Department for Science, Innovation & Technology

Government Digital Service

AI Insights

Agentic AI

What is agentic AI?

Artificial intelligence (AI) agents are small, specialised pieces of software that can make decisions and operate cooperatively or independently to achieve system objectives. Agentic AI refers to AI systems composed of agents that can behave and interact autonomously in order to achieve their objectives.

Recent advances in agentic artificial intelligence (AI) have driven remarkable growth in both its popularity and capabilities. This progress is due to the integration of large language models (LLMs) with agent-based systems. By providing reasoning and discovery abilities, LLMs enhance an agent's autonomy. This enables the agent to determine the most appropriate course of action to meet system objectives.

Traditional software typically follows fixed pathways to solve problems. In contrast, agent-based systems operate like independent assistants that choose and combine several actions to achieve their goals.

Legacy agents

Agent-based systems have been in use for a very long time. These legacy agents operate without autonomy and follow fixed steps in response to events.

These are often used to simulate large populations and their interactions. These systems bear little resemblance to agentic AI.

The fundamental difference with agentic AI is that the execution pathway is now derived intelligently, by utilising LLMs in the planning process.

How is it different?

In traditional systems, the order of instructions is set in advance. If we consider an online ordering system, traditional software systems create an order, add order items, process the payment, then send for distribution, and so forth. There is no variability in the process-flow that has not been hard-coded into the system.

Agentic systems, on the other hand, are given a set of capabilities and will then autonomously select which of those are best suited to achieve an objective - for example, fulfilling the order. This system may then look up the best price, consider competing prices, back order out-of-stock items, and offer different prices for different delivery schedules autonomously. This ability to use tools to solve problems is what makes agentic systems so powerful.

In this simple example, software engineers might argue that this flexibility could also be created through traditional coding. They would be entirely correct. However, as features and capabilities increase, the legacy codebase tends to become increasingly complex, harder to maintain and more difficult to extend. Agentic systems increase their capabilities by learning new abilities. These are then used to help the system to define an effective execution plan.

Capabilities

In agentic AI, functions and tools represent the different capabilities that the system has at its disposal. In the order management system we have used as an example, we may have tools or functions that check if items are in stock and to look up prices. This pricing function could also independently check if the user's account should have some discount applied, or equally may examine the latest pricing if a cached price is stale.

These capabilities are registered with the agentic system, with a natural language description of what they do, and any parameters they take. Tools are typically used to communicate externally through application programming interfaces (APIs), with databases or external systems. We don't differentiate between them here, and some frameworks may have different terminology. As far as we are concerned, capabilities are the set of all functionality which exist and have been described in natural language and made available to the LLM.

The LLM uses this information to select the best course of action at different points in the execution life cycle.

These effectively act as building blocks in the system. The agentic AI will combine them as necessary to process orders. That is the real power and flexibility of agentic AI.

Orchestration and choreography

Agentic AI uses two main methods to combine these building blocks to solve problems: orchestration and choreography. These are terms used to describe how an agentic system coordinates the execution of functions.

Orchestration

Orchestration behaves like a central controller that manages the order of operation for system requests. It is like a conductor in an orchestra.

In our order processing example, the system would first validate the user account, then check prices and availability, and so on. This is following the logical and well-established order fulfilment life cycle. There is no need for the agentic AI system to try to derive or learn what is already well known.

The individual steps are composed of the functions selected through dynamic routing in the agentic system. This affords tremendous flexibility and extensibility in agentic systems.

Choreography

Choreography expresses a greater degree of freedom than the mostly command-driven orchestration. Choreographed behaviour is more loosely-coupled, often as an event-driven mechanism.

For example, during order processing, the system notices an item is out of stock so it raises a back-order and registers a request with the notification system to send an email to our user when the item comes into stock. This will happen independently based on the item's arrival. Other examples include system maintenance and governance operations for efficient operations, like ensuring functions are viable, services are running, and so on.

Both approaches are typically combined together where we need to control an order of operations, even if those operations themselves are discovered and selected at runtime. aThese operations are used alongside other independent behaviours, which are generally invoked based on system events.

