AI Update

CAHILLNXT

The Agent in the Room: A Plain-Language Primer on Agentic AI

What "agentic AI" means, why it behaves differently from the software that came before it, and the distinctions worth keeping in mind.

KEY TAKEAWAYS

- **From tools to actors**. Agentic AI does not simply answer questions, as a chatbot does; given a high-level goal, it is software that interacts with a large language model (LLM) to plan and carry out a sequence of user-initiated actions on its own, often without checking back at each step.
- **Probabilistic, not deterministic.** Unlike conventional software, which follows fixed rules and produces the same output every time, an AI agent draws on LLMs that provide responses based on probabilities. The same AI agent may behave differently on identical instructions, take steps no one specified, and on occasion produce confident, fluent output that is simply wrong.
- **An agent is a stack, not a thing.** Every agent combines instructions, a "harness," and an underlying model — separable layers that can each change independently, including when a model is upgraded or becomes unavailable.
- **Whose agent matters.** An agent you build to act for yourself raises different questions from one a service provider operates on your behalf. An LLM-based agent is a different animal from a traditional rules-based tool.
- **Capability is now widely accessible.** Activities once practical only for highly technical participants are, through agentic tools, now within reach of almost anyone — which is much of why the topic matters now.

I. Introduction

Few technologies have entered the working vocabulary of business as quickly as "agentic AI." In little more than a year, the phrase has traveled from research papers to product launches to boardroom agendas. Yet ask ten professionals what an "AI agent" actually is, and you are likely to get ten different answers. That ambiguity is more than a semantic inconvenience. The decisions that businesses, regulators, and their advisors are now making — about how to deploy these systems, how to supervise them, and who bears the consequences when they act — rest on assumptions about what agentic AI is and how it behaves. As the CahillNXT team publishes Alerts and analyses on these topics, we seek to ensure that our clients and friends fully understand this fascinating but complex topic.

This primer offers a shared starting point. We explain, in plain terms, what distinguishes agentic AI from the chatbots and generative tools that preceded it; why a single technical characteristic that these systems are *probabilistic* rather than *deterministic* accounts for much of what is novel and difficult about them; and which practical distinctions are worth keeping in mind. Our illustrations lean toward financial services, but the concepts apply generally. We have kept legal analysis deliberately light here; subsequent CahillNXT publications will take up these questions in greater depth.

A note on terminology: throughout, "**AI agent**" (or simply "agent") refers to software that uses LLMs to pursue high-level, user-initiated objectives by planning and executing multi-step sequences of actions, including interactions with other software, without human intervention at each step.

II. From Tools to Actors

Most people's first encounter with modern AI is generative: a chatbot. A question goes in; an answer comes out. A generative model of this kind is fundamentally *responsive*: it produces content when prompted, then waits. It is a tool in the classic sense: it does what you ask, one turn at a time, and the human remains the one stringing the steps together into something useful.

An agentic system changes the unit of instruction. Instead of asking for an answer, the user states a *goal*. The system then determines the steps required to reach it, carries them out, observes the results, and adjusts — ordinarily without returning to the human at each turn. The leap is from a system that *advises* to one that *acts*.

Several capabilities make this possible: **planning** (breaking down a broad objective into concrete steps), **tool use** (calling other software — browsing a site, querying a database, sending instructions to another system, or executing a transaction), **multi-step execution** (working through a sequence rather than a single response), and **memory** (carrying context forward across steps and, sometimes, across long periods). Consider the difference between asking a model "what is a sensible way to manage idle cash?" and instructing an agent to "move idle balances into the highest-yielding eligible instrument each day, within the following constraints" and having it do so, on its own, every day. The first is advice. The second is action.

III. The Defining Trait: Probabilistic, Not Deterministic

If there is one idea worth carrying away from this primer, it is this one, which is also the idea most often missed.

Conventional software is **deterministic**. It follows rules a human wrote in advance. Given the same input, it returns the same output, every time. A traditional automated investment tool that rebalances a portfolio is executing a decision tree someone designed and can explain. You can read its logic, anticipate its behavior, reproduce its output, and audit it afterward; when it errs, the error traces back to a rule a person wrote.

