

NTT DATA

October 2026

# Enterprise agentic readiness: Exploring best practices and frameworks for agentic AI

Dr Anindita Desarkar
Somsuvra Chatterjee
Vishwanathan Raman

# Contents

1. Introduction 3

2. Challenges in implementing agentic AI in production 4

3. Unified production assurance lifecycle, from readiness to deployment 7

4. Agentic AI Production Readiness Framework 9

4.1. Agentic AI Readiness Checklist

4.2. Moving agents to production

4.3. Proposed architectural diagram for agentic application

5. Agentic AI Architectural Index Assessment Framework 16

5.1. Motivation for the assessment

5.2. Compiling the Architectural Index Score

5.3. Sample implementation

6. Quantitative Trust Score Framework 22

6.1. The four dimensions of the score

6.2. Diagnostic blueprint: From metric to action

7. Conclusion 25

8. List of abbreviations 26

9. References 27

 © 2026 NTT DATA, Inc. 

# 1. Introduction

Agentic AI empowers autonomous and semiautonomous systems to reason dynamically, decompose complex goals, execute multistep tool calls, coordinate across multiagent networks and automate end-to-end business workflows with minimal human oversight. As enterprises move beyond passive copilots and transactional chatbots toward stateful, goal-driven agentic AI systems, the strategic focus is shifting from isolated sandbox experimentation to production-grade deployment at scale.

While the transformative potential of autonomous agents is significant, deploying agentic architecture into live, business-critical environments introduces unique system-level operational challenges.

### These are categorized into four major pillars:

**Pillar 1: Strategic and system design layer**

Focuses on misalignment between business objectives, environmental assumptions and deployment architecture.

**Pillar 2: Reasoning, state and retrieval layer**

Focuses on state corruption, context degradation and poor grounding data.

**Pillar 3: Runtime, tooling and multiagent network layer**

Focuses on execution mechanics, API interactions and interagent communication dynamics.

**Pillar 4: Governance, security and observability layer**

Focuses on operational visibility, safety controls and continuous deployment risks.

This paper addresses the key challenges in each of these pillars by introducing a set of practical frameworks that support organizations throughout the entire agentic AI lifecycle.

### The key frameworks presented in this paper are as follows:

- **Design and predevelopment stage:** Agentic AI Production Readiness Framework and Agentic AI Readiness Checklist to assess organizational, architectural, governance and operational preparedness before development begins.
- **Postdevelopment and predeployment stage:** Quantitative Trust Score Framework and Architectural Index Measuring Framework for Agentic AI Systems to evaluate trustworthiness, risk posture, governance adherence and production readiness prior to deployment.

Collectively, these frameworks offer a comprehensive and systematic methodology for building, governing, validating and operationalizing agentic AI systems at enterprise scale, while ensuring alignment with business objectives, security requirements and responsible AI principles.



# 2. Challenges in implementing agentic AI in production

Figure 1 summarizes the key system-level challenges encountered when deploying agentic AI in production.[1,2,3]

**Figure 1: Challenges when implementing agentic AI in production**

| | |
|---|---|
| **Pillar 1: Strategic and system design layer** | Unbounded business case |
| | Mock APIs and real production environment |
| | Stateless microservice bottlenecks |
| **Pillar 2: Reasoning, state and retrieval layer** | Context rot and memory decay |
| | Retrieval pollution (retrieval-augmented generation [RAG] noise) |
| | Infinite execution loops |
| | Resource and cost volatility |
| **Pillar 3: Runtime, tooling and multiagent network layer** | Silent execution failures |
| | Tool schema mismatch and parameter drift |
| | Cascading hallucinations in multiagent pipelines |
| | Role ambiguity and deadlock |
| | Premature task termination |
| **Pillar 4: Governance, security and observability layer** | Monitoring infrastructure |
| | Offline evaluations and production drift |
| | Prompt drift and unchecked mutations |
| | Lack of human-in-the-loop triggers |

## Pillar 1: Strategic and system design layer

### Unbounded business case

Organizations often select impressive, open-ended tasks for pilots rather than constrained workflows. These tasks require months of foundational refactoring when complex reasoning fails in live conditions. In many cases, in-scope is not defined appropriately.

### Mock APIs and real production environment

Demos run on clean, curated data snapshots and mock APIs, whereas production environments introduce messy legacy data, dynamic authentication, rate limits and partial API timeouts that break fragile agent loops.

### Stateless microservice bottlenecks

Standard microservices wipe memory state if a container restarts or an API call times out midway through a multistep task, causing agents to lose all progress without a durable execution framework.

## Pillar 2: Reasoning, state and retrieval layer

### Context rot and memory decay

Long-running workflows degrade as the context window fills with conversation noise, system logs and redundant tool responses, causing the agent to forget initial constraints or lose track of its primary objective.

### Retrieval pollution (RAG noise)

Injecting low-relevance or unscored vector search chunks into the prompt pollutes the model's reasoning window, causing incorrect conclusions even if the agent's logic is sound.

### Infinite execution loops

Lacking hard execution budgets or stop conditions, agents frequently get stuck in recursive validation loops by rechecking inputs, retrying failed endpoints or endlessly modifying previous steps.

### Resource and cost volatility

Infinite loops and deep-reasoning chains cause token counts and execution latency to spike exponentially, driving up API costs without delivering improved output.



 © 2026 NTT DATA, Inc.

## Pillar 3: Runtime, tooling and multiagent network layer

### Silent execution failures

Unlike standard APIs that produce clear error codes or stack traces, agents often fail quietly by hallucinating improvised workarounds when tools return unexpected data, malformed schemas or empty payloads.

### Tool schema mismatch and parameter drift

Models frequently invent nonexistent tool names or generate valid tool calls with malformed arguments, triggering silent degradation or unauthorized downstream mutations.

### Cascading hallucinations in multiagent pipelines

A slight hallucination or ambiguous step output from an upstream agent is accepted as fact-based truth by downstream agents, causing compounding reasoning errors across the pipeline.

### Role ambiguity and deadlocks

Agents with loosely defined roles often duplicate efforts, contradict each other, withhold crucial context or fall into circular dependencies where one agent waits indefinitely for another.

### Premature task termination

Multiagent networks frequently fail to determine when an overall task is genuinely complete, leading to incomplete outputs being erroneously flagged as finished or jobs running indefinitely.

## Pillar 4: Governance, security and observability layer

### Monitoring infrastructure

Traditional application monitoring tools measure CPU, uptime and request latency, which stay “green” even when an agent is hallucinating actions or executing bad business decisions.

### Offline evaluations and production drift

Agents tested solely against static offline evaluation datasets fail when presented with dynamic user inputs, updated base model behavior or drifting API schemas in real-world traffic.

### Prompt drift and unchecked mutations

Minor tweaks to system prompts or tool definitions shipped without proper version control, regression testing or rollback mechanisms cause unpredictable drops in production performance.

### Lack of human-in-the-loop triggers

Failing to build policy-based pause states for high-risk actions allows agents to act with unchecked autonomy, creating regulatory, security and compliance liabilities.



 © 2026 NTT DATA, Inc. 



# 3. Unified production assurance lifecycle, from readiness to deployment

To address these production challenges, this paper proposes three distinct frameworks. They are designed to operate as interconnected stages of a unified production assurance lifecycle rather than isolated ones.

