IP Protection and Trustworthiness in Agentic Systems for Industrial Operations

Pieter van Schalkwyk

CEO, XMPro

pieter.vanschalkwyk@xmpro.com

February 2, 2026

Abstract

As enterprises deploy agentic AI systems for operational decisions, protecting intellectual property and ensuring trustworthy operation become critical. This paper examines these challenges through the Industrial Internet Consortium's (now Industrial IoT Consortium) trustworthiness framework, comparing file-based and runtime-based architectures.

File-based systems store business logic in accessible files (Skills, prompts, configurations), exposing most proprietary knowledge in plain text that can be copied in minutes. They introduce security vulnerabilities through file system access, arbitrary command execution, and credential exposure, while making governance enforcement difficult.

Runtime-based systems separate agent configuration from business logic execution. By keeping business logic in compiled code and configurations in databases, these systems significantly reduce IP exposure, eliminate file system vulnerabilities, and enable true governance enforcement through runtime validation.

For production industrial environments, runtime-based architectures are essential for achieving the trustworthiness, security, and IP protection required for mission-critical operations.

Contents

1 Introduction

1.1 The Agentic Systems Revolution

The industrial and enterprise landscape is experiencing a fundamental shift in how operational decisions are made and executed. Traditional automation, whether robotic process automation (RPA), workflow engines, or rule-based systems, follows predefined scripts and workflows, requiring human experts to anticipate every scenario and code explicit responses. This approach breaks down when facing the complexity, variability, and uncertainty inherent in modern industrial operations.

Enter agentic systems: a new paradigm where autonomous software entities perceive their environment, make decisions, take actions, and achieve goals without requiring explicit programming for every scenario. Gartner defines an AI agent as "an autonomous or semiautonomous software entity that uses AI techniques to perceive, make decisions, take actions, and achieve goals in its digital or physical environment." More conversationally, an AI agent is software that executes tasks with enough agency to decide how to work toward a goal and the tools needed to act in its environment.

We are already seeing industrial-strength examples of this paradigm. Claude Code treats the developer's workstation as an action space, using tools like the terminal and file system to iteratively plan, write, run, and refine code. OpenAI's Codex extends this into multi-agent software delivery pipelines, where specialized agents write features, run tests, and propose pull requests as coordinated workflows. Google's Agentspace enables organizations to deploy domain-specific agents that combine reasoning, search, and enterprise data to execute multi-step business processes.

This shift from scripted automation to agentic intelligence represents a quantum leap in capability. Where traditional systems execute predefined workflows, agentic systems adapt to novel situations, learn from experience, and optimize toward objectives under conditions of uncertainty. Where RPA bots follow scripts, AI agents reason about problems, evaluate options, and make informed decisions. This capability is particularly valuable in industrial environments where operational decisions (maintenance scheduling, resource allocation, quality optimization, production planning) require expert judgment, contextual understanding, and the ability to balance competing objectives.

1.2 The Trustworthiness Imperative

As agentic systems move from experimental deployments to production-critical operations, trustworthiness becomes paramount. The Industrial Internet Consortium (IIC), now part of the Digital Twin Consortium as the Industrial IoT Consortium, has established a comprehensive trustworthiness framework built on five pillars: Security, Privacy, Safety, Reliability, and Resilience. These pillars are not merely aspirational goals but essential requirements for industrial AI systems that control physical processes, manage critical infrastructure, and make decisions affecting safety, quality, and business outcomes.

Security ensures protection against unauthorized access, malicious attacks, and data breaches.

In industrial environments, security breaches can lead to production disruptions, intellectual property theft, or even physical harm. Privacy protects sensitive operational and business data from unauthorized disclosure, critical for maintaining competitive advantage and regulatory compliance. Safety prevents harm to people, equipment, and the environment through fail-safe mechanisms, emergency protocols, and human oversight. Reliability ensures consistent, predictable behavior with guaranteed availability and fault tolerance. Resilience provides the ability to withstand and recover from disruptions, whether technical failures, cyber attacks, or operational anomalies.

For agentic systems operating in industrial environments, making autonomous decisions about equipment operation, resource allocation, or process optimization, these trustworthiness requirements are non-negotiable. A security breach could expose years of accumulated operational intelligence. A safety failure could result in equipment damage or human injury. A reliability issue could cause significant production disruption. The architecture of agentic systems must therefore be designed from the ground up to support trustworthiness, not as an afterthought but as a foundational principle.

1.3 Two Architectural Approaches

As the agentic systems market matures, two distinct architectural approaches have emerged, each with profound implications for IP protection, security, and governance.

File-based agentic systems store business logic, workflows, and domain expertise in accessible files: Markdown documents defining Skills, JavaScript source code implementing integrations, JSON configurations specifying system architecture, and text files containing prompts and domain knowledge. Examples include development tools like Claude Code with Skills, custom LLM agent frameworks, and prompt-based automation systems. These systems prioritize developer productivity, rapid iteration, and ease of customization, making them popular for development and experimentation.

Runtime-based agentic business process runtimes, in contrast, separate agent configuration from business logic execution. Agents are parametric configurations, defining objectives, constraints, and behaviors, that are loaded and executed by a protected runtime engine. Business logic, algorithms, and accumulated knowledge reside in compiled code or protected runtime environments, not in accessible files. Examples include enterprise agent platforms like XM-Pro MAGS, industrial automation systems, and production-grade multi-agent systems. These systems prioritize IP protection, security, governance enforcement, and production reliability, making them essential for mission-critical industrial deployments.

