Microsoft logo white text version

Zzz

zzz



Responsible AI Workshop

Understanding intelligent agents in Artificial Intelligence

A starter guide for data engineers, data scientists, AI developers, and other AI practitioners to harness generative AI and agentic structures in their applications

Version 1.0 - January 2025

This page is intentionally left blank.

Table of contents

[Notice 4](#_Toc189149439)

[About this guide and the learning objectives 5](#_Toc189149440)

[Objectives of this guide 5](#_Toc189149441)

[Non-objectives of this guide 5](#_Toc189149442)

[Guide elements 5](#_Toc189149443)

[Requirements 6](#_Toc189149444)

[Module 1: Introducing intelligent agents and agentic applications 7](#_Toc189149445)

[What are intelligent agents? 7](#_Toc189149446)

[What about agentic applications? 8](#_Toc189149447)

[A glimpse into AI agents’ structure & characteristics 9](#_Toc189149448)

[Towards Microsoft Copilots’ agents? 11](#_Toc189149449)

[Module 2: An introduction to different agentic architectures 13](#_Toc189149450)

[Main AI agents' structural challenges 13](#_Toc189149451)

[Single vs. multi-agents’ architecture 15](#_Toc189149452)

[Microsoft Copilots’ architecture 18](#_Toc189149453)

[Module 3: Microsoft Copilots’ ecosystem 20](#_Toc189149454)

[Build your own Copilot agent starting with Microsoft 365 Copilot 21](#_Toc189149455)

[Create agents with Azure AI Foundry 21](#_Toc189149456)

[Module 4: Building multi-agents systems with AutoGen 24](#_Toc189149457)

[Build a simple agentic travel application 25](#_Toc189149458)

[Build an advanced agentic travel application 28](#_Toc189149459)

[Module 5: Understand and mitigate agent’s risks and limitations 35](#_Toc189149460)

[Accentuation and creation of new risks linked to AI agents 35](#_Toc189149461)

[Consequences of these risks 36](#_Toc189149462)

[Measures implemented to address these risks 37](#_Toc189149463)

[Agents and benchmarks 39](#_Toc189149464)

[As a conclusion 41](#_Toc189149465)

[Bibliography 42](#_Toc189149466)

[To go beyond 46](#_Toc189149467)

Notice

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user.

A drawing of a face

Description automatically generated

Attribution 4.0 International (CC BY 4.0)

Microsoft and any contributors grant you a license to this document under the [Creative Commons Attribution 4.0 International Public License](https://creativecommons.org/licenses/by/4.0/legalcode), see the [LICENSE](https://github.com/microsoft/responsible-ai-workshop/blob/main/LICENSE) file, and grant you a license to any code in the repository under the [MIT License](https://opensource.org/licenses/MIT), see the [LICENSE-CODE](https://github.com/microsoft/responsible-ai-workshop/blob/main/LICENSE-CODE) file.

Microsoft, Windows, Microsoft Azure and/or other Microsoft products and services referenced in the document may be either trademarks or registered trademarks of Microsoft in the United States and/or other countries. The license for this document does not grant you rights to use any Microsoft names, logos, or trademarks. Microsoft's general trademark guidelines can be found at <http://go.microsoft.com/fwlink/?LinkID=254653>.

Privacy information can be found at <https://privacy.microsoft.com/en-us/>

Microsoft and any contributors reserve all other rights, whether under their respective copyrights, patents, or trademarks, whether by implication, estoppel or otherwise.

# About this guide and the learning objectives

## Objectives of this guide

This guide aims at introducing agents in artificial intelligence on a large scale, giving clear definitions and examples as well as research benchmarks of different frameworks and an analysis of agentic AI in the scope of responsible AI, i.e. a safe, secure, and trustworthy AI.

## Non-objectives of this guide

This guide is not meant as an introduction to AI, nor as an in-depth guide on agentic frameworks and the current best research papers. Though it briefly touches on different agentic frameworks, it is not a complete overview of the research landscape.

This guide is not a technical demonstration, nor a product demonstration even if some specific offerings and solutions are being outlined throughout the guide.

This guide is not aimed at introducing the building blocks of responsible AI (RAI). For an introduction to RAI, and notably through Microsoft’s ongoing journey in the space, please refer to the guide [Establishing your own Responsible AI journey for your (non-generative vs. generative) AI-powered solutions](https://github.com/microsoft/responsible-ai-workshop/blob/main/responsible-ai-journey/docs/establishing-your-own-responsible-ai-journey.docx).

Note For a complete overview of Microsoft’s resources designed to help you responsibly implement (non-generative, a.k.a. traditional, vs. generative) AI systems, please refer to the [Microsoft Responsible AI resources page](https://aka.ms/rairesources).

## Guide elements

This guide consists of four main parts.

Module 1 introduces agents and agentic AI, providing definitions and real-world examples to better understand their advantages in dynamic environments.

Module 2 provides an overview of different agentic architectures, focusing on the structural challenges and construction of agents. It covers aspects such as memory, perception, and planning, and discusses both single and multi-agent systems. The module also explores various academic frameworks and re-evaluates the role of agents in Microsoft Copilots.

Module 3 explores the Microsoft ecosystem for developing Copilots agents and autonomous agents.

Module 4 demonstrates how to use the AutoGen open source framework to integrate agents into AI solutions by developing a travel agentic application.

Finally, Module 5 explores the risks and limitations of intelligent agents, a.k.a. AI agents, within responsible AI practices, thus addressing issues like unpredictability of user inputs, complexity of internal executions, and variability of operational environments. It proposes solutions such as transparency, explicability, interpretability, moderation, and governance.

## Requirements

The Responsible AI Workshop repo on GitHubcontains a hands-on tutorial for this guide. If you would like to use it, we recommend that you read the following notes:

* [Cloning this workshop GitHub repo](https://github.com/microsoft/responsible-ai-workshop/blob/main/perequisites/cloning-the-repo.md).
* [Fulfilling the prerequisites for the workshop](https://github.com/microsoft/responsible-ai-workshop/blob/main/perequisites/fulfilling-prerequisites.md).
* [Getting started with Azure for your environment](https://github.com/microsoft/responsible-ai-workshop/blob/main/perequisites/getting-started-with-azure.md).
* [Deploying a Generative AI model in Azure and using it in Python](https://github.com/microsoft/responsible-ai-workshop/blob/main/perequisites/deploying-a-model-in-Azure-and-using-it-in-python.md).
* [Installing Microsoft Visual C++](https://github.com/microsoft/responsible-ai-workshop/blob/main/perequisites/installing-microsoft-visual-C%2B%2B.md).

# Module 1: Introducing intelligent agents and agentic applications

“We’re not saying autopilot; we say co-pilot for a reason: we don't think the co-pilot can be independent of human agency.”

*Rajesh Jah, Microsoft Executive VP*

Intelligent agents, a.k.a. AI agents, have often been described as the next major direction in AI research and innovation following the trends set by generative AI (GenAI) models, i.e., large language models (LLMs) and small language models (SLMs).

However, the concept of agents is NOT new. Architectures like Microservices [1], widely used in DevOps, allow the development of applications using simple and independent blocks, i.e., microservices. Additionally, we cannot discuss this topic without mentioning the Microsoft Power Platform, particularly [Power Automate](https://www.microsoft.com/power-platform/products/power-automate), which enables the automation and orchestration of tasks across different applications. Before delving into the specifics of AI agents, it is crucial to establish a clear definition, as their interpretations are as numerous as their potential applications.

In a general context, agents are entities that perceive their environment through sensors and act autonomously upon effectors. For example, a programmable thermostat that regulates the temperature of a room by activating or deactivating the heating/cooling system according to set parameters is an example of an agent [2] [3].

## What are intelligent agents?

In the narrow scope of AI world, AI agents, are AI models and algorithms that can autonomously make decisions in a dynamic environment [4].

For instance, consider a travel AI agent that can plan your entire vacation. Such an agent can make personalized suggestions based on your preferences, such as favorite activities, preferred destinations, and budget constraints. It can research information on the internet about prices, accommodations, and local attractions, and then create a detailed itinerary for your trip.

It’s important to highlight that an AI agent operates within a predefined environment composed of the data fed into it, such as files or web pages, and the tools of action it can utilize, like various plugins. This structured setting ensures that the agent can optimally navigate and interpret the information, while also providing a secure framework that limits exposure to unverified or potentially harmful data sources.

## What about agentic applications?

These AI agents can be combined with other agents or integrated into more sophisticated modules to create what are known as agentic applications. This synergy allows for a higher level of functionality and complexity, empowering the system to tackle a wider array of tasks with greater efficacy [5].

### Using agents to automate tasks

Imagine the previous travel AI agent working in tandem with a language translation agent and a web booking agent to create a comprehensive travel planning application. This integration exemplifies an agentic application, where the collaboration of multiple AI agents enhances task automation.

The travel AI agent begins by suggesting destinations based on your preferences, budget, and available dates. Next, the language translation agent steps in to facilitate communication with local service providers, ensuring your requests are accurately understood. Simultaneously, the web booking agent manages all reservations, including flights, accommodations, and activities, and handles any changes or cancellations seamlessly.

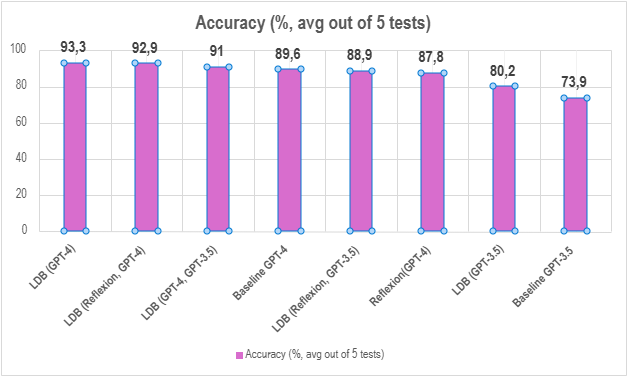
Throughout the entire process, these agents work together to provide personalized recommendations and updates, making your travel experience both efficient and enjoyable. This example illustrates how combining multiple AI agents can revolutionize task management, offering a seamless, superior user experience that adapts to your needs in real time with limited direct supervision.

