From LLM to Agentic AI: A Practical Guide to the Terms That Actually Matter

Published by AI² – Association for Industrial AI

Vlad Larichev — Industrial AI Lead at Accenture; Co-Founder, AI² – Association for Industrial AI
Alexey Samoshilov — Industrial Services Sales & Strategy, Siemens

May 2026

Abstract

Industrial AI discourse overloads terms such as LLM, AI agent, agentic, RAG, and multi-agent, yet the underlying concepts differ in ways that directly affect procurement, architecture, and governance. Recent work on deployments shows that semantic confusion across these layers drives mispriced investments, weak governance, and risky shop-floor rollouts (Bluethgen, 2025; Freeman et al., 2026; Osmond & Jego, 2026).

This guide presents a vendor-neutral, domain-driven vocabulary structured in five layers (A–E): foundation models and LLMs; retrieval and grounding; agents and multi-agent frameworks; agentic AI and agentic systems; and orchestration, governance, and economics. For each layer it states technical meanings, how layers compose, and practical implications for industrial and manufacturing contexts—including procurement reviews and regulatory alignment.

The aim is not hype or vendor framing, but operational clarity: which layer a product actually occupies, what must be true for production use, and how terminology maps to engineering and organizational decisions.

0.1 The Five-Layer Framework

Before diving into individual terms, it helps to see the full picture. The terminology in modern AI follows a natural progression, where each layer builds on the one below it and adds both capability and complexity:

- **Layer A - Foundation Models & LLMs:** The base models that provide language understanding and generation
- **Layer B - Retrieval & Grounding:** Mechanisms that connect model outputs to verifiable, domain-specific knowledge
- **Layer C - Agents & Multi-Agent Frameworks:** Autonomous components that observe, reason, and act via tools
- **Layer D - Agentic AI & Agentic Systems:** Orchestrated architectures where specialized agents coordinate to accomplish complex goals
- **Layer E - Orchestration, Governance & Economics:** The cross-cutting concerns that make everything production-ready

Each step up this ladder adds architectural complexity, governance needs, and organizational impact (Bhattacharya, 2025; Bluethgen, 2025; Freeman et al., 2026). Understanding which layer a product or solution actually operates at is the single most important question in any AI procurement or architecture review.

One critical caveat upfront: terminology is domain-contingent. Labels like "RAG," "agent," or "multi-agent" carry different architectural implications in manufacturing than they do in software engineering, healthcare, or aerospace. A "multi-agent system" in a chemical plant - where autonomous decisions can have physical safety consequences - is a fundamentally different engineering challenge than a multi-agent system that writes code. This guide is written with industrial and manufacturing contexts in mind.

1 Layer A: Foundation Models & LLMs

1.1 Foundation Model

A **foundation model** is a large neural network trained on broad, diverse data at massive scale. The term was introduced by Stanford's Center for Research on Foundation Models (CRFM) in 2021 to describe models that serve as a *base* - a foundation - for many downstream tasks.

What makes a foundation model different from earlier machine learning models is its generality. Traditional ML models were trained for one specific task: classify this image, predict this value, detect this anomaly. A foundation model learns general patterns from enormous datasets and can then be adapted to many different tasks without being retrained from scratch.

Foundation models can be text-based (like GPT-4, Claude, or Llama), image-based (like Stable Diffusion or DALL-E), or multi-modal (handling text, images, audio, and more). The key characteristic is scale and generality - not the specific modality.

A crucial practical point: foundation models alone are typically not production-ready for industrial contexts. They require orchestration, retrieval grounding, and tool integration as standard additions before they can operate reliably in enterprise environments (Cho, 2025; Freeman et al., 2026; Osmond & Jego, 2026). When a vendor says they built something "on top of a foundation model," the next question is always: what did they build around it?

1.2 Large Language Model (LLM)

A **Large Language Model** is the most common type of foundation model - specifically, one trained on text. In the Layer A-E framework, LLMs are domain-applicable instantiations of foundation models for language understanding and generation, often paired with domain adapters, safety constraints, and governance overlays in industrial contexts (Cho, 2025; Pellejero, 2025).

Under the hood, an LLM is a neural network (almost always based on the Transformer architecture, introduced by Google in 2017) trained on enormous amounts of text data. During training, the model learns to predict the next token (roughly: the next word or word fragment) in a sequence. Through this deceptively simple objective - repeated billions of times across terabytes of text - the model develops an internal representation of language, facts, reasoning patterns, code structure, and much more.

The "Large" in LLM refers to the number of parameters (the adjustable weights in the neural network). Current frontier models have hundreds of billions of parameters, though the exact numbers are increasingly kept confidential by the companies building them.

