O'REILLY®

Azure OpenAI for Cloud Native Applications

Designing, Planning, and Implementing Generative AI Solutions

Early Release
RAW & UNEDITED

Adrian Gonzalez Sanchez

Azure OpenAI for Cloud Native Applications

Designing, Planning, and Implementing Generative AI Solutions

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

Adrian Gonzalez Sanchez

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Azure OpenAI for Cloud Native Applications
by Adrian Gonzalez Sanchez

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisition Editor: Megan Laddusaw
Development Editor: Melissa Potter
Production Editor: Clare Laylock
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

May 2024: First Edition

Revision History for the Early Release
2023-09-22: First Release
2023-12-08: Second Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781098154998* for release details.

978-1-098-15493-6

[TO COME]

Table of Contents

Brief Table of Contents (*Not Yet Final*)

Preface

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the preface of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *mpotter@oreilly.com*.

I cannot even hide my excitement. The 2022-23 period has been one of the most amazing moments of the modern technology era. Some people call it the "iPhone moment" of artificial intelligence, and a lot of them are now discovering the actual potential of AI. But I don't think it is "just that". I believe we are entering an exponential phase where all technological advancements move so fast that it is even difficult to keep track of them. But that is wonderful. Several years of progress and industry competition in just a few months. What could be seen as impossible (or even magic) is now a reality… and it's just getting started.

That sense of innovation and complete disruption is how I felt the first time I tried the Azure OpenAI Studio. I got early access as an AI Specialist at Microsoft. It was a very initial version, and definitively not the same kind of Studio and related features and models we have today, but it was very promising. Little did we know, but this cloud-enabled service was about to become the super star of the Generative AI era. And it was real, not a concept or a future product. It was something we could use to create our very own GPT-kind of implementations, with different models and cost-

performance tradeoffs, but with a relatively low complexity to implement and deploy them.

After a few months of testing and tracking new functionalities, OpenAI released ChatGPT. Boom. I have never seen such a viral moment related to AI technologies. Even at Microsoft, the feeling of being witnesses to something extraordinary was there, every day and evening, during countless "nerd" discussions with my colleagues. But the key moment was the announcement of the "chat" functionality in Azure OpenAI Service. The first time any company could test and deploy a ChatGPT-ish kind of instance for their own purposes. Then came Bing Chat. Boom x2. The first time we could see the combination of classic search engines with a GPT chat experience, on the same screen… and it worked! For the first time, people could find information instead of just searching, and they were doing that with plain language. Not keywords, not complex combinations of words. Just asking for information and waiting for an answer.

Months passed and we started to deploy the first proof-of-concept (PoCs) with Azure OpenAI. I'm part of the field teams so I was very close to the reality of the adopters—their understanding of what generative AI is, their envisioned use cases, their concerns, etc. But I was also part of the AI community at Microsoft, with plenty of energy and creativity to explore new approaches, discover new architectures, and learn about the most recent techniques and accelerators. Trust me, I'm not the only one to feel lucky. This was pure energy.

At some moment, and I assume this was due to my academic background as university professor, I felt like the amazing amount of information was very useful for any learner or adopter, but also a bit overwhelming for any company or individual trying to get started with Generative AI and Azure OpenAI Service. And this technology deserves to be massively adopted.

It was then that I started drafting the main sections of a technical guide for application development with Azure OpenAI Service. Initially, it was just a way to keep track of all the URLs and pieces of information I was continuously collecting. Then, I continued adding my notes, based on my own implementation experiences. Finally, I kept changing or adding content based on the recurrent questions and discussions I was getting from clients, friends, and even family!

This was a great baseline, and I knew it could become an official technical guide, or even book. I decided to talk to my O'Reilly colleagues and present the topic. These conversations only took a matter of weeks. The potential was clear, but the challenge was huge: creating high quality O'Reilly-level content, in a timely manner (as soon as possible) so all Generative AI adopters could start reading and learning.

This has been one of the most challenging but still rewarding experiences. I feel really honored to write this book. So many Microsoft folks around the world could have

done it, but I have been chosen and allowed to do so. For this reason, I took this opportunity very seriously. My main goal was to create something that would include all critical elements for any Azure OpenAI learning, keeping in mind the (very) evolving context. Showing the best features and implementation approaches, but knowing that there will be others soon. Explaining new features in preview, that may be generally available now, as you're reading this book. But that's part of the charm, and the reason why I like this book and the creative process behind it so much.

One of my favorite things (and I hope you like it too) is the combination of the typical static content of a book, with the interactivity of online repositories, references to evolving documentation… and incredible power of the guest interviews. Having such an amazing amount of talent and knowledge (note: if you haven't seen the details from the table of contents, you will discover an amazing roster of AI pros, some of the best at an international level) is an authentic luxury, for you as readers and avid learners, but also for me as an AI professional.

Now, I hope that if you have decided to start reading this book, you are ready to explore all the sections, from the core technical aspects, to any other relevant business and ethical aspects that will help you during your first generative AI projects with Azure OpenAI Service.

And of course, thanks to all involved "stakeholders" for helping me to make this happen. My wife Malini, the amazing O'Reilly team for their methodology (and their unlimited doses of patience and support, especially Melissa), my Microsoft colleagues for being a continuous source of inspiration (including my boss Agustin and his unwavering support), all technical reviewers and interviewees for their wealth of knowledge, and so many learners around the world showing their interest to learn about this amazing topic. This book is for all of you. Please enjoy it.

Truly yours,

Adrian

Introduction

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the introduction of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *mpotter@oreilly.com*.

Artificial intelligence is finally here. Don't get me wrong, it was already amongst us, but let me please explain my point, and why we can consider the current 2020s as the beginning of the new era for modern artificial intelligence.

If you are reading this book you probably already know that AI is not a new concept. It's been several decades since its first appearance (at least as a concept, during a very famous university conference in the United States), and we can now say that people from almost all walks of life are beginning to understand the potential and considerations of artificial intelligence. After several AI summers and winters, hype and deceptions, the promise of AI value for companies and individuals is finally here, and terms such as generative AI, Generative Pre-trained Transformer (GPT), or Large Language Models (LLM) are everywhere.

The arrival of AI-enabled tools such as OpenAI's ChatGPT, Midjourney, or the new Bing Chat engine, is facilitating the interaction between the people and the algorithms. Even more, the generative AI wave can be considered as a democratizing element for widestream AI adoption, due to its unique value for natural language-based communication.

And this is not only for the general public. Companies, politicians, governments, observatories, startups, etc. are all talking about generative AI, adopting the technologies, analyzing its potential, and thinking about future regulations.

That's the key difference between then and now: *Awareness*. Before, AI-enabled capabilities were kept behind the scenes (e.g., face detection and classification engines for image repositories, natural language processing and generation NLP/NLG, and speech technologies for personal home assistants). Nowadays, most people are aware that behind a GPT-kind of application, there is a "machine" behind it with AI capabilities and power algorithms.

And what comes after awareness? Plenty of things depending on the involved actor, but if we observe the typical patterns from companies and startups, mostly learning and understanding the key technology elements, and an unstoppable willingness to adopt. That brings us to *enablement*, the key element for adoption. For many years, most organizations could not leverage powerful AI-enabled technologies. That was a privilege, reserved for just a few companies and research centers—a bit of an AI aristocracy with important entry barriers for innovation and competition. This is changing though, in many ways thanks to cloud computing.

During the last two decades, public clouds such as AWS, GCP, IBM, Oracle, and Microsoft Azure have enabled companies around the world to obtain infra capabilities and, depending on the use case, access to very advanced services. During the last years, areas such as big data, AI, and security were the superstars, and the key reasons for adopters to move to the cloud and leverage aaS (as-a-Service) kind of capabilities.

In 2022 and 2023, generative AI capabilities have suddenly taken the stage. For example, Microsoft Azure incorporated the Azure OpenAI Service, a cloud-based Platform-as-a-Service (PaaS) with enterprise-grade capabilities to leverage generative language, code, and image features (more on this later). This was the first and most advanced option for AI adopters, with a key competitive advantage from OpenAI's technologies. But you - dear reader - likely already know this. And that's why you're here, looking for a way to apply generative AI by using pre-built models that can be easily customized and integrated via APIs (application programming interface) with all security and moderation advantages a company may need.

Now it is time to dig into how to use Azure OpenAI and other Microsoft services to design, build, and integrate cloud-native solutions that will solve actual business needs, with clear business cases, that will enable you to deliver high quality services to your clients and users. If you are here, you certainly understand the cloud advantage, but you need some additional pieces of knowledge and guidance. That's what this book is about. The actual AI *Democratization* (illustrated in Figure I-1) for the whole innovation ecosystem. You are (or will be) part of it. Let's use this book to get you on board.

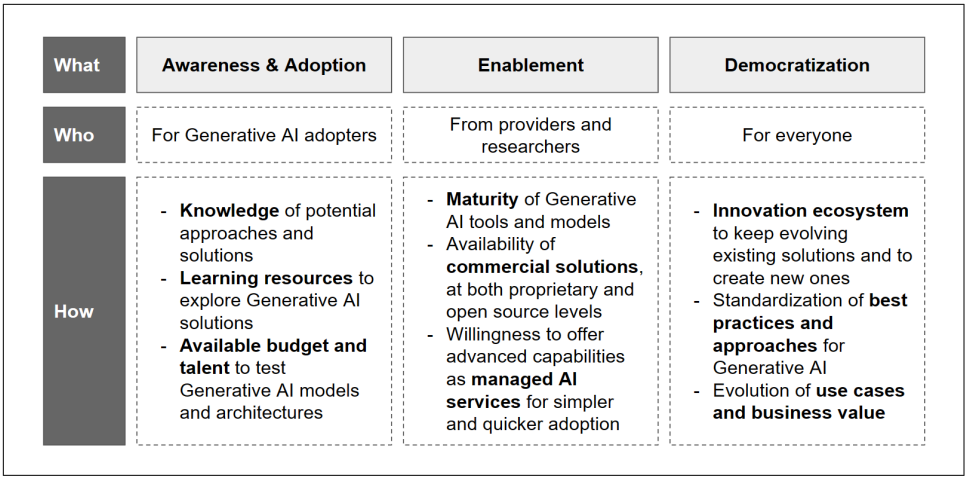

Figure I-1. The Ultimate AI Democratization

This book is an applied guide that contains the technology "building blocks" (managed services that deliver specific value and interconnect with other applications, within an end-to-end AI architecture) related to the implementation of Azure OpenAI-enabled applications. This means, you will be learning about technical settings, but also the business-related topics, such as return-on-investment (ROI) and Responsible AI.

Last but not least, the reading experience will be complemented with several expert interviews, and interactive activities. Generative AI and Azure OpenAI are highly evolving topics, so I want this book to become your living resource for your next professional projects.

CHAPTER 1

Introduction to Generative AI and the Azure OpenAI Service

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be Chapter 1 of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *mpotter@oreilly.com*.

This first Chapter covers the fundamentals of artificial intelligence (AI) as a way to contextualize the new Generative AI developments. It includes some technology-agnostic topics that will be useful for any kind of implementations, but it obviously focuses on the Azure OpenAI Service, as the key building block to enable cloud-native application development with Generative AI.

What's Artificial Intelligence

This section focuses on the historical evolution of the AI technologies and related use cases, as a way to demystify what artificial intelligence actually looks like, and to connect traditional approaches with new Generative AI techniques and capabilities.

Let's start with its origins. The term "AI" was coined during the 1950s. Concretely, Prof. McCarthy defined artificial intelligence in 1955 as "the science and engineering of making intelligent machines". It is also fair to say that Prof. Alan Turing had previously introduced the notion of thinking machines. One year later, Dartmouth College hosted the Summer Research Project on AI conference, with a reduced group of participants from the most relevant universities and companies. That conference was led by Prof. McCarthy and other renowned researchers, and it was the beginning of the AI area of research. Since then, there have been multiple cycles of hype, disappointments (often referred as AI winters), renovated expectations, and finally vast commercialization of AI-enabled solutions such as personal assistant speakers, intelligent autonomous vacuum cleaners, etc.

That said, artificial intelligence (AI) has evolved a lot during the last two decades, but the reality is that initially it was mainly and only adopted by some of the biggest companies, such as *Microsoft* (no, not necessarily for their famous Clippy!), Google, Amazon, Uber, and other technology unicorns. That first wave of adoption created a great baseline for them to offer these same capabilities as managed cloud services to other AI adopters out there, which gave them a clear competitive advantage. This started the stage of data and AI democratization we are currently experiencing, where smaller companies are developing or leveraging AI-enabled services, and those solutions are already part of our day-to-day.

But before going into the details, let's take a step back and analyze the context of what artificial intelligence is today, and what it means for companies and individuals.

Current Level of AI Adoption

The term "AI adoption" describes how organizations around the world are either implementing AI systems, or leveraging AI-enabled tools from other companies. Each company's level of adoption really depends on several factors, such as technology maturity, type of organization (big or small companies, public administration, startups, etc.), geography, etc. McKinsey says that the level of AI adoption in 2022 (from their State of The Art report) was 50% from all their responders, with an interesting increase at international level, and even more significant for developing countries. Additionally, they also estimate that generative AI could add to the global economy the equivalent of $2.6 trillion to $4.4 trillion annually.

In addition to that, Boston Consulting Group defined the level of success and AI maturity as a combination of the internal adoption, plus the knowledge of artificial intelligence within the organization, with only a 20% of the organizations being the actual pioneers in terms of AI adoption. Last but not least, Gartner predicts that by 2025, 70% of enterprises will identify the sustainable and ethical use of AI among their top concerns, and 35% of large organizations will have a chief AI officer who reports to the CEO or COO.

These figures show that even if the level of global adoption is increasing, there are still differences on how companies are using AI and how successful they are. The next section will showcase multiple examples of AI-enabled systems, at both technology and use cases levels.

The Many Technologies of AI

There are different ways to define artificial intelligence, but the reality is that there is not only one single technology under the umbrella of AI. Let's explore the main AI technologies:

Machine Learning (ML)
: A type of AI that relies on models that learn from past data, to predict future situations. Take a simple use case of classifying fruits based on their existing pictures. To describe an apple to the system, we would say it is somewhat round in shape and that its color is a varied shade of red, green or yellow. As for oranges, the explanation is similar except for the color. The algorithm then takes these attributes (based on past examples) as guidelines for it to understand what each of the fruit looks like. Upon being exposed to more and more samples, it develops a better capacity to differentiate oranges from apples and better correctly identifies them.

Deep Learning (DL)
: Deep learning can be defined as a subset of Machine Learning. The differentiating character of Deep Learning is that the algorithm uses a neural network to extract features of the input data and classify them based on patterns in order to provide output without needing the manual input of definition. The key aspect here are neural networks. The idea of neural networks comes from the fact that neural networks mimic the way the brain functions. Layered with multiple levels of algorithms designed to detect patterns, neural networks interpret data by reviewing and labeling its output. The network allows the raw output to be grouped together based on similarities also known as clustering. Clustering allows the algorithm to use its learning to recognize other data that fits the pattern. Learning without labels such as this is called unsupervised learning. Given the fact that the majority of data in the world is unlabeled, unsupervised learning has tremendous potential. If we consider our fruit example, instead of having to provide the attributes of what each fruit looks like, we have to feed many images of the fruits into the deep learning model. The images will be processed and the model will define definitions such as the shapes, sizes and colors.

Natural Language Processing (NLP)
: NLP combines computational linguistics (rule-based modeling of human language) with statistical, machine learning, and deep learning models. These kinds of models were initially only available in English (e.g., BERT from Google AI),

but the current trend is to create local versions or multi-language models to support others like Spanish, Chinese, French, etc. You can try this cool demo and see how the NLP engine detects key information from your text query.

Robotic Process Automation (RPA)
: This is a set of technologies that replicates the manual interactions of human agents with visual interfaces. For example, imagine you are working in HR and you need to do the same task every week, which could be checking some information related to the employees via an internal platform, then filling out some information, and finally sending a customized email. RPA tools are easy-to-implement and allow reducing time waste and increase internal efficiencies, so employees can focus on added value tasks and avoid monotonous work.

Operations Research (OR)
: Operational research is a very important area, often included as part of the family of AI technologies, and very related to ML and the previously mentioned reinforced approaches. University of Montreal defines Operations Research as "a field at the crossroads of computer science, applied mathematics, management, and industrial engineering. Its goal is to provide automated logic-based decision making systems, generally for control or optimization tasks such as improving efficiency or reducing costs in industry".

  OR usually relies on a set of variables and constraints which will guide some sort of simulations that can be used for different kinds of planning activities: manage limited healthcare in hospitals, optimize service schedules, plan energy use, plan public transit systems, etc.

These are the main kinds of AI technologies, but the list can change depending on the interpretation of what AI means. Regardless of the details, it is important to keep in mind these technologies as a set of capabilities to predict, interpret, optimize, etc. based on specific data inputs. Let's see now how these different AI technologies apply all sorts of use cases, which are likely to leverage one, or combine them depending on the implementation approach.

Typical AI Use Cases

Regardless of the level of technical complexity, there are many different kinds of AI implementations, and their usefulness usually depends on the specific use cases that organizations decide to implement. For example, one organization might say, "we would like to get automatic notifications when there is a specific pattern from our billing figures" and develop some basic anomaly detection model, or even a basic rule-based one, and this could be considered as an AI. Others will require more advanced developments (including Generative AI), but they will need to have a business justification behind it.

Before we explore the technical and business considerations for an adopter company, here are some examples of AI-enabled applications:

Chatbots
: You're likely very familiar with chatbots— those little friends which are embedded into websites, as well as automated phone bots, that allow companies to automate their communication and customer support. They are based on linguistic capabilities that allow them (with different levels of success) to understand the intent of what a client wants or needs, so they can provide them with an initial answer or hints to find the final answer. They also reduce the burden on support folks to answer initial requests, as chatbots can analyze, filter and dispatch cases depending on the topic.

Fraud detection
: Widely used by financial institutions, AI can help detect unusual patterns that may indicate some sort of misuse of financial assets, such as credit cards. This could be a card translation from a remote country, unusual type of purchases, repetitive attempts to get money from an ATM, etc. These AI-enabled systems make the surveillance more scalable, allowing humans to focus only on critical cases.

Voice-enabled personal assistants
: Integrated via smartphones, speakers, cars, TVs, and other kinds of devices, these personal assistants enable the interaction with human users by simulating conversation capabilities. It is widely used to reduce the accessibility barrier (i.e., it uses voice and does not require visual, writing, and reading capabilities) and allows users to free their hands while activating features such as apps, music players, etc.

Marketing personalization
: The actual rainmaker for big companies such as Google and Meta. The ability to first understand the features related to a user (their age, location, preferences, etc.) and to connect that with the business goals of companies advertising their products and services, is the key feature of modern online business. Marketing departments also use AI to segment their customer base and adapt their marketing techniques to these different segments.

In-product recommendations
: Companies such as Netflix and Amazon have in-product recommendations based on their understanding of the user needs. If someone looks for sports equipment, Amazon can recommend related products. It is the same for TV shows and movies on Netflix and other streaming platforms—they're able to make recommendations based on what you've watched previously. Everything is

based on customer data and it relies on relatively complex AI models that we will explore later.

Robots
: Examples include the Roomba vacuum cleaner, the incredible creations from Boston Dynamics that can even dance and perform complex tasks, the humanoid Sophia, etc.

Autonomous vehicles
: This type of system is equipped with different sets of advanced technologies, but some of them leverage AI techniques that allow cars to understand the physical context, and adapt to dynamic situations. For example, these vehicles can autonomously drive with no need to have a human driver, and they can make decisions based on different visual signals from the road and other cars. Tesla's Autopilot and other companies are great examples of this.

Security systems
: It includes both cyber and physical security. Same as fraud detection, AI helps security systems spot specific patterns from data and metrics, in order to avoid undesired access to precious resources. For example, the Microsoft Security Copilot detects hidden patterns, hardens defenses, and responds to incidents faster with generative AI. Another example would be AI-enabled cameras that can spot specific situations or objects from the video images.

Online search
: Systems such as Microsoft Bing, Google search, Yahoo, etc. leverage massive amounts of data and customized AI models to find the best answers to specific user queries. This is not a new concept, but we have seen how this kind of system has evolved a lot during the last years with the new Bing Chat and Google Bard apps. Additionally, we will see some examples for generative AI and web search applications in Chapter 3.

Predictive maintenance
: A very relevant case for industrial applications, it leverages different types of data to anticipate situations where machinery and industrial equipment may need maintenance before having specific issues. This is a perfect example of understanding past data to generate predictions, and it helps businesses avoid potential problems and approach maintenance activities in a proactive way.

Now that you understand the type of technologies and some of their typical applications, let's focus on how the AI models can learn, as this will be relevant for the general Generative AI topic of this book.

Types of AI Learning Approaches

As humans, we start to learn when we are babies, but the way we do it will depend on the process we follow. We can learn by ourselves, based on our own positive or negative experiences. We can also learn from the advice of adult humans, who previously learned from their own experience and that help us accelerate our own learning process. AI models are very similar, and the way to leverage previous experiences (in this case data and models) depends on the type of AI model learning approaches, as you can see in Figure 1-1:

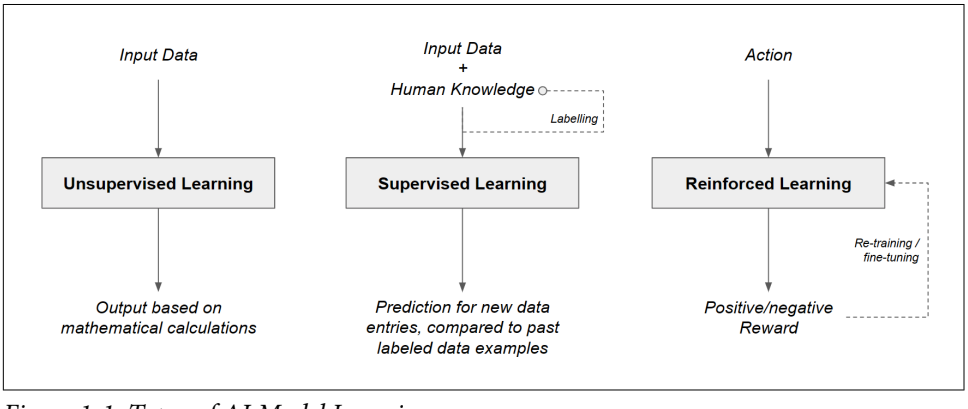

Figure 1-1. Types of AI Model Learning

Unsupervised learning
: It is based on unsupervised techniques that don't require human data annotation or support for the AI models to learn. This type usually relies on mathematical operations that automatically calculate values between data entries. It doesn't require any sort of annotation, but it is only suitable for specific types of AI models, including those used for customer segmentation in marketing. The king of unsupervised techniques is what we call "clustering", which automatically groups data based on specific patterns and model parameters.

Supervised learning
: Supervised learning is a very important type of learning for AI implementations. In this case, AI models use not only the input data, but also the knowledge from human experts (subject matter experts or SME) who can help AI understand specific situations by labeling the input data (e.g. What's a picture of a dog? What's a negative pattern?). It usually requires some sort of data annotation, which means adding additional information (e.g. an extra column for a table-based dataset, a tag for a set of pictures). In general, this is a manual process and getting it right will impact the quality of the AI implementation, being this as important as the quality of the dataset itself.

Reinforced learning

Last but not least, we have reinforced learning (RL) methods. Without getting too into the weeds with technical details, the main principle is the ability to simulate scenarios, and to provide the system with positive or negative rewards, based on the attained outcome. This kind of learning pattern is especially important for Generative AI, because of the application of *Reinforcement learning from Human Feedback (RLHF)* to Azure OpenAI and other models. Concretely, RLHF gets retrained based on rewards from human feedback (i.e. reviewers with specific topic knowledge). We will explore the details in Chapter 3.

There are different ways in which models learn, depending on the internal architecture, the type of data sources, and expected outcomes. For the purpose of this book, it is important to differentiate and understand the high level differences, as we will be referring to some of them in the context of Generative AI.

Let's get some expert insights to see the evolution from traditional AI to the new era of Generative AI, and how this is changing the adoption patterns from organizations out there. Concretely, our first industry guest is Mr. David Carmona, General Manager for AI & Innovation at Microsoft, and author of the wonderful O'Reilly's The AI Organization book.

Generative AI is here to stay, and the Azure OpenAI Service is already a key factor for adoption and democratization. Let's now explore the fundamentals of Generative AI, to understand how it works and what it can do for you and your organization.

About Generative AI

The term Generative AI refers to the field of artificial intelligence that focuses on creating models and systems that have the ability to generate new content, such as images, text, music, videos, etc.

As you may already know, this term has gotten a lot of relevance in 2022 and 2023, but it is not new. We can talk about probabilistic models in the 90s, such as latent variable models and graphical models, which aimed to capture and generate data distribution. Also, recent advancements in deep learning, specifically in the form of generative adversarial networks (GANs) and variational autoencoders (VAEs), have significantly contributed to the popularization and advancement of Generative AI.

The term "generative AI" has gained momentum as researchers, companies, and practitioners began to explore the potential of these techniques to generate realistic and creative outputs. The result is now obvious, as AI encompasses a wide range of applications and techniques, including image synthesis, natural language processing, music generation, etc. Obviously, this is an evolving field and both academia and industry continue to innovate.

As you can see in Figure 1-2, the generation capability can be seen as an extension of other existing types of AI techniques, which are more oriented to either describe, predict, or prescribe data patterns, or to optimize specific scenarios. Advanced AI techniques, including Operational Research (OR) and Generative AI allow adopters to go from "only insights", to automated decision making and actions:

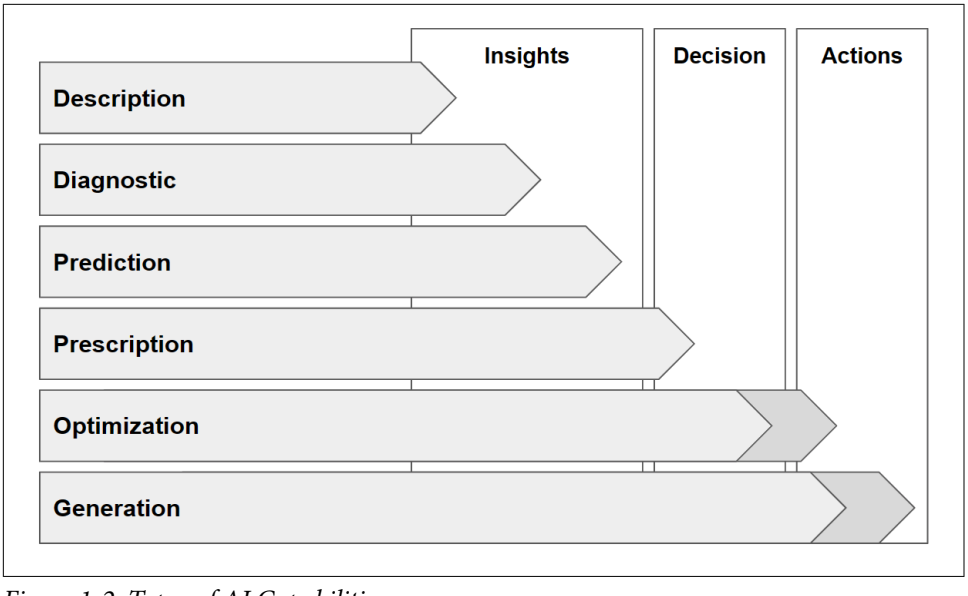

Figure 1-2. Types of AI Capabilities

From a technical point of view, these models work in a very particular way. Instead of "just" predicting a certain pattern for a data entry (e.g., forecasting the ideal insurance premium for a specific client), they generate several outcomes to a specific instruction. The interaction with the Generative AI model happens in a question-answer fashion, and this includes both direct instructions from humans (based on natural language instructions), and automated actions.

