

WHITE PAPER · ADTECH & AGENTIC AI

Agentic AI & media buying

The investment guide

Where to invest, what to build, what to standardize - and what to avoid investing in at all costs.

Contents

Introduction	2
Key terminology	2
What this guide covers	3
Part 1 - Mastering connectivity with the MCP standard	4
1.1 Connecting to a system with an MCP connector	4
1.2 Wrapping your own ecosystem in an MCP server	6
1.3 A key distinction: internal and external use	7
Part 2 - MCP and industry protocols: making the right choice	8
2.1 Connecting in-house agents to AAMP or AdCP	8
Part 3 - The strategic choice: in-house or protocol-based agents	10
3.1 Case 1 - External transaction agents (buyer / seller agents)	10
3.2 Case 2 - Agents for purely internal use	11
Part 4 - How agents are built today	13
4.1 The end of single, isolated agents	13
4.2 The development of agentic systems	13
4.3 Falling development costs	14
4.4 The new role of development	15
4.5 Recap: in-house versus protocol-based agents	16
Part 5 - Other investment considerations	17
5.1 Investing amid the risk of obsolescence	17
5.2 The golden rule: do not overengineer	18
Part 6 - Managing costs, evidence and control	20
6.1 The real cost: what falls and what rises	20
6.2 Performance evaluation: objective and recurring	20
6.3 Auditability	21
6.4 Guardrails	21
Conclusion - Key takeaways	24

WHY THIS DOCUMENT

Introduction

Agentic AI is reshaping media buying. But with in-house agents, MCP, AdCP, AAMP, agentic frameworks and proprietary solutions, technology choices are multiplying - and poor decisions can quickly become costly.

Should you develop your own agents or rely on emerging industry standards? When is MCP truly useful? When does adopting a protocol become essential? Above all, where should you invest today without building an architecture that will be obsolete in twelve months?

This guide offers a practical framework for making the right choices. It explains the architectures gaining ground, the costs that are falling - and those that are rising - the pitfalls of overengineering, and the new requirements for performance, auditability and control.

Key terminology

Four concepts recur throughout the following pages. Defining them now will avoid confusion.

AI agent	Autonomous software capable of performing varied actions (analysis, operational tasks, etc.) adaptively. It combines an LLM's capabilities with a proprietary software layer. An AI agent has decision-making autonomy: it plans, executes, interacts with external platforms and iterates until it achieves its assigned objective.
Agentic system	A group of narrowly specialized agents that collaborate under a master agent's orchestration. Unlike a traditional AI agent handling an entire task, an agentic system comprises many agents collaborating and working in parallel toward the same objective.
MCP	A connector comparable to next-generation APIs that draw on generative AI capabilities. It exposes a system's functions (e.g. SaaS, CRM, database) as tools that a model or AI agent can call directly, without a custom integration for each platform.
Industry protocol	Primarily AAMP and AdCP: open-source agentic protocols designed to automate digital advertising operations. Each protocol provides a predesigned technical structure, standardized vocabulary and exchange schemas, with industry practices built into its foundations. AAMP is developed by IAB Tech. Lab, while AdCP is developed by the Agent Advertising Organization.

What this guide covers

- **Part 1 - Mastering connectivity with the MCP standard.** When and how to connect to a third-party system via MCP, and how to open up your own.
- **Part 2 - MCP and industry protocols.** Why "MCP or a protocol?" is not an either-or choice, and when to connect your own agents to AAMP or AdCP.
- **Part 3 - The strategic choice.** The decisive question when choosing between developing your own agents and relying on those provided by protocols.
- **Part 4 - How agents are built today.** The end of isolated agents, the rise of agentic systems and the collapse in development costs.
- **Part 5 - Other investment considerations.** Investing amid the risk of obsolescence, and the golden rule for keeping overengineering in check.
- **Part 6 - Managing costs, evidence and control.** What falls and what rises, performance evaluation, auditability and guardrails.

The document concludes with eight key takeaways.

PART 1

Mastering connectivity with the MCP standard

MCP (Model Context Protocol) is a standardized connector system that links independent AI systems. It has two symmetrical uses: connecting to a third party's system or opening up your own.

Technically, MCP acts as an API layer designed natively for AI: it exposes a system's functions as tools that a model or AI agent can call directly, without a custom integration for each platform - a kind of API 2.0, consumed by AI rather than handwritten code.

BEFORE YOU BEGIN

Two points to watch

Most MCP servers deployed today expose only read functions (data extraction, reporting). Write functions - launching a campaign, changing a budget or pausing an ad set - remain less common for security and risk-control reasons. They will become more widespread as confidence in agent autonomy grows (see section 6.4).