What an LLM can do on its own:

- Generate text, translate languages, summarize documents
- Answer questions based on what it learned during training
- Write and explain code
- Reason through problems step by step
- Follow complex instructions

What an LLM *cannot* do on its own:

- Access real-time information (it only knows what was in its training data)
- Take actions in the real world (send emails, query databases, control machines)
- Learn from new interactions without retraining
- Reliably do math beyond what can be reasoned through step by step
- Access your proprietary data unless you provide it in the prompt

This distinction is critical. A raw LLM is a powerful text-processing engine with a broad but frozen knowledge base. It has no memory between conversations, no access to external systems, and no ability to act. It can reason, but it can't do.

Examples: GPT-4, Claude, Gemini, Llama, Mistral, Command R, DeepSeek

Industrial context: When someone tells you they "integrated an LLM" into a workflow, the first question should be: integrated *how*? A standalone LLM

can help draft emails and summarize reports. But for anything that touches real enterprise data, operational systems, or decision-making, the LLM alone is not enough. You need additional architecture around it - which is everything from Layer B onward.

1.3 Prompt, System Prompt & Prompt Engineering

A **prompt** is the text input you send to an LLM. Everything the model knows about your task comes from what's in the prompt - plus what it learned during training.

A **system prompt** is a specific type of prompt that defines the model's overall behavior, role, and constraints. Unlike the user prompt (the actual question or task), the system prompt is set by the application developer and stays constant across interactions. Think of it as the model's job description: who it is, what it should and shouldn't do, how it should respond, and what rules it must follow.

A system prompt for a manufacturing support assistant might include:

- "You are a technical support assistant for CNC machining operations."
- "Only answer questions related to machining, tooling, and machine maintenance."
- "If a question is about safety, always recommend consulting the machine manual and onsite safety officer."
- "When referencing machine parameters, always include the parameter code."

Prompt engineering is the discipline of designing effective prompts to get reliable, high-quality outputs from LLMs. Key techniques include zero-shot prompting (no examples), few-shot prompting (providing examples), chain-of-thought reasoning (instructing step-by-step thinking), and structured output formatting.

Industrial context: Prompt engineering is not a gimmick - it's a core competence. The difference between a well-engineered prompt and a naive one can be the difference between 60% and 95% accuracy on the same model. In production systems, prompts are version-controlled, tested, and iterated on just like code.

1.4 Context Window

The **context window** is the maximum amount of text an LLM can process at once - both input and output combined. It's measured in tokens, where one token is roughly 3/4 of a word in English.

Early LLMs had context windows of 2,000-4,000 tokens. Current frontier models offer 128,000 to over 1,000,000 tokens. This expansion matters because LLMs have no persistent memory. Everything the model needs to know about your task, your data, and your constraints must fit inside the context window.

Industrial context: In industrial applications, you're dealing with long technical documents - maintenance logs, specification sheets, engineering change orders, regulatory standards. The context window determines whether you can feed a complete 300-page equipment manual into the model or whether you need to break it into pieces. It directly affects architectures like RAG (Layer B).

1.5 Hallucination

Hallucination is the term for when an LLM generates text that sounds confident and plausible but is factually wrong or entirely fabricated. The model isn't lying - it has no concept of truth. It's producing the statistically most likely next token, and sometimes those patterns lead to plausible-sounding errors.

Types include factual errors stated with confidence, fabricated references to papers or standards that don't exist, technically-worded explanations that sound correct but aren't, and invented statistics or measurements.

Hallucination rates have decreased with newer models, and techniques like RAG reduce them further. But they haven't been eliminated. Recent research on industrial LLM hallucination reduction emphasizes that engineering consistent procedures for hallucination control is an active field - not a solved problem (Freeman et al., 2026).

Industrial context: In manufacturing and engineering, a hallucinated torque specification, a fabricated material property, or an incorrect safety procedure can cause real physical harm. This is the core reason why "just plugging in an LLM" is not sufficient. You need grounding, guardrails, verification, and human-in-the-loop processes - especially for anything touching safety or compliance.

1.6 Fine-Tuning

Fine-tuning takes a pre-trained foundation model and trains it further on a smaller, specialized dataset to improve its performance on specific tasks.

Approaches range from full fine-tuning (updating all parameters - expensive) to LoRA/QLoRA (updating a small number of additional parameters - much cheaper) to RLHF (using human ratings to align the model's behavior with human preferences).

Fine-tuning is different from RAG. RAG gives the model access to new information at query time. Fine-tuning changes the model's behavior and capabilities permanently. They're complementary: you might fine-tune a model to better understand your domain's technical language and then use RAG to give it access to current documents.