Concretely, there is a new term called *prompt engineering*. This term has emerged more recently in the context of natural language processing (NLP) and the development of language models. While there isn't a specific origin or a definitive moment when the term was coined, it has gained popularity as a way to describe the process of designing and refining prompts to elicit desired responses from language models.

Prompt engineering involves carefully crafting the instructions or input provided to a language model in order to achieve the desired output. It includes selecting the right wording, structure, and context to guide the model towards generating the desired response or completing a specific task. There are ongoing efforts to develop systematic approaches for designing effective prompts, fine-tuning models for specific tasks, and mitigating biases or undesired behaviors in language generation.

From the previously mentioned question-answer dynamic, as you can see in Figure 1-4, *prompt* is the question, and the answer is called *completion*. The term "completion" in the context of natural language processing and language models refers to the generation or prediction of text that completes a given prompt or input, and it became more widely used as larger and more powerful models like OpenAI's GPT were developed. In summary, the term "completion" in language models emerged from the evolving field of language modeling, reflecting the ability of models to generate or predict text that fills in or completes a given context or prompt.

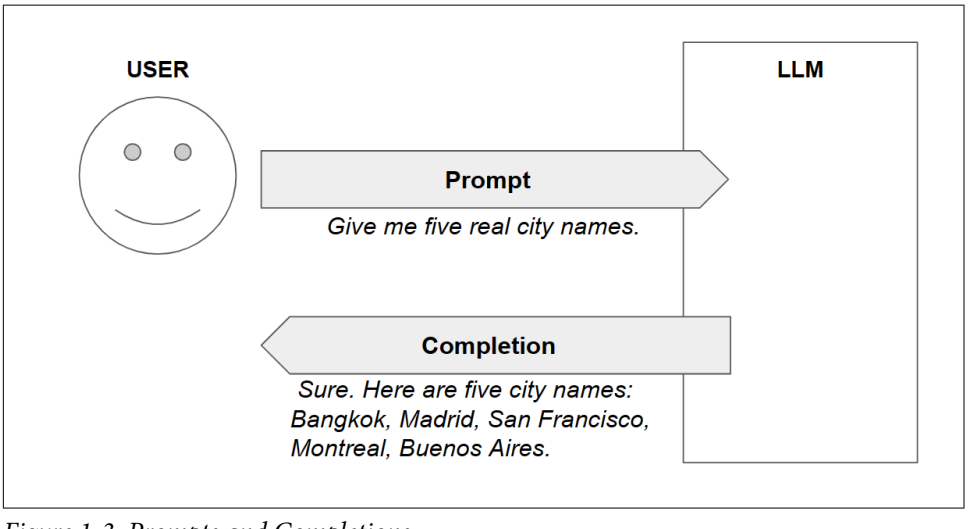

Figure 1-3. Prompts and Completions

Generative AI is a new kind of artificial intelligence, and its main advantage for wide adoption is the ability to enable communication between users and the Generative AI models via natural language prompts and completions. That's a great game changer, but let's see now the main kind of capabilities we can get from these models.

Main Capabilities

It is true that language and text-based information are a key aspect of Generative AI. However, language-based prompts can serve for other purposes. Concretely, companies and researchers are working on several streams:

Language
: Besides the core ChatGPT-kind of functionality, with questions and answers between the AI model and the human user, there are other related tasks that rely on linguistics but go a step further. What if you can use language as the creation catalyst for:

Code
: Technically, a programming language is just that… a language. LLMs are good at handling English or Spanish, but they are also great at understanding and generating code, and handling Java, Python, or C++ as any other spoken language. This may not be intuitive, but it makes sense to treat coding languages as any other language. And that's what Generative AI does.

Melodies
: Based on musical notes, LLMs can generate melodies the same as regular sentences. The potential of Generative AI in this area is still unexplored, but it shows promising results for music creation.

Lyrics
: Another example of linguistics, lyrics can be built based on specific criteria explained via prompt, in which the users can specify the type of words, inspiration, style, etc.

Image
: The principle behind it is surprisingly intuitive: Writing down the description (with simple natural language) of a potential image, to include it as part of the "prompt", then wait for the Generative AI engine to return one or several results matching that prompt, based on its own interpretation of previously consumed images. This kind of capability is very interesting for creative and marketing activities, where human pros can leverage image generation tools as an inspiration system.

Audio
: Imagine a technology that allows you to record your own voice for some minutes, and then reproduce and replicate it for whatever purpose you want. Some sort of scalable voice licensing that leverages audio data to detect patterns and then imitate them. In this case, the user doesn't necessarily rely on a specific prompt to obtain the audio creation piece.

Video
: Same as image generation, the input can be a prompt describing specific scenes with different levels of detail, for which the model will deliver an outcome of a video scene according to these details.

These are just some of the capabilities that Generative AI offers. They are fairly impressive, but certainly not the last step of the new AI era, as there are very relevant actors making sure that's the case. Let's see who are the main contenders next.

Relevant Industry Actors

While this book focuses on the Azure OpenAI Service, which is related to both Microsoft and OpenAI companies, it is important to understand the competitive landscape for generative AI. As you already know, this field is witnessing significant advancements and competition. Researchers and organizations are actively working to develop innovative models and algorithms to push the boundaries of generative AI capabilities. Here are some examples of relevant actors speeding up the competition:

OpenAI

Probably the key actor of the Generative AI wave in 2022-23. Not "open" from an open-source perspective (most of their code is proprietary, but they also have open source projects), but a great enabler for everyone to adopt Generative AI technologies, and for other companies and communities to replicate their strategy and to join the wave. It is an AI research laboratory and company known for developing advanced natural language processing models. OpenAI's origins can be traced back to December 2015 when it was founded as a non-profit organization by Elon Musk, Sam Altman, Greg Brockman, Ilya Sutskever, John Schulman, and Wojciech Zaremba. The organization's mission was to ensure that artificial general intelligence (AGI) benefits all of humanity.

OpenAI initially focused on conducting research and publishing papers in the field of artificial intelligence to foster knowledge sharing and collaboration. In 2019, OpenAI created a for-profit subsidiary called OpenAI LP to secure additional funding for its ambitious projects. The company's objective is to develop and deploy AGI that is safe, beneficial, and aligned with human values. They aim to build cutting-edge AI technology while ensuring it is used responsibly and ethically. They have democratized the access to different kind of AI models:

- *Conversational GPT models*, with their ultra famous ChatGPT application which relies on AI language models. It is based on the GPT architecture, which stands for "Generative Pre-trained Transformer", which are state-of-the-art language models known for their ability to generate human-like text and engage in conversational interactions. ChatGPT is designed to understand and generate natural language responses, making it well-suited for chat-based applications. It has been trained on a vast amount of diverse text data from the internet, allowing it to acquire knowledge and generate coherent and contextually relevant responses.
- *Generative AI models* for text (GPT-3.5 turbo, GPT-4, GPT-4 turbo, etc.), code (Codex), and images (DALL·E 3). These and other models are available, as we will see in Chapter 3, via Azure OpenAI Service and its playground.

- *State-of-the-art speech-to-text models*, such as Whisper, available as an open-source repository, but also as an OpenAI paid API. Additionally, Whisper models are available via Microsoft Azure.

Microsoft

Along with OpenAI, the other key actor and one of the earliest adopters of Generative AI technologies. Besides the Azure OpenAI Service (main topic of this book, which we will explore deeply along these book Chapters), Microsoft has adopted LLMs as part of their technology stack to create a series of AI Copilot for all their productivity and cloud solutions. We will explore more details during the next chapters, but the company strategy has clearly become AI-first, with a lot of focus on Generative AI and the continuous delivery of new features and integrations.

Hugging Face

Hugging Face is a technology company specializing in natural language processing (NLP) and machine learning. It is known for developing the Transformers library, which provides a powerful and flexible framework for training, fine-tuning, and deploying various NLP models. Hugging Face's goal is to democratize and simplify access to state-of-the-art NLP models and techniques. Hugging Face was founded in 2016 by Clément Delangue and Julien Chaumond. Initially, the company started as an open-source project aiming to create a community-driven platform for sharing NLP models and resources. Their Hugging Face Hub is a platform for sharing and accessing pre-trained models, datasets, and training pipelines. The hub enables users to easily download and integrate various NLP resources into their own applications, making it a valuable resource for developers and researchers. In addition to their open-source contributions, Hugging Face offers commercial products and services.

Meta

Formerly known as TheFacebook and Facebook, Meta is a multinational technology company that focuses on social media, digital communication, and technology platforms. It was originally founded by Mark Zuckerberg, Eduardo Saverin, Andrew McCollum, Dustin Moskovitz, and Chris Hughes in 2004. During the last years, they have created a very powerful organizational AI structure with relevant AI researchers, and meaningful open-source AI contributions. They have released several models, including their LLMs LLaMA and LLaMA2, an interesting data-centric option with good performance (based on the industry benchmarks) and lower computing requirements than other existing solutions. The latest LLaMA2 models are also available via Microsoft Azure (Azure is Meta's preferred platform for LLaMA2), with new features to fine tune and evaluate them via Azure AI Studio.

Google

Google is another key competitor and some of the most relevant AI innovators. Their Google Cloud Platform (GCP) introduced new AI-powered features in Google Workspace and G-Suite, and their Google Cloud's Vertex AI platform is used to build and deploy machine learning models and AI applications at scale. Same as Microsoft Azure, Google Cloud offers tools that make it easier for developers to build with generative AI and new AI-powered experiences across their cloud, and low-code generative AI tools. Last but not least, Google released Bard as their alternative to OpenAI's ChatGPT and Microsoft's Bing Chat.

Others

There are other actors working with more or less intensity in this area. For example:

- Amazon Web Services (AWS) took some time to release Generative AI-related products, but they recently announced their AWS Bedrock platform, a foundation model AI service to directly connect to Generative AI models. They offer their own foundational models, and others from third parties such as Cohere or Anthropic.
- Anthropic, another AI company, founded by former OpenAI employees. They also have their own ChatGPT-style kind of bot called Claude, which is accessible through a chat interface and API in their developer console. Claude is capable of a wide variety of conversational and text processing tasks while maintaining some good degree of reliability and predictability. Their Claude 2 model is available via APIs.
- Databricks, the Lakehouse platform (available as a native service on Microsoft Azure) that also released their own LLM. Concretely, an open source called Dolly 2.0, trained by their own employees, and the first open source LLM for commercial purposes.
- IBM, with the recent announcement of their new WatsonX platform, which includes model catalog, a lab/playground environment, and API-enabled integrations.
- Cohere, a LLM-first company, with their own offering of language models, and their Coral productivity chatbot, that works as a knowledge assistant for companies.

You can see in Figure 1-4 the exponential evolution of the Generative AI market with the timeline of new models by company, especially after ChatGPT release in 2022, with a 2023 full of model and platform releases:

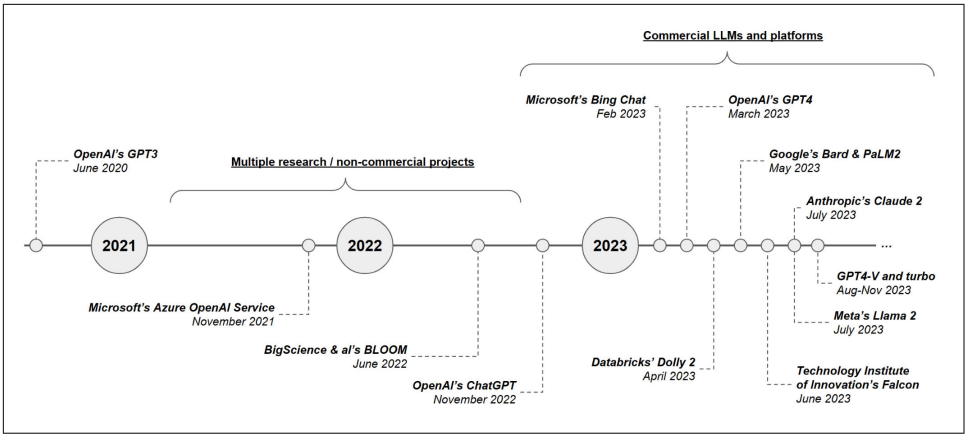

Figure 1-4. Simplified Generative AI Timeline

This timeline is a highly simplified version of the advancements and releases from different open source teams, and big companies. For more details, both the State of AI Report and the Stanford AI Index Report contain plenty of details about research and commercial models, as well as other relevant actors we haven't mentioned here. The list of Generative AI innovations will certainly evolve during the next months and years, and future implementations of existing models like Meta's LLaMA 2 and OpenAI's GPT-4 will likely focus on the efficiency of the models.

Now, let's see why Generative AI is a special kind of artificial intelligence, and explain a new concept called Foundation Models, which is the key differentiator when compared to traditional language models.

The Key Role of Foundation Models

There are several reasons why Generative AI is a total disruption. The perception of a never seen level of performance is one of them. The ability to interact by using plain language to send our instructions and to interpret the results is another one. However, one of the fundamental aspects for Generative AI to deliver the value we see nowadays, is the notion of *Foundation Models*.

Foundation models are base models pre-trained with a vast amount of information (e.g., LLMs) that are able to perform very different tasks. This is something new as traditional AI / NLP models focus on unitary tasks, one specific model by task (e.g., language translation).

For example, Azure OpenAI models such as GPT-4 can do plenty of things, by leveraging one single model. They can perform diverse tasks related to a specific Generative of capability, such as text/language, and help you analyze, generate, summarize, translate, classify, etc. all with only one model. In addition to that, if the models are

able to handle different kinds of inputs at the same time, like text and image, they qualify as *multimodal models*. You can see the main differences in Figure 1-5 below:

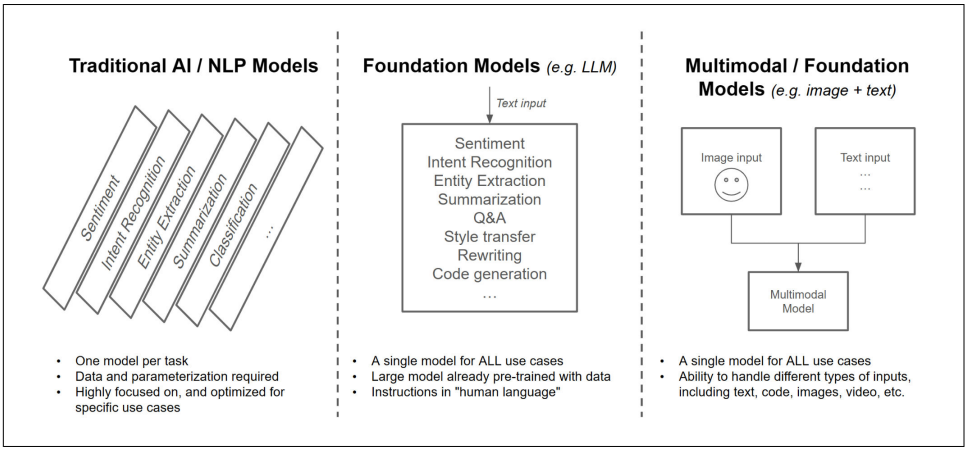

Figure 1-5. Traditional AI vs. Foundation Models

This flexible approach provides multiple options for the development of new use cases, and you will see later (in Chapters 2 and 3) how Azure OpenAI facilitates the configuration, testing, and deployment of these foundation models. But what does it represent in terms of AI disruption? Let's see first one of the fundamental reasons why Generative AI and companies such as OpenAI got so much attention during the last years.

Road to Artificial General Intelligence (?)

Before we dig into the core part of this book, it's important to contextualize all these innovations within the general state of artificial intelligence, and the current discussions on *Artificial General Intelligence (AGI)* due to the unexpected capabilities of GPT-4 and other LLMs.

You may remember some cinematographic references to what a lot of people imagine as an artificial intelligence—Skynet, Ultron, "I, Robot", etc. All of them showed some sort of superior intelligence, usually represented by strong and dangerous humanoid robots that evolve over time, and that plan somehow to replace or even destroy the human race. Well, even if it is not the purpose of this book to show a naive vision of what AI and its capabilities are, we will start by demystifying and clarifying the current level of development of artificial intelligence, so everyone can understand where we are and what are the realistic expectations of an AI system. For that purpose, here are three types of AI depending on their scope and level of intelligence:

Narrow AI
: The current kind of capabilities that AI systems and technologies are offering. Basically, an AI that can get a relatively big sample of past data, and then generate predictions based on that, for very specific tasks. For example, detecting objects from new images, recognizing people from audio voices, etc.

General AI (Artificial General Intelligence, AGI)
: The next goal for AI researchers and companies. The idea is to generalize the training process and its generated knowledge for AI to leverage them within other domains. For example, how can we make an AI-enabled personal assistant aware of the changing context? And then adapt previous learnings to new situations? Not 100% feasible today, but likely to happen at some moment.

Super AI
: The kind of artificial intelligence that movies and books are continuously showing. Its capabilities (cognitive, physical, etc.) are far superior to humans, and they can in theory surpass them. However, this kind of super intelligence is currently a futuristic vision of what an artificial intelligence could be. It's still not feasible and probably will not happen during the next years or even decades (this opinion will be different, depending on who you ask).

Bringing this back to the topic of generative AI, current discussions focus on the current stage or type of artificial intelligence. But the real question is, are we still talking about narrow AI? Are we getting closer to general AI? It is a fair question given the new level of performance and flexibility of foundation models to perform a variety of tasks. Regardless of the answer (that can go from the technical to the philosophical), reality is that generative AI in general, and Azure OpenAI Service in particular, are delivering capabilities we never dreamt about before.

Concretely, there was an early analysis of the GPT-4 model capabilities from the Microsoft team that explored this relation between the foundation models, and talks about a "close to human-level performance", and an "early version of an AGI system." Also, companies like OpenAI have declared the pursuit of AGI as one of their main goals. But this is an ongoing field, and there will be certainly news during the next months and years to come.

Up to now, we have covered all the fundamentals related to Generative AI topics, including evolution from traditional AI, recent developments, and ongoing discussions around performance and the impact of Generative AI. Let's now explore the details of the Azure OpenAI Service, with special focus on the story behind it, and the core capabilities.

Microsoft, OpenAI, and the Azure OpenAI Service

Microsoft, one of the main technology incumbents, and *OpenAI*, a relatively young AI company, have collaborated and worked together during the last years to bring some of the most impressive technologies ever, including AI supercomputers and large language models. One of the main aspects of this partnership is the creation of the Azure OpenAI Service, the main reason for this book, and a platform-as-a-service (PaaS) cognitive service that offers an enterprise-grade version of the existing OpenAI services and APIs, with additional cloud native security, identity management, moderation and responsible AI features.

The collaboration between companies became more famous in 2023, but reality is that it had several stages with very important milestones at both technical and business level:

- It started in 2019, when Microsoft announced a $1 billion investment in OpenAI to help advance their AI research activities, and to create new technologies.
- In 2021, they announced another level of partnership to build large-scale AI models using Azure's supercomputers.
- In January 2023, they announced the third phase of their long-term partnership through a multiyear, multibillion dollar investment to accelerate AI breakthroughs to ensure these benefits are broadly shared with the world.

Obviously, every step of this partnership has deepened the level of collaboration and the implications for both companies. The main areas of work are:

Generative AI Infrastructure
: Building new Azure AI supercomputing technologies to support scalable applications for both OpenAI and Microsoft generative AI applications, and porting existing OpenAI services to run on Microsoft Azure.

Managed Generative AI Models
: Making Microsoft Azure the preferred cloud partner for commercializing new OpenAI models via Azure OpenAI Service, which for you as an adopter means that any OpenAI model is available via Microsoft Azure, as a native enterprise-grade service in the cloud, in addition to the existing OpenAI APIs.

Also, Azure OpenAI Service is not the only Microsoft AI service, and it is part of the Azure AI Suite (shown in Figure 1-6) which includes other PaaS for a series of advanced capabilities that can co-live and interact to create new AI-enabled solutions:

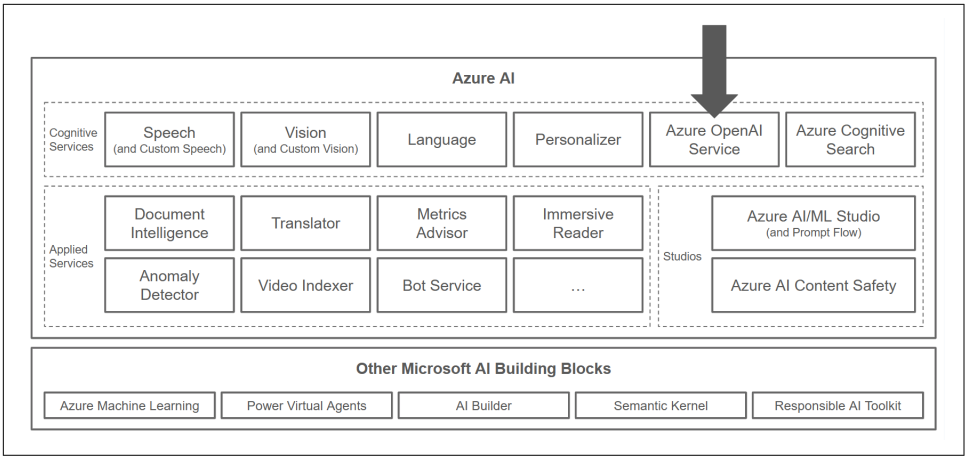

Figure 1-6. Azure OpenAI and other Azure AI Services

We will refer to some of these building blocks in Chapters 3 and 4, as most of these services interact seamlessly with the Azure OpenAI Service, depending on the envision solution architecture. But this is a highly evolving field, and Figure 1-7 shows the timeline of key Azure OpenAI breakthroughs during the last months and years:

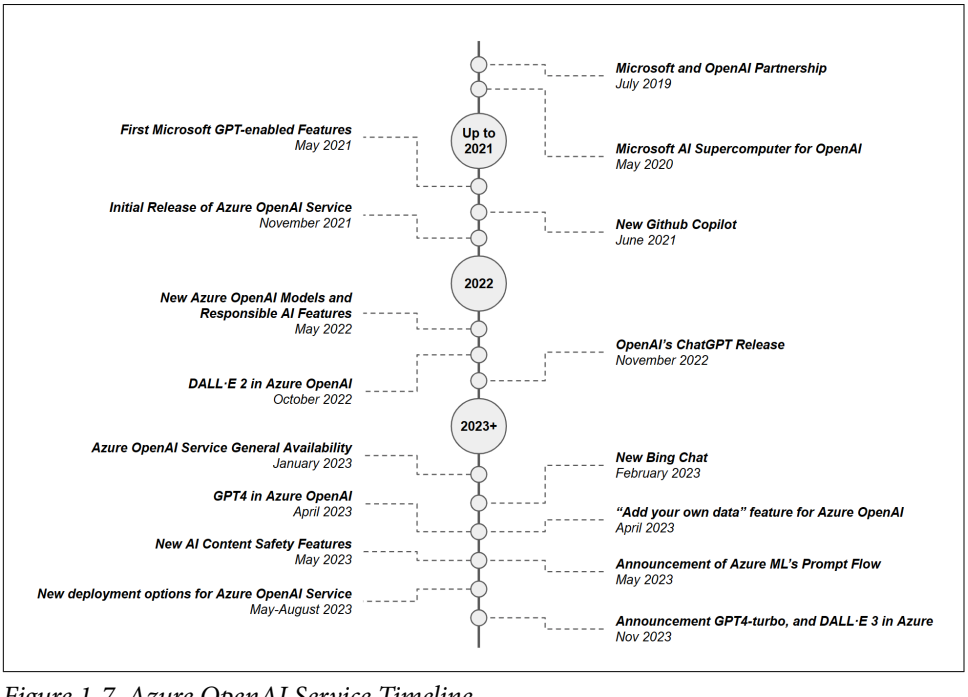

Figure 1-7. Azure OpenAI Service Timeline

If you want to understand more about the origins of the partnership and the initial developments, this podcast episode with Microsoft's CTO Kevin Scott and the Co-founder (and former CEO) Bill Gates is very interesting and explains how everything started.

The Rise of the AI Copilots

As part of their AI-enabled offerings, Microsoft is promoting the concept of *AI copilots*. They are personal assistants equipped with Microsoft's AI, OpenAI's GPT models, and other Generative AI technologies, designed to assist users in their tasks and goals, but not to replace humans and their jobs. Copilots work alongside users, providing suggestions, insights and actions based on AI. Users always have the control and the choice to accept, modify or reject the copilot's output. From a visual point of view, copilots are usually on the right side of the screen, and Microsoft has included them in several solutions:

GitHub Copilot
: GitHub Copilot is an AI-powered pair programmer that helps developers write better code faster. It suggests whole lines or entire functions right inside the editor, based on the context of the code and comments. GitHub Copilot is powered by OpenAI Codex, a system that can generate natural language and computer code. The Github Copilot is the original case and the first copilot of the Microsoft suite.

Bing Chat
: Bing Chat is a conversational AI service that helps users find information, get answers and complete tasks on the web. It uses OpenAI ChatGPT, a generative model that can produce natural language responses based on user input. Users can chat with Bing Chat using text or voice on the Edge browser or the Bing app. This is the first search engine to incorporate Generative AI features for a chat-based discussion.

Microsoft 365 Copilot
: Microsoft 365 Copilot is an AI-powered copilot for work that helps users unleash their creativity, improve their productivity and elevate their skills. It integrates with Microsoft 365 applications such as Word, Excel, PowerPoint, Outlook, Teams and Business Chat. It also leverages large language models (LLMs) such as OpenAI GPT-4 and ChatGPT to generate content, insights and actions based on natural language commands.

Windows Copilot
: Windows Copilot is an upgraded AI assistant for Windows 11 that helps users easily take action and get things done. It integrates with Bing Chat and ChatGPT,

as well as with Windows features and third-party applications. Users can interact with Windows Copilot using natural language commands.

Security Copilot
: Security Copilot is an AI-enabled security solution that helps users protect their devices and data from cyber threats. It uses AI to detect and prevent malware, phishing, ransomware and other attacks. It also provides users with security tips and recommendations based on their behavior and preferences.

Clarity Copilot
: Clarity Copilot is a feature that incorporates Generative AI into Microsoft Clarity, an analytics tool that helps users understand user behavior on their websites. It allows users to query their Clarity and Google Analytics data using natural language and get concise summaries. It also generates key takeaways from session replays using AI.

Dynamics 365 Copilot
: Dynamics 365 Copilot is a feature that brings next-generation AI to traditional Customer Relationship Management (CRM) and Enterprise Resource Planning (ERP) solutions. It helps users optimize their business processes, improve customer engagement and increase revenue. It leverages LLMs such as OpenAI's GPT-4 to generate insights, recommendations and actions based on natural language commands.

