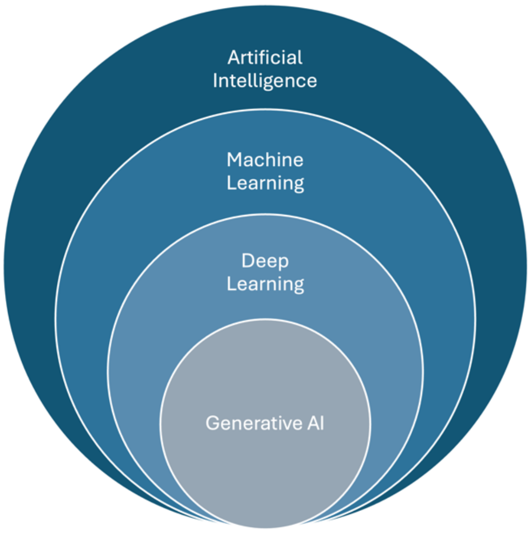
Agentic AI

1 AGENTIC AI: FUNDAMENTALS

1.1 Overview of Generative AI

Introduction to Generative AI

Artificial Intelligence (AI) is the field of making machines behave like humans by solving problems, making decisions, or learning from experience. A common example of AI is a chatbot that can answer your questions. Within AI, **Machine Learning (ML)** is a technique where computers learn from data instead of being programmed with specific rules. For example, email systems use ML to detect and filter out spam messages based on user behaviour. A more advanced form of ML is **Deep Learning (DL)**, which uses large neural networks inspired by the human brain to handle complex tasks, such as understanding speech or recognizing faces in photos. Building on this, **Generative AI (GenAI)** is a type of AI that creates new content – such as text, images, music, or code by learning patterns from existing data. Examples include tools like ChatGPT, which can write stories or answer questions, and image generators that create pictures from simple text description often called as prompts or user prompts.



Key features of Generative AI

- **Content Creation:** Unlike traditional AI, which primarily analyzes and responds to existing data, generative AI actively creates new, unique content. This includes generating human like text, realistic images, synthetic data and so on.
- **Model Learning Mechanisms:** Generative AI models are trained on vast amounts of data using complex algorithms, often employing unsupervised or semi-supervised learning techniques. This allows the model to learn from unlabelled data sets and generate novel outputs.

- **Applications:** The scope of Generative AI is broad, encompassing applications in natural language processing, digital art, music composition, drug discovery, and more. It enhances creativity, automates mundane tasks, and improves data analysis across various industries.

Scope of Generative AI

The scope of generative AI is vast and rapidly expanding. It includes:

1. **Creative Industries:** Generative AI is used in art, music, and writing to create new pieces that mimic human creativity. Tools like DALL-E and ChatGPT are popular examples.
2. **Coding Assistants:** Generative AI is transforming the software development landscape through coding assistants, IDE extensions like - GitHub Copilot, Amazon Q, Cline, CodeGPT, etc and AI powered IDE's like - Cursor, Windsurf, etc. These tools provide real-time code suggestions, generating unit test cases, automate repetitive coding tasks, and improve code quality by identifying bugs and suggesting optimizations.
3. **Business Applications:** It enhances customer service through AI-powered chatbots, automates report generation, and brings in optimizations in various processes in terms of cost and manual effort.
4. **Scientific Research:** Generative AI helps in various domains with research, including healthcare domain where the models aid in drug discovery by generating molecular structures with desired properties, reducing the time and cost associated with traditional methods.
5. **Data Augmentation:** It aids in generating synthetic data to augment existing datasets, improving the performance of machine learning models, especially in areas where original data is limited or sensitive.

1.2 Overview of Agentic AI

Remember the movie "The Matrix"? A key set of characters in this movie were "The Agents". They looked and behaved like humans. But they were not flesh and blood. They were just code. There were a bunch of them moving about the simulated world of the Matrix amidst humans. They would move more efficiently than humans. They could do things faster than the humans surrounding them. They interacted with other agents. They also interacted with humans. They didn't have emotions. They had goals. They had "agency", that is, the ability to act or to choose what action to take. And they would use the tools at their disposal to attempt to achieve those goals. That's pretty much what Agentic AI is as well. A network of Agents that can align themselves with each other or other tools or humans to accomplish a goal. Let's dive deeper.

1.3 Evolution of Agentic Systems

The evolution of Agentic AI can be traced through time as follows:

1. Traditional Programming
2. Gen AI (Narrow AI)
3. Agentic AI

We have mastered **Traditional Programming** over the past few decades. This is where humans have used languages like Java, C# or even COBOL to get computers to act in a pre-determined manner. Once you create a simple traditional program in Java you know exactly how it will behave for a given set of inputs. For instance, if you implement a simple "add" method you can accurately predict that if you give it input as 2 and 3, you will get 5 as the output. Most of the software applications in existence as of now fall under this category. These programs cannot decide. But they can follow a set of pre-defined rules set by the programmer. Trouble brews if it encounters a novel scenario which the programmer had not foreseen. What it lacks in adaptability it makes up for in predictability or dependability.

Simple applications of GenAI or **Narrow AI** came next. (We are intentionally glossing over Traditional AI just to keep this concise.) These are your simple applications of GenAI, where the overall workflow is fixed. But the output of certain LLM calls add a certain element of non-determinism to the overall program. Consider the simple example of a GenAI powered application that summarizes a document. Even if you were to stick with the exact same LLM, temperature, top-P and top-K settings, you could get a slightly different summary for each execution on the same document. The essence of the summary could be the same. But they would differ ever so slightly. That's Narrow AI for you. Another example is where you employ a LLM to generate an image based on a textual description. It's highly unlikely that 2 image generations will be exactly alike.

Finally, we have **Agentic AI** at the peak of non-determinism. Here not just the output but even the path taken to reach the output is difficult to determine. In Agentic AI we develop agents with reasoning capabilities. We empower them to analyse the scenario and make plans. We intentionally decide to allow the agents to take certain actions on their own (sometimes with a human review). When properly built, it becomes difficult to predict with precision how exactly an Agentic System will behave for a given set of input. It's not that they are chaotic. That's not how they should be built. But since we are allowing the agents to reason and act it's not unnatural for certain agentic plans to be surprising to the original system creators. But that's the whole point of Agentic AI. Reaching the goal is important. The path taken to reach it is of secondary importance.

1.4 Narrow AI vs Agentic AI

Aspect	Narrow AI	Agentic AI
Definition	Uses GenAI models in predefined rigid workflows to perform specific tasks.	Uses GenAI in flexible workflows for formulating the rest of the workflow, enabling autonomous decision-making and adaptability.
Task Scope	GenAI is limited to specific, predefined tasks in the overall workflow.	GenAI is used to formulate the plan itself. The plan may further need to invoke other GenAI on non-GenAI based agents.
Flexibility	Inflexible; operates within a fixed workflow and cannot adjust to new situations.	Highly flexible; adjusts workflows and behaviours based on context or unforeseen inputs.

Aspect	Narrow AI	Agentic AI
Decision-Making	Certain rule-based decisions might be delegated to GenAI. E.g.: if sentiment is classified as positive, execute Action X, else Action Y.	Higher level decisions would be delegated to GenAI. E.g.: which tools to invoke? In which order to invoke the tools? What info to obtain from the environment?
Predictability	Output cannot be predicted. But workflow can be predicted.	Both output and workflow cannot be predicted.
Use Cases	Examples include automated report generation, fixed-response chatbots using GenAI.	Examples include self-driving cars, automated cloud infra remediation systems.
Impact on Productivity	Using GenAI appropriately in a fixed workflow can help boost the productivity of the SMEs using that workflow. But this impact will not be a lot (e.g.: >70%) since the humans will still be in the driver's seat and choosing to use NarrowAI as the appropriate tool when required.	Agentic AI when implemented properly can help boost productivity significantly. Mainly because it removes the bottlenecks from the overall workflow i.e. dependency on human SMEs for every choice. Only the important decisions can be the points where the system awaits input from SMEs. Rest of the flow can be autonomous.