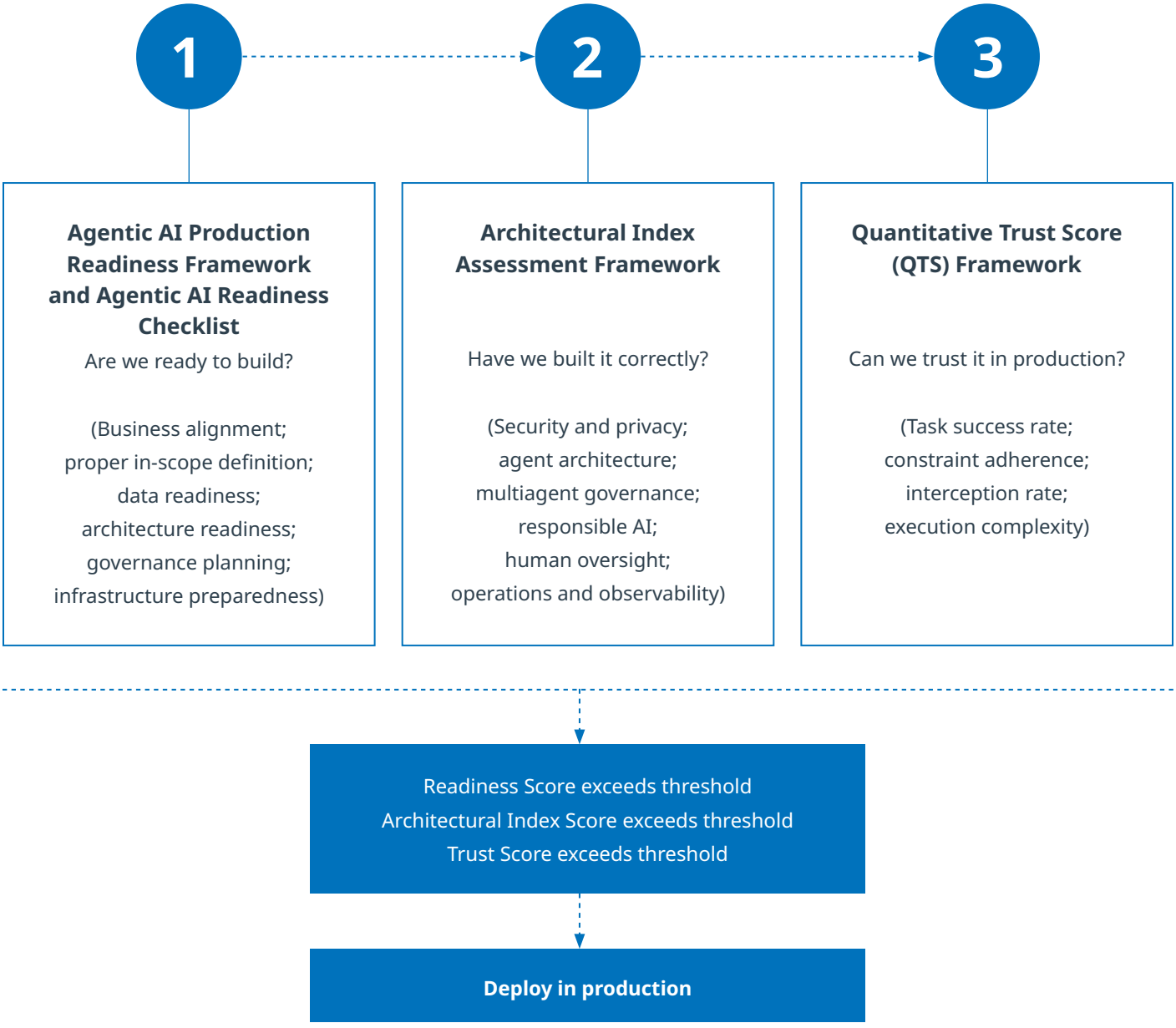
The journey begins with **readiness**, which evaluates whether the foundational capabilities required to build an enterprise-grade agentic AI solution are in place. These include business alignment, use-case selection, architectural preparedness, data readiness, governance planning, security considerations and operational capabilities. The Agentic AI Production Readiness Framework and Agentic AI Readiness Checklist provide a systematic approach to establishing these foundations before development begins.

Once development is completed, the focus shifts to **architectural compliance**. The Architectural Index measures whether the implemented solution conforms to enterprise architecture standards, governance requirements, security controls, regulatory obligations, responsible AI principles and operational best practices. The proposed Architectural Index Assessment Framework enables organizations to quantitatively evaluate adherence across multiple domains, producing an objective measure of deployment preparedness.

However, architectural compliance alone does not guarantee reliable runtime behavior. Agentic systems remain probabilistic in nature and may exhibit unpredictable behaviors despite meeting governance and architectural requirements. Therefore, a third layer of validation, **trust**, is required. The Quantitative Trust Score (QTS) Framework measures actual operational trustworthiness by evaluating business-outcome achievement, cognitive alignment, execution quality and operational efficiency. This provides a quantitative understanding of how the agent behaves under real-world conditions.

Together, these three stages create a progressive assurance model, as presented in Figure 2.

**Figure 2: Unified production assurance lifecycle, from readiness to deployment**





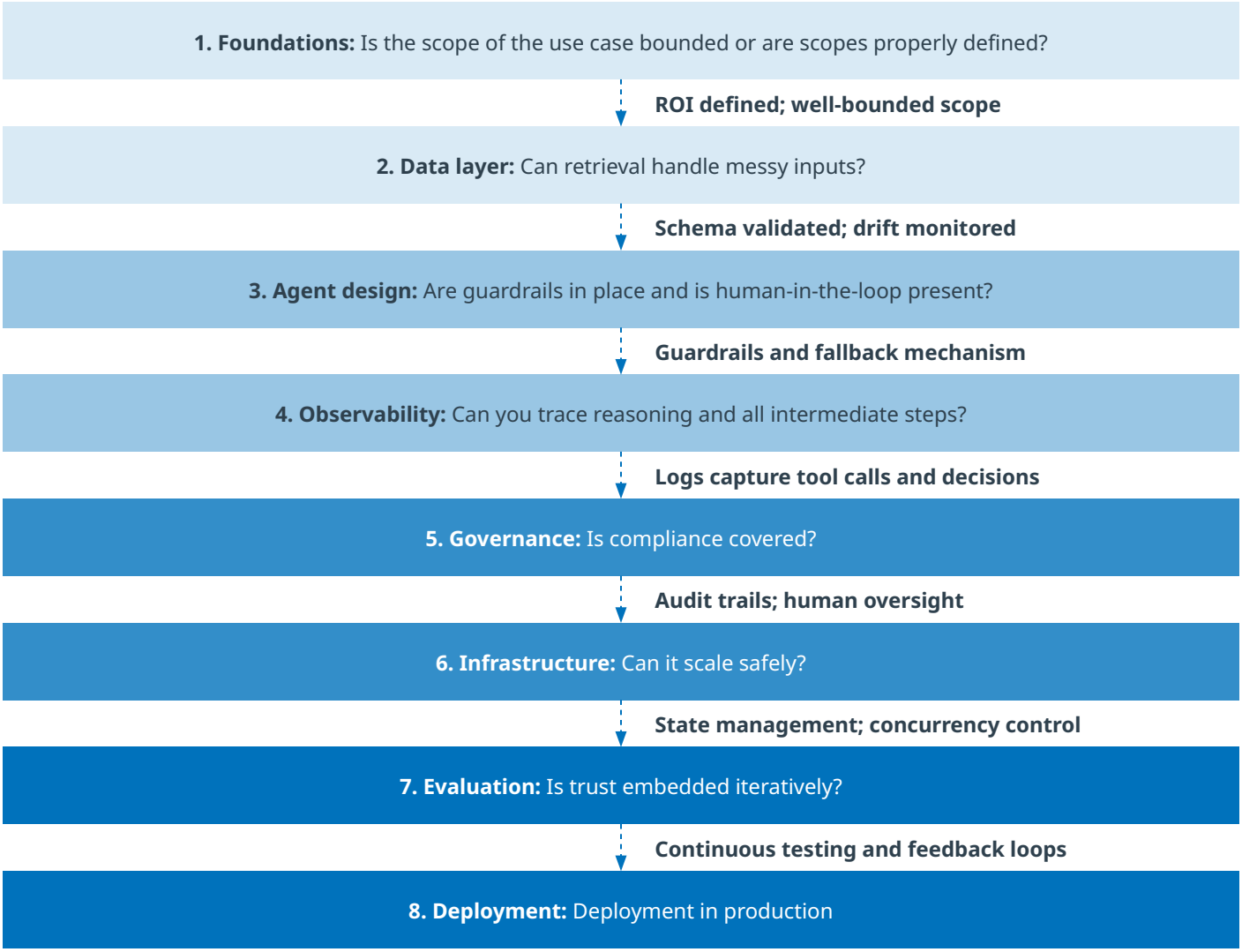

 © 2026 NTT DATA, Inc. 