Summarizing, Microsoft has released a series of AI copilots for their product suite, and reality is that the Azure OpenAI Service is the key piece to *creating your own copilots*. We will analyze different building blocks of an AI copilot for cloud native applications (e.g., new terms such as plugins, orchestrators), but you can see in Figure 1-8 an adapted version of the "AI Copilot" layered architecture that Microsoft presented during the Microsoft Build 2023:

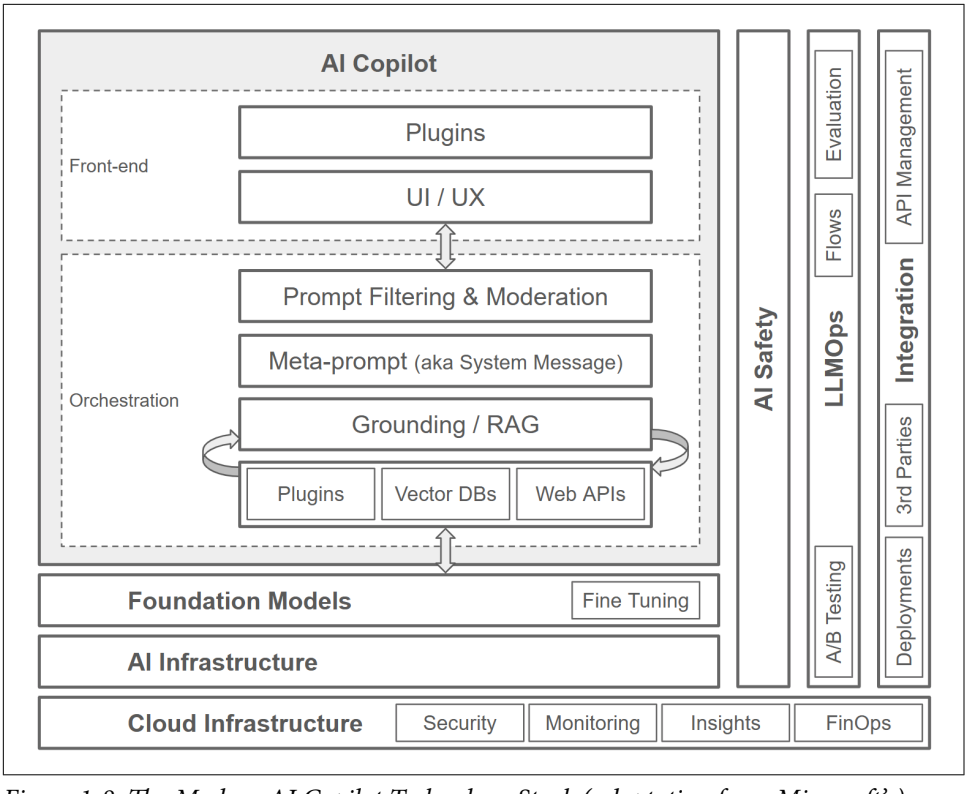

Figure 1-8. The Modern AI Copilot Technology Stack (adaptation from Microsoft's)

As you can see in the figure, the AI infrastructure and the foundation models are just part of the equation. Both a cloud-native architecture and specific Generative AI pieces are required to develop AI copilots for your existing and new applications, and that's exactly what we will cover in Chapters 2, 3, and 4. But before that, let's explore the high level capabilities and typical use cases with Azure OpenAI.

Azure OpenAI Capabilities and Use Cases

We will focus now on the core capabilities and potential use cases of Azure OpenAI-enabled systems, before going into architectural and technical considerations. Keeping in mind the flexible nature of foundation models, it is easy to imagine the multiple applications of the Azure OpenAI models. Let's explore the main capabilities in Table 1-1 (there are more, but you can use this as a baseline for your initial use ideation), aligned with those we have previously seen in this Chapter:

Table 1-1. Main Azure OpenAI Capabilities and Use Cases

Type		Capability and illustrative example	
Language	Content Generation / Analysis	Text Generation	Automatic creation of SMS with dynamic formats and content
		Topic Classification	Detect book topics based on their content, for automatic labeling
		Sentiment Analysis	Detect sentiment from social media reviews to detect pain points
		Entity Extraction	Find key topics from specific
		Call to APIs	Generate an API call and integrate it with other internal systems
		Subject Matter Expert Documents	Creation of role-based documentation based on books or repositories
		Machine Translations	Ondemand website translation
		Technical Reports	Generation of reports based on database and other information
		Agent Assistance	Step-by-step, dynamic blueprints for customer agents
	Summarization	Book Summaries	Summarization of long documents (e.g. books) with specific format and sections
		Competitive Analysis	Extraction of key factors from two companies for competitive analysis
		Social Media Trends Analysis	Summarization of keyword trends and connection with online news
		Reading Comprehension	Reformulation of key topics with a simpler language
	Search	Internet Results	Semantic search for internet topics
		Social Reviews Search	Detailed search of specific topics from social reviews in the internet
		Knowledge Mining	Extraction of knowledge from different sources, from the same topic
		Document Analysis	Search of key topics and other related terms for a specific document
	Automation	Claim Management	Automatic structuration of text-based information to send it as a JSON file
		Financial Reporting	Quarterly reporting based on social media summarization, figures from databases, and automation of the final report and its distribution
		Automatic Answers to Clients	Automatic voice-enabled answers, or chatbot discussions for L1 support

Type	Capability and illustrative example	
Coding	Natural language to Coding Language	Generating a Java loop, based on natural language instructions
	Coding Recommendations	Live coding recommendations from the development tool
	Automatic Comments	Automatic comment generation based on the written code
	Refactoring	Automated code improvements
	Code translation	Translation from one programming language to another
	SQL Queries in natural language	Database queries in natural language
	Code review	AI-enabled pair review
	Pull Request Info	Automated Pull Request comments
	Text JSON-ization	Conversion of plain text into JSON file with specific parameters
Image	Creative ideation	Random image generation related to a specific topic
	Podcast and music playlist images	Image generation based on podcast transcript or music lyrics
	Content syndication	Material for partner-enabled marketing
	Hyper-personalization	Visual customization based on user context
	Marketing campaign personalization	Visuals for marketing campaigns, based on user segment, topic, etc.

These are just a few examples of how to use the multiple capabilities of the Azure OpenAI Service models. They can be combined with other services, and the models may also evolve, so don't discard scenarios for audio or video generation.

Regardless of the type of capability and use case, the Azure OpenAI Service can provide support to different kinds of scenarios:

Completion
: Completions are used to generate content that finishes a given prompt. You can think of it as a way to predict or continue a piece of text. Completions are often useful for tasks like content generation, coding assistance, story writing, etc.

Chat
: Chat scenarios are designed to simulate a conversation, allowing back-and-forth exchanges with the model. Instead of giving a single prompt and getting a continuation, users provide a series of messages and the model responds to them in kind. Chat scenarios (like those powering ChatGPT) are useful for interactive tasks, including but not limited to tutoring, customer support, and of course, casual chatting.

Embeddings
: We will explore the notion of embeddings by the end of Chapter 2, but they basically allow us to consume specific knowledge from documents and other sources. We will leverage this sort of capability in several Chapter 3 scenarios.

The dynamic behind all these examples is the same. Azure OpenAI is a PaaS that works based on cloud consumption. Unlike other cloud services or APIs that bill their capabilities based on a number of interactions, Azure OpenAI (and other commercial LLM platforms) measure the service usage based on a new concept called "tokens". Let's see what this is about.

LLM Tokens as the New Unit of Measure

In general terms, cloud and Software-as-a-Service (SaaS) providers use very diverse ways to bill their services. From fixed monthly fees, passing by usage tiers based on volume discounts, to very granular units of measure such as characters, words, or API calls.

In this case, Generative AI has adopted the notion of *tokens*, which is a set of words or characters in which we split the text-based information. The tokens unit is used for two purposes:

- For *consumption*, to calculate the cost of the configuration and interactions with the Azure OpenAI models. Any API call, prompt (text request) sent to the model, and completion (answer) delivered by Azure OpenAI follows this unit. Concretely, the service pricing is based on cost by 1000 tokens, and it depends on the model type (GPT-3.5 turbo, GPT-4, GPT-4 turbo, DALL·E 2 and 3, etc.).
- For *capacity*, at both model and service levels:

  Token limit
  : The maximum input we can pass to any Azure OpenAI model (and Generative AI models in general). For example, GPT-3.5 turbo offers two options with a 4k and 16k token limit, GPT-4 reaches 32k, and this is likely to evolve during the next months and years. For updated information, visit the model availability page and check the "Max Request (Tokens)" column.

  Service quotas
  : The maximum capacity at resource, configuration, and usage level for any Azure OpenAI model. This is also an evolving information, and it is available via official documentation and the Quota section from the Azure OpenAI Studio. These limits are important for any deployment plan, depending on the type of application (e.g., if we are planning to deploy a service for massive Business-to-Consumer B2C applications). Also, there are recommended best practices to handle this limitation.

The specific amount of tokens depends on the number of words (other providers calculate tokens based on characters, instead of words), but also on their length and lenguaje. The general rule is 1000 tokens ~= 750 words, but OpenAI explains the specific way to calculate them depending on the case. Additionally, you can always use the Azure OpenAI Playground or OpenAI's tokenizer to calculate the specific estimation of tokens based on the input text.

Conclusion

We have concluded the first Chapter of this book—a mix of intro-level information related to AI and Generative AI, and the preliminary introduction to Azure OpenAI topics, including its recent developments, main capabilities, typical use cases, and its value as an AI copilot enabler for your own Generative AI developments.

Depending on your background, this information may be just a L100, but reality is that the concepts behind Azure OpenAI, even if they are new and include some new terms, can be seen as simple as it looks. A managed PaaS that will allow you to deploy your own cloud native, Generative AI solutions.

In Chapter 2, we will analyze the potential scenarios for cloud native development, its connection with Azure OpenAI, and the architectural requirements that will help you prepare everything, before even implementing your Azure OpenAI-enabled solutions. Same as Chapter 1, if you already have some preliminary knowledge of cloud native and Azure architectures, you may read it as a way to connect the dots and to understand the specific of these topics, adapted to Generative AI. If you are totally new to the topic, feel free to read the content and explore any external resource that may support your upskilling journey. We're just getting started!

CHAPTER 2

Designing Cloud-Native Architectures for Generative AI

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be Chapter 2 of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *mpotter@oreilly.com*.

Cloud native architecture is a way of designing and building applications that can take advantage of the cloud's unique capabilities and constraints. Cloud native applications are typically composed of microservices that run in containers, orchestrated by platforms like Kubernetes, and use DevOps and CI/CD practices to enable rapid delivery and scalability. Cloud native architectures are at the core of the Generative AI era.

Organizations such as the Cloud Native Computing Foundation (CNCF) are great catalysts of the cloud native best practices, and community development. Their goal is to be *"the vendor-neutral hub of cloud native computing, to make cloud native universal and sustainable"*. The CNCF is a great source of information and learning material for these topics.

As part of the cloud native movement, there are several projects and communities oriented to the use of cloud native architecture to enable scalable, reliable, and robust AI systems. They often require large amounts of data, complex algorithms, and specialized hardware to perform tasks such as image recognition, natural language processing, or recommendation systems. This is not always possible with traditional IT architecture patterns (e.g., monolithic applications).

The need for cloud native architecture for AI systems arises from the following reasons:

System performance
: AI systems need to process large volumes of data and run complex computations in a fast and efficient manner. Cloud native architecture enables AI systems to leverage the cloud's elastic resources, such as compute, storage, and network, to scale up or down according to the demand. Cloud native architecture also allows AI systems to use specialized hardware, such as GPUs or TPUs, that are optimized for AI workloads.

Agility
: AI systems need to adapt to changing business requirements, user feedback, and data quality. Cloud native architecture enables AI systems to deploy new features, models, or updates quickly and reliably using DevOps and CI/CD practices. Cloud native architecture also allows AI systems to experiment with different architectures, algorithms, or parameters using techniques such as A/B testing or canary deployments.

Innovation and integrability
: AI systems need to leverage the latest advances in AI research and technology. Cloud native architecture enables AI systems to access the cloud's rich ecosystem of AI services, tools, and frameworks that offer state-of-the-art functionality and performance. Cloud native architecture also allows AI systems to integrate with other cloud services, such as data analytics, IoT, or edge computing, that can enhance the value and intelligence of AI systems.

The most important areas for cloud native are described by CNCF as: CI/CD (Continuous Integration/Continuous Deployment), DevOps, microservices, and containers, as shown in Figure 2-1:

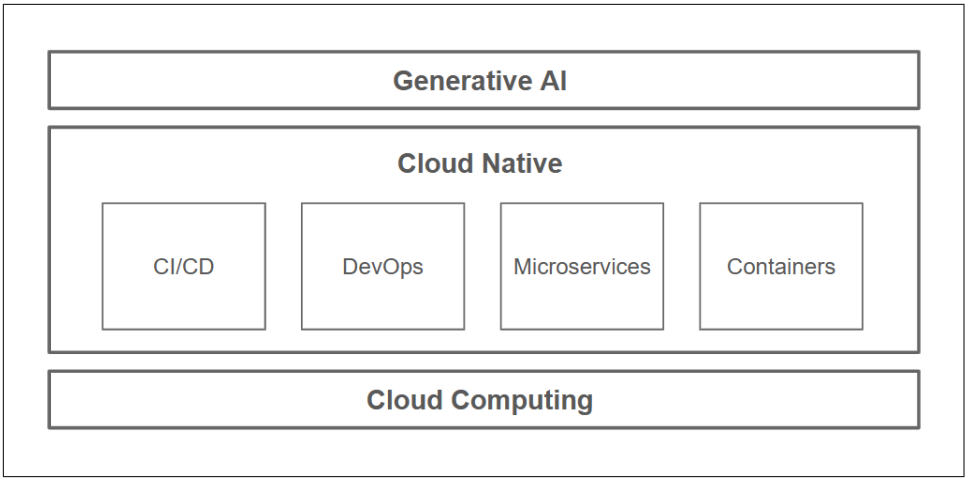

Figure 2-1. Cloud Native Building Blocks for Generative AI (adaptation from CNCF's)

Concretely, these four areas are relevant for Generative AI applications:

CI/CD (Continuous Integration/Continuous Deployment)
: Enables a streamlined and automated process for integrating code changes, building, testing, and deploying AI models and applications, and facilitates faster iterations and reduces time-to-market for generative AI developments.

DevOps
: Combines the principles and practices of DevOps for AI technologies to improve the development, deployment, and operations of AI systems, and facilitates the integration of generative AI into the overall software development lifecycle. It also ensures reliable monitoring, logging, and feedback loops, enabling quick identification and resolution of issues in generative AI systems.

Microservices
: Allows complex generative AI systems to be broken down into smaller, independent services, which enables modular development and deployment of different components of the AI system. It enhances scalability and flexibility too, as individual microservices can be developed, deployed, and scaled independently.

Containers
: Offers a lightweight and portable way to package and deploy generative AI models and applications, and enables easy scaling, replication, and orchestration of generative AI workloads.

Cloud native architecture is a key enabler for developing advanced and intelligent AI systems that can deliver high performance, agility, and innovation on the cloud platform. In this Chapter, we will explore how to prepare a cloud native architecture for

an AI-enabled system that leverages the Azure OpenAI Service, regardless of the kind of application you are planning to develop. Let's start by digging into some typical scenarios for AI cloud native development.

Modernizing Applications to Make Them Generative AI-Ready

This book focuses on the development of new cloud native applications with Azure OpenAI Service and the rest of the Microsoft Azure stack. However, there can be scenarios in which a company may try to leverage these capabilities for their existing applications. Let's compare both scenarios and see the approaches:

- *New cloud native applications* are designed from scratch using containerization and microservices architecture, enabling scalability, resilience, and elasticity. They leverage the four areas we have previously mentioned, and they make the deployment and maintenance of Generative AI applications a bit simpler.
- On the other hand, *existing apps* are likely required to be either migrated or modernized. This means they'll either be migrated to the cloud, or modified to align with cloud-native principles, such as breaking down monolithic architecture into microservices or introducing containerization. The modernization process involves step-by-step upgrades, addressing scalability, resilience, and fault tolerance, and adopting DevOps practices gradually.

Microsoft's modernization guide describes the process for migrating and modernizing existing on-prem / monolithic applications to the cloud, with specific cloud native features. Figure 2-2 illustrates the different levels of cloud modernization:

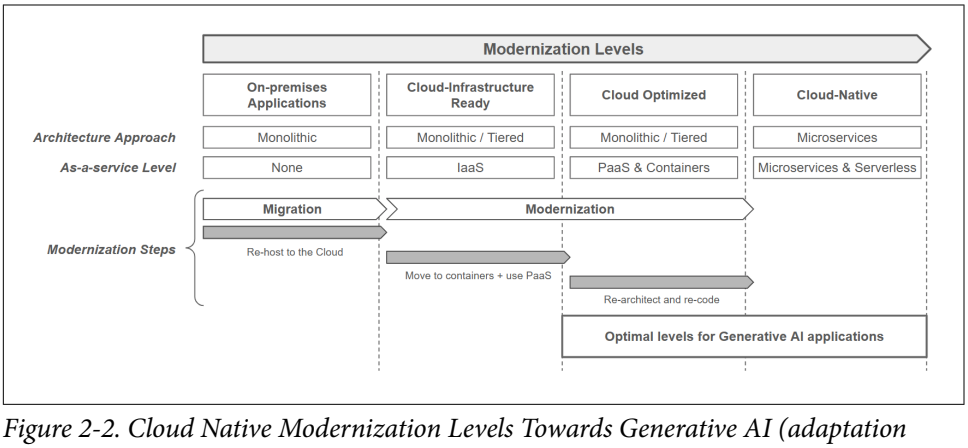

Figure 2-2. Cloud Native Modernization Levels Towards Generative AI (adaptation from Microsoft's)

Based on the modernization steps, there are different levels of maturity that go from existing on-premise applications, to full cloud-native ones. This is relevant for implementations with Azure OpenAI Service, as a native cloud-enabled PaaS, because new and existing applications will need some level of cloud readiness before integrating the Generative AI capabilities. Think of this as the way for the rest of the application blocks to connect with the Azure OpenAI Service in a cloud-enabled way, with native and simple integrations. Concretely, the levels of maturity are:

Cloud Infrastructure-Ready applications
: In this migration strategy, you simply transfer or relocate your existing on-site applications to an infrastructure as a service (IaaS) environment. While the structure of your applications remains largely unchanged, they are now hosted on virtual machines in the cloud. This straightforward migration method is commonly referred to as "Lift & Shift" within the sector, but it only gets part of the cloud value you can get from managed PaaS/SaaS kind of services.

Cloud Optimized applications
: At this stage, without making major code changes or redesigning, you can tap into the advantages of running your application in the cloud using contemporary technologies like containers and other cloud-managed services. This enhances your application's flexibility, allowing for quicker releases by optimizing your business's development operations (DevOps) practices. This enhancement is made possible by tools like Windows Containers, rooted in the Docker Engine. Containers address the challenges posed by application dependencies during multi-stage deployments. In this maturity framework, you have the option to deploy containers on either IaaS or PaaS, leveraging additional cloud-managed services such as database solutions, caching services, monitoring, and continuous integration/continuous deployment (CI/CD) workflows.

Cloud-Native applications
: This migration approach typically is driven by business needs and targets modernizing your mission-critical applications. At this level, you use cloud services to move your apps to PaaS computing platforms. You implement cloud-native applications and microservices architecture to evolve applications with long-term agility, and to scale to new limits. This type of modernization usually requires architecting specifically for the cloud, and even writing new code (or rewriting it), especially when you move to cloud-native application and microservice-based models. This approach can help you gain benefits that are difficult to achieve in your monolithic and on-premises application environment.

The last one is the end goal for optimal Generative AI-enabled applications, but any of these levels (especially the last two) would be "good enough" for any application to "connect" to the Azure OpenAI Service. The rest of the guide will focus on new cloud native applications, but if you plan to leverage Azure OpenAI Service for existing

applications, please do start by evaluating them and analyzing the next migration or modernization steps towards AI adoption.

Now, let's focus on the key advantages of cloud native, and the key Azure-enabled building blocks that will allow you to build your Azure OpenAI solutions.

Cloud Native Development

Now, part of the idea behind cloud native architectures is to split code development into different pieces called microservices, so all modules communicate based on a functional flow, without being part of the same technical block. This has a series of advantages, not only for Azure OpenAI-enabled development, but any cloud native implementation. We can imagine several reasons to leverage a microservices architecture:

Modular and Granular AI Functionality
: In AI applications, different tasks such as data preprocessing, feature extraction, model training, inference, and result visualization may be involved. By implementing each of these functionalities as separate microservices, the AI system becomes more modular and granular. This allows developers to focus on building and maintaining individual services, making it easier to understand, develop, test, and deploy specific AI components.

Scalability and Performance Optimization
: AI workloads can vary in intensity, with some tasks requiring more computational resources than others. By breaking down an AI application into microservices, each service can be scaled independently based on its specific resource needs. This scalability ensures efficient resource utilization and improved performance. For example, model training and inference services can be scaled independently to handle varying workloads, providing better response times and overall system performance.

Flexibility for AI Algorithm Selection
: AI applications often require experimenting with different algorithms, models, or data sources to achieve the desired outcome. With microservices, developers can easily swap out or update individual AI services without affecting the rest of the system. This flexibility enables rapid prototyping, experimentation, and iteration with different AI approaches, facilitating the discovery of the most effective algorithms or models for specific tasks.

Integration with External Services
: Microservices architecture promotes loose coupling and well-defined APIs, making it easier to integrate AI services with external systems, tools, or services. This allows AI functionality to be leveraged across different applications, domains, or platforms. For instance, an AI service for natural language processing can be

exposed via an API and utilized by multiple applications or integrated into a chatbot or customer support system.

Now, if we think about Generative AI-enabled applications with Azure OpenAI Service, the goal is to structure the end-to-end architecture in a way that makes sense and connects the "AI pieces" to both back-end elements (code, cloud resources), and front-end interfaces (one or several, depending on the application), as you can see in Figure 2-3:

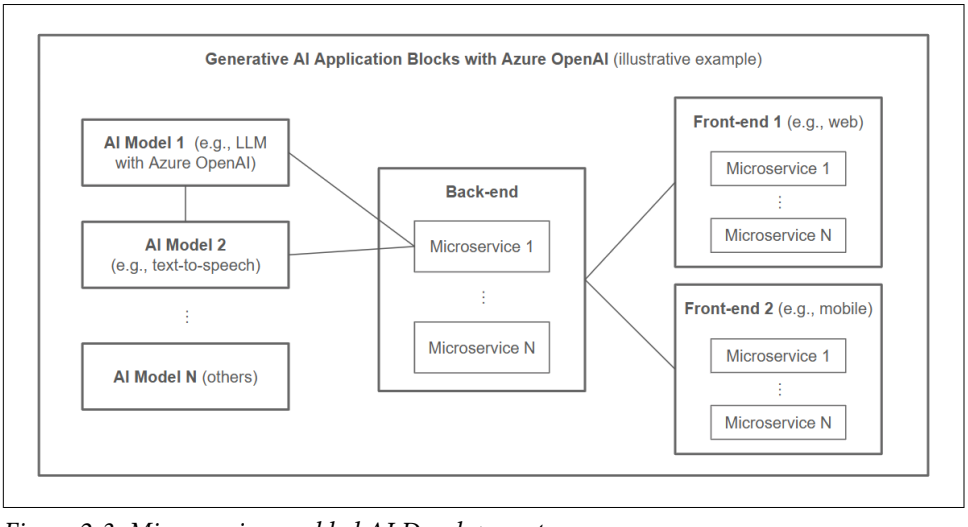

Figure 2-3. Microservice-enabled AI Development

All the involved elements need to be interoperable, replaceable, and available. For that purpose, organizing the building blocks in microservices is key. The next two sections contain containerization and serverless approaches. Let's see their role as cloud native enablers.

Microservice-based Apps and Containers

Cloud-native development approaches leverage the power of the cloud, by choosing the right way to develop and to deploy applications. They rely on containerization, which often refers to Docker-kind of containers, and Kubernetes orchestration. As they are both based on international standards (e.g., the Open Container Initiative OCI), cloud-native applications are usually portable and scalable in different public and private cloud providers.

For Microsoft Azure, the key managed containerization services are AKS (Azure Kubernetes Service) and ARO (Azure Red Hat Openshift). While both are managed Kubernetes services offered by Microsoft, they have some key differences:

- AKS is a managed Kubernetes service provided by Microsoft Azure, utilizing the native Kubernetes technology. It offers a fully managed Kubernetes cluster on Azure infrastructure and focuses on providing a streamlined and simplified Kubernetes experience on Azure. It provides essential Kubernetes features, including scaling, load balancing, and deployment management. AKS integrates well with other Azure services and provides native Azure resource management and monitoring capabilities. You can find pricing information here.
- On the other hand, ARO is a joint offering between Microsoft and Red Hat, built on the Red Hat OpenShift Container Platform. ARO incorporates the Kubernetes technology but provides additional features and integrations from the OpenShift platform. ARO provides a more comprehensive and enterprise-focused platform with additional security, compliance, and management capabilities.

In summary, they differ in terms of the underlying technology, vendor, and platform features. The choice between AKS and ARO depends on the specific requirements and preferences of your organization, such as the need for additional enterprise features and any existing investments or partnerships with Red Hat.

Now we have explored the containerization options in Azure, let's understand the notion of serverless and its relevance for microservice-based implementations.

Serverless Workflows

An alternative or complementary option is the serverless approach. Serverless computing is a cloud computing model that allows developers to build and run applications without the need to manage underlying infrastructure. It is particularly beneficial for AI workloads, including generative AI, as it provides a scalable and cost-effective solution.

In serverless architecture, developers focus on writing code for specific functions or tasks, known as serverless functions. These functions are executed in containers that are managed and scaled automatically by the cloud provider, as you can see in Figure 2-4. This eliminates the need for developers to provision and manage servers, making it easier to deploy and maintain AI applications.

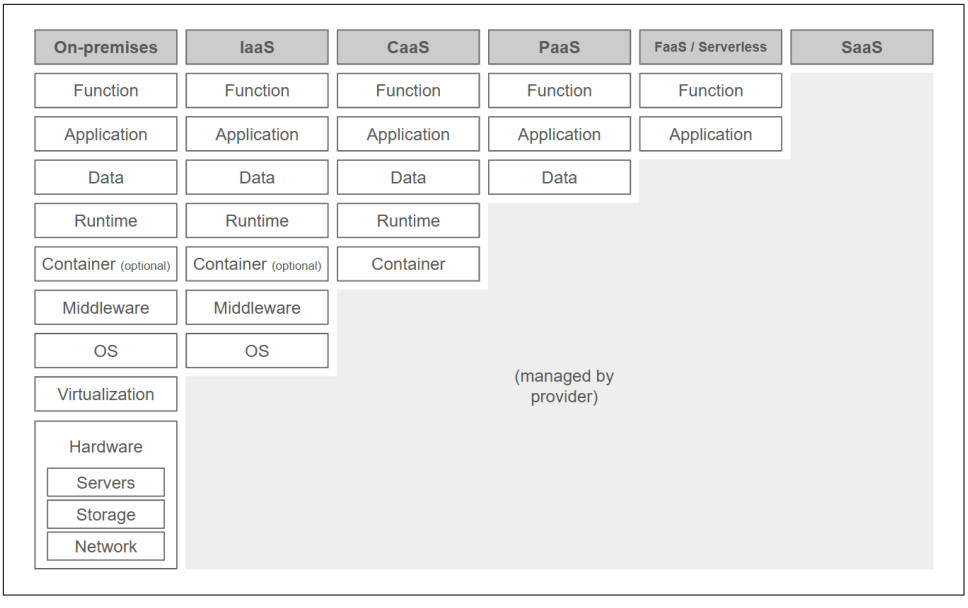

Figure 2-4. Managed Cloud As-a-Service Levels

Much like other cloud native elements, one of the key advantages of serverless for AI workloads is scalability. Generative AI models often require significant computational resources, especially when training large models or generating complex outputs. Serverless platforms automatically scale resources based on demand, allowing AI applications to handle fluctuations in workload without manual intervention. This scalability enables efficient resource utilization and cost optimization, as developers only pay for the actual compute resources used during execution.

Another advantage of serverless computing is its event-driven nature. Serverless functions are triggered by specific events, such as HTTP requests or messages from message queues. This event-driven architecture is well-suited for AI workloads that require real-time or asynchronous processing. For example, generative AI applications can be triggered by user interactions or scheduled tasks, allowing them to generate outputs on-demand or periodically. Additionally, serverless can be used to perform actions within a Generative AI pipeline.

There are some limitations related to serverless platforms, such as execution time limits, memory constraints, and deployment package size limits. However, techniques like function composition, caching, and parallel execution can help improve the efficiency and responsiveness of generative AI applications running on serverless architectures. Fine-tuning resource allocation and optimizing data processing pipelines can also contribute to better overall performance.