1.5 Key Concepts

What is **Agency**? It's the ability to exercise action to reach one's goal with sufficient reasoning and invocation of necessary tools. It's easier to explain with an example. A few years ago, I (Cladius) had my daughter enrolled in kindergarten. Her school had provided a soft copy of a document that had to be printed, photos of the child and parent affixed and then stamped by the school office on top of the snaps. This was an important safety measure. Most parents had printed it on an A4 page, and some had even laminated it. Parents were asked to carry it every time they showed up to pick up their kids. Some parents had folded the A4 print into an A5 size and as such made it a little less bulky. I, on the other hand, created a new template while ensuring that all fields mentioned by the school were present. I got a special approval from the school admin team for the new format and got it stamped. The end result? My pickup card was the size of a credit card that I could neatly place in my wallet. In a simple way, this is what agency is about. When we build an Agentic AI system, we want to simply marry reasoning with tools and direct it towards a goal. The real "agency" of the system is demonstrated when it breaks out of rigid patterns and is able to solve unforeseen problems in ways that were not clearly spelled out in code beforehand.

An **Agent** in the context of Agentic AI is an entity that:

1. Is usually powered by an AI model - often a LLM - to provide reasoning capabilities
2. Has access to 1 or more Tools available for it to use to execute steps/tasks
3. Is programmed with a Goal which serves as a guide for its steps

Example: consider an Agent tasked with obtaining and summarizing a YouTube video transcript. It might use an LLM like OpenAI o1 as its "brain", a tool like YouTube API to fetch the transcript, and the same LLM to create the summary - all directed towards the singular goal of delivering a concise summary.

An Agent should ideally exhibit the following characteristics:

1. **Autonomy**: It can function independently, without requiring undue human intervention, allowing it to perform tasks on its own.
2. **Decision-Making**: It has the capability to analyse information from its environment and decide which tools to use and in what sequence, enabling it to respond effectively to a variety of situations.
3. **Goal-Oriented Behaviour**: It takes actions aimed at achieving specific objectives, aligning its efforts with the Goal that its programmed with.

One of the key limitations of LLMs used to be that they could not directly interact with the real-world outside of receiving input (text, image, video, audio or document) and respond back with similar modalities of output. They couldn't connect to a DB, or make an API call, or make a phone call. Hence, the onus of building tools to carry out these actions and "binding" them to the LLM used to be the AI engineer's job. However, this trend is changing. Anthropic has integrated an optional web search "tool"[1] as part of its LLM calls for select models. This works in a 3-step manner[2]:

1. Claude decides when to search based on the prompt.
2. The API executes the searches and provides Claude with the results. This process may repeat multiple times throughout a single request.
3. At the end of its turn, Claude provides a final response with cited sources.

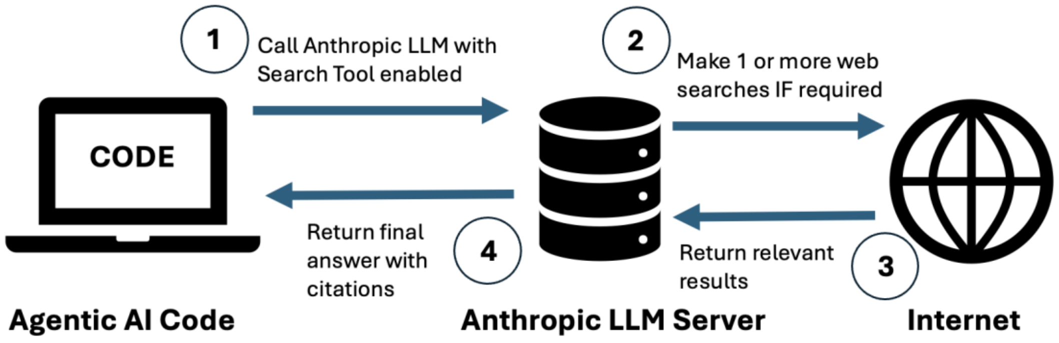

1
Call Anthropic LLM with Search Tool enabled
2
Make 1 or more web searches IF required
CODE
Return final answer with citations
4
Return relevant results
3
Agentic AI Code
Anthropic LLM Server
Internet

Figure 1 Augmented LLM on Server side

As of the time of writing this book, Claude had provisioned 2 "server" tools – Search & Code Execution[3]. But more tools and more LLM providers are bound to follow suit. As such, AI engineers may then have 2 options while building their systems – use the

[1] https://www.anthropic.com/news/web-search-api
[2] https://docs.anthropic.com/en/docs/agents-and-tools/tool-use/web-search-tool
[3] https://docs.anthropic.com/en/docs/agents-and-tools/tool-use/code-execution-tool

pre-built tools from the LLM providers OR bind your own custom tool or tools built by others to the LLM "agent".

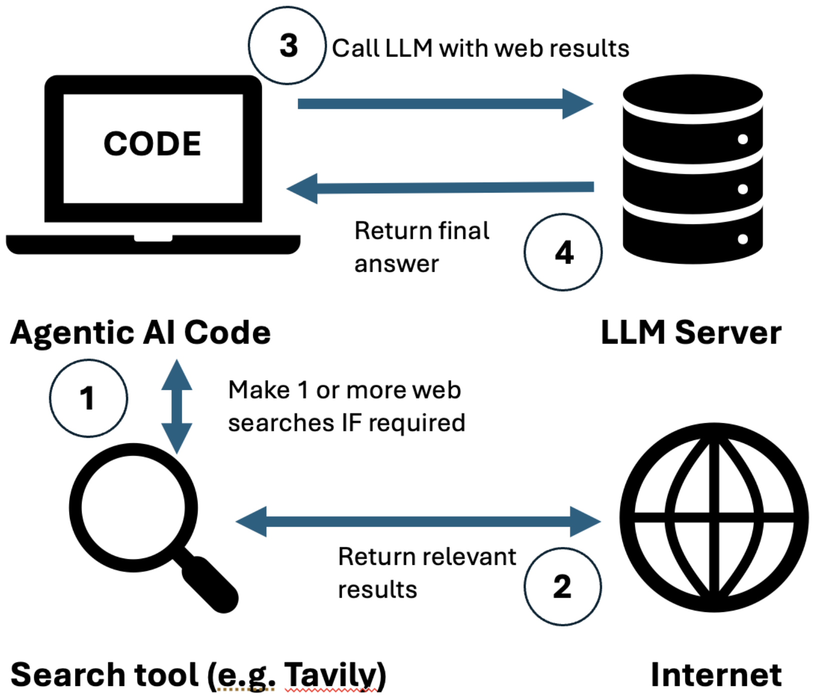

3
Call LLM with web results
CODE
Return final answer
4
Agentic AI Code
LLM Server
1
Make 1 or more web searches IF required
Return relevant results
2
Search tool (e.g. Tavily)
Internet

Figure 2 Augment LLM with tool on Client side

Agentic AI is overall a sophisticated system composed of one or more Agents, Tools, LLMs and their interconnections all arranged in a manner that allows for Users to interact with the system and employ the agents to work in co-ordination with each other and with limited oversight to achieve goals that would otherwise be overly complex for either a single agent or human. A key trait of an Agentic AI system is that the overall workflow or sequence of steps would not always be completely predictable. It should be designed in a manner that there is room available for the "Planner Agent" to craft a plan with the available Tools to handle a scenario which the System Designer might not have foreseen. This is by far one of the biggest advantages of Agentic AI as compared to Narrow AI or Traditional Software.

Example: An Agentic AI system could have a Database Connector Tool and an Email Connector Tool. Based on just these 2 tools, it could potentially handle a variety of situations like but not limited to:

1. Monitor a support ticket database for unresolved issues and send an internal email to the support team to investigate it OR send an external email to the customer asking for more details OR perform a RAG on previous support tickets and suggest a "try these steps" mail to the customer. Note: that the last option if done with enough accuracy can drastically reduce the load on the support team.

