EXPERT INSIGHT

Generative AI Application Integration Patterns

Integrate large language models into your applications

Foreword by:
Dr. Ali Arsanjani
Director of Applied AI Engineering, Google

Juan Pablo Bustos
Luis Lopez Soria

packt

Generative AI Application Integration Patterns

Integrate large language models into your applications

Juan Pablo Bustos

Luis Lopez Soria

packt

Generative AI Application Integration Patterns

Senior Publishing Product Manager: Tushar Gupta
Acquisition Editor – Peer Reviews: Tejas Mhasvekar
Project Editor: Meenakshi Vijay
Content Development Editor: Shazeen Iqbal
Copy Editor: Safis Editing
Technical Editor: Gaurav Gavas
Proofreader: Safis Editing
Indexer: Pratik Shirodkar
Presentation Designer: Rajesh Shirsath
Developer Relations Marketing Executive: Maran Fernandes

First published: August 2024

Production reference: 1290824

Published by Packt Publishing Ltd.
Grosvenor House
11 St Paul's Square
Birmingham
B3 1RB, UK.

ISBN 978-1-83588-760-8

`www.packt.com`

Foreword

The field of Artificial Intelligence is in the midst of bringing about a profound transformation in business. One of the pivotal areas of this transformation is in the integration with applications that are already running businesses, and adding value to them. This book, *Generative AI Application Integration Patterns* explores these recurrent themes as patterns of integration with generative AI. It serves as a timely guide for navigating the nuanced landscape of integrating GenAI into existing business applications. It peers into the fog of the immense potential of GenAI, and provides practical clarity that may help you revolutionize key competitive aspects of your business operations; areas like enhancing customer experiences to domains such as streamlining internal processes. By focusing on the practical aspects of integration, the authors equip readers with the background knowledge and tools they need to leverage this transformative technology even more effectively.

Juan and Luis delve into the underlying practical aspects of generative AI, and in doing so, provide a solid foundation for understanding and actualizing its capabilities and navigating its limitations. They explore the different architectural integration patterns that can be employed for more seamless integration, and consider how factors such as scalability, performance, and security should be taken into account. The practical case studies presented throughout the book showcase how successful implementations of GenAI can be realized across industries. These examples are exemplary blueprints that demonstrate how businesses can leverage this technology to achieve tangible outcomes.

In addition, this book addresses some of the critical considerations of responsible AI development and deployment. It emphasizes the importance of ethical considerations, data privacy, and bias mitigation, that help ensure that GenAI is utilized in a manner that aligns with ethical principles and societal values. This holistic approach helps readers not only gain technical expertise but also develop a deeper appreciation of the ethical challenges and implications of their work.

Generative AI Application Integration Patterns is a well-written, engaging and very relevant set of blueprints that technology and business leaders, as well as developers, should be aware of as they seek to integrate applications with the promise presented in GenAI. I encourage you to dive deep into the examples, reflect on the concepts presented in this book, and embark on the exciting journey of discovery and innovation in harnessing the potential of GenAI. The future of business is being shaped by AI, and this book is an essential companion on that path.

Dr. Ali Arsanjani

Director of Applied AI Engineering, Google

Contributors

About the authors

Juan Pablo Bustos is a seasoned technology professional specializing in artificial intelligence and machine learning. With a background in computer science, Juan has held leadership positions at major tech companies including Google, Stripe, and Amazon Web Services. His expertise spans AI services, solution architecture, and cloud computing. Juan is passionate about helping organizations leverage cutting-edge technologies to drive innovation and deliver value.

I'm deeply grateful to my wife Cinthia for her constant support, encouragement, and for being my sounding board for even my craziest ideas. Thanks to Penny and Andrew, my kids, for their patience. I'd like to acknowledge my father, Dr. Sergio Bustos, for encouraging me to pursue computer science. I'm indebted to Dr. Ali Arsanjani, Robert Love, and Todd Reagan for their invaluable mentorship and friendship. A special thanks to my friend Luis, my partner in crime for this book. Finally, I'd like to recognize Gemini, Claude, and ChatGPT for their invaluable help and for democratizing access to GenAI.