In general terms, you will be combining PaaS such as Azure OpenAI, plus containerized and/or serverless pieces depending on your implementation approach. We will now explore the web development part of your applications, for you to get an initial idea of the services that Azure OpenAI leverages to deploy Generative AI-enabled web-based applications.

Azure-based Web Development and CI/CD

Now, let's focus on the development building blocks that go beyond the core AI capabilities. As a cloud native practitioner, you will likely split your application code in several pieces. As you have already seen, those blocks are microservices that could contain the back-end and front-ends modules (mobiles applications, websites, intranets, etc.).

The interesting part comes when you discover you can host the web-based applications directly via Azure App. The Azure App Service is a platform-as-a-service (PaaS), a fully managed service that allows adopters to build, deploy, and scale web applications and APIs without the need to manage underlying infrastructure. It supports various programming languages and frameworks.

It enables web, mobile, and API app development, as well as integration and workflows (Logic Apps), integration and continuous integration-delivery (CI/CD), monitoring, and the simple integration with the whole Microsoft Azure suite.

Overall, Azure App Service simplifies the process of building, deploying, and scaling web applications and APIs in the Azure cloud. It offers a robust and feature-rich platform that enables developers to focus on application development while benefiting from the scalability, availability, and management capabilities provided by the Azure platform.

You will see in Chapter 3 how Azure OpenAI offers simple deployment options that leverage the Azure App Service, to create chat-based applications with pre-existing templates.

If you want to dive deeper into any of these topics, please visit the following links:

- Application hosting: Overview - Azure App Service | Microsoft Learn
- Github for CI/CD Configure CI/CD with GitHub Actions - Azure App Service
- Explicative video How to deploy your web app using GitHub Actions | Azure Portal Series

I will now cover the fundamentals of the Azure Portal, mostly for readers with none or low Azure experience, as a way to help you understand how to search, configure, and deploy Azure OpenAI and other related services. If you have already worked with Azure and its portal, you may skip this section.

Understanding the Azure Portal

The Azure portal is a web-based user interface provided by Microsoft Azure that allows users to manage and interact with their Azure resources. It serves as a central hub for accessing and managing various Azure services and functionalities, including Azure OpenAI Service. The portal provides a visually appealing and intuitive interface that simplifies the management and monitoring of Azure resources.

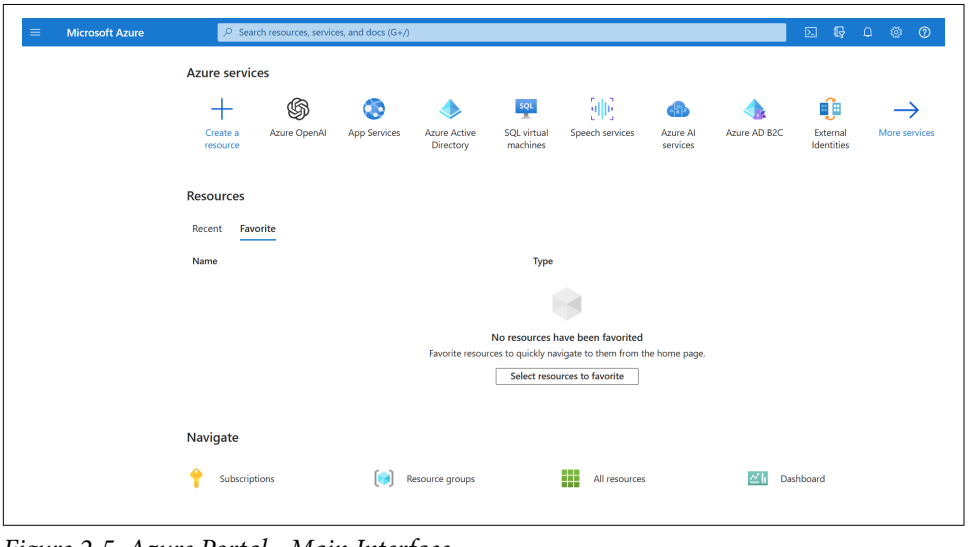

Figure 2-5. Azure Portal - Main Interface

As you can see in Figure 2-5, it includes a customizable dashboard that provides an overview of your Azure resources, recent activities, and personalized tiles for quick access to frequently used services.

The navigation pane on the left side of the portal allows you to access different categories of Azure services, including Compute, Storage, Networking, Security + Identity, AI + Machine Learning, and more. You can see the sequence in Figure 2-6.

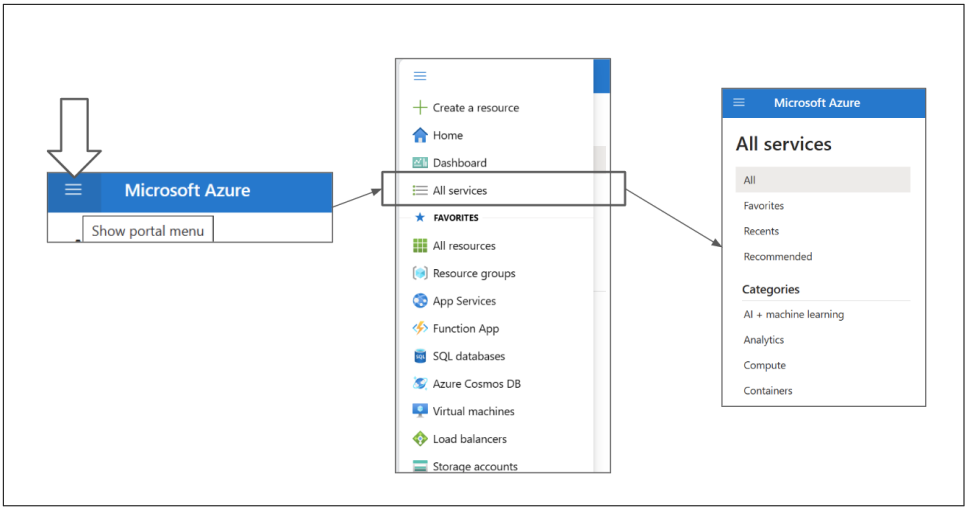

Figure 2-6. Azure Portal - Left Panel

Also, clicking on a specific category expands a menu with subcategories and services within that category. You can actually find the Azure OpenAI Service within the "AI + Machine Learning" category:

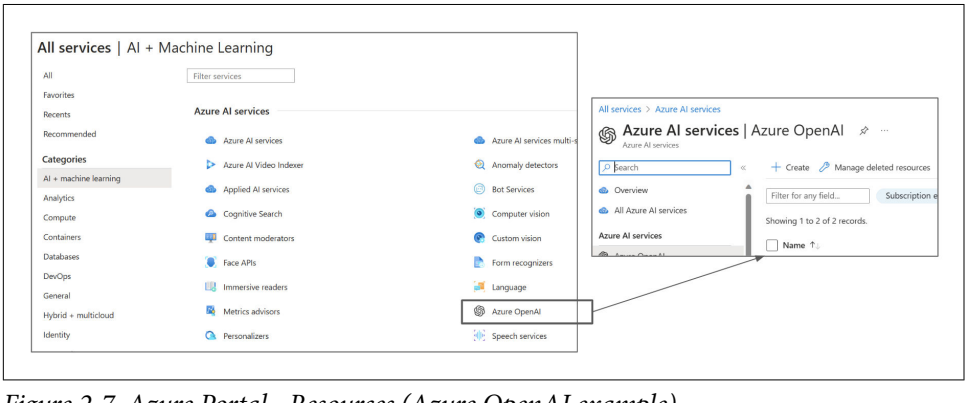

Figure 2-7. Azure Portal - Resources (Azure OpenAI example)

Alternatively, the Azure Portal offers a search bar at the top, allowing you to quickly find services, resources, or documentation. As you can see in Figure 2-8, you can search by keywords or use the natural language query to locate specific functionalities or resources within Azure. Basically, you can find "Azure OpenAI" by just typing it there.

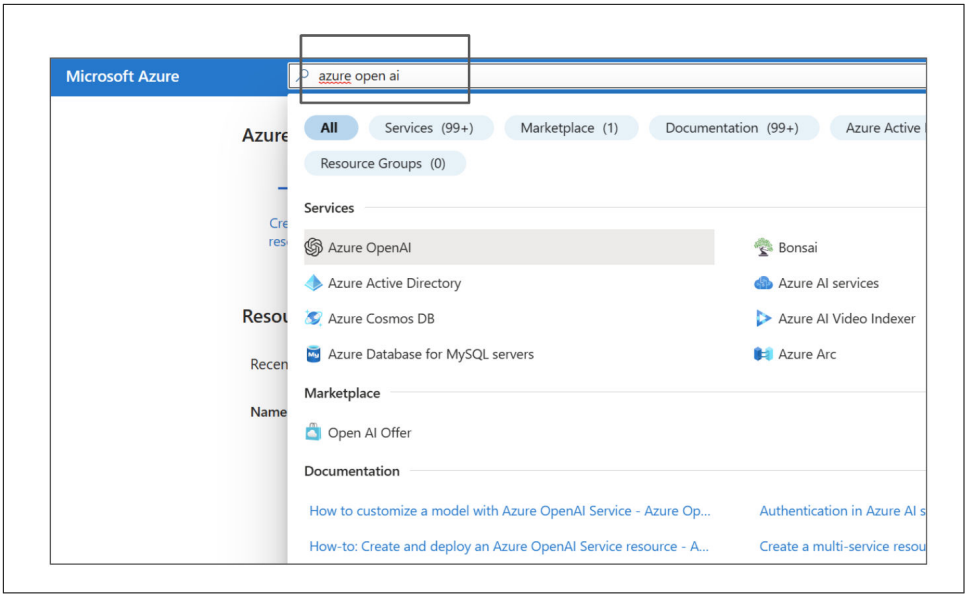

Figure 2-8. Azure Portal - Search (Azure OpenAI example)

Each Azure service has its own dedicated blade, which is essentially a panel that provides detailed information and management options for that service. If you choose "Azure OpenAI" from either the search engine or the left panel, you will enter your resource details you can see in Figure 2-9. Basically, you are able to create new resources of type Azure OpenAI, or manage those previously deployed. If you choose Create, you can see the main information required to deploy a new Azure OpenAI Service.

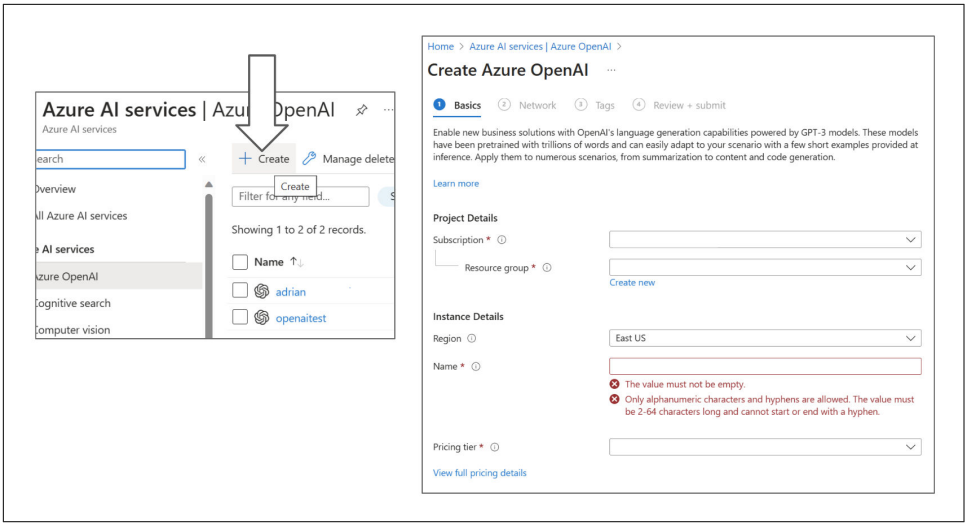

Figure 2-9. Azure Portal - Resource Details (Azure OpenAI example)

Concretely, you can find details related to your subscription, geographic region preferences, the unique name chosen for your Azure resource, and the pricing tier (i.e. tiers are the level of pricing based estimated usage, but for now there is only one option for Azure OpenAI, called "Standard SO". Any update should be available via the official pricing page, and the Azure calculator).

In addition to managing individual resources, the Azure portal allows you to create resource groups to logically organize and manage related resources together. This is an interesting feature, and a recommended best practice to group the required resources for your Generative AI implementations with Azure, including Azure OpenAI Service and others we will need for our projects.

If you haven't created an Azure account before, the preliminary step is to create a free one here. It usually includes credits for a value of USD $200 for initial experimentation. It requires a corporate email for the specific account and payment information.

We will explore the details of Generative AI implementation approaches with Microsoft Azure in Chapters 2 and 3, but the idea behind the Azure Portal is to facilitate the deployment, management, and maintenance process of the different resources required to create these architectures, regardless of the type of service. Deploying any Azure services from the Azure portal involves several steps, so remember the high-level process:

1. *Sign in to the Azure portal*
   Open a web browser and navigate to the Azure portal and signup with your Azure account credentials.

2. *Create a resource*
   To deploy an Azure service, you need to create a resource. A resource represents a service or component in Azure, such as a virtual machine, a storage account, or a database. Click on the "Create a resource" button in the Azure portal.

3. *Select a service*
   In the resource creation wizard, you'll see a list of available Azure services. Choose the service you want to deploy by browsing through the categories or using the search bar.

4. *Configure the resource*
   Once you've selected a service, you'll be taken to a configuration page where you can specify the settings for the resource. The options available depend on the specific service you're deploying. Fill in the required information, such as resource name, region, pricing tier, and any other relevant settings.

5. *Review and create*
   After configuring the resource, review the settings to ensure they are correct. You can also enable additional features or add-ons if available. Once you're satisfied, click on the "Review + Create" button.

6. *Validation and deployment*
   Azure will validate the configuration settings and check for any potential issues. If everything is in order, click on the "Create" button to initiate the deployment process.

7. *Monitor the deployment*
   Azure will start provisioning the resources based on your configuration. You can monitor the deployment progress in the Azure portal. Depending on the service, the deployment may take a few minutes to complete.

8. *Access and manage the deployed service*
   Once the deployment is finished, you can access and manage the deployed service through the Azure portal. You can view its properties, make changes to its configuration, monitor its performance, and perform other administrative tasks as needed.

This is the process for most of the Azure resources, but there are other deployment methods such as Azure Resource Manager templates, or command-line tools such as Azure CLI or Azure PowerShell, all of them for more advanced admin/technical users. Feel free to explore them if you want to learn more.

For the Azure OpenAI Service, you can always visit the official resource deployment guide which summarizes the steps we just walked through. Other information you may want to review before deploying the service includes the main product page, the previously mentioned pricing guide, the service availability by geographic region (for example, if you deploy the service from the European Union, you may want to get the closer region such as West Europe in Amsterdam, for better latency, performance, and maybe pricing), and the general documentation.

Now that you know how to use the Azure Portal, and the key information about the Azure OpenAI Service deployment process, let's analyze some important considerations at model, general architecture, and integration level). This will be key to create the end-to-end implementations we will see in Chapter 3.

General Azure OpenAI Considerations

Now that we have explored the notion of cloud native development with Azure, and the fundamentals of the Azure Portal for Azure OpenAI Service, let's go deeper into the different AI models that are available, and the high level architectures so you can know how to make sense of the Azure-enabled Generative AI offerings.

Available Azure OpenAI Models

First of all, most of the cloud-enabled PaaS resources from any public cloud, including those from Microsoft Azure, leverage native end-points and APIs as a way to connect, and to consume their models. This is the case for Azure OpenAI Service and the rest of Azure AI Services we have seen before in this Chapter 2.

Also, there are visual elements such as the Azure AI/ML Studio (formerly known as "Azure ML Studio", not to be confused with the Azure OpenAI Studio that we will explain and leverage in Chapter 3), which provides access to different proprietary and open source AI / Foundation Models. It includes a model catalog to leverage the curated selection of models, including those from Azure OpenAI, Meta, and Hugging Face (i.e. the Hugging Face Hub in Azure, announced by both Microsoft and Hugging Face during the Microsoft Build 2023). It also allows us to test and deploy those models in a very simple way.

As you can see in Figure 2-10, if you visit the Studio page, you will get access to your existing workspaces, or you will be able to create a new one if it is your first time connecting to the studio.

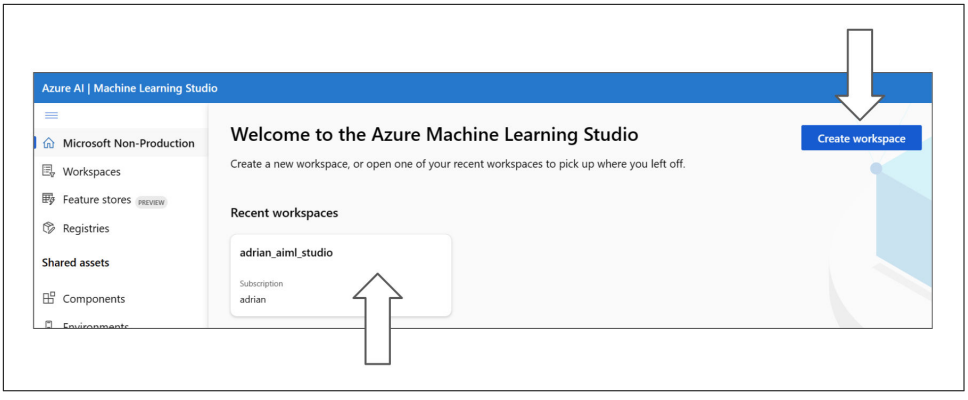

Figure 2-10. Azure AI Studio - Main Interface

If you access the workspace, you will see the same kind of visual interface we have seen before in this same chapter. In Figure 2-11, the left panel for the workspace menu offers all options related to data, models, endpoints, required resources, etc. For the sake of simplicity, we will focus on two main features. The model catalog, and later in Chapter 4, the Prompt Flow functionality.

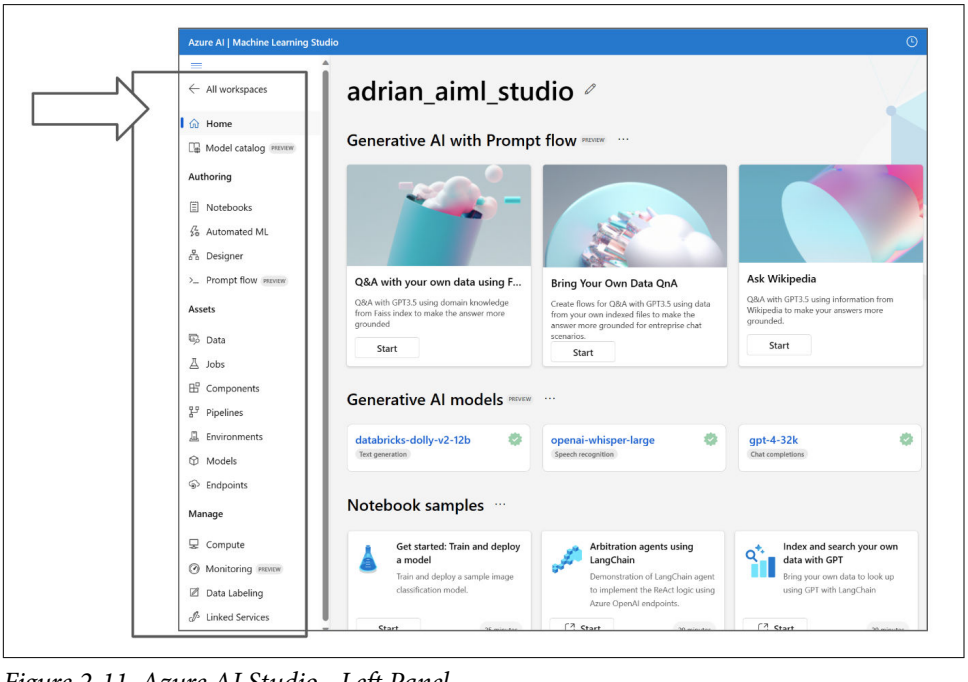

Figure 2-11. Azure AI Studio - Left Panel

Concretely, if you choose the Model catalog option and search "Azure OpenAI", or click directly on the tile as shown in Figure 2.12, you will get access to the updated list of available Azure OpenAI models:

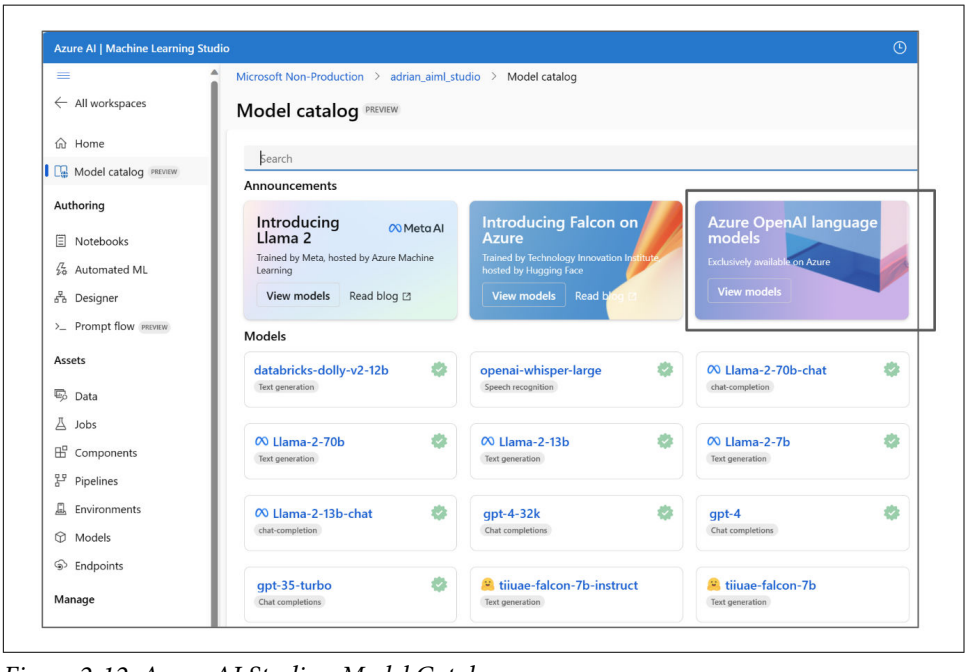

Figure 2-12. Azure AI Studio - Model Catalog

The models you can see in Figure 2-13 are those available at the moment of writing, but depending on when you check the catalog, you will likely find these and / or others. An alternative way to check all the available models at the moment is to use the Model List API.

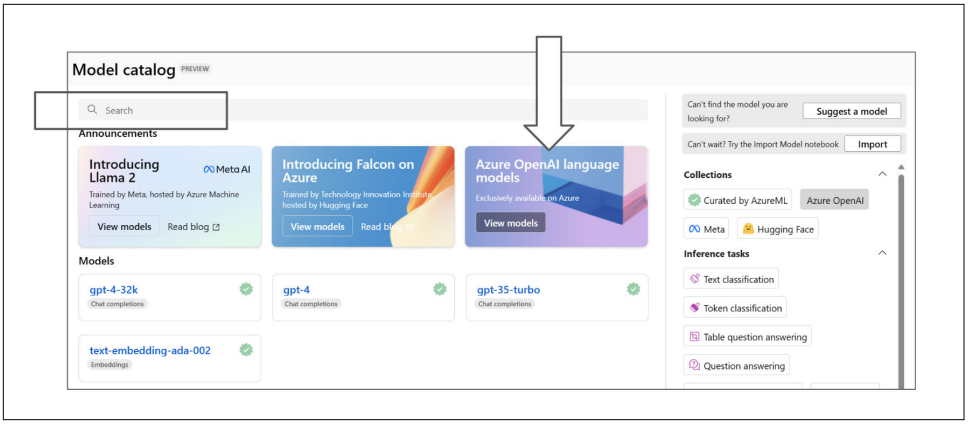

Figure 2-13. Azure AI Studio - Azure OpenAI Models

Now, keeping in mind the evolving nature of the availability of Azure OpenAI models, explore the key model families and some examples of specific models that you will leverage for your Generative AI projects. This will certainly change over time, but it is a good beginning.

The Azure OpenAI Service splits its capabilities into different *model families*. A model family typically associates AI models by their intended task, such as natural language understanding, code generation, or image synthesis. Some of the most popular Azure OpenAI model families are:

Language-related models

GPT-3.5 turbo
: A model that improves on previous GPT-3 versions, and can understand and generate natural language and code. This is the underlying model for ChatGPT from OpenAI, which relies on Microsoft's Azure infrastructure. There are several versions with different context length limits, including those for 4K and 16K tokens, which is the measure of the maximum text input.

GPT-4 and GPT-4 turbo
: LLMs with better performance (and higher cost) than GPT-3.5 turbo, which can handle more complex tasks and generate more accurate and diverse outputs. It can also get bigger text inputs (which we usually define as "context") than their predecessors.

Speech
: There are several options in Azure to use the Speech-to-text Whisper model from OpenAI: The Azure AI Studio's model catalog, the Azure AI Speech service, and of course the Azure OpenAI Studio.

Other models

Codex for programming code
: A series of models that can understand and generate code, including translating natural language to code. The reality is that Codex was initially a separate model, but after some time OpenAI added its capabilities to the regular GPT-3.5 turbo and GPT-4 / GPT-4 turbo language models. This means, the same models handle both natural language and programming code.

DALL·E for image
: A series of models that can generate original images from natural language. It is the model behind tools like Bing Create or Microsoft Designer, and it is directly available from the Azure OpenAI Studio, as we will see in Chapter 3.

It is important to differentiate the different model families and their specific capabilities to understand which ones we will use for our Generative AI projects. Also, the tradeoff of different Azure OpenAI models depends on the use case and the available budget. Generally speaking, the more capable models like GPT-4 can handle more complex tasks and generate more accurate and diverse outputs, but they also consume more resources and incur higher costs. We will explore several scenarios in Chapter 3 that can work with both GPT-3.5 turbo y GPT-4. You can also explore the whole variety of OpenAI models, including some deprecated ones which are still traced via OpenAI's documentation.

Besides all these functionalities, one of the key features for LLM-enabled systems is called *embeddings*. This is a general term related to Natural Language Processing (NLP) and LLMs. Concretely, embeddings are a way of representing data in a low-dimensional space. They are often used to capture the semantic meaning of words, images, or other types of data. For example in Figure 2-14, an embedding model can map a word to a vector of numbers, such that words with similar meanings have similar vectors. This means, we can connect pieces of information which are not directly connected, but that may have a mathematical or linguistic connection (e.g., several knowledge bases from companies of the same sector, internal and external sources, etc.).