The same need for control makes it risky to connect AI assistants or agents directly to third-party platform APIs without using official channels: since early 2026, several advertising accounts connected to unauthorized AI tools on Meta Ads have been suspended for usage deemed non-compliant with the platform's terms of service.

1.1 Connecting to a system with an MCP connector

There are two ways to interact with a third-party platform. They differ only in the degree to which the interaction is automated.

Method	How it works and key features
AI chat connection <i>(ChatGPT, Claude, etc.)</i>	The user adds the third-party platform's MCP URL as a connector in their chat. The chat then lets them query the platform in natural language. The exchange therefore works like a standard "prompt → response" chat with the third-party platform.
AI agent connection <i>(system to system)</i>	Agents connect directly to the third-party system via MCP. The agents, each assigned a specific objective beforehand, interact with the system instead of the user doing so through chat.

Method	How it works and key features
	Just as a chat user might conduct a series of question-and-answer exchanges with a third-party MCP, the agent completes a complex sequence of dialogues, queries and actions, but without human intervention. Execution is complete and dynamic, with the agents summarizing the final output.

The shared benefit of both methods

- Users no longer need to navigate a UI and perform several tasks to obtain answers (e.g. open a campaign, extract a report and analyze it).
- Developers or automated systems no longer need to make conventional API calls to a platform to retrieve information, which requires both a dedicated script and further processing of the raw data returned.
- In every case, queries are made "naturally," key information is found automatically, and actions can also be performed.

In practice, connecting an AI chat to a third-party MCP means obtaining a URL such as <https://mcp.platorme.com/mcp> and entering it in the chat's "connectors" or "plugins" section (e.g. ChatGPT, Claude, etc.).

Add a custom connector ✕

Connect Claude to your data and tools. [Learn more about connectors](#) or get started with [ready-to-use ones](#).

Shown in the connector list.

The HTTPS URL where the server accepts MCP requests, e.g. <https://mcp.example.com/mcp>.

⤴ Advanced settings

Only use connectors from developers you trust. Anthropic does not control the tools that developers provide and cannot guarantee that they will work as expected or remain unchanged.

Developing an MCP server? [Report issues and subscribe to updates here](#).

Figure 1 - An example of connecting through AI chat: adding the third-party MCP server as a connector.

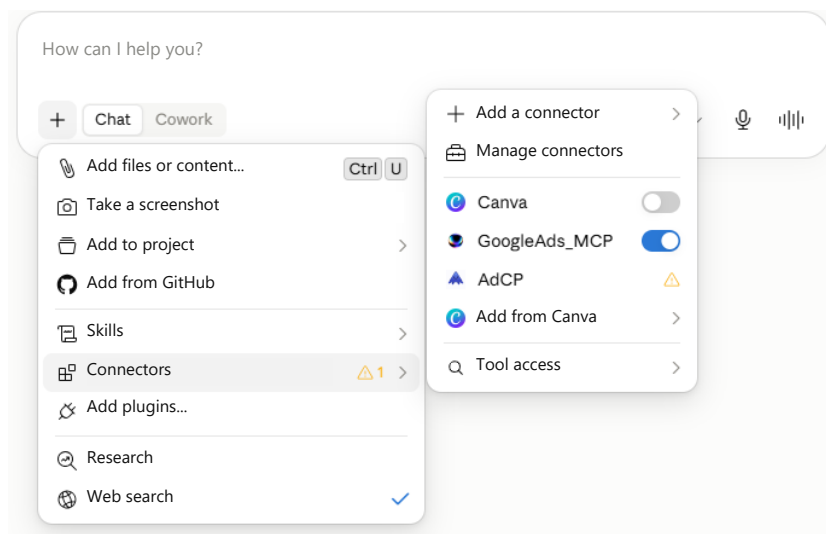


Figure 2 - The connector then becomes accessible directly from the conversation.

1.2 Wrapping your own ecosystem in an MCP server

The reverse approach is to deploy an MCP server around your own technology so that third parties can connect in an "AI-friendly" way. Two use cases justify this investment:

- Case A - Upgrading a traditional platform.** A company has conventional technology (accessed through an API / UI) without native AI or agentic capabilities. Wrapping it in MCP lets external or internal users query and use it through their own AI chats or agents. This makes a system "AI-friendly" even though it has not historically offered AI features.
- Case B - Providing access to an agentic solution.** A company has an agent system it wants to open to third-party interactions. This was not MCP's primary intended use case (agent-to-agent protocols were originally designed to provide access to third-party agents). Yet it has become the most common and effective way to open agents to the outside world, because MCP works as an all-purpose connector, equally suited to AI agents and traditional platforms.

INSIGHT

Why MCP works in every case

A2A (Agent2Agent, a protocol initiated by Google and later transferred to the Linux Foundation) was designed for agent-to-agent relationships; MCP was designed for interaction with systems.