The choice between these approaches is not merely technical but fundamentally determines whether an organization can adequately protect its intellectual property, enforce governance policies, maintain security, and achieve the trustworthiness required for industrial operations. The remainder of this paper examines these architectural differences in detail, analyzes their implications for IP protection and trustworthiness, and provides practical guidance for enterprises navigating this critical decision.

2 IIC Trustworthiness Framework

The Industrial Internet Consortium (now part of the Digital Twin Consortium as the Industrial IoT Consortium) developed a comprehensive trustworthiness framework specifically designed for industrial systems.[1] This framework recognizes that industrial environments require a holistic approach beyond traditional IT security models. The framework is built on five interdependent pillars that together define what trustworthy industrial AI systems must achieve.

2.1 Security: Protection Against Threats

Security encompasses protection against unauthorized access, malicious attacks, and data breaches throughout the system lifecycle. For agentic systems, security must address multiple dimensions: protecting the agents themselves from compromise, securing the data they process and generate, ensuring secure communication between agents and with external systems, and preventing unauthorized modification of agent behavior or objectives.

Authentication and authorization: Every agent, user, and system component must be positively identified and granted only the minimum necessary permissions. For multi-agent systems, this includes agent-to-agent authentication, ensuring that consensus processes and collaborative decisions involve only authorized participants. Role-based access control (RBAC) must govern who can create, modify, or delete agents, change objectives, or access decision traces.

Data protection: All data at rest and in transit must be protected through encryption, access controls, and secure storage mechanisms. For agentic systems with shared memory and decision spaces, this includes protecting accumulated knowledge, historical decisions, and proprietary algorithms from unauthorized access. Sensitive operational data (such as production parameters, quality metrics, or maintenance schedules) must be encrypted and access-logged.

Secure execution environment: Agents must execute in controlled environments that prevent unauthorized operations. This includes eliminating unnecessary attack surfaces such as file system access, arbitrary command execution, or network access beyond defined integration points. Sandboxing and containerization provide isolation, ensuring that a compromised agent cannot affect other agents or the underlying infrastructure.

Vulnerability management: The system must be designed to minimize vulnerabilities and provide mechanisms for rapid patching and updates. For agentic systems, this includes secure update mechanisms that don't expose business logic, automated vulnerability scanning, and the ability to roll back to known-good configurations if issues are discovered.

2.2 Privacy: Protecting Sensitive Information

Privacy in industrial contexts extends beyond personal data to include proprietary operational information, trade secrets, and competitive intelligence. Agentic systems accumulate vast knowledge through continuous operation, learning optimal parameters, discovering efficiency patterns, and capturing expert decision-making processes. This accumulated intelligence represents competitive advantage and must be protected.

[1] Industrial Internet Consortium, *The Industrial Internet of Things Trustworthiness Framework Foundations*, `https://www.iiconsortium.org/pdf/Trustworthiness_Framework_Foundations.pdf`

Data minimization: Agents should collect and retain only the data necessary for their objectives, with clear retention policies and automated data lifecycle management. Decision traces should capture reasoning and outcomes without unnecessarily exposing sensitive operational details.

Access segregation: Different agents, teams, and users should have access only to the data necessary for their functions. Multi-agent systems must implement data segregation ensuring that agents working on different processes or for different business units cannot access each other's sensitive information unless explicitly authorized.

Audit and compliance: All access to sensitive data must be logged with complete audit trails showing who accessed what data, when, and for what purpose. This supports both security incident investigation and regulatory compliance requirements such as GDPR, CCPA, or industry-specific regulations.

2.3 Safety: Preventing Harm

Safety is paramount where agentic systems control physical processes or make decisions affecting human safety. The IIC framework emphasizes that safety must be designed into architecture, not added through policies.

Bounded autonomy: Agents must operate within explicitly defined boundaries that prevent unsafe operations. This includes hard constraints that cannot be violated (temperature limits, pressure thresholds, safety zones), policy constraints defining what agents must, must not, and may do, and escalation protocols that trigger human intervention when approaching boundaries.

Fail-safe mechanisms: The system must fail safely when errors occur, defaulting to safe states rather than continuing with potentially dangerous operations. For multi-agent systems, this includes consensus mechanisms that prevent individual agent failures from causing system-wide unsafe conditions.

Human oversight: Critical decisions must include human-in-the-loop approval workflows, with clear escalation protocols defining when human intervention is required. Agents must be able to explain their reasoning and provide sufficient context for humans to make informed override decisions.

2.4 Reliability and Resilience

Reliability means consistent, predictable behavior with guaranteed availability. For agentic systems making operational decisions, reliability encompasses both the determinism of mathematical optimization and the consistency of AI reasoning.

Deterministic optimization: Where possible, agents should use deterministic mathematical functions (objective and utility functions) that produce consistent results for the same inputs. This ensures that decisions are repeatable, explainable, and auditable, critical for industrial environments where variability can indicate problems.

Availability guarantees: Production agentic systems must provide high availability with automatic failover, redundancy, and recovery mechanisms. For operations that run continuously,

downtime represents significant business impact, making availability non-negotiable.

Resilience is the ability to withstand and recover from disruptions. For agentic systems, this means self-healing capabilities, fault tolerance where individual agent failures don't cause system-wide disruptions, adaptive recovery that learns from failures, and distributed resilience across agents rather than centralized.