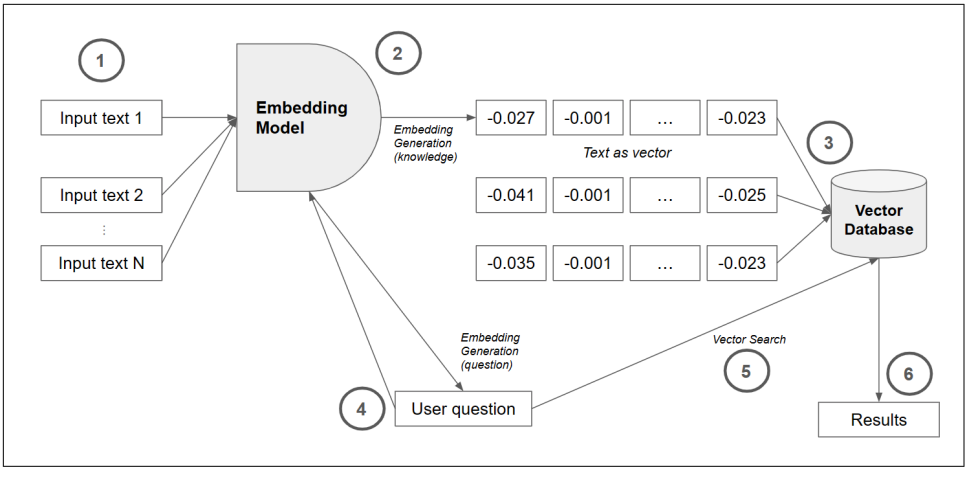

Figure 2-14. Embedding Models

Concretely, this examples illustrates the typical *generation and search process*:

1. We collect different data inputs (PDFs, word, URL, etc.) to create our knowledge base. This is a simplified view, sources are previously processed to extract the text-based information. We will see options such as official accelerators and Azure AI Document Intelligence in Chapter 3.
2. We leverage the Embeddings API to generate the embeddings from diverse sources. A basic API call with the text input, that returns the generated vectors.
3. The generated vectors / embeddings are stored in a vector database. Same as before, we will explore several database options in Azure, later in Chapter 3.
4. After the generation process, we can assume end users will want to search for specific topics or information that will be included as part of the different data inputs we have collected and vectorized. For that purpose, we will use the same Embeddings API to generate the embeddings of the questions itself (note: we need the same embedding model for both knowledge and questions).
5. The vector database will support search functions. This means we will bring the vectorized user questions, and use it as input to find information from the vector database that contains our knowledge base.
6. If there are related topics, the search function will return a "Top K" variety of results that we can use to generate the answer (either by directly printing the results, or by passing them as input for a chat-based scenario).

The variety of embeddings use cases available in Azure OpenAI Service are:

Text similarity
: A set of models that provide embeddings that capture the semantic similarity of pieces of text. These models are useful for many tasks such as clustering, regression, anomaly detection, and visualization.

Text search
: A set of models that provide embeddings that enable semantic information retrieval over documents. These models are useful for tasks such as search, context relevance, and information retrieval.

Code search
: A set of models that provide embeddings that enable finding relevant code with a query in natural language. These models are useful for tasks such as code search and relevance.

At technical level, the recommended model option for embeddings with Azure OpenAI Service is called "Ada", which is an improved and more cost-effective model than its predecessors. This is pretty useful to increase the knowledge scope of Azure OpenAI, by consuming information from PDFs, websites, text files, etc.

As previously mentioned, the embeddings generation is based on a very simple API call/response dynamic, and the specific details on how to generate embeddings for a given source are available in the official documentation, as well as the specific context length limits (e.g. 8K tokens for Ada version 2). Concretely, generating embeddings is as simple as calling the embedding API with the desired text input you want to vectorize. For example, in Python:

```
import openai
openai.api_type = "azure"
openai.api_key = YOUR_API_KEY
openai.api_base = "https://YOUR_RESOURCE_NAME.openai.azure.com"
openai.api_version = "2023-05-15"

response = openai.Embedding.create(
    input="Your text string goes here",
    engine="YOUR_DEPLOYMENT_NAME"
)
embeddings = response['data'][0]['embedding']
print(embeddings)
```

We have completed the review of Azure OpenAI models and their capabilities. While I will cover the details of project examples and architectures in Chapter 3, the next section will explore general architecture building blocks for Azure OpenAI-enabled implementations, as well as general cloud infrastructure topics.

Architectural Elements for Generative AI Systems

Azure-based architectures rely on a series of interconnected services that can communicate with each other for a specific purpose. In this case, Azure OpenAI plays a crucial role to enable interactions between any customer-side application, but we rely on more building blocks to build our Generative AI solutions.

In Figure 2-15, you can see the main building blocks of an Azure OpenAI-enabled (simplified) architecture:

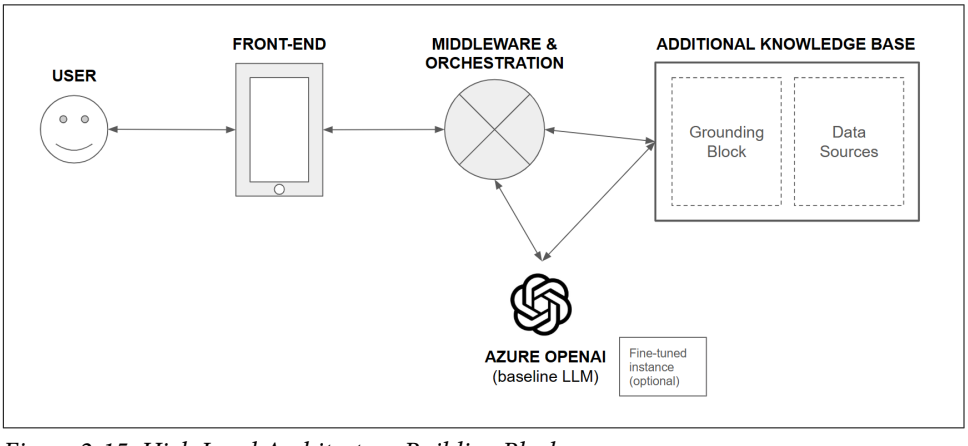

Figure 2-15. High Level Architecture Building Blocks

Application
: Any app-side front and back-end element that leverages the rest of Generative AI capabilities.

Middleware / Orchestration
: We will explore this element later in Chapter 3, but the orchestration piece basically allows us to connect different Azure OpenAI skills with other relevant services. Also, the middleware can include API management and other topics that we will see by the end of this Chapter.

Azure OpenAI Service
: For text-based skills, such as explaining the answer to a complex question, for both completion or chat-based scenarios.

Additional Knowledge Base
: This is a combination of the core data sources (databases, blob storage, etc.) and knowledge extraction elements such as Embeddings, Azure Cognitive Search, Bing Search, etc. For now, we will define them as "grounding blocks", but we will see the details in Chapter 3.

Now, if you develop an application that leverages Azure OpenAI and other Azure services, and that implementation is part of a bigger data/AI-enabled platform, the end-to-end architecture might start to look like something similar to Figure 2-16:

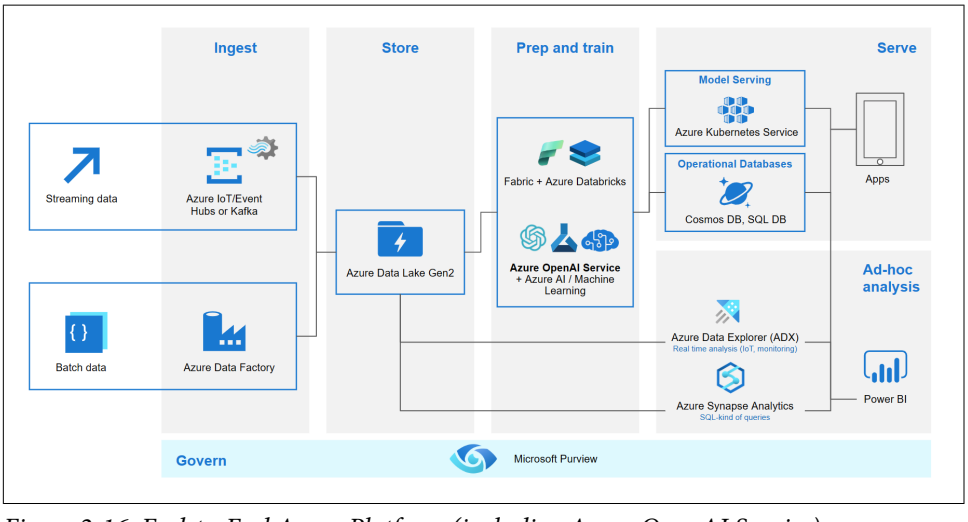

Figure 2-16. End-to-End Azure Platform (including Azure OpenAI Service)

In this case, Azure OpenAI Service is just part of a bigger end-to-end that includes data sources, integration processes, SQL/NoSQL databases, containerization, analytics, etc. The final setup depends on the structure of the platform itself, but this is a good overview to understand where Azure OpenAI sits for any data and AI implementation with Microsoft Azure.

If you want to learn more about Azure-enabled architectures and the details of all these cloud services, please check Learning Microsoft Azure by Jonah Carrio Andersson. Also, the main reference for architecture is the official Microsoft Architecture Center for specific Azure OpenAI scenarios. You may want to bookmark this resource as the Microsoft teams continuously update the content with new visual architectures and explanations, including some examples with Azure OpenAI Service.

Another interesting architecture you can explore is the Azure OpenAI Landing Zone reference architecture, which includes the end-to-end cloud considerations, including core infrastructure topics such as identity and security, monitorization, cost management, user and API management, FinOps, etc. A very rich and complete overview of what an enterprise-grade implementation would include, beyond the core Generative AI capabilities.

Before we finish Chapter 2, let's get some expert insights from one of the most recognized cloud native experts out there, former Google employee from the Kubernetes

team, and currently the Corporate Vice President of Open Source Cloud Native at Microsoft, Dr. Brendan Burns.

Conclusion

As you can see, cloud native architectures are really valuable for Generative AI development, as they seamlessly integrate with the Azure OpenAI and other Azure services.

We will explore different implementation approaches in Chapter 3, but all of them rely on the capabilities and key building blocks we have seen here. As an adopter, you may face situations where you will need to optimize existing applications so they can incorporate Generative AI capabilities (as we have seen in the modernization section), but you will also have the opportunity to develop new Azure OpenAI-enabled applications from scratch. In this case, leveraging containerization, serverless, and PaaS pieces will help you design well architectured and scalable architectures and solutions.

Depending on your current level of knowledge, it will be important for you to understand the cloud fundamentals behind Microsoft Azure, and specific services for development, APIs, and Kubernetes container orchestration.

Let's continue with Chapter 3, which will focus on different alternatives for enhancing your Azure OpenAI applications with specific company knowledge, as well as the main features and interfaces that you will leverage for your next projects. It also includes new terms that we have briefly explored in Chapter 2, such as vector database and orchestration. One more step, let's continue.

CHAPTER 3

Implementing Cloud-Native Generative AI with Azure OpenAI

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be Chapter 3 of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *mpotter@oreilly.com*.

This chapter will focus on the implementation of Generative AI architectures with Microsoft Azure and Azure OpenAI models, always aiming to present all available options, and minimize the required development, integration, and usage cost, while accelerating the operationalization. For that purpose, I've included a series of best practices and typical architectures that will allow you to choose the best building blocks for your specific scenarios.

Concretely, we will include the most relevant Azure OpenAI implementation approaches, based on existing features and repositories which will continue evolving, improving, and including new functionalities. I've included URLs to the original documentation because they are continuously updated with new features, so these links will allow you to explore any detail you need. Most of them rely on official accelerators from Github repositories, and projects that you will be able to follow and/or fork.

But before getting into the details, let's explore some fundamental topics that will help you understand the full extent of what a Generative AI with Azure OpenAI Service means.

Defining the Knowledge Scope of Azure OpenAI-enabled Apps

Generative AI applications on Microsoft Azure are not only for regular ChatGPT-kind of applications. They are advanced architectures that rely on diverse technology pieces, including the core infrastructure (e.g. servers, graphic processing units or GPUs) that is required to run all these last generation AI models, and that allow adopters to create conversational applications, search engines, develop and integrate new AI copilots into their applications, customize customer attention, etc.

From an Azure OpenAI point of view, we are talking about a managed service that includes advanced functionalities that will allow you to implement *different levels of knowledge*, depending on the desired scope of your applications, and based on by default-capabilities and specific adjustment and customization techniques. By levels of knowledge we mean something that goes beyond the initial scope of the LLM and its massive dataset (e.g. adding new information for an internal company application, based on their own data). Some of the options to adjust that knowledge include:

Baseline LLM

Azure OpenAI's language models are trained on enormous data sets containing billions of sentences. These data sets are carefully curated to include a wide range of topics, genres, and writing styles. The size and diversity of the training data help the models develop a broad understanding of human language. The specific details of the training data are unknown, but it includes text data from a variety of sources, including books, articles, websites, and other publicly available written material. Additionally, the training process includes human reviewers who help annotate and curate the data, flagging and addressing potential biases or problematic content. Feedback loops with reviewers are established to continuously improve and refine the models. The end-to-end process is explained in OpenAI's public paper called "Training language models to follow instructions with human feedback", and their official GPT model card, shown in Figure 3-1.

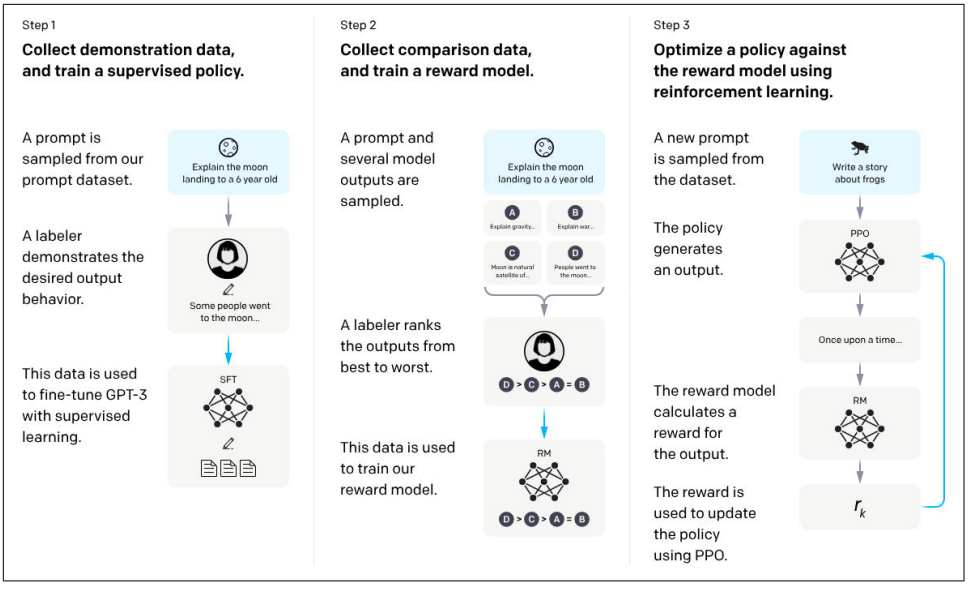

Figure 3-1. The ChatGPT Training Process (source: OpenAI, Creative Commons 4.0 license)

Additional knowledge

You can provide the LLMs with some additional context or knowledge, making them specific to the activity scope of the developed system. This could go from setting the topic of discussion for a chatbot, to specifying URLs that are related to the topics we want to include. There are different ways to implement grounding:

Fine tuning

Using small knowledge bases or private data to retrain the LLM with new additional information. Available via Azure OpenAI Service, it's a good option to augment the knowledge scope of the LLM, but a less cost-efficient option (as we will explore in Chapter 5 when we calculate the cost of the Azure OpenAI-enabled implementations).

Grounding, Embeddings-based retrieval

Based on Microsoft's definition, embeddings are "representations or encodings of tokens, such as sentences, paragraphs, or documents, in a high-dimensional vector space, where each dimension corresponds to a learned feature or attribute of the language. Embeddings are the way that the model captures and stores the meaning and the relationships of the language, and the way that the model compares and contrasts different tokens or units of language. Embeddings are the bridge between the discrete and the continuous, and between the symbolic and the numeric, aspects of language for the model".

Grounding, Indexation-based retrieval

The ability to index existing files so we can locate them when interacting with the LLM engine. Microsoft defines indexes as "crawlers that extract searchable content from data sources and populates a search index using field-to-field mappings between source data and a search index. This approach is sometimes referred to as a 'pull model' because the search service pulls data in without you having to write any code that adds data to an index".

Other grounding techniques

Other techniques include contextualization (providing information about topics and/or specific URLs to define a reduced knowledge scope), and live internet results to complement the LLM information and to include external sources.

As you can see in Figure 3-2, all these elements contribute to the creation of an extended knowledge domain from regular LLMs, based on internet and private data. The rest of this chapter will focus on different techniques to implement them with Azure OpenAI and other Microsoft services:

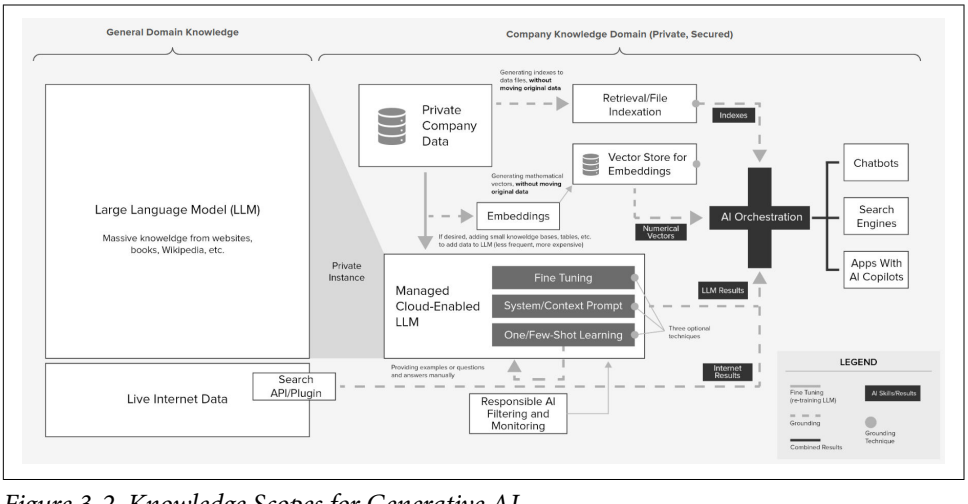

Figure 3-2. Knowledge Scopes for Generative AI

Summarizing, any Generative AI architecture or approach will depend on the knowledge domains and levels we require for the end solutions. If our application can rely on (just) the LLM, which already contains a massive level of information, then we can implement the model with no additional building blocks. On the other hand, if we need to add specific information from other sources (including PDFs, text documents, websites, databases, etc.), then we will leverage the so-called fine tuning and grounding techniques.

Let's now explore the available interfaces and tools for you to create new applications with Azure OpenAI Service. You will understand the key building blocks before moving into the step-by-step guide of the most relevant implementation approaches.

Generative AI Modelling with Azure OpenAI

One of the key adoption factors that enable different types of people to use Azure OpenAI is the availability of different visual and code-based interfaces that you can leverage while using the service. In this section we will explore them, and how to use these interfaces depending on your Generative AI implementation approach.

Initially, Microsoft released Azure OpenAI Service as a "Gated General Availability", meaning that any organization willing to use the service had to *complete a detailed application form* to explain the potential use cases and guarantee a good usage of the platform. Microsoft's goal was to validate that any application enabled by Azure OpenAI is always aligned with their Responsible AI approach and the intended use of the platform. If you are getting started with Azure OpenAI Service, check first if you still need to apply for access and prepare the required information for the application form.

Azure OpenAI Service Building Blocks

Before diving on the "how to" part, let's explore the available building blocks for any Azure OpenAI practitioner to prepare and deploy new solutions.

Essentially, there are two main types of components for the Azure OpenAI Service: the *visual interfaces* that allow users to test, customize, and deploy their generative AI models, and the *development interfaces* that enable the exploitation and integration of those advanced capabilities with any application.

Both elements are complementary, and great assets for any kind of adopter, as they require a relatively low level of AI knowledge to make them work. For example, *citizen users* (hybrid technical-business profiles that are not very technical, but that understand the principles of Generative AI, and have some knowledge of prompt engineering) and *regular developers* are great candidates for these platforms.

Visual Interfaces: Azure OpenAI Studio and Playground

As any other Azure AI service, Azure OpenAI includes the notion of a "Studio" (i.e. the Azure OpenAI Studio) that makes the interaction with the Generative AI models very simple, by providing an intuitive user interface (UI) that facilitate service deployments, and leverage existing Azure OpenAI APIs without no code required from the user perspective.

The Azure OpenAI Studio includes the access to all available models, pre-defined prompting scenarios and examples, and several applications called *Playgrounds*. The Azure OpenAI Playgrounds are different apps within the Azure OpenAI service, which include (as you can see in Figure 3-3) a customizable ChatGPT-kind of instance (*Chat*), other GPT language models for non-chat scenarios (*Completion*), a third playground to connect the AI models with your data (*Bring your own data*), and the last one for image generation applications with OpenAI's *DALL·E* models.

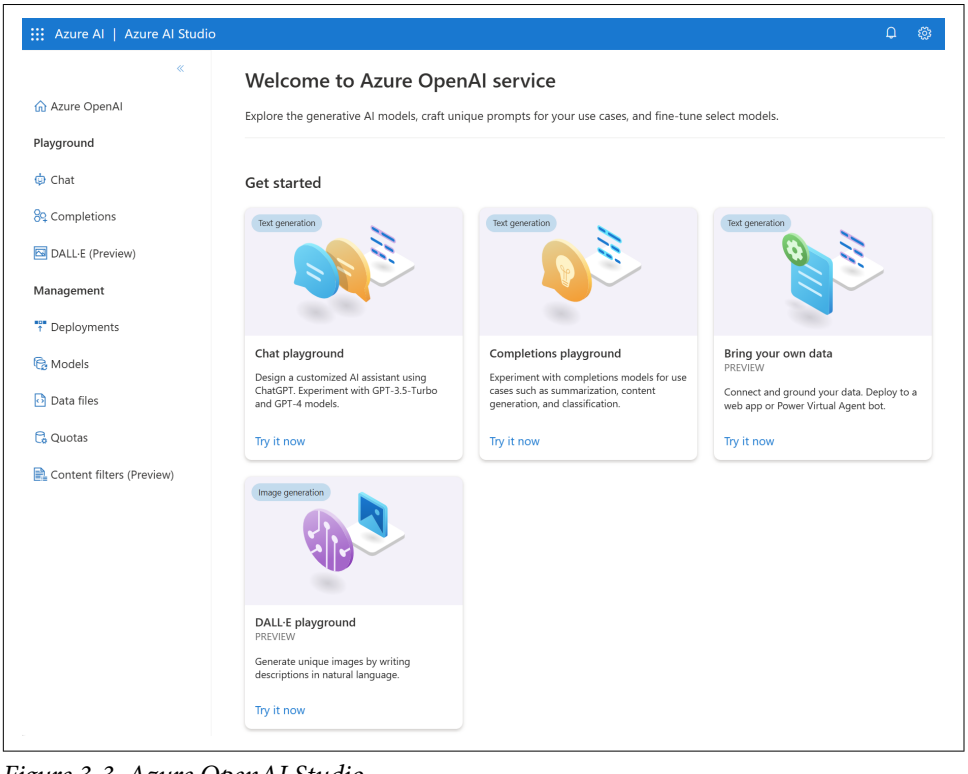

Figure 3-3. Azure OpenAI Studio

You can access each playground (and their related management features) from the left panel of the Studio, or visit them directly by following URLs linked in the following sections.

Playground #1. It includes both the *conversational "Chat" playground* with the features and settings required to create a private ChatGPT implementation, and the *Bring your own data* (represented as the 4th playground in Azure OpenAI Studio) functionality that I will explain later in this same section. Figure 3-4 shows the main tiles of the Chat playground:

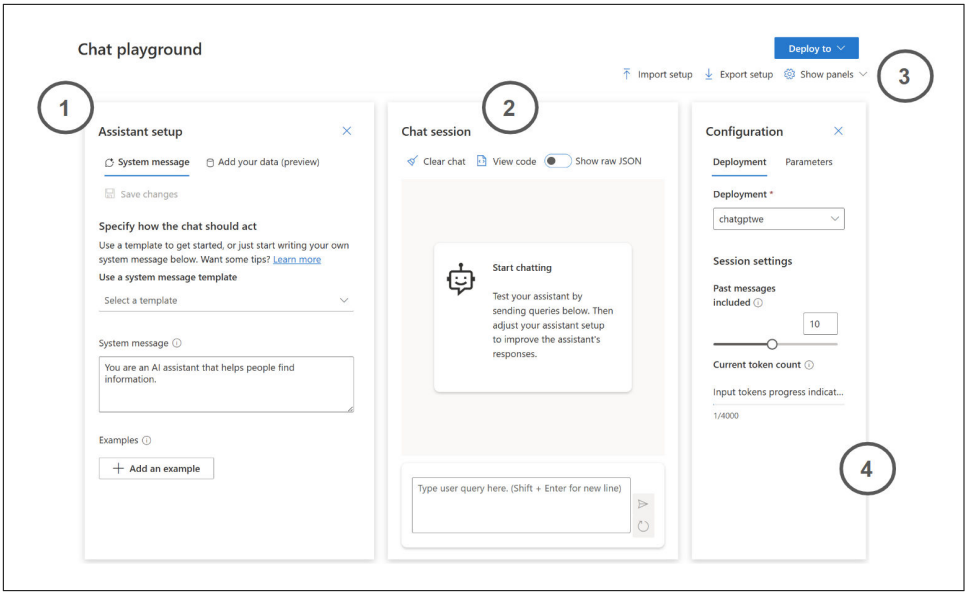

Figure 3-4. Azure OpenAI Studio - Chat Playground

1. *Assistant Setup* - This area is located on the left side of the screen and allows users to configure the chatbot's behavior. Users can choose from templates or create their own custom system messages. This section helps users define how the chatbot should act and respond to user queries.

   System message

   A type of meta-prompt (i.e. a prompt that sets the by-default context of the discussion) to guide the AI system's behavior. It can be used to introduce the system, set expectations, provide feedback, or handle errors. One important thing to remember is that even if there is no token limit for this message, it will be included with every API call, so it counts against the overall token limit / context length of the model.

   Examples

   This area is located at the bottom-left corner of the screen. You can add examples to the bot intelligence, so it learns the proper way to answer specific questions. It is a good option when we don't need to fully retrain a model. For example, when you need to add a couple of topics from your company's knowledge base, and you want to define the best way to answer. From the official description: *"Add examples to show the chat what responses you want. It will try to mimic any responses you add here so make sure they match the rules you laid out in the system message"*.

2. *Chat Session* - This area is located in the middle of the screen and serves as the main interaction point between you and the chatbot. You can type your queries here, and the chatbot will respond accordingly. The chat session allows you to test the chatbot's performance and make adjustments to the assistant setup as needed, as well as importing and exporting bot configurations, or getting the result as a JavaScript Object Notation (JSON) file.
3. *Deploy to* - This option allows you to deploy your chatbot to a specific platform or environment. Concretely, Azure OpenAI Studio allows direct deployments to both Azure WebApps, and Power Virtual Agents (from Microsoft's Power Platform suite). We will explore these deployment options later in this chapter.
4. *Configuration* - This area is located in the top-right corner of the screen. It provides options for you to access deployment and session settings. Users can also clear the chat history and manage parameters related to the chatbot's deployment.
   - *Deployment* - To handle session-level configurations, such as the Azure OpenAI deployment resource you want to use (e.g. you may have several for different geographic regions), as well as the memory of the session, which will impact on how many interactions the system can remember when getting new questions.

     Deployment instance
     : You will select one option, from the resources you have previously deployed (if you haven't, you will need to create one before using the Azure OpenAI Studio), based on the geography and model needs you may have.

     Past messages included and current token count
     : Session-level parameters you may want to adjust for the specific test you do via the chat playground. These parameters will be gone when you finish the playground session, except if you deploy an application (we will see the deployment options in a couple of sections).

   - *Parameters* - This right panel includes all technical settings that will allow you to configure the expected output message, including the level of creativity versus determinism of the answer.

     Max response
     : This parameter helps you set a limit on the number of tokens per model response. The max response is measured in the number of tokens, and it is shared between the question (including system message, examples, message history, and prompt / user query) and the model's response.

Temperature

This parameter and the "Top P" parameter are direct alternatives to control the AI models randomness. Lowering the temperature means that the model will produce more repetitive and deterministic responses. Increasing the temperature will result in more unexpected or creative responses. Try adjusting temperature or Top P, but not both.