2. Check inventory database regularly and send an internal emails to the procurement team OR be more autonomous and send a purchase email to the external vendor directly.

Agentic AI systems unlike traditional AI that relies on predefined instructions, can analyse complex situations, create plans, and make decisions without human involvement. This allows it to weigh options, assess available tools or sub-agents, and come up with solutions on its own. Agentic AI systems may also exhibit a certain degree of adaptability where they can observe the error of their ways, reflect and then offer course-correction to themselves. Traditional programming involves a team of software engineers carefully crafting out code to handle all the different types of scenarios that the system is meant to handle. In Agentic AI, the engineers of the system "program" the agent(s) with their role and equip them with tools. The actual creation of the plans to handle specific scenarios is more or less delegated to the LLMs plus the Agents. It is of course possible to fully cement the ways in which multiple agents of an Agentic AI system interact with each other. But in that case, we strip the system of the capability to handle unforeseen yet valid scenarios. Agentic AI is a philosophy and there are different ways to grant agency to a system. We will study some of the key tenets of this as follows.

1.6 ReAct Framework

ReAct is one of the key design philosophies that forms the foundation of Agentic AI. It was described in detail in a whitepaper titled "ReAct : Synergizing Reasoning And Acting in Language Models[4]". It was published as a conference paper at ICLR 2023. The underpinning concept of the paper is that previously LLMs abilities to Reason and Act were studied separately. But the ReAct paper focused on generating reasoning traces (why a plan was formulated) and task-specific actions (which tools were called and their consequence) in an interleaved manner. The Reasoning portion taps into the thinking capabilities of the LLM. And in a ReAct approach this interleaves with the Action portion where the LLM can tap into tool calling to interact with the real world (e.g.: explore a web page, connect to a DB, send an email, execute some code, interact with cloud infrastructure). According to the paper, ReAct outperforms imitation and reinforcement learning methods by an absolute success rate of 34% and 10% respectively for certain datasets.

[4] https://arxiv.org/pdf/2210.03629

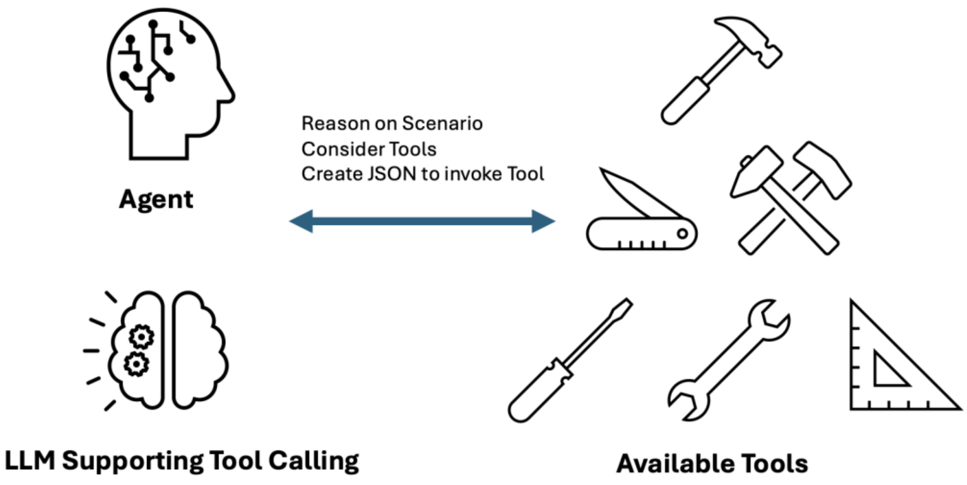


Figure 3 Essence of the ReAct technique

But why ReAct? In more ways than one, we as a species have observed the divine design of the universe and applied the same principles to our creations. For instance, the Wright brothers based several of their design decisions for their artificial flying machine by observing natural flying ones aka birds. The field of Artificial Intelligence has benefitted from a similar observation of Natural Intelligence aka ourselves. When we undertake a journey towards a goal, we usually enter a planning stage, take stock of the tools at our disposal, engage a feedback loop, reflect on our work, and take a step by step approach until we reach the final destination. This shuttling back and forth between Reasoning and Acting multiple times until we reach a goal is something we give little thought to usually. E.g. consider riding a motorbike to college. You would make a plan regarding the route. In some cases, instead of you "thinking" about it, you might just retrieve it from prior history (DB lookup). You would constantly use the visual feed through your eyes throughout your journey e.g. stop at a red light, avoid obstacles on the road. If there is a road-block you would go back to Reasoning mode and find an alternative path (Exception Handling). You might use Tools along the way. If you are unaware of an alternative path, you might launch Google Maps to aid you. You might "phone a friend". The basic premise is that children are not "programmed" or taught by their parents or teachers for each and every scenario that they will encounter in life. They are given training on the foundational principles and a finite set of scenarios. Based on that learning, they are able to interact with their environment and figure out plans and act. Sometimes, the plans might fail. Which helps in fine-tuning the underlying model. This is overall the premise of the ReAct philosophy.

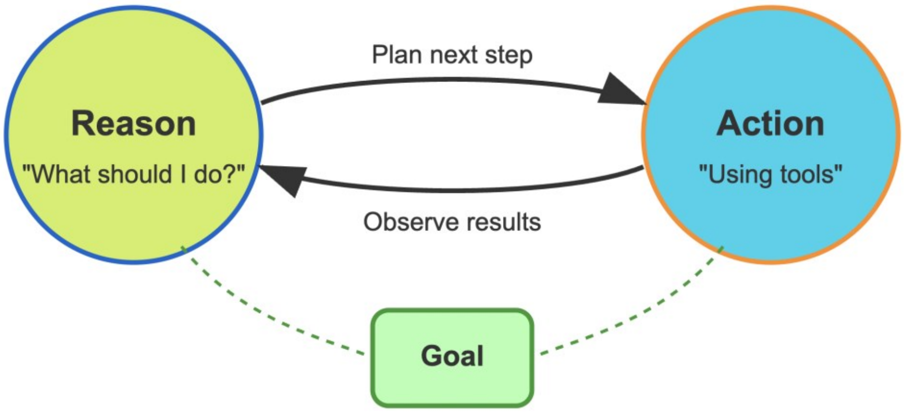


Figure 4 Interplay between Reasonings and Actions via Observations

Consider the following situation where an Agentic AI system could leverage ReAct philosophy to achieve its goal. Let's say we are building a smart assistant for a CEO. A sample use-case could be where the CEO triggers our assistant and types in "Book me a flight so that I can attend the board meeting in person." Can a plain LLM without tools handle this task? No. We need special tools for this scenario. Ideally speaking this agentic assistant would need tools like

- **Email Connector:** Read the CEO's email and/or access his calendar and determine the date, time and venue of the board meeting.
- **Flight Search & Book:** Determine suitable flights from the CEO's home base to the destination city.
- **Payment Handler:** Tool to authorize necessary payments.

Armed with the above tools a well-crafted agent can "reason" and come up with a plan. This plan would entail taking "actions" using 1 tool at a time until the final goal is achieved. There are other factors at play here like how much autonomy do we want to bestow upon this agent. How does the agent "remember" the CEO's home base or flight preferences? We'll address these and other related questions in the subsequent sections.

Coming back to the ReAct approach we see that it provides certain other advantages. Hallucinations are a major problem for Chain of Thought or just Reasoning. Because ReAct employs tools to get information from its environment, the overall probability for hallucinating goes down substantially. Having said that, the accuracy of a ReAct system has direct correlation with its ability to fetch accurate and relevant information via its Tools. Garbage In will lead to Garbage Out. E.g. If a motorist cannot distinguish between red and green lights, he's bound to end up in an accident. So it is of utmost importance to ensure that the tools available to a ReAct system function correctly.

