BCG

AI Platforms Group

Building effective enterprise agents

AI Platforms Group Briefing

Tom Martin, David Heurtaux, Caitlin Barber, Mathilde M. Solberg, Niels Degrande, Djon Kleine, Dan Sack, Julien Marx, Dan Martines, Gene Sheenko, Nicolas De Bellefonds

NOVEMBER 2025

Much has been published on building **AI Agents.** Most are theoretical, provide guidance that works only at small scale, are hyperbolic, or conveniently ignore the complexities of the world's businesses – old technology stacks, messy data, international footprints and complex governance.

This brief aims to plug this gap, exploring how to **build reliable, trusted AI Agents in the enterprise**; the patterns, platforms, techniques, and capabilities needed to realize effective production grade agents.

Building Effective Enterprise Agents means facing a sea of legacy

Source: ChatGPT

01

Why is it hard to build agents in the enterprise?

02

How do you design an enterprise agent?

03

How do you build an enterprise agent?

04

How do you assemble an agent platform?

01

Why is it hard to build agents in the enterprise?

Two years of experiments and AI-hype have left leaders looking for answers

Leadership face many questions…

How do I keep AI efforts value focused?

To ensure what I do hits the P&L, is adopted by users, and is a net benefit, not a distraction?

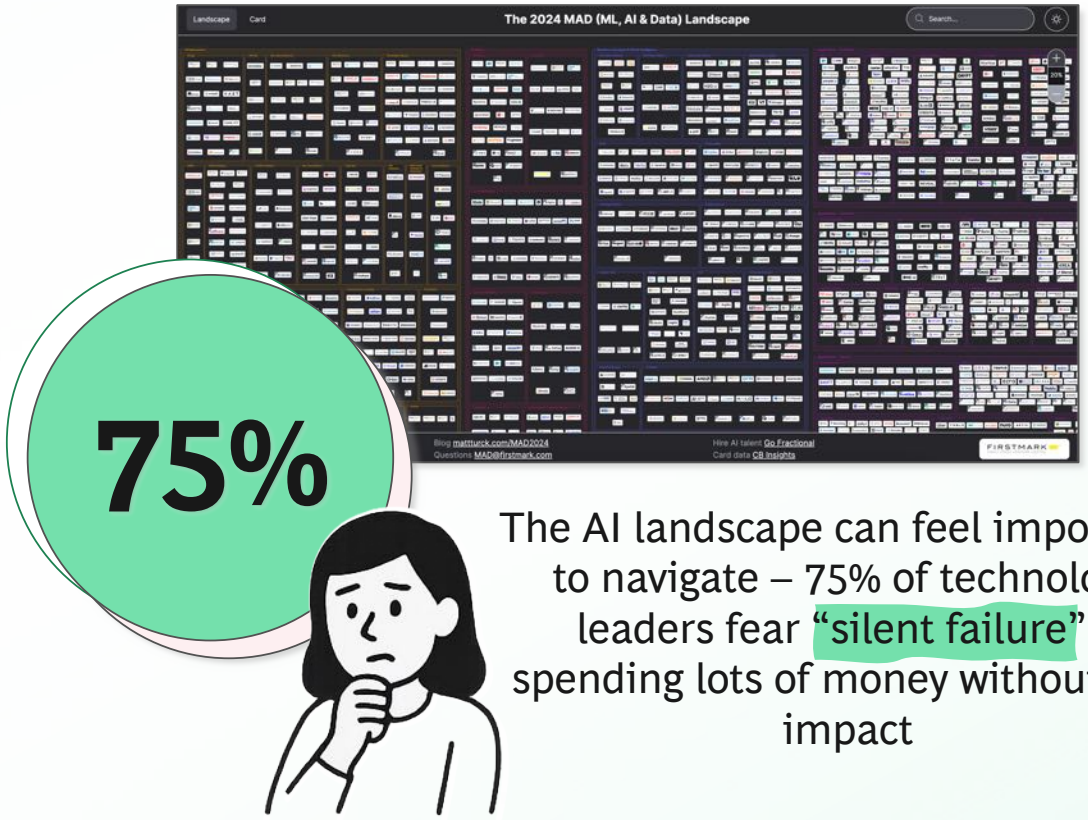
How do I keep AI under control?

To make them reliable, avoid unnecessary cost, avoid cyber and data risks, keep it secure, all whilst avoiding lock-in and FOMO based purchasing

How do I scale reliably?

To do it 100 times, not just once or twice, to build and manage agents without exploding complexity, to prepare my data and core systems, and should I build or buy?

… when trying to navigate the world of AI



The AI landscape can feel impossible to navigate – 75% of technology leaders fear "silent failure" – spending lots of money without real impact

Source: BCG market research Beyond AI Islands; MAD Fristmark AI and Data landscape

The promise of agents bring a new set of implementation demands

Agents promise to transform knowledge work …

Global Financial Data Firm's Compliance agent **reduces time-to-decision by 30-50%**

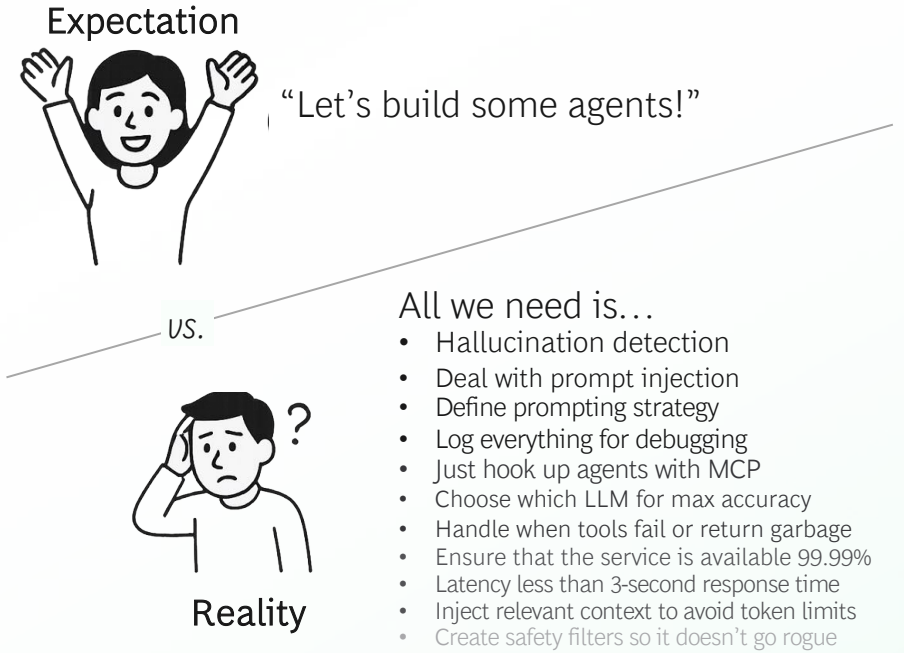
Global Travel Platform's and **Quantitative Trading Firm's** coding agents cut cycle times by **30%+**

AI Research Platform's agents turn 10,000 pages of legal and financial text into crisp decision-ready takeaways

BCG BOSTON CONSULTING GROUP

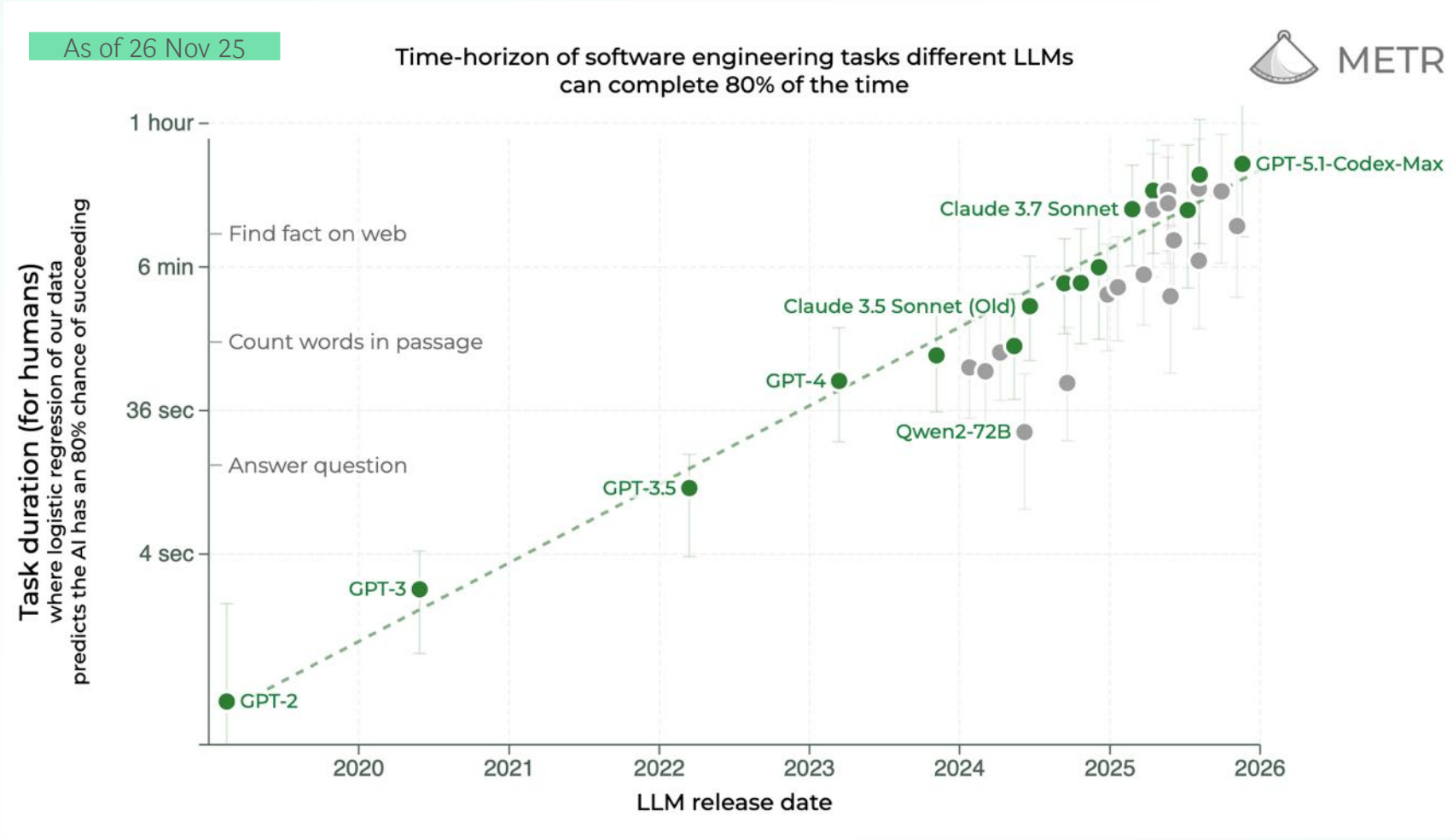
BCG delivered 300+ Agents across clients unlocking up to cost reduction, faster execution, and **30–40% productivity uplift**

… but bring their own challenges



Metrics are illustrative, depending on specific implementation context, and not guaranteed
Source: BCG

Research labs continue to push LLM capabilities, driving agent capabilities



Future outlook

Only as **reasoning** and **evaluation** systems mature, will fully autonomous agents be able to handle complex, open-ended tasks

Constrained agents are already operating at scale for more deterministic problems, providing predictable and granular controlled outcomes

Deep agents will lead the enterprise adoption of agents, orchestrating complex problems into smaller tasks to the right sub-agents

1. State Of The Art (SOTA) Large Language Models (LLMs)
Source: METR; BCG

The limiting factors for agents aren't LLMs, but legacy systems and processes

5 Key blockers we have observed in the last 2 years of building agents…

1 Brownfield integrations

Stitching agents into legacy stacks, heterogeneous APIs, and fine-grained RBAC **creates security, approval, and change-control risks**

2 Unreliable enterprise data

Siloed, low-trust, and slow-moving data makes agent decisions brittle; success at scale demands clean, real-time, well-governed data

3 Lack of evaluations

Complex reasoning agent paths hide failure modes; tracing tool calls, red teaming and evaluating on comprehensive data is non-trivial

4 Governance & Audit overhead

Enterprises demand explainability, guardrails, and policy compliance from day one to avoid **regulatory and reputational risk**, increasing upfront complexity

5 OpModel & Scale frictions

Moving from PoC to durable ops requires **proper ownership, incident mgmt., cost/latency control**, versioning, and change tracking in a complex environment

Source: BCG

Reality Check!

Enterprises should focus on building deep agents, integrated with legacy systems

Build up to Horizon 2 today

Horizon 0 — *Constrained agents*

At scale

An agent follows a predefined set rules for a single repetitive task, lacking flexibility in exchange for robustness

A simple customer service Q&A chatbot

Horizon 1 — *Single agents*

Adoption Rising

An agent that independently handles multi-step tasks in a set environment, planning and acting alone

A chatbot that guides users to solutions & acts on their behalf

Horizon 2 — *Deep agents*

Adoption Rising

An orchestrator agent splits complex tasks for specialist agents, assigning each a specific subtask

An agent that autonomously helps onboard an employee, sets up email, access, etc.

