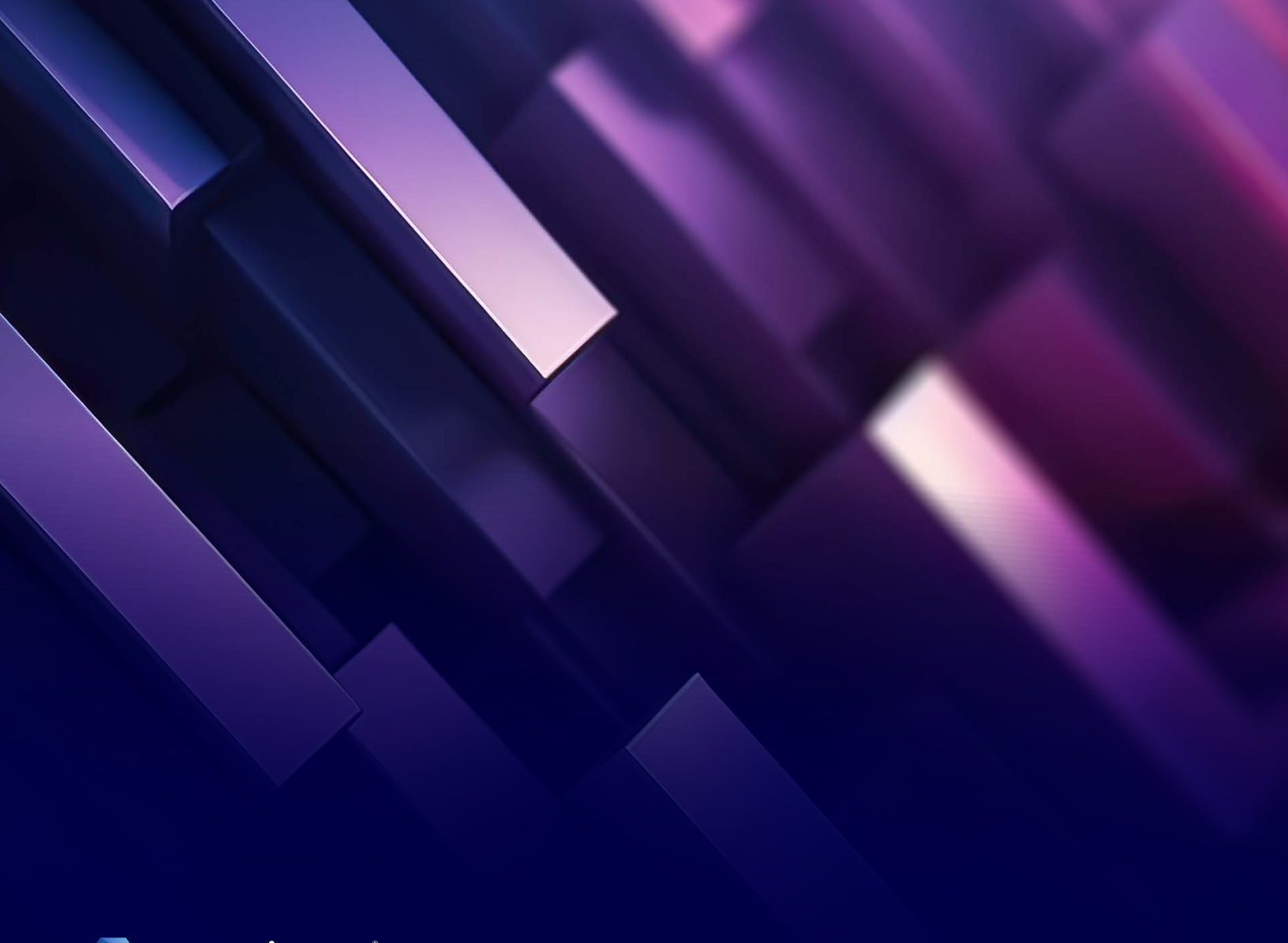


cognizant®

From architecture to intelligence: Building the agentic technology stack

Contents

Executive summary

Enterprises today face a widening AI velocity gap—the growing mismatch between the pace at which AI capabilities are evolving and the ability of core operational architectures to absorb and operationalize them. While experimentation with large language models and automation has accelerated, many organizations remain constrained by monolithic systems, fragmented data foundations, and operating models not designed for autonomous intelligence.

The first part, "Confronting the AI velocity gap: A new architecture for enterprise operations", of the three-part white paper series examined how this gap can be addressed by rearchitecting operations around an AI-led, human-enabled operating model—which is driven by straight-through processing (STP) and supported by progressive autonomy, modular agent design and strong governance.

This second part, "From architecture to intelligence: Building the agentic technology stack", of the white paper series focuses on the next architectural question: how enterprises should design operations as AI systems increasingly act, not just assist, within enterprise workflows. It introduces pragmatic operations architecture centered on agentic systems, outlining how intelligence can be structured, coordinated and governed at scale.

Through concepts such as vertical and horizontal agents, agent networks, ontology-driven grounding and context engineering, this second part of the white paper series presents a blueprint for embedding intelligence as a distinct architectural layer. Rather than advocating wholesale system replacement, it shows how organizations can innovate at AI speed while maintaining operational integrity and trust. The paper concludes with a practical playbook, guiding enterprises from readiness assessment to the deployment of reliable and scalable agents.

Architecting for agentic operations

The central premise of this white paper is that closing the AI velocity gap requires more than adopting advanced models. It requires rethinking how intelligence is embedded into enterprise operations. As AI systems transition from recommendation engines to autonomous actors, traditional process architectures and automation frameworks are no longer sufficient.

Building on the foundations established in part one of this white paper series, this one, part two, explores a modern operations architecture designed for agent-first execution. It examines how agentic systems can be organized into domain-aligned and cross-cutting capabilities, how agents can coordinate through networks rather than rigid process flows, and how autonomy can be scaled without sacrificing control.

At the heart of this architecture are two foundational elements—the ontology layer, which provides shared semantic grounding across agents, and context engineering, which ensures that agents act with the right information, constraints and intent. Together, these layers enable scalable, resilient and governable AI-powered operations.

Here, we offer a practical look at how enterprises, particularly in complex BPO environments, can evolve incrementally toward agentic operations, embedding intelligence as a distinct layer while preserving trust, accountability and operational stability.

The agent architecture: Vertical specialists and horizontal utilities

Just as a human organization is composed of specialists with deep expertise in a particular function and generalists who provide common services, an effective agentic ecosystem is composed of two types of agents—vertical agents and horizontal agents. Understanding the distinction between these two types, and how they work together, is crucial for designing effective AI-powered operations.

