taazaa

Modernizing Legacy Systems with AI

Agents, Pipelines, and Knowing the Difference

Syed Shabih Hasan, Ph.D.

Table of Contents

Executive Overview

The adoption of AI in enterprise software has followed a predictable pattern: initial excitement, rapid prototyping, and then the hard realization that production systems demand a fundamentally different engineering discipline than demos.

Given that most organizations will leverage the frontier models built by OpenAI, Anthropic, Google, and the growing open-source ecosystem, the differentiator is not access to AI. It is how well an organization understands which AI architecture solves a given problem, and whether it can build that architecture to production standards.

In this white paper, we present a framework for thinking about AI-driven modernization across three distinct architectural patterns:

- Intelligent automation
- Orchestrated AI pipelines
- Agentic systems

We draw on five production systems built by Taazaa across municipal government, financial services, healthcare, consumer fintech, and food service. Each system required a different architectural approach. None of them required the same one.

The first impetus for this white paper is the growing tendency to label any multi-step AI workflow as “agentic.” This is imprecise, and imprecision in architecture leads to over-engineering, unnecessary costs, and fragile systems.

The second is the observation that most legacy modernization conversations begin with the technology (“Should we use agents?”) rather than the problem (“Where are humans doing work that machines should handle, and what is the minimum viable AI architecture to replace that work reliably?”). We propose that the latter question is the one that produces results.

The AI Architecture Spectrum

Before discussing the specific systems, it is important to establish a taxonomy. The industry has compressed a wide range of capabilities into the term “agentic AI,” but there is a meaningful distinction between an AI system that classifies documents and routes them through a deterministic pipeline, and one that autonomously reasons about which tools to invoke based on a user’s natural-language query.

Conflating the two leads to poor architectural decisions: autonomous agents deployed where a pipeline would be faster and more reliable, or rigid pipelines deployed where the problem genuinely requires dynamic reasoning.

We classify AI systems into three categories.

1. Intelligent Automation

The AI performs specific, well-defined tasks—classification, extraction, transformation—within a pipeline whose control flow is deterministic once the AI produces its output. The system is sophisticated, but the routing is engineered, not emergent. This is analogous to a well-designed assembly line where each station performs complex work, but the sequence of stations is fixed.

2. Orchestrated AI Pipelines

Multiple AI components operate in sequence with conditional branching. The system adapts its path based on intermediate results, but within predefined boundaries. There is intelligence in the routing logic, not only in the individual processing steps. One can think of this as a decision tree where each node performs substantive AI work.

3. Agentic Systems

The AI reasons about its next action, selects tools dynamically, and modifies its plan based on what it discovers. The workflow is not predetermined; it emerges from the agent's interaction with its environment and its evaluation of intermediate results. This is the most flexible and the most complex pattern, and it is appropriate for a narrower set of problems than its current popularity would suggest.

Each pattern has a domain where it excels. The engineering skill is in selecting the right one. In the following sections, we present five production systems that illustrate the full spectrum.

Intelligent Document Processing for Municipal Tax Administration

The Problem

A municipality in Ohio processed every tax return manually. Auditors spent thousands of hours each year scanning bulk documents, sorting them by form type, keying extracted data into their taxation portal, and then—in theory—auditing the results.

In practice, the volume of returns meant that most were approved without substantive review. The auditors were not auditing. They were performing data entry.

This is a pattern we observe repeatedly in legacy operations: the humans in the system are consumed by mechanical work, leaving no capacity for the judgment work they were hired to do.

The Architecture

Taazaa built an event-driven AI platform that replaced the entire manual pipeline.

The system ingests tax documents from three channels—bulk batch scans via a file-sharing platform, an e-filing portal, and an auditor portal—and processes them through four automated stages:

- QR-code-based document separation
- AI classification
- Optical character recognition
- Structured field extraction

The architecture is purely asynchronous and Kafka-native, exposing zero REST endpoints. A Separator container detects QR code boundary markers (supporting four separator types, tenant-configurable) to split bulk PDFs into individual returns.