Luis Lopez Soria is an experienced software architect specializing in AI/ML. He has gained practical experience from top firms across heavily regulated industries like healthcare and finance, as well as big tech firms like AWS and Google. He brings a blended approach from his experience managing global partnerships, AI product development, and customer-facing roles. Luis is passionate about learning new technologies and using these to create business value.

I want to thank my parents and sister. Your unwavering support, willingness to lend an ear, and readiness to brainstorm ideas have been invaluable. To my grandfather Felix and uncle Ricardo: your presence and support by my side made this dream a reality.

A special thanks goes to Chris K. and Juan B., whose early influence on my career cannot be overstated. Your constant push for excellence and valuable input have left an indelible mark on my professional growth and, by extension, on this book.

To all of you, and the many others who have contributed in ways both big and small, I offer my heartfelt gratitude. This book stands as a testament to your belief in me and your ongoing support.

About the reviewer

Aditi Khare holds 8+ years of experience in the AI research and product engineering space.

She is passionate about AI research, open source, and building production-grade AI products. She has worked for Fortune 50 product companies. She has completed a big data analytics course at the Indian Institute of Management, Ahmedabad, and a master's in computer applications at K. J. Somaiya Institute of Management, Mumbai. In her spare time, she enjoys reading AI-related research papers and publishing research paper summaries through her LinkedIn newsletter. For more information about her, visit `https://www.linkedin.com/in/aditi-khare-5840977b/`.

> *I'd like to dedicate my contribution to this book to the loving memory of my beloved mom, the late Mrs. Shashi Khare, who has always been my inspiration and the reason for my achievements.*
>
> *I'd like to thank my father Mr. Alok Khare and my brother Ayush Khare for being very supportive and acting as a guiding force in all my achievements.*

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

`https://packt.link/genpat`

Table of Contents

Preface

More than five years ago, before the widespread adoption of generative AI, we were searching for new ways to enhance application development and user experiences. A few years later, we found ourselves deeply immersed in the world of generative AI, which has opened countless possibilities for innovation. Before discovering the transformative potential of this technology, we, the authors, explored various machine learning techniques, experimenting with different models and reading countless research papers. With generative AI, we found more than just a powerful tool; we discovered a new paradigm that is reshaping how we approach software development and problem-solving.

This book is about sharing the excitement and insights we have gained while exploring and implementing generative AI solutions. It is intended to guide you through the wide world of generative AI applications, with a focus on practical design patterns and real-world implementations. We will cover concepts ranging from basic to advanced, taking you on a journey like the one I experienced while learning to harness the power of generative AI.

Who this book is for

This book caters to a wide audience with a keen interest in generative AI and its practical applications:

- Software developers and engineers with foundational knowledge of AI/ML and Python
- Software architects seeking generative AI best practices and design patterns
- Data scientists, researchers, and analysts looking to incorporate generative AI into their workflows
- Technical product managers with a background in software development
- AI enthusiasts who want to deepen their understanding of generative AI implementation strategies

What this book covers

Chapter 1, *Introduction to Generative AI Patterns*, provides an overview of generative AI concepts, architectures, and their potential impact on application development.

Chapter 2, *Identifying Generative AI Use Cases*, guides readers through the process of identifying and evaluating potential use cases for generative AI across various domains.

Chapter 3, *Designing Patterns for Interacting with Generative AI*, explores different strategies for effectively communicating with and leveraging generative AI models in applications.

Chapter 4, *Generative AI Batch and Real-Time Integration Patterns*, discusses the different approaches for integrating generative AI into both batch-processing and real-time systems.

Chapter 5, *Integration Pattern: Batch Metadata Extraction*, demonstrates how to implement generative AI to extract metadata from large datasets in batch mode.

Chapter 6, *Integration Pattern: Batch Summarization*, covers techniques for using generative AI to create summaries of large volumes of text data.