One of the main advantages of the ReAct approach is that it allows systems to break free from the shackles of pre-defined rules of workflows. This freedom enables such agentic systems to handle unforeseen scenarios. They are able to combine LLM's reasoning with the new information obtained via tool calling to adjust their approach to reach the pre-defined goal. This approach would comprise of breaking down a large goal into smaller sub-goals or steps. Thus it is usually beneficial to use a larger and smarter model as the central planner to properly reason. The models used to power some of the peripheral agents might still employ smaller and faster models though.

One of the drawbacks of ReAct is that these systems tend to get stuck in a never-ending loop with the same reasoning and action without making any progress towards the goal. Thus it becomes important to program in a "maximum number of loops" or an "exit condition".

One of the advantages offered by the approach is clear Explainability. The overall path chosen by such a system might be complex. Nevertheless, it would still be possible to examine the reasoning and action at each and every step of the overall path. This makes it easier to debug.

1.7 CodeAct Framework

One of the key tenets of the ReAct framework is tool calling. For this to work properly we need to use select LLMs which can create structured output in the JSON format. Not all LLMs support this feature. Also in some scenarios it might take a lot of "Plan & Action" turns to arrive at the final goal. There's another technique that addresses some of these issues – CodeAct.

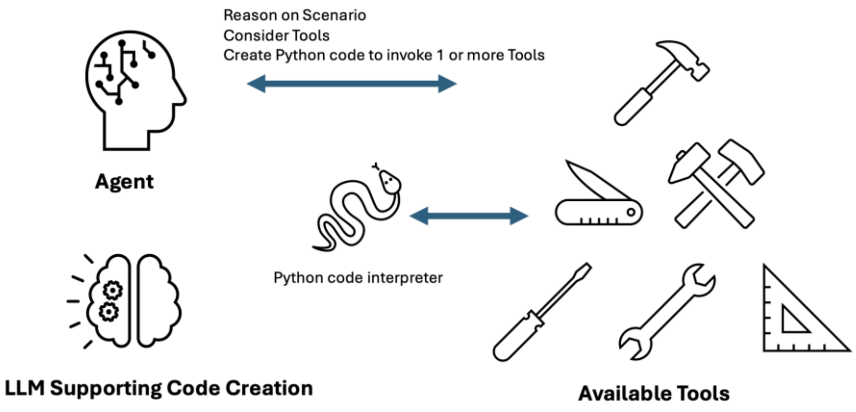


Figure 5 Essence of the CodeAct technique

CodeAct was described in detail in a whitepaper titled "Executable Code Actions Elicit Better LLM Agents[5]". The central concept of the paper is that LLMs can be used to generate code to invoke tools programmatically instead of using JSONs to invoke actions. This approach helps us to circumvent several issues.

- First off, almost all LLMs are able to generate code. As such this approach can be used to leverage a LLM without traditional tool calling support to be used to power a Planner agent.
- LLMs are usually more capable of generating accurate code rather than just generating JSON payloads. This approach improves the overall accuracy of the tool invocation as compared to just ReAct. A large corpus of code exists for training model to improve CodeAct technique's accuracy.

[5] https://arxiv.org/pdf/2402.01030

- Allowing the LLM the list of tools and the option to invoke them via code presents the opportunity to create code to handle complex scenarios and invoke multiple tools in just a single "Action". E.g.: use a simple for loop or if-else ladders interleaved with tool invocations. In the traditional ReAct approach, complex scenarios could lead to multiple invocations of various tools one after the other.
- This also opens up the possibility of tapping into pre-written Python libraries that can be used in combination with the tools that the agent is equipped with. Otherwise, in ReAct it would have become the AI engineer's responsibility to create custom tools to act as a channel or adapter for these libraries.
- By creating the code and triggering its execution via a Python interpreter the LLM can observe any failures or gaps in reaching the goal and self-adjust. The fact that a Python interpreter is already battle tested and highlight errors in the generated code or its execution work in CodeAct's favour for correction via reflection.

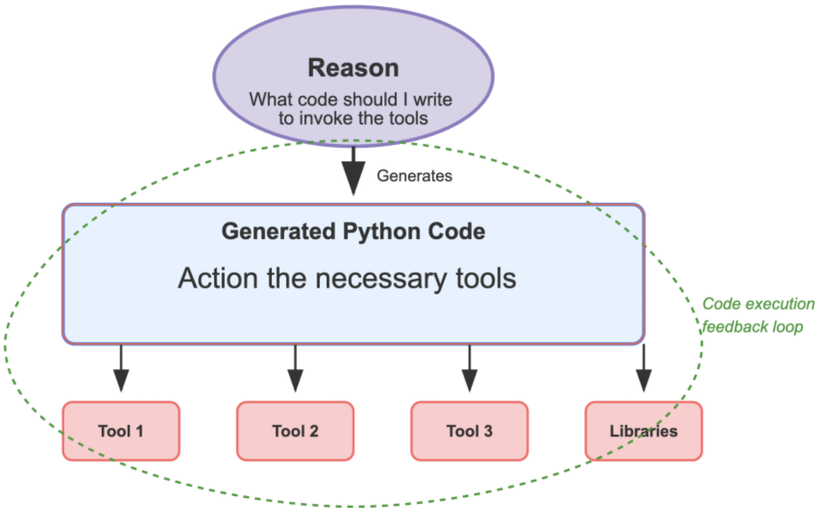

Reason
What code should I write to invoke the tools
Generates
Generated Python Code
Action the necessary tools
Code execution feedback loop
Tool 1
Tool 2
Tool 3
Libraries

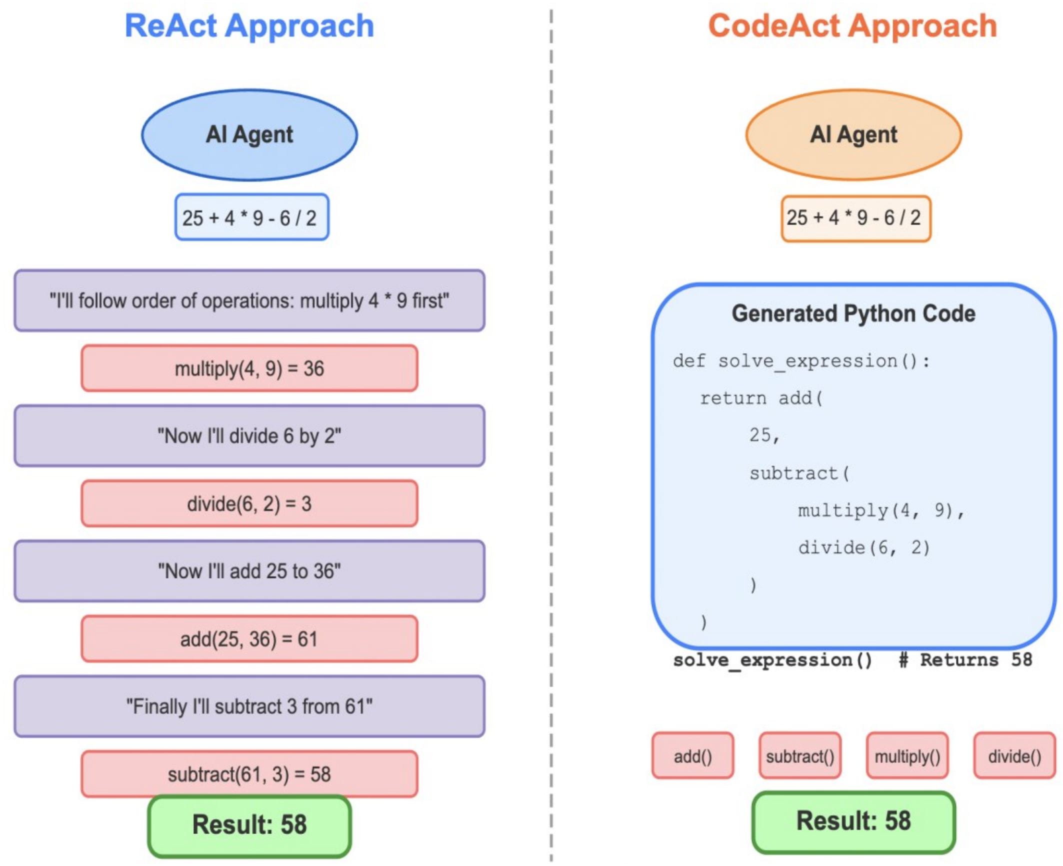
Figure 6 CodeAct Framework

LangChain has a library that caters to this framework that can be leveraged with LangGraph called as langgraph-codeact[6].