# 4. Agentic AI Production Readiness Framework

This section presents a comprehensive set of readiness frameworks, detailed assessment checklists, architectural guidelines and industry best practices to help development teams design, build and deploy production-ready agentic AI systems. It provides a structured approach for assessing business readiness, data preparedness, security, governance, trust, observability and operational excellence across the agent lifecycle. By following these recommendations during the design stage, teams can ensure that agentic solutions are scalable, secure, compliant, reliable, and aligned with enterprise standards and responsible AI principles.[4,5,6]

The Agentic AI Production Readiness Framework, presented in Figure 3, acts as a structured maturity assessment for agentic AI solutions, ensuring that organizations address business value, data readiness, agent safety, governance, operational excellence and continuous trust evaluation before moving to production.[7,8,9,10]

**Figure 3: Agentic AI Production Readiness Framework**




 © 2026 NTT DATA, Inc. 

# 4.1. Agentic AI Readiness Checklist

The Agentic AI Readiness Checklist, presented in Table 1, provides a comprehensive set of best practices for designing, deploying and operating enterprise-grade agentic AI solutions at a scale. It covers the complete lifecycle, from business alignment and data readiness to agent design, governance, security, observability and continuous evaluation.

The objective is to help organizations establish a robust, trustworthy and production-ready foundation while minimizing operational, security and compliance risks. By following these guidelines, teams can accelerate the transition from proof of concept to production, improve system reliability and support responsible AI adoption.[11,12]

**Table 1: Agentic AI Readiness Checklist**

| Layer | Topic | Description |
|---|---|---|
| **Foundation layer** | Use-case clarity | Initiate projects with well-bounded, narrow scopes targeting clearly articulated business problems.<br>Establish quantitative ROI metrics, define explicit key performance indicators (KPIs) and determine acceptable error tolerance thresholds before building. |
| | Architecture discipline | Apply a modular, bottom-up architectural design for key components (building agents, memory design, orchestration, evaluation, deployment).<br>Avoid the "built backward" trap of committing to end-to-end goal flows without first solidifying the core underlying infrastructure. |
| | Pilot-to-production plan | Construct an explicit, phased roadmap detailing the transition from proof of concept (POC) demos to production-scale deployment.<br>Define criteria for infrastructure scaling, load management and production-readiness gates. |
| **Data and knowledge layer** | Data integrity checks | 1. Automated ingestion and normalization pipelines<br>a. Implement pipelines capable of parsing messy real-world formats (for example, PDFs, scraped web content, raw OCR logs, JSON fragments).<br>b. Standardize encoding, remove duplicates, handle missing values and scrub redundant noise before storage.<br>2. Strict schema validation and quality gates<br>a. Enforce preagent input validation using schema-enforcement frameworks (for example, Pydantic, JSON Schema, Great Expectations).<br>b. Filter out malformed payload structures, unexpected data types or out-of-range inputs to prevent downstream agent execution failures or security vulnerabilities (for example, prompt injection embedded in raw input data).<br>3. Automated outlier detection<br>Deploy real-time checks to flag corrupt text files, truncated payloads or unexpected character sets before they are vectorized or indexed. |
| | Retrieval evaluation | Continually test vector/graph retrieval performance under real-world production-query volumes.<br>Systematically monitor for embedding drift, vector index degradation and schema mismatches over time. |
| | Knowledge freshness | Apply CDC real-time triggers (synchronization mechanisms).<br>Tag context chucks with timestamps and version IDs to maintain verifiable document provenance (lineage and versioning). |

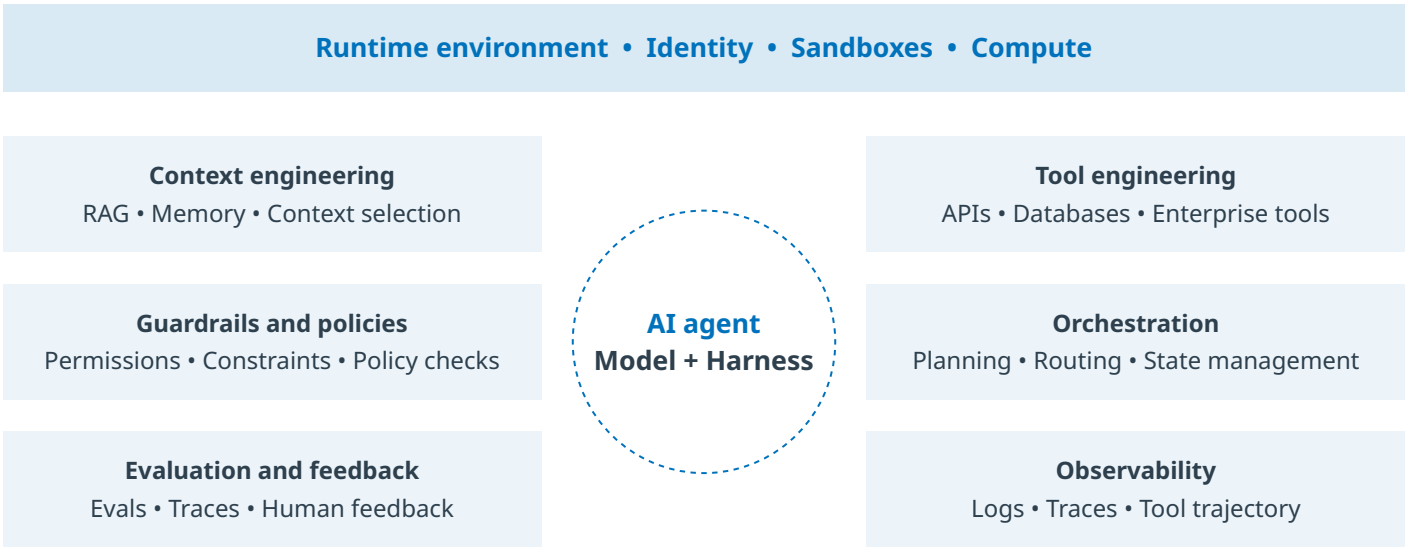
| Layer | Topic | Description |
|---|---|---|
| **Agent design** | Graduated autonomy | Deploy agent permissions in structured phases:<br>**Phase 1: Recommendation-only, with zero execution authority**<br>**Phase 2: Supervised execution requiring human sign-off**<br>**Phase 3: Limited, bounded autonomy under monitored conditions** |
| | Agent memory architecture | 1. Short-term and long-term memory separation<br>a. Short-term (in-session) memory<br>Manage active context windows dynamically using conversational buffer windows, sliding context windows or dynamic key-value state variables (for example, using frameworks like LangGraph, CrewAI or temporal state engines).<br>b. Long-term (episodic/semantic) memory<br>Store past user preferences, execution histories and learned behaviors across sessions in external persistent stores (for example, vector databases, relational/key-value stores, knowledge graphs).<br>2. State persistence and checkpointing<br>Implement externalized state management (for example, Redis/Postgres) with step-level snapshotting for fault-tolerant crash recovery and replay.<br>3. Context pruning and summarization<br>Enforce dynamic token-aware context trimming and hierarchical trajectory summarization to prevent context window bloat. |
| | Guardrails | 1. Structured output validation<br>Validate rigid Pydantic/JSON Schema prior to tool execution.<br>2. Infinite-loop detection<br>Analyze agent state history and tool-call sequences to detect repetitive cycles.<br>3. Token consumption and cost allocation caps<br>Enforce strict upper boundaries on total input/output token usage per interaction chain (for example, hard cap per user workflow).<br>4. Apply generic and domain-specific guardrails. |
| | Fallbacks | Configure fail-safe default actions and immediate human-in-the-loop (HITL) escalation protocols whenever an agent encounters low confidence scores, model failures or execution exceptions. |
| **Observability and monitoring** | Reasoning traces | Maintain end-to-end observability by capturing full execution traces — including raw tool calls, full context window payloads, intermediate reasoning steps and final decision paths. |
| | Metrics and dashboards | Monitor KPIs in real time via visual dashboards, tracking latency per turn, accuracy scores, token cost allocations and error-compounding rates across multistep agent chains. |
| | Failure stack tracking | Systematically isolate and log breakdown root causes across individual stack layers, explicitly pinpointing errors in state memory, task planning decomposition, retrieval steps or tool permissions. |