Chapter 7, *Integration Pattern: Real-Time Intent Classification*, shows how to implement generative AI to classify user intents in real-time applications.

Chapter 8, *Integration Pattern: Real-Time Retrieval Augmented Generation*, explores advanced techniques for building question-answering systems using generative AI and retrieval augmented generation.

Chapter 9, *Operationalizing Generative AI Integration Patterns*, provides guidance on deploying, monitoring, and maintaining generative AI systems in production environments.

Chapter 10, *Embedding Responsible AI into Your GenAI Applications*, addresses ethical considerations and best practices for responsible use of generative AI in applications.

To get the most out of this book

To fully benefit from this book, you should have:

- A solid understanding of Python programming
- Familiarity with basic machine learning concepts
- Experience with software development and application architectures
- Access to a development environment capable of running Python and installing necessary libraries

The chapters contain both theoretical explanations and practical code examples. To run the code in the book, you can follow these steps:

1. Clone the GitHub repository associated with this book.
2. Set up a Python environment with the required dependencies (listed in the repository).
3. Download or access the necessary generative AI models as instructed in each chapter.
4. Run the provided Jupyter notebooks or Python scripts.

Alternatively, you can use cloud-based platforms that offer pre-configured environments for AI development, such as Google Colab or Amazon SageMaker, to run the examples without setting up a local environment.

Download the example code files

The code bundle for the book is hosted on GitHub at `https://github.com/PacktPublishing/Generative-AI-Integration-Patterns-1E`. We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `https://packt.link/gbp/9781835887608`.

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter/X handles. For example: "Mount the downloaded `WebStorm-10*.dmg` disk image file as another disk in your system."

A block of code is set as follows:

```
generation_config = {
  "max_output_tokens": 8192,
  "temperature": 0,
  "top_p": 0.95,
}
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
responses = model.generate_content(
    [prompt],
    generation_config=generation_config,
    safety_settings=safety_settings,
    stream=False,
)
```

Any command-line input or output is written as follows:

```
# cp /usr/src/asterisk-addons/configs/cdr_mysql.conf.sample
     /etc/asterisk/cdr_mysql.conf
```

Bold: Indicates a new term, an important word, or words that you see on the screen. For instance, words in menus or dialog boxes appear in the text like this. For example: "Select **System info** from the **Administration** panel."

Warnings or important notes appear like this.

Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: Email `feedback@packtpub.com` and mention the book's title in the subject of your message. If you have questions about any aspect of this book, please email us at `questions@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you reported this to us. Please visit `http://www.packtpub.com/submit-errata`, click **Submit Errata**, and fill in the form.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packtpub.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `http://authors.packtpub.com`.

Share your thoughts

Once you've read *Generative AI Application Integration Patterns*, we'd love to hear your thoughts! Please `click here to go straight to the Amazon review page` for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere?

Is your eBook purchase not compatible with the device of your choice?

Don't worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don't stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily.

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below:

   https://packt.link/free-ebook/9781835887608

2. Submit your proof of purchase.
3. That's it! We'll send your free PDF and other benefits to your email directly.

1

Introduction to Generative AI Patterns

This chapter provides an overview of key concepts, techniques, and integration patterns related to generative AI that will empower you to harness these capabilities in real-world applications.

We will provide an overview of generative AI architectures, such as transformers and diffusion models, which are the basis for these generative models to produce text, images, audio, and more. You'll get a brief introduction to specialized training techniques, like pre-training and prompt engineering, that upgrade basic language models into creative powerhouses.

Understanding the relentless pace of innovation in this space is critical due to new models and ethical considerations emerging constantly. We'll introduce strategies for experimenting rapidly while ensuring responsible, transparent development.

The chapter also introduces common integration patterns for connecting generative AI into practical workflows. Whether crafting chatbots that leverage models in real time or performing batch enrichment of data, we will introduce prototyping blueprints to jumpstart building AI-powered systems.

By the end, you will have a one-thousand-foot view of which generative AI models are available, why experimentation is important, and how these integration patterns can help create value for your organization leveraging generative AI.