3 File-Based Agentic Systems: Architecture and Risks

File-based agentic systems have gained popularity in the development community due to their accessibility, rapid iteration capabilities, and ease of customization. These systems store agent behavior, business logic, and domain expertise in human-readable files that developers can easily create, modify, and share. While this approach accelerates development and experimentation, it introduces significant risks when deployed in production industrial environments, particularly regarding IP protection, security, and governance enforcement.

3.1 Architecture Overview

File-based systems typically consist of several components that together define agent behavior and capabilities:

Skills are Markdown or text files that define agent workflows, decision logic, and operational procedures. These files contain the business logic that determines how agents respond to situations, make decisions, and execute tasks. A typical enterprise deployment might have many Skills representing years of accumulated operational expertise.

MCP (Model Context Protocol) servers are source code files, typically JavaScript, TypeScript, or Python, that implement integrations with external systems, databases, and APIs. These servers contain custom integration logic, data transformation algorithms, and system-specific protocols that enable agents to interact with industrial systems, enterprise applications, and data sources.

Prompts and system instructions are text files that provide agents with domain expertise, decision frameworks, and operational knowledge. These prompts encode years of expert knowledge, proprietary methodologies, and competitive intelligence that guide agent reasoning and decision-making.

Configuration files in JSON, YAML, or similar formats specify system architecture, API endpoints, credentials, and operational parameters. These files define how components connect, what resources agents can access, and how the system is deployed.

All of these components reside in accessible file systems, developer laptops, Git repositories, file shares, or cloud storage, where they can be read, modified, and copied by anyone with appropriate file system access. This accessibility, while beneficial for development, creates critical vulnerabilities in production environments.

3.2 IP Protection Challenges

The file-based architecture creates severe intellectual property exposure that enterprises must carefully consider before production deployment.

In file-based systems, virtually all intellectual property resides in accessible files. A typical enterprise deployment might have many Skills containing complete business logic for operational decision-making, representing years of development and refinement. Numerous MCP servers with custom integration code and proprietary algorithms. Extensive prompts encoding domain expertise and competitive intelligence. Dozens of configuration files documenting system architecture and integration patterns.

Ease of unauthorized copying: The mechanics of IP theft from file-based systems are trivially simple. An employee with legitimate access to the file system can copy all Skills to external storage, clone the MCP server repository, and export prompts and configurations, all in minutes. The copied files contain everything needed to recreate the system at a competitor: complete business logic, integration code, domain expertise, and system architecture.

Unlike stealing physical assets or accessing protected databases (which leave audit trails and trigger alerts), file copying appears as normal developer activity and may go undetected. Once copied, the IP can be recreated at a competitor very quickly, simply installing the development environment, copying the files, and configuring connections. The original company loses years of competitive advantage overnight, with limited legal recourse since files are "just text" and proving trade secret status is difficult.

Legal recourse difficulties: Protecting file-based IP through legal means faces significant challenges. Skills and prompts, being text documents, are difficult to protect through copyright (which protects expression, not ideas). Trade secret protection requires proving that reasonable steps were taken to maintain secrecy, difficult when files are accessible to all developers. Non-disclosure agreements and non-compete clauses provide some protection but are expensive to enforce, vary by jurisdiction, and often fail to prevent IP loss.

Consider a manufacturing company that has invested substantially over multiple years building an extensive Skills library for predictive maintenance, quality optimization, and production scheduling. A senior engineer with access to all files accepts a position at a competitor. On their last day, they copy the entire Skills directory, all MCP servers, and configuration files to a personal cloud storage account, an operation that appears as normal file access.

Shortly afterward, the competitor has replicated the company's entire operational intelligence capability. The original company has lost its investment and years of competitive advantage. Legal action is initiated but faces challenges proving trade secret status of "text files." Even if successful, the damage is done, the competitor already has the capability.

3.3 Security Vulnerabilities

File-based architectures inherit significant security vulnerabilities from their dependence on file system access and execution of user-provided content.

File system access creates multiple attack vectors: path traversal attacks enabling access

to sensitive system files, unauthorized modification of Skills or configuration files, injection of malicious content into agent instructions, and privilege escalation through file permission manipulation.

Arbitrary code execution introduces severe risks when systems execute user-provided code (JavaScript, Python, shell scripts): malicious code injected into Skills or integration scripts, command injection through unsanitized inputs, resource exhaustion through infinite loops or memory leaks, and lateral movement to other systems through executed code.

Credential exposure occurs when file-based systems store credentials and API keys in configuration files or environment variables: plain text credentials in configuration files, API keys embedded in Skills or prompts, connection strings exposing database access, and service account credentials in environment configurations.

Prompt injection becomes possible since business logic exists as text prompts: attackers can inject malicious instructions through carefully crafted inputs, override safety constraints, modify agent objectives or decision criteria, extract sensitive information through social engineering of LLMs, and bypass governance policies through prompt manipulation.

3.4 Governance Challenges

When governance policies exist as text instructions in Skills or prompts, enforcement becomes problematic. Policies can be edited by anyone with file access. LLMs may interpret policies inconsistently. Prompt injection can override policy instructions. There's no runtime validation ensuring compliance, governance relies on correct file contents and agent cooperation.

Configuration drift occurs as files diverge from approved configurations over time. Local modifications accumulate without oversight. Version control may not be enforced. Testing environments drift from production. No audit trail tracks unauthorized changes.

Since business logic executes by interpreting files, there's no mechanism to prevent policy violations at runtime. Governance relies on correct file contents. Modifications bypass approval workflows. Compliance cannot be guaranteed. The system trusts agents to self-govern based on instructions they read from files.