Horizon 3 — *Role-based agents*

Very early

A team of AI agents collaborate, each performing distinct roles, handing off work to each other

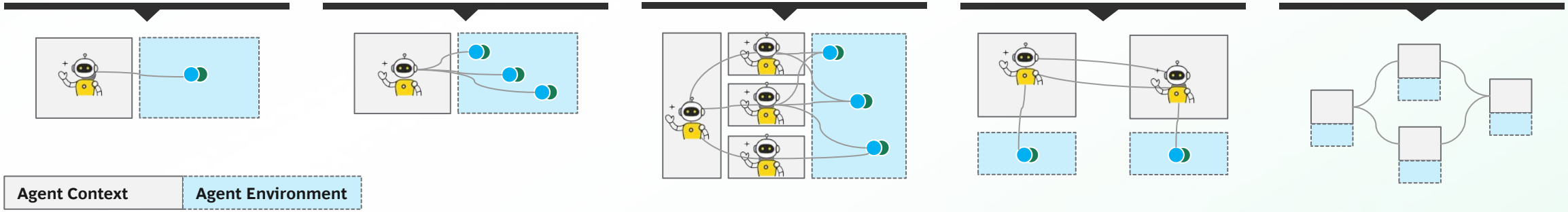
Agents managing a marketing campaign, one agent identifies the audience, and the other creates ads

Horizon 4 — *Agent mesh*

Early R&D

A network of autonomous agents that self-organize and generate sub-agents to tackle business challenges

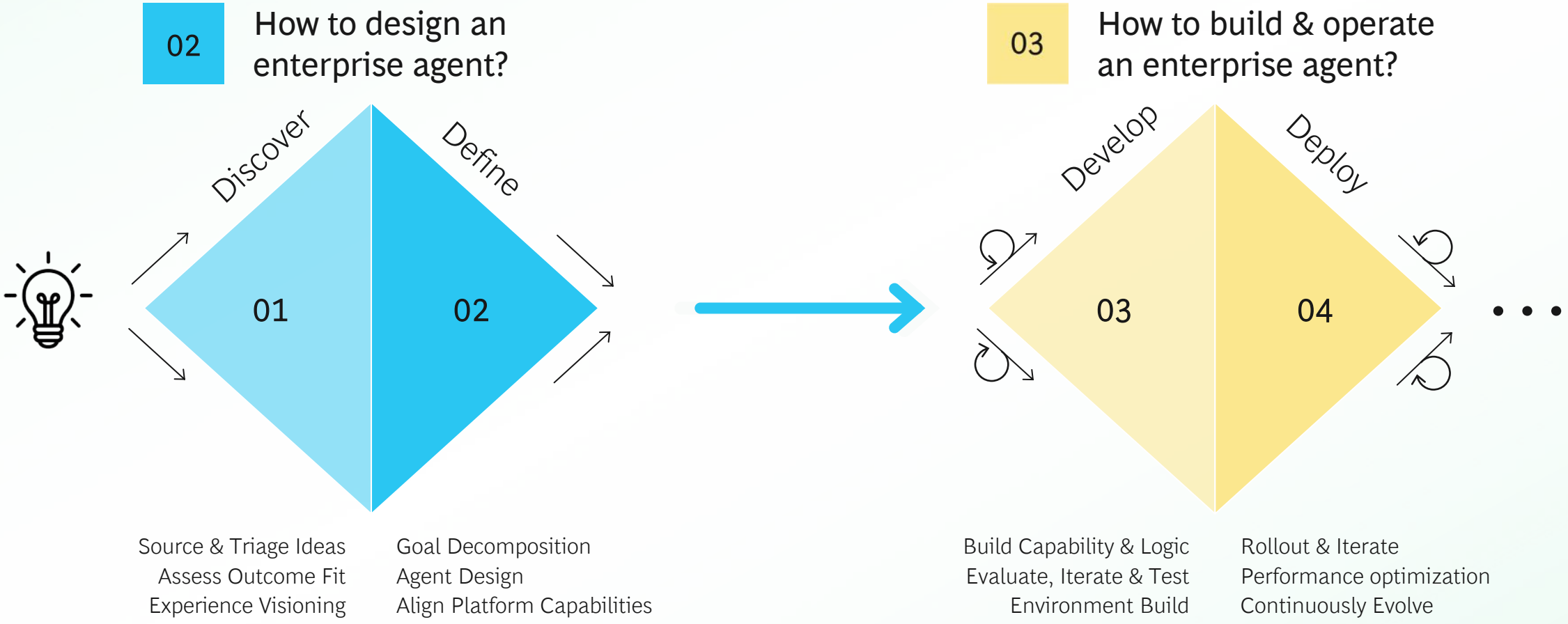
Agents autonomously improve supply chain efficiency with agents from various departments



Agent Context | **Agent Environment**

Source: BCG

BCG's approach to effective enterprise agents adapts the classic Double Diamond

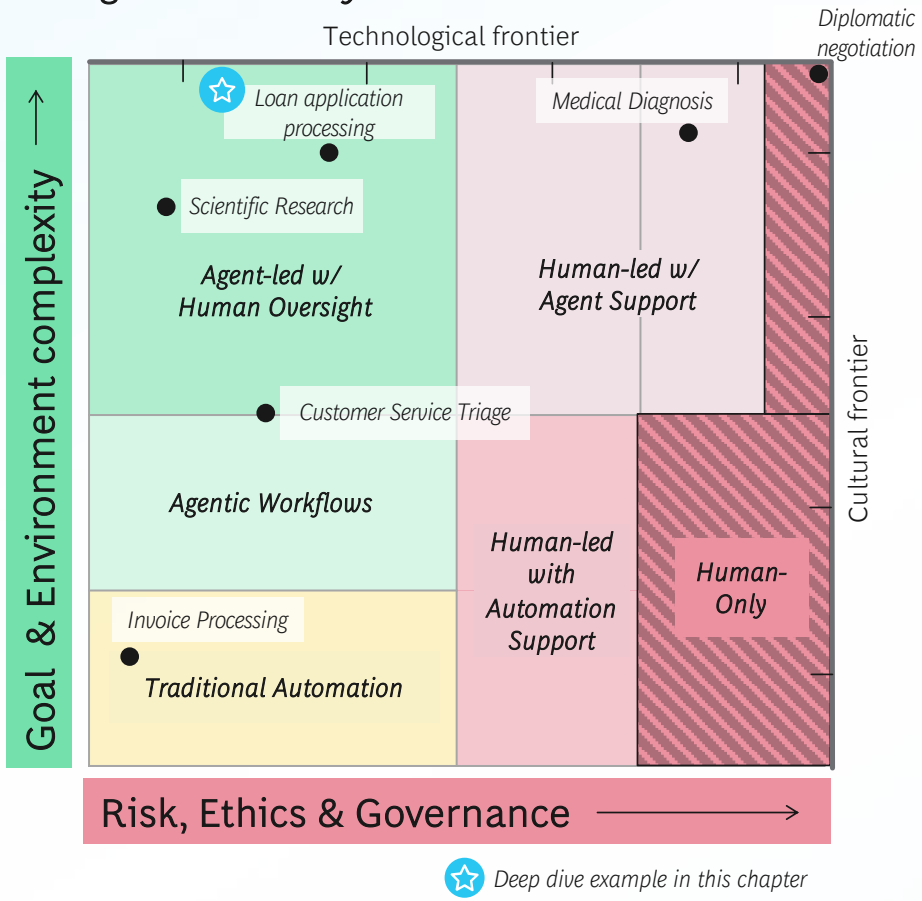


02

How do you design an enterprise agent?

Agents are best suited for complex problems that don't demand human oversight

The Agent Suitability Framework



Goal & Environment complexity is high when a solution must navigate a number of moving parts with adaptive reasoning required to achieve an outcome

Risk, Ethics & Governance requirements are high when decisions require moral judgment, consequence of error is severe, or regulatory demands mean human involvement is necessary

Determining agent suitability is a key step in the design process - think about outcome clarity, task complexity, human-judgment needs, guardrails, and the value case versus simpler automation

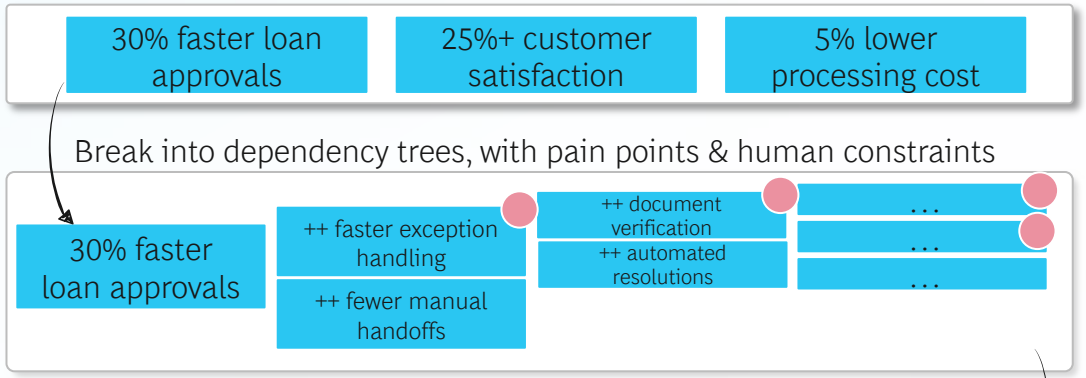
If clear rules & basic automation deliver the desired outcome, avoid building agents for agents' sake

Source: BCG

Agent design begins by anchoring on business outcomes, not process outputs

Example: Loan Application Processing

Start with outcomes as business goals ("What are we trying to achieve")



Prioritise opportunities systematically, consider Impact × Feasibility × Agent-fit

Key pain point

Outcome-first design defines success over process, forcing clarity on what good looks like helps prioritize where agents can add measurable value

Decomposition reveals leverage points, breaking outcomes intro dependencies and pain points uncovers the specific tasks and decision areas that move the KPIs

Agent opportunities then emerge naturally; clear dependencies allow for roles to be defined early, and the design process can develop from here

***BCG's Agentic Outcome Maps** for functions within different industries accelerate design phase ideation*

The key mantra - ***outcomes-not-outputs***

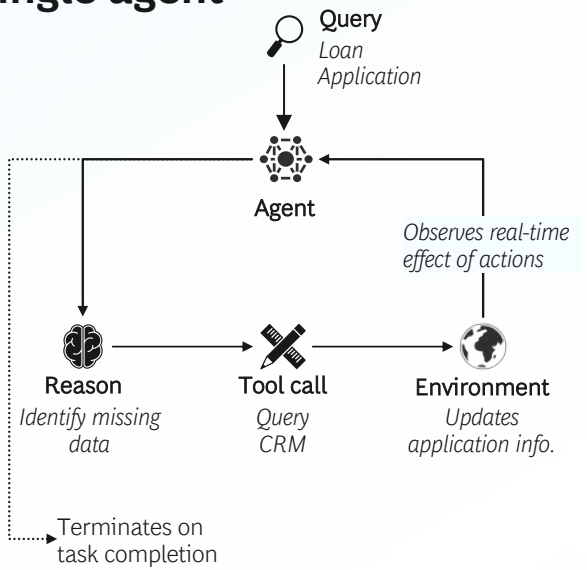
Source: BCG

Effective agents start simple in design, with added complexity only when needed

Success when building AI Agents comes from starting simple, with gradual decomposition into sub-flows or multi-agents as failure points indicate more complexity is needed. **Always go with the simplest solution for the task at hand**

Example: Loan Application Processing

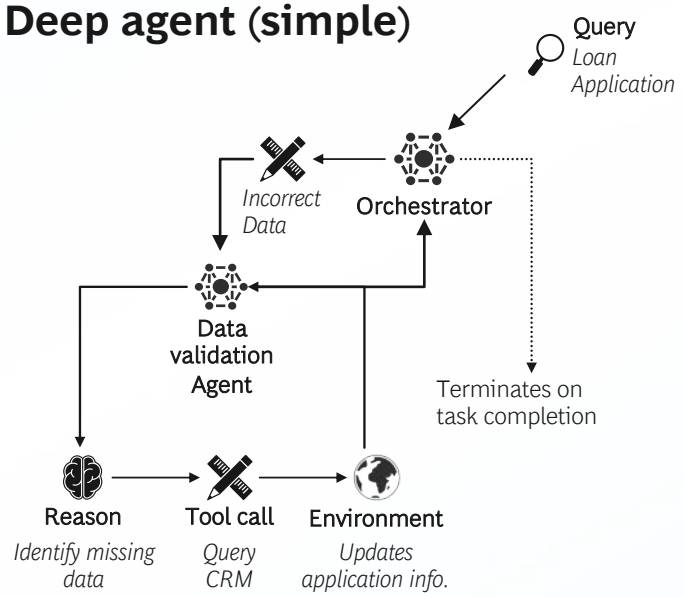
Single agent



(Single reasoning loop, narrowly scoped task)

Start here: reason, act, observe loops until desired output is achieved

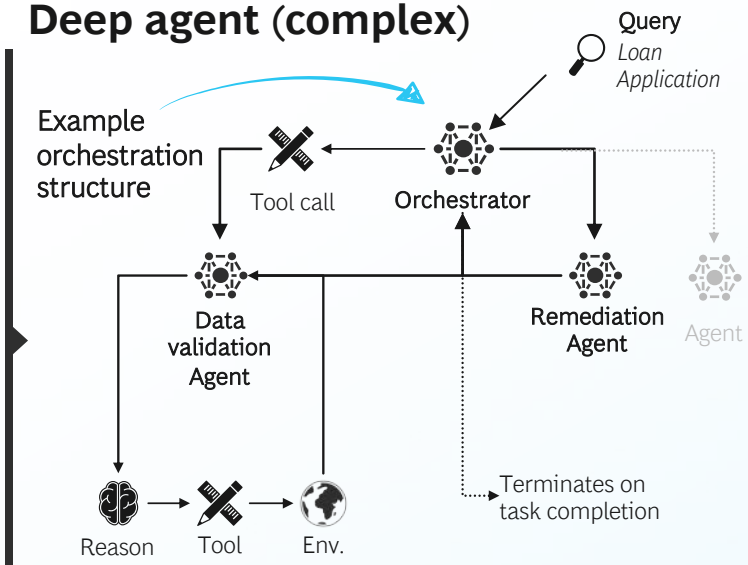
Deep agent (simple)



(Introduces orchestration, structured sub-flows, limited specialization)

Introduce sub-flows if added complexity means brittle outputs or context is lost

Deep agent (complex)



(Multi-agent orchestration, decomposed tasks, specialized reasoning)

Designated specialized agents handle domain-specific tasks

Source: LangGraph; BCG

Design how your agent fits into your workflow to deliver the best user experience

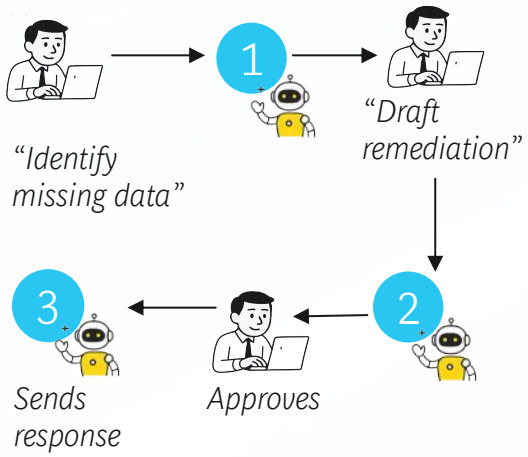
Agent goal: accelerate loan application process

How the agent is initiated is a key design choice; should a user manually trigger actions or can it operate proactively?