Vertical agents: The domain experts

A vertical agent is a goal-oriented specialist designed to handle a specific, end-to-end business function. It has deep, domain-specific knowledge, understands the unique rules and regulations of its function, and is responsible for achieving a specific business outcome. Vertical agents focus on achieving specific business outcomes in a particular domain, orchestrate multiple tasks and subprocesses to accomplish their goals, make decisions based on deep domain-specific knowledge and essentially replace, alter or significantly augment specific job roles. Following are a few examples of role-specific vertical agents and the impact they create.

- Mortgage processing: A mortgage underwriting agent would be a vertical agent. It would understand different types of loans, the bank's credit policies and the regulatory requirements for mortgage lending. Its goal would be to approve or deny a loan application based on a holistic assessment of the applicant's financial situation.
- Wealth management: A portfolio rebalancing agent would be a vertical agent. It would understand the client's risk tolerance, investment goals and the firm's asset allocation models. It would monitor the client's portfolio and automatically execute trades to bring it back into alignment with the target allocation, potentially interacting with systems such as Broadridge for execution.
- Insurance: A claims processing agent would be a vertical agent. It would handle insurance claims from submission to resolution, understanding coverage terms, fraud indicators and regulatory requirements for claims handling.

Horizontal agents: The reusable utilities

A horizontal agent is a task-oriented generalist that provides a common, reusable service that can be called upon by multiple vertical agents. These agents are not tied to a specific business function, instead they provide a foundational capability that is needed across the enterprise. They focus on executing specific tasks with high efficiency and expertise, provide specialized capabilities that can be leveraged by multiple vertical agents, excel in narrow, but widely applicable functions, and serve as reusable components in agent networks. Here are a few examples of horizontal agents and their impact.

- Document intelligence: A document extraction agent would be a horizontal agent. It would be an expert at ingesting documents (invoices, contracts, bank statements), classifying them and extracting key information. The mortgage underwriting agent could call this agent to extract data from a loan application package, and a different vertical agent from the accounts payable department could call the exact same agent to extract data from a vendor invoice.
- Customer communication: A customer notification agent would be a horizontal agent. It would be responsible for sending emails, SMS messages, and other communications to customers. Any vertical agent that needs to communicate with a customer could simply pass the message content to this agent, which would handle the complexities of formatting, delivering and tracking.
- Compliance: A compliance checking agent would be a horizontal agent that ensures adherence to regulatory requirements across multiple business processes and domains.
- Risk assessment: A risk analysis agent would be a horizontal agent that evaluates various types of risks, providing its specialized assessment to any vertical agent that requires it.

The power of collaboration

The power of this architecture comes from the collaboration between vertical and horizontal agents. The vertical agents own the business process and the domain-specific logic, while the horizontal agents provide the scalable, reusable infrastructure that prevents every team from having to reinvent the wheel. This creates a composable, agile ecosystem where new business processes can be assembled quickly by combining existing horizontal agents with new, domain-specific vertical agents. It is the key to achieving both specialization and scale.

The key distinction is that vertical agents are responsible for end-to-end outcomes in specific domains, while horizontal agents provide specialized capabilities that support multiple vertical agents across different domains.

Agent network model: Scaling collaboration for complex processes

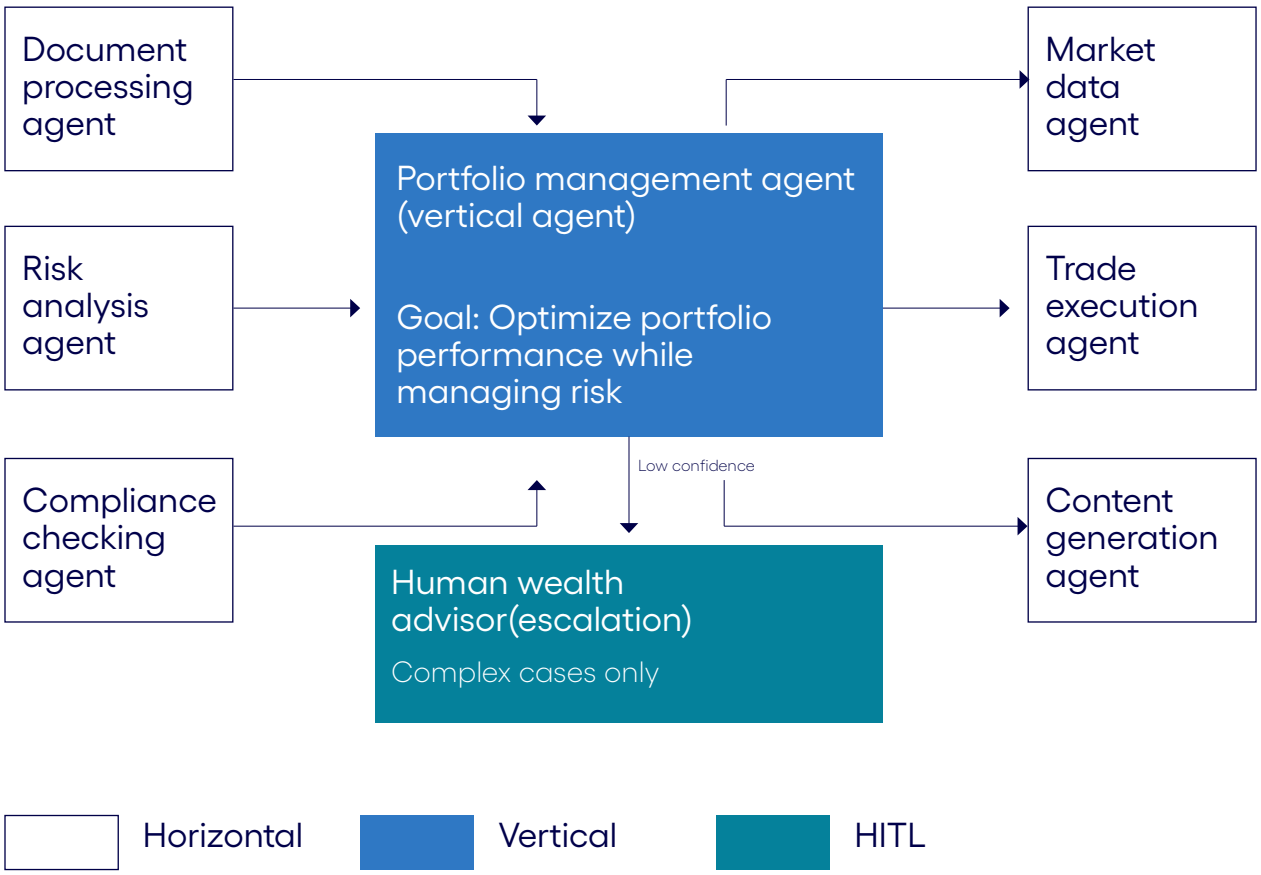
In mature AI-powered BPO operations, vertical and horizontal agents work together in agent networks to execute complex business processes. This model offers several significant advantages over monolithic AI systems and represents the future of enterprise AI architecture.