Industrial context: Fine-tuning is relevant when you need behavior that can't be achieved through prompting alone - for example, generating outputs in specific structured formats, understanding niche technical terminology, or following domain-specific reasoning patterns. It's more expensive and complex than RAG, so it's the escalation path when prompt engineering and retrieval aren't sufficient.

1.7 Model Sizes: Frontier, Mid-Tier & Small Models

Not every task needs the most powerful model:

- **Frontier models** (GPT-4o, Claude Opus, Gemini Ultra): Most capable, most expensive. Best for complex reasoning and nuanced understanding
- **Mid-tier models** (Claude Sonnet, GPT-4o-mini, Gemini Flash): Strong capability at significantly lower cost. Suitable for most production applications
- **Small/edge models** (Llama 3 8B, Mistral 7B, Phi-3): Run on local hardware or edge devices. Lower capability but faster, cheaper, and work without cloud connectivity

In agentic architectures (Layer D), it's common to use different models for different agents. The planning agent might use a frontier model, while a structured data-extraction agent uses a smaller one.

Industrial context: Model selection is an optimization problem. In manufacturing environments with latency requirements, data sovereignty constraints, or edge deployment needs, the answer is often: use the smallest model that reliably handles the task, and route complex edge cases to a more capable model.

1.8 Open Source vs. Closed Source Models

- **Closed source / proprietary** (GPT-4, Claude, Gemini): Accessed via API. The provider hosts the model; your data is sent to their servers
- **Open source / open weight** (Llama, Mistral, Qwen, DeepSeek): Weights are publicly available. Can be hosted on your infrastructure, fine-tuned, and modified. No data leaves your environment

Industrial context: For companies with strict data governance - especially in defense, automotive, pharma, and critical infrastructure - running models on-premise is often a hard requirement. Open source models make this possible. The trade-off: you own the infrastructure, scaling, and maintenance.

---

2 Layer B: Retrieval & Grounding

2.1 RAG (Retrieval-Augmented Generation)

Retrieval-Augmented Generation is an architecture pattern that gives an LLM access to external

knowledge at inference time. In the industrial context, this means augmenting outputs with retrieved documents from domain registries, equipment manuals, CAD libraries, knowledge bases, and other enterprise sources (Freeman et al., 2026; Stein et al., 2025; Subramanian et al., 2025).

Here's how it works:

1. Your proprietary data (documents, manuals, databases) is split into chunks and converted into numerical vectors (embeddings) that capture semantic meaning
2. These embeddings are stored in a vector database
3. When a user asks a question, the question is also converted into an embedding
4. The system searches the vector database for semantically similar chunks
5. The most relevant chunks are inserted into the prompt alongside the question
6. The LLM generates a response based on both the question and the retrieved context

RAG solves two problems: it gives the LLM access to data it was never trained on, and it reduces hallucination by grounding answers in source material.

But RAG is not magic. The quality depends heavily on the retrieval step. If the system retrieves the wrong documents, the LLM will either hallucinate or give an incomplete answer. Building production-grade RAG requires careful work on chunking strategies, embedding models, retrieval algorithms, and re-ranking.

Industrial context: RAG is the most common architecture for enterprise AI assistants today. If you're building a system that answers questions about your internal data - maintenance history, engineering standards, quality records - you're building some form of RAG. The critical insight is that a demo on five curated documents may collapse when pointed at 50,000 unstructured PDFs from your actual document management system.

2.2 Grounding & Provenance

Grounding is the broader practice of connecting an LLM's outputs to verifiable sources of truth. RAG is the most common technique, but the concept includes cross-referencing against structured databases, requiring citation of specific passages, and using knowledge graphs to validate facts.

The literature is clear on this: grounding must align with domain ontologies and provenance controls to avoid ungrounded outputs. Deterministic or graph-grounded approaches can improve reliability in high-stakes contexts (Bekbergenova et al., 2025; Martim, 2025; Subramanian et al., 2025).

Provenance is the audit trail: where did each piece of information come from? Which document, which version, which retrieval step? In industrial deployments, provenance isn't a feature - it's a requirement. When an AI system recommends a specific torque value or references a maintenance interval, you need to trace that number back to the OEM manual, the standard, or the test report.

Industrial context: The combination of grounding and provenance is what makes AI outputs auditable and trustworthy. Without it, you have a sophisticated text generator. With it, you have a system that engineering and quality teams can actually rely on.

2.3 Embeddings & Vector Databases

Embeddings are numerical representations of text that capture semantic meaning. An embedding model converts text into a vector (a list of hundreds or thousands of numbers) where similar meanings cluster together. "Turbine blade failure" and "rotor component breakdown" end up close in vector space even though they share no words.