An Extractor container then classifies each document by filing type (Business, Individual, or Withholding) using Google Gemini, extracts text via Azure Document Intelligence with custom models per form type and automatic fallback to a prebuilt layout model on failure, and maps the OCR output to over 50 structured tax fields using LLM-powered extraction with database-driven prompts and field definitions.

Ensuring Determinism in a Multi-Stage AI Pipeline

Every step in this pipeline is deterministic once the AI component produces its output. The Gemini classifier identifies the form type; the system routes it to the correct extraction path. There is no reasoning about what to do next—the pipeline knows.

The AI components within each step are doing genuinely complex work (classifying ambiguous scanned documents, extracting structured data from noisy OCR output, mapping fields across different form types), but the orchestration logic is fixed.

This distinction matters: a fixed orchestration is easier to test and debug and produces more predictable throughput than an agent-based approach would for this problem.

Architectural Decisions of Note

Two design decisions are worth highlighting.

First, the system uses a cross-pipeline bridge: when the batch extractor classifies a document as Individual or Withholding rather than Business, it atomically transitions that document from the batch processing schema to the form extraction pipeline in a single database transaction. This prevents data loss and orphaned records during pipeline transitions.

Second, all prompts and field mappings are stored in the database rather than in code. This means the system's AI behavior—what it extracts, how it classifies, how it maps fields—can be updated, A/B tested, or rolled back without redeployment.

Multi-tenant isolation ensures that each municipality's data resides in its own PostgreSQL database, which is resolved dynamically from the Kafka message's tenant identifier. Taxpayer identifiers such as EINs and SSNs are protected by high-security encryption at rest and hash-based indexing that enables deduplication lookups without exposing raw values. Sensitive data never reaches the LLM in unencrypted form.

This is the legacy modernization pattern that is most often overlooked: you do not replace the tax portal. You eliminate the manual labor that feeds it. The auditors now spend their time on substantive review—the work they were originally hired to do.

Intelligent Coaching for Retail Traders

The Problem

Taurex, a regulated multi-asset trading platform, operates in a crowded brokerage market where competitors primarily differentiate on spreads, execution speed, and platform compatibility. While some competitors have begun introducing AI-adjacent features—typically third-party behavioral analytics integrations or lightweight trading record analyzers—none have built a purpose-designed data engineering pipeline that produces auditable, pre-computed coaching from the platform's own trade data.

The Architecture

Taazaa built the Taurex AI Coach as a daily analytics pipeline with a strict separation of concerns. Raw trade data is pulled from the platform's MSSQL database into PostgreSQL via a daily ETL process. Twelve dbt metric models then compute win rates, drawdown patterns, risk exposure, position sizing consistency, behavioral indicators, and other performance metrics.

These pre-computed analytics produce a composite Trading Health Score. The score and its constituent metrics are then fed into an LLM to generate personalized coaching reports optimized for mobile delivery. The pipeline is orchestrated via SQS and deployed on ECS Fargate.

The critical design principle: the LLM performs zero mathematical computation. Every number, every ratio, every trend in the coaching report is deterministically computed in the dbt layer. The LLM's role is strictly narrative—translating structured analytics into plain-language coaching that a retail trader can act on. When a trader sees a Health Score with a note about inconsistent position sizing, every figure in that report is auditable and reproducible. The LLM cannot hallucinate the math because it never performs any. This is an intentional architectural constraint, not a limitation.

It is worth noting that no direct competitor in the retail brokerage space has replicated this architecture. The AI features that exist across the industry rely on third-party integrations or surface-level analytics interfaces. A custom ETL-to-LLM pipeline that ingests the broker's own trade data and produces pre-computed, auditable coaching represents a genuine competitive moat for the platform.

