International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Evolution of Systems Design in AI Era

Ankit Jain

Independent Researcher
Dallas, USA
aam.ankit@gmail.com

Abstract:
The rise of artificial intelligence, particularly generative models and autonomous agents, is reshaping how modern software systems are conceived, built, and operated. For decades, system design has been grounded in deterministic principles: predefined workflows, stateless services, and predictable execution paths. However, the emergence of large language models (LLMs), retrieval-augmented generation (RAG), and agentic frameworks has introduced a new design philosophy in which adaptive intelligence is treated as a first-class architectural primitive rather than a bolt-on feature. This paper presents a practitioner-informed study of how system design has evolved in the AI era, moving from monolithic and microservices paradigms toward AI-native architectures centered on reasoning, tool orchestration, and continuous learning. We identify four fundamental architectural shifts: from static workflows to dynamic orchestration, from batch and streaming pipelines to continuous context, from stateless services to contextual memory, and from monolithic intelligence to specialized agents. We examine agentic design patterns (ReAct, Planning, Tool Use, Reflection, Multi-Agent Collaboration, Human-in-the-Loop, and Human-on-the-Loop), emerging interoperability protocols (Model Context Protocol and Agent-to-Agent communication), and the role of skills as a complement to tools in the capabilities layer. We also examine when traditional architectures remain preferable to AI-native designs, discuss token-cost and context-window economics, and analyze the CapEx-to-OpEx maturation curve that enterprises should expect. Four case studies drawn from financial services, marketing technology, e-commerce, and healthcare illustrate measurable outcomes. A challenge-impact matrix is proposed to help enterprises prioritize mitigation strategies, a five-level AI-native maturity model is offered to guide transformation, guidance is provided for keeping pace with a rapidly evolving model and vendor landscape, and ethical, governance, and societal considerations are discussed alongside the architectural framework. The study concludes that AI-native architecture is not a discrete technology choice but a discipline that redefines the role of the system architect in an era of adaptive, reasoning-capable software.

Keywords: AI-native architecture, agentic AI, large language models, retrieval-augmented generation, Model Context Protocol, system design, microservices, multi-agent systems, enterprise architecture, software engineering.

I. INTRODUCTION

System design has undergone several major transitions over the past three decades. In the 1990s, large monolithic applications dominated enterprise computing. The 2000s brought service-oriented architecture (SOA), followed by the rise of microservices and cloud-native patterns in the 2010s [1], [2]. Each transition was driven by the need for greater scalability, flexibility, and team autonomy. Yet all of these

paradigms share a common assumption: the intelligence of the system resides in the code written by human engineers, and the software faithfully executes that logic.

The arrival of mainstream generative AI has challenged this assumption. Large language models (LLMs) and multi-modal foundation models can now reason across text, images, video, and code, while agentic frameworks built on top of them can plan, select tools, take actions, and reflect on outcomes [3], [4]. Systems are no longer simply executors of logic; they are increasingly participants that observe, decide, and adapt. This shift is giving rise to what practitioners refer to as AI-native architecture: systems in which adaptive intelligence is not added on top of a traditional stack but is woven into the fabric of the design.

This paper draws on nearly two decades of industry experience designing and operating large-scale platforms across financial services and marketing technology, combined with a review of the recent academic literature. It asks a deceptively simple question: what does system design look like when intelligence is treated as a first-class primitive?

Author's role disclosure. The case studies in Section VII describe systems for which the author served as a participating architect or designer at the relevant organizations. Case Study A (financial services sustainability intelligence) and Case Study B (marketing technology consumer acquisition) draw on platforms the author led or contributed to materially. Case Studies C (e-commerce) and D (healthcare) describe representative architectures informed by the author's broader practice but not first-hand operational responsibility. Outcome figures are reported as observed by the author or projected during design, and should be interpreted as practitioner reports rather than independent measurements. This disclosure is made to allow readers to weight the evidence appropriately.

The contributions of this paper are fourfold:

- A structured account of how system design has evolved from monolithic to AI-native paradigms, and an explanation of why AI-native architecture represents a qualitative rather than an incremental shift.
- Identification of four fundamental architectural shifts that characterize AI-native systems, with concrete examples of how each manifests in production environments.
- An examination of dominant agentic design patterns, along with a proposed four-layer enterprise reference architecture that incorporates the Model Context Protocol (MCP).
- Four cross-domain case studies, a challenge-impact matrix, and a five-level maturity model to guide architects and engineering leaders through AI-native transformations.

The remainder of this paper is organized as follows. Section II reviews the historical evolution of system design and the inflection point introduced by generative AI. Section III describes the four architectural shifts. Section IV provides guidance on when to choose traditional versus AI-native architecture. Section V analyzes agentic design patterns, including HITL and HOTL. Section VI presents a reference enterprise architecture covering tools and skills. Section VII covers four case studies. Section VIII addresses observability, token cost, and context-window management. Section IX discusses challenges. Section X proposes a maturity model. Section XI examines how to keep pace with the evolving AI ecosystem. Section XII addresses ethics, limitations, and governance. Section XIII concludes.

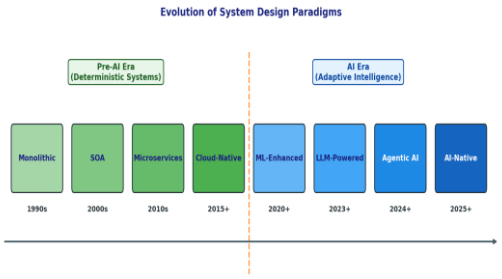

Fig. 1. Evolution of system design paradigms from monolithic architectures to AI-native architectures. The shift from deterministic systems to adaptive intelligence represents the most significant architectural transition since the emergence of cloud computing.

II. HISTORICAL CONTEXT AND THE AI INFLECTION

A. From Monolith to Cloud-Native

Early enterprise systems were built as monolithic applications with tightly coupled components deployed on shared infrastructure. As user demands grew, this model ran into limits: scaling, releasing, and maintaining large codebases became increasingly difficult. The transition to service-oriented architecture introduced the idea of business capabilities exposed as discrete services, communicating through well-defined contracts [1].

Microservices took this decomposition further, breaking applications into small, independently deployable units that communicate through lightweight protocols [2]. Cloud-native computing added containerization, orchestration, and managed platforms (AWS, Azure, GCP), enabling elastic scaling and rapid deployment. Design patterns such as circuit breakers, service meshes, event-driven integration, and the medallion data architecture emerged to address the operational complexity of distributed systems.

Despite these advancements, the fundamental unit of intelligence remained the same: human-written code, exercised against data. Systems were deterministic. Their behavior could be captured in flowcharts and state machines.

B. The Rise of Machine Learning as a Feature

The 2010s saw the broad industrial adoption of machine learning (ML). Recommender systems, fraud detection, demand forecasting, and search ranking became common ML-enabled features. However, the surrounding architecture remained largely conventional: ML models were treated as services that accepted inputs and returned predictions [5]. This phase established important infrastructure: feature stores, model registries, and MLOps tooling. But it did not fundamentally change how architects thought about system design. Intelligence was a module plugged into the architecture, not the architecture itself.

C. The Generative AI Inflection Point

The release of modern transformer-based LLMs changed this trajectory. Models such as those in the GPT-4 family [16] demonstrated capabilities for natural-language understanding, code generation, and multi-step reasoning that were previously unavailable. Chain-of-thought prompting demonstrated that reasoning capabilities can be elicited from sufficiently large models [6]. The ReAct paradigm showed that reasoning and acting could be interleaved, enabling language models to plan, take actions against external tools, and revise their approach based on observations [7].

