** Agentic systems can take actions, not just produce recommendations. This means the framework must support approval points, audit logs, policy enforcement, and escalation rules.
- **Maintainability:** Teams need to understand why an agent made a decision, which tools it used, and what data influenced the outcome.
- **Integration fit:** Some frameworks align better with Python ecosystems, some with Microsoft and .NET environments, and some with RAG-heavy document intelligence use cases.
- **Time to value:** A lightweight role-based framework may be faster for pilots, while a graph-based orchestration framework may be better for production-grade workflows.

Example: A customer support triage agent that classifies tickets and drafts responses may only need tool calling, retrieval, and human review. A claims-processing agent that checks policy documents, validates eligibility, updates a core system, and triggers payment approval requires stronger workflow control, auditability, role-based access, and human-in-the-loop checkpoints.

1.2 Why There Is No Single "Best" Agentic AI Framework

There is no universal best framework because agentic AI projects differ widely in risk, workflow structure, deployment model, enterprise ecosystem, and governance expectations. A framework that is excellent for fast multi-agent experimentation may not be the best choice for regulated enterprise automation. Similarly, a highly controlled

orchestration framework may be unnecessarily complex for a simple knowledge assistant.

- **LangGraph-style frameworks** are often appropriate when workflows require explicit states, branching, retries, checkpoints, and deterministic control paths.
- **CrewAI-style frameworks** are useful when the problem naturally maps to role-based teams, such as researcher, analyst, reviewer, and writer agents completing a sequence of tasks.
- **AutoGen-style frameworks** are useful for conversational multi-agent collaboration, simulations, research workflows, and interactive agent-to-agent problem solving.
- **Semantic Kernel-style frameworks** can be attractive for enterprise teams already invested in Microsoft, .NET, plugin-based architecture, and structured AI orchestration.
- **LlamaIndex-style workflows** are useful where document retrieval, knowledge indexing, RAG pipelines, and enterprise data grounding are central to the solution.

The selection question should therefore be: "Which framework best fits our specific use case, risk level, technical stack, governance model, and operating environment?" rather than "Which framework is most popular?"

1.3 How to Use This Comparison Matrix

Use this worksheet before committing to a framework. The goal is to convert a broad technology choice into a structured decision supported by business requirements, architecture constraints, governance needs, and delivery realities.

1. **Start with the use case:** Define the business problem, users, desired outputs, and decisions the agent may support or execute.
2. **Define autonomy level:** Decide whether the system will only recommend, draft, route, approve, update records, or trigger external actions.
3. **Map workflow complexity:** Identify whether the process is linear, branching, iterative, exception-heavy, or multi-agent.
4. **Assess governance requirements:** Document approval checkpoints, audit logs, escalation paths, data restrictions, and compliance obligations.
5. **Score candidate frameworks:** Compare frameworks using weighted criteria such as orchestration, integrations, observability, security, deployment readiness, and developer experience.
6. **Validate with a pilot:** Build a controlled proof of concept using realistic data, real tools, failure scenarios, and measurable success criteria.

Worksheet prompt: Before comparing frameworks, write a one-paragraph description of your agentic AI project. Include the business process, users, systems touched, data

used, actions allowed, and approvals required. This paragraph becomes the anchor for the rest of the matrix.

2. Define Your Agentic AI Project Requirements

Framework comparison should begin with requirements, not vendor features. Many failed agentic AI pilots start with a tool-first mindset: teams pick a framework, build an impressive demo, and later discover that the system cannot meet enterprise requirements for auditability, security, scale, or human oversight. A requirements-first approach reduces this risk.

This section helps you define the project characteristics that will drive framework selection. Each subsection includes guidance, examples, and practical prompts that can be used in workshops with product owners, architects, risk teams, and developers.

2.1 Primary Business Use Case

Begin by defining the business outcome the agentic AI system must support. The use case determines the required autonomy, data access, integration depth, governance controls, and success metrics. A vague use case such as "build an AI agent for operations" is not enough. A better use case describes the workflow, user, trigger, decision points, and expected output.

- **Knowledge assistant:** Answers questions using approved documents, policies, or knowledge articles.
- **Process automation agent:** Executes repeatable steps such as ticket routing, data validation, or status updates.
- **Decision-support agent:** analyzes information and recommends actions for a human to approve.

- **Multi-agent research system:** Coordinates specialized agents to gather, compare, critique, and synthesize information.
- **Operational control agent:** Takes or recommends actions in live systems, often requiring strong guardrails and audit trails.

Example: Instead of defining the use case as "AI for HR," define it as "An HR policy assistant that answers employee questions using approved policy documents, identifies when a query involves sensitive employee data, and escalates unresolved or high-risk questions to HR." This level of specificity makes framework comparison more meaningful.

Worksheet questions: What business process will the agent support? Who will use it? What decisions or actions will it perform? What does success look like in measurable terms, such as reduced handling time, improved accuracy, faster response, fewer escalations, or better compliance evidence?

2.2 Single-Agent vs Multi-Agent System

Decide whether the project needs one capable agent or multiple specialized agents. A single-agent design is usually easier to build, test, govern, and maintain. A multi-agent design may be useful when tasks require specialization, parallel work, critique, negotiation, or staged handoffs.

- **Single-agent fit:** FAQ answering, simple document retrieval, ticket classification, drafting summaries, or guided form completion.

- **Multi-agent fit:** complex research, software development workflows, compliance review, financial analysis, proposal generation, or tasks requiring independent reviewer and approver roles.

- **Risk consideration:** Multi-agent systems can increase complexity because agents may disagree, repeat work, pass incomplete context, or amplify errors across handoffs.

Example: A procurement policy assistant may work well as a single agent that retrieves policy clauses and drafts answers. A vendor risk review system may benefit from multiple agents: one checks vendor documentation, one reviews cybersecurity posture, one assesses financial risk, and one consolidates a final recommendation for a human reviewer.

Framework implication: If coordination between specialized agents is central to the use case, compare frameworks for native multi-agent support, communication patterns, task delegation, shared memory, conflict resolution, and observability across agent interactions.