Advantages	Description
Specialization and expertise	Each agent focuses on what it does best. Vertical agents develop deep domain expertise; horizontal agents develop specialized task capabilities.
Reusability and scalability	Horizontal agents can be used across multiple processes and domains, reducing development effort and enabling rapid scaling.
Flexibility and adaptability	Agent networks can be reconfigured to support new processes without rebuilding entire systems.
Incremental improvement	Individual agents can be updated independently without disrupting the entire system.
Resilience and fault tolerance	If one agent fails, others can compensate. The network degrades gracefully rather than failing completely.

Portfolio management agent for high-value clients

Here's an example of how vertical and horizontal agents might work together in a wealth management operation that serves high-net-worth individuals. The portfolio management agent (vertical) is responsible for managing client investment portfolios, with the goal of optimizing portfolio performance while managing risk and adhering to client objectives and constraints.

The workflow proceeds as follows:



Wealth management network

The portfolio management agent receives market data, client objectives and portfolio guidelines. It develops a rebalancing strategy based on current allocations, market conditions and client goals. To implement this strategy, it delegates specific tasks to horizontal agents.

- It calls the document processing agent to extract data from account statements, trade confirmations and other financial documents. This agent uses computer vision and NLP to handle documents in various formats, returning structured data.
- It requests risk assessment from the risk analysis agent, providing current portfolio composition and proposed changes. This agent performs sophisticated risk calculations such as value at risk, stress testing and correlation analysis, and returns risk metrics and recommendations.
- It consults the compliance checking agent to ensure that proposed portfolio changes comply with regulatory requirements such as concentration limits and client-specific constraints such as ESG preferences or restricted securities.
- It uses the market data agent to access real-time and historical market data, pricing information and research from multiple data providers.

Based on inputs from these horizontal agents, the portfolio management agent makes portfolio adjustment decisions. It then calls the trade execution agent to execute trades through appropriate channels, and the content generation agent to draft client communications, explaining the changes and rationale.

If the portfolio management agent encounters a situation beyond its capabilities, perhaps a client requests a complex estate planning strategy, it escalates to a human wealth advisor, providing comprehensive context to facilitate that person's decision-making.

In mortgage underwriting, a vertical agent can automate the evaluation of loan applications by orchestrating horizontal agents for document processing, data validation, credit analysis and property valuation before making a decision or escalating to a human underwriter. Similarly, in insurance, a claims processing agent can manage a claim from submission to resolution by coordinating agents for fraud detection, damage assessment and coverage analysis, escalating complex cases to human adjusters.

Design principles of agent networks

Designing effective agent networks requires attention to several key principles:

- Clear interfaces and contracts: Each agent should have a well-defined interface, specifying what inputs it requires, what outputs it provides, what assumptions it makes and what guarantees it offers.
- Appropriate granularity: Agents should be neither too coarse-grained (limiting reusability) nor too fine-grained (requiring excessive coordination overhead).
- Loose coupling: Agents should depend on interfaces rather than implementations, enabling agents to be updated or replaced without cascading changes.
- Explicit orchestration: Vertical agents orchestrate the workflows for their domains. In more complex scenarios, dedicated orchestration agents or human orchestrators may be needed.
- Error handling and recovery: Agent networks need robust error handling, retrying using alternative agents, escalating to humans or proceeding with degraded capabilities.
- Monitoring and observability: Visibility into which agents are being called, how long operations take, what errors occur and what decisions are made is crucial for troubleshooting, optimization and governance.
- Security and access control: Not all agents should have access to all data or capabilities. Appropriate security and access controls ensure agents only access what they need and are authorized to use.

The evolution from monolithic to network architecture

The shift from monolithic AI systems to agent networks parallels the evolution in software architecture from monolithic applications to microservices. Early AI implementations in BPO often involved monolithic systems designed for specific processes. This approach has limitations such as difficulty in updating specific capabilities without affecting the whole system, limited reusability across different processes, scaling challenges and brittleness where failures in one component affect the entire system.

Agent networks address these limitations by decomposing functionality into specialized, reusable agents. This enables more rapid innovation, better scalability, greater resilience and more efficient resource utilization. However, agent networks also introduce complexity; more components to develop and maintain, coordination overhead and more complex testing and debugging. The key is finding the right balance between monolithic simplicity and network flexibility based on your specific needs and capabilities.

The ontology layer: The unseen foundation

Large language models (LLMs) are masters of language, but they are not masters of fact. They do not know things in the way a human expert does. They are probabilistic systems that predict the next word in a sequence based on the patterns they have learned from their training data. This is both their greatest strength and their greatest weakness. It allows them to generate fluent, human-like text, but it also makes them prone to hallucination, which means they invent facts, misinterpret concepts and make logical errors.

For an agentic system to be reliable enough for enterprise use, it cannot rely on the LLMs' internal, probabilistic knowledge alone. It needs an external, authoritative source of truth. This is the role of the ontology layer.

What is ontology?

Ontology is a formal, machine-readable representation of the knowledge in a specific domain. It defines the key concepts, the properties of those concepts and the relationships between them. It is a structured map of a business domain.

- Concepts: The key entities in the domain such as customer, policy, claim, shareholder, Committee on Uniform Security Identification Procedures (CUSIP) and property.
- Properties: The attributes of those concepts. For example, a customer has a name and an address; similarly, a policy has a coverage limit and a premium.
- Relationships: The connections between concepts. For example, a customer owns a policy, a policy covers a claim and a shareholder is linked to a CUSIP.

This knowledge is typically stored in a knowledge graph, a specialized type of database that is optimized for storing and querying highly interconnected data. Knowledge graphs use graph traversal methods such as cypher or SPARQL to follow explicit nodes and edges, making them ideal for answering questions such as identifying all borrowers connected to property X, who also have a loan with bank Y.

What makes ontology nonnegotiable

The ontology layer is the anchor that tethers the probabilistic reasoning of the LLM to the factual reality of the enterprise. It serves several critical functions:

- Grounding: It provides the agent with a reliable source of facts to use in its reasoning process. When an agent needs to know the coverage limit for a specific insurance policy, it can query the knowledge graph instead of trying to guess the answer from its training data.
- Disambiguation: It helps the agent understand the precise meaning of terms in a specific business context. For example, the word premium means one thing in insurance and something entirely different in investment banking. The ontology provides the context needed to avoid misinterpretation.
- Governance: It provides a mechanism for enforcing business rules and constraints. By encoding these rules in the ontology, you can ensure that the agent's actions are always compliant with business policies, even if the LLM suggests a different course of action.
- Explainability: Because the knowledge graph stores explicit relationships, every reasoning step that relies on the ontology is fully traceable. This is essential for regulatory compliance, audit trails and building trust with stakeholders.