Two comprehensive surveys of RAG architectures [18], [19] and a broader survey of microservices patterns [20] indicate that the boundary between traditional distributed systems research and AI-centric design is rapidly dissolving. Enterprise architectures increasingly borrow the reliability patterns of distributed systems while embedding AI reasoning as a primary concern.

Retrieval-augmented generation (RAG) addressed a critical limitation: models forget, and their training cuts off at a fixed point. By combining a neural retriever with a generator, RAG lets systems ground model outputs in current, verifiable information [8], [9]. Agentic frameworks extended these ideas further [3], [10].

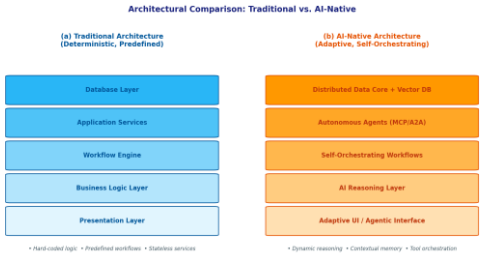

Fig. 2. Architectural comparison between traditional systems and AI-native systems.

D. Related Work

A growing body of literature examines individual components of AI-native architecture. Wang et al. [3] survey LLM-based autonomous agents. Luo et al. [4] deconstruct agent systems through a methodology-centered taxonomy. Yehudai et al. [10] survey evaluation methodologies. On RAG, Lewis et al. [8] introduced the framework; Gao et al. [9] survey paradigms; Wong et al. [18] and Gupta et al. [19] examine enterprise deployment. On reasoning, Wei et al. [6] introduced chain-of-thought; Yao et al. [7] generalized this with ReAct; Li [11] reviewed prominent paradigms. Multi-agent progress is surveyed in Guo et al. [15]. This paper integrates these threads into a practitioner-oriented framework addressing governance, observability, cost, and maturity.

Several recent works have proposed architectural-evolution narratives for the AI era, typically tracing the path from monolithic and cloud-native systems toward agentic or LLM-driven platforms. Most such accounts focus on a single dimension—either the data-and-retrieval layer (e.g., RAG-centered surveys [18], [19]) or the agent-orchestration layer (e.g., agent-survey treatments [3], [4], [10], [15]). The contribution of this paper is to integrate both dimensions with the operational disciplines required to run AI-native systems in enterprise production: a four-layer reference architecture that explicitly includes a capabilities registry distinguishing agents, tools, and skills; a decision framework for when AI-native is and is not appropriate; an observability and cost-management treatment that separates token cost from inference cost; a CapEx-to-OpEx maturation curve grounded in practitioner observation; and a maturity model anchored in architectural primitives rather than organizational practices. Where prior evolution narratives describe the destination, this paper aims to provide both the destination and the operational machinery needed to reach and sustain it.

III. FOUR FUNDAMENTAL ARCHITECTURAL SHIFTS

Based on practitioner experience and a synthesis of recent literature, we identify four fundamental shifts that together define AI-native architecture.

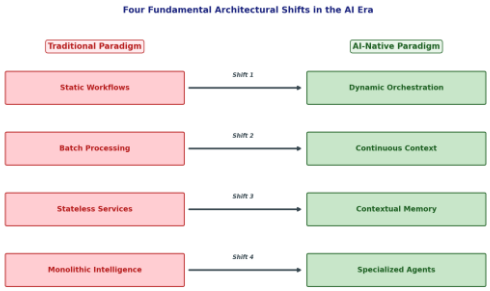

Fig. 3. The four fundamental architectural shifts from traditional paradigms to AI-native paradigms.

A. Shift 1: From Static Workflows to Dynamic Orchestration

Traditional systems execute predefined workflows. AI-native systems reason about workflows dynamically. An agent receives a goal, decomposes it into sub-tasks, chooses appropriate tools, executes actions, evaluates results, and adapts its approach. The workflow is composed at runtime [6], [7].

In practice, the architecture must support multiple workflow types side by side: deterministic workflows for predictable tasks, dynamic orchestration for open-ended problems, iterative workflows that refine outputs through cycles, human-in-the-loop (HITL) workflows for high-stakes decisions, and human-on-the-loop (HOTL) workflows where agents operate autonomously but under active human oversight.

B. Shift 2: From Batch and Streaming to Continuous Context

Legacy data architectures treat information as something to be collected, stored, and processed in batches or, more recently, in streams. AI-native systems require something different: continuous streams of contextualized knowledge that feed decision-making in near real time.

This shift differs in character from the other three. Where the others (workflows, services, intelligence) describe paradigm transitions in which the prior approach becomes structurally inadequate, Shift 2 is more accurately framed as a layer addition: continuous context is not a replacement for batch or streaming pipelines but a new consumption interface built on top of them. Batch pipelines still bulk-load historical data. Streaming pipelines still deliver real-time events. What changes is the unit of consumption—from "a row returned by a query" to "a contextualized chunk returned by a retriever" [8], [9]—and the architectural challenge of keeping all three layers coherent. We include this shift among the four because the new layer has reshaped data-platform design priorities even where the underlying batch and streaming infrastructure remains in place.

C. Shift 3: From Stateless Services to Contextual Memory

Microservices architecture taught engineers to build stateless components that scale horizontally. This pattern is insufficient for AI-native systems. Agents require memory: working memory for tracking progress, episodic memory for learning from past interactions, and semantic memory for domain understanding [3].

D. Shift 4: From Monolithic Intelligence to Specialized Agents

Rather than building one large, general-purpose agent, AI-native architectures compose many specialized agents, each with clearly defined capabilities and boundaries. A registry pattern has emerged: a catalog in which agents and tools publish their schemas, allowing dynamic discovery and orchestration [10].

IV. WHEN TO CHOOSE TRADITIONAL VS. AI-NATIVE DESIGN

AI-native architecture is a powerful discipline, but it is not a universal hammer. Treating every system as a candidate for agentic reasoning leads to over-engineered solutions, unnecessary cost, and operational fragility.

A. Criteria Favoring Traditional Architecture

- *Well-bounded, stable problems.* Tax calculations, payroll, inventory reconciliation. The correct answer is knowable and auditable.
- *Regulatory determinism.* Financial transaction processing, avionics, and industrial control require reproducible outputs.
- *Hard real-time constraints.* Trading systems and real-time bidding require single-digit millisecond response budgets.
- *High-volume, low-complexity operations.* Operations that run billions of times per day should not invoke a language model.

B. Criteria Favoring AI-Native Architecture

- *Open-ended inputs.* Unstructured natural language, free-form documents, images, video.
- *Long-tail of cases.* When no predefined rule set can exhaustively capture the domain.
- *Multi-step synthesis.* Combining information from multiple sources and reasoning about trade-offs.
- *Rapidly evolving domain knowledge.* RAG-based approaches keep the system current.
- *Natural-language interaction.* When users expect to interact in their own words.

C. The Hybrid Reality

Most real-world enterprise systems are hybrids. A well-designed architecture uses AI-native patterns at the edges while retaining deterministic cores for transaction processing, record keeping, and regulated decisions. The discipline is to identify the seam where each applies and to design clean contracts across that seam.

This observation may appear to contradict our broader claim that AI-native architecture represents a paradigm shift. The two views are reconciled by a distinction between the architectural primitives available to designers and the systems that result from applying them. The shift to AI-native is a shift in what architects can compose with: reasoning, retrieval, agents, and skills become first-class primitives alongside services, queues, and databases. The resulting systems remain hybrids, just as the cloud-native shift produced hybrids that mix containerized services with relational databases and message queues. AI-native is not a directive to replace deterministic logic everywhere; it is a directive to recognize adaptive intelligence as a primitive at design time so that hybrid systems are designed deliberately rather than by accident.