A **vector database** is optimized for storing and searching these vectors. When you have millions of document chunks as embeddings, a vector database finds the most semantically similar ones in milliseconds. Popular options include Pinecone, Weaviate, Qdrant, Milvus, and pgvector.

Industrial context: Embeddings and vector databases are the backbone of RAG systems. They're also used for semantic search across unstructured industrial data, clustering similar maintenance tickets or defect reports, and anomaly de-

tection on text data.

---

3 Layer C: Agents & Multi-Agent Frameworks

3.1 AI Agent

This is where the terminology starts to get genuinely important - and genuinely confused in the market.

An **AI agent** is an LLM wrapped in an architecture that gives it the ability to observe, reason, plan, and *act*. In the literature, an agent is defined as a component that observes its environment, reasons about what to do, and acts via tools (Bhattacharya, 2025; Bluethgen, 2025; Ghosh, 2025).

The core components:

1. **LLM (the brain):** The foundation model that handles reasoning and decision-making
2. **System prompt (the role definition):** Instructions that define purpose, constraints, and behavior
3. **Tools (the hands):** External capabilities the agent can invoke - API calls to ERP systems, database queries, code execution, web search, file operations
4. **Knowledge (the memory):** Access to relevant information through RAG, document stores, or knowledge graphs
5. **Orchestration logic (the executive function):** The control flow that manages how the agent reasons, decides which tool to use, evaluates results, and determines next steps

The critical distinction: a raw LLM can tell you what *should* be done. An agent can actually *do it*.

Consider the difference:

- **LLM:** "Based on the maintenance schedule, pump P-4401 is due for bearing replacement. You should create a work order in SAP PM."
- **Agent:** "Pump P-4401 is due for bearing replacement. I've created work order 4401-2026-0513 in SAP PM, assigned it to the Tuesday maintenance window, and notified the maintenance lead."

The agent didn't just reason - it acted. It called the SAP API, created the work order, checked the schedule, and sent a notification.

Industrial context: When you hear "AI agent" from a vendor, the key questions are: What tools does it have access to? What systems can it read from and write to? What are its boundaries? An agent with read-only access to a knowledge base is very different from one that can modify production parameters or approve purchase orders. The scope of the toolset defines the scope of the risk - and the value.

3.2 Tool Use / Function Calling

Tool use (also called **function calling**) is the mechanism that lets an LLM interact with external systems. It works like this:

1. The developer defines available tools with names, descriptions, and expected parameters
2. When the model determines it needs a tool, it generates structured output specifying which tool to call and with what parameters
3. The application code executes the actual call (the model never directly accesses external systems)
4. The result is fed back to the model for further processing

This is an important nuance: the LLM *decides* that a tool should be called and *specifies* the parameters. The actual execution happens in the application layer. This provides a natural security boundary.

Industrial context: Tool use is what makes the difference between an AI chatbot and an AI colleague. Every practical enterprise AI application relies on it. Understanding this mechanism helps you design appropriate security boundaries and govern what the AI can and cannot do.

3.3 Multi-Agent System (MAS)

A **multi-agent system** is a coordinated set of agents that collaborate for planning, tool use, and data access. It supports scalable, auditable pipelines by distributing work across specialized

agents (Bhattacharya, 2025; Bluethgen, 2025; Ghosh, 2025; Rahman, 2025).

The term "multi-agent system" predates LLMs by decades - it originates from classical computer science (distributed control systems, AGV fleet management, market-based scheduling algorithms). What's new is that LLM-based agents can handle ambiguity, interpret unstructured data, reason across domains, and communicate in natural language.

Here's a nuance the literature emphasizes that matters enormously in practice: many so-called agentic systems actually rely on simple prompt-driven tool calls. True MAS implies structured coordination, negotiation, and explicit communication protocols with governance hooks (Osmond & Jego, 2026; Rahman, 2025). The difference between "an LLM that calls three tools in a loop" and "three specialized agents with defined roles, communication protocols, and coordination logic" is not semantic - it's architectural, and it has direct implications for reliability, auditability, and scalability.

Industrial context: If a vendor says "multi-agent," ask whether they mean classical MAS (rule-based, deterministic, well-understood) or LLM-based MAS (more flexible but less predictable). Then ask about the coordination mechanism. The engineering and governance requirements are very different.

---

4 Layer D: Agentic AI & Agentic Systems

4.1 Agentic AI

This is the term that generates the most confusion - and where precision matters most.