Memory in agentic AI

Some agentic AI systems maintain state as they execute. This means that they keep a running tally of data and, sometimes, behaviour. This is useful for transparency, logging, or accumulating pertinent information as the system executes. The output of one system step is often the input into subsequent steps.

Agentic systems may also maintain longer-term memories, such as user preferences, recent interactions and outcomes, and so on. These are updated and extended with each operation.

Reinforcement learning

Some agentic systems will capture data during execution which can ultimately be used as input into reinforcement learning methods. These are intended to enable the system to learn from recent interactions, and can be extremely powerful as the system refines, adapts and improves itself over time.

Benefits of agentic AI

Adaptability

The AI may independently choose execution pathways based on the task at hand and other environmental considerations. For example, during times of peak activity where the system is effectively overloaded, an agentic system may defer some operations until the system is less overwhelmed. The same could apply during order delivery, with the system offering an incentive to the buyer to choose delivery on a date where other local orders are also in transit.

Extensibility

It is easy to add new capabilities to an existing agentic system and make them available for selection during dynamic task routing. We can extend the command vocabulary of the agentic system through this mechanism and enrich these behaviours by providing alternative functions.

For example, we could provide more than one pricing lookup behaviour, or apply special discounts to preferred customers, and know that these pathways will be chosen, intelligently, if the right conditions are met.

Dynamic routing

Rather than creating rigid systems of branching, looping, and step-wise instructions, we can instead register our abilities with the system and have the agentic AI select the set of operators appropriate to the task. This can help to eliminate a great deal of internal scaffolding code required to route behaviours manually. This, combined with the system extensibility described above, gives an agentic AI system the ability to become increasingly powerful over time.

We can also use the same processes to reduce, remove, or refactor functions over time, which would streamline the overall processes and eliminate redundant or obsolete behaviours.

Dynamic task routing can combine orchestration and choreography to achieve its results more efficiently. In our order processing case, if an item is out of stock, it can be processed independently. Perishable items could also be sent at a higher priority.

Automation

The same reactive, dynamic capabilities apply to automation through agent-based systems. Non-critical maintenance operations can be deferred during busy times and registered with a choreographed system to reattempt when the system is quieter. This intelligent adaptability far supercedes the wasted processing effort, or the fail-and-retry model which prevails on many linear systems.

Application programming interfaces

An application programming interface (API) is a mechanism used that allows systems to communicate with other systems. Very often, dealing with legacy APIs requires highly-specialised expertise. This leads to resource deficits, an extended development and testing time, and an increase in overall cost and risk.

Solution providers are increasingly exposing those same APIs through agent-based systems. This removes or softens the learning curve considerably. Making APIs more

generally available by abstracting away the intricacies makes them much easier to use. See our Integrated Agents article for more information.

Risks

Agentic AI is a very exciting form of automation and delivery. We are still in the early stages of this technology's development, though, so it is important that we temper our enthusiasm with a few sobering thoughts:

Full autonomy

Complete autonomy in agentic AI systems, as for almost all AI systems, can be problematic where they may remove critical layers of human oversight and ethical judgement. Without human intervention, these systems could make decisions that are misaligned with our intentions, potentially leading to harmful or unintended consequences.

By maintaining a level of human control and oversight, it becomes easier to ensure that the AI's actions remain consistent with our broader interests and safety standards.

Premature deployment

Deploying systems from proof of concept prototypes straight to production is the cause of much failure and frustration in agentic AI systems.

These systems can often be challenging to engineer in the first instance. Once a system is working, a team may mistakenly assume their work is done, and the solution is simply deployed to production. These systems are typically insufficiently tested, inadequately profiled, and generally do not meet the non-functional requirements of a production system. This is more common than we would like, with global technical communities awash with a great many examples.

Architectural concerns

Unclear architecture is an anti-pattern characterised by a lack of structure with minimal separation of concerns.

When adding new features, extending existing capabilities, or fixing errors, the item is unceremoniously bolted-on anywhere. They are rarely removed or refactored, even if they become obsolete or unnecessary. This quickly leads to an accumulation of technical debt over time.