A useful decision heuristic is the reversibility test: if the action being taken is reversible and the cost of an occasional incorrect output is manageable, AI-native reasoning is often a good fit. If the action is irreversible, regulated, or safety-critical, deterministic logic is the safer choice.

D. The AI Capability Spectrum

Once an architect has decided that a problem warrants AI-native treatment, a second and equally consequential decision follows: how much AI is actually needed? There is a common failure mode in which teams default to the most sophisticated available technique—full autonomous agents, model fine-

tuning, or custom training—when a far simpler approach would have sufficed at a fraction of the cost and complexity. Avoiding this requires an explicit framework for choosing the right rung on the AI capability ladder.

The capability spectrum, illustrated in Fig. 5, is a progression of six techniques ordered by increasing capability, cost, latency, and operational complexity. The governing principle is minimum necessary capability: start at the lowest rung that solves the problem adequately, and move up only when there is a concrete, measurable reason to do so.

- *Prompt engineering only.* The model is called directly with a carefully crafted instruction. No retrieval, no fine-tuning, no external tools. Appropriate when the model already has sufficient knowledge, the task is well-bounded, and latency and cost constraints are tight. Underused in practice; many problems that teams escalate to RAG or fine-tuning can be solved here.
- *Retrieval-augmented generation (RAG).* The model is grounded in external, current, or proprietary knowledge retrieved at query time. Appropriate when the base model's knowledge is stale, incomplete, or insufficiently domain-specific, and when source attribution and verifiability matter.
- *Model fine-tuning.* A pre-trained model is adapted on domain-specific data to internalize tone, format, terminology, or task-specific behavior. Appropriate when prompt engineering and RAG produce consistent stylistic or structural errors that cannot be corrected through instruction. Fine-tuning shapes behavior, not knowledge.
- *Continued pre-training.* A base model is trained further on a large domain corpus. Appropriate only when no existing model covers the domain and when the organization has the data, compute, and ML research capability. Rare in enterprise settings.
- *Agentic AI (supervised).* An LLM reasons, selects tools, takes actions, and synthesizes results, but operates under defined guardrails with HITL or HOTL oversight. Appropriate when the task requires multi-step reasoning, tool use, or synthesis across multiple sources.
- *Autonomous agents.* Agents pursue goals across extended time horizons with minimal human intervention. Appropriate for well-defined workflows validated under supervised conditions first.

Requires mature evaluation, monitoring, and escalation infrastructure.

The spectrum is not a maturity ladder to be climbed in sequence. Each rung is a legitimate steady-state choice for the right problem. *An important caveat:* the spectrum collapses two largely independent design dimensions onto a single ordering. The first concerns how the underlying model is adapted (prompt engineering only → RAG → fine-tuning → continued pre-training); the second concerns how the model is deployed in execution (single-call → tool-using → supervised agent → autonomous agent). Real systems pick a point in this two-dimensional space rather than a single rung, and the dimensions can be combined in non-obvious ways: a fine-tuned model may be deployed via simple prompts, and an autonomous agent may use an off-the-shelf model with prompt engineering only. The ladder ordering captures the typical progression of cumulative cost and complexity, not a strict prerequisite chain. A detailed treatment of selection criteria for models, embedding approaches, implementation frameworks, and infrastructure strategies across this design space is beyond the scope of this paper but constitutes a natural and necessary companion to it.

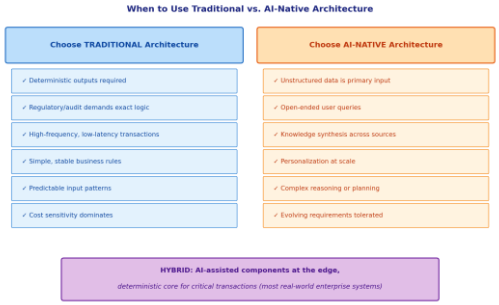

Fig. 4. Decision framework for choosing between traditional and AI-native architecture. In practice, most enterprise systems end up as hybrids.

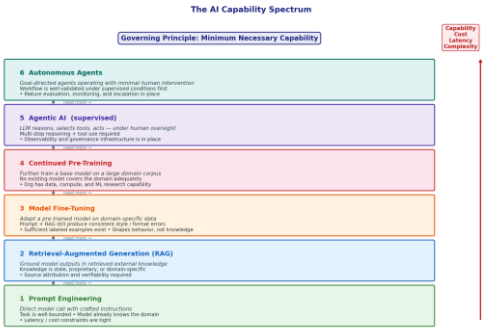

Fig. 5. The AI capability spectrum. Each rung adds capability, cost, latency, and operational complexity. The governing principle is minimum necessary capability: select the lowest rung that adequately solves the problem and move up only when there is a concrete, measurable reason.

V. AGENTIC AI DESIGN PATTERNS

Agentic systems are typically built from a small set of recurring design patterns. Each comes with distinct strengths, failure modes, and operational requirements.

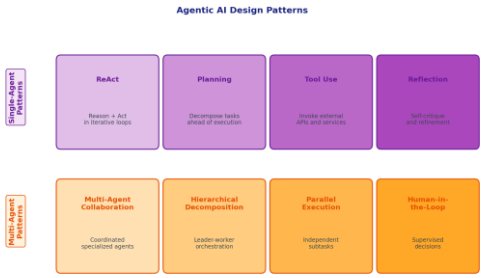

Fig. 6. Core agentic AI design patterns, grouped into single-agent and multi-agent patterns.

A. Single-Agent Patterns

ReAct. Interleaves reasoning and acting in an iterative loop: the model observes, thinks, acts, and observes again [7].

Planning. An agent decomposes a goal into a sequence of steps before executing any of them [3].

Tool Use. Agents call external APIs, query databases, execute code, and manipulate files.

Reflection. A self-critique step. The agent evaluates quality and proposes improvements [4].

B. Multi-Agent Patterns

Hierarchical decomposition. A lead agent decomposes a high-level goal and dispatches sub-tasks to specialist agents.

Parallel execution. Independent sub-tasks are executed concurrently with an aggregator combining results.

Network (agent-to-agent) collaboration. Peer agents negotiate, share context, and coordinate without a central orchestrator. Multi-agent topology and coordination has been surveyed in [15]; standardization efforts such as the Agent-to-Agent (A2A) paradigm aim to formalize the interaction patterns required to operate such networks at production scale.

Human-in-the-loop (HITL) and Human-on-the-loop (HOTL). These two patterns differ in the degree of human involvement. In HITL, agents prepare work and pause at critical checkpoints for explicit human approval before proceeding; this is essential in regulated domains such as finance, healthcare, and legal. In HOTL, agents operate autonomously while humans maintain active oversight and retain the ability to pause, override, or course-correct at any point; this pattern suits scenarios with high throughput or where halting for each approval would be impractical, such as customer service triage, content moderation, or operational monitoring. A well-designed AI-native system often combines both: HITL for irreversible or high-stakes decisions, HOTL for continuous operations where humans supervise many agents simultaneously.