2.3 Workflow Complexity and Orchestration Needs

Workflow complexity is one of the strongest indicators of framework fit. Some projects follow a simple sequence: receive input, retrieve information, generate a response, and ask a human to approve. Others require branching, retries, loops, exception handling, tool sequencing, and stateful execution across multiple steps.

- **Linear workflow:** The agent follows a predictable sequence with limited branching.
- **Branching workflow:** Different paths are selected based on data, user role, confidence score, risk level, or policy conditions.
- **Iterative workflow:** The agent repeats steps until a condition is met, such as collecting missing information or refining an answer.
- **Exception-heavy workflow:** The process must handle missing data, conflicting evidence, failed tool calls, policy violations, and escalation triggers.
- **Long-running workflow:** The process may span minutes, hours, or days and must preserve state across interruptions.

Example: A support-ticket summarization agent may use a simple linear flow. In contrast, an incident response agent may need to classify severity, retrieve runbooks, check system status, notify teams, request approval for remediation, execute approved actions, and document the outcome. This requires stronger orchestration, state tracking, and rollback planning.

Framework implication: For complex, stateful, or regulated workflows, prioritize frameworks that make control flow explicit, support checkpoints, allow deterministic routing, and provide clear visibility into each step of execution.

2.4 Human-in-the-Loop Requirements

Human-in-the-loop controls define when a person must review, approve, reject, override, or escalate an agent action. This is especially important because agentic systems may take actions in enterprise systems rather than merely generate recommendations.

- **Low-risk actions:** Drafting a response, summarizing a document, or suggesting tags may only require optional review.
- **Medium-risk actions:** Updating a ticket status, sending a non-sensitive notification, or assigning a task may require approval based on confidence or policy rules.
- **High-risk actions:** Financial transactions, legal commitments, access changes, HR decisions, security remediation, and customer-impacting actions should require explicit human authorization.
- **Escalation triggers:** Low confidence, policy conflict, sensitive data, unusual request, missing evidence, or failed tool call should route the workflow to a human.

Example: An AI agent may draft an email to a customer automatically, but it should not send a refund approval or change contract terms without human review. In a well-designed workflow, the framework pauses at the approval checkpoint, presents evidence and rationale, records the approver decision, and then proceeds only if authorized.

Framework implication: Compare whether each framework supports interruptible workflows, approval checkpoints, persistent state, audit trails, role-based authorization, and clear handoff between agent and human decision-maker.

2.5 Data, Tools, and External System Integrations

Agentic AI becomes valuable when it can use trusted data and tools. However, every integration increases complexity, risk, and governance requirements. The framework should make it easy to connect tools securely, pass context correctly, validate outputs, and restrict what the agent is allowed to do.

- **Data sources:** policy documents, knowledge bases, SharePoint libraries, databases, data warehouses, ticketing systems, CRM records, and email or chat history.
- **Tools and actions:** web search, database queries, workflow triggers, API calls, code execution, email drafting, ticket updates, document generation, or notification sending.
- **Security controls:** authentication, authorization, least-privilege access, data masking, logging, secrets management, and tool permission boundaries.
- **Quality controls:** source citation, confidence scoring, validation rules, schema checks, and fallback behavior when data is missing or unreliable.

Example: A finance operations agent may need to read invoices from a document repository, compare them with purchase orders in an ERP system, check vendor master

data, and create an exception report. This requires strong tool orchestration, identity controls, structured outputs, and evidence tracking.

Framework implication: Evaluate the framework's connector ecosystem, custom tool support, schema handling, permission model, RAG capabilities, compatibility with enterprise identity, and ability to log every tool call with inputs, outputs, and errors.

2.6 Existing Technology and Cloud Ecosystem

The best framework is often the one that fits your existing architecture, developer skills, platform standards, security model, and operational tooling. A technically strong framework can still be a poor choice if it does not align with the organization's cloud, programming languages, governance processes, or DevOps practices.

- **Microsoft-centered environments:** Teams using Azure, Microsoft 365, Entra ID, Power Platform, .NET, and Microsoft security tooling may prefer frameworks that integrate naturally with those services.
- **Python-first AI teams:** Teams already using Python, LangChain, vector databases, notebooks, and ML engineering workflows may prefer Python-native frameworks with rich community examples.
- **RAG-heavy environments:** Projects centered on document indexing, retrieval, metadata filtering, and knowledge grounding may prioritize frameworks with strong retrieval and indexing capabilities.

- **Enterprise architecture standards:** Consider approved languages, deployment environments, container strategy, monitoring tools, CI/CD pipelines, and internal security requirements.

Example: If an organization already runs key workflows through Azure, Microsoft 365, and enterprise identity controls, a Microsoft-aligned orchestration approach may reduce integration friction. If the data science team has deep Python and LangChain experience, LangGraph-style development may be more natural for complex agent workflows.

Framework implication: Score each framework not only on features, but also on organizational fit: developer skill availability, cloud compatibility, security review effort, production support model, vendor roadmap, and long-term maintainability.

2.7 Deployment and Scalability Requirements

Deployment requirements determine whether the framework can move from prototype to production. Agentic AI systems often need more than a working demo: they require secure deployment, monitoring, version control, cost controls, model routing, concurrency management, recovery from failure, and governance evidence.

- **Prototype scale:** A small pilot with limited users, synthetic data, and manual monitoring.
- **Department scale:** A business-unit deployment with real users, controlled integrations, service-level expectations, and support processes.

- **Enterprise scale:** A production system with multiple departments, sensitive data, audit requirements, resilience expectations, and integration with enterprise monitoring.

- **High-volume scale:** A system that must handle many concurrent requests, long-running tasks, tool failures, rate limits, cost ceilings, and model fallback strategies.

Example: A pilot agent that helps five analysts summarize research reports can be monitored manually. A production compliance agent that reviews thousands of cases per month needs traceability, queue management, exception handling, cost monitoring, versioned prompts, access control, and operational dashboards.

Framework implication: Evaluate deployment options, managed services, container support, async execution, retry handling, checkpointing, observability, cost management, testing support, versioning, and rollback capability before selecting a framework for production use.

3. Agentic AI Framework Evaluation Criteria

Evaluation criteria translate project requirements into a structured scoring model. Instead of comparing frameworks only by popularity, teams should assess how well each option supports orchestration, multi-agent design, memory, integrations, governance, deployment, and long-term maintainability.