### Using agents to increase performance

We have seen that agentic applications can be used for task automation. They can also significantly enhance performance. One of the pioneering areas where LLMs are already being leveraged to boost performance is code generation. In this domain, agents play a crucial role in refining outcomes. For instance, researchers Zhong et al. and Shinn et al. conducted experiments using two distinct agent frameworks: Reflexion and LDB [5].

These frameworks aim to enhance the performance of LLMs in code generation and debugging, demonstrating the practical benefits of agentic approaches. We will further explore these architectures in [Module 2](#_Module_2:_An).

To perform their evaluation, they used the HumanEval benchmark [6] which consists of 164 hand-crafted programming challenges, each including a function signature, docstring, body, and several unit tests, averaging 7.7 tests per problem. These challenges assess a model's understanding of language, algorithms, and simple mathematics, and are comparable to simple software interview questions. Finally, the accuracy is measured by how often one of the top 10 answers generated by the model is correct. They ran the benchmark five times and averaged the results to determine the final accuracy.



Let’s quickly summarize the important results:

* A single agent (LDB alone or Reflexion alone) offers an increase in performance, particularly for GPT-3.5 which shows a 6.3-point increase using LDB. The same framework offers a smaller increase of 3.7 points using GPT-4.
* A multi-agentic framework combining Reflexion and LDB shows an even bigger increase in performance, reaching an average of 92.9% using GPT-4. This same multi agentic framework based on GPT-3.5 reaches almost the same level of performance as the bare GPT-4 model.

Note LDB with GPT-4o reaches the almost perfect score of 98.2% on the same test. [6].

Therefore, AI agents have demonstrated their ability to improve a model's performance without altering the model itself. This is an encouraging development for the future of LLMs, given that creating these models is increasingly costly, time-consuming, and requires substantial computing power. Emphasizing the creation of agentic frameworks to boost existing models' performance might be an adequate alternative for many applications. However, our analysis focused only on performance, while cost and inference time are also critical factors. We will address these aspects in [Module 5](#_Module_5:_).

By showcasing the potential of integrating various AI agents, we can see how agentic applications can dramatically enhance task automation and performance, moving beyond the capabilities of a LLM (vs. SLM) or a single AI agent.

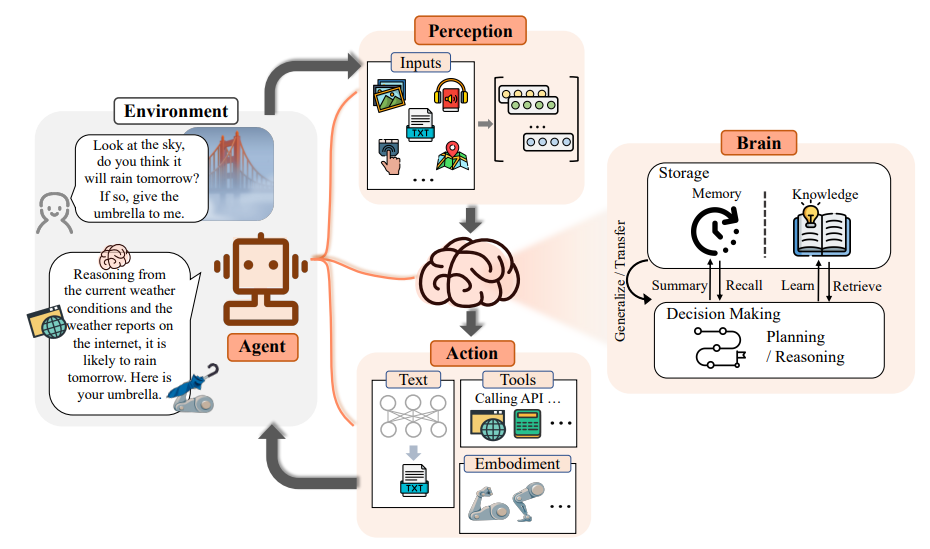
## A glimpse into AI agents’ structure & characteristics

Now that we have seen the significant impact and potential of agents in enhancing models’ performance, it is time to delve deeper into the structure of AI agents. In general, an agent can be schematized with three main components: a brain, a perception module, and an action module. The brain represents the underlying model and planning capabilities, enabling the agent to strategize and sequence actions to achieve specific goals. The perception module encompasses the tools and sensors that allow the agent to process information from its environment, making it context-aware and interactive. Finally, the action module is responsible for executing tasks using various tools and functions, thus allowing the agent to perform effectively in real-world or virtual scenarios [7].

In the AI world, the brain of an AI agent is fundamentally the model, typically a LLM, endowed with advanced planning capabilities and a persona that guides its interactions and decisions. This brain is responsible for strategizing and sequencing actions to achieve specific goals. It also incorporates memory capacities, enabling the agent to recall past interactions and refine its responses over time, fostering a sense of continuity and learning.

The action module comprises the various tools and functions connected with the LLM, much like plugins that extend the agent's capabilities. These can include APIs, external software, and other functionalities that the agent can invoke to execute tasks efficiently, making the agent versatile and adaptable to different scenarios.

Finally, the perception module represents the data input fed into the agent. This data can be multi-modal, encompassing text, images, sound, files, and other sensory information. By processing these diverse inputs, the agent can gain a rich, context-aware understanding of its environment, thereby enhancing its interactivity and responsiveness. Together, these modules form a cohesive system where each component plays a critical role in the overall functionality of the AI agent.



Structure of an AI agent [7]

This framework naturally grants AI agents several crucial characteristics. Autonomy is achieved through the agent's ability to operate independently, making decisions and executing tasks without or with little human intervention. Reactivity is enhanced by the perception module, allowing the agent to quickly adapt and respond to changes in its environment. Proactiveness is a result of the planning capabilities embedded in the brain, enabling the agent to anticipate needs and take initiative. Lastly, social activity is facilitated by the persona and memory components, which help the agent interact seamlessly and intuitively with users, other agents and applications, maintaining a coherent and engaging dialogue over time [7] [5] [4].

## Towards Microsoft Copilots’ agents?

In the Microsoft ecosystem, the agent concept is implemented under the name of Copilot agent. Copilots form an entire ecosystem in themselves.

They take different forms, whether:

* In your search engine with [Microsoft Copilot](https://copilot.microsoft.com/),
* In GitHub with [GitHub Copilot](https://github.com/features/copilot)
* Or directly in your Microsoft 365 applications with i) [Microsoft 365 Copilot](https://www.microsoft.com/en-us/microsoft-365/copilot/enterprise), the “best-in-class” personal AI assistant for work, and ii) [Microsoft 365 Copilot Chat](http://m365copilot.com/), a new offering that adds pay-as-you-go agents to the existing free chat experience for Microsoft 365 commercial customers [8].

In [Module 3](#_Module_3:_), we'll look at the tools and platforms you can use to deploy them.

However, if both autonomous agentic applications and copilots promise to assist you in daily tasks through their AI capabilities, they intend to do so with very different approaches, detailed below.

Technically speaking, Copilots are agentic applications: they use agents to extend their capabilities. With the Copilot wave 2 [9], customers are now able to create copilots in the form of standalone agents that act independently to plan tasks. However, Copilots are far from being what we consider the final form of the agentic application as a concept.

As their name suggests, they are not completely autonomous and are made to work side by side with their human counterpart by assisting them. They do not have the ability to complete whole tasks by themselves but rather are able to prepare those tasks for their human user. This allows the latter to perform quicker and finalize tasks after reviewing the copilot’s work.

Copilots are also designed to revolutionize the way we interact with technology, acting as intuitive interfaces between users and their computers or the connected world. By leveraging voice commands and natural language processing, Copilots make technology more accessible and user-friendly, bridging the gap between human intentions and digital execution. This seamless interaction enhances user experience and efficiency, allowing individuals to harness the full potential of AI without needing to master complex systems or interfaces. The vision of the Copilot is to have AI and humans share the work equally. More importantly, humans start and end the task. On the other hand, the vision of the agentic application is to have AI do most of the work, only using human inputs at the start to guide it.

For example, Microsoft 365 Copilot can help write a professional email in Microsoft Outlook from a short prompt given by its user. While it will redact it autonomously, impersonating its user, it does not have the ability to send the email. Even if the user reads the generated text and approves it, it cannot ask the Copilot to send it. The final and concrete action always must be done by a human, to ensure that a human has looked and verified the AI generated content.

Although the restricted autonomy granted to Microsoft Copilots may seem burdensome for users who must still oversee and act on the assistant’s suggestions, organizations using Microsoft Copilot report otherwise: 70% of users felt more productive, 83% of cybersecurity teams noted reduced efforts required for their tasks, and sales professionals saved an average of 90 minutes per day.

However, the fact that Copilots are not autonomous agentic applications does not mean they do not use agents to extend their capabilities. They simply do so in non-autonomous ways that allow the user to stay in the loop. Nevertheless, as advancements in technology, control systems, and user training continue, they are becoming increasingly autonomous. At Ignite, for example, we announced the possibility of Copilot agents being called from triggers [10].

# Module 2: An introduction to different agentic architectures

“Agents are not only going to change how everyone interacts with computers. They’re also going to upend the software industry, bringing about the biggest revolution in computing since we went from typing commands to tapping on icons.”

Bill Gates, Microsoft co-founder

## Main AI agents' structural challenges

As previously discussed, profiling, perception, memory, and planning are the fundamental components of agentic applications. While profiling and perception ensure smart, context-aware assistance, the primary challenges lie in managing memory effectively and developing sophisticated planning algorithms.