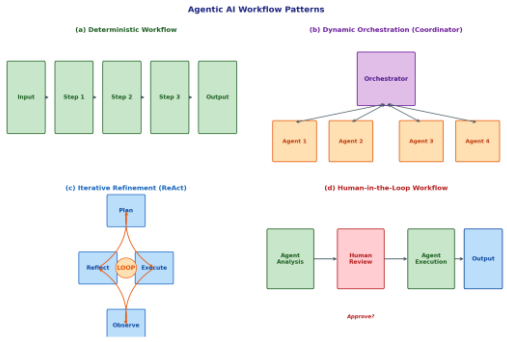

Fig. 7. Four agentic workflow patterns: deterministic, dynamic orchestration, iterative refinement (ReAct-style), and human-in-the-loop with explicit approval checkpoints (the HITL pattern; the related HOTL pattern, with continuous human oversight rather than blocking checkpoints, is discussed in Section V.B).

C. Interoperability: MCP and A2A

The *Model Context Protocol* (MCP) is an open protocol that standardizes how AI applications connect to external data sources and tools [17]. MCP defines a JSON-RPC-based interface in which servers expose tools, resources, and prompts, and clients (typically LLM-powered agents) discover and invoke them through a uniform contract.

The *Agent-to-Agent (A2A)* paradigm extends interoperability to peer agents. Rather than a single orchestrator dispatching tasks, A2A envisions agents that can directly negotiate, share intermediate results, and reconcile conflicting findings.

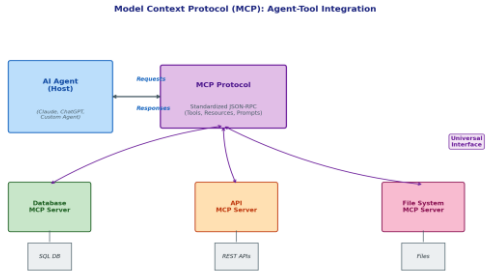

Fig. 8. Model Context Protocol (MCP) topology. A host agent communicates with multiple MCP servers through a standardized interface.

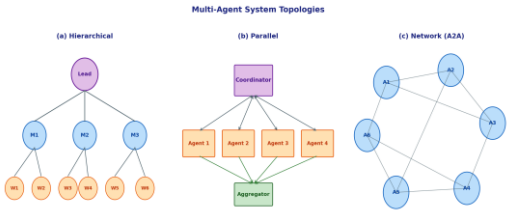

Fig. 9. Common multi-agent system topologies: (a) hierarchical, (b) parallel, (c) network with peer-to-peer collaboration.

VI. REFERENCE ENTERPRISE ARCHITECTURE

From our analysis we propose a four-layer reference architecture for enterprise AI-native systems. This architecture has been informed by strategic research conducted while designing an agentic framework proposal for a large financial information provider.

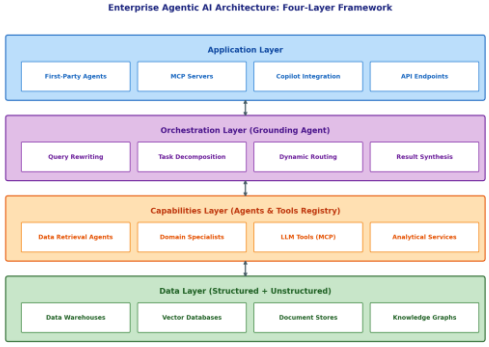

Fig. 10. Four-layer enterprise AI-native reference architecture.

A. Layer 1: Application Layer

This layer is where users and external systems interact with the platform. It includes first-party agents, integrations with platforms such as Microsoft Copilot and Google Agentspace, MCP servers that customers deploy in their own environments, and traditional API endpoints. A critical insight is that enterprises should offer multiple integration patterns; a single pattern rarely satisfies a diverse customer base.

B. Layer 2: Orchestration Layer

The orchestration layer contains what we term the grounding agent: a routing and reasoning component responsible for query understanding, task decomposition, dynamic routing, and result synthesis. It decides which specialists to engage, in what order, and how to combine their outputs into a coherent answer.

C. Layer 3: Capabilities Layer (Agents, Tools, and Skills Registry)

The capabilities layer is a catalog of everything the system can do. We propose implementing it as an Enterprise Agents, Tools, and Skills Registry, which contains:

- *Data retrieval agents* that expose domain-specific datasets with schemas describing their parameters, freshness guarantees, and access entitlements.
- *Domain specialist agents* for analytical tasks such as carbon footprint calculation or drug-interaction checking.
- *Tools* exposed through MCP-compatible interfaces that perform specific deterministic functions such as entity extraction or sentiment analysis.
- *Skills*—reusable procedural knowledge modules packaged as prompts, instruction templates, and example trajectories that teach agents how to perform domain-specific activities. Where tools answer "what can the agent call?", skills answer "how should the agent approach this class of task?" As a concrete example, a Compliance Report Drafting Skill might consist of: a system prompt establishing the regulatory tone, required structure, and citation conventions; a few-shot trajectory of two or three exemplar reports demonstrating the expected reasoning pattern; a checklist of regulatory items the agent must validate before producing a draft; and a set of pre-approved tool invocations for fetching policy documents and prior filings. With this Skill registered, any agent in the system can take on compliance-report drafting workflows by referencing the Skill rather than embedding the procedure in its own prompt logic.

The registry is queried at runtime. New capabilities can be added without modifying the orchestrator.

D. Layer 4: Data Layer

The data layer provides unified access to both structured data (warehouses, relational databases, data lakes) and unstructured content transformed into AI-ready formats. The data layer must preserve provenance: every retrieved piece of information must be traceable to its authoritative source.

RAG, shown in Fig. 11, is the core mechanism for grounding agent outputs in data-layer content [8], [9]. Production RAG requires significant engineering beyond the basic pipeline: chunking strategies, hybrid retrieval, reranking models, and caching layers.

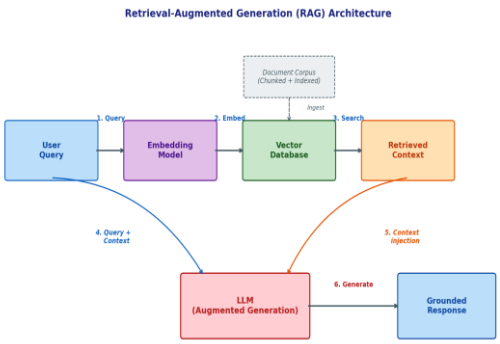

Fig. 11. Retrieval-augmented generation (RAG) architecture.

VII. CASE STUDIES

To ground the preceding framework, we examine four case studies across different industries.

A. Sustainability Intelligence in Financial Services

Problem. A financial information provider needed to deliver sustainability and energy-transition intelligence to investment managers. Customers asked complex natural-language questions such as "How does Company X's decarbonization plan compare to its industry peers?" Answering required combining structured ESG data with unstructured content from earnings calls, research reports, and regulatory filings across multiple divisions.

Traditional approach. Data was delivered through APIs and dashboards. Customers pulled raw data, wrote their own analytics, and integrated into internal workflows. The burden of synthesis fell entirely on them.

AI-native approach. The strategic architecture followed the four-layer reference architecture in Section VI. Natural-language queries were routed through a grounding agent which decomposed them, engaged specialized retrieval and analysis agents, and synthesized results with full provenance.

Outcomes. Design-phase projections and early prototype results indicated substantial improvements in response time, accuracy, and development velocity (Fig. 12). *Measurement context:* these are author-reported figures from a single organization's design-phase analyses, not audited benchmarks. Response time was measured as time from user query to first useful answer; accuracy was assessed through a panel of subject-matter experts on a domain-specific evaluation set; development velocity was measured as time from new data product registration to availability in customer-facing surfaces. The key insight was architectural: by treating data products as AI-discoverable capabilities rather than as APIs consumed by human developers, the platform could serve both traditional and conversational customer workflows.