| Layer | Topic | Description |
|---|---|---|
| **Governance and compliance** | Policy frameworks | Embed comprehensive audit trails, step-by-step explainability logs and direct policy alignment mechanisms to comply with industry regulatory standards (such as the GDPR, HIPAA, EU AI Act and SOC 2). |
| | Human oversight | Design clear operational escalation pathways and real-time manual override mechanisms, allowing operators to intercept, modify or cancel agent execution at any point. |
| | Ethical alignment | Establish proactive bias checks, fairness constraints and alignment checks to ensure agent decisions adhere to ethical guidelines and maintain nondiscriminatory outputs. |
| **Infrastructure and scalability** | Distributed systems patterns | Architect resilient distributed agent infrastructure, incorporating robust state management, retry logic with exponential backoff, API rate limiting and concurrency/lock control for shared resources. |
| | Identity and security | Enforce strict hyphenate zero trust security using scoped API credentials, robust prompt injection defenses (direct and indirect) and sandboxed runtime environments to contain potential security breach blast radiuses. |
| | Cost management | Control token usage and operational overhead by implementing response caching, persistent shared-state memory, model routing optimizations and hard budget caps per session/tenant. |
| | Scalability | To make an agentic system scalable, separate the agent workers from the database to easily add more servers when traffic spikes. Use message queues to organize incoming tasks into smooth waiting lines, and cache common responses to keep the system fast and cost-effective. |
| **Evaluation and iteration** | Continuous testing | Validate system stability in production via shadow deployments (parallel execution alongside legacy systems), side-by-side A/B testing and automated chaos testing experiments to identify vulnerabilities. |
| | Iterative trust injection | Embed reliability validation, safety checks, output transparency and full trace auditability directly into the core iterative development lifecycle rather than patching after launch. |
| | Feedback loops | Establish closed-loop operational workflows that continuously collect human and system feedback to drive dataset refinement, model fine-tuning, prompt optimization and seamless redeployment. |

In addition to the above checklist, to increase the number of production-deployable agents, a novel design approach has recently emerged: "harness engineering." Here, the relationship with the agent is best expressed by a simple architectural equation: Agent = {Model} + {Harness}.[13]

In any agentic application, the model functions as a stateless token predictor, providing raw cognitive reasoning, while the harness comprises the runtime software infrastructure that coordinates tool dispatch, context management and safety enforcement. Hence, while the model provides probabilistic reasoning, the harness converts probabilistic model outputs into controlled, policy-constrained, observable execution by managing context, tool execution, safety controls and session memory. The anatomy of an agent harness is presented in Figure 4.

**Figure 4: Anatomy of an agent harness**




Ultimately, it functions as a parameterized component capable of effortlessly pinpointing performance gaps so they can be readily addressed. As illustrated in Figure 4, the agent harness manages the execution loop, tool and function calling, context management, state and memory, control flow, safety and observability. Therefore, employing a harness is consistently recommended as a best practice for any agentic application.



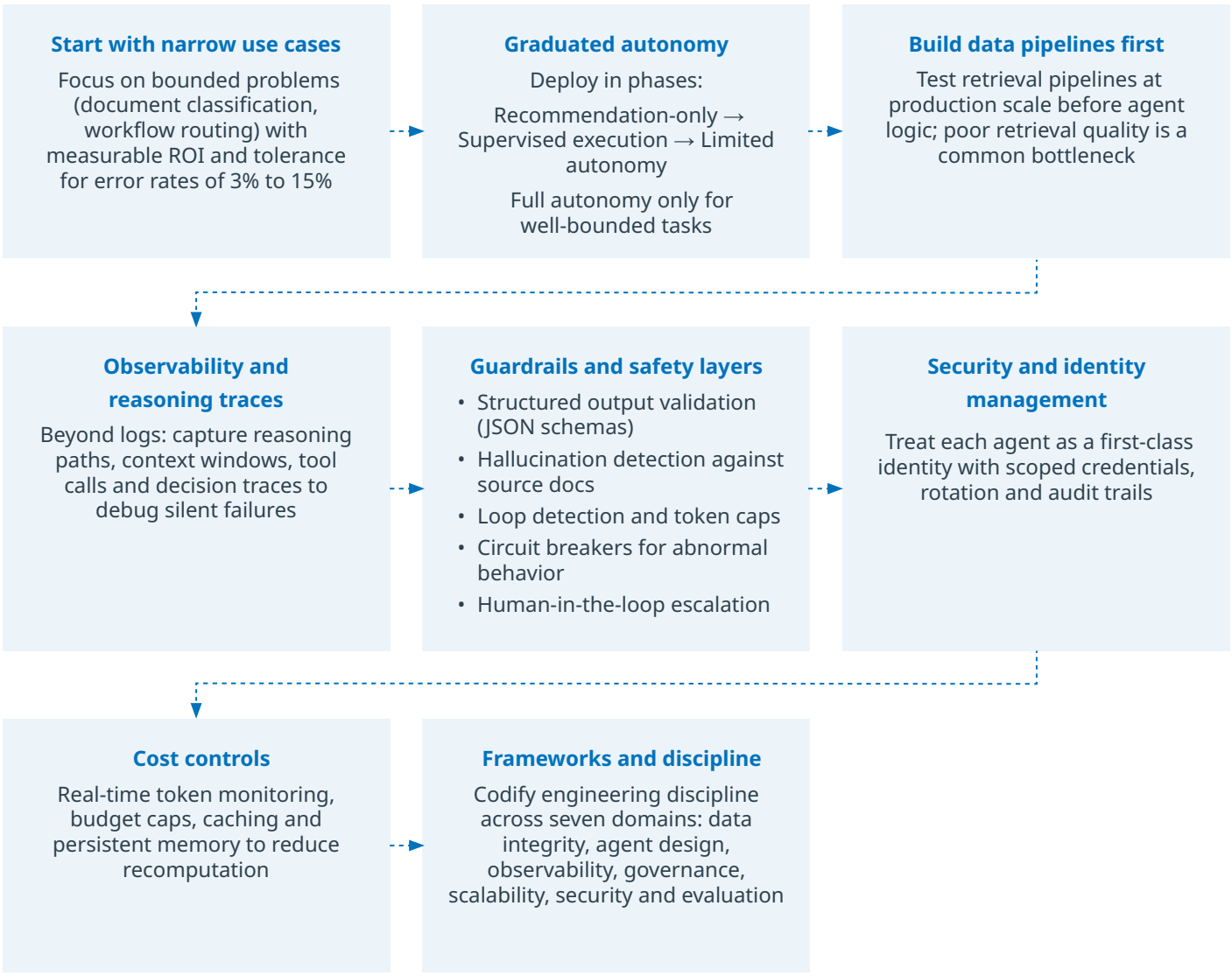
 © 2026 NTT DATA, Inc. 

# 4.2. Moving agents to production

To build a production-ready agentic AI system, teams should follow a structured and iterative approach rather than focusing solely on model development. Start by validating the business use case, defining success metrics and assessing organizational readiness. Next, establish a robust data foundation with proper governance, security and privacy controls. Design agents with appropriate guardrails, human oversight, memory and orchestration capabilities to ensure safe and reliable behavior. Implement observability, monitoring, auditability and compliance controls to maintain trust and operational excellence. Finally, continuously evaluate, test and refine the system before deploying to production and establishing ongoing feedback loops for continuous improvement.

These steps are set out in Figure 5.