Building an enterprise-grade ontology is a significant undertaking. It requires deep domain expertise, a disciplined approach to data modeling and a close collaboration between business and technology teams. But it is not optional. It is the unseen foundation upon which any reliable and scalable agentic system is built.

Context engineering: The fuel for intelligence

If ontology is an agent's long-term memory, then context is its short-term memory. An agent's ability to perform a task effectively is almost entirely dependent on the quality and relevance of the information it is given the moment it needs to act. The discipline of providing this information is called context engineering.

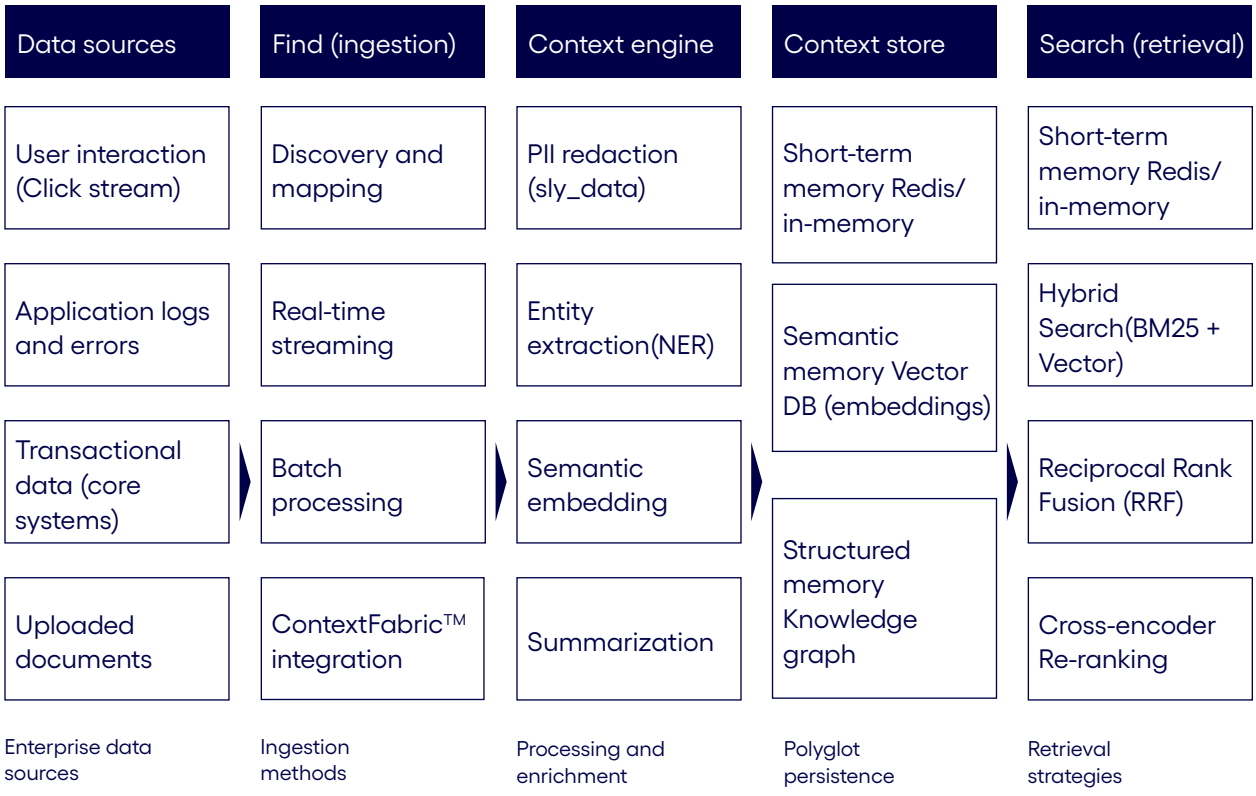
Context engineering is the process of finding, storing and searching for the information that an AI agent needs to perform its task. It is the bridge between the vast, unstructured universe of enterprise information and the specific, actionable context that an agent needs, to make decisions. This discipline is rapidly becoming one of the most critical capabilities in the enterprise AI stack.

The context lifecycle: Find, store and search

Effective context engineering involves a continuous, multistage lifecycle. The following diagram illustrates the end-to-end pipeline for real-time enterprise context ingestion.

Context capture strategy

End-to-end pipeline for real-time enterprise context ingestion



The pipeline flows left to right: Raw data is ingested from enterprise sources, processed through the context engine for security and enrichment, persisted in a polyglot store (short-term, semantic and structured memory), and retrieved via hybrid search strategies. The real-time event bus provides the foundational messaging layer that connects all stages.

- Find (ingestion): This is the process of identifying and ingesting the raw information that the agents will need. This can include structured data from databases, unstructured text from documents and emails, and semistructured data from internal websites and applications. Data sources include user interactions and clickstream data, application logs and errors, transactional data from core systems and uploaded documents. A key challenge here is that this information is often scattered across dozens of different systems. A capability that can automatically discover and map these sources, such as ContextFabricTM, can dramatically accelerate this process.
- Store (persistence): Once the information is found, it needs to be processed and stored in a way that makes it easy for the agents to consume. This typically involves a process called retrieval-augmented generation (RAG), where documents are broken down into smaller chunks, converted into numerical representations (embeddings) and stored in a specialized vector database. The context store is not a single database, it is a polyglot persistence layer with three distinct storage types.

Storage type	Technology	Purpose	Example use case
Short-term memory	Redis/In-memory	Session context for active conversations	Maintaining state during a multiturn customer interaction
Semantic memory	Vector DB (embeddings)	Similarity search across unstructured content	Finding similar past mortgage applications
Structured memory	Knowledge graph (relationships)	Explicit relationship traversal	Identifying all borrowers connected to property X

- Search (retrieval): When an agent needs to perform a task, it queries the context store to find the most relevant information. This information is then dynamically inserted into the prompt that is sent to the LLM, giving the model the specific context it needs to reason about the task and generate an accurate response.

Context engineering as a security layer

Context engineering is not just about performance. It is also a critical component of security and safety. A well-designed context engineering system provides several layers of defense.