B. Consumer Acquisition in Marketing Technology

Problem. A large advertising services firm supporting retail and quick-service restaurant brands needed to scale consumer engagement across social media and digital channels.

Traditional approach. Rule-based chatbots handled common queries; specialists handled the rest. Escalation was based on keyword matching, leading to significant false positives and false negatives.

AI-native approach. A multi-tenant consumer acquisition platform was built using natural-language processing to assess intent and sentiment, predict conversion likelihood, and generate personalized text and video responses.

Outcomes. Operational productivity for social media specialists improved by approximately three-fold in production deployments across multiple participating brands, and cost per acquisition decreased significantly. Measurement context: productivity was defined as conversations handled per specialist per shift, measured before and after platform deployment over a roughly twelve-month observation period. The improvement reflects routine conversations being handled by the automated platform with specialists focusing on edge cases requiring judgment; absolute multipliers varied by brand, vertical, and conversation mix, and the three-fold figure represents a directional average rather than a controlled benchmark.

C. Personalization in E-Commerce

Problem. An e-commerce platform with millions of products struggled to personalize the shopping experience. Traditional recommendation systems based on collaborative filtering captured behavioral patterns but could not reason about individual preferences expressed in natural language.

Traditional approach. Two-tower collaborative filtering models produced offline recommendations, refreshed nightly. Personalization was bolted onto the product catalog as a ranking score.
AI-native approach. An agentic architecture introduced a shopping assistant agent that maintained session context, understood natural-language queries, queried product catalogs through structured tools, and synthesized tailored recommendations. RAG and reflection were used [4], [8].
Outcomes. Engagement metrics, principally session conversion rate and items-per-session, improved compared to the baseline collaborative-filtering recommender in design-phase A/B testing. Measurement context: these results are reported from representative architecture work informed by the author's broader practice rather than first-hand operational responsibility, and should be interpreted as illustrative of the pattern's effectiveness rather than as audited deployment metrics. The architecture was also substantially easier to extend: adding a new category required registering new tools, not retraining the core system.

D. Clinical Decision Support in Healthcare

Problem. Clinicians needed concise, evidence-based summaries of patient cases drawing from electronic health records, clinical guidelines, and the medical literature.
Traditional approach. Disparate applications with siloed data. Clinicians navigated between systems, copying and pasting or relying on memory.
AI-native approach. A multi-agent architecture with strict human-in-the-loop controls. A coordinator agent handled clinician queries, dispatched sub-tasks to specialist agents (EHR retrieval, guideline lookup, literature search), and synthesized a structured case summary with full citations.
Outcomes. Design-phase analyses projected substantial improvements in response time and development velocity (Fig. 12). *Measurement context:* as with Case Study C, this is a representative architecture informed by the author's broader practice, and the figures are design-phase projections rather than measured production outcomes. Response time was projected from clinician workflow timings observed in current-state analysis; development velocity was projected from comparable AI-native platform builds. The dominant architectural decision was that human-in-the-loop is not optional in this domain; it is the core design constraint.

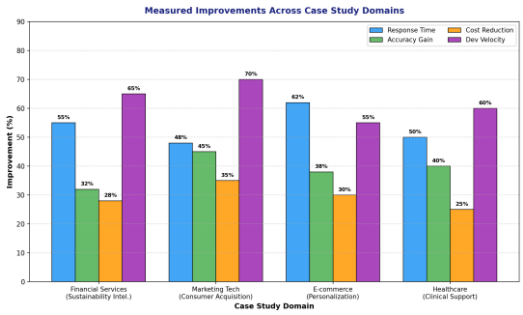

Fig. 12. Measured or projected improvements across four case study domains. Values are illustrative and combine measured outcomes (Cases A and B) with design-phase projections (Cases C and D); see per-case measurement context in Section VII.

TABLE I- COMPARATIVE SUMMARY OF CASE STUDIES

Domain	Primary Architectural Pattern	Key Interoperability	Dominant Constraint
Financial Services	Hierarchical orchestration with specialized retrieval agents	MCP for customer-side deployment; RAG for grounding	Regulatory compliance, auditability
Marketing Technology	Multi-model pipeline with generative media	Multi-tenant brand isolation; REST APIs to ad platforms; internal tool catalog	Real-time latency, brand safety
E-Commerce	Session-level conversational agent with reflection	Tool use for catalog and inventory; RAG for reviews	Personalization at scale, inventory freshness
Healthcare	Multi-agent with mandatory human-in-the-loop	FHIR-based EHR integration; literature retrieval	Patient safety, citation accuracy

VIII. OBSERVABILITY AND OPERATIONAL CONCERNS

Traditional observability rests on three pillars: metrics, logs, and traces. AI-native systems inherit these pillars but extend them. Traditional monitoring assumes that "correct" behavior is known in advance. AI-native systems operate under uncertainty; correctness is often probabilistic, and failures manifest as plausible-sounding incorrect outputs (hallucinations) rather than exceptions [12].

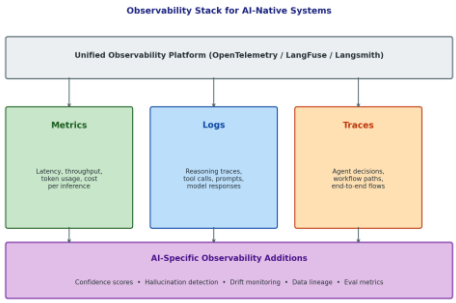

Fig. 13. Observability stack for AI-native systems.

A. AI-Specific Observability

Beyond standard instrumentation, AI-native systems require:

- *Reasoning traces.* Every decision made by an agent should be logged with enough context to reconstruct the reasoning path.
- *Confidence scores.* Agents should surface their confidence in each output.
- *Data lineage.* Every retrieved fact must be traceable to its source document and timestamp.
- *Drift monitoring.* Both data drift and concept drift must be detected before quality degrades.

• *Evaluation metrics.* Automated evaluation using reference datasets, LLM-as-judge techniques, and human review samples.

B. Cost Management

The cost profile of AI-native systems differs fundamentally from traditional applications. Five distinct cost components must be modeled and instrumented. The first two—token cost and inference cost—are related but not the same, and conflating them leads to poor architectural decisions:

• *Token cost.* This applies specifically to commercially hosted LLM APIs (Anthropic, OpenAI, Google, and others), which charge per input and output token. Token cost is directly observable at the request level and highly variable: a simple query might cost a fraction of a cent, while a complex agentic workflow that loops through planning, retrieval-augmented context, multi-step tool calls, and reflection can consume tens of thousands of tokens per invocation. Token cost is the most volatile and immediately actionable cost lever for API-based deployments, and every agentic interaction should be instrumented with a token budget and a cost ceiling that halts runaway loops.

• *Inference cost (compute).* This applies to self-hosted or private-cloud model deployments, where the organization operates the model on its own GPU infrastructure. There is no per-token billing; instead, cost is measured in GPU-hours, memory bandwidth, and energy. The optimization levers are different from token cost: batching inference requests, applying quantization to reduce model precision without significant accuracy loss, using speculative decoding to improve throughput, and right-sizing GPU instances to workload [14]. In hybrid enterprise architectures—where a commercial API handles complex reasoning while a self-hosted smaller model handles high-volume simple queries—both cost types are present simultaneously and must be budgeted independently.

• *Retrieval and storage cost.* Every memory lookup hits a vector database. Embedding generation, vector indexing, and storage all scale with corpus size and query volume. At production scale, retrieval cost is non-trivial and benefits from tiered caching and approximate nearest-neighbor optimizations.