Read more on triggers in LangChain's Ambient Agents blog

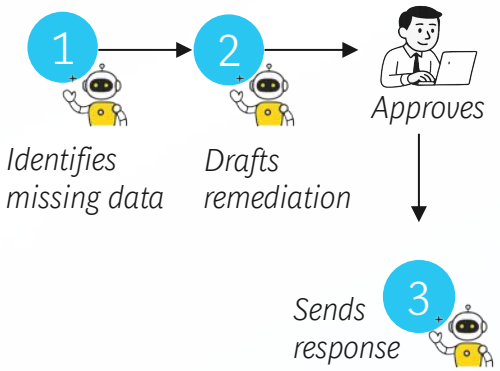
Agent-assisted

Agent provides output of bounded tasks to normal user workflow
e.g., ChatGPT



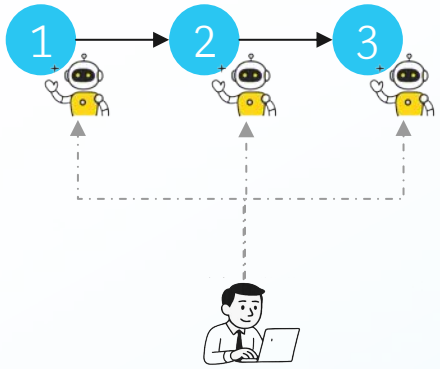
Human-in-the-loop

Agent makes a decision and explicitly awaits human approval
e.g., Claude Code



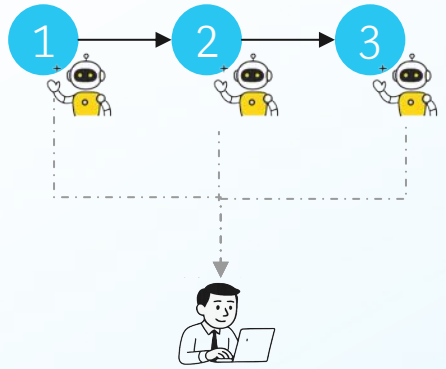
Human-on-the-loop

User observes outputs and can intervene if issues flagged
e.g., Crew AI



Human-out-of-the-loop

Agent acts without any explicit human oversight
e.g., Standalone support agent team

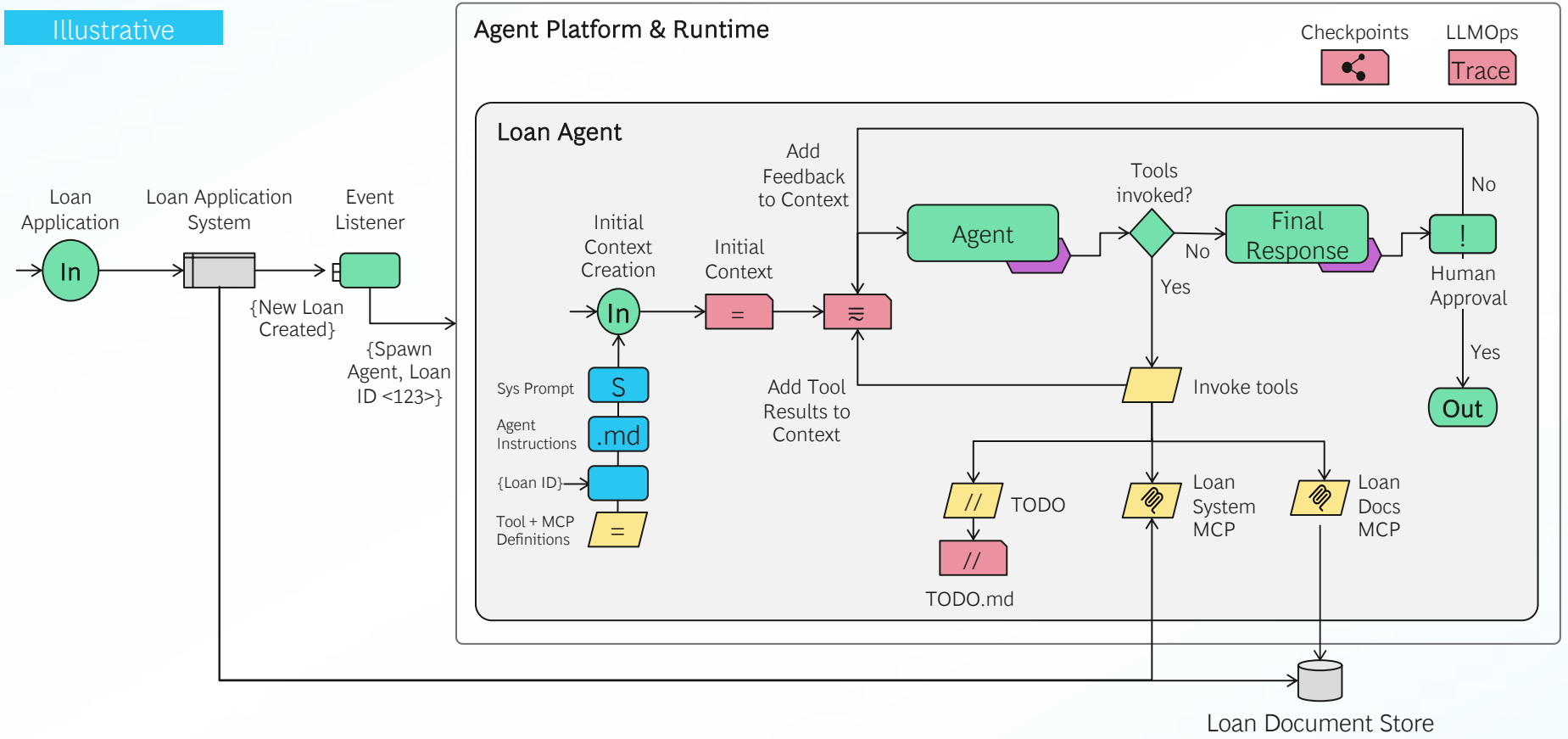


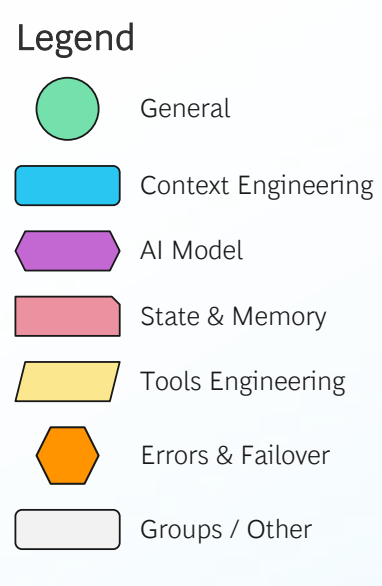
Source: LangChain; BCG

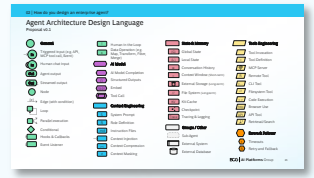
A shared Agent Design Language provides a blueprint for build

A defined framework **standardizes how to describe and document agents,** flexibly describing simple & complex flows

Illustrative







BCG's AI Agent Design Language

Source: BCG

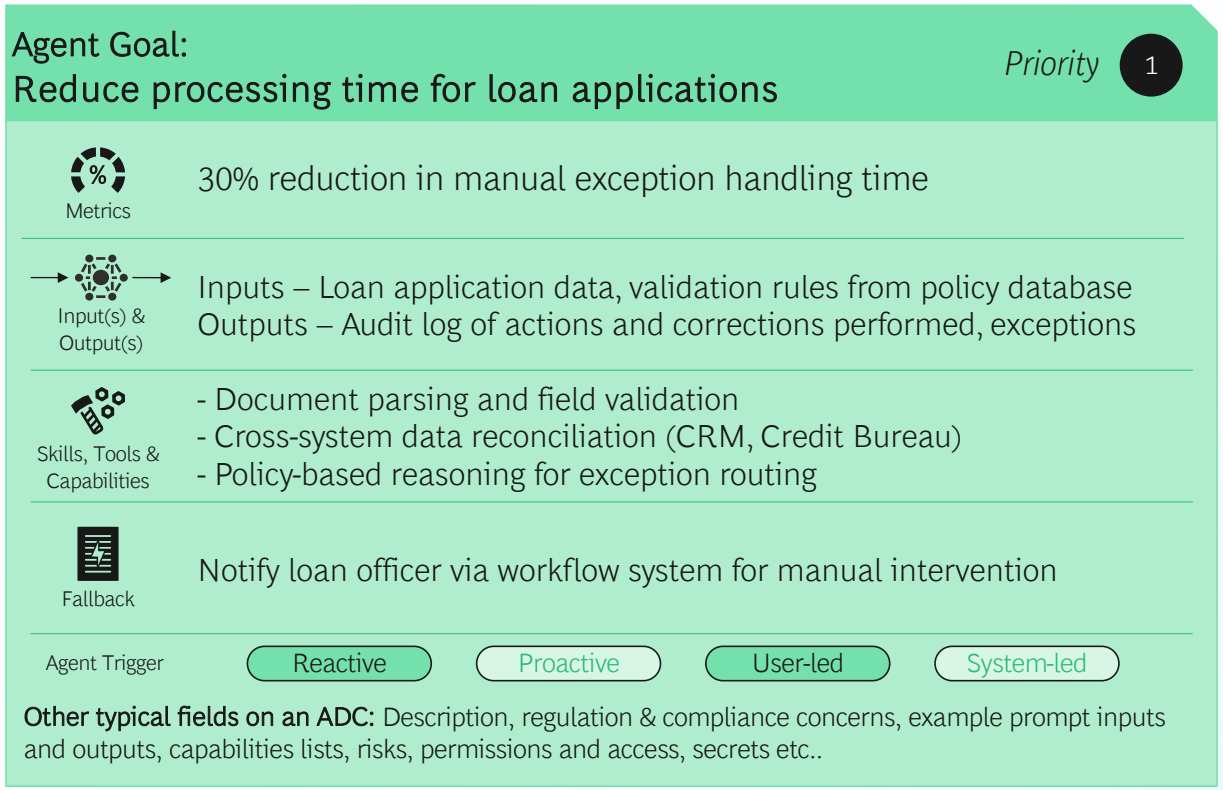
Agent Design Cards align choices to make business targets agent-achievable

A clear charter, defined triggers and agreed levels of human oversight form the foundation of an effective agent design card (ADC)

[Agent-Achievable Goal]
"What can the agent do?"

An effective ADC should:

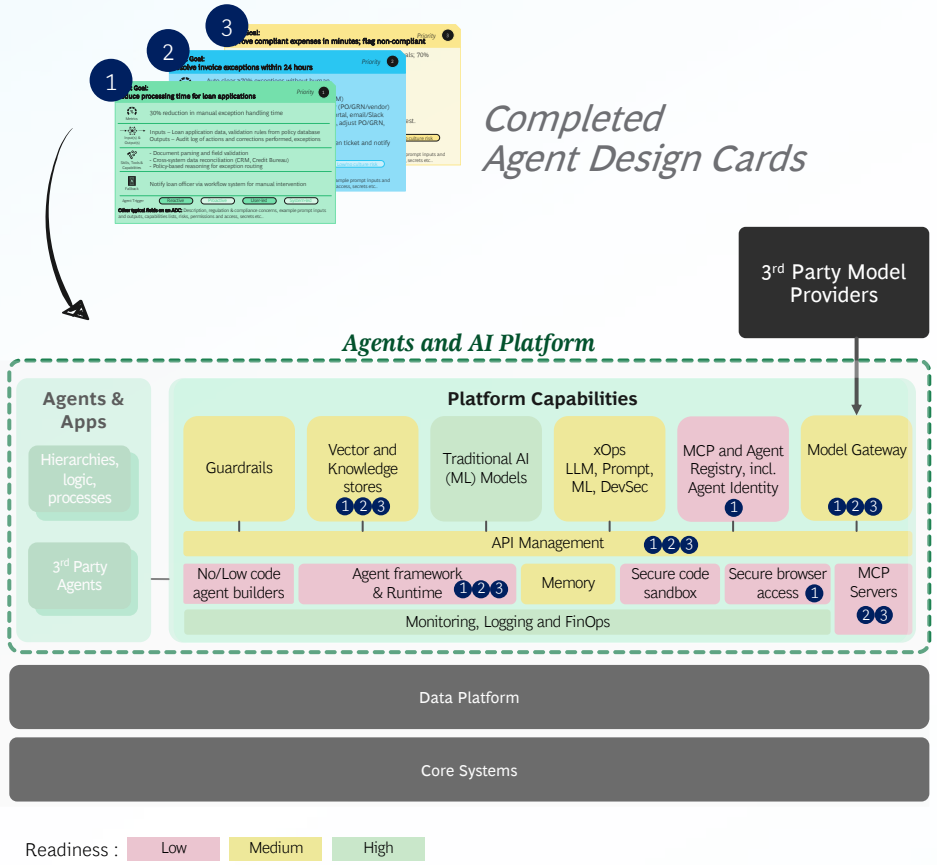
1. **Define purpose:** Clearly describe what the agent is designed to achieve
2. **Clarify boundaries**: specify the agent's role, scope, and points of human oversight
3. **Detail inputs and outputs:** make data sources, dependencies, and deliverables explicit
4. **Describe capabilities:** outline tools and capabilities needed for the agent's success
5. **Anticipate failure:** define fallback behavior, escalation paths, and guardrails



Source: BCG

Agent Design Cards then drive architecture needs that inform required capabilities

Illustrative



Assess your current stack first
Agents should ride on existing rails where readiness is high; platform design focuses on extending *not* replacing

Let design cards drive capability choices
From the cards, pull the *minimal* set you need now, build new tooling only where a card needs it

Prioritize a thin platform MVP
Assess readiness and close gaps in staged development aligned to outcome & risk priorities

Platform design for production
Build capabilities for clear guardrailing & observability faster; issues are diagnosable, and you avoid rework when scaling

No platform-for-platform's-sake, extend selectively as outcomes demand it

Source: BCG

03

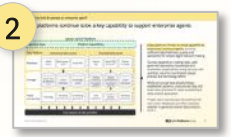
How do you build an enterprise agent?

Building effective enterprise agents – 14 core components

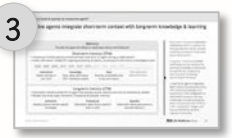
Building effective enterprise agents requires a holistic approach that sustains reliability, safety, efficiency, and compliance amid changing data, models, and business priorities. To avoid duplicate effort, enterprises should embed agent building capabilities into a shared agent platform, ensuring performance & compliance scale by design

1 

Agent Dev Lifecycle: Utilize the ML & SWE dev. lifecycles as starting point for agent building journey

2 

Data platform: Ensure the data platform is ready for agents, with clear structured & unstructured data serving patterns

3 

Memory: Integrate short- & long-term memory to equip agents with context & persistent knowledge

4 

Evaluation: Continuously evaluate and refine agents to improve accuracy and performance

5 

Agent orchestration: Coordinate agentic depth with sub-flows and specialization; consider emerging A2A protocol for frontier use cases

6 

Prompt Tuning & Iteration: Iterate prompts based on feedback & use consistent versioning for stability

7 

Agent Platform Build: Develop hybrid agent platform combining buy, configure & build solutions

8 

Context Engineering: Actively manage the context window so agents access the right context at the right time

9 

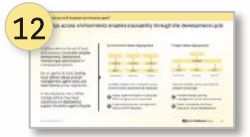
AI Gateway: Use a single AI gateway to enable model switching, efficient monitoring, quality & cost mgmt.