1.8 ReAct vs CodeAct

Consider an arithmetically challenged agent. Let's equip this fictional agent with 4 tools – add, subtract, divide and multiple. Each of these tools take 2 operands as input. Now, let's observe what happens when the user provides a simple expression like "25 + 4 * 9 - 6 / 2" to be calculated via each approach.

[6] https://github.com/langchain-ai/langgraph-codeact



```
def solve_expression():
  return add(
      25,
      subtract(
          multiply(4, 9),
          divide(6, 2)
      )
  )
solve_expression()  # Returns 58
```

Figure 7 ReAct vs CodeAct

Aspect	ReAct	CodeAct
Action Method	JSON-formatted tool calls	Executable code
Complexity	Multiple back-and-forth steps	Can handle complex logic in fewer steps
LLM Requirements	Needs JSON tool-calling capability	Works with any code-generating LLM
Error Handling	May need multiple attempts	Leverages Python's error messages
Integration	Limited to provided tools	Can use existing Python libraries

Use ReAct when:

- You need simple, predictable tool interactions
- You want clear visibility into each reasoning step
- Your tools are straightforward and don't require complex logic

Use CodeAct when:

- You need to process data with loops, conditions, or other complex logic
- You want to minimize the number of back-and-forth steps
- You need to leverage existing Python libraries

- Your LLM doesn't support structured JSON output

Now that we have analysed these approaches, let’s move on to the different types of agents.

1.9 Types of Agents

There are different types of agents and each of them serves a separate purpose. Let’s learn about them in detail:

1. **Planner Agent:** This agent analyses the input, considers the tools available at its disposal and either plans the next sequence of steps or just the immediate next step to be executed. Consider this scenario: The input to the Planner is to obtain some information from the web. 3 tools are available to the Planner – a Calculator, a Web Scraper and a Python Code Executor. The most probable choice over here would be to parse the incoming request and frame a request for the Web Scraper tool. It may even be possible in a very special scenario that the Planner creates some Python code to scrape from a webpage and gets that executed via the Python Code Executor. This is part of the non-deterministic aspect of an Agentic AI system’s workflow. Usually, the input coming into the Agentic AI system would be handled by a Planner agent.
2. **RAG Agent:** This agent takes care of doing a similarity search based on the task at hand and augments the current prompt with additional system-specific context. The goal is to augment the prompt with specific knowledge which is not part of the public domain. It achieved Retrieval Augmented Generation aka RAG. This could range from the basic naïve RAG to the complex graph RAG. E.g. Scenario: There is a query about how to fix a specific issue with a bespoke application. The RAG agent can employ a Jira agent to obtain relevant remediation steps from a similar past issue. The info obtained thus can be used to augment the prompt and be refined by the LLM to tailor it into a suitable response for the current query.
3. **Reflection Agent:** In several production grade GenAI applications, accuracy is more important than being economical or fast. In such cases, a simple way to boost accuracy is by using the same LLM or a different one to review the prior work created by GenAI. Compare this activity to someone pausing after having written an email to review it. The same LLM when put in a different mode (i.e. review vs edit) can help uncover issues in its own creation.
4. **HITL Agent:** A **H**uman **I**n **T**he **L**oop agent is merely a provision to allow a human SME to interact with the Agentic AI. It could take the form of an Agent awaiting a comment to be posted by a human on a Jira issue by tagging the Jira agent’s user. It could be handled by the system prompting a question to the human on a GUI. The purpose could be to get the human to provide course correction during the planning stage. It could also be used to get feedback or clarification during any of the various steps that the Agentic AI takes to find a balance between complete autonomy and convergence with desired outcome.

Note that some agents might invoke tools. These tools may or may not be powered by GenAI. It acts as the hands and feet for the overall Agentic AI system. This could be a simple tool like a Calculator for handling Basic Arithmetic (something that LLMs are notoriously bad at). It could also be a relatively complex tool that connects to a Database and obtains the relevant records. It could act as an adaptor to a GraphQL or Rest API that the Agentic AI system needs to interface with. E.g.: a Jira tool that is used to create, read, update or delete Jira tickets.

1.10 Types of Agentic AI systems

Now that we have learnt about the different types of Agents lets delve into different types of Agentic AI systems. There are different ways of classifying Agentic AI systems. We'll primarily focus on the Architectural, Workflow and Invocation aspects in this section.

Architecture-wise these systems can be classified into: Single Agent with Multiple Tools & Multiple Agents with Multiple Tools.

Single Agent with Multiple Tools: This is the most basic type where a single AI agent is equipped with and utilizes several tools to accomplish tasks. It is quite basic yet powerful. Such systems could be put together easily yet still potentially handle several situations which otherwise would take a lot of traditional programming effort. An example could be a Planner Agent that is equipped with a DB tool, a Calculator tool and an Email tool. It could easily handle scenarios like "Look up the customers who have not interacted with us since a quarter and send them an email with a discount offer."

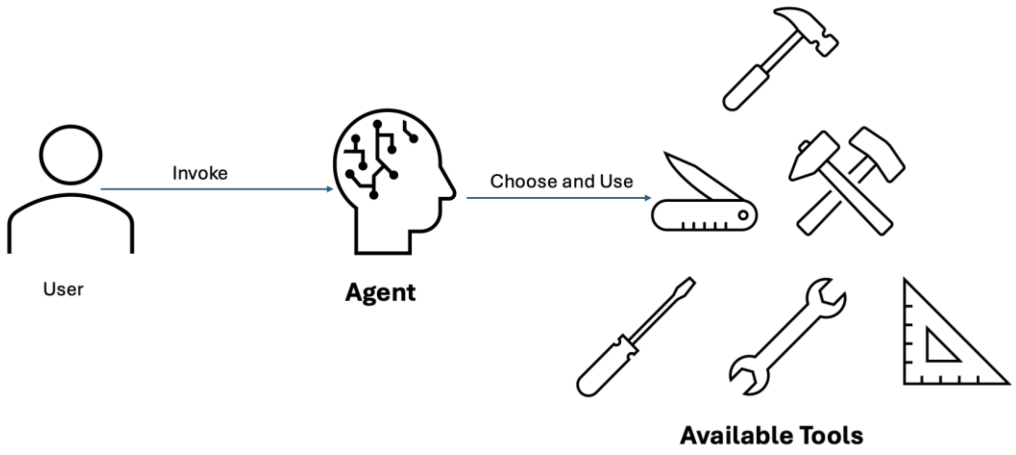


Figure 8 Single Agent with Multiple Tools

Multiple Agents with Multiple Tools: This is the more complex architectural variant for Agentic AI systems. This is where the scenarios to be handled are so complex that it becomes necessary to break down the overall goal into sub-goals that can be addressed by distinct agents each differentiated by a role. Each agent might be equipped with different tools suited for the role that the agent is programmed to play. For instance, the Ticket Management agent might be equipped with tools to perform CRUD operations on a ticketing system like Jira. The Infrastructure Management agent might be equipped with tools to interact with the infrastructure on the cloud e.g.: AWS, Azure, GCP, IBM etc. These agents might be programmed to invoke not just the tools but the other peer-agents in the eco-system. LLMs are powerful but they cannot do everything. It becomes necessary when dealing with complex scenarios to decompose a large problem into smaller tasks for an agent to tackle and then pass on to the next agent in the graph.