Actual usage took a different course: widespread MCP adoption created a network effect, and the industry began using it for everything because of its effectiveness and ease of deployment.

1.3 A key distinction: internal and external use

The notion of "third parties" or "independent systems" is not limited to interactions between two separate companies. MCP connectivity is equally relevant internally: it can link proprietary platforms and agents developed in different technical environments within the same organization.

ILLUSTRATION

Example: an agency's in-house DSP and a proprietary reporting tool, built by different teams at different times, can connect via MCP without writing another dedicated integration between them.

PART 2

MCP and industry protocols: understanding the trade-off

One question keeps coming up: do you have to choose between adding an MCP connector to your system and/or adopting an industry protocol such as AAMP or AdCP?

The answer is no. These are two fundamentally different tools, so this is not a direct trade-off. Where a trade-off does exist, it arises in far more subtle and advanced cases. In most situations, this question will not arise.

MCP	It is a connector. It links systems together.
AAMP and AdCP	These are open-source agentic protocols designed to automate operational tasks in digital marketing. They include pre-built agents with a standardized vocabulary, exchange schemas and various specializations in industry practices embedded at the protocol level. They perform specific tasks such as finding and analyzing inventory sources for media planning, negotiating deal IDs and automatically implementing campaigns. AdCP is backed by a consortium led by Scope3, with publishers and DSPs participating; AAMP is backed by IAB Tech Lab, the industry's long-established standards body.

Creating an MCP server does not rule out adopting AAMP or AdCP. In fact, AAMP and AdCP have their own MCP connectors:

- You can connect to and interact with AdCP and AAMP through their MCP connectors.
- You can connect AdCP and AAMP to third-party systems using an MCP connector.
- You connect in exactly the same way: through chat or agents.

2.1 Connecting in-house agents to AAMP or AdCP: feasibility and limits

It is technically possible to connect through chat, or to connect your own in-house agents, which do not natively follow the protocols, to these protocol environments (AAMP & AdCP) through an MCP connector.

Why connect a chat interface or agents to protocols?

Case 1 - Natural-language control

I want to direct the work performed by the protocol (e.g. inventory discovery, negotiation, campaign implementation) in natural language, without having to create a dedicated

interface. I then connect my AI chat to the protocol to control it and give it instructions.

Case 2 - Automating and enhancing control

Another option is to place an agent upstream that interfaces directly with the protocol, instead of performing actions manually through chat. This raises the question of the agent's actual added value.

In most cases, that value remains low. If the agent simply forwards requests to the protocol, it merely adds a step that increases latency, token costs and maintenance. A connected chat interface does exactly the same thing for free.

There is added value in one specific case: when the agent upstream of the protocol injects expertise or data that the protocol itself does not have. The upstream agent then becomes complementary and enriches the process.

The clearest example is combining the information returned by the protocol with domain expertise and/or proprietary data, then applying decision rules specific to the agency or advertiser. For example, the protocol may perform a task cycle (e.g. audience/inventory discovery and the associated planning) that needs to be reassessed or supplemented using the agency's best practices.

DECISION CRITERION

Connect through chat or an agent?

Does my agent add something the protocol does not do, or does less well?

- If yes, the agent is justified.
- If not, it is simply another intermediary with limited added value.

Is connecting agents to protocols viable?

The effectiveness of such a connection must be tested carefully, assessing potential obstacles case by case.

The protocol environment is highly standardized (required fields, prescribed taxonomies, expected formats), whereas an in-house agent is not. This creates friction when an in-house system and a protocol-based system need to work together:

- A non-compliant call from the in-house agent to the protocol will be rejected.
- A compliant but imprecise call (e.g. all fields are filled in, but contain errors) significantly reduces performance and quality and increases the likelihood of hallucinations.

Furthermore, even if the protocol's schemas are followed correctly, you must always check that the protocol can receive what the in-house agent submits and adapt its work accordingly. The risk is that this information is never truly used because the protocol lacks the flexibility to use it.

PART 3

The strategic choice: in-house agents or protocol-based agents

There is a real strategic choice between developing your own agents and relying on those provided by protocols. This choice depends primarily on the use case, then on the time and budget allocated to developing a new system. One question is enough to settle a first major group of scenarios.

DECISION CRITERION

The decision criterion

Will the agents transact with agents outside the company?

- If YES → A protocol is mandatory.
- If NO → Weighing an in-house solution against a protocol-based solution is relevant.

3.1 Case 1 - Agents for external transactions (Buyer / Seller agents)

Context. You want to create agents that interact with the outside world to discover inventory/audiences, negotiate and carry out transactions. Agencies and advertisers therefore need to communicate with publishers and inventory owners.