10 

Environment Design: Design the enterprise environment to enable agents to connect efficiently

11 

Low vs. Pro code: Balance low-code speed with pro-code flexibility in agent framework to scale effectively

12 

Enterprise LLMOps: Set up enterprise LLMOps to enable observability in agent life cycle

13 

Failure modes: Proactively address failure modes, using guardrails compliance & performance risks

14 

Regulatory & Compliance: Set enterprise guardrails for security, privacy, and data use

x *Deep-dives*

The agent development journey builds on the ML & SWE development lifecycles

1 | Frame the agent

- Identify **structured and unstructured data sources**
- Define **retrieval logic & context engineering strategy** (i.e., what data, when & how)
- Design the **simplest model-tools-memory-environment loop** to reach the goal
- Define **what KPIs to track & how** (e.g., behavior, output), incl. test data needs

2 | Design agent framework & logic

- Decide on **low vs. pro-code** framework
- Define **workflow triggers, orchestration** and **memory setup**
- Review model options and define **switching methodology & fallback models**
- Decide **runtime flexibility**, i.e., how tools, prompts & memory evolve

3 | Prepare evals & environment

- Design the **tools & actions** the agent can call via MCP, APIs or external systems
- Define **workflow hand-offs and integrations** to enterprise systems & other agents
- Set up **testing harness** and feedback-loop, incl. **human-in-the-loop** check-points
- Define **security, compliance & red-teaming tests**

4 | Engineer policy & prompts

- Establish **system prompts** (e.g., goal, role) & **meta-prompts** (e.g., self-eval, critical thinking)
- Set **control policies** (e.g., token limits) & add **guardrails**
- Establish **prompt versioning**
- Identify **main failure modes & define pass / fail gates** for roll-out

5 | Test & tune the agent

- **Automate testing** of every change
- Establish **runtime sandbox & version control mechanisms**
- Define **component baselines** for runtime & orchestration
- Define **autonomy tiers**; shadow, supervised, guided & full autonomy[1]
- Define **steps for gradual roll-out**, incl. shadow & canary tests

6 | Launch, monitor & evolve

- Embed defined feedback & eval loops to **enable learning & evolution over time**
- Introduce **retraining triggers** & track drift across components
- **Continuously reduce complexity**, e.g., simplify prompts & prune unused tools

1. Shadow mode: agent suggests, human acts; Supervised mode: agent acts, human approves; Guided autonomy: agent acts, human monitors; Full autonomy: agent acts independently

Source: BCG

Agents across the enterprise can co-exist on different platforms

Environmental complexity*

Standalone Agentic Solutions

- Turnkey agents with prewired orchestration
- SaaS-hosted, vendor-managed runtime
- Lowest integration and ops overhead
- Limited extensibility beyond app boundaries

Embedded Agentic Platforms

- Integrated inside enterprise suites and workflows
- Orchestration vendor-managed within host platform
- Moderate integration leveraging existing governance
- Configuration over customization, faster scale in-suite

Agent Builder Platforms

- Toolkits to construct and orchestrate multi-agent applications
- Higher engineering lift and ecosystem integration
- Rich connectors, policy, and lifecycle controls
- Scales via cloud services or low/no-code builders

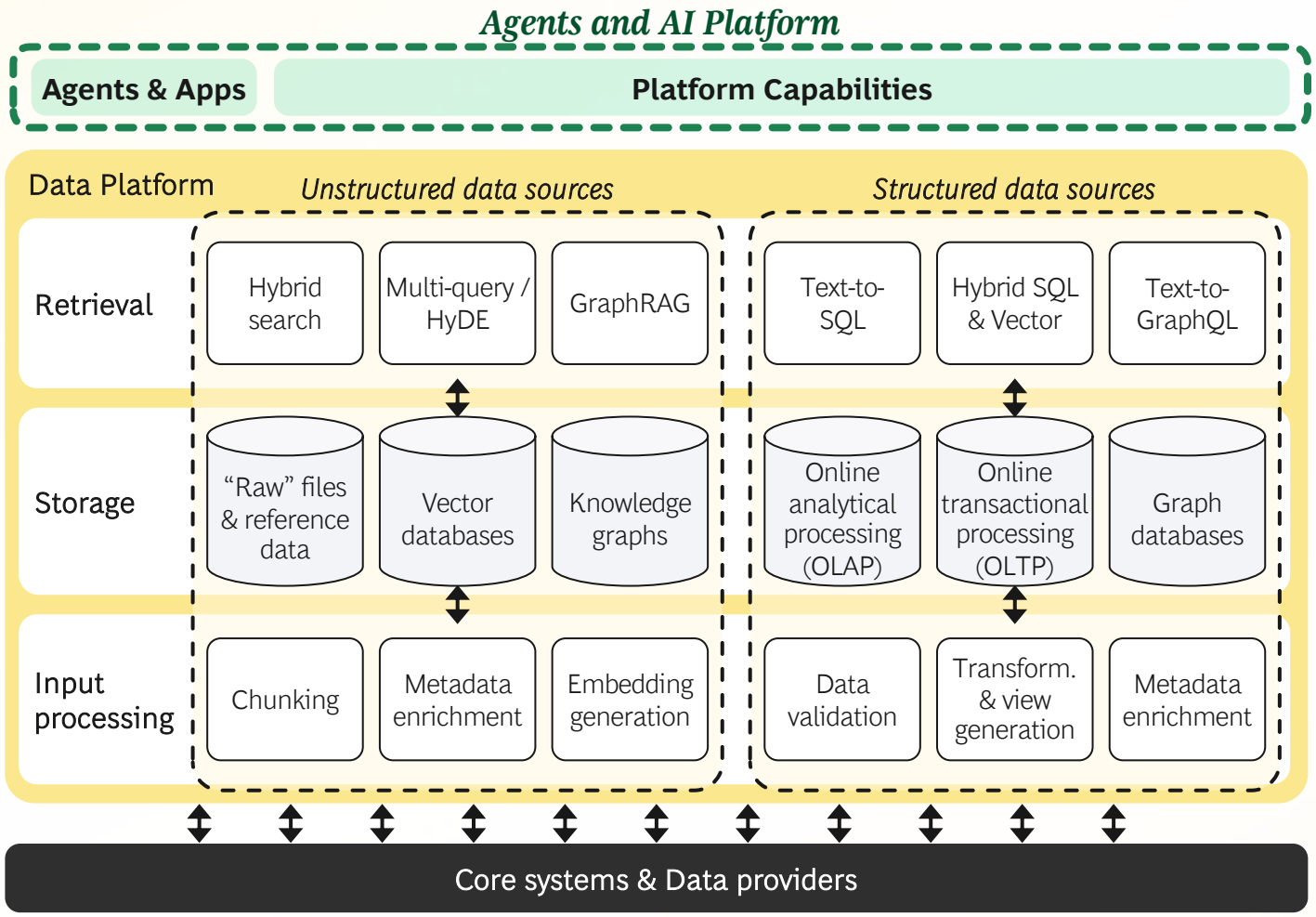
Custom-Built Agent Platforms

- Orchestration authored from scratch in code
- Highest environment complexity and operational burden
- Full control of dataflow, guardrails, and cost
- Requires strong MLOps, platform, and app teams

**Environmental complexity refers the number of moving parts and how tightly they are coupled in the ecosystem an agent must navigate to achieve a goal*

Source: BCG

Data platforms will evolve to serve the needs of agents in the enterprise



Data platforms remain a critical capability as enterprises introduce agents, ensuring sufficient data freshness, quality and availability for reliable agent decision-making

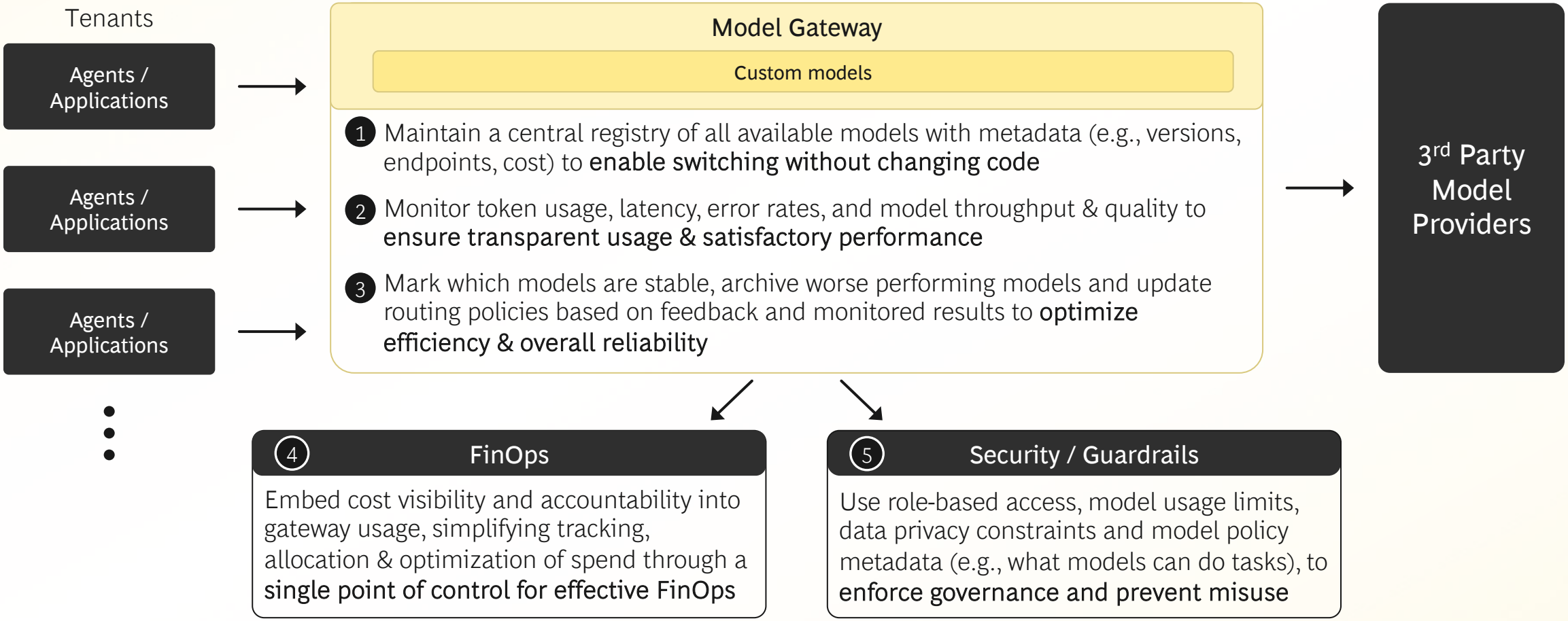
Success depends on making clean, well-governed data easily discoverable and accessible, supported by strong security and auditing, requiring coordinated people, process and technology efforts

While structured data already follows established patterns, unstructured data still lacks clear practices for data accessibility & data product generation

Project teams should start by building their own vector databases, and then evaluate whether to generate shared data products from it

Source: BCG

Unified AI gateways allow secure, observable and scalable model serving



Source: BCG

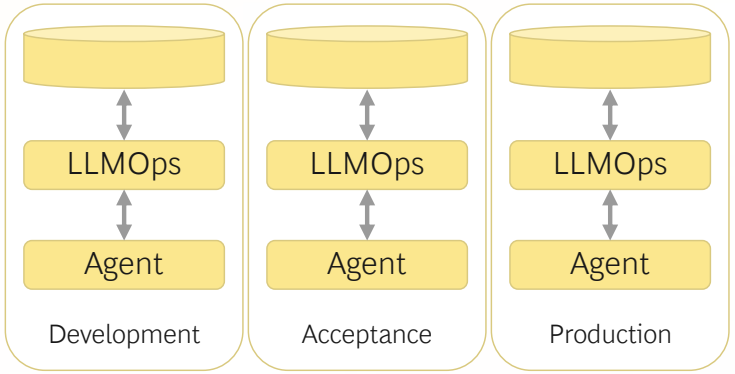
Enterprise LLMOps tooling delivers traceability through the agent life cycle

LLMOps[1] refers to the set of tools and practices that **enable reliable development, deployment, monitoring & optimization** of LLM-powered systems

For an agentic AI build, **tooling must deliver robust prompt management, agent evals and observability** across trajectories

In the enterprise, the LLMOps strategy defines how these capabilities are **deployed to support the entire agent lifecycle**

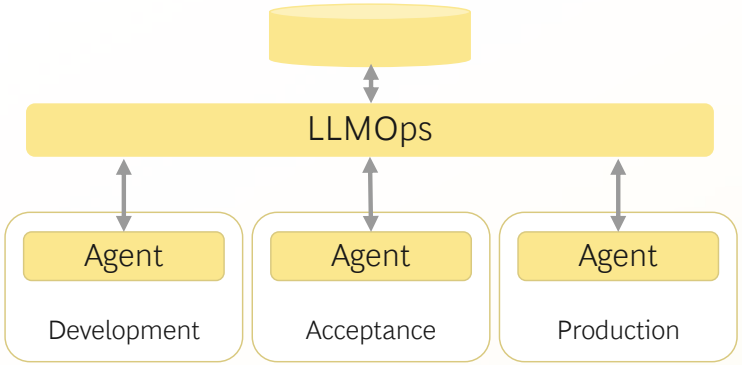
Environment level deployment



LLMOps deployment is isolated across environments (application & database)

- ⊕ Faster experimentation in early dev. with low governance overhead
- ⊖ Fragmented prompt management tied to CI/CD lifecycles

Project level deployment

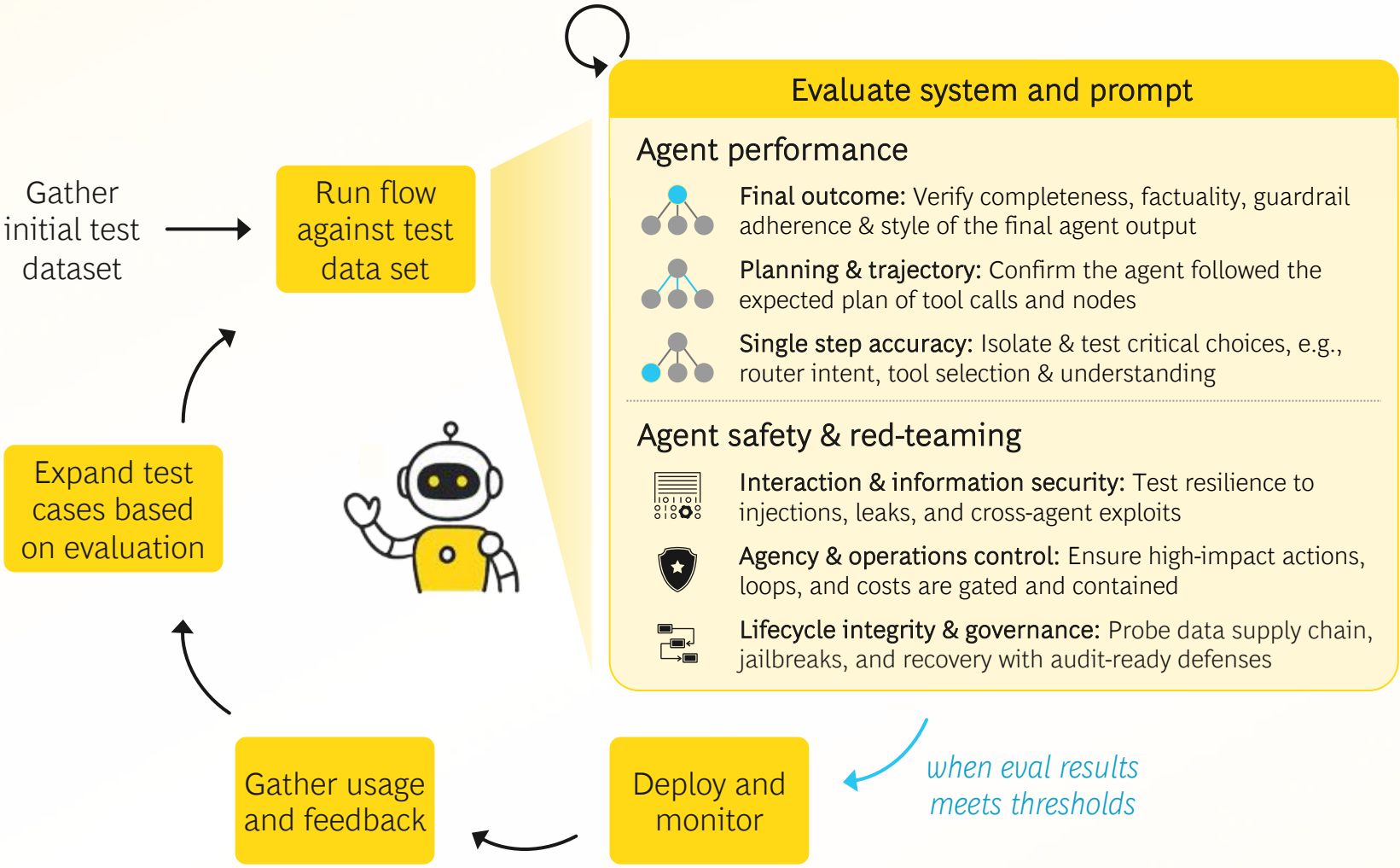


LLMOps tooling sits across project environments (shared database)