- Mitigating prompt injections: One of the biggest security risks in agentic AI is a prompt injection attack, where a malicious user inputs text that is designed to trick the agent into ignoring its original instructions and performing an unauthorized action. By grounding the agent in a trusted set of contextual documents, you can significantly reduce its vulnerability to these attacks. The agent is instructed to base its reasoning on the provided context, not on the user's input alone.
- Building a trust layer: The context engineering pipeline itself can act as a trust layer. Before a piece of information is made available to the agents, it can be vetted, validated and tagged with metadata, indicating its source, its level of sensitivity and its reliability. This allows you to build a system where agents are only given access to the information they are authorized to see.
- Enabling human supervision: When an agent makes a decision, it can be told to cite the specific sources from its context that it used to arrive at that decision. This creates an auditable trail that allows a human supervisor to quickly verify the agent's reasoning and identify the source of any potential errors.

In an agentic enterprise, context is not just data. It is the fuel, the guardrail and the audit trail for autonomous intelligence.

Inside the context engineering technical pipeline

This chapter provides a detailed technical deep dive into the context engineering pipeline, covering the ingestion process, the storage architecture, the retrieval strategy and the reasoning mechanisms that transform raw enterprise data into grounded, actionable intelligence.

Finding and storing: The ingestion and polyglot persistence pipeline

The ingestion pipeline is a multistage process that transforms raw, unstructured enterprise data into structured, searchable context. Each stage adds a layer of intelligence.

Context lifecycle: Finding and storing deep dive

Ingestion and polyglot persistence pipeline

Stage 1: Extraction and OCR
Extract text from raw source documents
- PDFs, scanned images
- Handwritten forms
- Complex multicolumn tables
- Azure AI Document Intelligence

Stage 2: Semantic chunking
Break text into meaningful chunks
- Naive chunking (fixed tokens) vs. semantic chunking
- Split by paragraphs, clauses
- Preserves meaning and context

Stage 3: Entity resolution
Link entities to knowledge graph
- Identify shareholders, CUSIPs
- Property addresses
- Map to graph nodes
- Bridge unstructured -> structured

Polyglot persistence

Chunks

Vector store (semantic)
Stores embeddings for similarity search

Example:
- Azure AI Search (hybrid)
- ANN search on embeddings
- Unstructured data

Entities

Knowledge graph (structural)
Stores relationships and dependencies

Example:
- Azure Cosmos DB (Gremlin)
- Graph traversal (Cypher/SPARQL)
- Explicit relationships

Records

Relational store (transactional)
Stores high-integrity financial records

Example:
- Azure SQL database
- Standard SQL queries
- ACID compliance

Illustrative tool stack
Azure AI Doc Intelligence | Unstructured.io | LangChain

Synchronization plane
Example: Azure event grid + change data capture (CDC) ensure all stores remain consistent in real-time

Stage 1 (extraction and OCR): The first step is to extract text from the raw source documents. This is not a trivial task in an enterprise environment, where documents come in a wide variety of formats: PDFs, scanned images, handwritten forms, complex multicolumn tables and more. Modern extraction tools such as Azure AI Document Intelligence can handle this complexity, using computer vision and OCR to extract text with high accuracy even from challenging source material.

Stage 2 (semantic chunking): Once the text is extracted, it needs to be broken down into manageable chunks for embedding and storage. The critical distinction here is between naive chunking (splitting text at fixed token windows) and semantic chunking (splitting text by logical sections such as paragraphs, clauses or sections). Semantic chunking preserves the meaning and context of the text, which dramatically improves the quality of downstream retrieval.

Stage 3 (entity resolution): The final ingestion step is to identify and link entities within the text to the enterprise knowledge graph. For example, in a financial services context, this means identifying shareholders, CUSIPs, property addresses and other key entities, and linking them to their corresponding nodes in the knowledge graph. This step bridges the gap between unstructured text and structured knowledge, enabling powerful hybrid retrieval strategies with polyglot persistence. The processed data is stored in three complementary stores, each optimized for a different type of query. While the polyglot persistence architecture enables hybrid retrieval strategies, which are discussed in the next section, it ensures that the right data is accessible in the right format for any given query.

Store	Technology (Using Azure tech stack as an example)	Retrieval method	Strength
Vector store	Azure AI search (hybrid)	Approximate nearest neighbor (ANN) search on high-dimensional embeddings	Handles unstructured data extremely well; fast and scalable
Knowledge graph	Azure cosmos DB (gremlin)	Graph traversal (Cypher/SPARQL) following explicit nodes and edges	Explainable, deterministic; captures complex dependencies perfectly
Relational store	Azure SQL database	Standard SQL queries	High-integrity transactional data; ACID compliance

Synchronization plane: Powered by Azure Event Grid and change data capture (CDC), synchronization plane ensures all three stores remain consistent in real-time, so that an update in one store is immediately reflected in the others.

Context storage: Vector database versus knowledge graph

Choosing the right memory architecture for enterprise agents is a critical design decision. The two primary options are vector databases and knowledge graphs. They have complementary strengths and weaknesses.

Vector database	Knowledge graph
Retrieval: Approximate nearest neighbor (ANN) search on high-dimensional embeddings	Retrieval: Graph traversal (Cypher/SPARQL) following explicit nodes and edges
Best for: Example: “Find similar past mortgage applications” or ”Retrieve relevant policy documents”	Best for: Example: “Identify all borrowers connected to Property X, who also have a loan with Bank Y”
Pros: Handles unstructured data (text, images) extremely well; fast and scalable	Pros: Explainable, deterministic; captures complex dependencies perfectly
Cons: “Black box” retrieval; struggles with precise multihop reasoning or exact fact lookup	Cons: Requires rigid schema definition; hard to populate from unstructured data automatically

Recommendation: Hybrid "GraphRAG" architecture

Combine both for optimal results. Use vector search to find relevant unstructured context, then use the knowledge graph to verify facts and traverse relationships for deep reasoning.