**Figure 5: Step-by-step journey for transitioning agentic AI solutions from development to production**

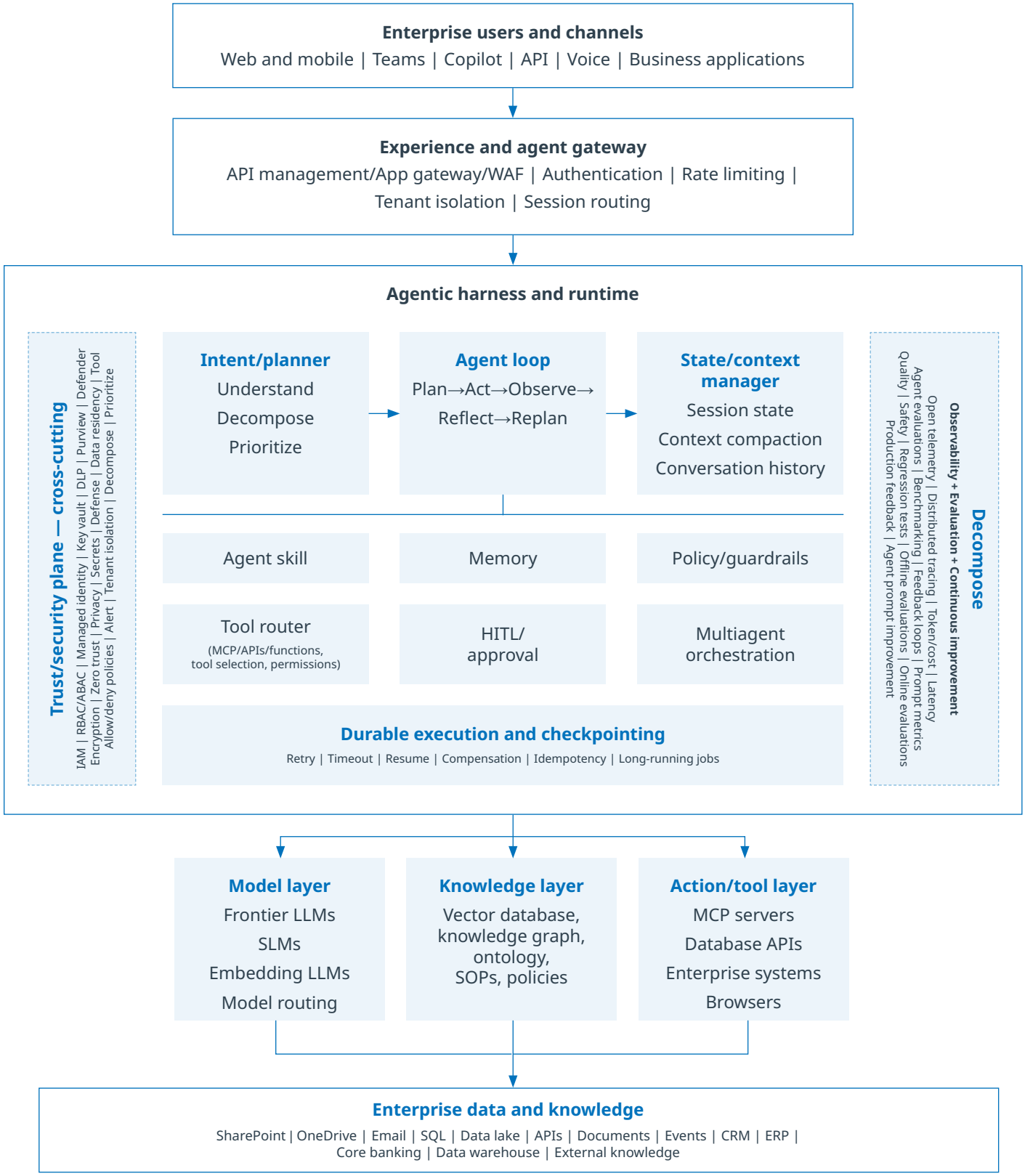



 © 2026 NTT DATA, Inc. 

# 4.3. Proposed architectural diagram for agentic application

One sample architecture comprising all the proposed modules is presented in Figure 6.

**Figure 6: Proposed architectural diagram for agentic application**




 © 2026 NTT DATA, Inc. 

# 5. Agentic AI Architectural Index Assessment Framework

Having detailed the foundational principles for validating agentic AI systems in the preceding section, we now propose a structured framework designed to quantify and measure the architectural compliance of the agentic AI systems.

## 5.1. Motivation for the assessment

As agentic AI systems evolve from assistive tools to autonomous decision-making systems, organizations must ensure that agents operate within defined business, security, regulatory and ethical boundaries. Traditional software assessments focus on static controls; however, agentic AI introduces dynamic behaviors such as autonomous planning, tool execution, memory persistence, multiagent collaboration and adaptive reasoning. Therefore, adherence in agentic AI must be continuously measured across architectural, operational, governance, security and regulatory dimensions to ensure the system remains trustworthy, auditable and aligned with enterprise policies throughout its lifecycle.

We therefore propose a framework that evaluates architectural alignment across 10 key domains that collectively represent enterprise-grade agentic AI readiness and governance. These domains are derived from the Well-Architected Assessment Framework and cover both technical and organizational controls.[14] The domains and their objectives are summarized in Table 2.

**Table 2: Architectural Index assessment domains**

| Domian/layer | Objective |
|---|---|
| 1. Business governance | Alignment to business objectives, ROI, ownership and operating model |
| 2. Agent architecture | Agent reasoning, planning, goals, constraints and guardrails |
| 3. Multiagent and orchestration | Agent coordination, delegation, workflow governance and state management |
| 4. Data and knowledge governance | Data quality, lineage, freshness, access control and retrieval governance |
| 5. Security and privacy | Identity, access control, encryption, secrets management and zero trust |
| 6. Responsible AI and safety | Bias mitigation, explainability, transparency, prompt injection and hallucination protection |
| 7. Human oversight | Human-in-the-loop approvals, escalation mechanisms and override controls |
| 8. Operations and observability | Monitoring, tracing, logging, incident management and AgentOps practices |
| 9. Reliability and resilience | Availability, disaster recovery, failover and fault tolerance |
| 10. Regulatory and compliance | Audit trails, retention policies, privacy regulations and AI governance mandates |

 © 2026 NTT DATA, Inc. 

Each domain and layer comprises a set of control attributes evaluated by respective subject matter experts (SMEs), such as architects, project managers and lead AI engineers (refer to the sample presented in Table 3). Aggregating these evaluations across all domains yields a single composite score.

**Table 3: Sample assessment attributes**

| Domain/layer | Component | Owner |
|---|---|---|
| Agent architecture | Agent design, constraints, goals, persona, planner, reasoner, reviewer, router | Lead AI Architect |
| AgentOps and GenAIOps | CI/CD, evaluation, experimentation, monitoring, release | GenAIOps Engineer |
| AI safety | Content safety, grounding, hallucination, jailbreak, prompt injection | AI Security Engineer |

## 5.2. Compiling the Architectural Index Score

This section details how to derive the composite score from alignment ratings (Compliant, Partially Compliant, Noncompliant and Under Review) assigned to each control attribute, across domains and layers, as detailed in Table 4.

**Table 4: Architectural compliance rating scale**

| Rating | Score | Description |
|---|---|---|
| Compliant | 1.0 | Fully implemented and operational |
| Partially Compliant | 0.5 | Implemented but requires improvement |
| Noncompliant | 0.0 | Control not implemented |
| Under Review | 0.0 | Pending assessment |

 © 2026 NTT DATA, Inc. 