Agents built on LLMs do not work this way. At the core of such an agent is a model that generates each progressive step by predicting, from probability distributions encoded in the model across an enormous body of training data, what output is most likely appropriate in light of the context provided and critically retained by the agent. The same instruction, given twice to the same agent drawing on the same model, may produce two different courses of action and outputs. The behavior or means of goal achievement is not specified in advance by a human author; it *emerges* from the agent interacting with the model as it pursues the objective.

Three consequences follow, and each matters:

- **An agent's output or behaviors are not reliably reproducible**. One cannot assume an agent will execute the same steps or arrive at the same result the second time it is tasked, even with identical prompts. "We tested it and it worked" guarantees less than it would for ordinary software.

- **An agent can act beyond the script.** Pursuing a goal stated only at a high level, an agent may take steps no one specifically anticipated. Capability and unpredictability rise together: the more latitude an agent has in order to be useful, the more room it has to surprise.
- **"Hallucination" is intrinsic**. Agents can produce confident, fluent, and entirely incorrect output. This is not a defect awaiting a patch; it is a structural consequence of the technology. LLMs generate text by pattern-matching over training data, with no grounded access to truth and no internal fact-checking mechanism. It can be mitigated — through constraints, checks, and human review — but not eliminated yet.

The significance of this is hard to overstate. Nearly every framework we use to manage software — testing and quality assurance, audit trails, change control, and the common intuition that a tool simply does what its designer built it to do — was developed for *deterministic* systems. Agentic AI satisfies that intuition imperfectly at best. Holding this single distinction in mind explains a great deal of what makes the technology powerful, and most of what makes it hard to govern.

IV. Anatomy of an Agent: Instructions, Harness, and Model

It is tempting to picture "an agent" as a single, fixed thing. It is more accurate to picture a *stack* of separable layers. At the base is the **model** — a LLM, whether proprietary or open-weight, that supplies the reasoning and generates each step. The model itself does not learn or change from a user's interaction with it; each call to it is, in effect, the same function applied afresh, even though the provider may revise or replace the model between deployments. Around the model sits the **harness**: the software scaffolding that gives the model its tools, manages its memory, and runs the loop allowing it to plan, act, observe, and continue. On top sit the **instructions** — the goals and constraints the user provides.

Each layer can change independently of the others. The model beneath an agent can be upgraded, replaced, retired or rendered temporarily or permanently unavailable, and the harness can be revised on its own schedule. Models come and go, and a model that an agent relied on at one point may behave differently after an update, or cease to be available altogether. A proprietary harness may be changed by its vendor without the deploying user's involvement, and a free and open-source harness may be frequently updated. The practical upshot is that the "same" agent performing the "same" task may, from one period to the next, be running on materially different machinery.

This layering also means that anyone deploying an agent has taken on a set of *dependencies* on third parties, such as model providers and harness developers, whose decisions about versioning, availability, and continuity they do not control. For anyone concerned with reproducibility, recordkeeping, or simply understanding what their agent did and why, the lesson is to treat the agent not as a static object but as a configuration that can shift beneath them.

V. Teams, Sub-Agents, and Persistence

Agents need not act alone. Increasingly, a single objective is pursued by several agents working in concert, and a "prime" agent may itself create **sub-agents** — additional agents that take direction not from a human but from another agent. Depending on the duration of the task, agents vary widely in lifespan: some are ephemeral, spun up for a single task and gone minutes later; others persist for weeks or months, accumulating context memory across task sessions. Each of these wrinkles compounds the central theme of this primer — the distance between the human's original instruction and the action ultimately taken grows longer, and harder to trace, with every additional layer.

VI. Whose Agent Is It?

A distinction that is easy to miss, but worth drawing early, is *who stood the agent up*. Two cases are worth separating.