Agentic AI refers to an architecture where an orchestrator decomposes goals, plans across specialized agents, manages memory, and coordinates external actions with governance overlays (Amalfitano, 2025; Ge, 2025; Ghosh, 2025; Osmond & Jego, 2026). The word "agentic" describes the *property* of a system that operates with agency: autonomy, goal-directed behavior, and coordinated action.

Here's the progression:

- **LLM:** A single brain with no tools
- **AI Agent:** A brain with tools - it can observe, reason, and act
- **Agentic AI:** An architecture where an orchestrator coordinates multiple specialized agents, each with their own tools and roles, to accomplish complex goals that no single agent could handle alone

The literature distinguishes between two related but distinct terms:

Agentic AI is the architectural pattern - the design principle of decomposition, planning, delegation, memory, and governance.

Agentic system is the deployed instantiation: a running system that includes the orchestrator, the delegate agents, and integration with domain tools and data. It exhibits planning, memory, tool use, and cross-agent coordination (Amalfitano, 2025; Bluethgen, 2025; Ge, 2025; Pellejero, 2025).

This distinction matters in practice. A vendor might have agentic AI capabilities in their platform (Layer D architecture) but the agentic system you deploy will be specific to your environment, your data, your tools, and your governance requirements.

4.2 What Agentic Looks Like in Practice

In an agentic system, you might have:

- A **planning agent** that breaks down complex tasks into subtasks
- A **research agent** that gathers information from multiple sources
- A **domain expert agent** that applies specialized knowledge (regulatory compliance, engineering standards)
- An **execution agent** that carries out actions in target systems
- A **review agent** that checks work for quality and compliance
- A **coordinator/orchestrator** that manages workflow and resolves conflicts

These agents communicate through structured messages, share context, and can escalate to each other or to humans when they reach their limits.

A concrete industrial example:

Consider an agentic system handling an unplanned equipment failure in a chemical plant:

1. The **monitoring agent** detects an anomaly in vibration data from a pump
2. The **diagnostic agent** pulls the pump's maintenance history from the CMMS, retrieves the OEM manual, analyzes the vibration signature, and hypothesizes bearing degradation
3. The **planning agent** evaluates options: immediate shutdown vs. reduced operation until next planned maintenance. It checks production schedules in the MES and pending orders in the ERP
4. The **procurement agent** checks spare parts in the warehouse management system and queries supplier catalogs for lead times
5. The **compliance agent** verifies the proposed action against safety protocols and regulatory requirements
6. The **coordination agent** compiles findings into a recommended action plan and routes it to the shift supervisor for approval (human-in-the-loop)
7. Upon approval, the **execution agent** creates the work order, reserves parts, updates the production schedule, and notifies the maintenance team

No single agent does all of this. The value comes from specialization and coordination.

Key characteristics of agentic systems:

- **Autonomy:** Agents make decisions without human intervention at every step
- **Multi-step reasoning:** The system plans and executes workflows spanning many actions
- **Delegation:** Tasks are broken down and assigned to specialized agents
- **Self-correction:** Agents evaluate their own outputs, detect errors, and retry
- **Human-in-the-loop:** Well-designed systems include checkpoints for human review of critical decisions
- **Memory:** Semantic and episodic memory to support repeatable decision logs (Bluethgen, 2025; Subramanian et al., 2025)

Industrial context: Agentic systems represent the direction the industry is heading, but every agent-to-agent interaction is a potential failure point and every autonomous decision a potential risk. The governance, testing, and monitoring requirements are fundamentally different from those for a single chatbot.

4.3 A Critical Distinction: Agentic vs. "Agentic-Washed"

The literature is pointed about this: the definition of "agentic AI" varies significantly across the market. Some works stress orchestration and planning as core requirements. Others equate "agentic" with any multi-agent prompting (Ge, 2025; Ghosh, 2025; Osmond & Jego, 2026).

The architecture-centered view - which we advocate - requires that agentic AI exhibits structured orchestration, planning, tool use, and cross-agent coordination beyond simple multi-prompt tool calling.

A practical litmus test: if removing the word "agentic" from the product description doesn't change what the product actually does, it's marketing. Agentic architecture implies coordination protocols, shared memory, planning loops, and governance hooks. If those aren't present, what you have is an LLM with tools - which is valuable, but it's Layer C, not Layer D.

---

5 Layer E: Orchestration, Governance & Economics

5.1 Orchestration

Orchestration is the cross-agent sequencing of perception, planning, tool use, and memory updates. It's central to reproducible, auditable industrial deployments (Amalfitano, 2025; Bhattacharya, 2025; Bluethgen, 2025).