Dimension	Vector database	Knowledge graph
Retrieval mechanism	ANN search on high-dimensional embeddings	Graph traversal (Cypher/SPARQL) following explicit nodes and edges
Best use case	Find similar past mortgage applications or retrieve relevant policy documents	Identify all borrowers connected to Property X, who also have a loan with Bank Y
Pros	Handles unstructured data (text, images) extremely well; fast and scalable	Explainable, deterministic and captures complex dependencies perfectly
Cons	Black box retrieval; struggles with precise multihop reasoning or exact fact lookup	Require rigid schema definition; hard to populate from unstructured data automatically

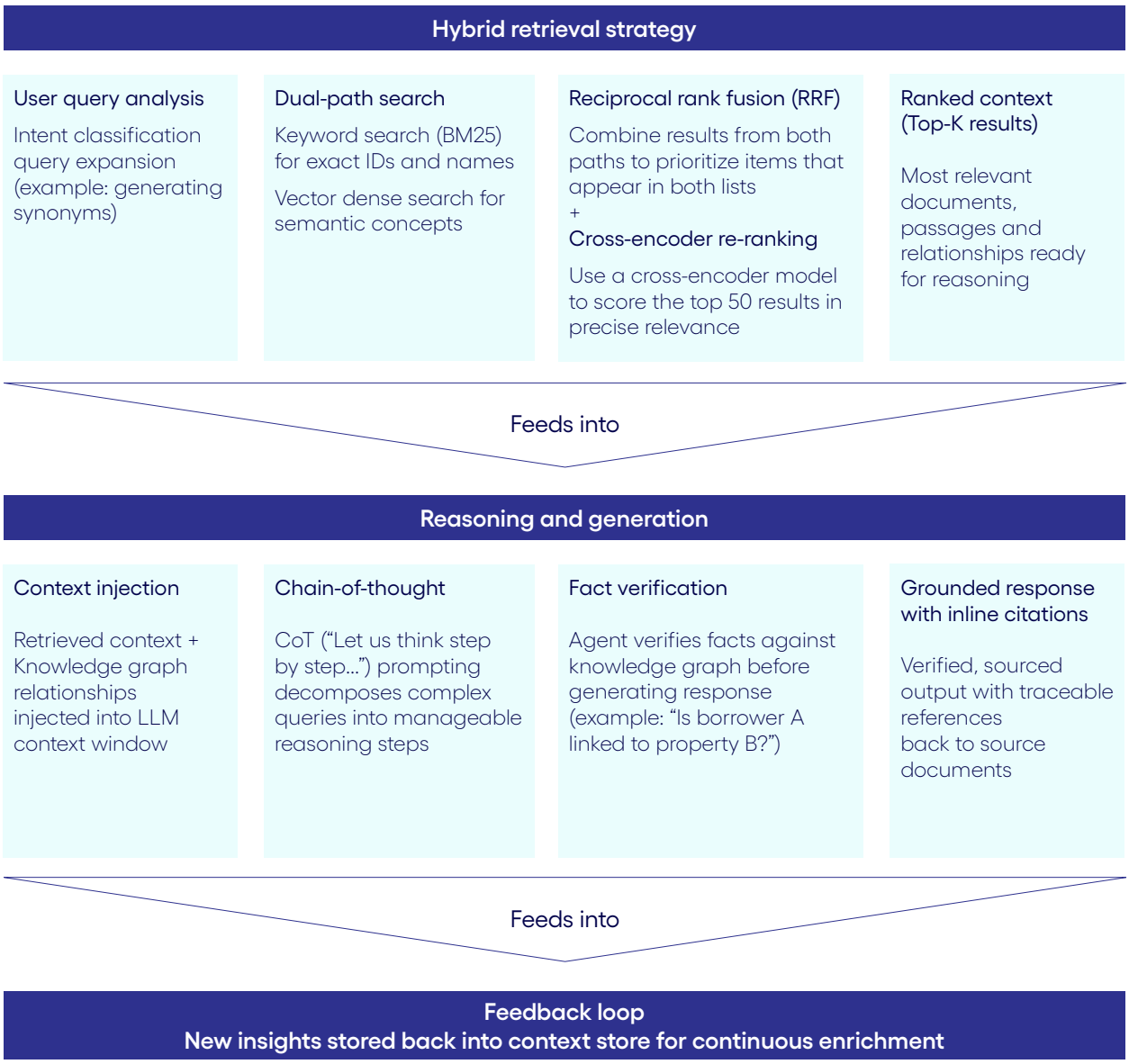
The recommendation is to have a hybrid GraphRAG architecture and combine both for optimal results. Use vector search to find relevant unstructured context, then use the knowledge graph to verify facts and traverse relationships for deep reasoning. This hybrid approach delivers the best of both worlds: the flexibility and scalability of vector search with the precision and explainability of graph-based reasoning.

Context retrieval and reasoning: From raw data to grounded intelligence

The retrieval and reasoning pipeline is where context is transformed from stored data into actionable intelligence. This pipeline has two main stages: a hybrid retrieval strategy and a reasoning and generation process.

Context retrieval and reasoning

From raw data to grounded intelligence



- Hybrid retrieval strategy: The retrieval process begins with user query analysis, where the system performs intent classification and query expansion (generating synonyms and related terms). The query is then executed through a dual-path search: a keyword-based search using BM25 for exact IDs and names, and a vector-based dense search for semantic concepts. The results from both paths are combined using reciprocal rank fusion (RRF), which prioritizes items that appear in both result lists. Finally, a cross-encoder model re-ranks the top 50 results for precise relevance.
- Reasoning and generation: Once the most relevant context is retrieved, it is injected into the LLM's context window along with knowledge graph relationships. The LLM then uses chain-of-thought (CoT) prompting (for example, "Let's think step by step..."), to decompose complex queries into manageable reasoning steps. The agent verifies its facts against the knowledge graph (for example, "Is borrower A linked to property B?"), before generating a grounded response with inline citations linking back to source documents.
- Feedback loop: New insights generated during the reasoning process are stored back into the context store, creating a continuously enriching knowledge base. This feedback loop ensures that the system becomes more knowledgeable and more accurate over time.

The strategic choice between AI agents and core system modernization

For decades, the answer to aging, inefficient enterprise systems has been to modernize them. This typically involves a multiyear, multimillion-dollar project to replace a legacy mainframe or client-server application with a modern, cloud-native platform. But the rise of agentic AI presents a radical new alternative. What if you could simply wrap it with a layer of intelligence that handles all the complexity—instead of modernizing the core?

This is one of the most profound strategic questions that enterprise leaders face today. The answer has significant implications for IT budgets, roadmaps and organizational structures.

The case for modernization

The traditional modernization approach has several advantages:

- Technical debt reduction: It directly addresses the root cause of the problem, replacing brittle, difficult-to-maintain legacy code with a modern, supportable technology stack
- Improved performance and scalability: Modern cloud-native applications are more performant, scalable and resilient than their legacy counterparts
- Easier integration: Modern systems are built with APIs, making it easier to integrate them with other applications

However, this approach also has significant downsides:

- High cost and risk: Core system modernization projects are notoriously expensive, time-consuming, and prone to failure. Industry data suggests that a significant percentage of large-scale modernization projects exceed their budgets and timelines.
- Slow time to value: It can take years before the business sees any tangible benefit from the investment.
- Business disruption: Migrating from a legacy system to a new platform can be highly disruptive to business operations.