### Memory

The memory challenge can be broken down into three parts: the data structure used by the agent, the format in which the data is stored and finally the operations performed.

#### Memory structure

The structure of the data used by these agents is divided into short-term memory and long-term memory. The short-term memory, often referred to as the working memory, is utilized within the context window of the LLM, allowing the agent to keep track of immediate tasks and conversations.

On the other hand, long-term memory is responsible for storing external knowledge and the historical context of interactions, enabling the agent to maintain continuity and reference past exchanges effectively.

#### Memory format

There are three primary solutions for managing the format of memory storage in agentic architectures. The first is using natural language, which allows for a comprehensible format rich in semantic information, making it highly useful for future reference and interaction. Another solution is storing data in the form of embedded vectors, which offers better memory management by efficiently capturing the contextual essence of information.

Lastly, database storage is essential for scenarios that require precise recording and processing of information, as it provides a structured and organized storage format that facilitates accurate retrieval and manipulation of data. Agents typically utilize various data sources and must analyze data flows that contain a mixture of these different formats to ensure comprehensive and accurate processing.

The Microsoft ecosystem, with its extensive suite of integrated tools, efficiently supports and manages these different types of storage, enhancing overall functionality and performance. To go further, see [11].

#### Memory operation

There are two primary uses for data: retrieval and writing. The first involves dealing with an ever-increasing flow of data, which the agent must filter to extract the main information for further use. This information is selected according to recency, relevancy, and importance criteria, ensuring that the agent retains only the most pertinent data for its tasks. When it comes to writing, two significant challenges emerge: duplication and memory overflow.

To address duplication, it's essential to compare different datasets, summarize them, and store the refined result. Handling memory overflow requires a balanced approach, combining user selection, automatic deletion based on specific criteria, and file compression. These challenges can potentially lead to investment in new storage solutions and to performance losses.

### Static vs. dynamic planning

After looking at the various challenges linked to memory in AI agents, we're going to tackle the subject of planning. This can be done statically or dynamically.

#### Static planning

Static planning is a method where the plan is formulated before the execution of the agentic system and remains unchanged throughout the execution. This approach can be realized using either a single path or multiple paths.

A single path involves decomposing tasks sequentially. Each task is broken down one after the other in a linear fashion. This straightforward approach ensures a clear and manageable sequence of actions, allowing the agent to follow a predetermined route without deviation.

On the other hand, the multiple path approach employs tree architecture. Here, the system evaluates different possible actions after each step, creating a branching structure of options.

At the end of the planning phase, the agent selects a path from all the possibilities within the tree. This chosen path is the one that will be executed, ensuring that multiple contingencies and possibilities are considered before committing to a final course of action.

Static planning provides a robust and predictable framework for agent operation, ensuring that the system adheres to a well-defined plan without the need for real-time adjustments.

#### Dynamic planning

In the context of more complex tasks with many steps, it may be necessary to adapt the plan based on the results, the situation, etc. This is why dynamic planning allows the modification of the planning during the execution of the agent application. To question the pre-established one, the application can rely on three types of feedback:

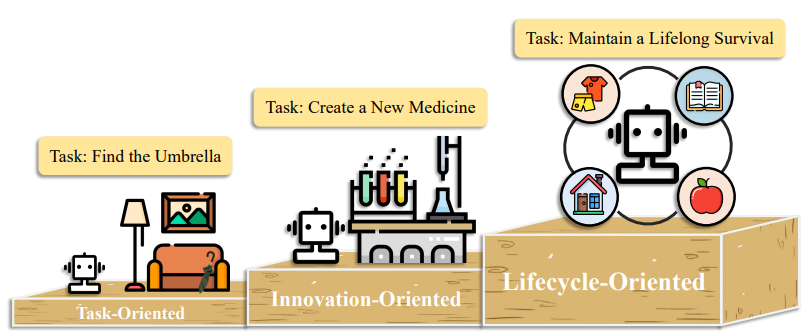
1. Firstly, the application can generate feedback based on the outputs obtained and modify the planning if necessary.
2. The application also interacts with external elements that can produce feedback, allowing the planner to take these recommendations into account to adapt the planning.
3. Finally, the application can rely on humans by asking them to choose between different proposed steps.

## Single vs. multi-agents’ architecture

Single agents and multi-agent systems serve different purposes with single agents focusing on specific tasks while multi-agent systems involve cooperation or competition, each having distinct advantages and disadvantages in terms of complexity, flexibility, and efficiency.

### Single agent architecture

A single agent is an AI model, such as an early version of ChatGPT, designed to perform specific tasks independently with limited autonomy, suitable for simple, well-known tasks that do not require feedback from other agents. However, we can distinguish three different categories of agent according to Zhiheng Xi et al [7]: task oriented, innovation oriented and lifecycle oriented.



Single agent categories [7]

#### Task-oriented agent

A task-oriented agent is an AI designed to accomplish specific tasks or achieve particular goals by engaging with its environment and making decisions based on predefined objectives. This type of agent follows high-level instructions from users, undertaking tasks such as breaking down goals into smaller sub-goals, planning the sequence of actions needed, and interactively exploring the environment until the final objective is reached.

The travel agent used in the introduction is an example of a task-oriented agent. This task should have been done with Copilots using web search tools. When Copilot answers a question using the web, it first analyzes past interactions to understand the context. Then, it uses advanced AI search tools to gather relevant information from the web. This information is synthesized, filtering out irrelevant data and compiling the most pertinent details. Finally, the answer is personalized based on the user’s query and context, ensuring clarity and relevance. This process allows Copilot to provide accurate, detailed, and user-friendly responses efficiently [12].

#### Innovation-oriented agent

AI agents can also contribute to innovation, particularly in content generation. They have good capabilities for repetitive tasks, but science is sometimes complicated to model. In addition, understanding, abstraction and innovation capabilities are not currently sufficient in the context of single agents.

One of the main areas of application is code generation. We saw in [Module 1](#_Using_agents_to) that AI agents can generate better content than a simple LLM. One of the frameworks used was Reflexion [13]. The principle of Reflexion is to have an LLM generate code and then generate feedback in verbal language so that it regenerates better quality code. The advantage of this method is that it is lightweight, easy to understand and provides nuanced feedback.

#### Lifecycle-oriented agent

The last type of single agent is known as a milestone to the implementation of AGI (Artificial General Intelligence). In this case, the objective of agents is to maintain a long-term life cycle in an open, unknown world by exploring and developing new skills. It needs low-level control and high-level planning. We can imagine an agent which can explore the computer environment and learn how to use it like us and realize complex tasks with it.

An example of a lifecycle agent is the embodied agent Voyager [14] deployed in the virtual world of Minecraft. The agent has had to learn to plan his tasks according to his abilities in his environment, and to explore on his own. Among the main qualities that it has managed to develop, we can highlight the automatic exploration of its environment, the creation of a library of already acquired abilities that it can reuse, the development of an iterative prompt process to correct itself and constant learning despite being confronted with more complex tasks and environments.

### Multi-agents architecture

Multi-agent systems consist of multiple interacting agents that can be either cooperative or competitive. These agents work together to solve complex problems that are difficult or impossible for a single agent to handle alone. Each agent operates autonomously, has its own goals, and can communicate and coordinate with other agents to achieve common objectives [15] [7].



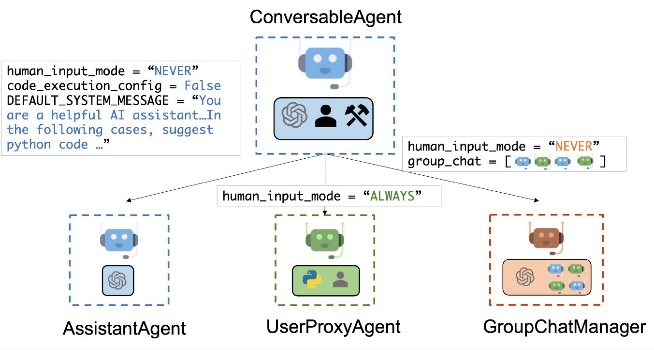
Multi agent categories [7]

#### Cooperative architecture

Multi-agents can work in a cooperative way. In this situation, agents are AI entities that work together to achieve shared objectives. They assess each other’s needs and capabilities, actively seeking collaborative actions and information sharing [7].

##### Ordered architecture

The ordered architecture of agents is based on structured communication where each agent intervenes successively, building on the previous contributions. This means that each agent adds its specific knowledge to that already provided by the previous agents, thus creating a coherent and cumulative chain of contributions. This architecture can be vertical or horizontal.

Much like vertical human hierarchy, vertical agents' architecture involves a leading agent guiding sub-agents with specific tasks or roles, who report only to the leader, resulting in separate feedback and integration of their outputs. This architecture implies limited collaboration between agents. We use this architecture for tasks which reproduce human organization. The example of the travel agentic application can be implemented with this architecture with the [AutoGen](https://github.com/microsoft/autogen/blob/main/python/packages/autogen-studio/README.md) open-source framework. We have an orchestrator which supervises the operations of the other agents and the needs of the user.

On the other hand, horizontal agents’ architecture enables communication between all agents, sharing knowledge and results with each other. This architecture is particularly suitable for situations such as scientific research and software development, for example. We talked about the LDB architecture previously in the [Module 1](#_Using_agents_to) to improve the code generation. This architecture is a single agent one which separates code into blocks to analyze them and debug it. By adding a coding agent and defining the communication between both agents, as made in the paper, we get a horizontal multi agents' system [5] [16].

##### Disordered architecture

The disordered horizontal architecture of agents is a structure where multiple agents communicate simultaneously in an unstructured manner, sharing their opinions and progress based on their tools and roles. This approach is particularly suited for tasks that require the generation of new and creative ideas, as it allows for free and dynamic interaction among agents, like an unstructured debate [7].