**The architectural index for each domain is calculated as:**

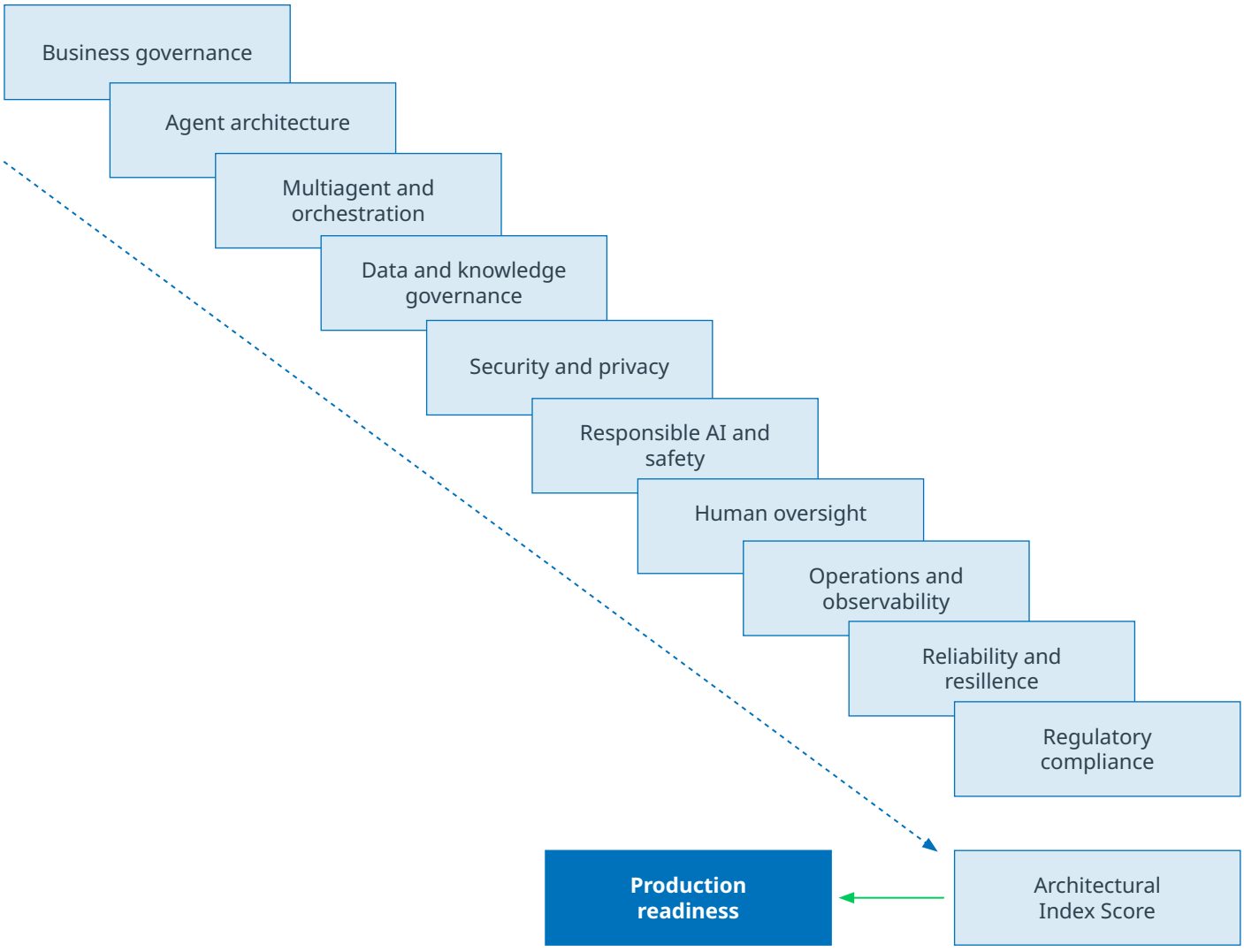
$$\text{Domain Index} = \left[\frac{(\text{Compliant} + (0.5 \times \text{Partially Compliant}))}{(\text{Total Controls})}\right] \times 100$$

**The overall Agentic AI Architectural Index is calculated as:**

$$\text{Architectural Index \%} = \frac{\sum \text{Domain Scores}}{(\text{Number of Domains}) \times 100} \times 100$$

The functional flow of the assessment framework follows the sequence presented in Figure 7.

**Figure 7: Steps to compiling the Architectural Index Score**



# 5.3. Sample implementation

Table 5 outlines the user selections for the **agent architecture** layer. Using these selections, the system automatically evaluates the domain and calculates a specific alignment score.

**Table 5: Deriving an Architectural Index Score based on sample inputs on agent architecture**

| Component | Checklist validation question | Assessment of criteria and verification evidence | Risk or criticality | Compliance status | Assigned owner | Remediation and architectural guidance |
|---|---|---|---|---|---|---|
| **Agent design** | Is the agent architecture pattern (for example, ReAct, Plan-and-Solve, Reflection, Hierarchical) explicitly documented, modular and matched to task complexity? | Architecture design document (ADD), architectural decision record (ADR) justifying agent loop topology and state management | Critical | Compliant | AI Solution Architect | Selected the simplest viable pattern; avoided multistep iterative loops where single-turn structured generation suffices. |
| **Planner** | Does the planning mechanism support step decomposition, dynamic plan revision, loop prevention and maximum depth limits? | Plan decomposition logic, recursion limit safeguards (for example, maximum 10 steps), cycle detection algorithms and plan replay logs | Critical | Partially Compliant | Lead AI Engineer | While planning decomposition is implemented, loop detection and recursion controls are missing. Add a 10-step planning depth limit, repeated-action detection, cycle prevention and fallback paths for planning loops. |
| **Reasoner** | Is the reasoning engine configured with structured reasoning strategies (for example, chain or thought, tree of thought) and validation checks on intermediate steps? | Prompt engineering artifacts, intermediate thought validation schemas and reasoning step verification tests | High | Non-compliant | AI Engineer | Intermediate reasoning is not validated before generating conclusions. To fix this, implement rule-based checks, confidence scoring or secondary evaluators. |

 © 2026 NTT DATA, Inc. 

| Component | Checklist validation question | Assessment of criteria and verification evidence | Risk or criticality | Compliance status | Assigned owner | Remediation and architectural guidance |
|---|---|---|---|---|---|---|
| **Router** | Is there an intelligent intent classification and routing mechanism to direct requests to the optimal agent persona, workflow or specialized tool? | Intent classification model/router test suite, routing latency benchmarks, fallback routing handlers for ambiguous intents | High | Partially Compliant | AI Engineer | Routing capability exists, but routing accuracy, confidence thresholds and fallback handling are not demonstrated. Implement intent-confidence scoring, routing evaluation benchmarks and fallback workflows that redirect low-confidence requests to a general-purpose agent, alternate workflow or clarification process. |
| **Persona** | Are agent system prompts, persona boundaries, tone of voice, domain constraints and behavioral guardrails strictly defined and version-controlled? | System prompt repository, version-controlled prompt registries, prompt diffs, behavioral compliance test results | High | Compliant | Prompt Engineer/AI Lead | Stored system prompts in Git/ Prompt Registry; prevented dynamic runtime prompt overriding without strict authorization. |
| **Goals** | Are terminal goal criteria and step-level completion definitions deterministic, measurable and unambiguous to prevent open-ended execution? | Goal-evaluation criteria, stopping-condition code assertions, success validation functions | Critical | Partially Compliant | Lead AI Engineer | Task completion currently relies on agent judgment rather than objective verification. Define deterministic success criteria, implement measurable completion checks and enforce programmatic stopping conditions to avoid premature termination or endless execution. |

| Component | Checklist validation question | Assessment of criteria and verification evidence | Risk or criticality | Compliance status | Assigned owner | Remediation and architectural guidance |
|---|---|---|---|---|---|---|
| **Constraints** | Are explicit operational, temporal, computational and ethical constraints enforced within the agent's context and execution framework? | Constraint enforcement layer, context-window token budgets, execution timeouts, domain-boundary validation rules | High | Under Review | AI Solution Architect | Enforced constraints via runtime middleware wrappers outside the model's stochastic reasoning space. |
| **Reviewer** | Does the agent incorporate a self-reflection, critique or secondary review agent to evaluate intermediate outputs before final action execution? | Critique prompt templates, secondary evaluator agent logs, output-score thresholds required for execution | Medium | Under Review | AI Engineer | Used lightweight judge agents or rule-based validators to review high-risk schema generation prior to tool calls. |

Based on the formula in section 5.2, the Architectural Index Score for the sample inputs in Table 5 are presented in Table 6. The score signifies the level of architectural compliance for the agent architecture domain.

**Table 6: Architectural Index Score on agent architecture**

| Domain/ assessment area | Total items | Compliant | Partial | Noncompliant | Under Review | Architectural Index Score |
|---|---|---|---|---|---|---|
| **Agent architecture** | 8 | 2 | 3 | 1 | 2 | 43.8% |

 © 2026 NTT DATA, Inc. 

# 6. Quantitative Trust Score Framework

Artificial intelligence capabilities are advancing at an unprecedented pace. Models are becoming more powerful, enterprise data ecosystems are expanding and computational infrastructure continues to scale. The primary challenge for organizations now is not technological capability but the ability to establish and maintain trust in AI-driven decisions and actions.