Playground #2 . Completions: As you saw in Chapter 1, the completion skill is (along with chat and embeddings models) one of the classics for NLP and modern LLMs. Completion focuses on unitary interactions for all kinds of text-based requests (with no need for memory between interactions, as you may need for chat-based applications in which the model keeps the discussion context). As shown in Figure 3-5, the Completions playground allows you to type a prompt, or choose from a series of examples. It also includes the same kind of setting parameters you have seen in the Chat playground.

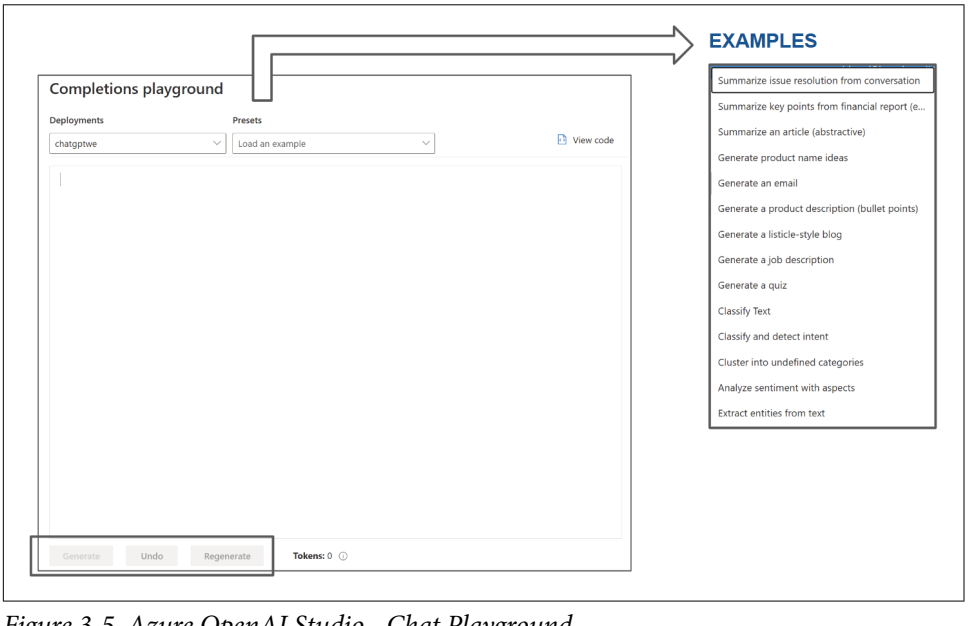

Figure 3-5. Azure OpenAI Studio - Chat Playground

You can generate an answer (completion), and even regenerate it to obtain a totally new output. If you choose one of the examples from the drop down menu, you will see an automatic prompt appear, and the corresponding completion, highlighted as you can see in Figure 3-6:

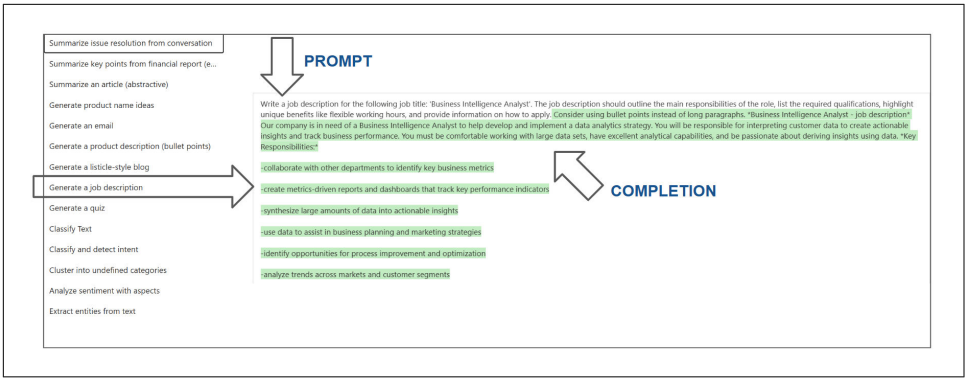

Figure 3-6. Azure OpenAI Studio - Chat Playground (example)

Summarizing, you may use chat for multi-step scenarios where you need to maintain a sequence of interactions with the AI model, while completion can be used for specific unitary cases. As you will see later, these two playgrounds are just visual interfaces that consume existing Azure OpenAI completion and chat APIs.

Playground #3 . *Bring your own data:* Even if the Azure OpenAI Studio shows this feature as a separate playground, it is technically part of the Chat playground I have previously mentioned. In order to access this functionally, as you see in Figure 3-7 you can either use the Chat playground's assistant setup and select the "add your data" tab, or go directly to the "bring your own data" tile of the Studio. For both cases, the result will be the same.

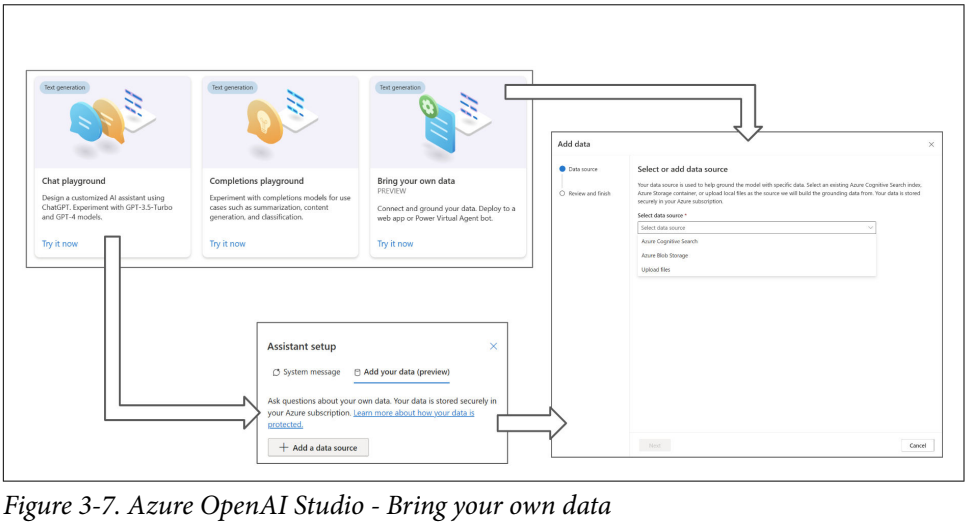

Figure 3-7. Azure OpenAI Studio - Bring your own data

Once you reach this point, the sequence of steps is pretty simple. As you can see in Figure 3-8, the system will allow you to select your own sources of data, to combine their knowledge with the baseline LLM. Concretely, that knowledge can come from PDF files, text-based documents, slides, web files, etc. In this case, besides the Azure OpenAI resource previously deployed, the Bring your own data functionality will leverage other resources such as Azure Data Lake Gen2 or Storage, to save the files, and Azure Cognitive Search, to index the files. This last one offers a vector search functionality based on embeddings, that I will explain by the end of the chapter. Last but not least, you can always check the official documentation to follow the latest updates for this Azure OpenAI feature, as it is a very evolving one due to the continuous incorporation of new functionalities.

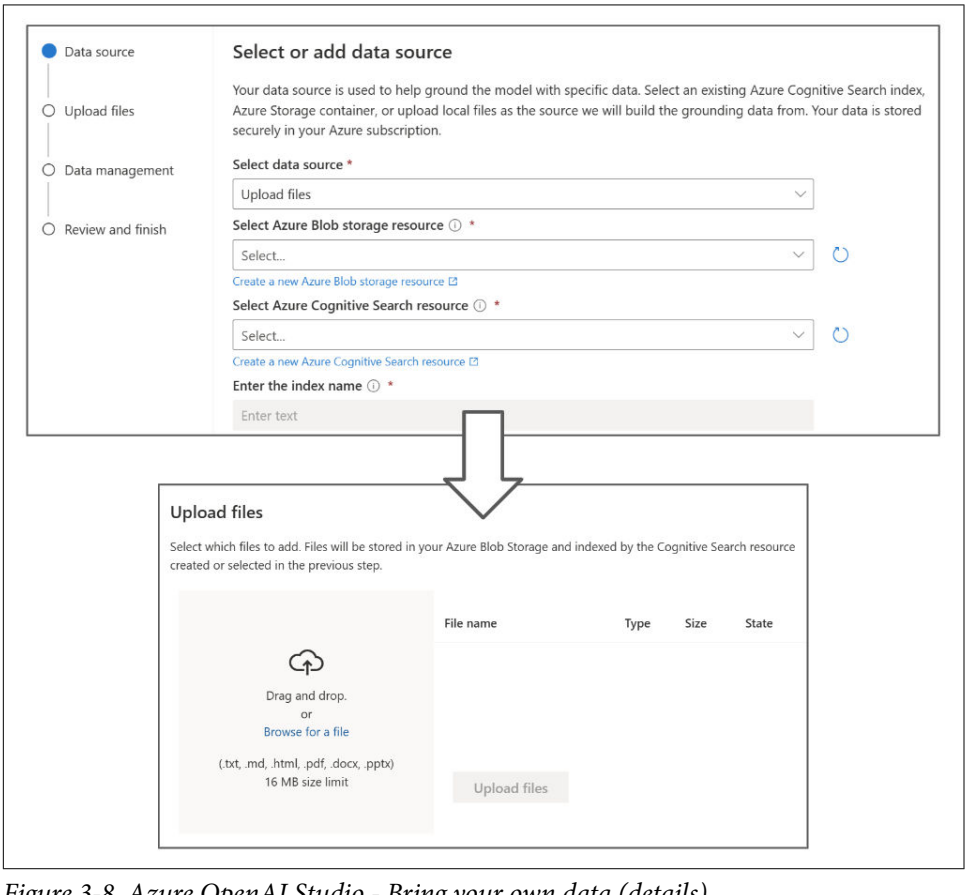

Figure 3-8. Azure OpenAI Studio - Bring your own data (details)

Playground #4 . DALL·E: The last playground tile provides direct access to the Generative AI DALL·E models (versions 2 and 3), from OpenAI. This one is a text-to-image

model that allows you to create new images from just text-based descriptions. Imagine describing a place or a scene, and getting a visual representation in the form of images that are freshly created on demand. This means they didn't exist previously, and that you can integrate this capability into your solutions, and combine it with the rest of language models. Figure 3-9 shows all relevant aspects of the playground.

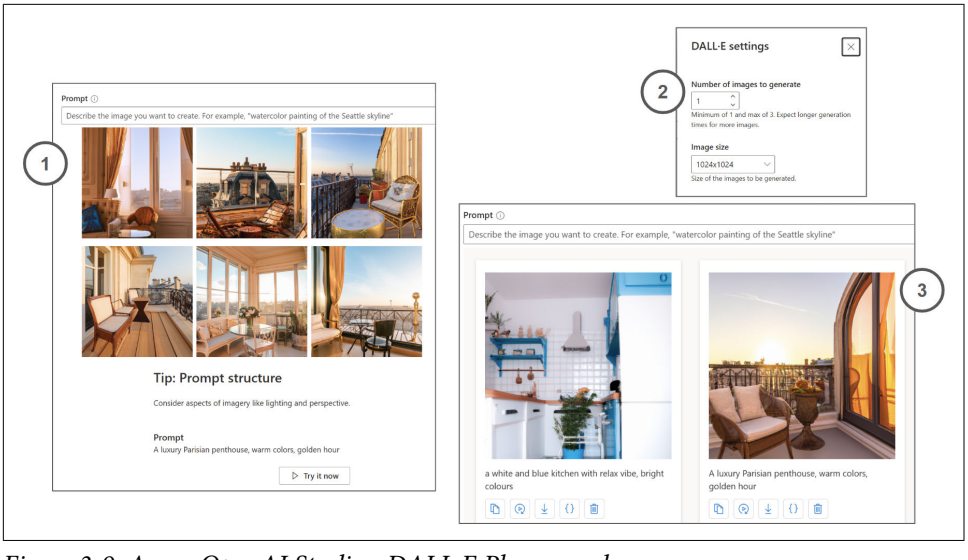

Figure 3-9. Azure OpenAI Studio - DALL·E Playground

1. Playground

The DALL·E playground is visually simple—a prompt field and the results (image) below. It's similar to the structure of the Bing Create application, but with the option to deploy the DALL·E model for your own developments.

2. Settings

The settings panel offers you the option to choose the number of images you want to generate, and the image size.

3. Album

The album section showcases all past image experiments, offering you the option to receive the previously created images, generate new ones, etc.

Besides the different playgrounds, you can also explore the left-side *Management panels* in Figure 3-10, that include options such as deployments, models, data files, quotas, and content filters. Let's explore the most important features:

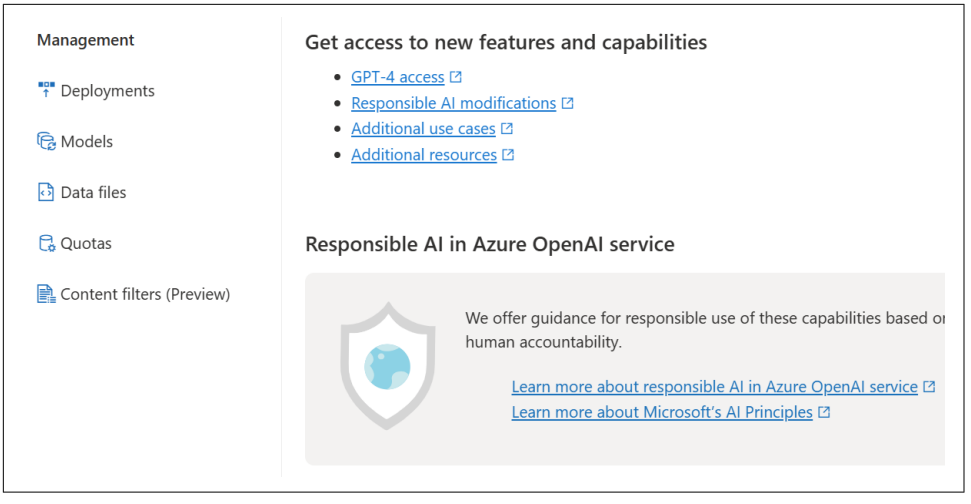

Figure 3-10. Azure OpenAI Studio - Management

Deployments

Allows you (as shown in Figure 3-11) to deploy specific model instances for any Azure OpenAI model available within your chosen geographic region, and to visualize those previously deployed.

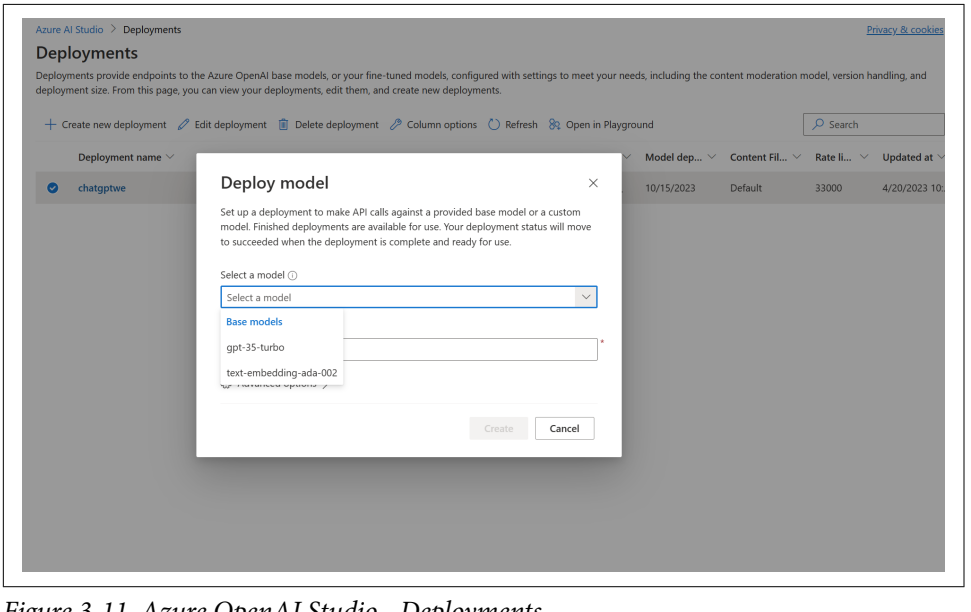

Figure 3-11. Azure OpenAI Studio - Deployments

Content filters, for Responsible AI moderation

Each filter from those in Figure 3-12 (e.g. hate, sexual, self-harm, and violence topics for both prompts and completions, with different levels of filtering) can be applied to the "deployments", and those deployments will include the content filter for each chat or completion implementation. We will explore this feature in Chapter 4, as part of the Responsible AI measures for Generative AI implementations.

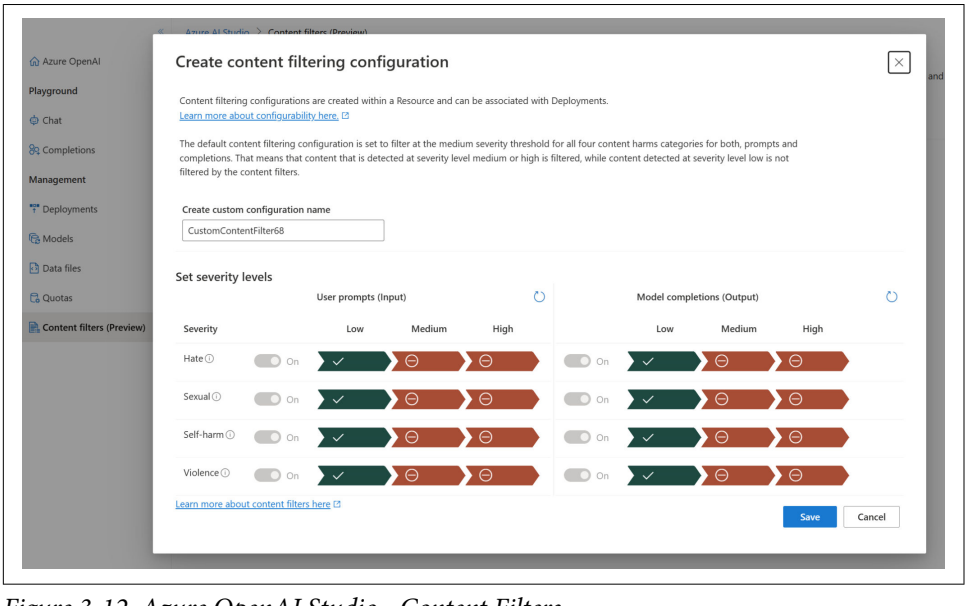

Figure 3-12. Azure OpenAI Studio - Content Filters

Models

This option shows the available Azure OpenAI models, related to the specific geographic region of the chosen deployment.

Data files

This file management feature allows you to prepare the dataset for fine tuned implementations. We will explore what fine tuning is, later in this chapter.

Quotas

The quota panel shows the usage quotas related to different models and geographic regions. It also helps you request a quota increase if you need more.

We will explore some of these functionalities later in this chapter and in Chapter 4, as they will all be relevant, depending on the type of Azure OpenAI implementation you plan. Now, let's see what you can do to deploy these models via Azure OpenAI Studio.

Deployment Interfaces: WebApps and Power Virtual Agents

As I have previously mentioned in this chapter, the Chat Playground includes some easy-to-use deployment options. They are not available for the rest of playgrounds, but they can simplify the preliminary deployment of Azure OpenAI models for internal testing and use purposes, without any coding required. These no code deployments can incorporate the specific knowledge from the Bring your own data functionality. Concretely, there are two possibilities:

Web Apps with Azure AppService
: The first available deployment option, that you can use with or without the "bring your own data" feature activated. As you have seen in Chapter 2, AppService is the Azure option to deploy native webapps, it allows integrations with both external and internal systems, and web development with a variety of programming languages. From the Azure OpenAI Studio and its Chat playground, you can simply "Deploy to", then choose the options you can see in Figure 3-13:

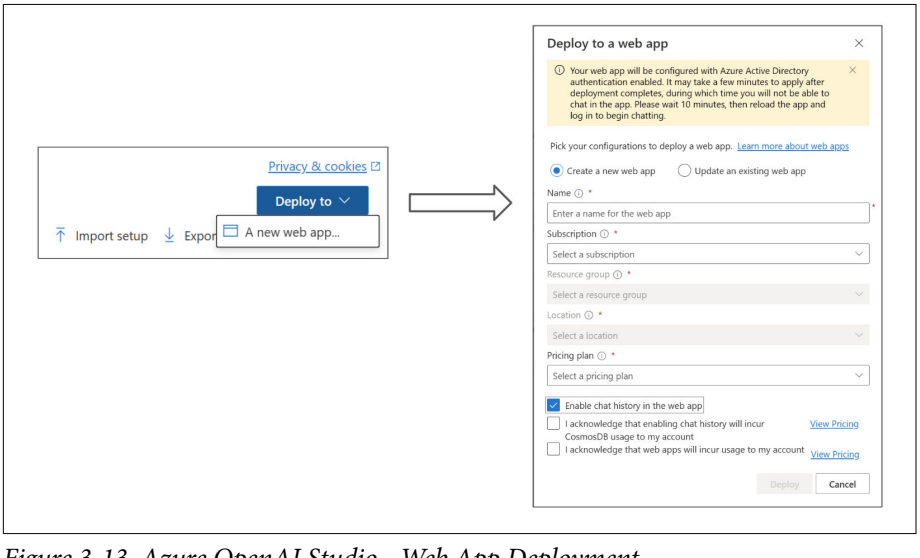

Figure 3-13. Azure OpenAI Studio - Web App Deployment

Choosing the web app
: You can create a new AppService resource directly from this feature (in that case, you will need to define the "app name" that will be part of your web app URL), or choose an existing one if you have previously deployed via Azure Portal's App Service panel.

Pricing plan
: To select the preferred pricing tier for the AppService web app.

Chat history

A functionality that allows the web app users to recover their previous interactions with the chat. It relies on CosmosDB (Azure's NoSQL database), which obviously adds cost to the existing Azure OpenAI and AppService resources.

Once you have selected all these options, you can click on "Deploy". You will need to wait around 10 minutes for all the resources to be deployed, then you will be able to launch your web app from the studio, or by typing the URL ***https://<appname>.azurewebsites.net***. The look and feel will be something like the interface you see in Figure 3-14.

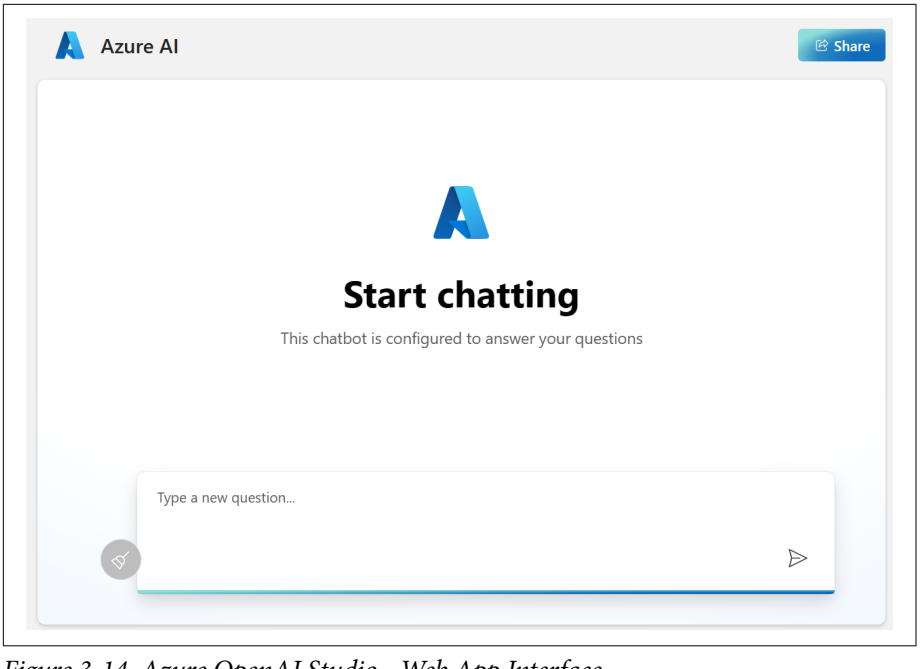

Figure 3-14. Azure OpenAI Studio - Web App Interface

The user interface of the new app will contain a regular chatbot setup, with options to share and to check previous discussions in the top-right side of the window. You can also customize the visual aspect of the application by using the official source code, and deploying it by hand instead of using the Azure OpenAI Studio.

Bots with Power Virtual Agents (PVA)

This option is available for Chat playground implementations that include the "bring your own data" feature. That means, if you don't add the extended knowl-

edge from PDFs or other documents, the Chat playground won't include Power Virtual Agents as a deployment option in the top-right corner in Figure 3-15.

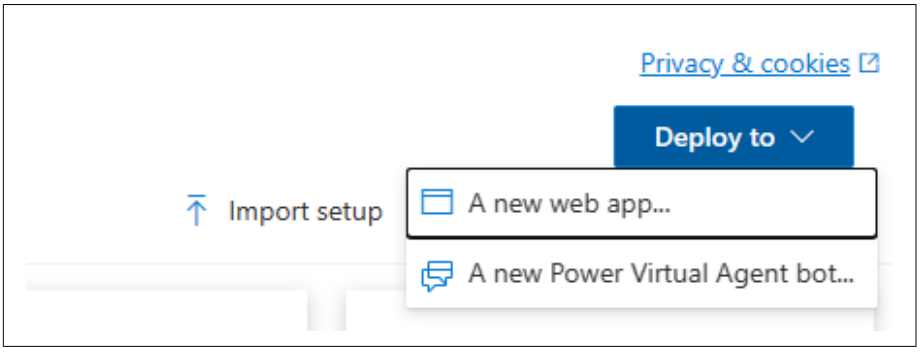

Figure 3-15. Azure OpenAI Studio - PVA Deployment

How to handle Power Virtual Agents is outside of the scope of this book, but you can explore the detailed instructions from the official documentation, that shows how to use PVA with Azure OpenAI for the *generative answers* feature. This option is available only for certain geographic regions, so you will need to validate if your deployments with Azure OpenAI models show the PVA deployment option in the Chat playground. If it is not the case, you may want to deploy new models in other regions.

Summarizing, these visual interfaces can help you leverage Azure OpenAI models in a simple manner. They provide an intuitive way to launch the Azure OpenAI APIs in just a few clicks. However, you will need code-based tools to implement other advanced architectures you will see later in Chapter 3. Let's now explore those API and other development kits for you to leverage everything that the Azure OpenAI Service has to offer.

Development Interfaces: APIs and SDKs

In addition to all the previously explored interfaces, one of the key enablers for integrating Azure OpenAI with existing or new applications is the ability to consume the pre-configured models as regular endpoints. From a development point of view, we can call those models by using the application programming interfaces (APIs) and related software development kits (SDK) and pass any input and configuration parameters within the code. This section covers the main pieces you need to know—the *Azure OpenAI Service REST APIs*, that include the official API reference documentation, with specific details for preview and stable model deployments. There is also an official repo with the full swagger specifications. There are general APIs that will help you with the configuration and deployment of Azure OpenAI Services, while the service APIs help you consume the models to bring the AI capabilities to your Generative AI applications.

The main APIs you need to know, and their high level call details are:

General Management APIs
: For Azure AI service account management (including Azure OpenAI), with tasks such as account creation, deletion, listing, etc.

APIs for model-related information
: To obtain the list of available Azure OpenAI, and information about the specific model capabilities, and the model lifecycle (including potential deprecation details).