Orchestration determines:

- Which agent handles which task
- In what order tasks are executed
- How data flows between agents
- When to involve humans
- What happens when an agent fails
- How the overall state of a complex process is managed

Orchestration can be as simple as a fixed sequence or as complex as a dynamic graph where the next step depends on previous results. Research points to Router-Planner-Coordinator (RPC) patterns with provenance and memory as practical, transparent architectures for multi-agent workflows in scientific and engineering spaces (Bhattacharya, 2025; Bluethgen, 2025).

Industrial context: Orchestration is where the engineering complexity lives. A well-orchestrated system is reliable, traceable, and governable. In industrial settings where you need auditability and reproducibility, the orchestration layer is arguably more important than the AI models themselves.

5.2 Governance

Governance encompasses the policies, standards, and procedures governing data use, model risk, interpretability, auditing, and regulatory compliance. In the context of agentic AI, this includes permissions, sandboxing, access controls, and auditable logs (Axetorn et al., 2025; Bhattacharya, 2025; Bluethgen, 2025; Osmond & Jego, 2026).

Governance spans multiple levels:

- **Input governance:** What data can the system access? What's sanitized? What's blocked?
- **Output governance:** How are outputs validated? What hallucination checks are in place?
- **Action governance:** What can agents actually do? What requires approval? What's read-only?
- **Behavioral governance:** Does the system stay within its defined role?
- **Audit governance:** Is every decision traceable? Every tool call logged? Every retrieval documented?

The literature is convergent on this point: governance and provenance are central to trust and regulatory alignment in industrial AI deployments. This is not a later-stage concern - governance must be established early, including provenance traceability, access control, data lineage, and auditability aligned with organizational risk tolerances (Axetorn et al., 2025; Bhattacharya, 2025; Osmond & Jego, 2026).

Industrial context: In manufacturing and engineering, governance isn't optional. It's the difference between an AI assistant and an AI liability. Every action an agent takes in a production system must be bounded by governance appropriate to the risk level.

5.3 Human-in-the-Loop (HITL)

Human-in-the-loop refers to system designs where humans review, approve, or correct AI outputs at defined checkpoints before actions are taken.

The spectrum of autonomy:

- **Human-in-the-loop:** Human approval required before action
- **Human-on-the-loop:** AI acts autonomously but human monitors and can intervene
- **Human-out-of-the-loop:** Fully autonomous operation within defined parameters

Industrial context: The appropriate level of human involvement depends on consequences of error. Adjusting a non-critical temperature setpoint within a validated range is very different from approving a design change on a safety-critical component. Industrial organizations need a clear framework for where each AI-driven process falls on this spectrum.

5.4 Determinism vs. Stochasticity

This is a tension that doesn't get enough attention in vendor conversations but matters enormously in industrial production.

LLMs are inherently stochastic - they produce probabilistic outputs that can vary between runs. Industrial deployments often require auditable, repro-

ducible decision paths. Architectures that emphasize deterministic components or auditable pipelines - for example, graph-grounded retrieval with canonical primitives - are advantageous in high-stakes contexts (Martim, 2025).

This doesn't mean you can't use LLMs in industrial settings. It means you need to design systems where the stochastic components (LLM reasoning) are wrapped in deterministic structures (fixed validation rules, structured output schemas, logged decision paths) that provide the auditability your compliance team needs.

Industrial context: When evaluating AI solutions for production environments, ask: "If I run this exact query twice, will I get the same answer?" If the answer is "not necessarily" - which it often will be - the follow-up is: "What deterministic controls ensure the output is within acceptable bounds?"

5.5 Economics

The economics of AI in industrial settings is often underestimated. Understanding the cost structure is essential for realistic business cases:

- **Inference costs:** LLM pricing is per token. A complex agentic workflow with five agents, each making two tool calls, might consume 10-15 separate LLM calls. Each includes the system prompt, conversation history, tool results, and output
- **Data costs:** Maintaining vector databases, indexing enterprise documents, keeping retrieval pipelines current
- **Tool integration costs:** Building and maintaining connections to enterprise systems (ERP, PLM, MES, CMMS)
- **Governance costs:** Logging, auditing, compliance monitoring, human review processes
- **Lifecycle costs:** Model updates, prompt maintenance, retraining/re-embedding when source documents change

Governance directly informs procurement criteria, licensing decisions, data storage requirements, and ongoing operational budgets (Bhattacharya, 2025; Cho, 2025; Freeman et al., 2026; Ge, 2025).

Industrial context: Token economics directly affect the business case. A customer support chatbot uses far fewer tokens per interaction than an engineering copilot retrieving long technical documents and reasoning through multi-step problems. Model the token consumption of your intended workflows before committing to an architecture.