4 Runtime-Based Agentic Systems: Enhanced Security and Governance

Runtime-based agentic business process runtimes represent a fundamentally different architectural approach that addresses the IP protection, security, and governance challenges inherent in file-based systems. By separating agent configuration from business logic execution and enforcing policies at runtime rather than through file-based instructions, these systems provide the trustworthiness required for production industrial deployments.

4.1 Architecture Overview

Runtime-based systems employ a layered architecture that separates concerns and protects critical intellectual property:

At the foundation is a **compiled runtime engine**, typically implemented in languages like C#, Java, or Go, that contains the core business logic, algorithms, and operational intelligence. This runtime includes consensus protocols, self-healing mechanisms, mathematical optimization functions, and coordination logic that represent years of development and refinement. Because this logic is compiled into binary executables rather than stored in accessible source files, it cannot be easily copied or reverse-engineered.

Parametric agent configurations define individual agent behavior through structured data, typically stored in databases rather than files. These configurations specify agent objectives, utility functions, policy constraints, tool access, and behavioral rules, but contain no business logic or algorithms. An agent configuration might specify "optimize equipment uptime using exponential utility function with steepness 3.5" without containing the actual optimization algorithm or utility calculation code.

Database-backed state and knowledge stores agent memory, decision history, accumulated knowledge, and operational data in multi-database architectures. Graph databases capture relationships and structure, time-series databases store temporal data and decision traces, vector databases enable semantic search, and optional triple stores support semantic reasoning. This knowledge is access-controlled, encrypted, and audit-logged, preventing unauthorized access or copying.

API-driven configuration management provides the interface for creating, modifying, and deploying agents. All configuration changes go through authenticated, validated APIs that enforce access control, validate inputs, log changes, and maintain version history. There is no direct file system access, all operations are mediated through controlled APIs that can enforce governance policies and maintain audit trails.

This architecture provides multiple layers of protection: business logic in compiled code, configurations in access-controlled databases, and all operations mediated through validated APIs. An employee with access to agent configurations can see what objectives agents pursue and what constraints they operate under, but cannot access the algorithms, business logic, or accumulated knowledge that represent the core intellectual property.

4.2 IP Protection Advantages

The runtime-based architecture provides substantially better IP protection than file-based alternatives, reducing exposure from most IP to a minority.

Reduced IP exposure: In runtime-based systems, the most valuable intellectual property, business logic, algorithms, and operational intelligence, resides in protected components that cannot be easily accessed or copied.

Business logic in compiled code means that consensus algorithms, planning strategies, self-healing mechanisms, and optimization logic are compiled into binary executables. While not

impossible to reverse-engineer, this requires significant expertise, time, and effort compared to simply copying source files. Most importantly, it creates a clear legal boundary: accessing compiled code requires unauthorized access to production systems, which is unambiguously illegal.

Algorithms protected in runtime ensures that proprietary mathematical functions, utility calculations, significance scoring, and objective optimization remain inaccessible. An employee can see that an agent uses "exponential utility function" but cannot access the actual implementation, parameters, or refinements that make it effective.

Knowledge in databases means that accumulated operational intelligence, learned patterns, historical decisions, performance data, is stored in access-controlled databases rather than files. Exporting this knowledge requires database access (which is logged and monitored), produces large data volumes (making exfiltration difficult), and leaves clear audit trails.

Only configurations exposed means that the minority of IP that is accessible consists of agent profiles, objective function formulas, and policy constraints, valuable information, but insufficient to recreate the system. An employee who copies configurations still lacks the runtime engine, business logic, algorithms, and accumulated knowledge needed to build a competing system.

Technical barriers to copying: Runtime-based systems create multiple technical barriers that make IP theft significantly more difficult and risky.

Cannot simply copy files because the core IP isn't in files, it's in compiled code and databases. An employee cannot simply copy a directory to external storage and walk out with complete IP. They would need to export databases (requiring elevated privileges and generating audit logs), access production systems (unauthorized and illegal), or reverse-engineer compiled code (time-consuming and difficult).

Requires database export for configurations and knowledge, which is a logged, monitored operation requiring administrative privileges. Database exports generate large files, take significant time, and trigger monitoring alerts. Unlike file copying which appears as normal developer activity, database exports are unusual operations that security teams can detect and investigate.

Compiled code difficult to reverse engineer provides a significant technical barrier. While not impossible, reverse engineering requires specialized skills, dedicated tools, and substantial time investment. More importantly, it produces code that is legally distinct from the original, making it harder to prove theft while still requiring substantial work to understand and replicate.

Stronger legal protection: Runtime-based architecture provides clearer legal boundaries and stronger protection mechanisms.

Compiled code equals trade secret protection because courts recognize compiled software as protectable trade secrets when reasonable security measures are in place. Access controls, encryption, and audit trails demonstrate that the company took reasonable steps to protect the IP, strengthening legal claims.

Database access equals unauthorized access, which is clearly illegal under computer fraud and abuse laws. An employee who exports production databases without authorization has com-

mitted a crime, providing clear legal recourse. This is much stronger than trying to prove that copying text files constitutes trade secret theft.

Real-world protection: Consider the same manufacturing company scenario, but with a runtime-based system. The senior engineer has access to agent configurations in the database but not to the compiled runtime code or accumulated knowledge. On their last day, they export agent configurations, which generates audit logs.