Volatile frameworks

There is a fast pace of change in the agentic AI domain. Frameworks compete with each other to adapt to new features of underlying LLMs. Very often these adaptations break compatibility with previous versions. This lack of backwards compatibility makes these frameworks rather brittle. That is, they are prone to break when upgrading to the next iteration of that framework. This creates a much tighter coupling between code and package versions than usual.

This also imposes a few constraints on system designers. Not least of these are strict adherence to version management and control to ensure that only the fully-tested and targeted versions of frameworks are deployed. In addition, framework versions tend to have shorter lifespans than traditional software systems. Framework providers simply cannot support so many multiple concurrent versions for very long. This compels a faster testing and upgrade cycle than that of many legacy systems.

Clear Definitions

While self-describing tools and functions afford tremendous flexibility, they are also vulnerable to potential ambiguities in language that may make it difficult or impossible to resolve satisfactorily.

If the language used to define their functions isn’t clear and precise, the system can misinterpret what it's supposed to do.

As a slightly extreme example, if we accidentally described our pricing tool as a weather forecasting function, the LLM would choose the wrong tool as it tries to predict the weather instead of calculating prices - with all the success we might expect.

Small language nuances or developer comments can subtly influence the system’s decisions, leading it to select the wrong function. This kind of issue doesn’t affect traditional software, which follows strictly coded sequences. Agentic AI depends on clear language definitions to work correctly, and even slight ambiguities can result in inefficient or incorrect outcomes.

Output consistency

Output may vary where an underlying LLM is used to resolve a response. This is a well-known LLM concern, and we are especially vulnerable to this where we are dealing with edge cases or sparse data.

For example, this might be an issue if we wanted a specific path to be followed for an order date for 29 February, or a different path again where we have only a subset of data. This can also happen in medical systems for rare ailments. Therefore, we must ensure that our data are consistent across the range of acceptable input parameters.

Lack of transparency

Given the innate autonomy of agentic solutions, and their runtime selection of actions to take, it may not always be entirely clear how agentic solutions arrive at their conclusions.

That is, given that the agentic systems themselves decide which tools or functions to use, and where multiple agents collaborate (and they, in turn, also decide which path to execute), it may not be obvious to system administrators or engineers exactly which route was followed.

Development costs

Agentic systems also have token costs to consider. Tokens are words or parts of words that are processed by LLMs. Using external LLMs typically incur a financial charge. While these charges might not be too high at first, over many iterations of test cycles, performance profiling, and development, they add up.

It may be tempting to bypass these charges during development or testing by using locally-hosted LLMs but we must ensure that we know what we are measuring.

If we decide that we will only test against a local model which is hosted internally but deploy to a production environment which uses a different LLM, then we should expect some variability in our results. Often a hybrid solution, a mixture of local and remote

LLMs, will suffice but we must test and measure to ensure consistency between environments.

Mitigations

Agentic systems should be built to manage these risks at a fundamental architectural level. This can be done by isolating agent interactions through the solutions architecture. These don’t remove the issues related to, for example, short version life cycles, or breaking changes. Rather, they minimise vulnerability by abstracting away framework interactions behind smaller modules. This also facilitates testing and change management.

Composition and interfaces

Composition allows us to build agents from smaller, well-defined components that can be tested and maintained independently. Rather than creating monolithic agents that handle multiple concerns, we compose them from specialised modules - one of which might handle API interactions, another natural language processing, and yet another the decision-making logic.

Interfaces provide clear contracts for how these components interact and establish logical boundaries that make the system more predictable and testable. Interface segregation takes this further by breaking down these contracts into smaller, more focused, interfaces tailored to specific behaviours. For example, rather than having a single large interface for all agent operations, we might separate out interfaces for reasoning, tool manipulation, and state management.

This separation means that changes to one aspect of the agent's behavior are less likely to cascade through the system, making it more resilient to framework updates or LLM changes. When a new version of an agent framework is released, we can update the language processing interface implementation without affecting the tool manipulation or state management code.

This architecture also creates natural boundaries in the system where we can insert logging, monitoring, and testing hooks, making it easier to test and validate system behaviour.