Recommendation. Use the protocols directly (AAMP or AdCP).

Why. These protocols were standardized specifically so buyer agents and seller agents could communicate under common rules. Departing from the standard means losing the ability to communicate effectively. It would be like connecting systems that speak substantially different languages, increasing errors and hallucinations while reducing performance (see section 2.1).

A simple rule. When different entities exchange information to execute transactions, the entire value chain must follow the same strict rules.

PERSPECTIVE

The industry's protocol history and the OpenRTB analogy

The programmatic example: no one creates their own real-time media buying protocol. We follow OpenRTB because the standard was designed to optimize transactions between DSPs and SSPs. A custom transaction system that does not follow oRTB but tries to connect to it to execute programmatic transactions would introduce additional technical and operational challenges.

The foundation of protocols: a protocol's value is proportional to the consistency of its adoption across participants. If all stakeholders use it, everyone can benefit. If some use it while others bring a "variant", complexity and the risk of failure increase without realizing all the benefits.

3.2 Case 2 - Agents for purely internal use

Context. The agents are not intended to transact externally (i.e. they are not Buyer / Seller Agents). Instead, they serve an internal purpose, such as increasing productivity and operational efficiency: data analysis agents, campaign audit agents, campaign implementation agents, etc.

NOTE

An internal agent can absolutely connect to an external system such as an SSP or DSP. "Internal" is used here to emphasize that the agent is not intended to "talk" to the protocols' buyer and seller agents.

In this case, the choice concerns the starting technical foundation: build outside a protocol, or reuse the open-source agentic framework provided by AAMP or AdCP.

Option A - Non-protocol agent (custom)	Option B - Agent built on a protocol framework
Principle. Build your own agentic solution without using the standardized framework of AAMP or AdCP. Advantages. Full customization. The agent precisely reflects the agency's or advertiser's internal practices, incorporating the organization's specific core skills, expertise and added value. Potentially far higher performance on its particular use case.	Principle. Reuse the pre-built agentic framework that the protocols provide as open source. Advantages. A substantial reduction in initial development costs. AdTech specialization is already built in. Disadvantages. The system is standardized and is not designed for unrestricted customization. Technical consideration. You need enough expertise to remove the elements

Option A - Non-protocol agent (custom)	Option B - Agent built on a protocol framework
Disadvantages. Longer development time and higher cost.	of the protocol that will not be used (e.g. disable buyer agents), activate only the agents needed for the use case (e.g. a campaign implementation agent), and feed this system once it has been separated from its original architecture.

DECISION CRITERION

The three questions that decide between A and B

Question 1 - Does my added value lie in a distinctive way of working?

In other words: does my performance come from practices, methods or expertise that my competitors do not have?

- **If yes → Option A is the only one that preserves this value.** A standardized framework will lead you towards standard industry practices, in other words, everyone else's practices.

Question 2 - Are development cost and time a genuine obstacle?

- If not → The decision should be based on questions 1 and 3.
- If yes → The trade-off between A and B must be assessed specifically, without assuming that B will always be cheaper. The protocol framework saves time on the foundation, but the actual difference in development cost/time depends on more complex factors (see below).

Question 3 - Am I willing to let my agents' evolution depend on the protocol's?

Choosing option B means inheriting a roadmap you no longer control: updates, deprecations and specification changes will be imposed at the pace and according to the priorities of the organization that produces the protocol.

- If not → Option B is not viable, regardless of the immediate development savings.

Question 1 determines the strategy, question 2 the resources, and question 3 is a veto: it can rule out option B even when the first two point towards it.

PART 4

How agents are built today

The paradigm has changed completely in twelve months, both technically and financially. What was true last year no longer holds, yet many investment decisions still rely on outdated reference points.

4.1 The end of single, isolated agents

The first wave produced many agents considered technically "monolithic". These monolithic agents were extremely basic, using only an LLM with two or three capabilities for interacting with third-party platforms. Given their limited sophistication, many were commonly built directly with no-code solutions. The conclusion is clear: inexpensive, certainly, but their reasoning, reliability and business customization remained very limited.

These single agents are now less popular and are used only for very specific, generally simple operational tasks. Their performance remains fairly rudimentary. This limited reliability considerably narrows their use cases and reduces the willingness to deploy them.

4.2 The development of agentic systems

We increasingly talk about agentic systems rather than single agents. The monolithic agent is replaced by a cohort of agents with narrower specializations that collaborate under the orchestration of a master agent.