Unfortunately, this structure hasn't been widely adopted because some research proves that even for creative content generation, the disordered architecture can lead to inefficiencies and conflicts among agents, making it difficult to achieve cohesive and high-quality results. Furthermore, the lack of structured communication often results in duplicated efforts and misunderstandings, as agents may not have a clear understanding of each other’s progress and contributions. This can lead to fragmented outputs that require significant post-processing to integrate into a coherent whole, ultimately reducing the overall efficiency and effectiveness of the multi-agent system [17].

#### Adversarial architecture

Conversely, multi-agent systems can operate in an adversarial manner. It’s a system where agents engage in competitive interactions to enhance their strategies and behaviors. In such environments, agents dynamically adjust their actions in response to others, aiming to select the most advantageous or rational moves. This argumentation and debate among agents lead to more robust and efficient outcomes by encouraging agents to abandon rigid beliefs and engage in thoughtful reflection, ultimately improving the quality of their responses [7].

This pioneering architecture was developed to address certain limitations of LLMs, particularly their tendency to be overly confident in their outputs, even when incorrect. To mitigate this issue, the Multi-Agent Debate (MAD) architecture was introduced. In this one, multiple agents engage in a “tit for tat” debate (strategy where agents take turns presenting arguments or counterarguments in response to each other), and a judge oversees the process to determine the final solution. This setup encourages divergent thinking and improves LLM performance on complex tasks.

However, this architecture, which imitates human debate, has the same flaws as its human counterpart. In fact, the system can have difficulty processing long debates, which can lead to over-computation. It can also reach a totally erroneous consensus [18].

## Microsoft Copilots’ architecture

The Microsoft Copilots’ “universe” as previously introduced is designed to accompany various user experiences (UX), and thus to harness a multitude of capabilities and features, from web searching to producing content within and for the considered Microsoft solution or offering, e.g., Microsoft 365, and even creating its own copilot. These tasks are typically executed in a multi-modal manner, employing both voice and natural language, which enhances the user experience and interaction. However, as mentioned in [Module 1](#_How_do_agents), the autonomy of Copilot is limited compared to the broader concept of agents. This limitation ensures that critical oversight and human intervention remain integral to its operation, aligning with the Microsoft AI principles and approach. [19]

Nonetheless, we can position the copilot architecture in the reference frame of agent architectures presented above. Copilot Agents are single agents that communicate with humans and external resources. They are positioned halfway between task-oriented and innovation-oriented single agents. These agents can also be combined to form multi-agent systems.

To support this, Microsoft has introduced various tools and platforms to develop and orchestrate these Copilots, which we will detail in the following section.

# Module 3: Microsoft Copilots’ ecosystem

“Agents are going to be the new graphical user interface.”

*Steven Bathiche, Microsoft Technical Fellow*

The Copilot ecosystem comprises a range of tools for creating these agents, with options for different user requirements, including no-code, low-code or pro-code tools. These allow Copilots to be deployed in Microsoft 365 applications or even in custom-built applications.

Une image contenant texte, capture d’écran, Police, diagramme

Description générée automatiquement

Build agentic applications with Microsoft

As shown in the above diagram, there are two options for building Copilots: i) via Microsoft 365 Copilot (M365 Copilot) or ii) via Azure AI Foundry Services.

## Build your own Copilot agent starting with Microsoft 365 Copilot

Let’s start by defining what is [Microsoft 365 Copilot](https://www.microsoft.com/microsoft-365/copilot). It is an AI-powered productivity tool that helps users complete tasks more efficiently and improve their overall work experience. It integrates large language models (LLMs) with data from Microsoft Graph and Microsoft 365 apps like Word, Excel, PowerPoint, Outlook, and Teams. This combination allows Copilot to assist with tasks such as drafting, summarizing, and answering questions in the context of the user's work. It’s basically your agents in your Microsoft 365 apps [20].

If the default Copilots do not meet your requirements, you can create your own Copilots using [Microsoft Copilot Studio](https://www.microsoft.com/en-us/microsoft-copilot/microsoft-copilot-studio). It’s a comprehensive development SaaS platform that empowers users to customize and deploy their own Copilot solutions.

With Copilot Studio, developers can customize a wide range of features to create tailored AI solutions specific to their needs. This includes configuring the interactions between the Copilot and users, designing bespoke workflows, integrating domain-specific knowledge bases, and setting custom response behaviors. This is a low-code and no-code platform. While most actions are no-code, users have the option to add small portions of code to meet their requirements, such as integrating specific functions [21].

## Create agents with Azure AI Foundry

Although the Copilot Studio allows for the creation of highly personalised Copilot agents, it is intended for use in creating Copilots only. As previously mentioned in this guide, Copilot agents are not autonomous agents.

Additionally, it is not feasible to customise the orchestration between different Copilots. This limitation has led to the development of pro-code tools around [Azure AI Foundry](https://azure.microsoft.com/products/ai-foundry) [22].

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

Tools to build agents

Azure AI Foundry is a unified AI app platform that includes both the Azure AI Foundry portal (formerly Azure AI Studio), and code-first unified SDK experiences (Azure AI Foundry SDK) with pre-built app templates for developers. Azure AI Foundry unifies the AI toolchain, integrating a comprehensive suite of Azure AI capabilities, models, with familiar tools for developers like GitHub and Visual Studio to design, customize, and manage AI applications and agents. It simplifies the development and management of AI solutions for developers, engineers, and IT professionals, enabling efficient customization and innovation.

The Azure AI Foundry SDK tools provide access to a range of libraries and tools for creating these agents, including Azure AI Agent Service, [Semantic Kernel](https://github.com/microsoft/semantic-kernel) and [Autogen](https://github.com/microsoft/autogen).

Azure AI Agent Service [23] is a new set of capabilities enabling developers to build, deploy and scale enterprise-grade AI agents that automate complex business processes by completing specific tasks while keeping humans at the center. It ensures data privacy and compliance, keeping all sensitive information securely contained. With a rich ecosystem of models from the Azure AI Foundry model catalog, knowledge sources such as Microsoft Bing, [Microsoft SharePoint](https://www.microsoft.com/en-us/microsoft-365/sharepoint/collaboration?msockid=1be0b4b902a56bab3e28a03e03096af0), [Microsoft Fabric](https://www.microsoft.com/en-us/microsoft-fabric), and [Azure AI Search](https://azure.microsoft.com/en-us/products/ai-services/ai-search/), and more than 1, 400 of actions connectors with Azure Logic Apps, the service simplifies the process of building AI agents that can increase productivity, enhance customer experience, and improve decision-making.

When building multi-agent solutions, organizations should start with Azure AI Agent Service for reliable, scalable, and secure agents. They can then leverage Microsoft's orchestration libraries, AutoGen and Semantic Kernel, to coordinate agent collaboration.

AutoGen [24], developed by Microsoft Research in Python, evolves to optimize agent and human interactions, while Semantic Kernel [25], Microsoft’s enterprise AI SDK for Python, .NET, and Java, incorporates production-ready features from AutoGen for robust, stable support. Together with Azure AI Agent Service, these frameworks enable agents to interact dynamically. These frameworks can be used to orchestrate Azure AI Agent Service agents, as well as agents from other frameworks and even agents written in other languages. All that's needed is for communication between them to be standardised, as it is for humans, so that they can find a way of understanding each other.

Finally, you can integrate your hardware solutions into your own applications or into Microsoft 365 using the [Microsoft 365 Agents SDK](https://github.com/microsoft/agents) [26].

This SDK simplifies building enterprise-grade agents. It includes features like channels, conversation management, and integration capabilities. It acts as the glue and scaffolding for developers to prototype and deploy line-of-business agents, allowing them to integrate various AI services and models. It is intended to be a comprehensive toolkit for integrating agents that can operate across multiple platforms like Teams, Microsoft 365, Slack, Messenger, and more.

Une image contenant texte, capture d’écran, logiciel, multimédia

Description générée automatiquement

Breakdown of the different agent services across the Microsoft 365 Agents SDK

The Microsoft 365 Agents SDK is focused on the integration and deployment of AI agents into channels like Microsoft 365 Copilot and the third-party messaging platforms, while the Azure AI Agent Service provides the necessary infrastructure and services to develop and manage these agents effectively.

When combined with Copilot Studio and Microsoft 365 Agents SDK, Azure AI Agent Service unlocks even greater value. Copilot Studio allows you to rapidly prototype and deploy agents with an intuitive natural language interface, while Azure AI Agent Service provides advanced customization and scaling for pro-code AI development. Together, these tools enable businesses to:

* Start fast: Use Copilot Studio to quickly prototype and deploy agents without the need for custom infrastructure.
* Scale seamlessly: Incorporate enterprise-grade models, vectorized indexes, and multi-agent orchestration tools like AutoGen and Semantic Kernel from Azure AI Foundry to extend agent capabilities.
* Deliver across channels: Leverage the integration between Azure AI Agent Service and the Microsoft 365 Agents SDK to deploy agents to multiple channels like Microsoft Teams, Microsoft 365 Copilot, and Outlook, ensuring wide accessibility for end users.

# Module 4: Building multi-agents systems with AutoGen

“AI agents will become an integral part of our daily lives, helping us with everything from scheduling appointments to managing our finances. They will make our lives more convenient and efficient.”

*Andrew Ng, Co-founder of Google Brain and Coursera*

Following a review of the theoretical underpinnings of agents and the various tools for implementing them at Microsoft, we will proceed with the implementation of our inaugural agentic application. For this task, we are going to use the AutoGen because it supports multi-agent conversation patterns, enabling the creation of sophisticated LLM applications with minimal effort [27].