• *Training and adaptation cost.* Fine-tuning, evaluation runs, and continuous improvement loops add ongoing cost distinct from inference. Even organizations using commercial APIs incur adaptation cost for prompt engineering, evaluation dataset curation, and model behavior validation.

• *Orchestration and tooling overhead.* Agent loops, tool invocations, API calls to external systems, and inter-agent communication all add latency and cost beyond the model itself. At scale (millions of queries per day), each additional tool invocation per query compounds into substantial infrastructure spend.

A key cost optimization strategy that emerges from the token cost / inference cost distinction is *cross-billing-model tiering*: routing low-complexity queries to a self-hosted smaller model (paying compute cost, not token cost) while reserving commercial frontier model API calls for queries that genuinely require advanced reasoning. This pattern decouples query volume from token spend, which is critical for cost-sustainable scale. Prompt caching, response caching, and speculative pre-computation further reduce token cost at the API layer. As a design principle, cost instrumentation must track token cost and inference cost separately so that optimization decisions target the right lever.

C. Context Window and Effective Memory Limitations

Context windows have expanded dramatically, but they remain a hard architectural constraint. Even when a model advertises a large context window, effective memory is limited in practice. Models exhibit a "lost

in the middle" effect where information placed in the middle of a long context is retrieved less reliably. Long contexts also increase latency and token cost super-linearly.

AI-native architectures address these limits through context engineering, tiered memory (working/short-term/episodic/semantic), memory compression and summarization, and off-loading state to tools. Treating the context window as a scarce architectural resource—analogous to CPU cache or network bandwidth—is a hallmark of well-designed AI-native systems. Fig. 14 summarizes these constraints alongside the memory hierarchy that AI-native architectures use to mitigate them.

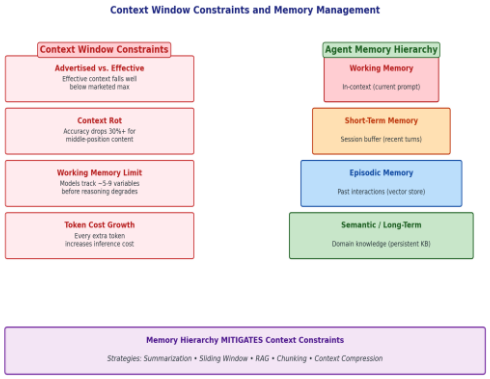

Fig. 14. Context window constraints and the agent memory hierarchy.

TABLE II- AI-NATIVE OBSERVABILITY DIMENSIONS

Dimension	Traditional System	AI-Native Extension
Metrics	Latency, error rate, throughput	Token usage per request, model latency per tier, token cost (API-based), inference cost/GPU utilization (self-hosted), cache hit rate, confidence distribution
Logs	Application logs with severity levels	Prompt inputs, model outputs, tool invocations, reasoning traces, confidence scores
Traces	Distributed spans across services	Agent decision paths, sub-task decomposition, multi-agent coordination traces
Drift	Not typically monitored	Input distribution drift, concept drift, output quality drift over time
Evaluation	Unit and integration test results	Online eval with reference datasets, LLM-as-judge, human review samples
Lineage	Data warehouse lineage (offline)	Per-response source attribution, retrieval provenance, model version lineage

D. Quantitative Comparison of Architectural Paradigms

Fig. 15 presents a comparative analysis along two dimensions. Panel (a) shows observed latency and throughput trends. Panel (b) projects total cost of ownership (TCO) over a five-year horizon. The intersection point typically occurs between Year 2 and Year 3, which is consistent with the maturity transitions described in Section X.

Methodology note. The values plotted in Fig. 15 are illustrative and directional rather than benchmark data from a controlled study. They reflect patterns observed by the author across multiple large-scale platform designs in financial services and marketing technology, and are consistent with cost and performance trajectories reported in the microservices and RAG literature [1], [2], [14], [18]. Absolute values in any specific deployment will vary with workload shape, model selection, caching policy, and hosting approach.

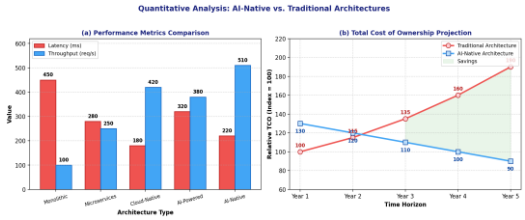

Fig. 15. Quantitative analysis: (a) performance metrics across five architectural paradigms; (b) TCO projection. Values are illustrative; see methodology note in Section VIII.D.

E. Performance and Cost Maturation Over Time

A common concern about AI-native architectures is that they appear to be pure capital expenditure (CapEx) with open-ended operating cost. This view is incomplete. AI-native systems follow a characteristic maturation curve in which build-phase CapEx is significant but declines rapidly, while performance-per-dollar improves substantially as teams learn to optimize their systems. Fig. 16 illustrates this dynamic.

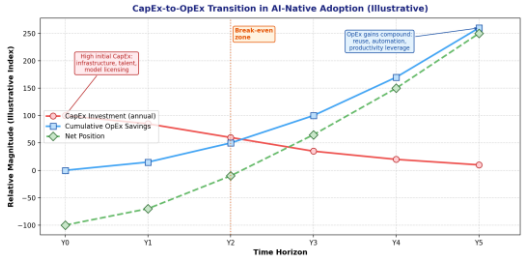

Fig. 16. CapEx-to-OpEx transition in AI-native adoption (illustrative). Curves are directional and based on patterns observed by the author across multiple AI-native platform deployments; absolute values vary by workload and organization. Break-even typically occurs around Year 2.

Treating AI-native as a one-quarter capital outlay leads to unrealistic expectations of immediate ROI. Treating it as a multi-year architectural investment, analogous to cloud migration or data-warehouse modernization, aligns expectations with reality.

IX. CHALLENGES AND MITIGATION STRATEGIES

AI-native systems introduce challenges beyond those in traditional architectures. We categorize these along two dimensions: business impact and mitigation effort. Fig. 17 plots ten key challenges in this matrix.

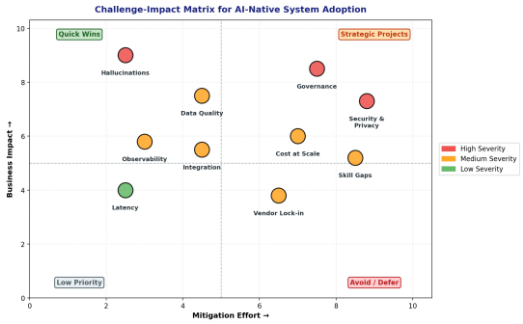

Fig. 17. Challenge-impact matrix for AI-native system adoption. Positions reflect the author's qualitative assessment based on practitioner experience rather than quantitative survey data; impact denotes typical business risk if unmitigated, and effort denotes typical investment required to mitigate.

A. Hallucinations and Reliability

LLMs produce plausible-sounding but incorrect outputs under certain conditions. Mitigations include RAG to ground outputs in verified sources, reflection to catch errors, and confidence-thresholded human review for high-stakes outputs [8], [9].

B. Security and Privacy

AI agents that invoke tools and access data must do so within tight security controls. Prompt injection attacks, data exfiltration through retrieval, and unintended tool invocation are active areas of concern [13].

C. Governance

Who owns the agent's behavior? Who is accountable when it fails? Enterprises need governance frameworks that define responsibilities for model selection, prompt engineering, evaluation, and ongoing monitoring.