3.1 Orchestration and Workflow Control

Orchestration determines how tasks move from one step to another. Strong orchestration support is important when workflows include branching, retries, loops, conditional routing, exception handling, or human approvals. For regulated and production-grade processes, explicit control flow is usually preferable to hidden or purely conversational execution.

- **Look for:** graph-based flows, conditional routing, reusable workflow steps, retries, parallel execution, timeout handling, and failure recovery.
- **Strong fit example:** an incident response agent that routes cases differently based on severity, affected system, business impact, and approval status.
- **Scoring prompt:** Can the framework make the workflow visible, testable, and recoverable?

3.2 Multi-Agent Capabilities

Multi-agent capability refers to how well the framework supports collaboration among specialized agents. This includes role definition, task delegation, message passing,

handoffs, shared context, reviewer patterns, and conflict resolution. Multi-agent design is useful when a workflow benefits from specialization, but it should not be used merely because it sounds advanced.

- **Look for:** agent roles, managed handoffs, supervisor-worker patterns, peer review loops, shared memory, and visibility into agent-to-agent interactions.
- **Strong fit example:** a proposal-generation system with separate researcher, pricing analyst, compliance reviewer, and final editor agents.
- **Risk note:** More agents can mean more coordination overhead, higher cost, and more difficult debugging.

3.3 Memory and State Management

Memory and state management determine what the agent remembers during and across workflow runs. State includes the current task, intermediate results, tool outputs, approval status, errors, and next action. Memory may include conversation history, user preferences, prior cases, or retrieved knowledge.

- **Look for:** typed state, checkpointing, session memory, long-term memory options, rollback, resumability, and clear state inspection.
- **Strong fit example:** a claims workflow that must pause while waiting for a missing document and resume later without losing prior validation results.

- **Governance note:** Persistent memory should follow data retention, privacy, and access-control policies.

3.4 Tool and API Integration

Tool and API integration is central to agentic AI. A framework should support safe, structured, and observable tool calling. This includes calling internal APIs, querying databases, retrieving documents, invoking workflow automations, sending notifications, or interacting with external services.

- **Look for:** function calling, tool schemas, input validation, API connectors, error handling, retries, permission boundaries, and support for emerging standards such as MCP-style tool access.
- **Strong fit example:** a finance agent that retrieves invoices, validates purchase orders, checks vendor records, and creates an exception ticket.
- **Security note:** Agents should never receive unrestricted access to tools or credentials.

3.5 Human-in-the-Loop Support

Human-in-the-loop support allows people to review, approve, reject, correct, or escalate agent actions. This is essential where decisions affect customers, employees, finances, security, legal obligations, or regulated processes.

- **Look for:** approval checkpoints, interruptible workflows, reviewer dashboards, rationale display, evidence packaging, decision logging, and escalation routing.

- **Strong fit example:** an access-management agent that recommends a permission change but waits for manager and security approval before execution.
- **Scoring prompt:** Can the framework pause safely and resume after a human decision?

3.6 Observability and Tracing

Observability helps teams understand what the agent did, why it did it, which model was used, which tools were called, what data was retrieved, and where failures occurred. Without observability, agentic systems are difficult to debug, monitor, audit, or improve.

- **Look for:** step-by-step traces, prompt and response logging, tool-call logs, latency metrics, token and cost tracking, error classification, evaluation hooks, and integration with monitoring platforms.
- **Strong fit example:** a support automation agent where managers need to review why certain tickets were escalated or misclassified.
- **Governance note:** Logs should avoid exposing sensitive data unnecessarily and should follow retention rules.

3.7 Governance and Security Controls

Governance and security controls ensure that agent behavior aligns with enterprise policy, regulatory obligations, and acceptable risk levels. Because agentic systems can

use tools and perform actions, governance must be designed into the workflow rather than added after launch.

- **Look for:** role-based access, least-privilege tool permissions, guardrails, input and output validation, policy checks, audit trails, secrets management, data masking, and sandboxed execution.
- **Strong fit example:** an HR agent that can answer policy questions but must not expose confidential employee records or make employment decisions.
- **Scoring prompt:** Can controls be enforced consistently across prompts, tools, memory, and external actions?

3.8 Model and Provider Flexibility

Model flexibility matters when organizations want to switch between providers, use different models for different tasks, control cost, meet data residency requirements, or avoid vendor lock-in. Some frameworks are model-agnostic, while others are optimized for a specific model provider.

- **Look for:** support for multiple LLM providers, local models, model routing, fallback models, embeddings flexibility, and configuration-based model switching.
- **Strong fit example:** using a lower-cost model for classification, a stronger model for reasoning, and a private model for sensitive data.

- **Trade-off:** Provider-specific frameworks may be simpler but less portable.

3.9 Deployment Options

Deployment options determine how easily a framework can move from local development to production. Teams should consider whether the framework supports cloud deployment, containers, serverless patterns, managed runtimes, private networks, enterprise identity, and production monitoring.

- **Look for:** container support, async execution, queue integration, managed services, environment configuration, CI/CD friendliness, scaling patterns, and rollback support.
- **Strong fit example:** a high-volume customer operations agent that must run reliably across many concurrent requests.
- **Scoring prompt:** Can the framework be operated by your platform and DevOps teams using existing practices?

3.10 Learning Curve and Developer Experience

Developer experience affects speed, quality, adoption, and long-term maintainability. A framework with excellent production controls may still fail if the team cannot learn it, test it, or troubleshoot it efficiently.

- **Look for:** clear documentation, examples, templates, type safety, debugging tools, testability, community support, and compatibility with current developer skills.

- **Strong fit example:** a business automation team may prefer a role-based framework that is easy to understand, while a platform engineering team may accept a steeper learning curve for greater control.

- **Trade-off:** Simpler abstractions are faster initially but may limit complex production workflows later.

3.11 Production Maturity and Ecosystem Support

Production maturity reflects whether a framework is ready for real enterprise workloads. Evaluate stability, release cadence, documentation quality, community adoption, enterprise references, ecosystem integrations, migration risk, and long-term roadmap.