Completions
: The required API for non-chat, language scenarios. This and other APIs are versioned by using the "YYYY-MM-DD" date structure as `api-version`, and you will need to copy the `resource name` and `deployment ID`, from the Azure OpenAI model you have previously deployed (remember the step-by-step process from the Azure Portal, in chapter 2). To create a completion resource, the POST operation is:

```
POST https://{your-resource-name}.openai.azure.com/openai/deployments/{deployment-id}/com
```

The request and response dynamic follows this structure, with the *prompt* parameter being the input for the model to generate a specific completion, and a series of optional parameters such as the `max_tokens length` (the limit of tokens for the expected answer) or the *number* "`n`" of expected completions/answers:

```
Request:
curl https://YOUR_RESOURCE_NAME.openai.azure.com/openai/deployments/YOUR_DEPLOYMENT_NAME/
  -H "Content-Type: application/json" \
  -H "api-key: YOUR_API_KEY" \
  -d "{
  \"prompt\": \"The best thing in life\",
\"max_tokens\": 5,
\"n\": 1
}"

Response:
{
    "id": "cmpl-4kGh7iXtjW4lc9eGhff6Hp8C7btdQ",
    "object": "text_completion",
    "created": 1646932609,
    "model": "gpt-35-turbo-instruct",
    "choices": [
        {
            "text": ", is eating burgers with a milkshake",
            "index": 0,
            "logprobs": null,
            "finish_reason": "length"
        }
```

```
        ]
    }
```

The answers (completions) contain the `finish_reason` parameter. The `finish_reason` defines why the model stopped generating more information, and most of the cases that will be due to the max_tokens, which is used by the model to stop once it reaches the limit. However, there is another option that we will explore in Chapter 4, which allows the model to stop due to what we call *content filters*.

Chat completions

Dedicated API for chat scenarios, including the configuration parameters we have previously seen on the Chat playground. It includes input parameters we have previously seen from the Azure OpenAI Playground, such as `temperature` or `max_tokens`. There is one important parameter for the chat messages, known as the Chat Role. This allows you to split the interactions based on different roles:

- *System* - It helps you set the behavior of the assistant
- *User* - It provides input for chat completions.
- *Assistant* - It provides responses to system-instructed, user-prompted input
- *Function* - It gives function results for chat completions. We will explore this concept later in this chapter, after we cover the different Azure OpenAI APIs.

The sequence for a typical chat scenario follows these steps:

1. *Resource creation*, with a similar structure to what you have seen in the regular completion API call (including the date as API version). The regular POST operation for chat completion is:

```
POST https://{your-resource-name}.openai.azure.com/openai/deployments/{deployment-id}
```

2. *System message*, which is your way to set the context of the chat engine, by defining the scope of the discussion, allowed or forbidden topics, etc. The system message is also called context prompt or *meta-prompt*. The messages parameter, along with the role sub-parameter, is the place where you will define your system message, by using:

```
{"messages":[{"role": "system", "content": "the specific context and system message y
```

3. *User-assistant interaction*, which leverages the same *messages* parameter, with the *user* and *assistant* roles. The structure for both roles is similar to the one you have already seen for the system message, and the response includes the same *finish-reason* parameter that will give you a hint about the result (i.e. if the completion has finish due to the max_lenght assigned to the answer, or if there is a filtering reason due to negative topic detection).

Image Generation

The API call to generate images based on text-to-image DALL·E models. Same as the visual playground, the input parameters include the text-based prompt, and two optional inputs such as the number "n" of desired images (if you don't include it, the system will generate only one image), and the size (by default 1024x1024, with alternative 256x256 and 512x512 options). The POST operation to create the image generation resource is:

```
POST https://{your-resource-name}.openai.azure.com/openai/images/generations:submit?api-v
```

The end-to-end process includes three different steps:

1. *Request* the image generation (via POST operation), which helps you pre-generate the images based on the text-based input prompt. It returns an operation ID that you will leverage for the next step.
2. *Get* the result of the image generation (GET operation), which allows you to recover the pre-generated images for the specific operation ID.
3. *Delete* the previously loaded images (DELETE operation) from the server, for the specific Azure OpenAI resource, and the existing operation ID. If you don't use this option, the images will be automatically deleted after 24 hours.

Speech to Text

Based on the Azure OpenAI Whisper model, these APIs will allow you create transcriptions from audio pieces, for a variety of languages and accents, with a great performance and the possibility to combine it with the rest of Azure OpenAI models. You can specify the input audio file, language, discussion style, output format (by default a JSON file), etc. This Azure OpenAI Speech-to-text (S2T) feature has a limitation of 25 MB for the input audio file, but you can leverage the batch transcription mode of Azure AI Speech (not Azure OpenAI) to transcribe bigger files. The POST operation looks similar to the previous APIs:

```
POST https://{your-resource-name}.openai.azure.com/openai/deployments/{deployment-id}/aud
```

Embeddings

The API call that will allow you generate embeddings from specific text inputs, from some of the architectures you will see in this chapter. The model and its specific input length will depend on the model availability at the time of your implementation. The POST operation is very similar to the previous ones, and the dynamic is as simple as requesting the embeddings for a text input, and obtaining a JSON response with the generated embeddings, for you to store them (we will see several vector store / databases options by the end of the chapter) and leverage them later.

```
POST https://{your-resource-name}.openai.azure.com/openai/deployments/{deployment-id}/emb
```

Fine Tuning

As you have seen in the beginning of this chapter, one of the implementation options include the ability to fine tune the pre-built models with your specific, available information. We will see more details later in this chapter, but for now keep in mind that if you choose this option, there is a specific set of APIs that you can leverage to create, manage, explore, and delete new fine tuning "jobs". Also, you will handle your own input files for the fine tuned models.

Other relevant APIs:

- Bing Search: The Bing Search API enables you to leverage Microsoft Bing's search engine for your own developments. In this case, we will see how you can extend the capabilities of your Azure OpenAI-enabled implementations, with live search functionalities.
- Form Recognizer (known as Azure AI Document Intelligence): helps you extract information from forms and images into structured data. It includes advanced OCR functionalities that will support your Azure OpenAI developments with specific data sources such as PDF or DOC files.

In addition to these APIs, there is an *Azure OpenAI* library for .NET developers, which essentially replicates the features of the official API, for a .net development environment. It provides an interface with the rest of the Azure SDK ecosystem, and it facilitates the connection to Azure OpenAI resources or to the non-Azure OpenAI inference endpoints.

This set of visual and development interfaces are your toolkit for most of the Azure OpenAI implementations out there. They are highly evolving, but the links to the official documentation will help you access updated information, any time. Now, before moving to the implementation approaches, let's take a look at a powerful feature that will enable your Generative AI systems to interact with other external APIs. It is called "Function Calling".

Interoperability Features: Function Calling and "JSONization"

The Azure OpenAI function calling option is a way to leverage language models to generate API calls and structure data outputs, based on a specific target format. Technically, it is one of the options within the chat completion API you have seen before—concretely, the Function chat role. You can see several samples on how to use this functionality, but it essentially relies on the following steps:

Calling the chat completions API, including the functions (based on the official FunctionDefinition format) and the user's input.

Using the model's chat response to call your API or function.

Calling the chat completions API again, including the response from your function to get a final response.

This is a relatively new functionality, so you can expect some feature improvements over the time. You can always check the official documentation to get the latest details and advice.

This completes the first part of Chapter 3. You have learned about the knowledge domains and how to leverage different building blocks to improve and increase the level of knowledge of your Generative AI solutions, and the availability tools you will use for the implementations. Now, we will move to the core part of this chapter, in which we will explore some of the most relevant development approaches, based on the industry's best practices. Let's get started.

Potential Implementation Approaches

There are several ways to implement generative AI applications with Azure OpenAI Service. The type of implementations you use will mostly depend on your specific use case needs, as well as the technical and financial context for adoption. This means, there are situations where the most expensive option is not always the best, or other options may have limitations such as when we don't have specific data besides our website, etc. Let's explore the main kind of implementations, based on the customization levels of Figure 3-16:

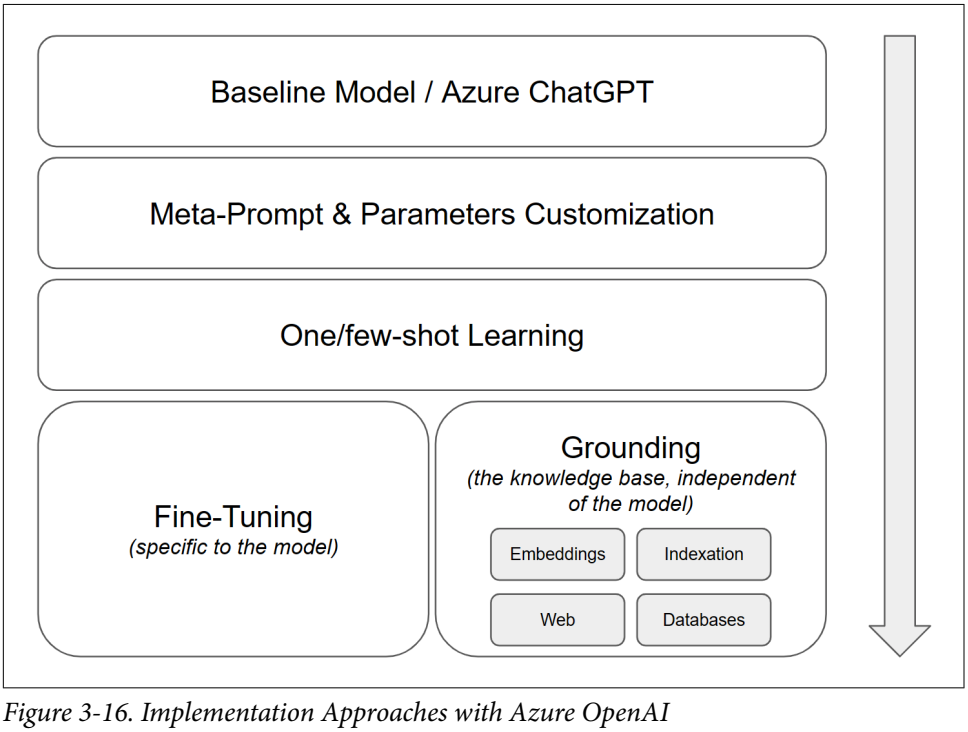

Figure 3-16. Implementation Approaches with Azure OpenAI

As you can see from the figure, you can customize a model by preparing a good meta-prompt, adjusting technical parameters, providing one or few "shots" as examples to guide the model, and implementing fine-tuning and/or grounding techniques. The next sections will go into the details of how to do all of this.

Basic Azure ChatGPT instance

A basic private GPT-kind of instance is the simplest kind of implementation, and one of the most popular Azure OpenAI cases nowadays. When companies want to have a private "ChatGPT" for their employees, this is the answer. It keeps your own data safe and private, and deploys the instance within your own cloud infrastructure. It's one of the favorite options for internal use with employees.

The deployment process is relatively simple:

- Within your Azure OpenAI Studio, deploy a GPT-3.5 turbo, GPT-4, or GPT-4 turbo model instance. This type of model is technically similar to what ChatGPT is, and it will deliver that level of performance. Remember to choose the specific geographic region that is closer to you.
- Once you have created the resource, go to the visual Playground. There, you will see the left menu with the option "Chat".
- Once there, you can prepare the system message to contextualize the chatbot by telling it something like "You are an AI assistant for company X, to answer questions from the employees" (internal use) or "You are an AI assistant for company X with website Y. If anyone asks something that is not related to this topic, say you cannot answer" (for clients).
- You can also *customize parameters* such as the max length of the answers or the temperature of the messages, which is a metric between 0 and 1 to define the level of creativity of the model.
- Once you have *tested the performance* and you are ready to *deploy the model*, you can come back to the resource page (Azure Portal) and find both the endpoint and the keys for that specific resource. That page contains examples of code to know how to call the APIs.

The end-to-end architecture you can see in Figure 3-17 is pretty simple—a predeployed model that we can directly consume from our applications, based on the existing endpoints and APIs.

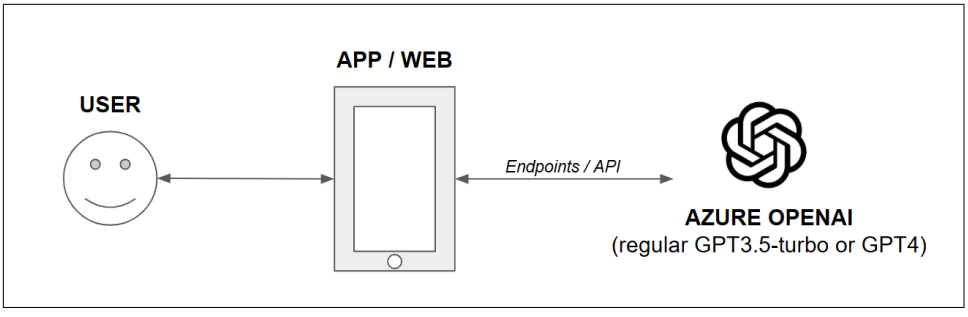

Figure 3-17. Simplified Azure ChatGPT Architecture

This type of implementation is good enough for those internal company cases where you don't require any customization based on private data. For example, internal chatbots for employee productivity based on general internet information, or search engines for intranet sites. For the rest of the cases where there is some custom data involved, we will explore other options. Let's dig into the first one of them next.

Minimal customization with one or few-shot learning

Besides the baseline model, and the system message / meta-prompt and parameters customization, there is an option to perform *one or few-shot learning*, which means providing the LLM with examples of discussions based on the expected output for a specific topic. This is a useful and simple option for small adjustments, and it relies on a very similar architecture to the previous one, with relatively light changes. The main difference, as you can see in Figure 3-18 when compared to the previous approach, is the inclusion of one or few examples to guide the LLM before starting using it.

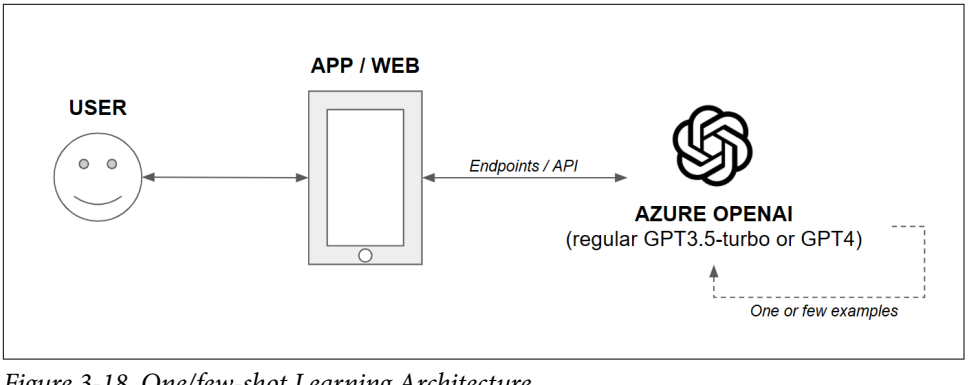

Figure 3-18. One/few-shot Learning Architecture

The one/few-shot learning process can be achieved in four different ways:

Via APIs (code)

Option 1 - Use the Chat Completion API with the ChatGPT (preview) or GPT-4 (preview) models, which are designed to take input formatted in a chat-like transcript. You can provide conversational examples that are used by the model for in-context learning.

Option 2 - Use the Completion API with the GPT-3 models, which can take a string of text with no specific format rules. You can provide a set of training examples as part of the prompt to give additional context to the model.

Via Playground (visual)

Option 3 - Use the Chat playground to interact with the ChatGPT (preview) or GPT-4 (preview) models. You can add few-shot examples in the chat transcript and see how the model responds.

Option 4 - Use the Completion playground to interact with the GPT-3 models. You can write your prompt with few-shot examples and see how the model completes it.

Overall, all these customizations are intended to improve the performance of the model vs a regular vanilla "ChatGPT" implementation like the one we have previously explored, but there are ways to retrain the model in a deeper way, like the next one we will explore now.

Fine-tuned GPT models

As previously mentioned earlier in the chapter, there are different ways to customize a LLM to expand its knowledge scope. Most of them rely on the orchestration / combination of the LLM with other knowledge pieces, without really combining the data sources (i.e. grounding). In this case, we will focus on the only way to "retrain" an Azure OpenAI model with custom company data: the Azure OpenAI Service fine-tuning feature.

This approach may have some advantages for companies with very specific and valuable data intellectual property, but its cost (you will need to add hosting cost to the regular API calls for the fine-tuning process) and technical complexity will probably lead you (and most of the adopters out there) to other kinds of grounding approaches with better performance-cost balance.

Also, the fine-tuning feature relies on a very special kind of training process. It is not the regular label-based training process you can do, for example, in classification tasks with traditional AI models. We are talking about a new kind of supervised process that leverages Azure OpenAI's prompting system to inject information based on the JSONL file format.

For GPT-3.5 turbo, you will leverage the system and user roles to re-educate the model:

```
{"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."},
```

Other legacy models such as Davinci require a prompt/completion format based on a question-answer logic:

```
{"prompt": "<prompt text>", "completion": "<ideal generated text>"}
{"prompt": "<prompt text>", "completion": "<ideal generated text>"}
```

This new way to inject data and knowledge allows to reeducate the model in a very granular manner, but it is a complex way to do so. You can see the overall architecture in Figure 3-19, in which you will basically customize the model, based on a fine-tuning process that relies on specific organization data:

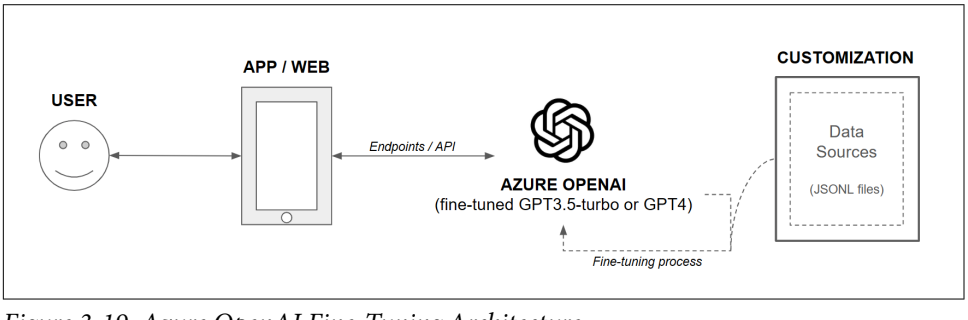

Figure 3-19. Azure OpenAI Fine-Tuning Architecture

Concretely, the steps to perform *fine tuning* with Azure OpenAI are:

1. *Preparing your dataset* in JSONL format. For recent models such as GPT-3.5 turbo or GPT4, you will leverage the chat completion API structure for system and user messages.
2. Launching the *customized model wizard* from Azure OpenAI Studio as shown in Figure 3-20, to train your new customized model.

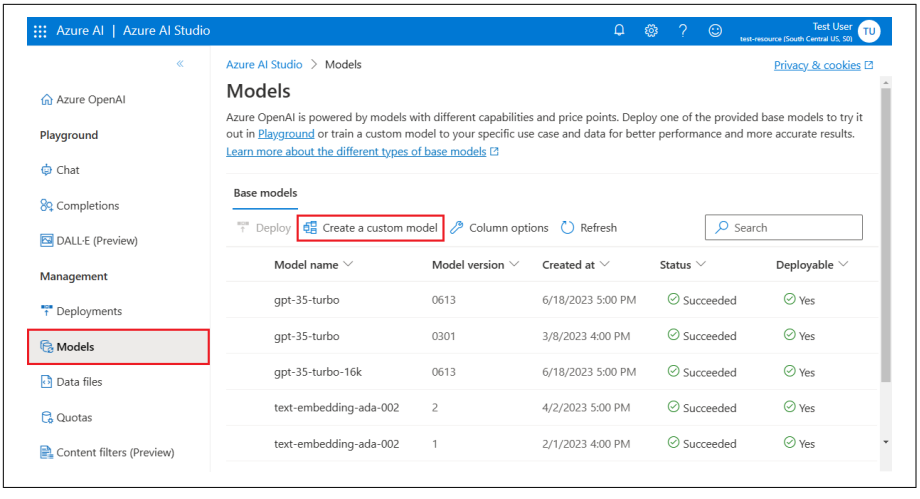

Figure 3-20. Azure OpenAI - Custom Model Wizard

3. *Selecting a base model* (e.g., GPT-3.5 turbo), choosing your training data and, optionally, your validation data to evaluate the model performance. Those datasets are the JSON files you have previously prepared.
4. Reviewing your choices and *launching the training* of the new customized model. Check the status of your customized model and wait for the training to finish.
5. *Deploying your customized model* for use in an application or service, via APIs.

All these options can be good depending on the type of application, and the intended scope of the model customization. However, there are ways to combine the LLM with internal data sources, from which you can extract knowledge, and then refer to that information from the Azure OpenAI completion and chat completion models. This is what we call grounding, and there are different ways to implement it. The next sections contain different grounding alternatives.

Embedding-based grounding

As you now know from earlier chapters, embeddings are mathematical representations of text-based information. These embeddings are stored and managed as mathematical vectors that represent distances between topics. This means, if we are looking for information about animals and we have a vectorized knowledge base that includes animal-related topics, we can recover the Top K answers (i.e. the most relevant "K" number of pieces of information).

Concretely, an Azure OpenAI project with embeddings uses the Azure OpenAI embeddings API to generate vector representations of text that capture the semantic

meaning and similarity of the text. Some possible use cases for embeddings are document search, text classification, or text similarity.

The end-to-end process to create and use an embedding-based system is aligned with what you have seen thus far in this chapter. From an Azure OpenAI perspective, the steps are:

1. *Select the knowledge base* that contains the information that will complement the baseline LLM knowledge domain. This may include pdf, doc, ppt, txt, and other file formats. In Azure, you may store that information via Azure Blob Storage or Azure Data Lake Gen2. Keep in mind if your files are similar to any general information that may be available on the internet, you probably don't need to ground them. However, if you have very specific files with information on how to answer questions, or perform internal tasks, those may be good candidates for embeddings generation.
2. *Choose and deploy your database / vector store*. By the end of this chapter, you will see all available options for implementations in Azure, with Azure OpenAI-generated embeddings.
3. *Prepare the input dataset*. This includes two different steps:
4. *Extract the information* from your documents. For example, you can use Azure Document Intelligence / Form Recognizer to extract text from your PDFs with the OCR feature. You can also use other non-Azure tools.
5. *Split the information*. For this to work, it is important to keep in mind the embeddings model token limit (e.g., 8k for Ada model version 2), to prepare the input without exceeding the limit (you can use OpenAI's tokenizer tool to understand the extent of what 8k means in terms of document length). This means, you will need to make one API call for each of the limited-size blocks you have prepared before.
6. *Leverage the Azure OpenAI* embeddings models. Use the API operation you saw earlier in this chapter, and get the mathematical vectors from the API response. *Store the vectors* within the chosen vector store.
7. Any time you want to find information from your knowledge base, or if you want to leverage it from any chat or search application, you will need to *generate the embeddings of the question* itself, then perform the search against the vector search. Keep in mind that you will need to leverage the same model (e.g. Ada version 2) for both your knowledge base and the question. You can send the result of the search, with the Top K results, to the chat or search application, directly or by including it as content for the answer of the completion.

This process is similar for other embeddings and conversation models (for example, those that are available via Azure AI Studio's model catalog and Hugging Face), and

the high-level architecture includes the elements you can see in Figure 3-21: basically, the baseline Azure OpenAI model gets complemented with the internal knowledge base that contains PDFs, docs, etc. Instead of re-training / fine-tuning the model, we just combine it with that knowledge base so it can find similarities between the users' questions, and the information contained within the data sources.

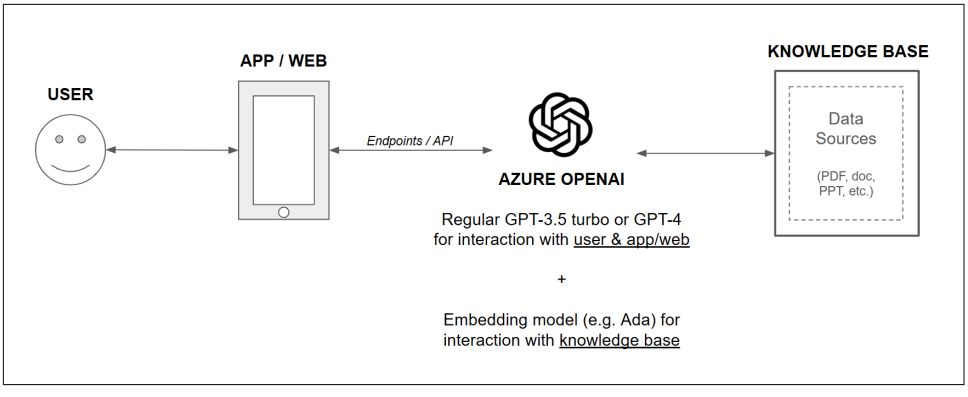

Figure 3-21. Embedding-based Grounding Architecture

You can find more information and code examples on how to create embeddings from the official Microsoft documentation (in addition to the API definitions I covered earlier in this chapter).

Additionally, there is one official Microsoft accelerator for this type of implementation that you can leverage during the development phase. There are several deployment and storage options. Feel free to explore the code to see the API call details.

Document indexation / retrieval-based grounding

The document indexation/retrieval-based grounding approach is an alternative to the embedding-based approach. In this case, we will not generate mathematical vectors. Instead we will generate indexes of specific documents, so the Azure OpenAI Service can find the information from those sources and include it as part of its answers.

For that purpose, we will also use Azure Cognitive Search, which is a service that allows you to index, understand and retrieve the relevant data from a knowledge base or a collection of documents.

The combination of both services enables powerful chatbot applications that can communicate with users in natural language and provide intuitive and personalized interactions, based on specific data from the organization. Much like the embedding-based approach, there is an official Microsoft accelerator available for you to deploy your first PoCs, in addition to a second one called GPT-RAG from the Microsoft Argentina team, with some additional functionalities for bigger implementations. You

can explore both to see updated details and implementation approaches with Azure OpenAI and Azure Cognitive Search. You can also see the high-level architecture of the key building blocks in Figure 3-22:

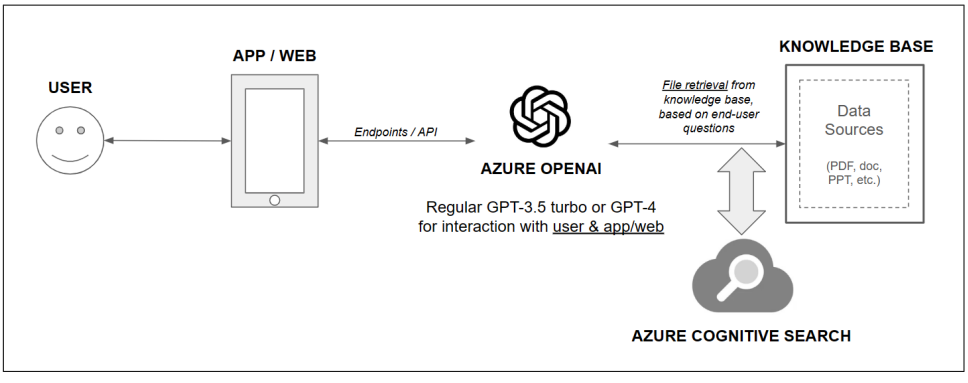

Figure 3-22. Retrieval-based Grounding Architecture

The main difference when compared to the embedding-based approach is that instead of generating embeddings for both the knowledge base and the user question, you will just perform a search against the Azure Cognitive Search engine.

You may see this option as something a bit simpler than the embeddings approach, and a better fit for applications where you need to find the source of information (and even provide a link to the original document, as part of the answer), while embeddings can potentially handle bigger datasets and deliver better performance. However, it really depends on the specific dataset with its knowledge scope and file format, and the envisioned use case, so my recommendation is for you to try both options and evaluate the one that delivers best results from a user perspective.

Now, let's explore some additional grounding options you have to add more knowledge scope to your Generative AI applications.