- ⊕ Central management reduces operational overhead at scale
- ⊕ Provides holistic prompt management and versioning

1. Large Language Model Operations
Source: BCG

Setup eval harnesses early to hill climb[1] on agent performance from the get-go



Build or leverage an evaluation / test framework to validate the quality of the system and prompts

Requires **upfront and continuous gathering** of test examples, generating meaningful synthetic data can be a struggle

Need a **sandboxed development environment** enabling easy use of agent dev with read only production data

Evaluation techniques include:

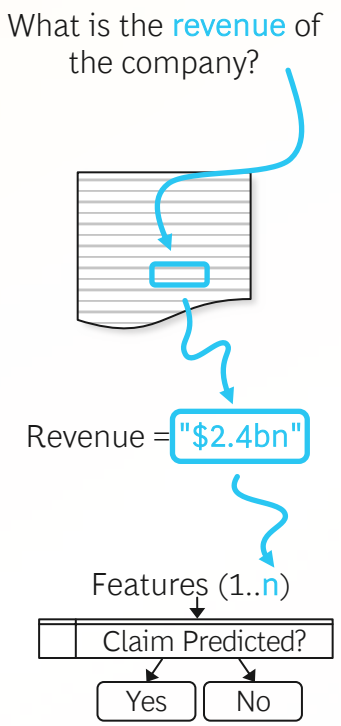
- (Fuzzy) matching
- Model graded eval (LLM-as-judge)
- Heuristics
- Human evaluation
- Subsequence and step-diff checks

1. Hill climbing in AI terms means improving agent performance on a particular task
Source: BCG

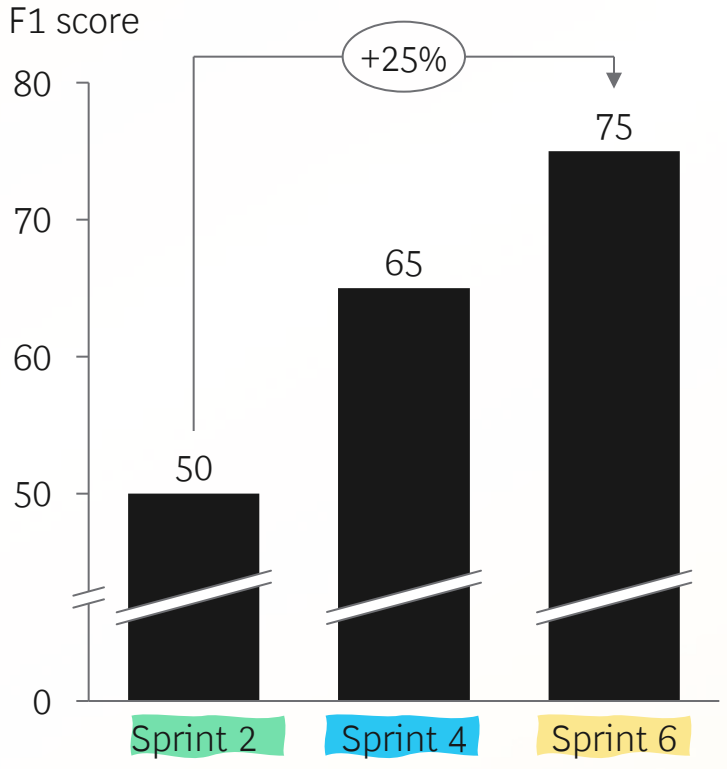
Example | Setting up a testing harness is fundamental to measure improvement

Entity extraction is about finding a specific data point from a document set…

1. Describe the entity you want extracted
2. Ask an LLM to find the entity in a large document (>10k words)
3. The LLM returns what it thinks is the answer
4. Feed the answer into TradAI to predict claims better – higher accuracy = more $$



… in an insurance client we achieved great precision and recall[1] in only six Sprints with testing harness setup



	Prompts	RAG	Tools	F1[2]
		Context engineering		Target outcome
Sprint 2	Zero-shot, naive	No	No	~50
Sprint 4	Zero-shot, iterated	Yes, naive	OCR	~65
Sprint 6	Few-shot, tailored by entity	Yes, tuned chunk size & params	Tuned OCR	~75

Average F1 of 75 translated to $ million top line impact

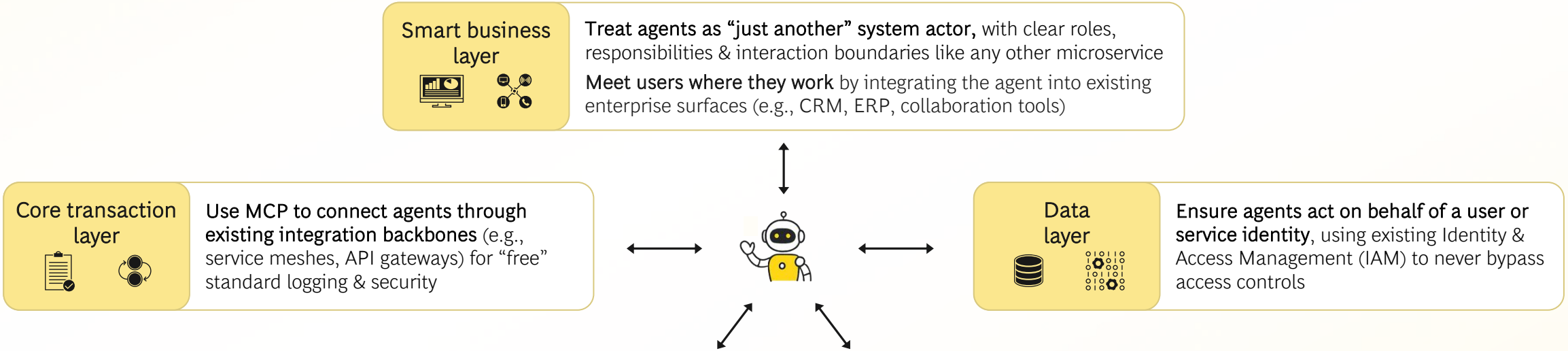
1. F1 score for entity recognition task from BCG case experience – F1 score averaged across an 8 features extracted from PDFs for underwriting
2. F1 is a proxy for accuracy, and in this case was directly correlated with monetary value – higher F1 == more profit from insurance premiums

Source: BCG

Ensure enterprise environment readiness by addressing agent integration barriers

Integrating with systems is hard. Vendor tech looks great on paper, but real-world issues persists including security gaps, scaling limits, latency & network complexity, and many tools are still lacking maturity and battle testing

Key activities can ease the process of agent integration



Source: BCG

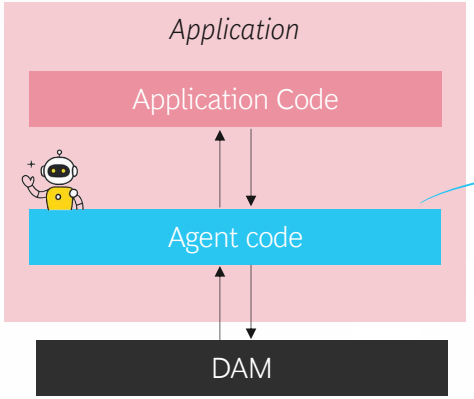
04

How do you assemble an agent platform?

Agents and the platforms they live on are decoupling over time

Emergence of tightly coupled agents
2023-24

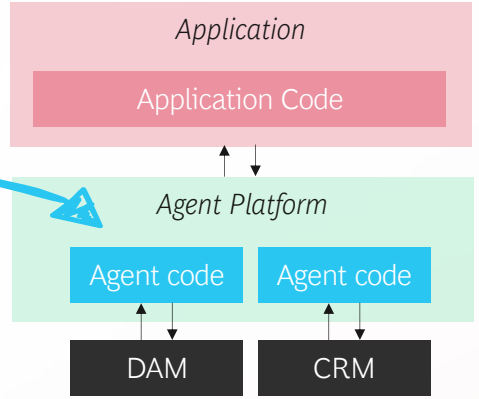
Agents built directly into existing apps, with hardcoded workflows & system-specific integrations, limiting adaptability



Code, data and deployment in the same stack **constrains scalability and reuse**

Progression to decoupled agent platforms
2025

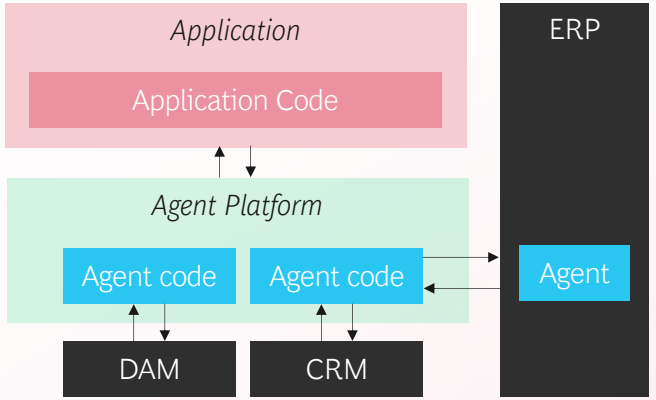
Rise of agent platforms; agent logic and orchestration separate to existing backend systems to facilitate reuse and scaling



Decoupling **offers modularity and flexibility** breaking down integration barriers

Rise of agent interoperability across platforms
2026+

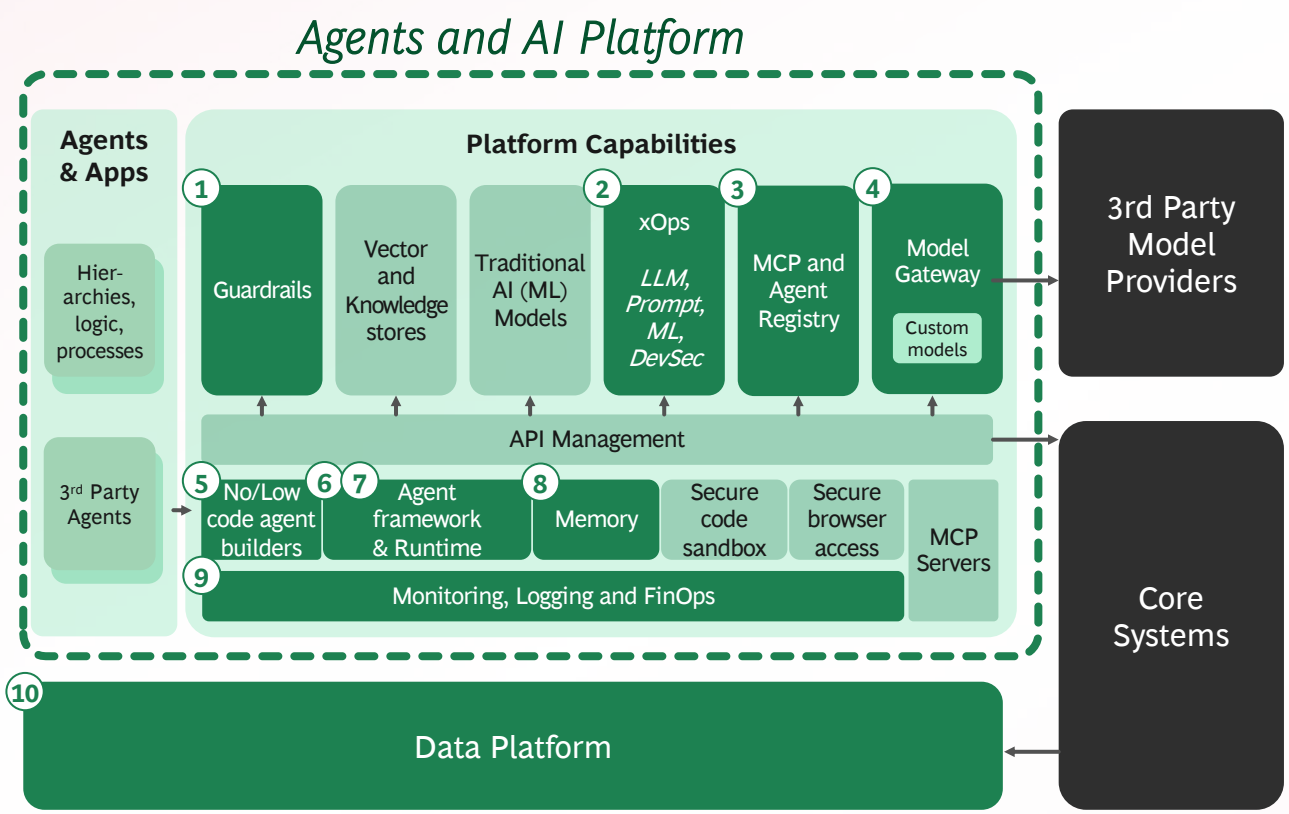
Next-generation architectures may mix agents across platforms, with shared protocols **allowing for cross-platform communication**



Hybrid architectures enable **interoperability[1], adaptability and connected agent ecosystems**

1. Interoperability refers to cross-platform orchestration
Source: BCG

Agent & AI Platforms provides the foundation to build agents within an enterprise



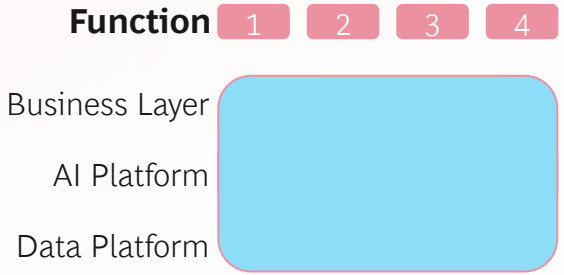
1. **AI Guardrails** | Provide guardrails as a service to all GenAI apps
2. **LLMOps** | Delivers robust prompt lifecycle management, agent evaluation, and observability
3. **MCP & Agent registry** | Store & make MCP server / tool definitions and agents available in one place
4. **Model Gateway** | Unified access layer for model endpoints to the rest of the AI platform (abstracting away quotas & scaling)
5. **No/low code agent builders** | Enables rapid agent creation and customization through visual UI and drag-&-drop components
6. **Agent Framework** | Coordinates multi-agent workflows with planning, selection & policy-based routing
7. **Agent Runtime** | Manages the agent runtime, session state, and enforces runtime policies
8. **Memory** | Provides the agent with the ability to recall past actions and behaviours
9. **Monitoring, Logging, and FinOps** | Continuous traceability, cost visibility, and evaluation to manage prompts, outputs & spend
10. **Data Platform** | Structured and unstructured data sources agents have access to