- **Look for:** stable APIs, active maintenance, examples for production patterns, observability integrations, security guidance, enterprise support options, and migration paths.

- **Strong fit example:** a regulated workflow should favor a framework with checkpointing, traceability, supportability, and a strong ecosystem over an experimental library with limited documentation.

- **Scoring prompt:** Would your team be comfortable supporting this framework for the next three years?

4. Agentic AI Framework Comparison Matrix

The matrix below summarizes practical fit by framework. Ratings should be treated as directional because framework capabilities change quickly. Use the notes as a starting point, then validate against your own use case, governance requirements, cloud environment, and team skills.

Framework	Best Fit	Strengths	Watchouts	Typical Use Cases
LangGraph	Production-grade, stateful workflows	Explicit graph orchestration, durable state, branching, checkpoints, human approval patterns, strong LangChain ecosystem	Higher learning curve; requires careful workflow design	Regulated workflows, incident response, multi-step operations, complex automation
CrewAI	Fast role-based multi-agent	Intuitive agent roles, crews, tasks, delegation, easy	Less suitable for highly complex branching and strict state	Research teams, proposal drafting, analyst workflows,

	prototyping	business-process mapping	control unless extended	business automation pilots
OpenAI Agents SDK	OpenAI-native agent applications	Simple primitives, handoffs, guardrails, tracing, close fit with OpenAI models and tools	Provider alignment may limit portability if multi-provider flexibility is required	GPT-native assistants, tool-using agents, lightweight production agents
Google ADK	Google Cloud and Gemini-centered agent development	Designed for agent development with Google ecosystem alignment and model/tool integration	Best fit depends on commitment to Google Cloud and available enterprise patterns	Gemini-based agents, Google Cloud workflows, enterprise automation on Google stack
AutoGen	Conversational multi-agent	Flexible agent conversations, research-friendly patterns, strong	May be less ideal for new production builds if roadmap or	Research simulations, coding assistants,

	experimentation	history in multi-agent collaboration	maintenance status is uncertain	collaborative agent experiments
Microsoft Agent Framework	Microsoft enterprise agent architecture	Enterprise orientation, Microsoft ecosystem alignment, orchestration patterns, potential fit with Azure and Microsoft 365 environments	Teams should verify maturity, roadmap, and migration guidance for their environment	Enterprise agents, Microsoft-centered automation, governed business workflows
PydanticAI	Typed, Pythonic agent development	Strong schema validation, structured outputs, type-friendly developer experience	May require additional orchestration components for complex multi-agent workflows	Structured data extraction, validated outputs, Python services, typed tool calls

LlamaIndex Workflows	RAG-heavy and knowledge-centric agents	Strong document indexing, retrieval, knowledge grounding, data connectors, workflow support	Less focused on broad business-process orchestration than dedicated graph frameworks	Knowledge assistants, document intelligence, enterprise search, policy Q&A
Semantic Kernel	Plugin-based enterprise AI orchestration	Microsoft-aligned, plugin architecture, planning concepts, supports .NET and Python ecosystems	May need careful design for advanced agent runtime behavior	Enterprise copilots, plugin-based workflows, Microsoft and .NET environments
Strands Agents	Lightweight agent construction	Useful for composing agents with tools and model interactions in a focused way	Validate ecosystem depth, observability, and production references before adoption	Focused agent prototypes, tool-using assistants, experimental automation

Mastra	TypeScript-oriented agent applications	Developer-friendly for JavaScript and TypeScript teams, workflow and agent patterns	Production fit depends on ecosystem maturity and enterprise controls	Web application agents, TypeScript backends, product-integrated assistants
Haystack	Search, RAG, and pipeline-based AI systems	Mature retrieval pipelines, document search, modular components, production RAG history	Agentic orchestration may require additional design depending on complexity	Enterprise search, document Q&A, RAG pipelines, knowledge assistants
LangChain	Broad LLM application development	Large ecosystem, many integrations, chains, tools, retrieval support, rapid experimentation	Complex production agents may benefit from LangGraph for stronger state control	LLM apps, prototypes, tool integrations, RAG applications

smolagents	Minimal, lightweight agent experiments	Small abstraction surface, useful for simple agents and learning	May not include the enterprise controls needed for complex production use	Educational projects, simple tool agents, lightweight prototypes
Agno	Practical agent applications and automation	Designed for building agents with tools, memory, and workflows; useful for application builders	Assess ecosystem maturity, governance controls, and scalability for enterprise use	Business automation agents, application assistants, multi-tool workflows

4.1 LangGraph

LangGraph is a strong choice when the project requires explicit workflow control, stateful execution, branching, retries, checkpoints, and human approval points. It is especially suitable for production-grade agents where reliability, auditability, and recovery matter.

- **Best for:** complex multi-step workflows, regulated processes, long-running tasks, incident response, and systems that require deterministic routing.

- **Strength:** models agent behavior as a graph, making the flow easier to inspect, test, and modify.
- **Example:** a claims-review workflow that collects documents, validates eligibility, routes exceptions, requests approval, and records the final decision.

4.2 CrewAI

CrewAI is useful when the work naturally resembles a team of specialists. It allows teams to define agents by role, assign tasks, coordinate outputs, and quickly create multi-agent prototypes that business users can understand.

- **Best for:** rapid prototyping, analyst workflows, research synthesis, content generation, proposal drafting, and business-process simulations.
- **Strength:** intuitive role-based abstraction that is easy to explain and fast to implement.
- **Example:** a market-research crew with a researcher, competitor analyst, reviewer, and executive-summary writer.

4.3 OpenAI Agents SDK

OpenAI Agents SDK is a practical option for teams building directly on OpenAI models and tools. It is designed around simple agent definitions, tool use, handoffs, and guardrails, making it approachable for teams that want a lightweight path to agentic behavior.

- **Best for:** OpenAI-native applications, lightweight production agents, tool-using assistants, and workflows where OpenAI model capabilities are central.
- **Strength:** clear primitives for agents, handoffs, guardrails, and tracing.
- **Example:** a customer-service assistant that answers questions, calls account tools, and hands off complex cases to a specialist agent.

4.4 Google ADK

Google ADK is relevant for teams building agentic applications around Google Cloud, Gemini models, and Google's developer ecosystem. It is most attractive when the target deployment environment, data services, and AI platform are already Google-centered.