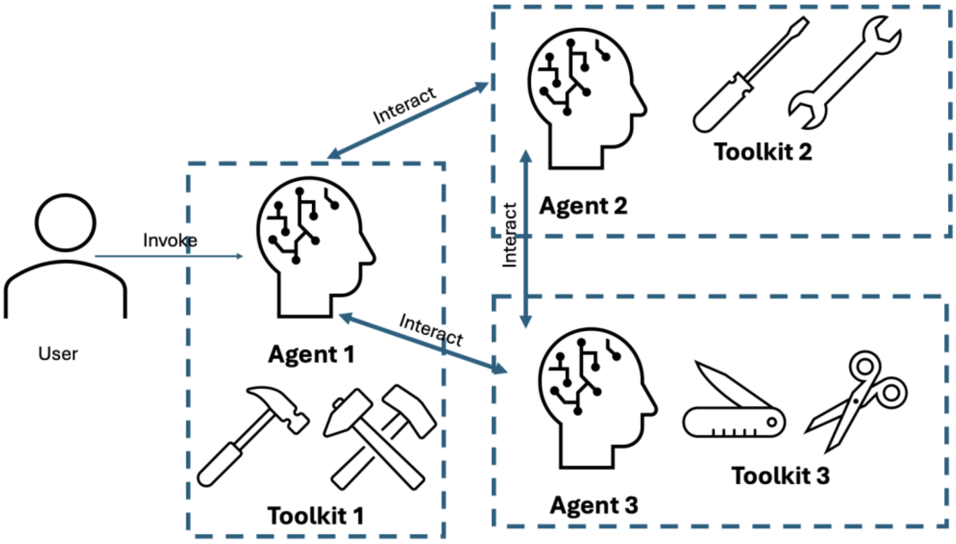


Figure 9 Multiple Agents with Multiple Tools

Now let's move to another factor on which we can categorize Agentic AI systems. **Workflow-wise** these systems can be classified into: Sequential, Hierarchical and Complex systems.

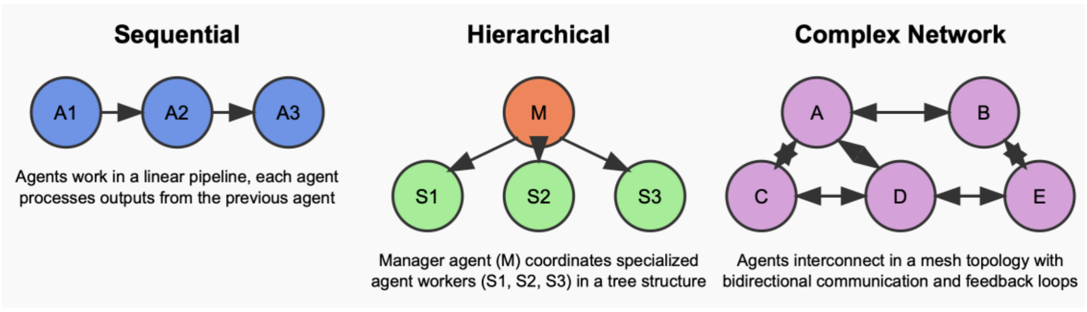

Sequential
A1
A2
A3
Agents work in a linear pipeline, each agent processes outputs from the previous agent
Hierarchical
M
S1
S2
S3
Manager agent (M) coordinates specialized agent workers (S1, S2, S3) in a tree structure
Complex Network
A
B
C
D
E
Agents interconnect in a mesh topology with bidirectional communication and feedback loops

Figure 10 Workflow based classification

Sequential: This is where agents are arranged in sequence to achieve a large goal by decomposing it into smaller sub-goals and addressing each sub-goal with an agent. The output of one agent is passed as input to the next agent. And the output of the final agent is passed back to the user. This is very similar to piping used in Linux commands. (e.g. **cat file.txt | grep "error" | sort**)

In this type of a system, the workflow is highly deterministic. We know exactly which agent is going to be invoked at which stage. The tool choice by each agent is still dependent on the input and the LLM's actual working during each execution. That is the portion that contributes to a small amount of non-determinism to the system's behaviour.

Hierarchical: Wouldn't it be convenient if we could simply equip one single agent with a multitude of tools and let it solve all possible problems? But that's not practical. A single agent is usually not able to do justice to problems once the inherent complexity of the problem increases beyond a certain threshold. This threshold does increase based on the actual LLM that powers the agent. But it doesn't disappear completely, not yet at least. Like humans, agents are able to achieve more when they collaborate rather than work in isolation.

The easiest way to foster this collaboration amongst agents is to create a hierarchy. We program a Planner agent which analyses the input and based on the goal and the agents available to it comes up with a detailed plan. Once the problem is decomposed into smaller sub-goals, they are handed over to the agents 1 level below the Planner agent. These agents work on their own assignments and return the output to the Planner agent. The Planner agent's job becomes to assign tasks, aggregate outputs & decide which agents to invoke. The actual paths of this agentic tree that get traversed for each execution of the system can be quite different. At the same time, this is what allows this type of a system to handle more scenarios as compared to a Sequential system. Do note, that the Hierarchical system need not just stop with 2 levels of agents. Each of the agents below the Planner agent could have multiple levels of agents available below them to do their bidding.

Complex: This is the most non-deterministic variant of Agentic AI systems. In this system the task assignments, agent invocations and output aggregation are not reserved for just the Planner agent. There was still a certain angle of determinism visible in the Hierarchical variation where the agents go back and forth with the Planner agent. In the Complex variation, agents are aware of their peers and have accessibility of each other. They are able to make use of their peer agents to help them with their tasks. This leads to a variety of actual invocation paths that the system may exhibit depending on the scenario. This is the most adaptive form of Agentic AI systems.

Next, **Invocation**-wise these systems can be classified into: Proactive & Reactive

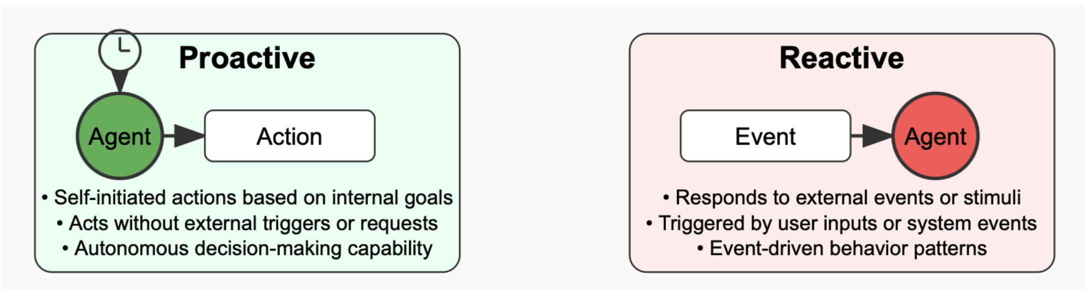

Proactive
Agent
Action
• Self-initiated actions based on internal goals
• Acts without external triggers or requests
• Autonomous decision-making capability
Reactive
Event
Agent
• Responds to external events or stimuli
• Triggered by user inputs or system events
• Event-driven behavior patterns

Figure 11 Invocation based classification

Proactive: This type of a system does not need to be triggered externally. These are the systems which are either continuously or regularly looking at one or more feeds and decide whether to act or continue observing. An example could be a system that is constantly going through application logs and jumps into action once it detects an anomaly. Another example could be a system that is set to run at a 5 min interval to analyse records in a database and act on it. Basically, this system is always in an ON mode and observing. When it finds the right scenario, it will act.

Reactive: This type of a system is passive and will not activate unless an external trigger puts it in motion. This could be your standard email handling system. There could be a connector built with an email inbox that as soon as an email is received it will make say an API call to invoke the Agentic AI system to read the email and take the necessary action. This system is by default in an OFF mode and runs only when triggered.

These are some of the ways in which we can classify Agentic AI systems. These classifications are neither mutually exclusive nor exhaustive. The point is to understand these different aspects of Agentic AI systems to provide AI engineers with the adjectives to precisely describe a system. E.g. "I'm building a sequential, multi-agent-multi-tool, reactive Agentic AI system".