Source: BCG

Enterprises will converge on a hybrid approach; no one-size-fits-all agent platform

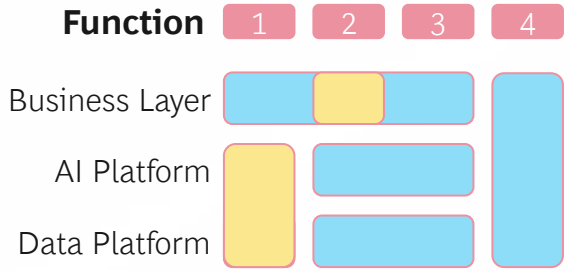
Complexity scales with custom, targeted solutions

Unified platform



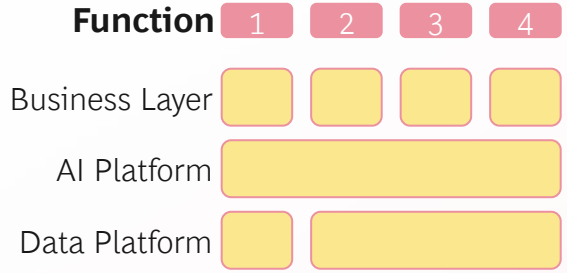
Single vendor for all layers across functions offers **fast deployment** with minimal integration effort, but **limited adaptability and strategic depth**

Hybrid platform



Hybrid platform foundations with targeted add-ons, ensuring **balanced flexibility, good integration readiness**, and scalability with governance

Custom, modular platform



Specialized modules orchestrated across platform, offers **high differentiation but greater complexity**; needs robust integration & lifecycle management

Custom | Vendor

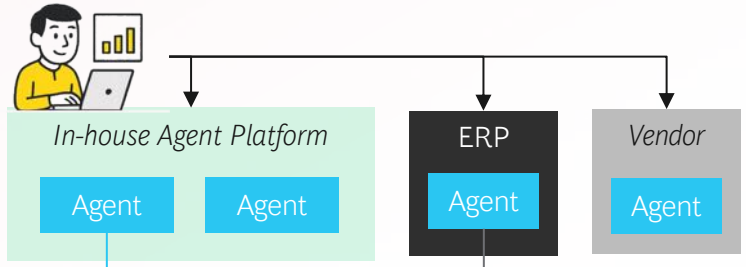
The extremes define a range; effective platforms adapt to functional needs across the enterprise

Source: BCG

As agent ecosystems grow, structure becomes the key to sustainable scale

Enterprise Orchestration

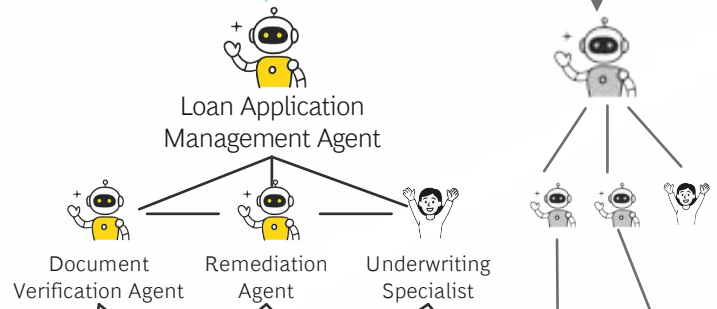
Governance practices across teams & platforms



Agents distributed across vendor and enterprise platforms require **unified orchestration, governance, and lifecycle management** with embedded traceability, audit, and policy guardrails

Domain Orchestration

Operational collaboration between people & agents



As agents become embedded in workflows, cross functional teams are key to own the entire agent lifecycle and **enforce agent identity, purpose, and decision boundaries across platforms**

Data & Tool Landscape

Foundational access to systems & context

Build and reuse shared tools and connectors to avoid duplication - **version, monitor, and retire agents and tools systematically**, treating them as evolving products to minimize future tech debt

Source: Amazon Web Services; BCG

Default to off-the-shelf platform solutions unless driving differentiating impact

! Review existing capabilities before considering new: *prioritize re-use where possible*

1 Differentiation & environment complexity:

- Is the (set of) use cases critical to competitive differentiation?
- Is the environmental complexity too high for a standard agent?

If not, Buy → **Adopt**

If not, Buy & configure → **Adapt**

2 Execution:

- Do we have the right capabilities to build? *(E.g., flexibility, costs, speed to launch, fit with existing capabilities, in-house engineering capabilities)*

If yes, Build → **Assemble**

3rd party vendors

Adopt

Third-party SaaS vendors are increasingly providing solutions for commodity tasks, making adoption of out-of-the-box apps easy without need for adaptation
e.g., A unified platform to build, deploy, and manage AI agents that streamline and automate business workflows

Adapt

If limited in-house resources, third-party vendors are preferred for stronger functionality, support & application management, with adaptability possible within their app
e.g., Addition of custom workflows and tools on a SaaS orchestration platform

In-house

Assemble

When customization need & environment complexity is high and in-house resource are available, building internally is preferred to drive differentiation, leveraging proprietary data, enterprise platforms & internal expertise
e.g., Internally built agent platform with domain-specific setup, enabling full control over agent logic, optimization & evolution

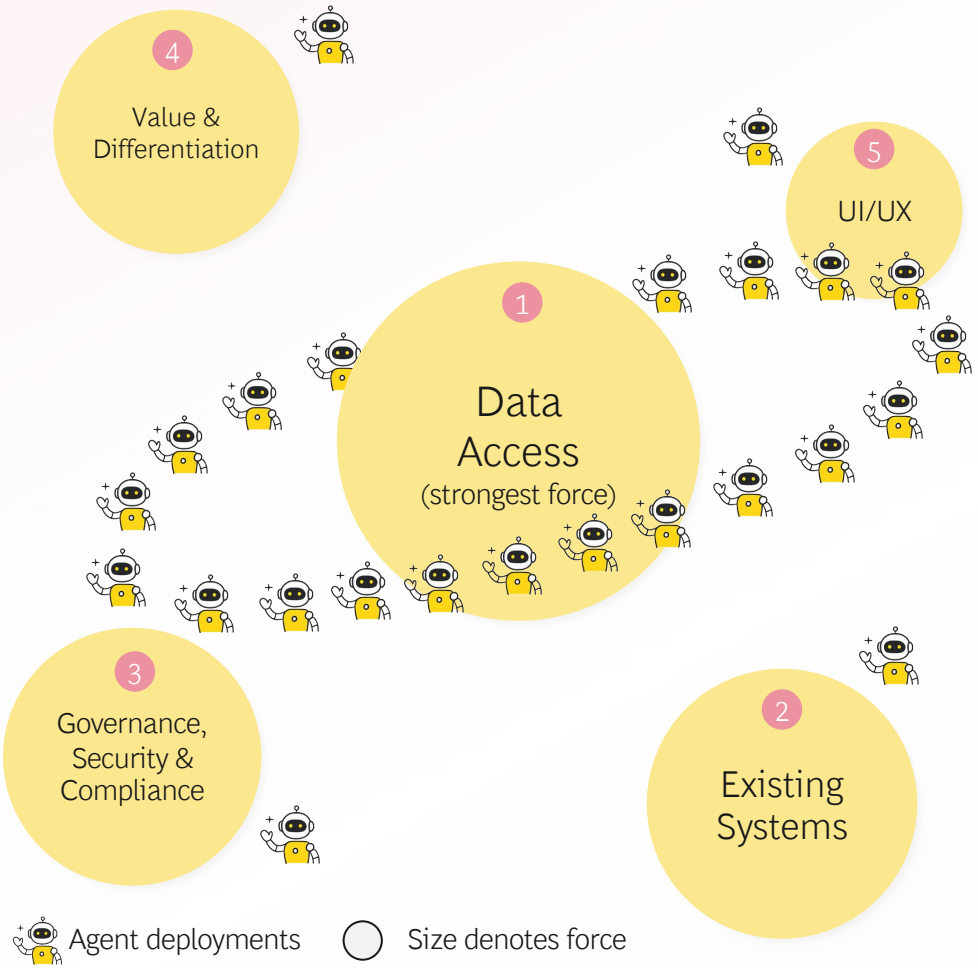
Buying = "Turnkey, fast, but narrow": Best for speed, adoption, and compliance inside core systems of record

Built = "Flexible, portable, but heavy": Best for bespoke, cross-enterprise orchestration or to avoid vendor lock-in

Hybrid is inevitable: most enterprises will buy agents embedded in major suites and build orchestration for cross-cutting workflows

Source: BCG

Choosing the platform for the agent should consider gravity factors & constraints



1. **Data Gravity |** strongest pull; agents must sit where enterprise data lives to be useful (proximity, sovereignty) moving data adds latency, fragility, and expense, making proximity essential (sovereignty, friction, overhead)
2. **Systems Gravity |** legacy ERP, CRM, and productivity platforms anchor where agents can operate (lock-in, inertia)
3. **Governance, Security & Compliance |** platforms must fit enterprise controls & requirements, and enable manageable shifts in people, process, and security needed for agent adoption (risk, auditability)
4. **Value & Differentiation |** pull strengthens when platforms deliver clear ROI, lower TCO & integration costs, faster time-to-value, and measurable impact; amplified when agents enable unique capabilities competitors can't replicate (efficiency, advantage)
5. **UI/UX complexity |** adoption is fastest when agents embed in the tools people already use every day (familiarity, habits)

Importance

Source: BCG

Key takeaways for building effective enterprise agents

1. **Design for outcomes, not outputs**
   Anchor every build on measurable business outcomes. Break high-level goals into agent-achievable objectives to ensure value creation and alignment with enterprise priorities

2. **Start simple and iterate with eval driven design**
   Begin with a single observe–reason–act loop. Instrument evaluation and feedback loops early to hill climb on accuracy and performance, enabling safe, data-driven increases in complexity

3. **Build on shared enterprise foundations**
   Standardize around common components: agent runtimes, model gateways, guardrails, observability, and FinOps; to improve reusability, reliability, and time-to-scale across teams

4. **Choose the right platform for your agent**
   Select between embedded, agent builders, or best-of-breed based on data and system gravity, governance needs, and differentiation value, not convenience or hype

5. **Engineer trust, compliance and resilience by default**
   Integrate strong identity, access control, monitoring, and evaluation to ensure safe operations, explainability, and long-term compliance as agents scale enterprise-wide

Looking ahead

Enterprises that master evaluation, governance, and architectural discipline will turn the rapid AI evolution into sustained competitive advantage

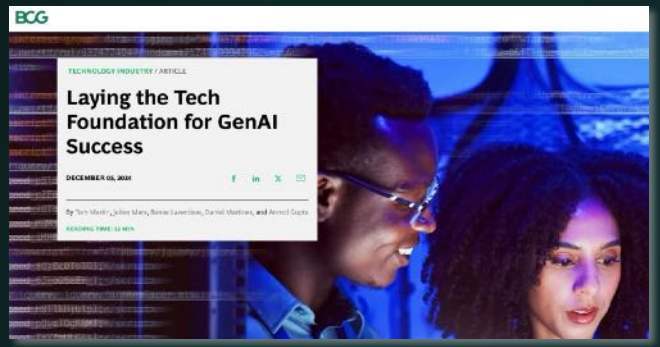
The pace of progress will reward organizations that build systems capable of scaling, and interfacing with their enterprise landscape with ease, improving in lockstep with the models that power them

While 2025 brought a wave of experimentation with agents but limited enterprise value, 2026 will be the year they are put to work to deliver real value

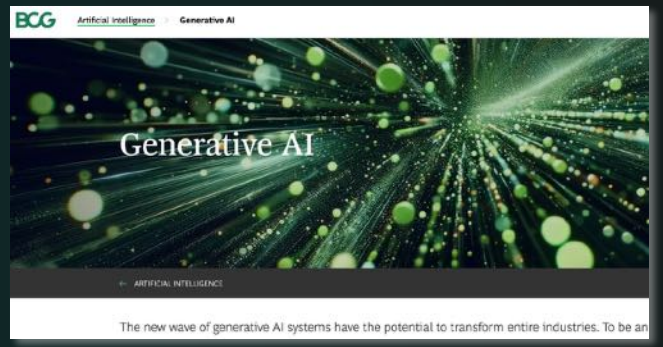
Source: BCG

Read more of BCG's perspectives

[Tech foundation for GenAI success]



[AI on BCG.com]



[Latest thinking on Agents]

[Our Executive Perspective Series]

Get in touch with our AI team

Co-Authored this paper


Vladimir Lukic


Nicolas De Bellefonds


Gene Sheenko


Djon Kleine


Tom Martin


Julien Marx


Jeffrey Walters


Julian King


Matthew Kropp


Dan Sack


Niels Degrande


Daniel Martines


Assaf Tayar


Darshana Thakker


David Heurtaux


Caitlin Barber


Mathilde M. Solberg


Doug Newton

Disclaimer

The services and materials provided by Boston Consulting Group (BCG) are subject to BCG's Standard Terms (a copy of which is available upon request) or such other agreement as may have been previously executed by BCG. BCG does not provide legal, accounting, or tax advice. The Client is responsible for obtaining independent advice concerning these matters. This advice may affect the guidance given by BCG. Further, BCG has made no undertaking to update these materials after the date hereof, notwithstanding that such information may become outdated or inaccurate.

The materials contained in this presentation are designed for the sole use by the board of directors or senior management of the Client and solely for the limited purposes described in the presentation. The materials shall not be copied or given to any person or entity other than the Client ("Third Party") without the prior written consent of BCG. These materials serve only as the focus for discussion; they are incomplete without the accompanying oral commentary and may not be relied on as a stand-alone document. Further, Third Parties may not, and it is unreasonable for any Third Party to, rely on these materials for any purpose whatsoever. To the fullest extent permitted by law (and except to the extent otherwise agreed in a signed writing by BCG), BCG shall have no liability whatsoever to any Third Party, and any Third Party hereby waives any rights and claims it may have at any time against BCG with regard to the services, this presentation, or other materials, including the accuracy or completeness thereof. Receipt and review of this document shall be deemed agreement with and consideration for the foregoing.

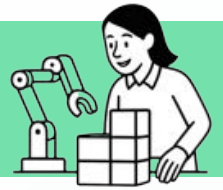
BCG does not provide fairness opinions or valuations of market transactions, and these materials should not be relied on or construed as such. Further, the financial evaluations, projected market and financial information, and conclusions contained in these materials are based upon standard valuation methodologies, are not definitive forecasts, and are not guaranteed by BCG. BCG has used public and/or confidential data and assumptions provided to BCG by the Client. BCG has not independently verified the data and assumptions used in these analyses. Changes in the underlying data or operating assumptions will clearly impact the analyses and conclusions.