What they have: Agent profiles, objective function formulas, policy constraints, representing a minority of the IP. What they don't have: The runtime engine (compiled code), consensus algorithms, self-healing logic, mathematical optimization implementation, or accumulated operational knowledge. To recreate the system at a competitor, they would need to rebuild the entire runtime engine, reimplement all algorithms and business logic, recreate the multi-database architecture, and rebuild accumulated knowledge through operational experience, representing many months of work and substantial investment.

The time and cost barriers are so significant that most employees won't attempt it, and competitors will find it more economical to build their own system from scratch. Meanwhile, the database export generated audit logs that security teams can investigate, and the company has clear legal recourse for unauthorized database access.

4.3 Security Advantages

Runtime-based architecture eliminates entire classes of security vulnerabilities that plague file-based systems.

Eliminated attack surfaces: No file system access means agents cannot read, write, or execute files. All data access is through database APIs, all configuration is through validated APIs, and all operations are through controlled interfaces. This eliminates file manipulation attacks, path traversal vulnerabilities, and file injection exploits.

No arbitrary command execution because agents don't have shell access. Tool execution is through controlled interfaces that validate inputs, sanitize parameters, and restrict operations to predefined capabilities. An agent can "query a database" or "adjust a setpoint" through controlled tools, but cannot execute arbitrary shell commands.

No shell access as agents run in containerized environments without shell access. Even if an agent is compromised, the attacker cannot execute commands, install backdoors, or perform lateral movement. The container provides isolation and limits the blast radius of any compromise.

Secure credential management: Credentials in encrypted database rather than configuration files means that API keys, passwords, and tokens are stored in encrypted database fields with access controls and audit logging. Credentials are never exposed in plain text files or environment variables.

Secure key management through integration with enterprise key management systems (KMS) or hardware security modules (HSM) provides additional protection. Credentials can be rotated automatically, access can be revoked instantly, and all credential usage is logged for audit.

Runtime validation: Perhaps the most significant security advantage is runtime validation of

all agent actions.

All actions validated against policies means that before any agent action executes, the runtime checks it against policy constraints (must/must not/may/must if rules), objective thresholds (critical/warning/target levels), and governance frameworks. Actions that violate policies are blocked automatically, regardless of what the agent "wants" to do.

Input sanitization and validation occurs at the runtime level, not just in agent code. All inputs are validated against expected schemas, sanitized to remove malicious content, and checked for injection attacks before being processed.

Boundary checking ensures agents operate within defined limits. If an agent attempts to allocate resources beyond budget, adjust parameters outside safe ranges, or take actions beyond its authority, the runtime blocks the action and triggers escalation protocols.

4.4 Governance Advantages

Runtime-based architecture enables true governance enforcement rather than relying on honor systems or policy documentation.

Runtime enforcement: Policies enforced at execution time means that governance is not optional or advisory, it's architecturally enforced. When an agent attempts an action, the runtime validates it against all applicable policies before allowing execution. This is fundamentally different from file-based systems where policies are instructions that agents are expected to follow.

Not "honor system" but actual validation ensures that even compromised, misconfigured, or malfunctioning agents cannot violate governance policies. The runtime acts as an independent enforcement layer that doesn't trust agents to self-govern.

Cannot bypass governance because the enforcement is in the runtime, not in agent code. An attacker who compromises an agent cannot simply remove policy checks or bypass constraints, the runtime validates every action regardless of agent state.

Complete audit trails: All actions logged automatically by the runtime, not by agent code. Every decision, action, configuration change, and access is logged with complete context: who (which agent or user), what (action taken), when (timestamp), why (reasoning and confidence), and outcome (success or failure).

Database-backed audit records provide structured, queryable audit data rather than unstructured log files. Audit records can be analyzed for patterns, compliance reporting, security investigation, or performance analysis using standard database queries.

Immutable logging ensures that audit records cannot be tampered with or deleted. Once written to the audit database, records are permanent and protected, supporting forensic investigation and regulatory compliance.

5 Comparative Analysis

Table 1 summarizes IP protection differences between architectures.

Table 1: IP Protection Comparison

Dimension	**File-Based**	**Runtime-Based**
Business Logic Location	Plain text files	Compiled code
Exposure Level	Most IP accessible	Minority accessible
Time to Copy	Minutes	N/A (requires reverse engineering)
Detectability	Low	High
Legal Protection	Weak	Strong
Recreation Difficulty	Very easy	Very difficult

5.1 Security Comparison

Runtime-based systems eliminate entire classes of vulnerabilities present in file-based architectures.

Table 2: Security Comparison

Vulnerability	**File-Based**	**Runtime-Based**
File System Access	High risk	Eliminated
Command Execution	High risk	Controlled only
Credential Exposure	Configuration files	Encrypted database
Prompt Injection	Significant risk	Minimal risk
Attack Surface	Extensive	Minimal

5.2 Governance Comparison

Governance enforcement differs fundamentally between the two architectures.

Table 3: Governance Comparison

Capability	**File-Based**	**Runtime-Based**
Policy Enforcement	Honor system	Runtime validation
Bypass Prevention	Not possible	Architecturally prevented
Audit Trails	Incomplete	Comprehensive
Version Control	Manual	Automatic
Access Control	File permissions	Multi-layer

5.3 IIC Framework Alignment

The following table shows how each architecture aligns with the five pillars of the IIC trustworthiness framework.

Table 4: IIC Trustworthiness Framework Alignment

Pillar	File-Based	Runtime-Based
Security	Limited	Robust
Privacy	Weak	Strong
Safety	Honor system	Enforced
Reliability	Variable	Consistent
Resilience	Limited	Robust

File-based systems struggle to meet IIC requirements across all five pillars, while runtime-based systems provide architectural support for trustworthiness.