To facilitate the creation of these applications, AutoGen has developed its own studio: [AutoGen Studio](https://github.com/microsoft/autogen/blob/main/python/packages/autogen-studio/README.md), which provides a user-friendly low-code interface for rapidly prototyping and managing multi-agent workflows. However, this studio remains limited in terms of customization, as it can only be used to create systems containing a maximum of 2 agents [28].

As of this writing, the [latest v0.4](https://www.microsoft.com/en-us/research/blog/autogen-v0-4-reimagining-the-foundation-of-agentic-ai-for-scale-extensibility-and-robustness/) available release introduces the Starfleet feature, which enhances the framework’s capabilities by optimizing agent collaboration and improving performance across diverse applications. This overhaul of the framework involves using an actors’ system, which has already been used for many years in projects such as the Orléans project [29]. The principle is not to create and destroy agents, but rather to have a repertoire of agents and to activate / deactivate them by varying their state. [24]

As part of our study, we're going to use AutoGen version 0.2 to create our agentic travel application.

### Requirements

Reminder: to be able to run the notebook associated with this guide, you must first have followed the following notes:

* Cloning the Repo
* Prerequisites Guide
* 1 – Let’s Start with Azure
* 2 – LLM model creation in Azure and using it in Python
* 3 – Intalling Microsoft Visual C++

We also recommande you to install the Markdown in VS Code to use the preview function for Markdown files (.md)

## Build a simple agentic travel application

#### Application overview

As a first step, we can create a simple travel agent application, made up of several agents who will have to deal with the problems at hand. Here, we're going to use Conversational Agents without any external tools (we'll add more in the advanced version of the application that we'll create in the next section).

The agents will communicate in the form of a peer chain. In other words, they will communicate two by two and pass on their discussion to the next group, which will try to go further or take other aspects into account.

Une image contenant diagramme, texte, cercle

Description générée automatiquementInteraction flow graph

Our application consists of five agents as follows:

* An agent who selects the destination.
* An agent for transport.
* An agent for accommodation.
* An agent for activities.
* A reporting agent who discusses with each agent.

#### Application implementation

The first step is to call the LLM you want to use. Here, we're using a GPT-4o model with Azure. To see how to create the model, please refer to the note 2 - LLM creation in Azure and using it in Python.

llm\_config\_dict = {"config\_list": [{"model": "gpt-4o",

                                    "api\_type": "azure",

                                    "api\_key": os.environ["AZURE\_OPENAI\_API\_KEY"],

                                    "api\_version": "2024-06-01",

                                    "base\_url": os.environ["AZURE\_OPENAI\_API\_BASE"]}],

                    "cache\_seed": 42,

                    "temperature": 0}

Next, we create all our agents one by one, specifying their name and role.

report\_agent = ConversableAgent(

    name="Report\_Agent",

    system\_message="You are responsible for creating a report by extracting insights from the chat history.",

    llm\_config=llm\_config\_dict,

    human\_input\_mode="NEVER",

)

# Destination Agent

destination\_agent = ConversableAgent(

    name="Destination\_Agent",

    system\_message="Help choose the destination based on your expertise."

                    "Provide at most 3 options with a brief description of each.",

    llm\_config=llm\_config\_dict,

    human\_input\_mode="NEVER",

)

# Transport Agent

transport\_agent = ConversableAgent(

    name="Transport\_Agent",

    system\_message="Find the best transport options to get to the chosen destination regarding time, cost, and environmental impact."

                    "In the end, select the best option and provide a brief description.",

    llm\_config=llm\_config\_dict,

    human\_input\_mode="NEVER",

)

# Accommodation Agent

accommodation\_agent = ConversableAgent(

    name="Accommodation\_Agent",

    system\_message="Recommend accommodation options that fit the traveler's needs and budget (hotels, Airbnb, hostels)",

    llm\_config=llm\_config\_dict,

    human\_input\_mode="NEVER",

)

# Activities Agent

activities\_agent = ConversableAgent(

    name="Activities\_Agent",

    system\_message="Suggest activities and attractions to do at the destination, based on the traveler’s interests.",

    llm\_config=llm\_config\_dict,

    human\_input\_mode="NEVER",

)

Finally, we launch a conversation by specifying a question for each discussion sequence. In this example, our problem is explained in the first prompt, the others being standard prompts that can be used for many cases in our application.

chat\_results = report\_agent.initiate\_chats(

    [

        {

            "recipient": destination\_agent,

            "message": "What are the best places for a sunny vacation in December, knowing that I'm in Paris?",

            "max\_turns": 1,

            "summary\_method": "last\_msg",

        },

        {

            "recipient": transport\_agent,

            "message": "What are the best transport options to get to these destinations whitout a big footprint?",

            "max\_turns": 1,

            "summary\_method": "last\_msg",

        },

        {

            "recipient": accommodation\_agent,

            "message": "What are the top-rated hotels in the selected destination?",

            "max\_turns": 1,

            "summary\_method": "last\_msg",

        },

        {

            "recipient": activities\_agent,

            "message": "What are the best activities in the selected destination?",

            "max\_turns": 1,

            "summary\_method": "last\_msg",

        }

    ]

)

We can access the history of exchanges between our agents, but this history can be long and difficult to read, so we use the following code to summarise the exchanges:

print("First Chat Summary: ", chat\_results[0].summary)

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

print("Second Chat Summary: ", chat\_results[1].summary)

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

print("Third Chat Summary: ", chat\_results[2].summary)

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

print("Fourth Chat Summary: ", chat\_results[3].summary)

The result of this experiment is store in the file *output\_simple\_app.md*. It is a well-structured report that summarises the stages of reasoning. Finally, our application responded well to our problem by adding details that were not requested but which are nevertheless essential if we want to continue programming our trip.

## Build an advanced agentic travel application

With this previous version, we saw how AI agents could be coordinated to meet our needs. However, we can use slightly more sophisticated architectures and empower them with tools. This version of the application is a bit more complex, but its development remains accessible to everyone.

#### Application overview

In this application, we are going to use a structure made up of agents, a manager and tools. Here is a diagram illustrating this structure:

Une image contenant diagramme, cercle, ligne, capture d’écran

Description générée automatiquement

Schema of our “advanced” agentic travel application

This structure is made up of:

* User which results in the generation of the original prompt and potential exchanges with the UserProxy agent if the application needs the user's itervention.
* UserProxyAgent which is a conversational agent that handles interaction between the application and the user.
* Planner Agent which plans the tasks to be carried out according to the problem to be solved. It breaks down the problem into sub-tasks and assigns them to an agent.
* Personnal Advisor agent with access to user files. It responds to a problem by using a document search agent (with RAG technology) to synthesise information based on the request.
* WebSearchAgent searches on the Internet for additional information and summarises it according to the problem.
* Report Creator which summarises the discussion between the various agents into a markdown file that addresses the initial problem. It also takes care of ending the conversation.

This architecture also involves the use of three tools:

1. A tool for finding information in documents using RAG technology. In our example, these documents are four records of past trips detailing the trips and the reviews the user has had.
2. An internet search tool to look for external information.
3. Aa report creation tool to write a summary of the discussion and the response to the user's query

This architecture shows us how our agents can communicate with each other. Now in our use case, we have a series of standard exchanges as follows:

Une image contenant texte, diagramme, cercle, capture d’écran

Description générée automatiquement*Agents Communication*

* Step 1 & 2: The User sends a prompt to the UserProxyAgent, which forwards it to the GroupChat Manger.
* Step 3 & 4: The GroupChat Manger will then call the most appropriate agent according to the request.
* Step 5 & 6: This agent will respond to the request, create new ones, potentially use tools and transmit all the information to the GroupChat Manager.

The GroupChat Manager then selects the most relevant agent and sends it the information, and so on until the task is completed and the process is stopped.

#### Application implementation

Note This section contains the main steps and lines of code from the associated notebook. We recommend that you access the notebook for a complete understanding of the tutorial.

##### Let’s define our agents

The first step is to define our agents by giving them a name, a description, whether or not they can end the discussion, and the associated LLM. For ease of reading, we have only reported the implementation of three agents.

user\_proxy = UserProxyAgent(

    name="User\_proxy",

    system\_message="""A human admin. """, #Interact with the planner to discuss the plan. Plan execution needs to be approved by this admin. Execute the plan by making sure each agent does their part and help in the report creation.

    code\_execution\_config={

        "last\_n\_messages": 2,

        "work\_dir": "groupchat",

        "use\_docker": False,

    },

    human\_input\_mode="TERMINATE",

    is\_termination\_msg=terminate\_conversation,

    max\_consecutive\_auto\_reply=3

)

personnal\_advisor = AssistantAgent(

    name="PersonalAdvisor",

    system\_message="""

                    You're a personal travel adviser who has access to files on my previous trips.

                    You are responsible for providing information about the previous trips to help in planning the new trip.

                    """,

    llm\_config=llm\_config\_dict,

    description="Personal Advisor, expert in travel planning and  have access to files on previous trips."

)

rag\_agent = RetrieveUserProxyAgent(

    name="RAG\_Agent",

    human\_input\_mode="NEVER",

    retrieve\_config={

        "task": "qa",

        "docs\_path": DOC\_PATH,

        "chunk\_token\_size": 1000,

        "model": llm\_config\_dict["config\_list"][0]["model"],

        "client": chromadb.PersistentClient(path="/tmp/chromadb"),

        "get\_or\_create": True,

    },

    code\_execution\_config={"use\_docker": False},

    description="Assistant who has extra content retrieval power for answering questions."

)

##### Empower them with tools

In this section we will create and empower our agents with tools such as WebSearch, Report Creation and Use of *Documents* (with RAG technology).