The case for AI agents

The agentic approach offers a different set of trade-offs:

- Faster time to value: An intelligent agent layer can be built and deployed in a fraction of the time it takes to modernize a core system, delivering business value in months rather than years.
- Lower upfront cost: While not inexpensive, the upfront cost of building an agent layer is typically an order of magnitude lower than the cost of a full system replacement.
- Business agility: The agent layer is more flexible and easier to change than a monolithic core system. Business logic is encoded in the agents and the ontology, not in hard-to-change legacy code.

But this approach also has its challenges:

- Leaves technical debt unaddressed: It leaves the underlying legacy system in place. It means you still have to bear the cost and risk of maintaining it.
- Integration complexity: The agents must be tightly integrated with the legacy system, which can be complex and brittle.
- Performance overhead: The agent layer adds an additional layer of processing, which can introduce latency.

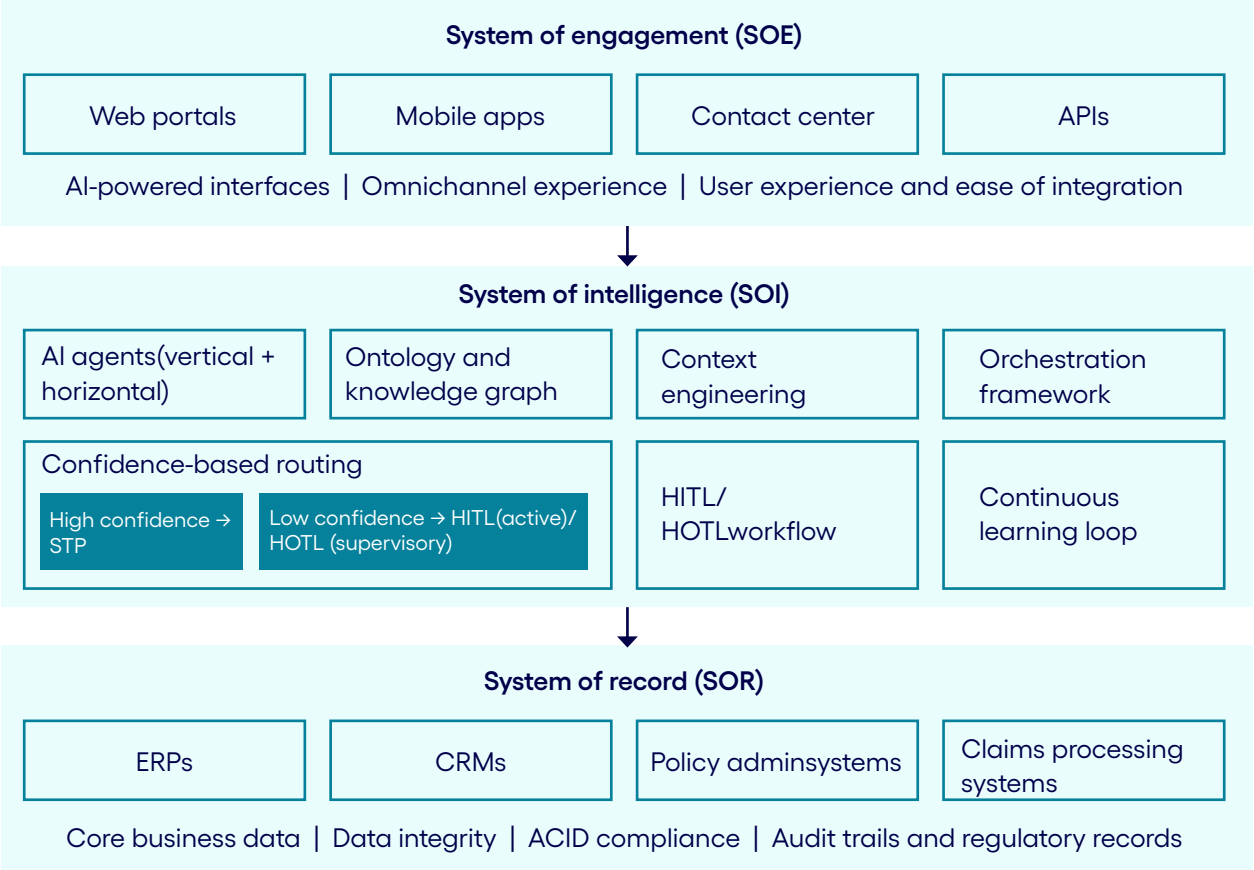
The hybrid path forward

For most enterprises, the optimal path is not an either/or choice, but a hybrid approach. The long-term goal may still be to modernize the core, but an agentic AI layer can be used as a transitional strategy to deliver immediate business value and de-risk the long-term transformation. By wrapping the legacy system with an intelligent layer, you can immediately improve the efficiency and effectiveness of the process, and the savings generated can be used to help fund the eventual modernization of the core. This allows you to innovate at the speed of AI without being held hostage by the timelines of legacy IT.

The three-layer architecture: Systems of record, engagement and intelligence

To make the agentic enterprise a reality, a new architectural paradigm is needed. While the traditional two-layer model consists of a system of record (SOR) for data and a system of engagement (SOE) for interaction, a third layer—a system of intelligence (SOI)—is also required.

The three layers defined



- System of record (SOR): This is the authoritative source of truth for the enterprise's data. It includes core systems such as ERPs, CRMs, policy administration systems and claims processing systems. The SOR is designed for data integrity, consistency and durability. It is the system that auditors and regulators care about. In the agentic architecture, the SOR is treated as a data persistence layer, not a business logic layer.
- System of engagement (SOE): This is the layer through which humans and external systems interact with the enterprise. It includes web portals, mobile apps, contact center applications and APIs. The SOE is designed for user experience, omnichannel consistency and ease of integration. It is the face of the enterprise.
- System of intelligence (SOI): This is the new layer that sits between the SOR and the SOE. It is the home of the AI agents, the ontology, the context engineering infrastructure and the orchestration frameworks. The SOI is designed for agility, learning and autonomous decision-making. It is the brain of the enterprise.

How the layers interact

The power of this three-layer architecture comes from the clean separation of concerns.

- A transaction arrives through the system of engagement (for example, a customer submits a form on a web portal)
- The SOE passes the transaction to the system of intelligence
- The SOI's orchestration engine routes the transaction to the appropriate vertical agent
- The vertical agent uses its reasoning capabilities, contextual knowledge from the SOI's knowledge base and the services of horizontal agents to process the transaction
- As part of this process, the agent reads from and writes data to SOR via APIs
- If the agent's confidence is high, the transaction is carried out via straight-through processing (STP). If confidence is low, the transaction is routed to a human via the appropriate human oversight workflow, which could be either human-in-the-loop (HITL) for active intervention or human-over-the-loop (HOTL) for supervisory review. The result is passed back to the SOE for delivery to the customer.