Other grounding techniques

We have explored several fine-tuning and grounding techniques, mainly based on text information from different sources. But what happens if you want to leverage other kinds of data? Or if the required information can only be found via live internet results? Here are some other grounding techniques you may want to explore:

LLM + web results
: This approach relies on the Bing Web Search API to extend the knowledge scope of the Azure OpenAI Service models. As you may know, all LLMs are based on training datasets that go up to a specific date (e.g. initial Azure OpenAI models were updated with data up to 2021). If you need updated information, you can use Bing Web Search API to find web pages, images, videos, news, etc., or use it

to create a custom search instance that filters the web results based on the criteria. The result of the API can be then used by Azure OpenAI to return an answer based on that information.

LLM + tabular data and/or databases
: Similar to other sources, tabular data (e.g. excel and .csv files) and regular SQL databases can be good grounding sources. Concretely, you can develop what the industry calls Database Copilots, to allow end-users to query information without any complex SQL syntax, just with natural language-based prompts.

Just as with the two previous grounding options, there is an official Microsoft accelerator that combines these grounding techniques, with specific code samples and updated implementations.

At the end of the day, each implementation approach (baseline, fine-tuned, or grounding-based) serves a different purpose, but the next section is a summarized guide for you to understand the pros and cons of each one, so you can make the best informed decision, and create your Generative AI applications with Azure OpenAI with the best balance of performance, cost, and technical complexity.

Approach Comparison and Final Recommendation

There is not a single answer to the question, "*which approach should I use for my Generative AI implementation?*". It really depends on the type of use case, sort and volume of available data, existing IT architectures, available budget and resources, etc.

Table 3-1 shows the general pros and cons of implementation approaches:

Table 3-1. Comparison of implementation Approaches with Azure OpenAI Service

	Approach	Pros	Cons
1	**Basic ~ ChatGPT instance** (vanilla, private)	• Relatively simple and quick to deploy • Good option for internal (employee) use cases • Available via Azure OpenAI's visual playground • Option to define the topic scope based on URLs, by leveraging the system message	• Lack of updated data • Very limited for client-side applications • Higher risk of model hallucination
2	**Examples with one/ few-shot learning**	• Easy to implement • Good option to adapt the system behavior based on specific pieces of knowledge from your company • Available via Azure OpenAI's visual playground	• Lack of updated data • Very limited for client-side applications • Higher risk of model hallucination
3	**Fine tuning** (e.g. fine tuning, contextualization)	• Good to fine tune an existing model with specific company data • Leverages mature product features	• Complexity to prepare input data for both fine tuning or few-shot learning • Increased cost for fine tune models

	Approach	Pros	Cons
4	**Embedding-based grounding** (vectors)	• Great for customization without fine-tuning required • Good fit for big amounts of data • Easy use of embeddings APIs	• Required preparation of the input data based on token limits • Need to scan files via Optical Character Recognition (OCR) to extract content first • Initial embeddings generation cost for custom data (depending on the data scope)
5	**Doc retrieval-based grounding** (indexation with Azure Cognitive Search)	• Good option for information retrieval from existing files • Indexation allows citing sources (good for explainability) • Option to use the "add your own data" option from Playground, for small implementations	• Potentially less performant than embeddings for big amounts of private data (to be confirmed during your preliminary experimentation)
5	**Other grounding techniques** (Bing Search, databases, etc.)	• Great to add live results to the LLM, and to explore internal sources such as databases and tabular files • Updated results with no need to retrain or adjust the model	• A bit more complex (it requires orchestration engines such as LangChain) • There is less documentation available for this kind of implementation

There are new implementation approaches based on hybrid-search techniques that combine embeddings and doc retrieval capabilities. This concept will be developed before the final release of this book, but due the evolving nature of the topic, feel free to explore details for now via Microsoft's documentation.

All these implementation approaches have different advantages and levels of complexity. One of the key aspects is the ability to evaluate how well they perform, and how good these Azure OpenAI models are for specific questions and tasks. Let's explore all of this in the next section.

AI Performance Evaluation Methods

One of the key stages of any Generative AI project is the model performance evaluation. However, it is not a simple task to evaluate the performance of a Generative AI system. There are several initiatives going on from some of the main industry actors (including Microsoft and OpenAI), but you can expect more news and tools during the next months.

This section is a placeholder for the Early Release version of the book. We will develop the details of how you can evaluate your Azure OpenAI model performance before the final release of the book, but meanwhile feel free to explore these sources:

- Microsoft's Evaluation Flows (Azure AI Studio)
- Microsoft's Solutions Playbook - AI Evaluation Metrics
- Microsoft's doc for LLM metrics monitorization
- OpenAI's Evals project https://github.com/openai/evals
- Langchain's Evaluation module https://python.langchain.com/docs/modules/evaluation

Additionally, there are other families of metrics that we could use to measure and analyze performance:

- *Positive/negative review of answers*, as a manual way to both track performance and potentially re-educate the model with weighted reconfigurations (e.g., few-shot learning with the good answers). You could enable this by using a positive/negative sign in the user interface, and by adding a binary numeric value at DB-level if you decide to store the questions and answers for review purposes (e.g., ID, question, answer, review) in a JSON file stored via CosmosDB. For this purpose, my recommendation is to create a set of test questions, and to involve subject-matter experts during the creation of that set, and during the evaluation of the system.
- *Traditional Product Analytics metrics* such as session time, number of re-questions to get the best answer, overall product rating, etc. This would require tools such as Microsoft Clarity, Pendo, Amplitude, Mixpanel, etc. connected to the cloud native app (e.g., iOS, Android, web, etc.). Alternatively, there are cloud native features such as Azure App Insights that can be deployed as part of the Generative AI app monitoring system. Additionally, these tools can be leveraged to track performance for A/B testing experiments (for example, if we launch two different versions of the AI model with different user sets).

Conclusion

This chapter has included not only the available visual and code-based tools for your Azure OpenAI implementations, but also the recommended implementation approaches, understanding the differences between regular, fine-tuned, and grounded LLMs. Once again, there is not a perfect or a single way to do it. All of these approaches try to leverage the existing power of the Azure OpenAI models, and the ability to increase the knowledge scope of your applications with examples, internal data sources, live internet search, etc.

Now, let's take a look at the additional building blocks for your Generative AI developments.

CHAPTER 4

Additional Cloud and AI capabilities

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be Chapter 4 of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *mpotter@oreilly.com*.

Generative AI applications are way more than "just a big model". As you have already seen, LLMs play a central role, but there are other relevant pieces that complement the Azure OpenAI Services capabilities: Fine tuning via Azure OpenAI APIs or playground, grounding with Azure Cognitive Search and/or Azure OpenAI embeddings, live search capabilities with Bing Search API, etc.

Additionally, we have new kinds of tools that allow us to expand the capacities of LLMs even more. A curated selection for any generative AI and Azure OpenAI adopter could include plugins, LMM integration, databases, and more. Let's dig into these in more detail.

Plugins

One of the most important new terms in AI applications is the notion of "plugins". We can define them as direct interfaces to advanced functionalities, interconnecting

Microsoft's Azure OpenAI (or OpenAI's ChatGPT) with other systems. For example, there are plugins from companies such as Expedia, FiscalNote, Instacart, KAYAK, Klarna, Milo, OpenTable, Shopify, Slack, Speak, Wolfram, and Zapier.

Additionally, Microsoft announced in May 2023 its own collection of plugins, defining them as "standardized interfaces that allows developers to build and consume APIs to extend the capabilities of large language models (LLMs) and enable a deep integration of GPT across Azure and the Microsoft ecosystem". Those plugins include direct interfaces to Bing Search, Azure Cognitive Search, Azure SQL, Azure Cosmos DB, and Microsoft Translator. As a developer, that means that you can connect Azure OpenAI with other Microsoft and Azure-related pieces, with minimal development and integration effort.

LLM Development, Orchestration, and Integration

There are also developer-oriented pieces that enable the combination of existing LLMs with other services, regardless of the programming language. Let's dig into some of those options now.

LangChain

LangChain is an open-source framework that you can use to develop applications that are powered by language models. It provides various language-related utilities and tools (e.g., embeddings, pipelines, agents, plugins). It is one of the key components for some of the accelerators you saw earlier in AI Modeling with Azure OpenAI, and the official documentation talks about six key areas, noted in increasing order of complexity:

LLMs and Prompts
: This includes prompt management, prompt optimization, a generic interface for all LLMs, and common utilities for working with LLMs.

Chains
: Chains go beyond a single LLM call and involve sequences of calls (whether to an LLM or a different utility). LangChain provides a standard interface for chains, lots of integrations with other tools, and end-to-end chains for common applications.

Data Augmented Generation
: Data Augmented Generation involves specific types of chains that first interact with an external data source to fetch data for use in the generation step. Examples include summarization of long pieces of text and question/answering over specific data sources.

Agents
: Agents involve an LLM making decisions about which Actions to take, taking that Action, seeing an Observation, and repeating that until done. LangChain provides a standard interface for agents, a selection of agents to choose from, and examples of end-to-end agents.

Memory
: Memory refers to the persisting state between calls of a chain/agent. LangChain provides a standard interface for memory, a collection of memory implementations, and examples of chains/agents that use memory.

Evaluation
: Generative models are notoriously hard to evaluate with traditional metrics. One new way of evaluating them is using language models themselves to do the evaluation. LangChain provides some prompts/chains for assisting in this.

Here is an article from one of the Microsoft AI specialists, explaining an example of implementation of Azure OpenAI Service with Langchain. You can also check the official integration doc for Langchain and Azure OpenAI Service.

Semantic Kernel

Semantic Kernel is an open-source SDK that helps combine Azure OpenAI Service and other LLMs with regular programming languages like C# and Python. The SDK includes features such as prompt chaining, recursive reasoning, summarization, zero/few-shot learning, contextual memory, long-term memory, embeddings, semantic indexing, planning, retrieval-augmented generation, external knowledge stores, and the "as your own data" product features.

The end-to-end notion of the kernel includes the building blocks you can see in Figure 4-1:

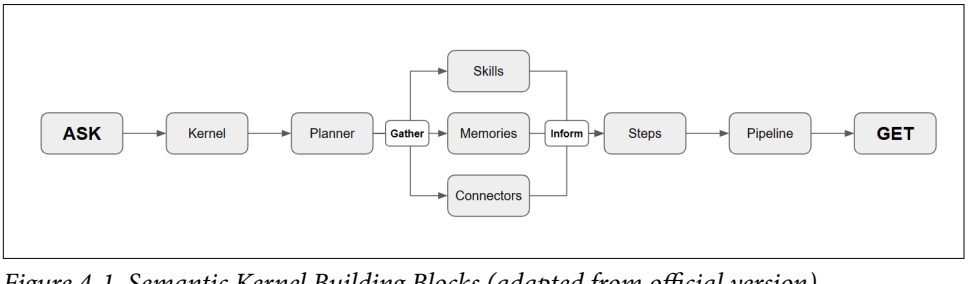

Figure 4-1. Semantic Kernel Building Blocks (adapted from official version)

Let's look at each part of the figure in more detail:

ASK
: Refers to the input you present to the Semantic Kernel, which might be a question or directive in natural language. Examples include inquiries like, "Can you name the tallest mountain in the world?" or commands like, "condense this piece of text". The Semantic Kernel seeks to interpret your ASK and cater to it using its diverse capabilities.

Kernel
: Represents the core processor that manages your ASK, choreographing a blend of AI services, memories, models, and add-ons to craft a response. One has the flexibility to tailor the kernel's settings and parameters as per their preference.

Planner
: An intrinsic facet of the kernel, the Planner ingeniously stitches functions together, formulating a strategy to address your ASK. If posed with a request like, "compose a haiku about the moon", the planner might amalgamate functions related to topic initiation, rhyming, and poem structuring.

Gather
: At this juncture of the plan, the kernel embarks on collating data from an array of outlets including AI models, memory storages, connectors, or even vast external repositories of knowledge. If you're curious about "the largest mammal on Earth", the kernel could tap into resources like Bing or Wikipedia for answers.

Skills
: Epitomize the proficiency range of the Semantic Kernel in addressing your ASKs, harnessing its vast components. Skills might span a spectrum from basic to intricate, contingent on the intricacy of steps and sources entailed. Examples of skills encompass areas like elucidation, interpretation, crafting, categorizing, and even answering queries. Consider skills as the functional "anatomy" of your AI application.

Memories
: The system's capacity to store, recall, and process information derived from previous interactions, tasks, or externally acquired data. It's analogous to how human memory works but in a digital format.

Connectors
: Serve as bridges, enabling Semantic Kernel to interface with a spectrum of AI platforms, models, memories, or external data troves. Using connectors, one can tap into platforms like OpenAI, Azure OpenAI, or even models like ChatGPT, besides memory systems or information havens like Encyclopaedia Britannica.

Inform
: This phase sees the kernel apprising you of your ASK's outcome, which could be manifested as text, visuals, audio, or a specific activity. Suppose you prompt, "illustrate a tranquil forest"; the kernel might respond by curating an image-based query and projecting the resultant picture.

Steps
: Constituting the foundational blocks in the plan to address your ASK, each step might be framed as a prompt or an innate function. Prompts entail language-based directives dispatched to the Azure OpenAI models. In contrast, native functions refer to standardized coding operations, often scripted in languages like C# or Python. For instance, in response to "describe the process of photosynthesis", one step might involve a prompt pinpointing the core mechanisms, while another leverages a native function to align them in a bulleted format.

Pipeline
: Essentially a series of actions initiated to address your ASK. Pipelines can either be pre-set or dynamically crafted by the planner. For a request like, "pen a narrative on the evolution of technology", the pipeline could encompass stages such as outlining, crafting an intro, articulating main content, concluding, and final arrangement.

GET
: Denotes an operation you can initiate on the Semantic Kernel to glean details or data from its constituents. You might, for example, utilize GET to peek into the kernel's memory bank, exploring its reservoir of past insights.

Last but not least, Semantic Kernel is one of the in-house Microsoft projects for Generative AI, an effort led by Dr. John Maeda, and the information is available on the official GitHub repository. Here are some additional resources if you want to continue exploring the Semantic Kernel:

- Official cookbook with Semantic Kernel "recipes"
- LinkedIn Learning free course from Dr. John Maeda
- Some words from the creators (interview, and video article)

Bot Framework

This Microsoft Bot framework is a classic of the pre-ChatGPT bot era. It does not rely on the Azure OpenAI Service, but some adopters are using it for specific scenario integrations (e.g, to deploy the GPT-enabled projects within Microsoft Teams or other communication channels), and it includes a set of tools and services intended to help build, test, deploy, and manage intelligent chatbots:

Bot Framework SDK
: This is a modular and extensible software development kit that allows you to build bots in C#, JavaScript, Python, or Java. The SDK provides libraries and templates for common bot scenarios, such as dialogs, state management, authentication, etc.

Bot Framework Composer
: This is an open-source visual authoring tool that lets you create bots using a graphical interface and natural language prompts. You can use the Composer to design dialogs, skills, and answers for your bot without writing code.

Azure Bot Service
: This is a cloud service that enables you to host your bot on Azure and connect it to various channels and devices, such as Facebook Messenger, Microsoft Teams, Skype, web chat, etc. The Azure Bot Service also provides features such as analytics, debugging, security, etc.

Bot Framework Emulator
: This is a desktop application that allows you to test and debug your bot locally or remotely. You can use the Emulator to send and receive messages from your bot, inspect the bot state and activities, and access the bot logs.

Bot Framework Connector
: This is a service that handles the communication between your bot and the channels or users. The Connector defines a REST API and an activity protocol for how your bot can send and receive messages and events.

As you can see, the Microsoft Bot Framework is a complete solution for classic bot (not LLM) scenarios, with Azure Bot Service being part of some of the official Azure OpenAI Accelerators. The full spec is available via its official Github repository.

Power Platform, Power Virtual Agents, and AI Builder

Besides the SDKs and development frameworks, there are other pieces for no-code and low-code implementations. Concretely, the Power Platform suite of tools and services helps build and manage low-code applications, automate workflows, analyze data, and build chatbots, and to connect these to any AI-enabled Azure feature, including Azure OpenAI. Power Virtual Agents (PVA) is one of the components of Power Platform that allows you to create intelligent chatbots using a low-code graphical interface. You can use Power Virtual Agents to build bots that can provide answers, perform actions, and interact with users in natural language.

There are three different ways these components can interact with Azure OpenAI for no/low-code Generative AI applications:

By using an *Azure OpenAI connector from PVA*, via Azure OpenAI APIs. Here is an example of implementation for that scenario. This is a highly manual option, but still simple to implement.

By leveraging the Boost Conversations (aka *Generative answers*) feature of PVA. This functionality allows the bot to find and present information from multiple sources. Generative answers can be used as primary in the chatbot, or as fallback when other authored topics are unable to address a user's query.

Besides these two bot-kind of applications, you can also leverage the Power Platform's AI builder component and its integration with Azure OpenAI for automation and apps. This video demo illustrates the implementation process.

All these development building blocks are your tools to continue evolving your Generative AI projects with Azure OpenAI. The list will probably grow over time, but this selection represents some of the most relevant pieces for any Generative AI practitioner today. Let's now check the available vector database for Azure-first implementations, that will enable your embedding-based projects with Azure OpenAI and allow you save the generated vectors.

Databases / Vector Stores

As previously mentioned, embeddings is a technique that generates mathematical representations of distances between topics, and that information is what we call a "vector". For that purpose, we rely on a specific kind of database called vector databases, as they are better suited to store and manage this kind of information.

The main advantage of a *vector database* is that it allows for fast and accurate similarity search and retrieval of data based on their vector distance or similarity. This means that instead of using traditional methods of querying databases based on exact matches or predefined criteria, you can use a vector database to find the most similar or relevant data based on their semantic or contextual meaning. Vector databases are used to store, search, and retrieve vectors (previously generated via embedding techniques) representing documents, images, and other data types used in machine learning applications.

From an Azure OpenAI Service point of view, there are different native Azure services, or open source pieces deployable via Azure, that will serve as vector databases. Let's go through those now.

Vector Search from Azure Cognitive Search

The Vector Search is a recent feature from one of the existing Microsoft Azure AI services, specifically the Azure Cognitive Search. This piece is part of the implementation options 3 and 4 we explored earlier in the chapter, for both embedding and retrieval-based approaches.

Vector search is a new capability for indexing, storing, and retrieving vector embeddings from a search index. You can use it to enable typical cases such as similarity search, multi-modal search, recommendations engines, or grounding / Retrieval Augmented Generation (RAG) implementations. The main differentiator (based on its creators words) is the ability to enable not only classic vector search, but also "a hybrid search approach that harnesses both vector and traditional keyword scores delivers even better retrieval result quality than a single search method alone", as illustrated in Figure 4-2:

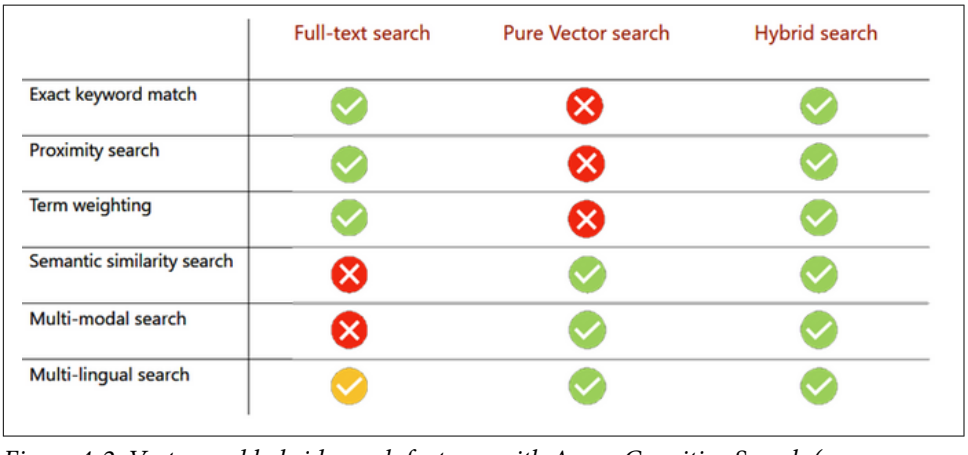

Figure 4-2. Vector and hybrid search features with Azure Cognitive Search (source: Microsoft)

You can leverage the official documentation for this highly evolving technology block, as well as the technical guide on how to save your previously generated vectors, and how to perform vector queries via Azure Cognitive Search's vector search feature.

Vector Search from CosmosDB

Vector Search is a similar vector feature from a different Azure native service, in this case Azure CosmosDB, which is a managed multi-type NoSQL database, which supports several types of key-value, column, graph, and document formats. It includes open-source options such as PostgreSQL, MongoDB, and Apache Cassandra.

The vector search feature comes from Azure Cosmos DB for MongoDB vCore products, which provides a fully managed MongoDB-compatible database service in Azure. The new functionality was announced in May 2023, and it's an alternative to the Azure Cognitive Search option. This is an option for environments where MongoDB is already on the technology stack. You can view some additional repo with implementation samples.

Redis Databases on Azure

An alternative option is the Azure Cache for Redis, which is a solution to accelerate the data layer of the applications through in-memory caching, and it is based on the Redis open source databases. It contains RediSearch, which is a Redis module that provides full-text search capabilities. The Azure version is built on top of the Redis engine and is designed to be used with Redis Enterprise.

Similar to the previous two options, Azure Cache for Redis has evolved and intercorporates a new vector search feature which combines the power of a high-performance caching solution with the versatility of a vector database, opening up new frontiers for developers and businesses.

As with CosmosDB, this option is great for those companies already using Redis or Azure Cache for Redis as part of their technology stack.

Other Relevant Databases (Including Open Source)

There are other options available, including native and open source solutions that can be leveraged via Azure:

Pinecone on Azure
: Available in private preview since July 2023, it allows deploying a Pinecone vector (commercial, fully managed) database directly via Azure. Here is an implementation sample with Azure OpenAI Service and the Pinecone database.

Milvus
: It is a very relevant open source project for vector databases, available on Azure. It is one of the main open source contenders, and a graduated project from the Linux Foundation.

Faiss's
: It is Meta's library for efficient similarity search and clustering of dense vectors. Index Lookup Tool via Azure ML Prompt Flow allows querying within a user-provided Faiss-based vector store.

Azure Data Explorer for Vector Similarity Search
: This is another vector store option to store embeddings by using an Azure native service. Here is a step-by-step explanation.

Feel free to explore all these vector store and database options, and others from the OpenAI Cookbook list. The simplest way to start is by leveraging native services such as Azure Cognitive Search or Azure CosmosDB, but the choice will depend on your implementation approach. Let's now take a look at some additional technology building clocks you may need for your Generative AI projects.

Others Microsoft Building Blocks for Generative AI

In addition to what I've already covered in this chapter, there are some consolidated services and ongoing research projects that can be leveraged for our Azure OpenAI projects. Let's dig into a few of those.

Azure AI Document Intelligence (Formerly Azure Form Recognizer) for OCR

Some of the grounding scenarios we have previously analyzed rely on images and PDF documents as the main source of knowledge, in addition to the base LLM. If we want to combine the LLM knowledge with the information from those images and PDFs, we need to extract the information from those documents in advance, and have it transformed from the source to relevant formats such as JSON or JSONL.

For PDFs, the classic technique that extracts text from the document is OCR (Optical Character Recognition). This is a mature kind of technique that recognizes every character of a document, to read and extract its information for later use.

If we want to leverage native Azure services to perform OCR tasks, there is an Azure AI service called AI Document Intelligence (previously called Form Recognizer). From the official website, it is "an AI service that applies advanced machine learning to extract text, key-value pairs, tables, and structures from documents automatically and accurately". This means, the preliminary step before performing fine tuning, embeddings, etc. This official article explains the end-to-end process that combines AI Document Intelligence and Azure OpenAI Service, to directly launch queries against the document.

Alternatively, the previously mentioned Azure Cognitive Search service, includes a similar OCR cognitive skill that works with both images (that contain text) and documents.

Ongoing Microsoft Open Source and Research Projects

Below I've included a selection of ongoing Microsoft research projects, all of them related to LLM development. Most of them are not production-ready building blocks, but even if they won't be used by regular Generative AI practitioners, you may want to take a look and see the latest Generative AI-related developments:

DeepSpeed
: A deep learning optimization library developed by Microsoft, designed to help researchers train large-scale models faster and more efficiently, between 10 and 100 times larger than previously possible. Additionally, DeepSpeed Chat is an open system framework for enabling end-to-end Reinforcement Learning

Human Feedback (RLHF) training experience to generate Generative AI models at all scales.

ONNX Runtime
: A cross-platform inference and training machine-learning accelerator, intended to improve customer experiences (by providing faster model inference) and to reduce training costs. It was open sourced by Microsoft in 2019, and it is based on the ONNX open format (co-developed by Microsoft with Meta and AWS). It includes the DirectML execution provider, a component of ONNX Runtime to accelerate inference of ONNX models.

JARVIS / HuggingGPT
: A project to use LLMs as interfaces to connect different AI models from Hugging Face and others, for solving complicated AI tasks.

ToxiGen
: A large machine-generated dataset for hate speech detection, from Microsoft.

LLM-Augmenter
: A project that aims to reduce hallucinations (i.e. LLMs delivering wrong answers), with external knowledge to LLMs, and automated feedback.

AdaTest
: Another Microsoft project to find and fix bugs in natural language / machine learning models using adaptive testing.

LoRa(Low-Rank Adaptation)
: It helps reduce the number of training parameters for large language models, making this process less storage and computing intensive.

Guidance
: A Microsoft project that enables control of modern language models more effectively and efficiently than traditional prompting or chaining.

PromptCraft-Robotics
: Another research project that aims to combine ChatGPT and robotic systems such as drones, camera-enabled robots, etc.

Gorilla LLM
: A project between Microsoft Research and the University of Berkeley, who have developed a LLM connected to APIs, which means that it can provide appropriate API calls for different topics including Torch Hub, TensorFlow Hub, HuggingFace, Kubernetes, OpenAPI, and others. A great step towards a more general kind of intelligence.

PowerProxy AI
: A project that helps monitoring and processing traffic to and from Azure OpenAI Service endpoints.

AutoGen framework
: It enables the development of LLM applications using multiple agents that can converse with each other to solve tasks.

UniLM
: A Microsoft repo that contains a series of research papers and links to other LLM-related Github repositories.

LIDA
: A Microsoft library for automatic generation of visualizations and infographics with large language models.

Algorithm of Thoughts
: A research paper that explores potential LLM improvements with human-like reasoning techniques.

Given that Generative AI is a highly evolving area, I aimed to provide you with a quick overview of what the industry is trying to achieve, in addition to the core Azure OpenAI LLM capabilities you already know.

Conclusion

This chapter is a continuation of the previous one, and it includes an ecosystem of projects and technologies that you can leverage to create very advanced cloud-native architectures. Most of them are additional and complementary to Azure OpenAI, and required to implement some of the Chapter 3 technical approaches.

This is a highly evolving area, so consider this chapter as the initial toolbox for your Generative AI practitioner journey. The next chapter will focus on the notion of LLMOps (LLM operations, and evolution of the DevOps and MLOps concepts), and how to handle production-level topics such as performance, security, and privacy. Let's explore it together.

About the Author

Adrian Gonzalez Sanchez is a Senior Cloud, Data and AI Specialist at Microsoft, as well as the Industrial AI Lead for the Spanish Observatory of Ethical AI (OdiseIA). He has previously worked as a Senior Consultant and Product Owner in AI and data science with CGI Canada and IVADO Labs, Head of AI Customer success for Peritus.ai, and other data-driven companies in Europe and Latin America.

He is a trainer for the École des Dirigeants at HEC Montréal, the Continuing Education Department at Concordia University, and online training courses for O'Reilly Media and the Linux Foundation. He also collaborates with 2U / GetSmarter as a tutor and facilitator for MIT Sloan's AI and Blockchain executive courses, and Harvard VPAL's Fintech course.

His areas of expertise are AI strategy and project management, responsible AI systems implementation, big data and cloud computing, data and AI governance, ethical and regulatory impact of technological innovation, telecommunications, and the Internet of Things.