5.6 MCP (Model Context Protocol)

MCP is an open standard introduced by Anthropic in late 2024 for connecting AI agents to external tools and data sources. Instead of building custom integrations for every tool, MCP provides a standardized interface for agents to discover and use tools.

The architecture follows a client-server model: the AI application connects to MCP servers that expose tools in a standardized format. The agent discovers what's available, understands parameters, and calls tools through a uniform interface.

Industrial context: In environments with fragmented tool landscapes - PLM, MES, ERP, CMMS, SCADA - integration is the bottleneck. MCP standardizes the AI-to-tool interface in a way that could reduce the cost of connecting agents to industrial systems. It doesn't solve enterprise integration by itself, but it addresses a real friction point.

---

6 Production-Readiness: What the Literature Tells Us

Across the research on industrial AI deployment, five production-readiness criteria emerge consistently:

1. Grounding and provenance: Explicit logging of retrievals, tool invocations, and reasoning episodes. Alignment with domain glossaries and data provenance controls (Bhattacharya, 2025; Freeman et al., 2026; Subramanian et al., 2025).

2. Tool integration and memory: Robust domain-tool integration with semantic and episodic memory to support repeatable decision logs (Bluethgen, 2025; Subramanian et al., 2025).

3. Governance and auditing: Formal policies for data access, model risk, human-in-the-loop requirements, and regulatory alignment (Axetorn et al., 2025; Bhattacharya, 2025; Osmond & Jego, 2026).

4. Determinism and evaluation: Where required, deterministic components or auditable pipelines (graph-grounded retrieval, canonical primitives) to support compliance (Martim, 2025; Subramanian et al., 2025).

5. Economics discipline: Explicit cost models for data access, tool usage, and lifecycle maintenance. Governance shapes procurement budgets and risk planning (Bhattacharya, 2025; Cho, 2025; Freeman et al., 2026).

If a vendor or internal team can't articulate their position on all five, the solution isn't production-ready for industrial deployment.

---

7 Industrial Scenarios: Mapping Terms to Reality

To make the Layer A-E framework concrete, here are three representative industrial scenarios:

7.1 Scenario 1: Engineering Manual Q&A System

A system that answers questions about equipment manuals, engineering standards, and CAD documentation.

- **Layer A:** Foundation model with domain vocabulary (through fine-tuning or prompt engineering)
- **Layer B:** RAG pipeline retrieving from manual libraries and CAD catalogs; grounding outputs with source citations
- **Layer C:** Agent loop for routing questions to appropriate domain tools and knowledge sources
- **Layer D:** Governance hooks for data provenance and usage constraints (who accessed what, when)
- **Layer E:** Economic model including licensing costs for domain data sources; audit logs for compliance

This is Layer A-B heavy. Most of the value comes from high-quality retrieval and grounding, not from complex agent coordination.

7.2 Scenario 2: Multi-Source Equipment Monitoring

A system that monitors equipment health by integrating sensor data, maintenance logs, diagnostic tools, and operator reports.

- **Layer B:** Retrieval of sensor logs and historical maintenance records
- **Layer C:** Multiple agents querying diagnostic tools and databases; episodic memory for tracking equipment state over time
- **Layer D:** Auditable decision logs linking every recommendation to source data; human-in-the-loop for critical maintenance decisions
- **Layer E:** Operational cost model for continuous data streams and tool calls; governance for alert thresholds and escalation procedures

This is Layer B-C heavy, with Layer D governance becoming critical for any autonomous decision-making.

7.3 Scenario 3: Robot Cell Orchestration with Memory-Augmented Planning

A system that coordinates multiple robots in a manufacturing cell, adapting to changing orders and conditions.

- **Layer C:** Multi-agent coordination between a planner agent and tool-using execution agents; memory stores for task context and learned sequences
- **Layer D:** Agentic orchestration with planning, delegation, and re-planning when conditions change
- **Layer E:** Strict governance for access to robot controllers; risk budgeting for autonomous actions; clear human-override protocols

This is Layer C-D heavy, with Layer E governance and safety controls as the critical enablers.

The point: not every industrial AI application needs all five layers at full intensity. Understanding which layers matter most for your use case prevents over-engineering simple problems and under-governing complex ones.