These systems are gradually addressing the problems and concerns encountered so far, including frequent errors, lack of expertise and questionable reliability, through four mechanisms:

- **Injecting deep expertise.** They enable highly specific domain knowledge to be incorporated and faithfully replicate operational expertise.
- **Breaking down tasks.** Complex work is divided among specialized agents that collaborate, increasing precision.
- **Iteration loops and self-evaluation.** The system continually reassesses its work and iterates until it reaches a stable solution, to make results more reliable.
- **Challenging outputs.** The system produces several valid results, compares them and selects the most valuable one.

These deeper, denser architectures significantly improve output quality. They can replicate highly specialized domain expertise while delivering substantially better outputs and minimizing the risk of errors.

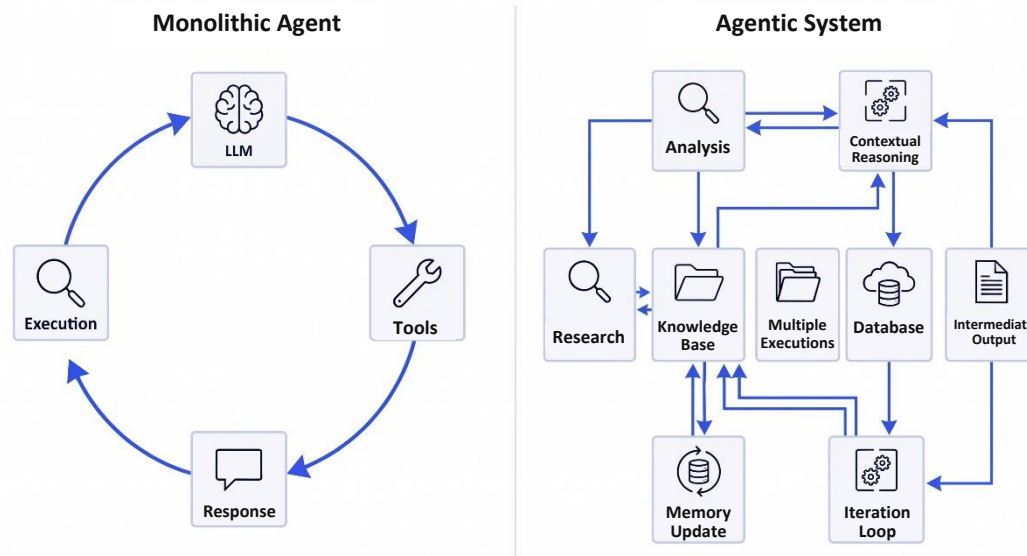


Figure 1 - A comparison of capabilities and execution steps between a monolithic agent and a modern agentic system

4.3 Falling development costs

Previously: significant costs, and why

Twelve months ago, a high-performing agent required long development cycles (at least one quarter, and up to 6-9 months for the most complex cases) and budgets running into several hundred thousand euros.

This cost stemmed mainly from the need to build a complete agentic architecture to maximize the agent's performance. These architectures were the precursors of today's agentic systems: they were designed to enable the injection of expertise, iteration, output evaluation and so on.

NOTE

Not to be confused with an entry-level no-code agent: this explanation of development costs concerns enterprise-grade agents. Their operation drew on the principles of today's agentic systems, and their quality was incomparable to that of low-cost no-code agents.

Today: lower development costs and shorter timelines

The main development reducing the cost of building agents, and especially a complete agentic system, is the emergence of mature agentic frameworks. These frameworks provide a technical foundation for implementing

the architecture for collaboration between agents, along with verification logic and iteration loops: precisely where the real technical difficulty lies.

These frameworks are now mature and widely adopted: LangGraph, CrewAI and AutoGen are common examples.

Building enterprise-grade agents no longer requires developing solutions from scratch. Frameworks greatly accelerate development by providing the agentic system's technical foundation, which explains the associated cost reduction. Work now focuses much more on integrating business practices and specializing the agentic system for operational needs.

THE ORDER OF MAGNITUDE

It is now possible to build a high-performing agentic system in less than a quarter, and cut development time and budget by a factor of three or four (in our experience, compared with building from scratch; varies with scope).

4.4 The new role of development

If the framework provides a strong technical foundation, design work shifts to other challenges:

- Highly specializing agents in the company's business.
- Customizing the agents with tools specific to the business.
- Connecting to third-party platforms (e.g. DSPs, ad servers, etc.).
- Building a knowledge "brain" and ensuring agents use that knowledge properly.

What I should do, and stop doing, to build my AI solutions

Developers' expertise and profiles are changing. There is less need to use a generalist developer or conventional AI providers. Instead, development must prioritize technical experts with a deep understanding of your business.

Expertise can no longer be added afterwards; it must be present from the design stage. Domain expertise is even more fundamental to solution development than before, because building an agentic system specialized in a business is different from building traditional software.