1.11 Control vs Autonomy

An important aspect of designing Agentic AI systems is to determine the desired balance between Control and Autonomy. Either the system will be fully autonomous or fully controlled by the users. But it cannot be both. The job of the architect here becomes to gauge the right level of control that would be required for the business use-case. The architect then needs to ensure that the system sits appropriately at the right level on the spectrum of Human-AI collaboration. There are 3 main decision strategies that we'll explore to better understand this trade-off between these 2 important factors.

Micro-Manage

In this strategy we favour control over autonomy. The system itself might be capable of handling different scenarios. Yet, we enforce restrictions on it to propose its plan to the user(s), receive feedback and only then execute the actions. Simply put, it has a human placed in between the Reason and Act steps. This strategy affords complete oversight. If the user performs all reviews accurately there is no chance of the overall system making mistakes. On the flip side, the system can only proceed when a human is around to approve/modify the plan proposed by the system. As a result, the speed at which the human SMEs (aka Subject Matter Experts) review and interact with the system becomes its limiting factor. If the SME is not around, the system can simply not act. If the SME does not review accurately, the system might execute an erroneous plan. This is the strategy that we select in high-stakes scenarios like medical diagnostics or legal document drafting. The overall execution can be best explained by the following illustration.

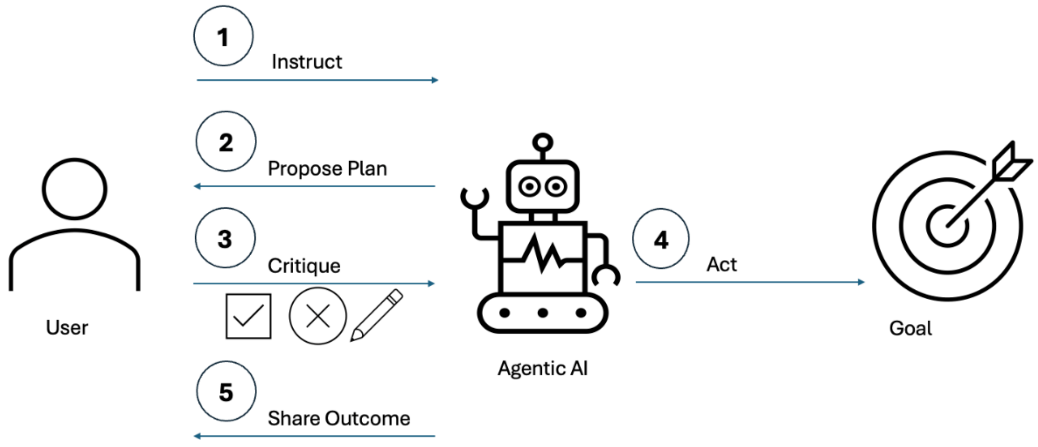


Figure 12 Micro-Manage

1. The user or some pre-programmed event will trigger or activate the system.
2. The system (which may be composed of multiple agents and tools) will formulate a plan to reach its goal. The plan will be shared with the user for review.
3. There are 3 broad courses of actions that the user might take.
    a. The user might approve the plan without making any changes.
    b. The user might reject the plan completely.
    c. The user might edit the plan and then approve it.
4. The system will plan its actions according to the user's review.
    a. In case of an approval the system will simply execute the unaltered plan.
    b. In case of a rejection the system might lead to inaction and termination OR it might go back to the planning step and propose a different plan. This depends on how the rejection flow is designed.
    c. In the case of an edit the system will execute the altered plan.
5. Finally, the outcome of the system's actions will be conveyed to the user. This might be displayed on a screen OR sent via an email OR read out loud etc.

Delegate and Forget

This strategy is the polar opposite of Micro-Manage. Here a greater value is placed on autonomy rather than control. In this strategy, the human relinquishes control over the system after granting full responsibility of the outcome to the system. This is possible in scenarios where either the cost of failure is minimal OR the accuracy of the system has been proved to be extremely high. It maximizes efficiency. Once configured properly, it does not need humans to be around to redirect moves or approve actions. Hence its highly scalable. Such systems would need a high level of exception handling and near-constant automated testing. Since this system is not throttled by the need for human review or intervention it is only limited by the system's ability to reason and act. This is the most efficient version of Agentic AI possible. This could be employed for internal classification of support tickets. The overall execution can be best explained by the following illustration.

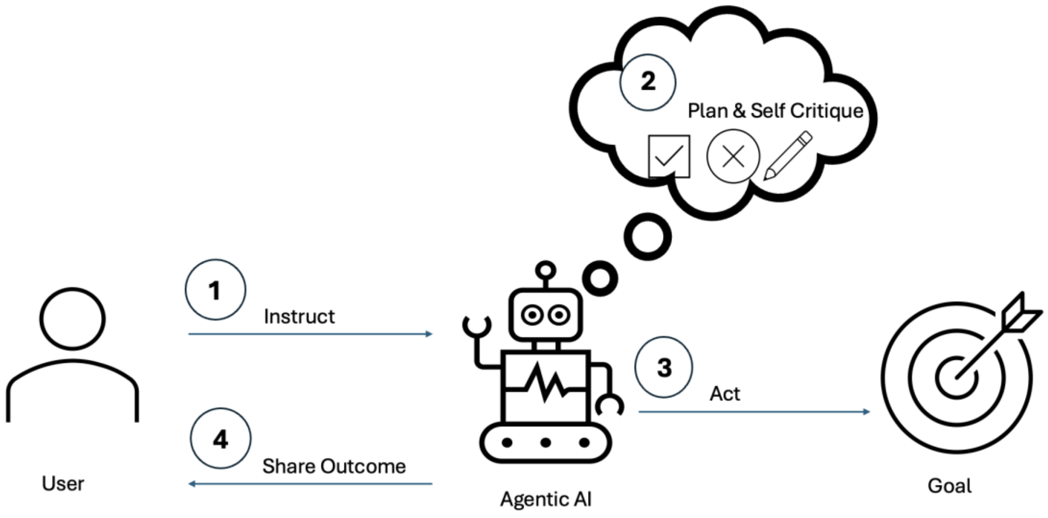


Figure 13 Delegate and Forget

1. The user or some pre-programmed event will trigger or activate the system.
2. The system will formulate a plan to reach its goal. The system itself will self-review its plan and make any changes deemed necessary. This entire step might be repeated multiple times.
3. The system will action the steps identified in the plan.
    a. Steps 2 and 3 may not always be sequential. Based on the system design and the actual scenario there might be multiple possible interweaving of Planning and Acting. (E.g. Plan – Act – Plan – Act OR Plan – Act – Act – Act – Plan, etc)
4. Finally, the outcome of the system's actions will be conveyed to the user.

Delegate and Manage

This strategy is a hybrid of Micro-Manage and Delegate and Forget. Here, we aim to find a sweet spot between the efficiency afforded by autonomy and the reliability provided by oversight. Here the system formulates the plan and then arrives at a decision either for each plan or for each step of each plan. It will try to determine either based on its self-perceived impact of potential failure OR pre-programmed rules whether it is allowed to execute the plan on its own OR seek review/permission for a human. The basic principle is that for low-risk tasks or scenarios the system might be programmed to go ahead with the execution. And likewise for high-risk tasks or scenarios, the system might be programmed to seek a review or redirection from the user. So, in essence, it will function as a "Delegate and Forget" system for low-risk scenarios. And it will function as a "Micro-Manage" system for high-risk scenarios. Obviously, the system will need to be taught which scenarios or steps are low-risk and which are high-risk. The classification of the steps itself might be left to a LLM or a more stringent rules-based engine. Ultimately, the system tries to gain a productivity boost for its users by allowing the system to behave autonomously for low-risk or routine tasks or workflows. And it allows the users to remain in control for the high-risk ones. It may also so happen that over a period of time, the approval of the users is deemed unnecessary due to an impeccable reasoning demonstrated by the system. This may also happen due to the system learning over a multitude of interactions with the users about its own mistakes. The accuracy of the system's planning might also simply improve over time due to refinements in the overall prompting or advancements in the underlying foundation models being employed. A sample scenario could be software development itself. The overall execution can be best explained by the following illustration.