5.4 Use Case Suitability

File-Based Systems Are Suitable For:

- Development and prototyping where rapid iteration is critical
- Internal tools with low IP value and minimal security requirements
- Experimental deployments where learning is the priority
- Non-critical automation where failures have minimal impact

Runtime-Based Systems Are Essential For:

- Production industrial operations where trustworthiness is critical
- Mission-critical decisions affecting safety, quality, or business outcomes
- High-value IP representing significant R&D investment
- Regulated industries requiring compliance and audit trails
- Competitive advantage where IP protection is business-critical

6 Case Study: XMPro MAGS Implementation

To illustrate runtime-based principles in practice, we examine XMPro Multi-Agent Generative Systems (MAGS),[2] a production-grade industrial agentic platform built specifically for safety-critical operational environments.

6.1 Architecture Overview

XMPro MAGS embodies key runtime-based design principles that separate configuration from execution and enforce governance at the architectural level.

Agent as parametric configuration: In MAGS, an agent is not a file containing business logic but a database-backed configuration specifying objectives (what the agent is responsible for achieving), constraints (boundaries within which the agent must operate), permissions (what

[2]XMPro MAGS architecture and documentation: `https://github.com/XMPro/Multi-Agent/tree/main`

actions the agent is authorized to take), integrations (what data sources and systems the agent can access), and triggers (what conditions activate agent reasoning).

The actual decision-making logic, how agents perceive situations, evaluate options, and select actions, resides in the MAGS runtime engine, protected as compiled code. This separation is fundamental: configurations describe *what* agents do, but the runtime controls *how* they do it.

APEX cognitive cycle: The MAGS runtime implements the APEX cognitive cycle (Observe, Reflect, Plan, Act) as compiled functionality:

Observe: Agents monitor operational data streams, receiving events from connected systems (sensors, historians, work orders, quality systems). The observation logic, what data is relevant, how it should be interpreted, what patterns indicate issues, is encoded in the runtime, not in agent configurations.

Reflect: Agents analyze observed conditions against historical patterns, decision traces, and domain knowledge. The reflection algorithms, how to evaluate situations, assess risks, identify opportunities, are protected runtime implementations.

Plan: Agents evaluate potential actions, considering constraints, costs, risks, and expected outcomes. The planning logic, how to generate options, evaluate trade-offs, select optimal actions, resides in runtime code.

Act: Agents execute selected actions through controlled integration points. The action execution framework, how to safely interface with operational systems, validate results, handle errors, is protected runtime functionality.

DecisionGraph for operational memory: MAGS implements decision traces through DecisionGraph, a graph-based memory system capturing what situations agents encountered (context nodes), what data was available (observation nodes), what options were considered (plan nodes), what actions were taken (act nodes), what outcomes resulted (result nodes), and relationships between these elements (edges).

DecisionGraph resides in a protected database with granular access controls, encryption, and audit logging. Even users with configuration access cannot directly access or exfiltrate decision traces without explicit authorization.

6.2 IP Protection Implementation

MAGS architecture provides multiple layers of IP protection that demonstrate how runtime-based principles work in practice.

Runtime logic protection: The core MAGS runtime, containing APEX implementation, decision algorithms, integration frameworks, and governance engines, is distributed as compiled code. Reverse engineering would require decompilation of protected binaries, understanding of complex multi-agent orchestration, recreation of decision-making algorithms, and rebuilding integration frameworks, representing many months of work.

Configuration abstraction: Agent configurations in MAGS are parametric descriptions (objectives, thresholds, integration endpoints) rather than executable business logic. Accessing configurations reveals what agents do but not how MAGS achieves those objectives. The valuable

knowledge, decision algorithms, optimization strategies, integration patterns, remains protected in the runtime.

An example illustrates the difference: A file-based system might have a Skill containing complete Python code implementing a predictive maintenance algorithm. MAGS instead has a configuration specifying "use degradation model with exponential parameters, trigger maintenance at 85% confidence, optimize for cost-availability trade-off." The configuration reveals the objective but not the implementation.

Decision trace security: DecisionGraph implements comprehensive security for accumulated operational intelligence through database-level encryption of decision traces, role-based access control requiring explicit authorization, audit logging of all trace queries and exports, and anonymization options for sensitive operational details.

Quantified protection: An employee with database access could export agent configurations, generating audit logs. What they would have: formulas, parameters, constraints. What they would not have: the runtime engine (substantial compiled code), consensus algorithms, self-healing logic, mathematical optimization implementation, or accumulated operational knowledge. To recreate MAGS at a competitor would require substantial development time and investment, a significant deterrent compared to the minutes needed to copy file-based systems.

6.3 Security Implementation

MAGS security architecture addresses the IIC trustworthiness framework through defense-in-depth principles.

Secure execution environment: Agents execute in controlled runtime environments with no direct file system access, restricted network access (whitelist of authorized endpoints), no arbitrary command execution, and resource limits preventing denial-of-service.

Integration security: MAGS connects to operational systems through secure integration frameworks using pre-built, security-audited connectors, centralized credential management (Azure Key Vault, AWS Secrets Manager), input validation and sanitization, and rate limiting and anomaly detection.

Multi-layer access control: MAGS implements defense-in-depth through application-level authentication (SSO, SAML, OAuth), role-based permissions (who can create, modify, or run agents), database-level access controls (table and row-level security), and network segmentation (infrastructure-level isolation).