In a nutshell, the following main topics will be covered:

- Interacting with AI

- Predictive AI vs generative AI use case ideation
- A change in the paradigm
- General generative AI concepts
- Introduction to generative AI integration patterns

From AI predictions to generative AI

The intent of this section is to provide a brief overview of artificial intelligence, highlighting our initial experiences with it. In the early 2000s, AI started to become more tangible for consumers. For example, in 2001, Google introduced the "Did you mean?" feature (`https://blog.google/intl/en-mena/product-updates/explore-get-answers/25-biggest-moments-in-search-from-helpful-images-to-ai/`), which suggests spelling corrections. This was one of Google's first applications of machine learning and one of the early AI features that the general public got to experience on a large scale.

Over the following years, AI systems became more sophisticated, especially in areas like computer vision, speech-to-text conversion, and text-to-speech synthesis. Working in the telecom industry helped me witness the innovation driven by speech-to-text in particular. Integrating speech-to-text capabilities into **interactive voice response** (**IVR**) systems led to better user experiences by allowing people to speak their requests rather than punch numbers into a keypad. For example, you could be calling a bank where you would be welcomed by a message asking you to say "balance" to check your balance, "open account" in order to open an account, etc. Nowadays we are seeing more and more implementations of AI, simplifying more complex and time-consuming tasks.

The exponential increase in available computing power, paired with the massive datasets needed to train machine learning models, unleashed new AI capabilities. In the 2010s, AI started matching and even surpassing human performance on certain tightly defined tasks like image classification.

The advent of generative AI has reignited interest and innovation in the AI field, introducing new approaches for exploring use cases and system integration. Models like Gemini, PaLM, Claude, DALL-E, OpenAI GPT, and Stable Diffusion showcase the ability of AI systems to generate synthetic text, images, and other media. The outputs exhibit creativity and imagination that capture the public's attention. However, the powerful capabilities of generative models also highlight new challenges around system design and responsible deployment. There is a need to rethink integration patterns and architecture to support safe, robust, and cost-effective implementations. Specifically, issues around security, bias, toxicity, and misinformation must be addressed through techniques like dataset filtering, human-in-the-loop systems, enhanced monitoring, and immediate remediation.

As generative AI continues maturing, best practices and governance frameworks must evolve in tandem. Industry leaders have formed partnerships like the Content Authenticity Initiative to develop technical standards and policy guidance around the responsible development of the next iteration of AI. This technology's incredible potential, from accelerating drug discovery to envisioning new products, can only be realized through a commitment to transparency, ethics, and human rights. Constructive collaboration that balances innovation with caution is imperative.

Generative AI marks an inflection point for the field. The ripples from this groundswell of creative possibility are just beginning to reach organizations and communities. Maintaining an open, evidence-driven dialogue around not just capabilities but also challenges lays a foundation for AI deployment that empowers people, unlocks new utility, and earns widespread trust.

We are witnessing an unprecedented democratization of generative AI capabilities through publicly accessible APIs from established companies like Google, Meta, and Amazon, and startups such as Anthropic, Mistral AI, Stability AI, and OpenAI. The table below summarizes several leading models that provide versatile foundations for natural language and image generation.

Just a few years ago, developing with generative AI required specialized expertise in deep learning and access to vast computational resources. Now, models like Gemini, Claude, GPT-4, DALL-E, and Stable Diffusion can be accessed via simple API calls at near-zero cost. The bar for experimentation has never been lower.

This commoditization has sparked an explosion of new applications leveraging these pre-trained models – from creative tools for content generation to process automation solutions infused with AI. Expect integrations with generative foundations across all industries in the coming months and years.

Models are becoming more knowledgeable, with broader capabilities and reasoning that will reduce hallucinations and increase accuracy across model responses. Multimodality is also gaining traction, with models able to ingest and generate content across text, images, audio, video, and 3D scenes. In terms of scalability, model size and context windows continue expanding exponentially; for example, Google's Gemini 1.5 now supports a context window of 1 million tokens.