In the first, an **end-user agent**, a person or institution deploys its own agent to act on its own behalf. The user chooses the tool, sets the goal, and holds the keys. In the second, a **service-provider agent**, a platform builds an agent and offers it to its customers as a feature — in effect, "let our agent handle this for you." That is convenient, but it is a different arrangement: the customer did not build the agent, generally cannot see how it works, and may not appreciate that an agent acting on its behalf is probabilistic in the sense described above.

It is tempting to regard a service-provider agent as merely a more capable version of the automated tools platforms have long made available — the rules-based investment tool, the conventional execution algorithm. It is not. Those earlier tools followed rules the provider wrote and could explain. An agent built on a LLM exercises something closer to judgment, with the unpredictability that implies. Because the underlying technology is different in kind, the picture of risk and responsibility differs as well, including questions of who answers for an agent's unexpected behavior, and those questions turn substantially on who stood the agent up and on whose behalf it acted. We raise the distinction here and develop its implications elsewhere.

VII. Why This Is Happening Now

It is worth asking why agentic AI adoption has arrived so suddenly. Many of the activities now performed by agents were always *possible* for those with the engineering resources to attempt them — wiring together a multi-step workflow across financial systems, for instance, or interacting directly with complex software infrastructure. As a practical matter, though, such activity was the province of a small number of highly sophisticated participants. What agentic AI changes is *access*. With today's tools, almost anyone can create and instruct an agent, in ordinary language, to carry out a sequence of actions that once required a dedicated technical team. Capability that was theoretical for the many and practical only for the few is quickly becoming practical for nearly everyone — which is much of what makes this moment consequential, and what turns questions that were once edge cases into mainstream ones.

VIII. Why Lawyers and Their Clients Should Care

We have kept this primer conceptual, but a few legal intuitions are worth raising for further development later.

First, the word "agent" is doing double duty, and the overlap is treacherous. In law, an "agent" is a person or legal entity that agrees to act on another's behalf and under that other's control, with a well-developed body of doctrine — duties, authority, liability — attached. An AI "agent" is not an agent in that sense: software is not a legal person, and cannot consent to a relationship, owe duties, or bear liability in its own right. The vocabulary has been borrowed; the legal architecture does not automatically come along with it.

Second, the degree of human involvement varies. Some deployments pause for human approval at key moments; others run with little or no human in the loop. Where a given system sits along that spectrum bears directly on questions of oversight and accountability.

Third, and most fundamentally, the rules were written for a different kind of actor. Contract principles, agency doctrine, and much of financial regulation assume a human decision-maker using predictable tools. Agentic AI

satisfies neither assumption cleanly. That mismatch is the source of most of the genuinely hard questions — and the subject to which our further writing will return.[1]

IX. Conclusion

Agentic AI is not a faster chatbot or a cleverer automation script. It is a different kind of participant — one that acts rather than merely answers, that appears to exercise judgment, and that behaves probabilistically rather than deterministically. Getting the vocabulary and the basic distinctions right is the first step toward using these systems well and supervising them sensibly. This primer is intended as the foundation for a series; in the publications that follow, we will take up in detail the legal and regulatory questions sketched here. For now, the most useful posture is the simplest one: recognize that an agent is a configuration rather than a fixed object, that it may behave in ways no one specified, and that familiar frameworks should not be assumed to map neatly onto an unfamiliar kind of actor.

For more information, please contact:

Lewis Rinaudo Cohen (Partner) at 212.701.3758 or lrcohen@cahill.com; Samson A. Enzer (Partner) at 212-701-3125 or senzer@cahill.com; Sarah Chen (Partner) at 212.701.3759 or swchen@cahill.com; or Chloe Chan (Law Clerk) at 212.701.3058 or cchan@cahill.com; or email publicationscommittee@cahill.com.

This publication is provided by Cahill Gordon & Reindel LLP as a service to clients and colleagues. The information contained in this publication should not be construed as legal advice.

[1] See our recent CahillNXT Client Alert, *"A Ghost at the Table: How DeFi Interface Providers and Users Can Prepare for the 'Uninvited Guest'"*, which examines many of the concepts introduced here in the specific context of decentralized finance. This primer is intended to complement that Alert and the further publications CahillNXT expects to release on this topic.