Explainable AI

The lack of transparency described above can be ameliorated by using explainable AI methodologies. While agentic systems can sometimes be considered “black boxes” where the internal workings are unknown, we can shed some light on this by providing an audit log of the changes in state and function calls as a request is resolved. This can facilitate development, debugging, and troubleshooting tremendously. This should be standard practice in any agentic system.

This may be further extended by maintaining a state machine which tracks the data consumed or produced, the changes made to that data and the functions called while execution is underway. This is especially useful for human-in-the-loop (HITL) solutions, so a user may have oversight and approval of the full audit trail.

Testing and evaluation

Generally, in testing, we examine three paths:

- a success path expecting a successful outcome
- a failure path expecting failure
- an exception path, expecting specific or generic exceptions to be raised

We typically also evaluate sets of input parameters sampled from a valid range of values against a range of expected results based on those inputs.

Software testing is generally non-trivial, but essential to maintaining software quality.

Testing LLM-based systems, and especially agentic AI, brings new challenges to the table. Unit testing, on its own, is likely to be insufficient. System and Integration testing will be needed, along with appropriate test suites which can demonstrate multiple execution pathways through the system.

If a test suite which is expected to call the pricing functionality described above fails to do so, we should have a failing set of tests.

Because of the natural language descriptions, the internal prompts, sub-prompts and dynamic route evaluations, testing agentic AI requires a thoughtful and considered approach.

There may also be concerns about safety where an agentic AI system exposes natural language querying capability to the outside world. However, this is query-related and not specific to agents. See our LLM Testing, our Evaluation Metrics, and our LLM Safeguarding Insights for more information.

Profiling

In addition to evaluating our solutions to ensure they are accurate, and behave as expected under a wide range of input parameters, we must also profile our systems so that we know how they behave under different conditions.

We use profiling to determine the costs of execution of:

- the number of parallel/synchronous requests that the system can support
- the number of concurrent users the system can support
- the marginal cost of additional requests
- the relative cost of different execution pathways

Within reason, it is difficult to be too well-informed about the profile of our systems, and profiling is essential to direct any optimisation efforts to the right areas.

In addition to profiling for performance reasons, we also profile the behaviour of our systems which have been deployed to production so that we may correlate expected and achieved performance, and detect any behavioural issues.

Agentic frameworks

While the volatility of frameworks can be a risk (see above), their usage also mitigates risk. Agent frameworks abstract away a great deal of interaction between the system, the foundation model, functions, tools, and agents in the agentic AI system.

This means that they hide the low-level interactions, instructions, and operations, leaving only high-level behaviours and configurations. This greatly simplifies the creation of agent-based systems by reducing the burden on the system designer. This further reduces potential for error by removing the need to wire up every prompt with every behaviour by hand.

A great many of these systems are open-source, which means that - assuming they are in widespread use - errors tend to be found faster and fixed sooner. This does not mean

that closed-source, paid-for systems are slow to update - they have a financial incentive to do so - but paid-for systems may do so at a time of their choosing, which is not a restriction that open-source systems are generally subject to.

Building your own framework

Regardless of whether the chosen framework is open- or closed-source, free-to-use (or at least licensed) or paid-for, using popular frameworks means that we have a widespread community of users with many different usage scenarios. There are a great many eyes on those frameworks. Developers and designers communicate with each other on public forums where issues, challenges, and solutions are openly discussed. This is a tremendous resource.

Hand-crafting our own frameworks removes these advantages, and we can become blind to the shortcomings in our own work. Building our own frameworks is not recommended unless there are especially peculiar concerns in play.

Conclusion

Agentic AI represents a significant evolution in how we build software systems, offering new levels of flexibility and autonomy. However, it is important to recognise that these systems are still information systems like the vast majority of our current software systems estate.

Most of what makes a successful agentic AI system has little to do with AI directly. Instead, fundamental to it is testing, monitoring, data management, profiling, appropriate solutions architecture and engineering discipline.

The decision to implement agentic AI should be informed by our business requirements and careful evaluation of the benefits and risks. While agent-based systems offer powerful capabilities, organisations that succeed with this technology are those that match it best to genuine business needs rather than simply adopting it for its own sake.