This separation of concerns is powerful because it allows each layer to evolve independently. You can modernize the SOR without disrupting the SOI. You can upgrade the SOE without changing the agent logic. And you can continuously improve the SOI by adding new agents, refining the ontology and improving the context engineering—without touching the underlying systems. This is the architectural key to long-term agility and scalability.

An AI builder's playbook: From readiness to reliable agents

Building production-grade AI agents is a discipline that requires a structured, engineering-led approach. This chapter provides a practical playbook for the journey from initial readiness assessment to the deployment of reliable and scalable agents.

The MAKER framework

A particularly valuable framework for building reliable agents is the MAKER (Maximal agentic decomposition, K-threshold error mitigation and red-flagging) framework, developed by Cognizant AI Lab. The MAKER framework provides a structured methodology for:

- Maximal agentic decomposition: Breaking down complex business processes into the smallest possible autonomous units of work. This decomposition is critical because it allows each agent to be focused, specialized and easier to test and validate.
- K-threshold error mitigation: Implementing a confidence-based threshold system where agents are required to meet a minimum confidence score before they can act autonomously. Transactions that fall below this threshold are automatically escalated for human review.
- Red-flagging: A systematic approach to identifying and flagging high-risk transactions that require additional scrutiny, regardless of the agent's confidence score. This provides an additional layer of safety for sensitive or high-value transactions.

The Neuro SAN orchestration platform

For orchestrating multi-agent systems at enterprise scale, the Cognizant Neuro synaptic agent network (SAN) provides the infrastructure needed to manage the complex interactions between vertical and horizontal agents. The Neuro SAN provides agent lifecycle management (deploying, monitoring and updating agents), inter-agent communication (enabling agents to call each other and share context), workflow orchestration (defining and executing complex multi-agent workflows), and performance monitoring (tracking agent performance, confidence scores, and error rates in real-time).

From readiness to deployment: A practical roadmap

The journey from readiness to reliable agents follows a structured path.

Phase 1, assessment and design (1–8 weeks): This phase involves a comprehensive assessment of the current process landscape, identification of high-value use cases, design of the target agent architecture and development of the ontology and context engineering strategy. The key deliverable is a detailed blueprint that maps each process to its target agent architecture and identifies the vertical and horizontal agents required.

Phase 2, foundation build (9–20 weeks): This phase focuses on building the core platform infrastructure such as the context engineering pipeline, the knowledge graph, the orchestration framework and the integration connectors. The key deliverable is a working platform that can support the deployment of the first agents.

Phase 3, agent development and testing (21–32 weeks): This phase involves building, training and rigorously testing the first set of agents. Testing should include unit testing of individual agents, integration testing of agent networks, stress testing under production-like volumes, adversarial testing for security and robustness and user acceptance testing with domain experts.

Phase 4, supervised deployment (33–44 weeks): The first agents are deployed in a supervised mode, where every decision is reviewed by a human. The focus is on building confidence, capturing training data and calibrating confidence thresholds.

Phase 5, progressive autonomy (45+ weeks): Based on demonstrated performance, agents are gradually given more autonomy. The STP rate is increased for low-risk transaction types and the focus shifts to scaling across more complex processes.

Architecture: The strategy in the agentic era

The transition to agentic operations is not a theoretical shift—it is a structural one. As AI agents move from experimental tools to autonomous actors within enterprise workflows, success will depend less on model sophistication and more on the architecture that surrounds them. Agent networks, grounded by ontologies and fueled by robust context engineering, provide a scalable and governable path forward—one that balances autonomy with accountability.

Critically, this architecture reframes how enterprises think about transformation. Rather than forcing an immediate choice between modernizing legacy systems or accepting their constraints, a system of intelligence creates a bridge, delivering near-term value while enabling long-term evolution. Organizations that adopt this layered, composable approach will be better positioned to close the AI velocity gap, operationalize trust at scale and turn autonomous intelligence into a durable competitive advantage. In the age of agentic AI, architecture is no longer a backend concern—it is the strategy.

Related reading

- "Confronting the AI velocity gap: A new architecture for enterprise operations"
- "The business of agentic AI: Costs, commercial models and the path forward"

References

- Andreessen Horowitz (a16z), “Unbundling the BPO: How AI Will Disrupt Outsourced Work,” 2024. Available at: https://a16z.com/unbundling-the-bpo-how-ai-will-disrupt-outsourced-work/
- Aviso AI, “The Role of Ontology in Agentic AI for Enterprise,” 2025. Available at: https://www.aviso.com/blog/ontology-layer
- Cognizant, “Context Engineering: A Key Layer for Reliable Enterprise AI,” 2025. Available at: https://www.cognizant.com/us/en/insights/insights-blog/context-engineering-for-reliable-enterprise-ai
- Cognizant, “Cognizant to Deploy 1,000 Context Engineers Powered by ContextFabric to Industrialize Agentic AI,” PR Newswire, 2025. Available at: https://www.prnewswire.com/news-releases/cognizant-to-deploy-1-000-context-engineers-powered-by-contextfabric-to-industrialize-agentic-ai-302541591.html
- Cognizant, “Cognizant Announces Multi-Agent Orchestration for its Neuro AI Platform,” 2024. Available at: https://investors.cognizant.com/news-and-events/news/news-details/2024/Cognizant-Announces-Multi-Agent-Orchestration-for-its-Neuro-AI-Platform/default.aspx

Author

Anoop Nair
Senior Vice President, Global Head of FSI - IOA
anoop.nair@cognizant.com

Follow

cognizant

Cognizant (Nasdaq-100: CTSH) engineers modern businesses. We help our clients modernize technology, reimagine processes and transform experiences so they can stay ahead in our fast-changing world. Together, we're improving everyday life. See how at www.cognizant.com or follow us **@Cognizant.**

World Headquarters
300 Frank W Burr Blvd
Suite 36, 6th Floor
Teaneck, NJ 07666, USA
Tel: (201) 801-233

European Headquarters
280 Bishopsgate
London
EC2M 4AG
England
Tel: +44 (01) 020 7297 7600

India Corporate Office
Siruseri-Software Technology Park of India (STPI)
SDB Block – Ground floor north wing
Plot No H4, SIPCOT IT Park
Chengalpattu District
Chennai 603103, Tamil Nadu
Tel: 1800 208 6999

APAC Headquarters
1 Fusionopolis Link, Level 5
NEXUS@One-North, North Tower,
Singapore 138542
Tel: + 65 6812 4000

WF 4486200