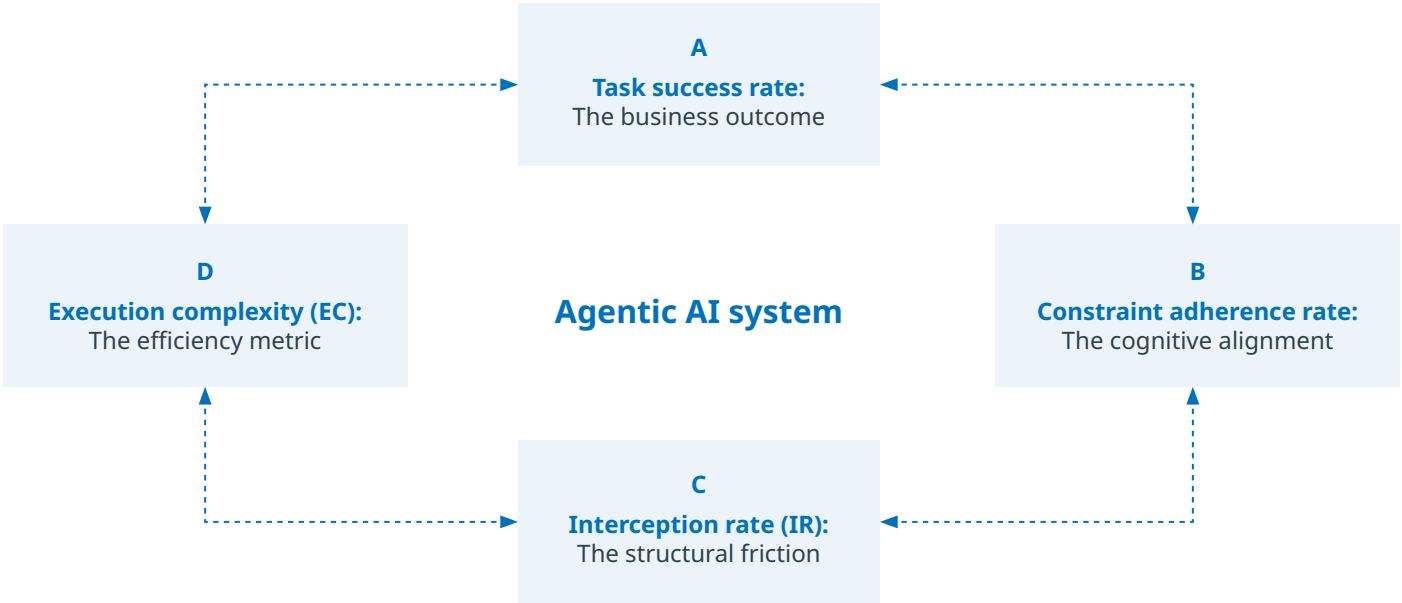
Unlike traditional software systems that operate deterministically, agentic AI systems are inherently probabilistic. Their outputs can vary based on context, data, prompts, reasoning paths and external interactions. As agents are entrusted with increasingly complex tasks and autonomous decision-making responsibilities, organizations must implement mechanisms that can measure the trustworthiness of agentic system.

We therefore propose a novel architecture, illustrated in Figure 8, that provides a Quantitative Trust Score for agentic AI systems.

## 6.1. The four dimensions of the score

The Quantitative Trust Score is derived from four distinct dimensions, as presented in Figure 8. A successful, production-grade agentic implementation is assessed and quantified across these four key operational dimensions, enabling a comprehensive evaluation of its trustworthiness.

**Figure 8: Four dimensions of the Quantitative Trust Score for an agentic AI system**



## A. Task success rate: The business outcome

### What it measures

Did the agent successfully achieve the user's ultimate intent, even if the user pivoted, looped or backtracked midway through the conversation?

### The math

A binary metric per instance (1 for absolute resolution, 0 for failure or customer abandonment).

### The objective

Maximize this number. It tells you if your agent is actually driving real-world business value.

## B. Constraint adherence rate: The cognitive alignment

### What it measures

Whether the agent's observable actions, tool calls, outputs and execution states comply with defined business, operational, security and policy constraints.

### The math

Ca = (Compliant evaluated actions or steps / Total evaluated actions or steps)

### The blueprint

Constraint adherence tracks whether the agent is aligned with your business logic. If the rule says, "Never offer a discount to a customer who has been with us for less than three months," an adhering agent reads a one-month tenure profile and reasons: "Customer is ineligible for a discount due to tenure. I must offer a free feature-tier downgrade instead."

## C. Interception rate (IR): The structural friction

### What it measures

How often the agent fails to properly format a tool payload or attempts to pass arguments that violate your software types and schemas, regardless of its underlying intent.

### The math

IR = (Tool-call attempts blocked by symbolic layer / Total tool activation attempts)

### The blueprint

Interception rate tracks whether the agent is compatible with your software interfaces. If the agent correctly reasons, "I will apply a 30% discount," but clumsily executes apply_discount(discount="thirty percent") instead of an integer 30, your deterministic Pydantic schema immediately intercepts and blocks it.

## D. Execution complexity (EC): The efficiency metric

### What it measures

Execution complexity measures the conversational and computing efficiency of the interaction. However, we must ensure our math doesn't punish an agent for doing its job. If a customer is looking to buy Product A, but decides to explore other product lines before backtracking and finally purchasing Product A that 30-turn deviation is a customer conversion victory, not an agent failure.

To prevent punishing an agent for natural human curiosity, we break down the actual conversational turns ($T_{\text{actual}}$) into independent vectors:

- **Target benchmark turns (T_optimal):** The baseline, optimal number of turns required to complete the task under clean conditions.
- **User-driven drift (D_u):** The number of turns where the user introduces new intents, compares products or asks informational questions. This is a positive indicator of conversational resilience.
- **Agent-driven looping (L_a):** The number of turns where the agent is stuck repeating identical prompts, failing internal tool validations or retrying the same failed API calls back-to-back.

 © 2026 NTT DATA, Inc. 

## The refined math

Instead of a raw turn counter, our metric completely isolates and penalizes only systemic agent looping (L_a) relative to the target benchmark turns (T_optimal), completely removing user-driven exploration (D_u) from the penalty box:

$$EC = 1 + (L_a/T_{optimal})$$

---

## The math in action

Let's look at how this mathematical model handles two drastically different 30-turn scenarios, assuming a target baseline optimization (T_optimal) of **10 turns:**

### Scenario A (the deep-dive customer):

The customer wants Product A but spends 20 turns exploring alternative choices, asking deep technical questions and comparing catalogs (D_u = 20). The agent responds flawlessly without a single internal software stutter or loop (L_a = 0).

**EC = 1 + (0/10) = 1.0 (perfect efficiency score)**

The framework completely filters out the human curiosity. The agent receives zero penalty for keeping a chatty customer engaged.

### Scenario B (the broken record):

The customer attempts to check out, but the agent encounters a software schema mismatch and spends six turns stuck in a repetitive internal error correction cycle with your database APIs (L_a = 6).

**EC = 1 + (6/10) = 1.6 (penalized for systematic inefficiency)**

The framework immediately exposes backend software friction that is actively burning computing tokens and wasting user time.

### The formula: Quantitative Trust score (QTS):

To evaluate a single execution instance or a baseline of production logs, these variables are synthesized into a composite trust index:

$$QTS = w_1 (SR) + w_2 (C_a) + w_3 (1 - IR) - w_4 (\max (0, EC - 1))$$

### Where:

- SR (task success rate) represents the proportion of executions that successfully achieve the intended business outcome.
- C_a (constraint adherence rate) represents the proportion of evaluated execution steps that comply with defined business and operational constraints.
- IR (interception rate) represents the proportion of tool-call attempts blocked by the deterministic validation layer due to invalid schemas or parameters.
- EC (execution complexity) represents execution inefficiency relative to the optimal baseline, where (EC >= 1).
- w1, w2, w3 are configurable weights reflecting business priorities, with (w1 + w2 + w3 = 1).
- w4 is a penalty multiplier applied only when execution complexity exceeds the baseline ((EC > 1)).

The (\max(0,EC-1)) function ensures that execution below or at the optimal baseline does not incur a penalty, while systemic agent looping or inefficiency above the baseline reduces the overall QTS.