# ---------------------------------------- CREATION of functions -----------------------------------

def save\_report(report: str) -> str:

    try:

        file\_name = "report\_"+str(datetime.datetime.now().strftime("%Y\_%m\_%d\_%H\_%M\_%S"))+ ".md"

        with open(file\_name, "w") as f:

            f.write(report)

        return "Report saved successfully."

    except Exception as e:

        print(e)

        return "Failed to save the report."

def browse\_web(query: str) -> str:

    try:

        with DDGS() as ddgs:

            results = [r for r in ddgs.text(query, max\_results=1)]

            return results if results else "Not Found"

    except Exception as e:

        print(e)

def retrieve\_content(

        message: Annotated[

            str,

            "Refined message which keeps the original meaning and can be used to retrieve content for question answering.",

        ],

        n\_results: Annotated[int, "number of results"] = 3,

    ) -> str:

    try:

        rag\_agent.n\_results = n\_results  # Set the number of results to be retrieved.

        \_context = {"problem": message, "n\_results": n\_results}

        ret\_msg = rag\_agent.message\_generator(rag\_agent, None, \_context)

        return ret\_msg or message

    except Exception as e:

        print(e)

        return "Failed to retrieve content."

# ---------------------------------------- TRANSLATION into Tools ----------------------------------

def web\_search\_tool (query: Annotated [str, 'Query string containing information that you want to search using the internet']) -> str:

    result=browse\_web(query)

    return result

def save\_report\_tool(report: Annotated[str, 'Report in markdown format']) -> str:

    result=save\_report(report)

    return result

def retrieve\_content\_tool(message: Annotated[str, 'Refined message which keeps the original meaning and can be used to retrieve content for question answering.'],

                     n\_results: Annotated[int, 'number of results'] = 3) -> str:

    result=retrieve\_content(message, n\_results)

    return result

# ---------------------------------------- Empower Agents ----------------------------------------

register\_function(

    web\_search\_tool,

    caller=travelWebSearchAgent,

    executor=travelWebSearchAgent,

    description="Web Browser Tool to search the internet for information.",

)

register\_function(

    save\_report\_tool,

    caller=reportagent,

    executor=reportagent,

    description="Tool to save a report in markdown format.",

)

register\_function(

    retrieve\_content\_tool,

    caller=personnal\_advisor,

    executor=personnal\_advisor,

    description="Retrieve content for question answering.",

)

##### Create the Group and the GroupManagerAgent

We are now going to organise our agents into a group with an associated manager.

group\_chat = GroupChat(

    agents=[user\_proxy, planner, personnal\_advisor, travelWebSearchAgent, reportagent],

    messages=[],

    max\_round=20,

    send\_introductions=True,

    speaker\_selection\_method="auto"

)

# Group Chat Manager

group\_chat\_manager = GroupChatManager(

    groupchat=group\_chat,

    llm\_config=llm\_config\_dict

)

#### Application test

The associated notebook contains three test examples, the first of which is used to demonstrate that the RAG function works by requesting information that is only available in the files provided. The second concerns internet searches. Finally, the last example, the one we are going to develop, concerns the final task of planning the next holidays using information in the documents provided, using the internet and generating a markdown file that summarises the conversation and the final schedule.

Here are the lines of code used to generate this request and display the conversation between the agents:

# Initiate the Conversation

chat\_result = user\_proxy.initiate\_chat(

    group\_chat\_manager,

    message="Plan my next vacation, I want a 5 days trip in a new country, similar to my previous trips for a budget of 1000€",

)

# Print the Discussion Summary Report

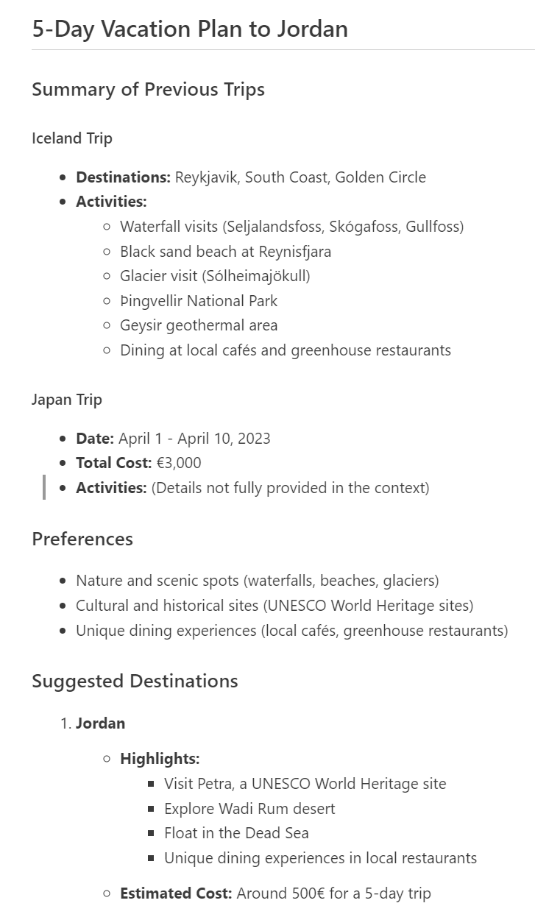
print(chat\_result.summary)

Here is the start of the discussion generated by the application. On this example, we can see the call to different agents, the use of the RAG function and the agent management by the chat manager.

Une image contenant texte, capture d’écran, document, Police

Description générée automatiquement

And here is a part of the report generated for our request:



This concludes this illustration.

# Module 5: Understand and mitigate agent’s risks and limitations

We’re looking at the future of interaction between ourselves and the machines”

*Mira Murati, ex-OpenAI CTO, on GPT-4o and its uses of agents*

Ensuring AI products act responsibly has been a key part of Microsoft’s approach to AI for years, and this new wave of agentic applications are no different: they need to be thoroughly monitored and evaluated to ensure they act as intended.

The main advantage of agents represents also their biggest limitation when it comes to ensuring they act responsibly: they are called autonomously. Yet one of the pillars of Microsoft’s principles approach to AI [19] is to keep humans in the loop when using AI and generative AI applications.

This module will look at solutions and best practices from research papers, some from Microsoft researchers, that can be implemented in any agentic application to help developers and agent hosts monitor the uses and triggering of individual agents.

## Accentuation and creation of new risks linked to AI agents

AI agents are being integrated into systems that have been under scrutiny for the past few years. AI and GenAI applications have been managed from the start to ensure high quality and safe results, limiting the risks inherent to their probabilistic nature. However, agentic applications introduce new risks due to the increased complexity, the new wave of interaction between humans and their environment, and the novel capabilities these agents possess. These factors not only create fresh challenges but also accentuate existing risks.

So, the main risks of AI agents include [30]:

* Unpredictability of multi-step user inputs: user inputs can be varied and complex, which can lead to unexpected reactions and security threats if the inputs are not well-specified.
* Complexity of internal executions: the internal processes of AI agents are often complex and difficult to monitor, which can hide security issues.
* Variability of operational environments: AI agents can operate in different environments, which can lead to inconsistent behaviors and security risks.
* Interactions with unreliable external entities: AI agents often need to interact with external tools or other agents, which can open attack surfaces if these entities are not reliable

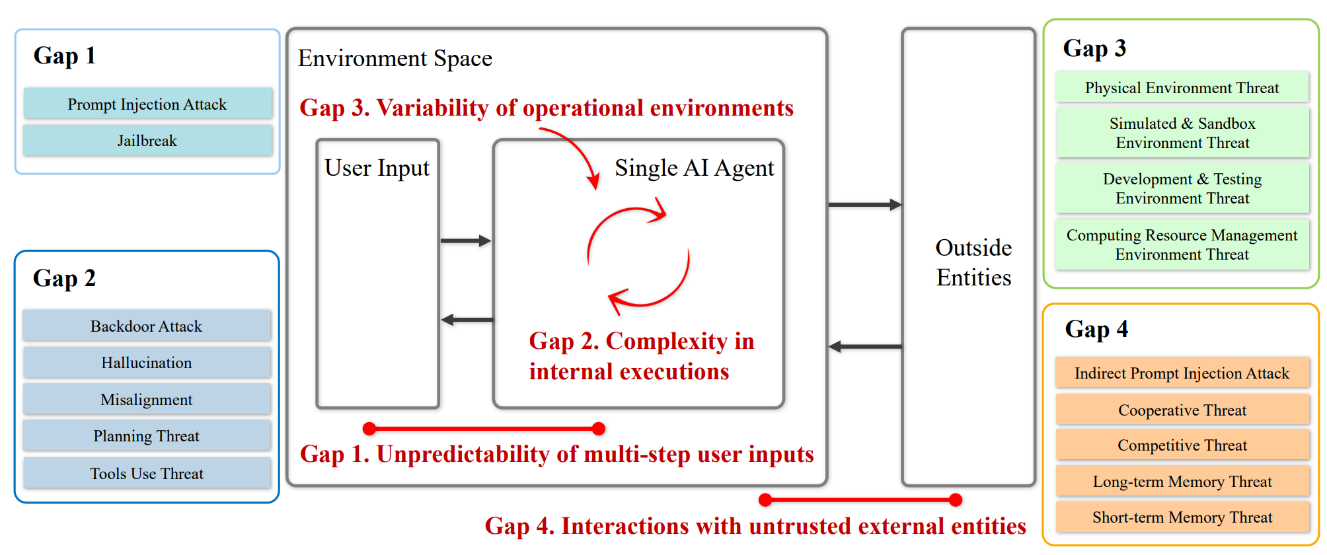


Illustration of knowledge gaps in AI agent security [30]

## Consequences of these risks

These risks can lead to potentially dangerous behaviors. Indeed, the complexity of executing agentic applications often appears as black boxes, making them difficult to understand. This inherent complexity can foster overconfidence in the content generated by the agent. This overconfidence is even worse when these agents interact with external entities. They may incorporate unverified information, leading to incorrect content.