6.4 Governance Implementation

MAGS enables true governance through architectural enforcement rather than policy documents or agent instructions.

Deontic framework: The MAGS runtime implements deontic logic encoding obligations (actions agents must take, such as alerting on critical conditions or seeking approval for high-risk actions), permissions (actions agents may take, such as starting equipment or adjusting parameters within ranges), and prohibitions (actions agents cannot take, such as exceeding safety limits

or bypassing approval requirements).

These policies are encoded in runtime code and validated at every action execution. Agents cannot override or bypass governance policies regardless of their reasoning or confidence.

Human-in-the-loop enforcement: For safety-critical decisions, MAGS enforces human approval requirements. Agents can recommend actions but cannot execute without approval. Approval requirements are defined in agent configurations. The runtime enforces approval workflows. Audit trails capture who approved what and why.

Progressive autonomy: MAGS supports progressive autonomy where agents gain additional permissions as they demonstrate reliability. Initial deployment with human approval for all actions. Gradual increase in autonomy based on performance. Automatic reversion to supervised mode if performance degrades. Permanent safety constraints regardless of autonomy level.

Standards alignment: MAGS implements industrial standards for trustworthiness and interoperability. Decision traces use W3C Provenance Ontology (PROV-O) to capture what entities were involved, what activities occurred, how entities influenced each other, and temporal relationships. MAGS integrates with Asset Administration Shell (AAS) for industrial asset management and supports ISO 14224 for collecting and analyzing reliability data.

6.5 Results and Benefits

MAGS deployments in production industrial environments demonstrate the benefits of runtime-based architecture in practice.

Organizations report confidence deploying proprietary operational knowledge into MAGS due to IP protection guarantees. Unlike file-based systems where contractors or temporary employees could copy Skills files, MAGS configurations reveal objectives without exposing decision algorithms or accumulated intelligence.

Security audits validate MAGS architecture against enterprise security requirements. The controlled execution environment, secure integration layer, and defense-in-depth access controls meet stringent requirements for production deployment.

Operations leaders trust MAGS to enforce governance policies because validation occurs at runtime, not through LLM interpretation of text prompts. Safety limits cannot be exceeded, approval requirements cannot be bypassed, and policy violations are prevented rather than detected after the fact.

MAGS decision traces provide the auditability required for regulatory compliance. Complete histories of what agents observed, what they decided, why they decided it, and what resulted enable forensic analysis and regulatory reporting.

7 Enterprise Recommendations

Based on our analysis of IP protection, security, and governance challenges in agentic systems, we provide practical recommendations for enterprises at different stages of agentic system adoption.

7.1 For Companies Using File-Based Systems

If your organization has invested in file-based systems, immediate risk mitigation is essential while planning longer-term architectural evolution.

Immediate Actions (This Week):

Conduct IP audit, inventory all Skills, MCP servers, prompts, and configurations to understand what intellectual property exists in files. Classify each component by IP value: Critical (proprietary algorithms, competitive advantage, years of development), Moderate (internal processes, some value but not unique), or Low (generic workflows, minimal proprietary value).

Assess security vulnerabilities, review all Skills and MCP servers for security issues including file system access, arbitrary command execution, credential exposure, or insufficient input validation. Identify which components have access to sensitive systems, production databases, or critical infrastructure.

Implement strict access controls, immediately restrict file system access to Skills and MCP servers using operating system permissions. Enable audit logging for all file access to detect unauthorized copying or suspicious patterns. Implement data loss prevention (DLP) tools to prevent bulk file copying to external storage, cloud services, or email.

Secure credentials, remove all plain text credentials from configuration files and environment variables. Migrate to secure secret management systems (HashiCorp Vault, AWS Secrets Manager, Azure Key Vault) that provide encryption, access control, rotation, and audit logging. Rotate all exposed credentials immediately to eliminate vulnerability windows.

Short-Term Actions (Months):

Move critical IP to protected services, identify the highest-value Skills with proprietary algorithms or competitive advantage logic and refactor them to call protected services rather than containing logic directly. For example, instead of a Skill containing a proprietary optimization algorithm, have it call an API endpoint where the algorithm resides in compiled code. This reduces IP exposure while maintaining Skill functionality.

Implement encryption for repositories, encrypt Skills and MCP server repositories using tools like git-crypt or similar encryption solutions. Ensure encryption keys are managed securely and access is logged. While this doesn't prevent authorized users from accessing files, it protects against unauthorized access and makes theft more difficult.

Establish rigorous version control, if not already in place, implement version control with branch protection, code review requirements, and approval workflows for changes to production Skills and MCP servers. This provides change tracking, rollback capability, and governance over modifications.

Deploy comprehensive monitoring, implement file access monitoring using tools like auditd (Linux), File Integrity Monitoring (FIM), or Security Information and Event Management (SIEM) systems. Alert on suspicious patterns including bulk file copying, access from unusual locations, or access outside normal working hours.

Long-Term Strategy:

Plan migration to runtime-based architecture for production deployments. This doesn't mean abandoning file-based systems entirely, they remain valuable for development and experimentation, but production systems should use runtime-based architecture for IP protection and security.

Implement a hybrid development-to-production pipeline where development occurs in file-based systems (rapid iteration, experimentation), production deployment uses runtime-based systems (IP protection, security, governance), and migration paths preserve learnings while adding protection.

7.2 For Companies Evaluating Agentic Systems

Organizations beginning their agentic systems journey should make informed architectural decisions from the start.