**QTS provides a quantitative measure of operational trustworthiness based on task outcome, policy adherence, execution correctness and execution efficiency.**

## 6.2. Diagnostic blueprint: From metric to action

This framework provides an engineering diagnostic matrix. By separating human drift from machine errors, we can give precise optimization recommendations, as illustrated by the scenarios presented in Table 7.

**Table 7: Examples of precise optimization recommendations**

| Metrics observed | Diagnosis | Strategic engineering action |
|---|---|---|
| **High $C_a$ (98%)**<br>**High IR (40%)** | **The agent is smart but clumsy**<br>It understands your business rules perfectly, but it keeps messing up the coding syntax of your tool payloads. | Do not touch the system prompt. Instead, rewrite your Pydantic schemas, clear up the parameter descriptions in your code or provide explicit JSON few-shot examples. |
| **Low $C_a$ (40%)**<br>**Low IR (5%)** | **The agent is well-behaved but rogue**<br>It writes incredibly clean, valid code that your backend APIs accept effortlessly, but it is completely ignoring company policies. | Your tools are fine, but your cognitive routing is broken. Tweak the system prompt instructions, inject stricter context guardrails or split the agent into an explicit neurosymbolic graph topology. |
| **High overall turns**<br>**High user drift ($D_u$), $L_a = 0$** | **The consultative salesperson**<br>The agent is doing exactly what a great human salesperson does: answering questions, handling objections and navigating human indecision. | Celebrate! This proves high conversational trust and context retention. The EC score remains a perfect 1.0. No code remediation is needed. |
| **High overall turns**<br>**High agent looping ($L_a$)** | **The broken record**<br>The agent is stuck in an internal error-correction loop with the symbolic layer, burning tokens and user patience. | The symbolic error message being fed back to the LLM is confusing it. Rewrite your backend validation exceptions so they explicitly instruct the LLM how to correct the payload. |

# 7. Conclusion

The three frameworks presented in this paper are designed to operate as complementary stages of a unified production assurance lifecycle. The Agentic AI Production Readiness Framework ensures that the organization, architecture, data and operational foundations are adequately established before development begins. The Agentic AI Architectural Index Assessment validates that the implemented solution adheres to enterprise governance, security, regulatory and architectural requirements. Finally, the Quantitative Trust Score evaluates the actual runtime behavior of the agentic system by measuring business outcome achievement, cognitive alignment, execution quality and operational efficiency.

Together, these frameworks create a structured progression from readiness and architectural compliance to trust and production deployment, enabling organizations to systematically reduce risk while increasing confidence in autonomous systems. Production deployment should be treated as the culmination of all three assessments rather than as a standalone technical milestone.

# 8. List of abbreviations

| Abbreviation | Meaning |
|---|---|
| **ADD** | architecture design document |
| **ADR** | architectural decision record |
| **AI** | artificial intelligence |
| **API** | application programming interface |
| **CDC** | change data capture |
| **CI/CD** | continuous integration and continuous delivery |
| **CPU** | central processing unit |
| **DB** | database |
| **EC** | execution complexity |
| **GDPR** | General Data Protection Regulation |
| **HIPAA** | Health Insurance Portability and Accountability Act |
| **HITL** | human in the loop |
| **IR** | interception rate |
| **JSON** | JavaScript Object Notation |
| **KPI** | key performance indicator |
| **LLM** | large language model |
| **OCR** | optical character recognition |
| **PDF** | Portable Document Format |
| **POC** | proof of concept |
| **QTS** | Quantitative Trust Score |
| **RAG** | retrieval-augmented generation |
| **ReAct** | reasoning and acting |
| **ROI** | return on investment |
| **SME** | subject matter expert |
| **SOC 2** | System and Organization Controls 2 |

# 9. References


1. AnAr Solutions, “Why 88% of Agentic AI Pilots Never Reach Production (And How to Be the 12%),” AnAr Solutions, Apr. 16, 2026. [Online]. Available: https://anarsolutions.com/why-agentic-ai-pilots-fail-production/. [Accessed: Oct. 7, 2026].
2. A. Sandhaliya, “Why AI Agents Fail in Production: The Agent Failure Stack Explained,” Sherlocks AI, Jun. 5, 2026, updated Sep. 17, 2026. [Online]. Available: https://www.sherlocks.ai/blog/why-ai-agents-fail-in-production. [Accessed: Oct. 7, 2026].
3. B. Nweke, “Most AI Agents Fail in Production Because They’re Built Backwards,” Towards Data Science, May 27, 2026. [Online]. Available: https://towardsdatascience.com/most-ai-agents-fail-in-production-because-theyre-built-backwards/. [Accessed: Oct. 7, 2026].
4. J. Khan, “Why Agentic AI Systems Fail and What to Do About It (Part 1),” LinkedIn, n.d. [Online]. Available: https://www.linkedin.com/pulse/why-agentic-ai-systems-fail-what-do-jamal-khan-t1qac/. [Accessed: Oct. 7, 2026].
5. M. Hannecke, “Why AI Agents Fail in Production: What I’ve Learned the Hard Way,” Medium, n.d. [Online]. Available: https://medium.com/@michael.hannecke/why-ai-agents-fail-in-production-what-ive-learned-the-hard-way-05f5df98cbe5. [Accessed: Oct. 7, 2026].
6. A. Kargwal, “Why AI Agents Break: A Field Analysis of Production Failures,” Arize AI, Jan. 2026. [Online]. Available: https://arize.com/blog/common-ai-agent-failures/. [Accessed: Oct. 7, 2026].
7. Q. Packard, “What Breaks When Agentic AI Reaches Production?,” Cockroach Labs, Jun. 4, 2026. [Online]. Available: https://www.cockroachlabs.com/blog/agentic-ai-production-infrastructure/. [Accessed: Oct. 7, 2026].
8. K. Shah, “Top 6 Reasons Why AI Agents Fail in Production and How to Fix Them,” Maxim AI, May 5, 2026. [Online]. Available: https://www.getmaxim.ai/articles/top-6-reasons-why-ai-agents-fail-in-production-and-how-to-fix-them/. [Accessed: Oct. 7, 2026].
9. Sage IT, “Why AI Agents Look Strong in Demos but Break in Production,” Sage IT, n.d. [Online]. Available: https://sageitinc.com/blog/why-most-ai-agents-fail-in-production. [Accessed: Oct. 7, 2026].
10. M. Cemri et al., “Why Do Multi-Agent LLM Systems Fail?,” arXiv preprint arXiv:2503.13657, Mar. 17, 2025, rev. Oct. 26, 2025. [Online]. Available: https://arxiv.org/pdf/2503.13657. [Accessed: Oct. 7, 2026].
11. IJOER Engineering Journal, “From Proof of Concept to Production: Why Agentic AI Workflows Fail at Scale,” IJOER Engineering Journal Blog, May 5, 2026. [Online]. Available: https://ijoer.com/blog/from-proof-of-concept-to-production-why-agentic-ai-workflows-fail-at-scale. [Accessed: Oct. 7, 2026].
12. Abhitesh, “The Hidden Reasons AI Agents Fail in Production (And How to Build Better Ones),” Medium, n.d. [Online]. Available: https://medium.com/@abhitesh9319/the-hidden-reasons-ai-agents-fail-in-production-and-how-to-build-better-ones-f39fc3edb464. [Accessed: Oct. 7, 2026].
13. V. Trivedy, “The Anatomy of an Agent Harness,” LangChain, Mar. 10, 2026. [Online]. Available: https://www.langchain.com/blog/the-anatomy-of-an-agent-harness. [Accessed: Oct. 8, 2026].
14. Microsoft, “Azure Well-Architected Framework,” Microsoft Learn, n.d. [Online]. Available: https://learn.microsoft.com/en-us/azure/well-architected/. [Accessed: Oct. 8, 2026].



**Visit nttdata.com to learn more.**

NTT DATA is a $30+ billion business and technology services leader in AI and digital infrastructure. We accelerate client success and positively impact society through responsible innovation. As a Global Top Employer, we have experts in more than 70 countries. NTT DATA is part of NTT Group.

NTT DATA


NTT DATA