To get a complete picture of the subject of overconfidence in agentic applications, we also need to consider the flaws that already exist in LLMs. Indeed, LLMs can generate unfounded content, known as fabrications or hallucinations. These errors occur because the model predicts text based on patterns rather than verified facts. In addition, depending on how the LLM has been trained and parameterized, the model can propagate bias, leading to the generation of incorrect content, perpetuating harmful stereotypes or inaccuracies [31].

The other behavior comes from the global architecture of agentic applications. From the unpredictability of the inputs, through the complexity of the executions, to the use of external entities, it creates a lot of vulnerabilities which can be used for misuses. From a security perspective, these vulnerabilities can be used to steal data by prompt injection for example. Additionally, AI agents can generate harmful content, including misinformation, hate speech, or dangerous instructions. Beyond this, the misuse of AI can erode trust in technology, hindering its adoption.

Some of these risks are already known in AI and GenAI applications. As frameworks and risks management measures have been put in place for those, we can take inspiration from them as they have been field-tested in real world applications for a few years and apply them with slight changes to adapt them for autonomous agentic applications.

## Measures implemented to address these risks

Some measures and frameworks have already been developed to address the issue of overconfidence. This one is a top priority because it concerns the principles of transparency and accountability, two pillars of RAI. To prevent overconfidence from user, one critical element of these frameworks is the implementation of agent identifiers [31].

### Agent identifiers

The objective of agent identifiers is to make clear that some actions or content are generated by agents to make users aware of the possible inaccuracies that may contain.

The first vector of information exchange between the user and the agentic application is the user interface (UI). This interface must contain a series of markers to make it clear that the content is generated by AI. To help developers to develop these markers, Microsoft has posted several rules that agentic application has to follow, available here [32].

Let's look at the copilot web interface to better understand what these different markers consist of.

We asked Copilot Web to give us information about the latest devices announced by Microsoft. Note, at the time of the test, they are the first generation of Copilot+PCs in the form of the Surface Pro 11, announced last May 2024, which is after GPT-4’s knowledge cutoff.

As you can see, despite its knowledge cutoff, Copilot autonomously calls upon its web searching agent to find the answer. A non-agentic application would not have been able to find the correct answer.

To let the users know that the information is unverified and retrieved on the web, Copilot gives users links to its sources, through hyperlinks in the answers, and the “Learn more” tab at the end. Additionally, as with every Copilot output, we remind the user that “AI-generated content may be incorrect”. This clear user interface allows for transparent use of a web searching agent.

If we'd used a voice interface, we could have had a presentation of the agent at the start of each conversation, explaining that it's an AI. So, each communication interface has its own codes.

Other AI-generated content identification vectors are also being introduced. Recently, watermarks [33] were announced, which are invisible markers on image content generated by AI but detectable by other AI to check if it has been produced or not by AI.

We could also mention the protected detection for code and text [34], primarily added for property rights issues, but which can also be seen as a guarantee of the quality of the AI output. These few recent examples demonstrate the drive to continually improve agentic applications. In the context of these agentic applications, we could imagine identification at start-up or for each call between agents to check that an agent has not been corrupted.

### Moderation and governance

To prevent misuse and mitigate risks, the moderation and governance of agentic applications are crucial. The aim is to add security functionalities to the various layers of the agentic application, from the user interface to the model itself, as well as the information processing phases in the various sub-layers.

The first step in preventing these risks is to filter the content of the user's prompt. To achieve this, Microsoft has developed a tool available with all its AI modules: [Azure Content Safety](https://azure.microsoft.com/en-us/products/ai-services/ai-content-safety). By leveraging advanced algorithms and real-time monitoring, Azure Content Safety scrutinizes user prompts and AI-generated outputs to identify and block harmful content before it can cause any damage. Different categories of danger have been identified, such as the generation of violent, hateful or sexual content, or content that could harm the user. It is also possible to create personalized categories. These new categories could address the issue of AI capability limits. For example, an AI cannot make decisions involving a high level of responsibility (medical, financial, etc.) and should be blocked by detecting it in the prompt. All these categories can have different levels of filtering, so you can manage output governance with precision. Finally, Azure Content Safety also analyses prompts for various attacks to protect the application.

As interactions with applications become multi-modal, a multi-modal component analysis layer has also been developed to reinforce security [35]. Another security enhancement in Azure is the availability of new tools for simulating and assessing the security of your AI application [36].

Finally, tools like Hidden Layer offered in Azure allow for the analysis of AI models and applications to assess their security level. Indeed, due to their structure or training, models can have vulnerabilities that are difficult to fix, so choosing a secure model is one of the first actions to take [37].

### Tracing and monitoring agentic activity

Developers and cloud hosts of AI agents, like Microsoft, have a responsibility to integrate tracing and monitoring capabilities in their agentic systems, to provide the user with a safe and transparent system despite the autonomy granted to agents during use.

A solution consists of monitoring agents in real-time, to flag any dangerous or unwanted activity, whether it be an agent called at the wrong time, or a dangerous output. Real time monitoring allows not only to flag the problematic agent’s output, but more importantly to block the application before it reaches its final output.

Keeping a human eye on autonomous agents also means keeping logs of each agent to have a trace of each call. When there is a problematic output, blocked by the real-time monitoring, logs allow developers to modify the agentic architecture after the fact, creating a custom solution based on the error that arose. Solutions could imply to change the agentic team’s structure, add or remove certain personas (see [Module 2](#_Module_2:_An) for explanations of possible architectures) [38].

## Agents and benchmarks

Benchmarking is crucial in the development and deployment of AI agents as it provides a systematic way to measure their performance against established standards. By setting benchmarks, developers can identify strengths and weaknesses in their models, ensuring that they meet specific criteria for accuracy, efficiency, and security. Moreover, providing a quality benchmark is necessary regarding the pillars or principles of transparency and responsibility for the Microsoft AI principles - these two principles are concerned by the fact of carrying out a benchmark of the application, the other principles are obviously included in the benchmark and its conclusions).

However, AI agents are complicated to evaluate because of their varied applications and complex structures. An evaluation can focus on different points such as the output of each sub-task or the result. Numerous metrics can also be used to measure the capabilities of these agents.

A recent study conducted by Kapoor et al. [5] explored the different specific challenges of benchmarking AI agents compared to LLM. The study highlighted two key features. The first concerns the way in which cost is considered in the assessment. If we take the example of [Module 1](#_Using_agents_to), where different LLMs and agents were tested for their code generation performance, we can compare these including the cost associated.



This study shows that, depending on the method used, the associated cost can vary greatly, from $0.05 for an average result with GPT-3.5 to more than $6 for the best solution. We can see from the graph that the best performing models are not necessarily the most expensive. It also leads us to reflect on the choice of a model according to its performance and the associated cost, even if it means choosing a model that performs a few percent less well but costs 2 to 3 times less than its counterpart.

So, improving agents does not mean improving performance, but rather a certain number of parameters, including cost. From a RAI point of view, cost could be seen as financial inclusivity, the aim being to enable everyone to use these technologies without distinction, especially with money. Another parameter that could be measured when benchmarking agents is the inference time because agents use different calls to different LLMs and external applications which can take few minutes or even more for specific results. By studying the performance, the cost and the latency, we get the Bermuda triangle of GenAI [39].

The second feature highlighted in the study relates to the human beings in the loop where current benchmarks do not test for this layer of safety. Since single LLMs accomplish simpler tasks and are not autonomously acting over longer ones, it made sense for these benchmarks to not consider this aspect of AI. When using applications powered by single LLMs, guaranteeing humans in the loop is a task mostly tackled by the UI designers, to make sure the model’s outputs are checked by the human users. However, as the beginning of this module explained, agentic applications will grow in autonomy until they are in a different league compared to single LLMs. Hence evaluating an agent should include measuring how well it was designed to keep a human eye on its action, both from a developer’s and user’s perspectives.

# As a conclusion

Autonomous agentic AI represents a promising advancement in developing more useful applications for everyday processes. These models are evolving at an unprecedented pace, achieving performances that seemed impossible just a few years ago. Agents have the potential to significantly broaden the utility of AI. The introduction of Copilot agents through Copilot Studio aims to bring this technology to market safely and swiftly, enabling our partners to join this AI (r)evolution.

However, autonomous agents also present challenges that need to be addressed promptly to avoid exacerbating access inequality. Academic research indicates that current LLM benchmarks are inadequate for accurately measuring performance, which could hinder equitable access to agentic autonomous capabilities:

1. Firstly, the autonomy of these agents necessitates more rigorous monitoring compared to current LLM applications.
2. Secondly, the combination of already expensive models is likely to make the most advanced agentic frameworks increasingly costly and inaccessible to some.

These challenges are not insurmountable. Collaborative efforts among industry leaders, such as Microsoft, will be crucial in ensuring this technology is developed responsibly and made accessible to all.

This concludes this guide.