- **Best for:** Gemini-based agents, Google Cloud workflows, enterprise search and automation in Google environments.
- **Strength:** alignment with Google's cloud and AI stack.
- **Example:** a cloud operations assistant that uses Google Cloud data, logs, and Gemini-based reasoning to support incident triage.

4.5 AutoGen

AutoGen is known for conversational multi-agent collaboration. It is useful for experimentation, research, coding workflows, and systems where multiple agents interact through structured dialogue. Teams considering it for new production systems should verify current roadmap and support status before committing.

- **Best for:** research prototypes, coding assistants, simulations, and conversational agent patterns.
- **Strength:** flexible multi-agent conversation and iterative refinement.
- **Example:** a software-engineering assistant where a planner, coder, executor, and critic collaborate to solve a coding task.

4.6 Microsoft Agent Framework

Microsoft Agent Framework is relevant for enterprises invested in Microsoft technologies, especially Azure, Microsoft 365, Entra ID, and Microsoft developer tooling. It may be suitable where governance, enterprise identity, and Microsoft ecosystem integration are important selection factors.

- **Best for:** Microsoft-centered enterprise agents, governed workflows, Azure-based deployment, and business automation.
- **Strength:** alignment with Microsoft enterprise architecture and security ecosystem.
- **Example:** an internal operations agent that uses Microsoft 365 content, Azure services, enterprise identity, and approval workflows.

4.7 PydanticAI

PydanticAI is a good fit for Python teams that value structured outputs, type safety, schema validation, and predictable data handling. It is particularly useful when agent outputs must be validated before being used by downstream systems.

- **Best for:** typed Python services, structured extraction, validation-heavy workflows, and API-backed agents.
- **Strength:** strong schema discipline and developer-friendly validation.
- **Example:** an invoice-processing agent that must return validated invoice fields in a strict schema before creating an ERP exception record.

4.8 LlamaIndex Workflows

LlamaIndex Workflows are well suited for knowledge-intensive applications where document retrieval, indexing, metadata filtering, and grounded answers are central. It is particularly useful when the agent needs to reason over internal documents, policies, manuals, or structured knowledge bases.

- **Best for:** RAG systems, document intelligence, enterprise knowledge assistants, policy Q&A, and research assistants.
- **Strength:** strong retrieval and data-grounding capabilities.
- **Example:** a compliance assistant that retrieves policy clauses, compares them against a user question, and generates a cited answer.

4.9 Semantic Kernel

Semantic Kernel is useful for teams that want plugin-based AI orchestration within Microsoft-aligned development environments. It supports patterns where AI capabilities are composed with conventional software functions, making it familiar to enterprise developers.

- **Best for:** enterprise copilots, plugin-based workflows, .NET and Python teams, and Microsoft-centered AI applications.
- **Strength:** plugin architecture and alignment with conventional software engineering patterns.
- **Example:** a sales enablement copilot that calls CRM plugins, retrieves approved content, and drafts customer-specific messaging.

4.10 Strands Agents

Strands Agents can be considered for lightweight agent construction where the team needs focused tool use, model interaction, and agent behavior without adopting a heavy orchestration layer. It may be useful for experimentation and targeted application features.

- **Best for:** lightweight assistants, focused tool agents, and small-to-medium prototypes.
- **Strength:** lower abstraction overhead for simple agent patterns.
- **Example:** a developer assistant that calls a small set of approved tools to answer operational questions.

4.11 Mastra

Mastra is relevant for JavaScript and TypeScript teams building agentic features into web applications or product backends. It can be attractive where the engineering team

prefers TypeScript-native workflows and wants to integrate agents closely into application code.

- **Best for:** product-integrated agents, TypeScript applications, web backends, and workflow automation features.
- **Strength:** strong fit for teams already standardized on JavaScript and TypeScript.
- **Example:** a SaaS application assistant that helps users configure workflows, explain product settings, and call application APIs.

4.12 Haystack

Haystack is a mature option for retrieval, search, and pipeline-based AI applications. It is especially suitable when the core requirement is to retrieve relevant documents, assemble context, and generate grounded responses.

- **Best for:** enterprise search, document Q&A, RAG pipelines, knowledge assistants, and production retrieval workflows.
- **Strength:** mature retrieval and pipeline design.
- **Example:** a legal knowledge assistant that retrieves precedent documents and summarizes relevant clauses for a reviewer.

4.13 LangChain

LangChain remains a broad ecosystem for building LLM applications with tools, chains, retrieval, and integrations. It is useful for experimentation and general LLM application

development, while LangGraph is often preferred when stronger stateful orchestration is required.

- **Best for:** prototypes, LLM applications, RAG, tool integration, and rapid experimentation.
- **Strength:** large ecosystem and broad integration coverage.
- **Example:** a prototype knowledge assistant that uses a vector database, document loader, prompt template, and tool calls.

4.14 smolagents

smolagents is useful when teams want a minimal, lightweight way to experiment with agents. It is appropriate for learning, simple tool use, and small prototypes where heavy orchestration would add unnecessary complexity.

- **Best for:** educational projects, small agents, simple tool calling, and lightweight experimentation.
- **Strength:** minimal abstraction and quick setup.
- **Example:** a simple research assistant that searches, summarizes, and returns a concise answer.

4.15 Agno (Formerly Phidata)

Agno, formerly known as Phidata, is designed for building practical agent applications with tools, memory, and workflows. It can be attractive for teams that want to develop business automation agents without building every component from scratch.

- **Best for:** business automation, application assistants, multi-tool agents, and practical prototypes.
- **Strength:** application-oriented agent development with useful building blocks.
- **Example:** an operations assistant that searches documents, calls internal tools, remembers context, and drafts status updates.

6. Governance and Production Readiness Comparison

Governance and production readiness determine whether an agentic AI framework can be used safely beyond a pilot. A framework may be impressive in a demonstration, but production use requires controls for human oversight, auditability, tool access, memory, failure recovery, security, and deployment operations. Industry guidance increasingly emphasizes that agentic AI shifts governance from simply validating model outputs to controlling actions, tool use, and accountability across the lifecycle.