Overall, the outlook points to a future where generative AI will become deeply integrated into most technologies. These models introduce new efficiencies and automation potential and inspire creativity across nearly every industry imaginable.

The table below highlights some of the most popular LLMs and their providers. The purpose of the table is to highlight the vast number of options available on the market at the time of writing this book. We expect this table to quickly become outdated by the time of publication and highly encourage readers to dive deep into the model providers' websites to stay up to date with any new launches.

Model	**Provider**	**Landing Page**
Gemini	Google	`https://deepmind.google/technologies/gemini`
Claude	Anthropic	`https://claude.ai/`
ChatGPT	OpenAI	`https://openai.com/blog/chatgpt`
Stable Diffusion	Stability AI	`https://stability.ai/`
Mistral	Mistral AI	`https://mistral.ai/`
LLaMA	Meta	`https://llama.meta.com/`

Table 1.1: Overview of popular LLMs and their providers

Predictive AI vs generative AI use case ideation

Predictive AI refers to systems that analyze data to identify patterns and make forecasts or classifications about future events. In contrast, generative AI models create new synthetic content like images, text, or code based on the patterns gleaned from their training data. For example, with predictive AI, you can confidently identify if an image contains a cat or not, whereas with generative AI you can create an image of a cat from a text prompt, modify an existing image to include a cat where there was none, or generate a creative text blurb about a cat.

Product innovation focused on AI involves various phases of the product development lifecycle. With the emergence of generative AI, the paradigm has shifted away from initially needing to compile training data to train traditional ML models and toward leveraging flexible pre-trained models.

Foundational models like Google's PaLM 2 and Gemini, OpenAI's GPT and DALL-E, and Stable Diffusion provide broad foundations enabling rapid prototype development. Their versatile capabilities lower the barrier for experimenting with novel AI applications.

Where previously data curation and model training from scratch could take months before assessing viability, now proof-of-concept generation is possible within days without the need to fine-tune a foundation model.

This generative approach facilitates more iterative concept validation. After quickly building an initial prototype powered by the baseline model, developers can then collect niche training data and perform knowledge transfer via techniques like distillation to customize later versions; we will deep dive into the concept of distillation later in the book. The model's primary foundation contains already encoded patterns useful for kickstarting and for iterations of new models.

In contrast, the predictive modeling approach requires upfront data gathering and training before any application testing. This more linear progression limits early-stage flexibility. However, predictive systems can efficiently learn specialized correlations and achieve a high level of confidence inference metrics once substantial data exists.

Leveraging versatile generative foundations supports rapid prototyping and use case exploration. But, later, custom predictive modeling boosts performance on narrow tasks with sufficient data. Blending these AI approaches capitalizes on their complementary strengths throughout the model deployment lifecycle.

Beyond the basic use – prompt engineering – of a foundational model, several auxiliary, more complex techniques can enhance its capabilities. Examples include **Chain-of-Thought** (**CoT**) and **ReAct**, which empower the model to not only reason about a situation but also define and evaluate a course of action.

ReAct, presented in the paper *ReAct: Synergizing Reasoning and Acting in Language Models* (`https://arxiv.org/abs/2210.03629`), addresses the current disconnect between LLMs' language understanding and their ability to make decisions. While LLMs excel at tasks like comprehension and question answering, their reasoning and action-taking skills (for example, generating action plans or adapting to unforeseen situations) are often treated separately.

ReAct bridges this gap by prompting LLMs to generate both "reasoning traces," detailing the model's thought process, and task-specific actions in an interleaved manner. This tight coupling allows the model to leverage reasoning for planning, execution monitoring, and error handling, while simultaneously using actions to gather additional information from external sources like knowledge bases or environments. This integrated approach demonstrably improves LLM performance in both language and decision-making tasks.