# Bibliography

|  |  |
| --- | --- |
| [1] | Microsoft, "What are Microservices," 5 10 2023. [Online]. Available: https://learn.microsoft.com/devops/deliver/what-are-microservices. |
| [2] | "AI - Agents & Environments," 10 10 2024. [Online]. Available: https://www.tutorialspoint.com/artificial\_intelligence/artificial\_intelligence\_agents\_and\_environments.htm. |
| [3] | Microsoft, "AI Summit," 2024. [Online]. Available: https://microsofteur.sharepoint.com/:p:/r/teams/Oxford-AICopilot/\_layouts/15/doc2.aspx?sourcedoc=%7BA5B07C6F-3285-4914-AAA6-BBE91B9CD0F1%7D&file=AI%20Summit%20-%20agents.pptx&action=edit&mobileredirect=true&DefaultItemOpen=1&share=IQFvfLClhTIUSaqmu-kbnNDx. |
| [4] | M. Heikkilä, "What are AI Agents ?," *MIT Technology review,* pp. 18-20, 5 July 2024. |
| [5] | K. Sayash , S. Benedikt , S. Zachary, N. Nitya and N. Arvind , "AI Agents That Matter," 2 July 2024. [Online]. Available: https://arxiv.org/pdf/2407.01502. |
| [6] | S. Walker, "HumanEval Benchmark," [Online]. Available: https://klu.ai/glossary/humaneval-benchmark. [Accessed 10 10 2024]. |
| [7] | Z. Xi and e. al., "The Rise and Potential of Large Language Model," 19 September 2023. [Online]. Available: https://arxiv.org/pdf/2309.07864. |
| [8] | "Copilot for all: Introducing Microsoft 365 Copilot Chat," [Online]. Available: https://www.microsoft.com/en-us/microsoft-365/blog/2025/01/15/copilot-for-all-introducing-microsoft-365-copilot-chat/?msockid=1be0b4b902a56bab3e28a03e03096af0. |
| [9] | Microsoft, "Copilot Wave 2," [Online]. Available: https://www.microsoft.com/en-us/microsoft-365/blog/2024/09/16/microsoft-365-copilot-wave-2-pages-python-in-excel-and-agents/?msockid=3b735dd006286dfb0cc8493e074b6c31. |
| [10] | Microsoft, "Add an event trigger," 19 11 2024. [Online]. Available: https://learn.microsoft.com/en-us/microsoft-copilot-studio/authoring-trigger-event. |
| [11] | Microsoft, "AI Agents," [Online]. Available: https://learn.microsoft.com/en-us/azure/cosmos-db/ai-agents. |
| [12] | Microsoft, "Traditional search vs. Copilot," 21 November 2023. [Online]. Available: https://www.microsoft.com/en-us/bing/do-more-with-ai/traditional-search-vs-bing-chat-ai-search?form=MA13KP&msockid=3b735dd006286dfb0cc8493e074b6c31. |
| [13] | N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan and S. Yao, "Reflexion: Language Agents with," 10 Octobre 2023. [Online]. Available: https://arxiv.org/pdf/2303.11366. |
| [14] | G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. J. Fan and A. Anandkumar, "VOYAGER: An Open-Ended Embodied Agent," 19 October 2023. [Online]. Available: https://arxiv.org/pdf/2305.16291. |
| [15] | T. Masterman, S. Besen, M. Sawtell and A. Chao, "THE LANDSCAPE OF EMERGING AI AGENT ARCHITECTURES," 17 April 2024. [Online]. Available: https://arxiv.org/pdf/2404.11584. |
| [16] | Z. Li, W. Zilong and S. Jingbo, "Debug like a Human: A Large Language Model Debugger via Verifying Runtime Execution Step by Step," 6 June 2024. [Online]. Available: https://arxiv.org/pdf/2402.16906. |
| [17] | X. Guo, K. Huang, J. Liu, W. Fan, N. Vélez, Q. Wu, H. Wang, T. L. Griffiths and M. Wang, "Embodied LLM Agents Learn to Cooperate in Organized Teams," 23 May 2024. [Online]. Available: https://arxiv.org/html/2403.12482v2. |
| [18] | T. Liang, Z. He, W. Jiao, X. Wang, R. Wang, Y. Yang, Z. Tu and S. Shi, "Encouraging Divergent Thinking in Large Language Models," 9 October 2024. [Online]. Available: https://arxiv.org/pdf/2305.19118. |
| [19] | Microsoft, "What is Microsoft’s approach to AI?," [Online]. Available: https://news.microsoft.com/source/features/ai/microsoft-approach-to-ai/. |
| [20] | Microsoft, "Microsoft 365 Copilot overview," 5 12 2024. [Online]. Available: https://learn.microsoft.com/en-us/copilot/microsoft-365/microsoft-365-copilot-overview. |
| [21] | Microsoft, "Microsoft Copilot Studio," [Online]. Available: https://www.microsoft.com/en-us/microsoft-copilot/microsoft-copilot-studio?msockid=2f900015dc326591366f148fddd26411. |
| [22] | J. Hawk, "The next wave of Azure innovation: Azure AI Foundry, intelligent data, and more," 19 11 2024. [Online]. Available: https://azure.microsoft.com/en-us/blog/the-next-wave-of-azure-innovation-azure-ai-foundry-intelligent-data-and-more/. |
| [23] | Microsoft, "Introducing Azure AI Agent Service," 19 11 2024. [Online]. Available: https://techcommunity.microsoft.com/blog/azure-ai-services-blog/introducing-azure-ai-agent-service/4298357. |
| [24] | AutoGen Team, "New AutoGen Architecture Preview," 2 October 2024. [Online]. Available: https://microsoft.github.io/autogen/0.2/blog/2024/10/02/new-autogen-architecture-preview/. |
| [25] | Microsoft, "Semantic Kernel," [Online]. Available: https://learn.microsoft.com/en-us/semantic-kernel/overview/. |
| [26] | Microsoft, "Introducing the Microsoft 365 Agents SDK," 19 11 2024. [Online]. Available: https://devblogs.microsoft.com/microsoft365dev/introducing-the-microsoft-365-agents-sdk/. |
| [27] | AutoGen Studio, "AutoGen Studio 2.0," [Online]. Available: https://autogen-studio.com/autogen-studio-ui. |
| [28] | V. B. G. Dibia, J. Chen, S. Syed, A. Fourney, E. Zhu, C. Wang and S. Amershi, "Introducing AutoGen Studio: A low-code interface for building multi-agent workflows," 17 June 2024. [Online]. Available: https://www.microsoft.com/en-us/research/blog/introducing-autogen-studio-a-low-code-interface-for-building-multi-agent-workflows/?msockid=3b735dd006286dfb0cc8493e074b6c31. |
| [29] | P. Bernstein and S. Burckhardt, "Orleans – Virtual Actors," 14 October 2010. [Online]. Available: https://www.microsoft.com/en-us/research/project/orleans-virtual-actors/. |
| [30] | Z. Deng, Y. Guo, C. Han, W. Ma, J. Xiong, S. Wen and Y. Xiang, "AI Agents Under Threat: A Survey of Key Security Challenges," 6 September 2024. [Online]. Available: https://arxiv.org/pdf/2406.02630. |
| [31] | S. Passi, S. Dhanorkar and M. Vorvoreanu, "Appropriate reliance on GenAI: Reaseach Synthesis," [Online]. Available: https://www.microsoft.com/en-us/research/uploads/prod/2024/03/GenAI\_AppropriateReliance\_Published2024-3-21.pdf. |
| [32] | Microsoft, "Hax Design Library," [Online]. Available: https://www.microsoft.com/en-us/haxtoolkit/library/. |
| [33] | Microsoft, "Watermarks in preview in Azure OpenAI Service," 24 September 2024. [Online]. Available: https://techcommunity.microsoft.com/t5/ai-azure-ai-services-blog/watermarks-in-preview-in-azure-openai-service/ba-p/4253344. |
| [34] | Microsoft, "Protected Material Detection for Code now in Preview in Azure AI Content Safety," 24 September 2024. [Online]. Available: https://techcommunity.microsoft.com/t5/ai-azure-ai-services-blog/protected-material-detection-for-code-now-in-preview-in-azure-ai/ba-p/4253346. |
| [35] | Microsoft, "Multimodal Public Preview Blog," 24 September 2024. [Online]. Available: https://techcommunity.microsoft.com/t5/ai-azure-ai-services-blog/multimodal-public-preview-blog/ba-p/4253816. |
| [36] | Microsoft, "New GenAI simulation and evaluation tools in Azure AI Studio," 24 September 2024. [Online]. Available: https://techcommunity.microsoft.com/t5/ai-ai-platform-blog/new-genai-simulation-and-evaluation-tools-in-azure-ai-studio/ba-p/4253020. |
| [37] | HiddenLayer, "HiddenLayer Model Scanner," [Online]. Available: https://hiddenlayer.com/model-scanner/. |
| [38] | A. Chan, C. Ezell, M. Kaufmann, K. Wei, L. Hammond, H. Bradley, E. Bluemke, N. Rajkumar, D. Krueger, N. Kolt, L. Heim and M. Anderljung, "Visibility into AI Agents," 4 February 2024. [Online]. Available: https://arxiv.org/pdf/2401.13138v3. |
| [39] | S. Duranton, "The Bermuda Triangle Of Generative AI: Cost, Latency, And Relevance," 5 February 2024. [Online]. Available: https://www.forbes.com/sites/sylvainduranton/2024/02/05/the-bermuda-triangle-of-generative-ai-cost-latency-and-relevance/. |
| [40] | Microsoft, "Azure Content Safety," [Online]. Available: https://azure.microsoft.com/en-us/products/ai-services/ai-content-safety/?msockid=3b735dd006286dfb0cc8493e074b6c31. |
| [41] | Microsoft, "Microsoft 365 Copilot Overwiew," 16 September 2024. [Online]. Available: https://learn.microsoft.com/en-us/copilot/microsoft-365/microsoft-365-copilot-overview. |
| [42] | Microsoft, "Power Automate," [Online]. Available: https://www.microsoft.com/power-platform/products/power-automate?msockid=38af2b1ab90d67ad20d13e09b8b966d7. |

# To go beyond

To continue learning about the incredible subject of responsible AI, you can follow the other tutorials and walkthroughs available in this workshop.

Une image contenant texte, motif, point

Description générée automatiquementYou can also scan this code or visit <https://aka.ms/RAIresources> where you can access the entirety of already available tools, guidelines, and other additional resources that will help you create your next AI solution in a (more) responsible manner.

Une image contenant texte, capture d’écran, Site web, Page web

Description générée automatiquement

Une image contenant bleu, brouillard, capture d’écran, bleu vert

Description générée automatiquement