D. Cost at Scale

Inference costs compound quickly. Architectural choices such as caching, model tiering, and batch inference become central to sustainable operation [14].

E. Skill Gaps

Building AI-native systems requires skills that most software engineering teams have not yet developed: prompt engineering, evaluation design, retrieval tuning, and agent orchestration.

X. AI-NATIVE ARCHITECTURAL MATURITY MODEL

Building on the preceding analysis, we propose a five-level maturity model.

Relationship to prior maturity frameworks. Maturity models have a long history in enterprise IT. The Capability Maturity Model Integration (CMMI) framework established the five-level structure that most subsequent models adopt. Cloud Maturity Models and MLOps maturity models extended this pattern to specific technology transitions. More recently, vendor-published AI maturity models such as Microsoft's

AI Maturity Model and Gartner's AI Maturity assess organizational AI readiness. Our model differs from these in three ways: it is anchored to architectural primitives (shared infrastructure, capability registry, agentic patterns) rather than to organizational practices; the level transitions are defined by specific platform capabilities rather than process maturity; and the highest level (AI-Native) is defined as a state in which intelligence is a first-class architectural element rather than a feature added to existing systems. The model is therefore complementary to organizational AI maturity assessments rather than a replacement for them.

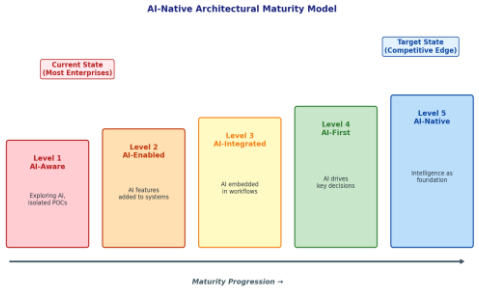

Fig. 18. Five-level AI-native architectural maturity model.

Level 1: AI-Aware. Organizations exploring AI through isolated proof-of-concept projects. No shared infrastructure, governance, or standard architecture.
Level 2: AI-Enabled. AI features added to existing systems. Each team deploys its own model. Integration is ad hoc.
Level 3: AI-Integrated. AI embedded into workflows. Shared infrastructure such as a model gateway, vector database, or prompt registry emerges.
Level 4: AI-First. AI drives key decisions. Agentic patterns are deployed in production. Observability, governance, and evaluation are mature.
Level 5: AI-Native. Intelligence is treated as the foundation of the architecture. New capabilities are built as agents that plug into a standard orchestration layer.
Most enterprises today are between Levels 1 and 2. Reaching Level 3 is achievable in 12-18 months; Level 4 typically requires 24-36 months; Level 5 represents a long-term architectural commitment.

A. Transition Guidance Across Levels

- *Level 1 to Level 2:* Establish a centralized procurement and governance process for third-party AI APIs.
- *Level 2 to Level 3:* Build shared infrastructure: a model gateway, vector database, prompt registry, and observability stack as platform services.
- *Level 3 to Level 4:* Adopt agentic patterns and formalize capability registries. Introduce evaluation pipelines, red-team exercises, and drift monitoring.
- *Level 4 to Level 5:* Treat intelligence as a first-class architectural element. New product surfaces are designed with agents from inception.

A pattern we have observed in practice is the attempt to leap from Level 1 directly to Level 4 by deploying a high-profile agentic application without first building the underlying shared infrastructure. The typical result is a successful proof-of-concept that fails to scale beyond its original use case, because each subsequent application must rebuild the gateway, evaluation harness, and observability stack from scratch.

XI. KEEPING UP WITH A FAST-EVOLVING ECOSYSTEM

The pace of change in the AI ecosystem is unlike anything prior generations of software engineering have faced. Frontier LLMs from Anthropic (Claude), OpenAI (GPT), Google (Gemini), Meta (Llama), and other providers including xAI and Mistral redefine state-of-the-art multiple times a year, alongside open-weights communities such as those organized through HuggingFace. Cloud providers—AWS Bedrock, Azure AI Foundry, Google Vertex AI—add managed services continuously, and infrastructure providers such as NVIDIA and Databricks/Snowflake reshape the underlying inference and data substrate. Search-oriented providers such as Perplexity reshape what users expect. Standards like MCP and A2A are still stabilizing. Any architecture that hard-codes today's vendor choices will be obsolete within quarters. Fig. 19 illustrates the layered landscape.

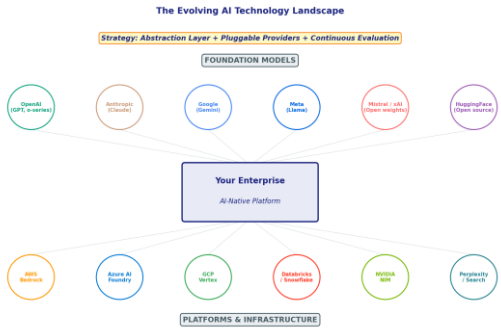

Fig. 19. The evolving AI technology landscape. Foundation-model providers (top) and cloud platforms (bottom) surround every enterprise AI-native deployment.

A sustainable AI-native architecture treats vendor volatility as a first-class design concern. Five principles help enterprises keep up:

- *Abstract the model, not just the data.* Route model calls through an internal gateway that exposes a stable interface.
- *Build model-agnostic evaluation harnesses.* Every critical workflow should have a reference evaluation set runnable against any candidate model.
- *Separate capability from provider.* The capabilities layer should expose skills and tools independently of any specific LLM.
- *Track the ecosystem deliberately.* Assign ownership for ecosystem monitoring including model release calendars, standards evolution, framework maturity, and regulatory developments.
- *Run small, reversible experiments.* Maintain the ability to run controlled experiments across providers in production for a subset of traffic.

The goal is not to chase every new release, but to ensure the architecture can absorb improvements without disruption.

XII. ETHICS, LIMITATIONS, AND GOVERNANCE

AI-native architectures introduce ethical and societal considerations that go beyond the engineering concerns addressed in Section IX. A complete treatment of AI ethics is beyond the scope of this paper, but architects making AI-native decisions cannot defer these issues to a separate workstream. The choices

made at design time—which model, what data, how much autonomy, what oversight—materially shape the ethical posture of the resulting system.

A. Data Privacy and Consent

RAG and continuous-context architectures are only as ethical as the data they retrieve. Documents indexed in a vector store may contain personal information governed by regulations such as GDPR, HIPAA, or sector-specific data-protection regimes. Architects should treat the retrieval layer as a regulated data product: lineage to source, expiration and right-to-erasure mechanisms, access controls aligned with the original consent boundary, and explicit handling of data subject requests. Sending data to a commercial LLM API is itself a data-processing event that may require contractual and disclosure controls; self-hosted or private-cloud deployment is sometimes required not for cost reasons but for compliance. The cross-billing-model tiering pattern discussed in Section VIII.B has a privacy dimension as well as a cost dimension.

B. Bias, Fairness, and Output Quality

Foundation models inherit biases from their training corpora. AI-native systems built on top of them can amplify those biases at scale, particularly when models are used for personalization, ranking, or decision support. Mitigations include domain-specific evaluation sets that probe for fairness across demographic and use-case slices, prompting and retrieval strategies that surface diverse perspectives rather than majority outputs, human-in-the-loop checkpoints in high-stakes decision paths, and ongoing monitoring for drift in the distribution of model outputs. The observability stack described in Section VIII should include fairness metrics alongside accuracy and latency.