The traditional development model relies on a chain of handoffs

The domain expert describes the challenge/use case. The product manager translates it into features and interfaces. The developer codes the tool. Each link in the chain reformulates what the previous one passed on, creating the familiar handoff risk: domain expertise is distorted or lost along the way. The tool is a container; the expertise is what the user brings when using it.

AI system development is interdependent

In an agentic system, this separation no longer exists. Expertise is not supplied by the user at the point of use: it is embedded in the system itself—in the architecture that feeds the agents, in the division of roles, and in the criteria they use to assess their own work. The agent does not provide a tool to the media trader: it carries out the media trader’s reasoning.

Technical design determines the quality of business execution, and business expertise determines technical design, from the very first line. Unless this interdependence is acknowledged from the start of the project, the system will underperform by design—and the problem cannot be fixed later, because it lies in the way the system was conceived.

4.5 Reminder: choosing an in-house vs protocol-based agent

Earlier (Part 3), we presented reusing the technical foundation of a protocol-based agent as an alternative that can reduce costs compared with an in-house agent developed end to end. We also stated that this choice—focused particularly on cost, technical control and dependence on a third-party protocol—must be assessed case by case.

Sections 4.3 and 4.4 clarify the factors informing this choice. The paradigm shift described, in which agentic system development no longer starts from scratch but from now-mature technical frameworks, also delivers substantial savings in time and engineering effort.

The savings promised by a protocol-based foundation were once measured against full development from scratch; today, they are measured against an in-house agent already built on a framework. The gap is narrowing and no longer consistently favours the protocol-based option.

This is precisely why no standard decision rule can apply. The choice must be assessed case by case.

PART 5

Other investment challenges

5.1 Investing despite the risk of obsolescence

Technology evolves so quickly that a system built today may be outdated in six to eighteen months. Should you wait for stability and risk falling behind, or move ahead and risk losing your investment?

RECOMMENDATION

Start now, but take a pragmatic approach.

Prioritise learning

Generative and agentic AI are evolving rapidly. Waiting means accumulating a knowledge gap that soon becomes difficult to close—and that gap concerns not so much the technology itself as the ability to embed it in your own processes.

What you learn by building systems today will directly inform the systems you build tomorrow. This value is difficult to quantify, but it more than offsets the money invested: it is what will enable you to seize the next opportunities.

Invest to remain adaptable

The risk of obsolescence is undeniable. It does not justify waiting, but it does require careful sizing of projects, their associated costs and the time committed. Four principles can guide this:

- **Small rather than broad.** No mega-projects pursuing perfection or an extensive scope. Favour lightweight systems focused on a specific problem.
- **Accept imperfection.** Aim for a high effort-to-benefit ratio, even if the system does not cover every case. The marginal gain from addressing imperfect cases generally causes this ratio to collapse.
- **Deploy fast, iterate fast.** Speed to production matters more than how comprehensive the initial version is.
- **Keep an eye on the market.** Track technical developments so you can pivot at the right time, rather than continuing to invest in a system already in decline.

If the technology is outdated in a year, the cost incurred remains low and controlled—and you approach the next generation with a head start rather than a project to dismantle.

Gain a competitive advantage, even a temporary one

The debate over whether to invest or wait for obsolescence is often framed incorrectly, because it focuses on the risk of sunk cost (i.e. money committed to a solution that could become outdated).

Opportunity cost is too often overlooked. While some wait, competitors move ahead. They gain operational efficiency or evolve their commercial offering. In either case, the competitor emerges stronger.

The real decision is therefore between a limited, controlled financial cost on one side and an opportunity cost on the other: watching a competitor gain market share or influence while others look on.

KEY TAKEAWAY

A temporary advantage is still an advantage. And the learning it leaves behind lasts.

5.2 The golden rule: do not over-engineer

Every agentic design encounters technical weaknesses in current models and frameworks that can be termed ‘structural weaknesses’. The term needs clarification: despite rapidly improving and already very high performance, blind spots exist—and always will—that prevent us from building the perfect system. Some shortcomings stem from particular LLMs, others from agentic frameworks, and so on. The latter also vary greatly from one industry to another.

This is a structural problem that must be accepted.

ILLUSTRATION

For example, when generative AI launched three years ago, models were criticised for their tiny context windows, unable to accommodate more than a few paragraphs or a short document. Other criticism concerned the fact that early models answered directly, without reasoning or the ability to connect several concepts.

Those criticisms have disappeared; others have taken their place.

Today, criticism focuses more on agents’ reliability in long, multi-step tasks (gradual drift, accumulating errors), or their difficulty distinguishing a legitimate instruction from a manipulation attempt (prompt injection) when processing untrusted third-party content.

Each generation of AI/agentic models and systems brings its own limitations, inherent in the innovation it offers.