Technical Appendix

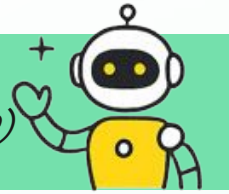
The anatomy of the Enterprise Agent; 5 systems at work

System 1: The User and Agent Experience
How humans and other agents interact with our agent, through apps, APIs, MCP servers and other protocols

System 2: The Agent "Environment"
Internal and External resources, services, and tools, what the agent can see and do



System 3: The Agent "Policy"
The control flow that guides the agent's actions and behavior, maps observations (context) to actions (tool use)



System 4: The Agent "Runtime"
The platforms agents live in, how they are served, scaled, and integrated in the enterprise

System 5: The Agent "Operations"
The monitoring, logging, observability, security, and lifecycle of all the other 4 systems

The anatomy of the Enterprise Agent; 5 systems at work

System 2: Agent "Environment"

Browser Use	APIs	Databases	Terminal Use (Sandbox)	Computer use	Code execution (REPL)	MCP servers	Other Agents
Tool registries				Env Playground		MCP registries	A2A registries

System 3: Agent "Policy"

LLMs SFT/RL FT	Control flow / context engineering	Threads	Input	Tool parsing	Output	Memory

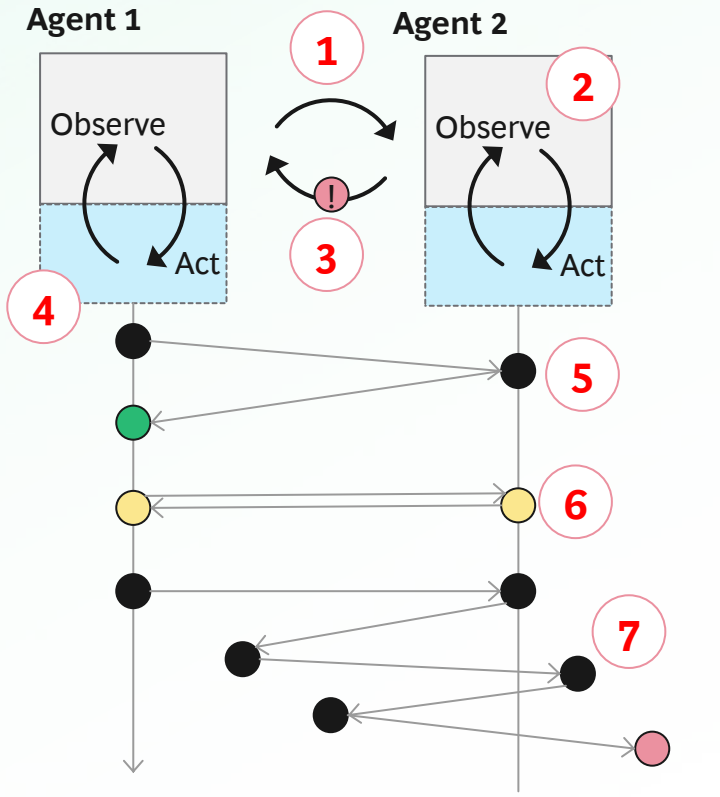
System 4: Agent "Runtime"

API Manager	Endpoints	Hosting	FinOps	Secrets Mgmt.	Model Gateway	Guardrails	AuthN + AuthZ

System 5: Agent "Operations"

PromptOps	Version Ctl.	Testing	Evals	Observability	Monitoring & Logging	SIEM Integration	Service Mgmt.

Why? Multi-agency remains not only a real technical challenge…



Fundamental computer science breakthroughs are needed to overcome these challenges – not just more prompting

1. **Context sharing & goal alignment**
   How to sharing effective context at scale is not solved, neither is effective goal alignment between distinct agent policies that don't have explicit hierarchy
2. **Coordination <> complexity problem**
   Building reliable agents typical means static environments (deterministic behavior) – when environments include other agents, reliable policies becomes harder to achieve
3. **Conflict Resolution**
   Without explicit hierarchy or rules, agents with conflicting proposals will get stuck
4. **Long range planning & memory**
   LLMs have "retrograde amnesia", don't learn dynamically and require tools for multi-session recall. In a multi-agent context, different interactions means different threads.
5. **Credit assignment – you can't improve what you can't measure**
   Following successful outcomes, it's not always easy to tell which agents contributed what to the outcome, making policy evaluation and improvement difficult
6. **Getting stuck in loops (local minimum)**
   Agents can frequently get stuck in local loops, requiring intervention or restart
7. **Task Drift**
   Agents can drift on task – resulting on loss of the original intention. More post training on task following and improved long context performance will help with task drift

Source: BCG

Goal decomposition makes prioritized outcomes achievable for agents

Breaking outcomes into **strategic, tactical, and agent-level goals** connects intent to execution; turning abstract objectives into clear, agent-achievable design targets helps to **steer subsequent design decisions before build can begin**

[Outcome]
"Why does this matter?"

▼

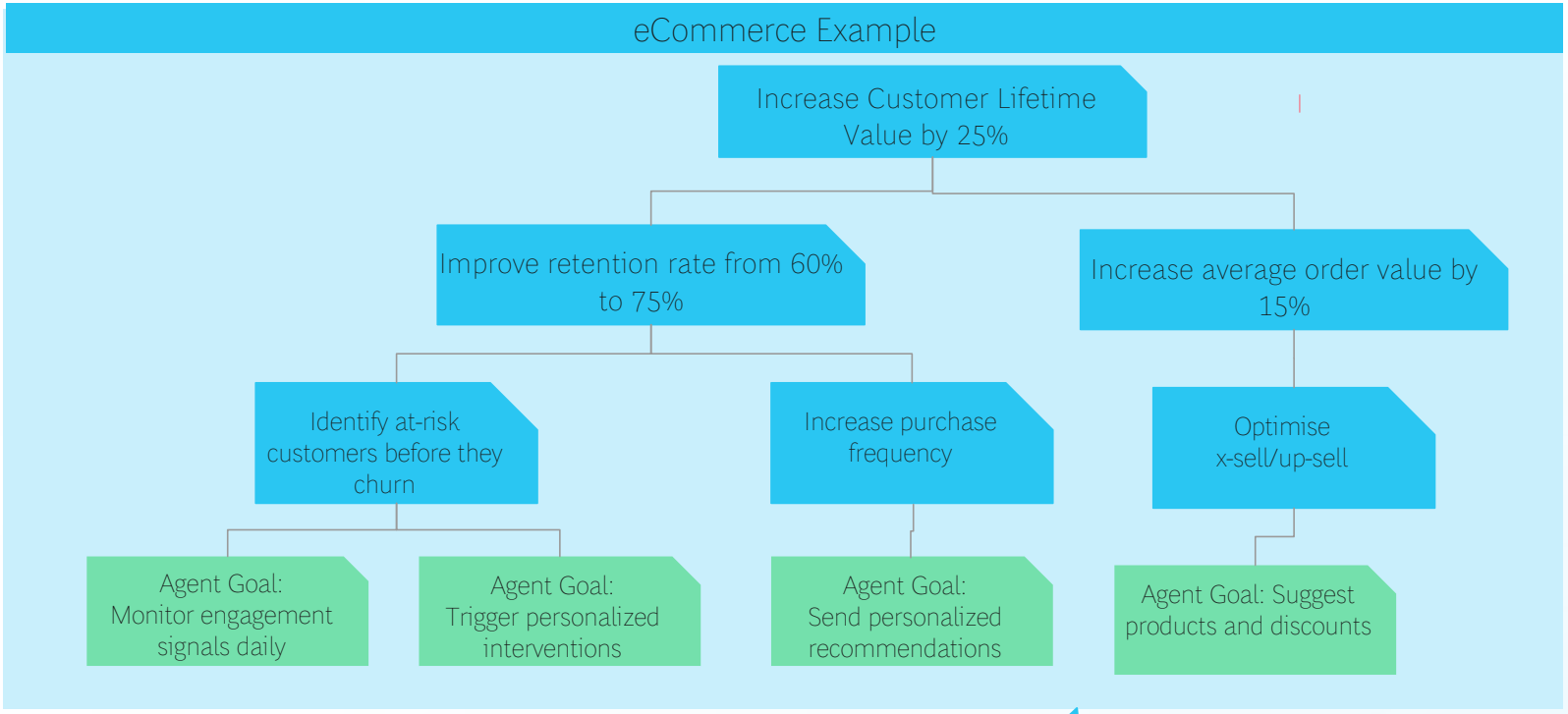
[Strategic Goal]
"What would achieve this?"

▼

[Tactical Goal]
"What specifically?"

▼

[Agent-Achievable Goal]
"What can the agent do?"



! This is a design tool, not an agent blueprint; *more on agent-to-agent communication in Chapter 3*

Source: BCG

Make a deliberate interaction choice to meet the user's needs in their workflow

Read more on triggers in LangChain's Ambient Agents blog

Timing
When the interaction starts

Context origin: How the interaction starts	Reactive: Agent responds only to specific, explicit trigger	Proactive: Agent initiates action without explicit trigger
User-led *Context provided by human* *(e.g., chat interface, voice, button click, co-pilot)*	User Asks & Agent Responds *User opens chat: "Draft an email to HR" Agent: Generates draft that the user reviews and sends*	User Acts & Agent Observes *User types in calendar: "Flight tomorrow at 6pm" Agent: Suggests taxi booking to airport → user accepts*
System-led *Context shared by design* *(e.g., webhook, cron job, API call, another agent)*	System Triggers & Agent Responds *CRM webhook: New high-priority ticket arrives Agent: Assigns to right team + prepares first response*	System Changes & Agent Observes *Agent monitoring customer data: Detects churn signals Agent: Flags at-risk account + drafts outreach playbook*

Source: LangChain; BCG

Choosing which agent platform for a given scenario will be driven by differentiation

	Standalone Agentic Solutions	Embedded Agentic Solutions	Agent Builder Platforms	Custom-Built Agent Solutions
BCG dev. framework	**Deploy** Reshape Invent	**Deploy** Reshape Invent	**Deploy** **Reshape** Invent	**Deploy** **Reshape** **Invent**
When to choose	When there is a need for a ***fast, narrow capability for one team with minimal integration***—clear vendor fit, light governance	When a major suite is already in use and the enterprise wants ***in-suite agents leveraging native data, workflows, and governance***	When there is a ***need for one governed low/no-code builder*** for broad citizen dev, reuse, and consistent runtime/ops	When the ***use case is differentiating*** and needs bespoke logic, heavy orchestration, strict controls, or hard SLAs
Example tech choices	Adobe Firefly for creative, Cursor for coding …	Salesforce Agentforce, SAP Joule Agents …	Copilot Studio and Power Platform, UI Path …	Build agents where you you can differentiate yourself using open source framework and cloud technologies

Source: BCG

Combine context engineering strategies to prevent context pollution

Store context outside the context window

- Enable agents to take notes during a session / task in a file or runtime object to **structure longer tasks**
- Use external memory to enable agents to store & retrieve **relevant information across sessions / tasks**

Optimize selection & retrieval timing

- Allow agents to read notes or state object fields to **simplify handovers & resumptions**
- Agents use memory to recall curated facts, instructions & example behaviors as **guidance**
- Define minimal tool sets & apply RAG to fetch only **most relevant tools**
- Inject context dynamically during runtime to **leave room for exploration**

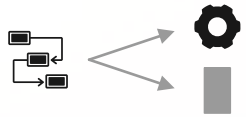
Compress context over time

- Summarize existing context as window nears its limit or at set points of sessions, to **only preserve key information**
- Prune old or irrelevant content using heuristics to **avoid conflicting context elements**
- Implement a ranking step to ensure agents see the **most relevant information**

Isolate context into separate containers

- Split task & context across sub-agents, enabling each to **use their full context window to handle a sub-task**
- Run heavy processes in isolated environments to **spare the context window for heavy processing**
- Define context fields in the runtime state object to enable **selective exposure of context to the agent**

Actively manage workflow impact



- Break workflows into clear steps to enable agents to **focus the context window on a clear, narrow task**
- Design simple, structured prompts and instructions to **efficiently guide tasks**
- Validate info before adding to context window to **avoid errors**
- Only use LLMs when tasks tools or algorithms do not work to **reduce token use**

Context

Read more on context engineering

Anthropic

LangChain

Building Manus

Human layer

Source: Anthropic; LangChain; BCG

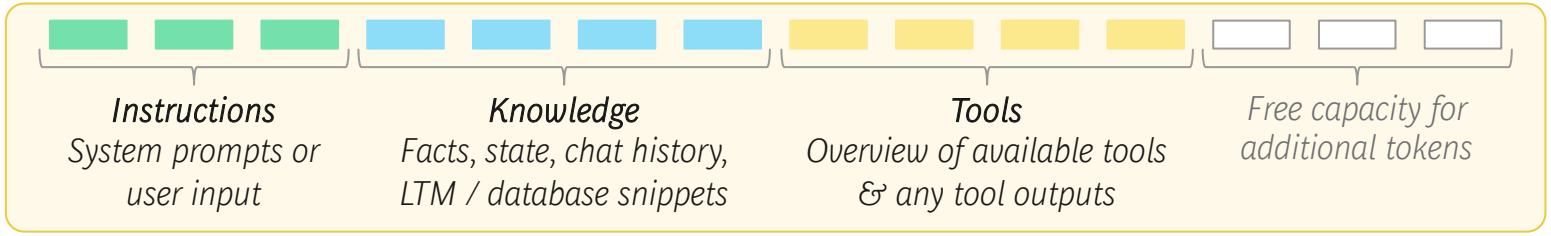
Effective agents integrate short-term context with long-term knowledge & learning

Memory

Provides the agent the ability to recall past actions and behaviors

Short-term memory (STM)

- Temporary, limited-capacity context window used by an AI agent during a single session
- Holds information needed for ongoing processing as tokens, consisting of instructions, knowledge & tools



Long-term memory (LTM)

- Information stored outside the AI agent that persists across sessions and can be retrieved as needed
- Divided into three types: Semantic, Procedural & Episodic memory

Semantic	Procedural	Episodic
Abstract, factual, domain-specific knowledge	*Information about how to perform tasks or skills*	*Information about past events as example behaviors*