For example, in question-answering and fact-verification tasks, ReAct combats common issues like hallucination and error propagation by utilizing a simple Wikipedia API. This interaction allows the model to generate more transparent and trustworthy solutions compared to methods lacking reasoning or action components. LLM hallucinations are defined as content generated that seems plausible yet factually unsupported. There are various papers that aim to address this phenomenon. For example, *A survey of Hallucination in Large Language Models – Principles, Taxonomy, Challenges, and Open Questions* deep dives into an approach to not only identify but also mitigate hallucinations. Another good example of a mitigation technique is covered in the paper *Chain-of-Verification Reduces Hallucination in Large Language Models* (`https://arxiv.org/pdf/2309.11495.pdf`). At the time of writing this book, hallucinations are a very rapidly changing field.

Both CoT and ReAct rely on prompting: feeding the LLM with carefully crafted instructions that guide its thought process. CoT, as presented in the paper *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models* (`https://arxiv.org/abs/2201.11903`), focuses on building a chain of reasoning steps, mimicking human thinking. Imagine prompting the model with: "I want to bake a cake. First, I need flour. Where can I find some?" The model responds with a potential source, like your pantry. This back-and-forth continues, building a logical chain of actions and decisions.

ReAct takes things a step further, integrating action into the reasoning loop. Think of it as a dynamic dance between thought and action. The LLM not only reasons about the situation but also interacts with the world, fetching information or taking concrete steps, and then updates its reasoning based on the results. It's like the model simultaneously planning a trip and checking maps to adjust the route if it hits a roadblock.

This powerful synergy between reasoning and action unlocks a new realm of possibilities for LLMs. CoT and ReAct tackle challenges like error propagation (jumping to the wrong conclusions based on faulty assumptions) by allowing the model to trace its logic and correct course. They also improve transparency, making the LLM's thought process clear and understandable.

In other words, **large language models (LLMs)** are like brilliant linguists, adept at understanding and generating text. But when it comes to real-world tasks demanding reasoning and action, they often stumble. Here's where techniques like CoT and ReAct enter the scene, transforming LLMs into reasoning powerhouses.

Imagine an LLM helping diagnose a complex disease. CoT could guide it through a logical chain of symptoms and examinations, while ReAct could prompt it to consult medical databases or run simulations. This not only leads to more accurate diagnoses but also enables doctors to understand the LLM's reasoning, fostering trust and collaboration.

These futuristic applications are what drive us to keep building and investing in this technology, which is very exciting. Before we dive deep into the patterns that are needed to leverage generative AI technology to generate business value, let's take a step back and look at some initial concepts.

A change in the paradigm

It feels like eons ago in tech years, but let's rewind just a couple of years, back when if you were embarking on solving an AI problem, you couldn't default to utilizing a pre-trained model through the web or a managed endpoint. The process was meticulous – you'd have to first clearly define the specific use case, identify what data you had available and could collect to train a custom model, select the appropriate algorithm and model architecture, train the model using specialized hardware and software, and validate if the outputs would actually help solve the task at hand. If all went well, you would have a model that would take a predefined input and also provide a predefined output.

The paradigm profoundly shifted with the advent of LLMs and large multimodal models. Suddenly, you could access a pre-trained model with billions of parameters and start experimenting right off the bat with these versatile foundational models where the inputs and outputs are dynamic in nature. After tinkering around, you'd then evaluate if any fine-tuning is necessary to adapt the model to your needs, rather than pre-training an entire model from scratch. And spoiler alert – in most cases, chances are you won't even need to fine-tune a foundational model.

Another key shift relates to the early belief that one model would outperform all others and solve all tasks. However, the model itself is just the engine; you still need an entire ecosystem packaged together to provide a complete solution. Foundational models have certainly demonstrated some incredible capabilities beyond initial expectations. But we also observe that certain models are better suited for certain tasks. And running the same prompt through other models can produce very different outputs depending on the underlying model's training datasets and architecture.

So, the new experimental path often focuses first on prompt engineering, response evaluation, and then fine-tuning the foundational model if gaps exist. This contrasts sharply with the previous flow of data prep, training, and experimentation before you could get your hands dirty. The bar to start creating with AI has never been lower.

In the following sections, we will explore the difference between the development lifecycle of predictive AI and generative AI use cases. In each section, we have provided a high-level visual representation of a simplified development lifecycle and an explanation of the thought process behind each approach.