> **The LLM cannot hallucinate the math because it never performs any. This is an intentional architectural constraint, not a limitation.**

Agentic Healthcare AI with Autonomous Reasoning

The Problem

MediPulse Intellix addresses a fundamentally different class of problem. Hospital staff need to ask natural-language questions against a large and diverse body of organizational policies, standard operating procedures, and training materials. The questions are unpredictable in both scope and complexity.

"What is the protocol for patient discharge after cardiac surgery?"

Requires finding and synthesizing information across multiple documents.

"Can a nurse practitioner authorize this medication during the night shift?"

Requires the system to understand role-based policies, shift-specific rules, and drug-specific protocols simultaneously.

A retrieval pipeline with fixed logic cannot handle this problem space reliably. The system needs to reason about which documents to retrieve, evaluate whether the retrieved context adequately answers the question, and decide whether to search again with reformulated parameters. This is the defining characteristic that moves a system from pipeline to agent: the AI must evaluate the quality of its own intermediate results and adapt its behavior accordingly.

The Architecture

Taazaa built a LangGraph ReAct agent—the Reasoning and Acting pattern where the AI observes the result of each step and decides its next action. The agent selects from available tools and retrieves documents using a three-level hierarchical RAG system (global policies, tenant-specific policies, and site-specific policies), with strict priority resolution that ensures a site-level policy always overrides a global default.

The retrieval mechanism itself is two-stage.

STAGE 1

A hybrid vector-plus-keyword search (via AWS Bedrock Titan embeddings and pgvector) retrieves the top 50 candidate passages.

STAGE 2

A cross-encoder reranker then narrows the results to the top two most relevant passages.

This two-stage approach eliminates a well-known failure mode: pure vector search can return passages that are semantically similar to the query but factually incorrect for the specific context.

What makes this system genuinely agentic is the decision loop. The agent does not follow a predetermined path. It evaluates retrieval quality, reformulates queries when results are insufficient, and maintains per-user semantic memory via Langmem backed by pgvector.

This means a follow-up question (such as, “What about for pediatric patients?”) correctly inherits the context of the prior cardiac discharge question without requiring the user to restate their context. Input classification runs asynchronously in parallel for analytics, ensuring the reasoning loop is not bottlenecked by secondary processing.

This is the correct application of an agentic architecture. The problem space is open-ended, the retrieval targets are heterogeneous, the quality of any single retrieval step is unpredictable, and the system must reason about adequacy before responding.

A deterministic pipeline would either return incorrect answers to complex queries or require so many hard-coded branches that it would become unmaintainable.

Orchestrated Pipelines with Conditional Intelligence

Not every system fits cleanly at either end of the spectrum.
Two additional production systems illustrate the middle ground.

Budget University: Curriculum-Scoped Financial Literacy

Budget University is a financial literacy chatbot covering 11 content categories that span banking, investing, credit, 401(k) planning, and related topics.

The system uses a LangGraph StateGraph with five nodes and conditional branching. It reformulates conversational queries for multi-turn support, routes to category-specific retrieval paths, and runs post-generation grounding verification via keyword-overlap scoring—without requiring an additional LLM call.

A dual-layer safety system combines prompt-based guardrails with regex post-checks to enforce strict curriculum-only answering. The system uses local embeddings (MiniLM-L6-v2, 384 dimensions) into Qdrant, which keeps latency low and eliminates dependency on external embedding APIs.

The conditional branching is the key differentiator from pure automation: the system's path through the graph depends on intermediate classification results. However, the set of possible paths is finite and predefined. This is more than automation, but it is not an agent.

CookinGenie MenuWizard: Multi-Use-Case Content Generation

CookinGenie's MenuWizard is a content generation microservice for restaurant operators and home chefs, orchestrating 11 distinct AI use cases. These range from à la carte menu generation to multi-course Chef Special menus, meal prep plans (leveraging Gemini's thinking mode for complex planning), field-level regeneration via JSONPath patching (enabling a chef to regenerate a single dish description without rerunning the entire menu), pricing estimation with parallelized per-course LLM calls via ThreadPoolExecutor, and two-stage image generation where Gemini produces a descriptive summary and Bedrock Nova Canvas renders the image.