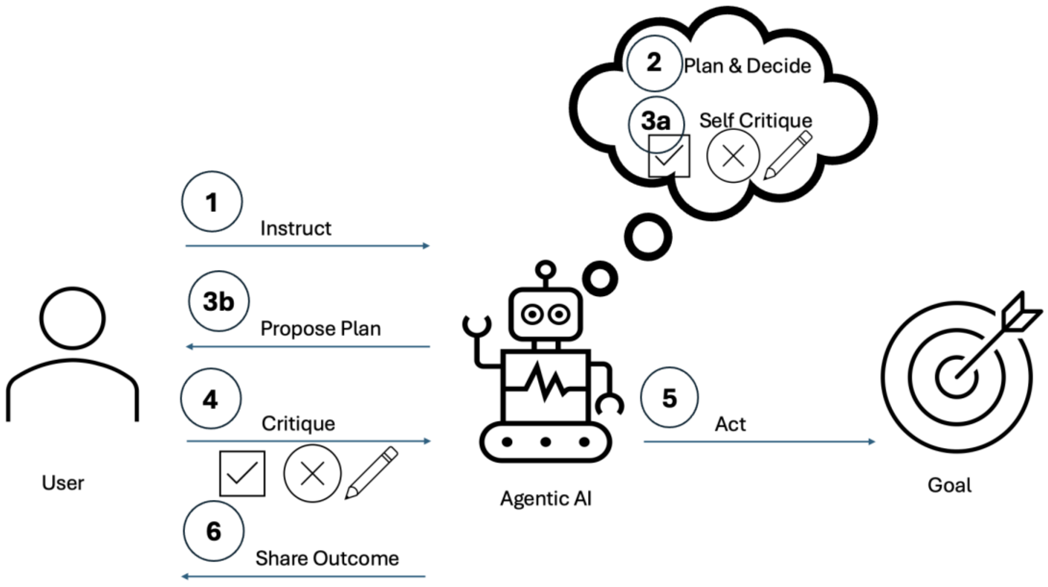


Figure 14 Delegate and Manage

1. The user or some pre-programmed event will trigger or activate the system.
2. The system will formulate a plan to reach its goal. The system will decide whether it needs to function autonomously for its next step OR seek review from the user.
3. Based on the decision made in the previous step the system will choose between the following 2 options:
    a. The system itself will self-review its plan and make any changes deemed necessary.
    b. The system will share the plan with the user for review.
4. If sent to the user for review, the user might approve/reject/modify the plan.
5. The system will act on the plan either self-approved by itself or the user. The system might repeat steps 2 to 5 multiple times in a different order to reach its goal.
6. Finally, the outcome of the system's actions will be conveyed to the user.

As we have seen, the 3 different strategies offer different ways of building Agentic AI systems. It is up to the AI engineering team to study the requirements and suggest the appropriate strategy. It is essential to focus on questions including but not limited to:

1. What will be impact of the system making an incorrect decision?

2. Can mistakes made by the system be corrected by human intervention post execution?
3. Which factors contribute to deciding which step is high risk vs low risk?
4. What is the risk tolerance for the overall system?
5. What is the accuracy of the system in different scenarios?
6. How complex are the sub-tasks that the system will need to execute?
7. Are user inputs for the system open-ended or restricted?
8. How fast is the system supposed to respond to requests?

The following chart illustrates the overall relationship between autonomy and control. It also indicates the suitability of the different decision strategies.

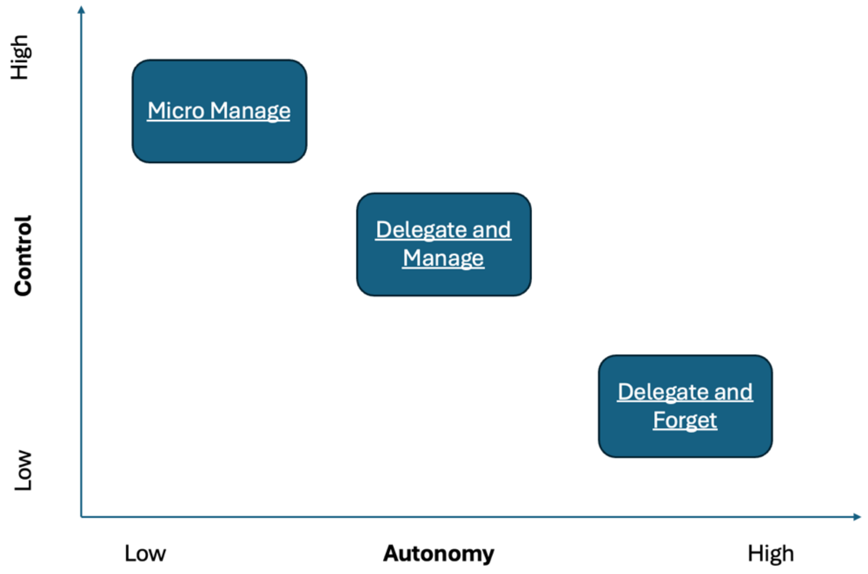


Figure 15 Control vs Autonomy

1.12 Summary

Agentic AI Evolution

- **Traditional Programming**: Deterministic, predictable systems with fixed rules.
- **Gen AI (Narrow AI)**: Fixed workflows with some non-deterministic outputs (e.g., document summarization).
- **Agentic AI**: Highly non-deterministic in both outputs and workflows, prioritizing adaptability over predictability.

Narrow AI vs. Agentic AI

- **Narrow AI**: Operates within rigid workflows, with limited flexibility and predictable paths (e.g., chatbots).

- **Agentic AI**: Features flexible workflows, autonomous decision-making, and adaptability for complex tasks (e.g., self-driving cars), boosting productivity by reducing human dependency.

-

Key Concepts

- **Agent**: An entity powered by an AI model (often a large language model, LLM), equipped with tools, and guided by a goal. Key traits include autonomy, decision-making, and goal-oriented behaviour.
- **Agentic AI System**: A network of agents, tools, and LLMs working together, designed for adaptability and unpredictable workflows to handle unforeseen scenarios.

-

Frameworks

- **ReAct Framework**: Interleaves reasoning and acting, using tools to ground decisions in real-world interactions, reducing hallucinations and enhancing adaptability.
- **CodeAct Framework**: Uses code generation to invoke tools, offering greater accuracy and flexibility, especially for complex tasks, by leveraging LLMs' coding capabilities.

-

Types of Agents

- **Planner Agent**: Plans steps using available tools.
- **RAG Agent**: Retrieves and augments information for context-specific responses.
- **Reflection Agent**: Reviews and refines system outputs.
- **HITL Agent**: Incorporates human feedback for oversight or correction.

-

Classification of Agentic AI Systems

- **Architecture**:
  - Single Agent with Multiple Tools.
  - Multiple Agents with Multiple Tools.
- **Workflow**:
  - Sequential: Fixed, step-by-step process.
  - Hierarchical: Planner delegates to sub-agents.
  - Complex: Dynamic agent interactions.
- **Invocation**:
  - Proactive: Self-triggers based on monitoring.
  - Reactive: Responds to external triggers.

-

Control vs. Autonomy

- **Micro-Manage**: High control, human reviews every plan (e.g., medical diagnostics).
- **Delegate and Forget**: High autonomy, system acts independently (e.g., ticket classification).
- **Delegate and Manage**: Hybrid approach, system seeks human input for high-risk tasks while acting autonomously for low-risk ones.

1.13 Exploratory Questions

1. Can you think of examples that GenAI can handle that traditional ML cannot?
2. Where would you use CodeAct as opposed to ReAct?
3. Where in the real world of "human" agents do you see patterns similar to Sequential, Hierarchical and Complex patterns?
4. What are the ethical and financials implications of non-determinism associated with GenAI?
5. What is more valuable – Autonomy (of Agents) or Oversight? How to find the balance?

For feedback/questions on this content use https://cladiusfernando.com/contact/