6.1 Human Approval and Escalation Support

Human approval and escalation support are essential whenever an agent can affect customers, employees, finances, systems, compliance outcomes, or sensitive records. The framework should allow the workflow to pause at predefined checkpoints, present evidence and rationale, capture the human decision, and resume only after approval.

- **Evaluate whether the framework supports:** approval checkpoints, reviewer assignment, escalation routing, timeout handling, override decisions, and decision logging.
- **Strong production pattern:** low-risk and reversible actions may be automated, while high-impact actions require explicit approval.
- **Example:** a service agent may draft a refund response automatically, but any actual refund approval above a defined threshold should be routed to a human approver.

6.2 Observability and Auditability

Observability makes agent behavior understandable, measurable, and auditable. Production teams need to know what the agent did, which data it used, which tools it called, what decisions it made, what errors occurred, and what costs were incurred. Without this visibility, teams cannot reliably debug incidents, prove compliance, or improve performance.

- **Evaluate whether the framework supports:** traces, logs, tool-call records, prompt and response capture, latency metrics, token usage, cost attribution, error categories, and evaluation results.
- **Audit-ready evidence:** logs should show the user request, agent plan, retrieved sources, tool inputs, tool outputs, approval decisions, and final action.
- **Example:** if a ticket was escalated incorrectly, an operations manager should be able to review the trace and identify whether the error came from retrieval, classification, tool failure, or business rule logic.

6.3 Tool Access and Permission Controls

Tool access is one of the highest-risk areas in agentic AI because agents may call APIs, update records, send messages, execute code, or trigger workflows. Production systems should enforce least privilege, scoped permissions, identity-aware access, and dispatch-time policy checks before a tool action occurs.

- **Evaluate whether the framework supports:** tool schemas, permission boundaries, user-context authorization, secrets management, allowlists, denylists, sandboxing, and policy checks before tool execution.

- **Strong production pattern:** separate read-only tools from write-enabled tools and require higher approval for irreversible actions.

- **Example:** an HR policy assistant may read approved policy documents but should not access confidential employee records unless the user is authorized and the workflow has a valid business purpose.

6.4 State and Memory Governance

State and memory governance define what the agent is allowed to remember, for how long, where it is stored, and who can access it. This matters because persistent memory can contain sensitive user context, intermediate reasoning, retrieved documents, tool outputs, or business decisions.

- **Evaluate whether the framework supports:** typed state, checkpointing, session isolation, memory expiration, encryption, access control, deletion, and audit of memory updates.

- **Strong production pattern:** store only what is needed for the workflow and apply data minimization to long-term memory.

- **Example:** a claims agent may need to remember the current claim status and missing documents, but it should not store unnecessary personal details beyond retention requirements.

6.5 Failure Handling and Recovery

Failure handling determines how safely the agent behaves when something goes wrong. Common failures include tool errors, missing data, conflicting evidence, model hallucinations, low confidence, rate limits, approval timeouts, and downstream system outages. A production-ready framework should support graceful degradation rather than uncontrolled continuation.

- **Evaluate whether the framework supports:** retries, fallbacks, timeouts, circuit breakers, exception routing, rollback logic, idempotency, checkpoint recovery, and incident replay.
- **Strong production pattern:** if an action cannot be completed safely, the agent should stop, preserve evidence, and escalate to a human.
- **Example:** if a payment validation tool fails, the agent should not infer approval from incomplete data; it should mark the case as blocked and route it for review.

6.6 Security and Deployment Considerations

Security and deployment readiness determine whether the framework can be operated in the organization's production environment. This includes identity integration, network controls, secrets handling, containerization, environment separation, monitoring, change management, and release governance.

- **Evaluate whether the framework supports:** enterprise identity, role-based access, private network deployment, secure secrets, environment variables, container deployment, CI/CD, version control, and staged rollout.
- **Strong production pattern:** deploy agents through the same release, testing, and change-control processes used for other enterprise software.
- **Example:** a compliance agent should have separate development, test, and production environments, with production access controlled through approved identities and monitored service accounts.

7. Framework Scoring Worksheet

The scoring worksheet converts qualitative discussion into a transparent decision. It helps stakeholders compare shortlisted frameworks consistently and document why a framework was selected, rejected, or deferred. The worksheet should be completed by a cross-functional group that includes business owners, solution architects, developers, security, compliance, operations, and end-user representatives.

7.1 Assign Importance Weights to Evaluation Criteria

Not all criteria matter equally. A knowledge assistant may prioritize retrieval quality and ease of integration, while a regulated automation workflow may prioritize auditability, human approval, and state management. Assigning weights forces the team to agree on what matters most before comparing vendors or frameworks.

Evaluation Criterion	Suggested Weight	Adjust for Your Use Case
Orchestration and workflow control	15%	
Multi-agent capabilities	8%	
Memory and state management	10%	
Tool and API integration	12%	

Human-in-the-loop support	12%	
Observability and tracing	10%	
Governance and security controls	13%	
Model and provider flexibility	5%	
Deployment options	7%	
Learning curve and developer experience	4%	
Production maturity and ecosystem support	4%	
Total	**100%**	

7.2 Score Your Shortlisted Frameworks

After assigning weights, score each shortlisted framework against each criterion. Use a consistent scale so the comparison remains transparent. A simple 1-to-5 scale works well for workshops and executive reviews.

- **1 = Weak fit:** significant gaps or heavy customization required.
- **2 = Limited fit:** partially supports the need but creates risk or effort.
- **3 = Adequate fit:** meets the minimum requirement with manageable limitations.
- **4 = Strong fit:** supports the requirement well with minor gaps.
- **5 = Excellent fit:** strongly supports the requirement and aligns with the target architecture.

7.3 Calculate Weighted Scores

Weighted scoring combines importance and performance. Multiply each criterion score by its weight, then add the weighted scores to calculate the total. The highest score is not automatically the final answer, but it provides an evidence-based starting point for decision discussion.