Evaluation Criteria:

Understand where business logic resides, does it sit in accessible files or protected runtime? How are agent configurations stored and protected? What execution environment do agents run in? How are integrations secured? Where are credentials stored and managed? How are governance policies actually enforced?

Assess vendor architecture, request detailed architectural documentation. Understand the separation between configuration and execution. Ask about IP protection mechanisms. Review security audit reports. Examine governance enforcement approaches.

Consider use case requirements, what IP value will the system contain? What security requirements apply? What governance needs exist? What compliance obligations must be met? What operational criticality does the system have?

Questions to Ask Vendors:

Where does your business logic reside, in files or in protected runtime? How do you protect proprietary algorithms and accumulated knowledge? What prevents an employee from copying all business logic? How quickly could a competitor recreate our system if they obtained your configurations?

What security measures prevent unauthorized access? How do you eliminate file system and command execution vulnerabilities? How are credentials protected? What penetration testing has been performed?

How do you enforce governance policies? Can agents bypass constraints through prompt injection or other means? What audit trails do you maintain? How do you support regulatory compliance?

Decision Framework:

For development and prototyping: File-based systems are acceptable. Rapid iteration and learning are priorities. IP value is low. Security requirements are moderate. No connection to production systems.

For internal tools: File-based systems may be acceptable with security measures. IP value is low-to-moderate. Security requirements are moderate. Comprehensive access controls and

monitoring are essential.

For production operations (non-critical): Runtime-based systems are preferred. IP value is moderate. Security requirements are high. Governance needs are significant. Consider file-based only with extensive security architecture.

For production operations (critical): Runtime-based systems are required. IP value is high. Security requirements are very high. Governance needs are very high. File-based systems cannot provide necessary trustworthiness.

For regulated industries: Runtime-based systems are required. Compliance obligations demand demonstrable governance and comprehensive audit trails. File-based systems typically cannot meet regulatory requirements for auditability and control.

7.3 Best Practices for Any Deployment

Regardless of architectural choice, certain best practices apply to all agentic system deployments.

IP Protection:

Use appropriate architecture for IP value, runtime-based for high-value IP, file-based acceptable only for low-value scenarios. Implement defense-in-depth with multiple protection layers. Maintain comprehensive audit trails of all access and modifications. Update legal protections including employment agreements, trade secret documentation, and exit procedures.

Security:

Minimize attack surfaces, eliminate unnecessary file system access, command execution capabilities, and network access. Use secure credential management, never store credentials in plain text. Implement runtime validation of all agent actions against security policies. Deploy comprehensive monitoring, alerting, and incident response capabilities.

Governance:

Encode policies in runtime, not in prompts or configuration files that agents interpret. Require human approval for safety-critical decisions with clear escalation protocols. Implement progressive autonomy where agents gain permissions gradually based on demonstrated reliability. Maintain complete audit trails for compliance, forensics, and continuous improvement.

Operational Excellence:

Start with clear objectives and success criteria. Implement comprehensive monitoring of agent behavior, decision quality, and system health. Establish feedback loops for continuous improvement. Plan for human oversight and intervention capabilities. Document decisions, rationale, and lessons learned.

8 Conclusion

The choice between file-based and runtime-based agentic architectures is not merely technical but strategic.

8.1 Key Findings

File-based systems excel in development scenarios where rapid iteration outweighs IP protection concerns. However, they fundamentally expose proprietary business logic in accessible files, create significant security vulnerabilities, and cannot enforce governance policies at runtime. For production industrial environments, these limitations are disqualifying.

Runtime-based systems provide superior IP protection by separating parametric configurations from protected business logic, enhanced security through controlled execution environments, and true governance enforcement through runtime validation. While requiring greater initial investment, these systems deliver the trustworthiness essential for mission-critical operations.

8.2 Implications

The Industrial Internet Consortium's trustworthiness framework provides essential guidance for industrial agentic systems. Our analysis demonstrates that runtime-based architectures align naturally with this framework, while file-based systems struggle to meet trustworthiness requirements in production environments.

For organizations deploying agentic systems:

- Trustworthiness is not optional but essential
- Architecture determines achievable trustworthiness levels
- Production deployment requires runtime-based architecture
- File-based systems remain valuable for development, not operations

8.3 Strategic Recommendations

Organizations should adopt a pragmatic, risk-based approach:

For development and learning: File-based systems offer valuable hands-on experience and rapid iteration, enabling teams to explore agentic capabilities and identify promising use cases.

For production deployment: Runtime-based architectures are essential, particularly for safety-critical operations, high-IP-value applications, and regulated industries.

For organizational success: Treat architecture selection as a strategic decision requiring input from security, legal, operations, and executive leadership, not just a technical choice made by development teams.

8.4 Future Directions

As agentic systems mature, we anticipate development of industry-specific standards building on IIC trustworthiness framework, increased regulatory attention to autonomous decision-making requiring demonstrable governance, and evolution of hybrid approaches combining file-based flexibility for development with runtime-based security for production.

Organizations investing in agentic systems today are building operational intelligence that will drive competitive advantage for years. Protecting this intelligence through appropriate archi-

tecture is not just prudent risk management but strategic imperative for long-term success.

About XMPro

XMPro provides Multi-Agent Generative Systems (MAGS) for industrial operations, enabling organizations to deploy trustworthy, production-grade agentic systems in safety-critical environments. Built on runtime-based architecture principles and aligned with IIC trustworthiness framework, XMPro MAGS delivers the IP protection, security, governance, and reliability required for mission-critical operations.

For more information, visit `https://xmpro.com` or contact info@xmpro.com.