Why you should not try to fix them

The golden rule is never to try to compensate for these structural shortcomings through complex engineering and heavy additional layers. That task belongs to the developers of foundation models and major frameworks.

These limitations will meet one of two fates: major technology players will address them, or market innovation will simply make them irrelevant. Either way, technology built to compensate for those weaknesses will no longer have a reason to exist.

ILLUSTRATION

- Data extraction and compaction architectures designed to fit information into narrow context windows lost all relevance when context windows expanded dramatically.
- Complex multi-call workflows built to simulate step-by-step reasoning became unnecessary once models gained that capability natively.

Those criticisms have disappeared; others have taken their place.

In both examples, time and money were invested for nothing. This is a sunk cost in the strict sense (unlike the risk of obsolescence described in Section 5.1).

The trap is all the more effective because, when the additional layer is being developed, it seems entirely justified: it solves a real problem.

Our role: the best users, not the designers

This distinction protects us from that pitfall. We do not design models or frameworks; we use them—and our ambition should be to use them better than anyone else and deliver their most optimised form.

In practice: use the system as it stands, apply it to our business, add industry-specific components and embed our knowledge. Investment should focus on packaging existing technologies into a useful asset for our industry—never on trying to solve what the major technology players have not yet solved.

PART 6

Managing costs, evidence and control

With efficiency and reliability now achievable, the challenges have shifted. Developing, integrating and using agentic systems requires a new approach to these challenges.

6.1 The real cost: what falls and what rises

Build costs are plummeting while operating costs are rising. Agentic systems are more powerful and more effective because they can consume more information and run multiple agents at once.

As a result, a cohort of agents that iterate, challenge each other and verify their work consumes far more tokens.

The unit cost of tokens steadily falls from one model generation to the next, but consumption grows even faster—more context, more iterations, more agents in parallel—which explains why the total bill rises despite falling unit prices.

RECOMMENDATION

Although cost is a genuine concern, it must not handicap the performance of the system being built.

It is strongly recommended to prioritise maximum system performance first, without restricting it, to assess its true capabilities. Optimising costs afterwards becomes an incremental cost/performance trade-off: each euro saved is weighed against the resulting loss in performance, rather than applied as an automatic cut.

Optimising the budget from day one is dangerous because it could significantly constrain the system's true potential, leading to the mistaken conclusion that it is not viable.

6.2 Performance evaluation: objective and recurring

A non-deterministic system cannot be tested like conventional software. The same input can produce two different outputs, both acceptable but potentially different in quality.

A performance evaluation process is therefore essential to—once again—avoid drawing mistaken conclusions.

This is not a topic to explore in depth here, but a point of vigilance to include in every project plan, with three minimum requirements:

- **A reference test set.** Built with the domain expert, covering normal situations and edge cases, and enabling systematic assessment of the system's performance (e.g. after a technical update, to check that performance does not decline over time, etc.).
- **An explicit, objective acceptance threshold.** Set before development, based on objective, measurable factors to avoid perception bias.
- **Periodic reassessment.** Defined as regular evaluation to ensure that no drift emerges.

6.3 Auditability

An agentic system, with its cohorts of agents, produces a considerable amount of information: reasoning, exchanges between agents and intermediate decisions. Its volume and speed far exceed what a human can read.

In the raw format produced by the agents, and accounting for the paths information takes between multiple agents, an agentic system is effectively impossible to audit within a useful timeframe.

The solution is not to read everything. An effective audit process does not seek to assess every action the system takes. It works in two stages:

- **Locate.** Automatically detect areas of friction, underperformance or failure—in other words, reduce an unreadable stream to a handful of points that deserve attention.
- **Analyse.** Apply the analytical capability suited to the type of problem being investigated to those specific points.

Depending on the case, these processes will be human-friendly (a readable account of the path taken), automated or simplified (traces reduced to the essentials). What matters is the principle: you do not audit the system; you audit what the system flags as abnormal.

KEY TAKEAWAY

This is a project in its own right, to plan for at the design stage rather than add after the first incident.

6.4 Guardrails

Moving to agentic systems greatly increases the autonomy granted to the machine. Dozens of agents interact simultaneously with each other and with third-party environments (e.g. databases, DSPs, ad servers, internal tools, etc.).

This autonomy is the source of performance gains on an entirely different scale from the first agents of twelve months ago.

It is also the source of risk: the more a system decides for itself, the further it can stray beyond its intended scope. The danger is not just that an agent 'breaks loose' and compromises

execution; it is that it compromises the third-party environments to which it is connected (e.g. deciding to pause or delete every campaign in the DSP).

Strict boundaries and human checkpoints are therefore essential.

THE PRINCIPLE

Three rules structure this approach, all based on the same principle:

Autonomy is not given; it is earned.