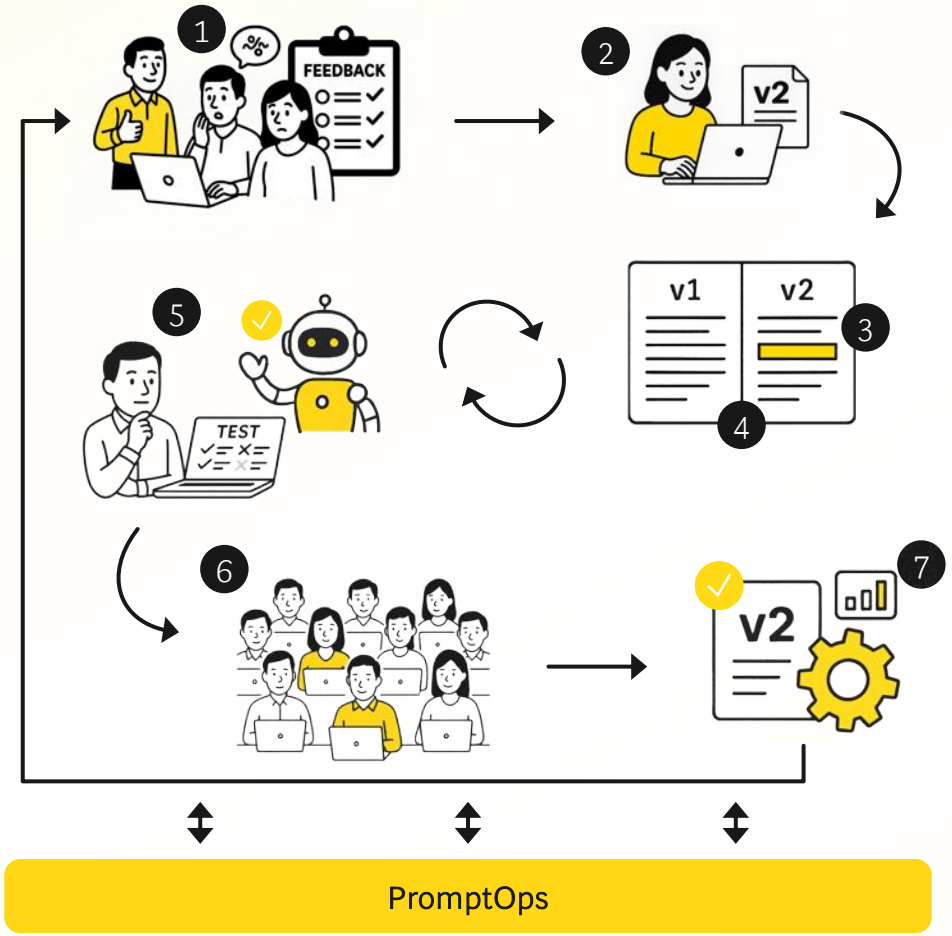
Short-term memory **maintains coherence** within a session by storing relevant recent context, enabling consistent multi-step planning & action

Long-term memory **provides continuity** across sessions by storing knowledge, preferences & experiences, supporting learning and knowledge accumulation & reuse

A **well-built agent integrates both**, balancing task grounding with lasting knowledge and experience. This **integration is not trivial** however, with key challenges including deciding when to promote from STM to LTM, compress & forget, and how to efficiently retrieve relevant memories

Source: BCG

Tune prompts by iterating based on feedback and using consistent versioning



1. Set up continuous feedback loops to track input from users, LLM judges, and safety checks to **identify problems or opportunities**
2. Always pin and version prompts in a registry so that **every change is traceable and reproducible**
3. Change one element at a time (e.g., examples, instructions, schema) with explicit success criteria to **isolate impact and enable attribution**
4. Use structured outputs and tool specifications instead of prose instructions to **ensure reliability**
5. Evaluate prompts through multiple layers, e.g., golden datasets, LLM judges, safety checks, and cost / latency monitoring to **build confidence in robustness, accuracy and scalability**

   Note: Some vendors provide "black box" prompt tuning, replacing human engineering with data-driven or agent-based systems that automatically tune prompts based on best practices or provided eval sets / criteria
6. Complete production A/B tests, start with canary rollouts and scale only if metrics confirm uplift to **validate improvements**
7. Ensure observability with full tracing and maintain rollback paths to **revert safely if issues occur**

Source: BCG

Agent failure modes are plentiful and left unaddressed can cause major risks

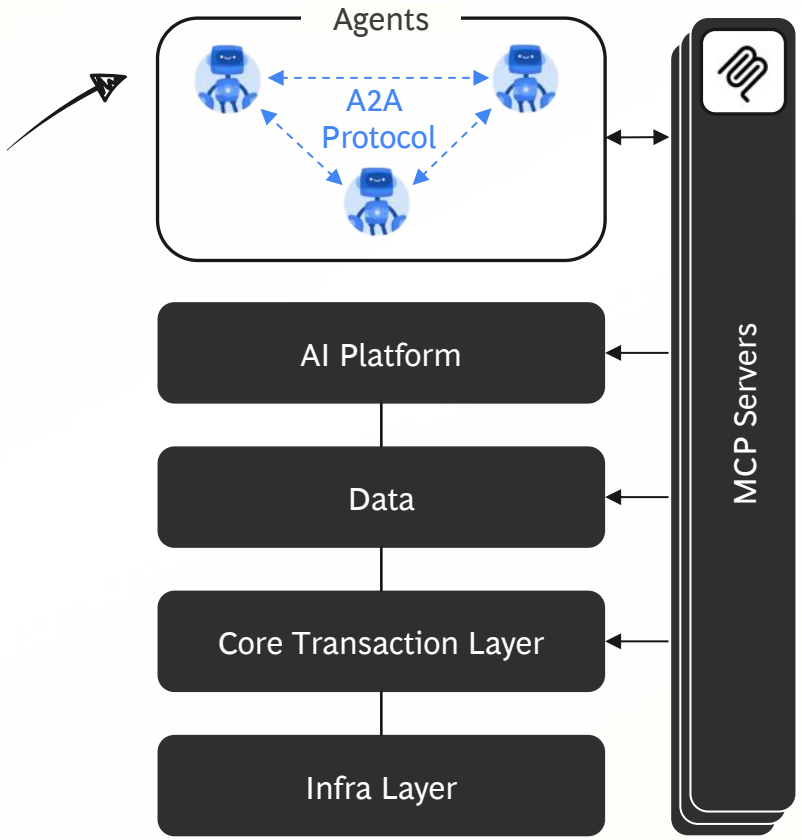
Failure mode	Identity, authN & authZ failures An agent's identity or permissions are wrong, missing, or abused	Data & content supply-chain failures An agent ingests, stores, retrieves, or emits wrong or harmful data	Orchestration, tools & integration failures Breakdowns in how an agent plans and uses capabilities	Objective, reasoning & alignment failures An agent's internal objective or reasoning misfires	Governance & human failures Failures stemming from people, process, and organizational context	Operational, cost & availability failures The system "works" but degrades service or explodes costs
Examples	*Agents are subverted, impersonated or taking unintended actions, e.g., sharing sensitive data* *Attackers exploit human / system errors to evade human-in-the-loop control*	*Instructions injected directly or hidden in external content (e.g., emails) drives harmful behavior* *Data passed between agents loses metadata* *Unfiltered harmful content surfaces to user*	*Poor multi-agent coordination degrade behavior & creates agent deadlocks* *Attackers sabotage agent flow or tool use* *Jailbreaks emerge from multi-agent interactions*	*Agents misinterpret tasks or hallucinate, producing misleading outputs* *Multi-agent drift due to misalignments* *Personalization embeds & amplifies biases across agents*	*Lacking agent traceability hinders accountability* *Agents request approval without giving users enough context* *Over-delegation to agents erodes org. knowledge*	*Malicious inputs drive agents to overuse system resources, degrading service quality or availability* *Multi-agent interactions get stuck in local loops, requiring intervention or restart*
Mitigations	➢ Assign **unique agent identifiers**, granular roles, and permissions ➢ Enable **audit trails** & continuous monitoring ➢ Apply **control flow guardrails**	➢ Enforce **strict memory controls** & validation ➢ **Limit trust in external sources** (XPIA protect.) ➢ **Monitor data flows** to detect malicious content	➢ Apply **control flow guardrails** ➢ Restrict agent interactions to **scoped environments** ➢ Continuously **log & audit** agent behavior	➢ **Design UX** to provide oversight ➢ Apply **control flow guardrails** ➢ **Monitor reasoning patterns** to catch hallucinations or bias	➢ Assign granular **agent roles & permissions** ➢ Build **UX safeguards** to support informed user decisions ➢ Continuously **log & audit** agent behavior	➢ Apply **rate limits, timeouts** & guardrails ➢ **Isolate environments** to contain overuse ➢ Monitor **usage patterns** to flag inefficiencies early

Source: Microsoft; BCG

Once integrated, Google's A2A protocol can shape how agents communicate

A2A defines **how agents talk, coordinate, negotiate, and share state—not how they're built**
It supports natural communication, plan refinement, task handoffs, and cross-boundary collaboration

Leading agent frameworks including Google's Agent Developer Kit (ADK), CrewAI, LangGraph, and GenKit have examples integrating A2A into agent building frameworks to enable **natural agent-to-agent collaboration with each other**



A2A and MCP solve different layers of the AI tech stack: A2A handles the dialogue between agents, while MCP enables agents to discover and call each other as resources via AgentCards[1] , and give them access to tools

Proceed with **curiosity and caution**. Protocols like A2A are promising but expect fragmentation, evolving specs, and competing standards

1. Agent Card: A public metadata file (usually at /.well-known/agent.json) describing an agent's capabilities, skills, endpoint URL, and authentication requirements. Clients use this for discovery. Source: Google; BCG

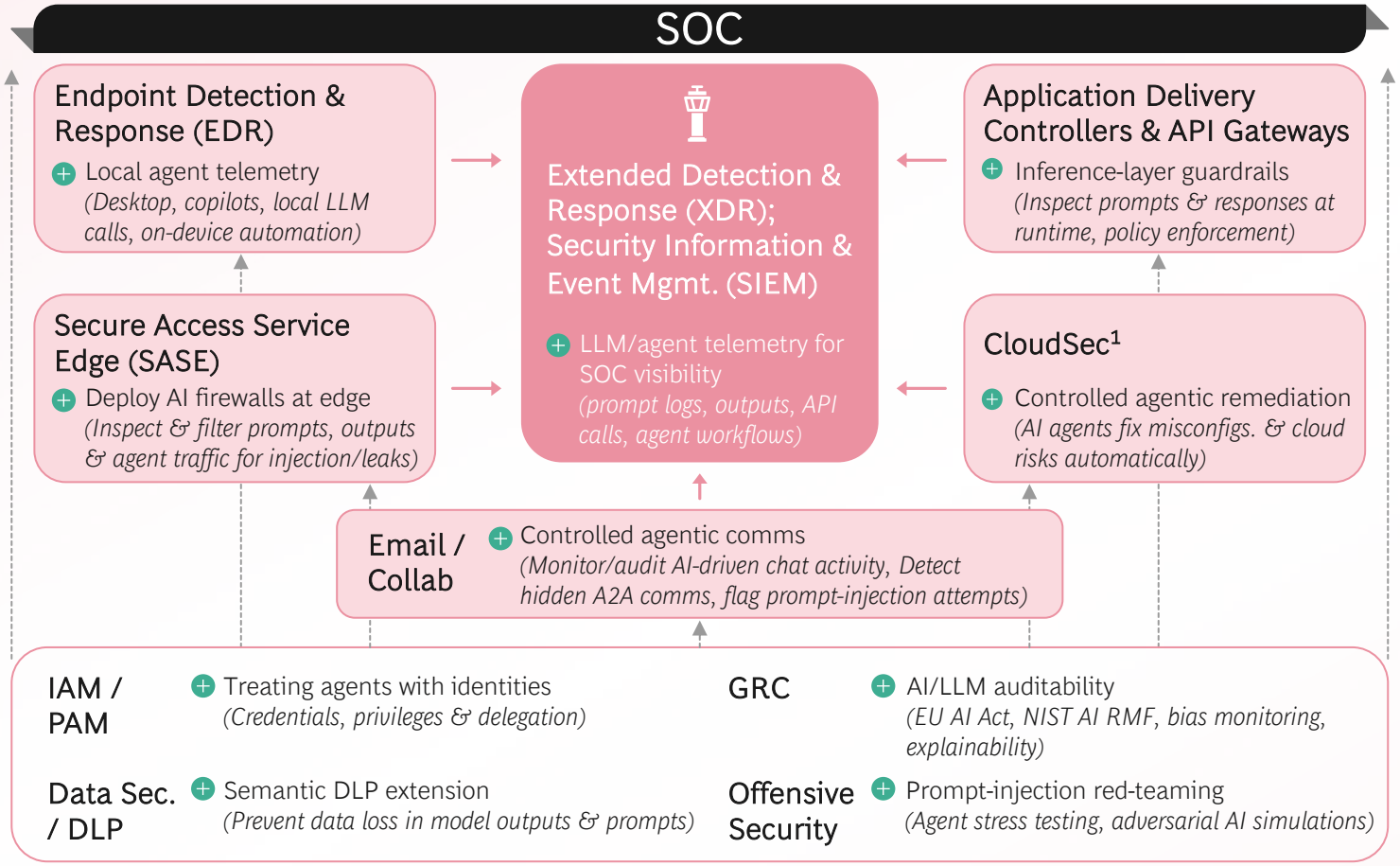
Agent framework decision trades off rapid low-code builds vs. pro-code flexibility

Decision criteria

	Speed to first value	Customization & autonomy depth	Integrations	Governance & compliance	Observability & testing	Cost & scale
Low- to No-code Platforms with visual interfaces, drag-&-drop components & pre-built connectors, limiting coding	**Days** to configure an agent using pre-built connectors and templates	**Offer rule / flow-based orchestration** & some multi-agent / routing capabilities; limited ability to design custom reasoning	**Rich ecosystem** of SaaS & API connectors & extensible via vendor SDKs, some constraints for legacy / custom integrations	**Increasingly built-in** enterprise governance, but depth varies by vendor and transparency can be limited	**Mostly basic functionality**, including dashboards, some vendors now provide tracing, metrics, and analytics out-of-the-box	**Lower entry cost**, per-user or per-runtime licensing can become costly as agent fleets scale
Pro-code Development using programming languages and AI frameworks to write code to design agents	**Weeks** to first deployment; requires coding, environment setup and testing	**Full flexibility** to implement custom planners, reasoning, hierarchical/multi-agent systems & eval strategies	**Unlimited** potential, can integrate with any API/SDK/legacy system with full control over error handling & orchestration	**Full control** as developers design & enforce logging, compliance pipelines, and fine-grained access controls for agent actions	**Advanced monitoring**, custom logs for reasoning, distributed tracing, rollback, debugging, and CI/CD testing pipelines	**Higher upfront build/ops cost,** but more efficient at scale because infra., caching, and agent lifecycles can be optimized

Source: BCG

Security control planes must evolve to tune into new agentic AI attack surfaces



1. Security operations centers (SOC) require **agent telemetry to drive visibility and response**
2. **Control planes must evolve** to absorb agentic risks **across existing security layers**
3. Organizations must secure identity, data, and compliance to **enable trusted agent adoption**

Those who adapt will:

- Avoid blind spots
- Demonstrate compliance early
- Build trust in AI deployments
- And be positioned to capture competitive advantage with safe, secure agent adoption

1. CloudSec refers to CNAPP (Cloud-native App. Protection), CSPM (Security Posture Mgt.), CWPP (Workload Protection), CDR (Detection & Response), CIEM (Infra Entitlement Mgt.); Source: SentinelOne; Prompt Security; F5 networks; CalypsoAI; Aim Security; Check Point Security; Lakera; Crowdstrike