Prompts are database-versioned and hot-swappable, enabling behavior changes without code deployment. The system uses optimistic concurrency on image generation jobs and a business markup on ingredient cost calculations. Like Budget University, the intelligence lies in the orchestration—the system branches, adapts, and verifies—within the boundaries the architecture defines.

Cross-Cutting Architectural Patterns

Across all five systems, we observe several architectural principles that repeat. These are, arguably, more important than whether a given system is classified as “agentic” or not. We present them here because we believe they represent the engineering discipline required for production AI systems, regardless of architectural pattern.

Separation of Computation from Narrative Generation

In every project, the LLM never performs mathematical computation. Pre-compute layers, dbt models, keyword-overlap scoring, cost calculations with business markup—all numeric work occurs in deterministic code. The LLM's responsibility is language. This is not a stylistic preference; it is a reliability constraint. LLMs are probabilistic systems. Deterministic computation must remain deterministic.

Prompts as Configuration, Not Code

Whether stored in TOML files, PostgreSQL tables, or database-versioned records with hot-swap capability, prompts are externalized and managed as data artifacts. This enables A/B testing of AI behavior, rollback to previous prompt versions, and tenant-specific customization—all without code deployment.

Structured Failure Paths

Dead-letter queues with retryable flags, status audit trails with full state history, per-document error tracking with pipeline step identification—these are not afterthoughts. Production AI systems fail in environments where LLM APIs are rate-limited, where OCR models return partial results, and where document scans are illegible. The question is whether those failures are silent or observable and recoverable.

PII Isolation from the Model Layer

Aggregated analytics, scoped retrieval context, encrypted identifiers, and hash-based deduplication—the data sent to language models is always sanitized or pre-processed. This is both a compliance requirement and a security design principle: it reduces the blast radius of any vulnerability in the model layer.

Event-driven Architectures for Scale Absorption

SQS queues, Kafka topics with configurable consumer groups and concurrent workers, async processing with semaphore-controlled concurrency—these systems are designed to absorb volume spikes without requiring architectural changes. This is particularly important in domains like tax processing, where volume is highly seasonal.

The Correct Architecture Determines Success

Every enterprise has workflows where humans are performing work that machines should handle. The municipality's tax auditors were doing data entry. Taurex's traders were interpreting their own performance without structured analytics. Hospital staff were searching through policy binders instead of asking questions in natural language.

The modernization opportunity in each case was real, but the correct AI architecture was different every time.

- An agent for the hospital
- A deterministic pipeline for the tax office
- A pre-compute-to-narrative chain for the trading platform
- Orchestrated conditional logic for the chatbot and the menu generation system

The organizations that will succeed in AI modernization are not those that pursue the most advanced architecture available. They are the ones that correctly match the architecture to the problem and possess the engineering discipline to build it for production operation, not demonstration.

Surface-level application of AI models can only take an organization to the level of how well it can prompt. Understanding the full spectrum of AI architectures, and when each is appropriate, is what produces systems that run reliably at scale.

If your team is evaluating where AI fits in your existing systems—or debating whether a given problem requires an agent, a pipeline, or something in between—that is a conversation worth having with engineers who have built all three. We welcome the discussion.

About the Author

Syed Shabih Hasan

Shabih is the Principal Architect of AI & Big Data at Taazaa. He holds a Ph.D. in Computer Science with a focus on AI for resource-constrained systems and brings deep expertise in AI, machine learning, and data engineering.

Over the years, Shabih has helped build and scale data capabilities across industries such as healthcare, real estate, finance, and retail.

About Taazaa

Taazaa helps enterprises modernize legacy systems with AI-readiness as a core objective. Our AI-first modernization approach extracts, structures, and deploys historical data assets to deliver competitive advantages that traditional modernization cannot achieve.

Contact: info@taazaa.com