Criterion	Weight	Framework A Score	Framework A Weighted Score	Framework B Score	Framework B Weighted Score
Orchestration and workflow control	15%				

Multi-agent capabilities	8%				
Memory and state management	10%				
Tool and API integration	12%				
Human-in-the-loop support	12%				
Observability and tracing	10%				
Governance and security controls	13%				

Model and provider flexibility	5%				
Deployment options	7%				
Learning curve and developer experience	4%				
Production maturity and ecosystem support	4%				
Total	**100%**				

7.4 Identify Critical Trade-Offs

A framework with the highest score may still involve important trade-offs. Decision-makers should explicitly document what they are accepting, avoiding, or deferring. This prevents hidden assumptions from becoming production risks later.

- **Speed versus control:** a simple framework may accelerate pilots but lack deep orchestration controls.
- **Provider alignment versus portability:** a provider-native framework may simplify development but create migration challenges.
- **Flexibility versus governance:** highly flexible frameworks may require more custom guardrails and monitoring.
- **Developer familiarity versus production maturity:** a familiar library may not be the strongest choice for regulated deployment.
- **Multi-agent richness versus operational complexity:** many agents can improve specialization but make testing and auditing harder.

7.5 Document Risks and Limitations

Every framework decision should include a risk register. The purpose is not to block innovation, but to make known limitations visible and manageable before production. Risks should be assigned an owner, mitigation, target date, and decision status.

Risk or Limitation	Impact	Mitigation	Owner	Status
Limited production references	Uncertain enterprise reliability	Run extended pilot with		

		operational monitoring		
Weak audit trail	Compliance evidence may be insufficient	Add external tracing and logging layer		
Provider lock-in	Reduced portability and negotiation leverage	Design model abstraction and fallback strategy		
Insufficient human approval controls	High-risk actions may occur without review	Add approval workflow before write actions		
Limited internal skills	Slower delivery and support risk	Plan enablement, documentation, and support model		

8. Final Framework Decision Matrix

The final decision matrix records the recommended framework choice and the evidence behind it. It should be completed after requirements definition, scoring, governance review, and proof-of-concept planning. This section can be used as an executive decision record or architecture review artifact.

8.1 Top Framework Choice

Document the primary recommended framework and the use case it will support. The decision should be tied directly to the project requirements rather than general framework popularity.

Decision prompt: The selected framework is _____ because it best supports our requirements for _____, _____, and _____.

8.2 Alternative Framework

Identify the second-best option or fallback framework. This is useful if the top choice fails proof-of-concept testing, procurement review, security review, or operational validation.

Decision prompt: The alternative framework is _____. It remains viable if _____, but it has limitations related to _____.

8.3 Key Reasons for Selection

- **Business fit:** the framework supports the target process, users, and decision flow.

- **Technical fit:** the framework aligns with existing architecture, data sources, tools, cloud, and developer skills.
- **Governance fit:** the framework supports auditability, approvals, permissions, and policy enforcement.
- **Operational fit:** the framework can be deployed, monitored, supported, and improved in production.
- **Scalability fit:** the framework can support future volume, use cases, and organizational expansion.

8.4 Known Trade-Offs

Known trade-offs should be accepted explicitly. If the selected framework has a steeper learning curve, limited connectors, provider dependency, or weaker built-in observability, these trade-offs should be recorded with mitigation actions.

- Which capability gaps require custom engineering?
- Which risks must be resolved before production?
- Which limitations are acceptable for the pilot but not for scale?
- Which dependencies could create lock-in or operational risk?
- Which governance controls must be added outside the framework?

8.5 Governance Requirements

List the minimum governance requirements that must be satisfied before the framework can be used in production. These requirements should be testable, assigned to owners, and reviewed before release.

- Defined agent purpose, scope, and prohibited actions.
- Human approval workflow for high-risk or irreversible actions.
- Tool permissions mapped to least-privilege access.
- Audit logs for prompts, tool calls, approvals, errors, and final actions.
- Memory retention and deletion policy.
- Security review for data access, secrets, identity, and deployment environment.
- Incident response process for agent errors, misuse, or policy violations.
- Monitoring dashboard for performance, cost, failures, and business outcomes.

8.6 Proof-of-Concept Requirements Before Adoption

Before full adoption, run a proof of concept using realistic data, realistic users, realistic tool access, and realistic failure scenarios. A successful proof of concept should validate not only output quality, but also governance, operations, security, and supportability.

PoC Requirement	Success Evidence	Owner	Status
Representative business workflow	Agent completes realistic end-to-end cases with acceptable accuracy		
Human approval checkpoint	Workflow pauses, presents rationale, captures approval, and resumes correctly		
Tool permission control	Agent can access only approved tools and cannot perform blocked actions		
Failure scenario testing	Tool failure, missing data, and low-confidence cases escalate safely		

Observability evidence	Trace shows prompts, data retrieval, tool calls, errors, approvals, and final action		
Security review	Identity, secrets, data access, and deployment controls are approved		
Operational readiness	Support model, monitoring, incident response, and rollback approach are documented		

9. Framework Selection Checklist

Use this checklist as a final readiness review before selecting an agentic AI framework. The purpose is to confirm that the decision is balanced across technology, business value, governance, people readiness, and production operations. A framework should not be selected only because it is popular, new, or easy to demonstrate; it should be selected because it can safely and sustainably support the target use case.

9.1 Technical Fit

- Does the framework support the required workflow pattern: linear, branching, iterative, long-running, or multi-agent?
- Can it integrate with the required data sources, APIs, tools, vector stores, ticketing systems, databases, and enterprise applications?
- Does it provide sufficient state management, checkpointing, memory control, and recovery capability?
- Can it work with the preferred model providers, embedding models, and deployment environments?
- Does it support structured outputs, validation, testing, and repeatable execution?

Example: If the project requires a compliance workflow with branching decisions, approval checkpoints, and audit evidence, a graph-based or strongly orchestrated framework may be a better fit than a lightweight prototype-oriented library.

9.2 Business Fit

- Does the framework support the business outcome, not just the technical demonstration?
- Can it improve speed, quality, accuracy, compliance evidence, cost control, or customer experience?
- Is the use case specific enough to evaluate framework fit objectively?
- Are business owners comfortable with the level of autonomy the framework enables?
- Does the framework allow clear ownership of agent decisions, outputs, and escalations?