A. Validate manually before granting autonomy

During initial development/deployment, all key actions undergo human evaluation, accompanied by measurement of the error rate and associated risk.

Only once those errors become rare and minor is human validation removed and the system allowed to operate on its own. Validation is not a start-up constraint: it is the measurement tool that authorises the move to the next stage.

B. Expand the scope gradually

An agentic system must earn the right to expand. When it is built and tested, it must handle a limited number of tasks within a narrow scope. Each time the observed errors and risks are judged acceptable, the scope and list of permitted actions can be expanded.

A critical point: every expansion temporarily puts previously validated work at risk again. An agent reliable across five tasks may deteriorate across ten—because the context grows, interactions multiply and edge cases intersect. Every extension therefore requires reassessing previous cases, not just the new ones.

C. Set rules that must never be crossed

Whatever level of maturity is reached, some limits must never be removed. Three mechanisms enforce them:

- **Absolute prohibitions and permanent approval requirements.** A list of actions the system will never execute alone, regardless of its confidence level—typically irreversible, binding or large-scale actions.
- **Automatic detection of boundary breaches.** An absolute precautionary principle requires the following philosophy: ‘Any boundary, however robust, can be bypassed sooner or later.’ Alerts must therefore be put in place to detect instantly when these theoretically impermeable boundaries are crossed, to guard against any catastrophic event.
- **Systematic backups.** Just as data centres replicate their data to survive the loss of a site, retaining restore points so that a loss of control remains recoverable is another good practice to adopt.

KEY TAKEAWAY

Building an agent is not just a technical task, in terms of either effort or time.

Once the initial design has been delivered, much of the project's time is spent validating, iterating, measuring performance and risk, and adjusting.

This phase is just as critical as development in validating the system's robustness. It must be budgeted accordingly from the outset.

A schedule that ends at technical delivery will produce a system that is functional, but fragile, incomplete or even dangerous.

SUMMARY

Key takeaways

Eight points summarize the choices explored in this guide. Each reference takes you back to the corresponding part.

01 MCP does not compete with industry protocols

MCP is a connector; AAMP and AdCP are industry protocols. There is no "MCP or protocol?" choice: the protocols themselves expose their capabilities through MCP. → Part 2

02 One question determines the strategic choice

Are your agents intended to transact with agents outside the company? If so, a protocol is mandatory. If not, choosing between an in-house solution and a protocol-based foundation becomes relevant again. → Part 3

03 Adding agents on top of a protocol must be justified

If an agent merely relays requests, it adds only latency, token costs and another maintenance point. A complementary agent must provide specific added value that the protocol can use. → Part 2

04 The standalone agent has had its day; enter agentic systems

Agentic systems represent the highest-performing agentic approach: a cohort of narrowly specialized agents, orchestrated by a master agent, that break down tasks, iterate and challenge their own results. → Part 4

05 Build costs are falling sharply; operating costs are rising

In our experience, mature frameworks cut development time and budgets by a factor of three or four. But operating costs rise: more context, more iterations and more agents running in parallel. → Parts 4 and 6

06 Do not wait for technological stability

Invest small, focused and fast rather than large and perfect. The opportunity cost of waiting outweighs the risk of the investment becoming obsolete. → Part 5

07 Never overengineer to compensate for model limitations

Blind spots in models and frameworks will be addressed by their creators or made irrelevant by the market. Any additional layer built to compensate for them is a sunk cost. → Part 5

08 Autonomy is earned, not granted

Human validation first, then a gradual expansion of scope, with non-negotiable rules throughout. Budget for this phase as you would for development. → Part 6

EPICFLARE

About Epicflare

AdTech consulting and AI development

Epicflare helps publishers and media buyers improve their advertising performance and operations. We combine industry expertise with technical delivery, taking each organization's tools and working practices into account.

AdTech consulting

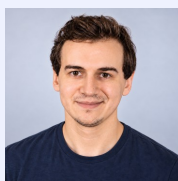
Publisher monetization and advertising stacks; buying methods, campaign setup and performance measurement for media buyers.

AI consulting and development

Use case identification, feasibility assessment and workflow design. Development of tailored agents, automations and integrations, from prototype to a solution integrated into the client's tools.

AI training

Helping leaders make informed decisions and supporting teams in adopting practices and solutions suited to their roles.



Raphaël Ambit

Founder of Epicflare

Former VP Advertising at Dailymotion and former Senior Director Technology at Jellyfish. Also CTO at Alliance Digitale. His publisher and agency experience connects business needs with operational and technical choices.

Selected references



Dailymotion



Homa



Everyday



InfoPro Digital



Waterwiser



Le Parisien



Technip Energies

LET'S DISCUSS YOUR PROJECT

contact@epicflare.io · epicflare.io