C. Transparency and Accountability

An agentic system that selects tools, retrieves context, and synthesizes a response presents an explainability challenge that traditional systems do not. Reasoning traces, source citations, and confidence scores (Section VIII.A) are necessary but not sufficient. Enterprises deploying AI-native systems should publish system cards or model cards describing intended use, known limitations, and evaluation results; establish a clear chain of accountability for outputs (which team owns the agent, who reviews edge cases, who authorizes changes); and maintain audit trails sufficient to reconstruct any decision the system made on a specific input.

D. Security of Autonomous Action

Section IX.B noted prompt injection and tool misuse as engineering risks. The ethical framing is broader: an autonomous agent with access to enterprise systems creates a new class of insider threat, in which an external prompt or a compromised data source can cause the agent to act on the attacker's behalf. The principle of least privilege, well established in conventional security, acquires new urgency. Tool authorization should be scoped per-task, not per-agent; sensitive actions should remain on the HITL side of the spectrum even when faster alternatives exist; and irreversible actions should require explicit human confirmation regardless of agent confidence.

E. Societal and Workforce Impact

AI-native systems often replace or substantially restructure work that humans previously performed. The marketing-technology case study in Section VII.B describes a three-fold productivity increase for social media specialists; the implication is that some routine specialist work has shifted to the platform. Responsible deployment includes transparent communication with affected workers, retraining and role-evolution paths, and architectural choices that augment human judgment in high-stakes decisions rather than replacing it. Architects who participate in these decisions should engage with workforce, legal, and policy stakeholders early rather than presenting completed designs for review.

F. Limitations of This Paper

This paper is a practitioner-informed synthesis, not an empirical study. The four shifts, the reference architecture, and the maturity model are framings derived from observed practice and the recent literature; they have not been validated through controlled experiments at scale. The case studies in Section VII are author-reported (Cases A and B) or representative (Cases C and D); none constitutes audited benchmark data. The CapEx-to-OpEx maturation curve and the challenge-impact matrix are illustrative and based on the author's qualitative assessment. Where this paper makes quantitative claims, those claims should be treated as design-phase practitioner reports rather than measured outcomes; the methodology footnotes throughout the paper attempt to make this distinction explicit at every relevant point. Future work, as discussed in the Conclusion, should validate these framings through longitudinal measurement in production environments.

XIII. CONCLUSION

The evolution of system design in the AI era is not a simple substitution of one technology for another. It is a qualitative shift in how architects think about intelligence, data, and control flow. Traditional architectures are built around deterministic execution of human-written logic. AI-native architectures are built around reasoning systems that decide what to do next based on context, tools, and goals.

Four fundamental shifts define this transition: from static workflows to dynamic orchestration, from batch and streaming pipelines to continuous context, from stateless services to contextual memory, and from monolithic intelligence to specialized agents. Supporting these shifts requires new patterns (ReAct, Planning, Reflection, Multi-Agent Collaboration, HITL, HOTL), new protocols (MCP, A2A), new capability types (tools and skills), and new operational disciplines.

The case studies show that AI-native architecture delivers measurable improvements across domains as varied as financial intelligence, marketing, e-commerce, and healthcare. But the transition is not without risk. Hallucinations, security, governance, and cost are persistent concerns.

For practitioners and engineering leaders, the question is no longer whether to adopt AI-native architecture but how quickly and in what sequence. The architects who master these patterns will shape the next decade of enterprise software.

Future work will focus on three concrete directions. First, longitudinal empirical measurement of the four shifts in production deployments at scale—instrumented case studies that capture latency, accuracy, cost, and developer velocity across at least three years of operation, allowing the CapEx-to-OpEx maturation curve in Fig. 16 to be tested against real organizational data rather than directional patterns. Second, formalization of the agents-tools-skills capabilities registry as a reusable open standard, including a JSON-schema-based specification for skill packaging compatible with MCP server discovery, and reference

implementations across the major foundation-model providers. Third, deeper investigation of agent-to-agent trust and negotiation in multi-vendor ecosystems—the design of identity, reputation, and accountability primitives that allow A2A interactions to be auditable when agents from different organizations collaborate on shared workflows, where current protocols leave open how disputes are resolved and how cost and liability are allocated [15].

REFERENCES:

[1] C. F. Azubuko et al., "Microservices architecture in cloud-native applications: Design patterns and scalability," Computer Science & IT Research Journal, vol. 5, no. 9, pp. 2107-2124, Sep. 2024.
[2] P. Raj, S. Vanga, and A. Chaudhary, Cloud-Native Computing: How to Design, Develop, and Secure Microservices and Event-Driven Applications, 1st ed. Hoboken, NJ, USA: Wiley, 2022.
[3] L. Wang et al., "A survey on large language model based autonomous agents," Frontiers of Computer Science, vol. 18, no. 6, Art. no. 186345, Mar. 2024.
[4] J. Luo et al., "Large language model agent: A survey on methodology, applications and challenges," arXiv preprint, arXiv:2503.21460, Mar. 2025.
[5] D. Sculley et al., "Hidden technical debt in machine learning systems," in Proc. 28th Int. Conf. Neural Information Processing Systems (NIPS), Montreal, Canada, 2015, pp. 2503-2511.
[6] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," in Proc. 36th Int. Conf. Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 2022, pp. 24824-24837.
[7] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," in Proc. 11th Int. Conf. Learning Representations (ICLR), Kigali, Rwanda, 2023.
[8] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Proc. 34th Int. Conf. Neural Information Processing Systems (NeurIPS), Vancouver, Canada, 2020, pp. 9459-9474.
[9] Y. Gao et al., "Retrieval-augmented generation for large language models: A survey," arXiv preprint, arXiv:2312.10997, Mar. 2024.
[10] A. Yehudai et al., "Survey on evaluation of LLM-based agents," arXiv preprint, arXiv:2503.16416, Mar. 2025.
[11] X. Li, "A review of prominent paradigms for LLM-based agents: Tool use, planning (including RAG), and feedback learning," in Proc. 31st Int. Conf. Computational Linguistics (COLING), Abu Dhabi, UAE, Jan. 2025.
[12] L. Huang et al., "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions," ACM Trans. Inf. Syst., vol. 43, no. 2, Art. no. 42, Mar. 2025.
[13] K. Greshake et al., "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," in Proc. 16th ACM Workshop on Artificial Intelligence and Security (AISec '23), Copenhagen, Denmark, 2023, pp. 79-90.
[14] H. Li et al., "Large language model inference acceleration: A comprehensive hardware perspective," arXiv preprint, arXiv:2410.04466, Oct. 2024.
[15] X. Guo et al., "Large language model based multi-agents: A survey of progress and challenges," arXiv preprint, arXiv:2402.01680, Feb. 2024.
[16] J. Achiam et al., "GPT-4 technical report," arXiv preprint, arXiv:2303.08774, Mar. 2023.

[17] Anthropic, "Model Context Protocol specification (revision 2024-11-05)," Anthropic Developer Documentation, 2024. [Online]. Available: https://modelcontextprotocol.io/specification
[18] E. Wong et al., "A systematic review of key retrieval-augmented generation (RAG) systems: Progress, gaps, and future directions," arXiv preprint, arXiv:2507.18910, Jul. 2025.
[19] S. Gupta, R. Ranjan, and S. N. Singh, "A comprehensive survey of retrieval-augmented generation (RAG): Evolution, current landscape and future directions," arXiv preprint, arXiv:2410.12837, Oct. 2024.
[20] T. Kohli and V. Nair, "Microservices architecture in cloud computing: A software engineering perspective on design, deployment, and management," Int. J. Research and Innovation in Social Science, vol. 8, no. 7, pp. 2815-2828, Jul. 2024.