Example: For an HR policy assistant, business fit means the framework can answer policy questions reliably, escalate sensitive cases, avoid unauthorized employee data access, and produce usage evidence for HR and compliance teams.

9.3 Governance Readiness

- Are human approval checkpoints defined for high-risk actions?
- Can tool access be restricted using least-privilege permissions?
- Are prompts, tool calls, retrieved sources, decisions, approvals, and errors logged?

- Is memory governed by retention, privacy, deletion, and access-control rules?
- Are prohibited actions, escalation triggers, and fallback procedures documented?
- Can the framework support audit evidence for internal review, risk assessment, or regulatory compliance?

Governance rule of thumb: If the agent can take an action that a human would need authorization to perform, then the agent workflow should include equivalent or stronger authorization, logging, and review controls.

9.4 Team Skills and Learning Curve

- Does the team already have skills in the framework's primary language, such as Python, TypeScript, or .NET?
- Are documentation, examples, templates, and debugging tools strong enough for the team's experience level?
- Can the framework be supported by internal platform, security, and operations teams?
- Is there a realistic enablement plan for developers, reviewers, and business owners?
- Will the team be able to test, troubleshoot, and maintain the solution after the initial pilot team moves on?

Example: A Python-first AI engineering team may adopt LangGraph, LlamaIndex, or PydanticAI more quickly. A Microsoft-centered enterprise development team may prefer Semantic Kernel or Microsoft-aligned agent tooling because it fits existing architecture, identity, and developer practices.

9.5 Production and Scalability Requirements

- Can the framework be deployed securely in the target environment?
- Does it support monitoring, tracing, alerting, cost tracking, and incident response?
- Can it handle expected request volume, concurrency, long-running tasks, and rate limits?
- Are rollback, versioning, prompt management, and release controls available or implementable?
- Can the solution be scaled from pilot to department or enterprise use without major re-architecture?

Selection checkpoint: If a framework performs well in a demo but cannot be monitored, governed, secured, or supported at scale, it should remain a prototype option rather than a production choice.

Final Recommendation and Next Steps

The final recommendation should be evidence-based, practical, and revisited over time. Agentic AI frameworks are evolving rapidly, so the best decision is not a permanent declaration; it is a documented choice based on today's use case, constraints, governance expectations, and proof-of-concept evidence.

Run a Small Proof of Concept

Start with a narrow proof of concept that reflects a real business workflow. Avoid building a broad demonstration that looks impressive but does not test the constraints that matter in production. The proof of concept should use representative inputs, realistic tools, expected user roles, and measurable success criteria.

- Select one high-value but bounded process.
- Define the agent's allowed actions and prohibited actions.
- Use realistic test data and realistic exception cases.
- Measure accuracy, completion time, escalation rate, tool-call reliability, cost, and user satisfaction.
- Compare at least two shortlisted frameworks when feasible.

Validate the Highest-Risk Capabilities

Do not validate only the easiest path. The proof of concept should intentionally test the areas most likely to fail or create risk. This includes failed tool calls, missing documents,

conflicting information, low-confidence outputs, sensitive data requests, and approval delays.

- Can the framework stop safely when required evidence is missing?
- Can it escalate when confidence is low or policy conflict exists?
- Can it recover from failed API calls without duplicating or corrupting actions?
- Can it prevent unauthorized access to sensitive tools or data?
- Can it preserve a trace that explains what happened?

Test Governance and Human Oversight

Governance should be tested as part of the proof of concept, not added later. Human approval, audit logging, tool permission controls, memory retention, and escalation rules should be exercised using realistic scenarios.

- Verify that high-risk actions pause for approval.
- Confirm that reviewers can see the evidence and rationale behind the recommendation.
- Test whether approval, rejection, override, and escalation decisions are logged.
- Confirm that tool access respects user role, system permissions, and policy boundaries.

- Review memory and logs for sensitive data exposure.

Document the Framework Decision

The framework decision should be written down as an architecture and governance record. This decision record helps future teams understand why a framework was selected, what assumptions were made, what risks were accepted, and what controls are required before production deployment.

- Selected framework and alternative framework.
- Target use case and scope of autonomy.
- Evaluation criteria and scoring results.
- Governance requirements and approval rules.
- Known trade-offs, risks, and mitigations.
- PoC findings and production readiness decision.
- Review date and decision owner.

Review the Decision as the Agentic AI Ecosystem Evolves

Agentic AI frameworks, model capabilities, tool standards, observability practices, and governance expectations are changing quickly. A good framework decision should therefore include a periodic review cadence. The goal is not to switch tools frequently, but to ensure the selected framework continues to meet business, technical, and governance needs.

- Review the decision after the proof of concept and again before production rollout.
- Reassess framework fit every six to twelve months, or sooner if the use case changes significantly.
- Monitor framework release notes, security advisories, ecosystem maturity, and enterprise support options.
- Update the scoring worksheet when new requirements, risks, or deployment constraints emerge.
- Retire or replace the framework if it no longer meets reliability, governance, or scalability expectations.

Final recommendation: Select the framework that best fits the use case, risk profile, operating environment, and team capability-not the framework with the most attention. For simple pilots, prioritize speed and learning. For enterprise production, prioritize orchestration control, governance, observability, secure tool access, and long-term supportability.

GSDC
Global Skill Development Council

AGENTIC AI FOUNDATION CERTIFICATION

AGENTIC AI FOUNDATION, BASED ON THE PRINCIPLES OF ETHICS AND RESPONSIBILITY, DRIVES AI INNOVATION.

ABOUT GSDC CERTIFICATION

LIFETIME VALIDITY

GSDC Certification is an globally accreditted certification with lifetime validity.

EBOOK

Extensive and exclusive Ebook created by world's experts to help you with understanding core concepts.

CREATED BY EXPERTS

GSDC certifications are created and authored by world's leading experts in the field.

LEARNING MATERIALS

Get access to learning materials such as videos, ebooks, templates, and practice exams, which will help you clear the certification exam.

LEARNING OBJECTIVE

- Access ready-to-implement templates for agentic AI solutions.
- Develop a deep understanding of agentic AI principles.
- Prepare for real-world challenges with agentic AI applications.

Enroll now with the code **LEARN20** To avail **20%** discount

Enroll Now

www.gsdcouncil.org