---

8 A Decoder for Vendor Conversations

Using the Layer A-E framework, here's how to cut through common vendor claims:

- **"We use AI / We have an LLM"** → Layer A only. Ask: what's built around it? How does it access your data? What can it do beyond generate text?
- **"We have AI agents"** → Claimed Layer C. Ask: What tools do the agents have? What systems can they read from and write to? Or is this a chatbot with a creative product name?
- **"Our platform is agentic"** → Claimed Layer D. Ask: How many agents? How do they coordinate? What's the orchestration logic? Is there shared memory? Or does "agentic" just mean "has an LLM with a few tool calls"?
- **"We use RAG"** → Layer B. Ask: What's the retrieval accuracy? How do they handle documents the system can't parse? What's the provenance model?
- **"Powered by GPT-4 / Claude / Gemini"** → Layer A brand name. Ask: How? As a raw LLM? With RAG? Inside an agent architecture? The model is 10% of the story
- **"AI-driven autonomous operations"** → Claimed Layer D-E. Ask: Autonomous within what boundaries? What's the human-in-the-loop process? What happens when the AI is wrong? What's the governance model?

Map every vendor claim to a Layer. If they can't tell you which layer they're actually operating at, they haven't thought it through.

---

9 Conclusion

Understanding these terms isn't academic - it's operational. Every decision about where to invest in AI, which vendors to trust, and how to govern AI in your organization depends on understanding what these technologies actually are and how they relate to each other.

The progression from LLM → Agent → Agentic System traces an increase in capability, but also in complexity, risk, and governance requirements. More capable systems deliver more value, but they require more sophisticated engineering, governance, and organizational readiness.

The Layer A-E framework gives you a shared vocabulary for these conversations - with your engineering teams, your vendors, your leadership, and your regulators. Start with clarity on the terminology. It's the foundation for everything that follows.

9.1 About the authors

Vlad Larichev is Industrial AI Lead at Accenture and co-founder of AI² – Association for Industrial AI. His work focuses on how industrial organisations adopt AI in engineering, manufacturing, and operations: from clarifying procurement and architecture decisions to aligning governance, data, and vendor claims with what is technically defensible on the shop floor and in the enterprise. Contact: vlad.larichev@ai2n.eu.

Alexey Samoshilov leads Industrial Services Sales & Strategy at Siemens. He works at the intersection of industrial customers, service portfolios, and digital offerings—helping organisations translate AI and automation ambitions into roadmaps that respect operational reality, delivery constraints, and long-term service relationships.

The AI² – Association for Industrial AI is an independent network of practitioners and leaders advancing the responsible adoption of AI in industrial and manufacturing environments. Learn more at industrial-ai-network.com.

This article draws on recent literature surveyed for the AI² – Association for Industrial AI research paper "A Domain-Driven Terminology for Industrial AI: Distinctions, Literature Alignments, and Practical Guidance" (Larichev & Samoshilov, 2025).

Have a term we should add? Reach out to Vlad at vlad.larichev@ai2n.eu; this is a living document that we update as the technology evolves.

Amalfitano, D. (2025). *A research roadmap for augmenting software engineering processes and software products with generative AI.*

Axetorn, J., Edholm, F., Dobslaw, F., & Gren, L. (2025). *Addressing trust requirements in the design of an open-source multi-agent LLM-based domain-specific chatbot.*

Bekbergenova, M. et al. (2025). *MetaboT: An LLM-based multi-agent framework for interactive analysis of mass spectrometry metabolomics knowledge.*

Bhattacharya, C. (2025). *AISAC: An integrated multi-agent system for transparent, retrieval-grounded scientific assistance.*

Bluethgen, C. (2025). *Agentic systems in radiology: Design, applications, evaluation, and challenges.*

Cho, W. I. (2025). *Behavior modeling for training-free building of private domain multi agent system.*

Freeman, B., Kicklighter, A., Erdman, M., & Gordon, Z. (2026). *Toward epistemic stability: Engineering consistent procedures for industrial LLM hallucination reduction.*

Ge, Y. (2025). *A survey of vibe coding with large language models.*

Ghosh, S. (2025). *Agentic AI systems in electrical power systems engineering: Current state-of-the-art and challenges.*

Larichev, V., & Samoshilov, A. (2025). *A domain-driven terminology for industrial AI: Distinctions, literature alignments, and practical guidance.*

Martim, H. de. (2025). *Deterministic legal agents: A canonical primitive API for auditable reasoning over temporal knowledge graphs.*

Osmond, M., & Jego, T. (2026). *Mind the gap: How the technical mechanism of agentic AI outpace global legal frameworks.*

Pellejero, J. (2025). *Agentic AI for mobile network RAN management and optimization.*

Rahman, M. (2025). *LLM-based data science agents: A survey of capabilities, challenges, and future directions.*

Stein, S. et al. (2025). *Leveraging MDS2 and SBOM data for LLM-assisted vulnerability analysis of medical devices.*

Subramanian, D., Unhelkar, B., & Chakrabarti, P. (2025). *Design and evaluation of a context-aware multimodal recommendation and QA system with retrieval-augmented generation.*