Predictive AI use case development – simplified lifecycle

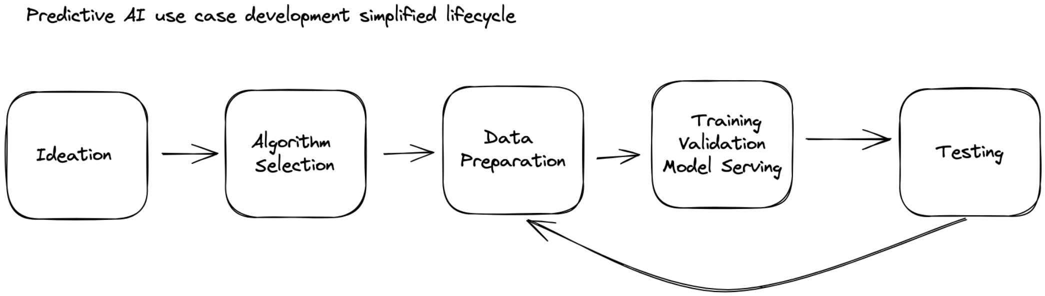


Figure 1.1: Predictive AI use case development simplified lifecycle

Let's dive into the process of developing a predictive AI model first. Everything starts with a good use case, and ROI (return on investment) is top of mind when evaluating AI use cases. Think about pain points in your business or industry that could be solved by predicting an outcome. It is very important to always keep an eye on feasibility – whether you can procure the data you need, etc.

Once you've landed on a compelling value-driven use case, next up is picking algorithms. You've got endless options here – decision trees, neural nets, regressions, random forests, and on and on. It is very important not to be swayed by the bias for the latest and greatest and to focus on the core requirements of your data and use case to narrow the options down. You can always switch it up or add additional experiments as you iterate through your testing.

With a plan in place, now it is time to get your hands dirty with the data. Identifying sources, cleaning things up, and carrying out feature engineering is an art and, more often than not, the key to improving your model's results. There is no shortcut for rigor here, unfortunately! Garbage in, garbage out, as they say. But once you've wrangled datasets you can rely on, then comes the fun part.

It's time to work with your model. Define your evaluation process upfront, split data wisely, and start training various configurations. Don't forget to monitor and tune based on validation performance. Then, once you've got your golden model, implement robust serving infrastructure so it scales without a hitch.

But wait, not so fast! Testing doesn't end when models are in production. Collect performance data continuously, monitor for concept drifts, and retrain when needed. A solid predictive model requires ongoing feedback mechanisms, as shown via the arrow connecting **Model Enhancement** to **Testing** in *Figure 1.1*. There is no such thing as set and forget in this space.

Generative AI use case development – simplified lifecycle

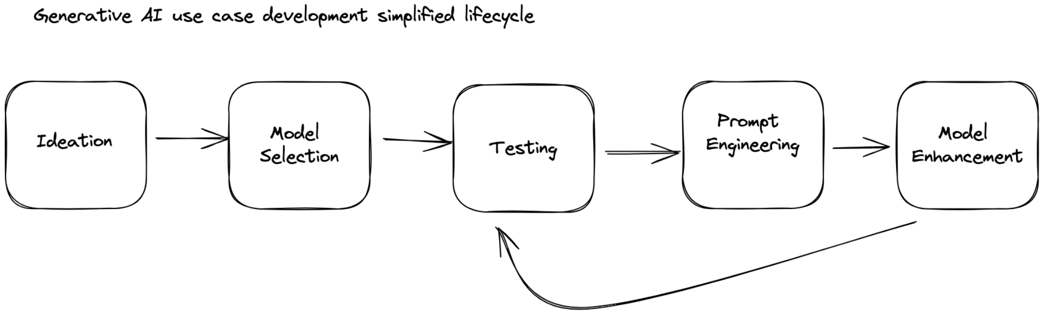


Figure 1.2: Generative AI use case development simplified lifecycle

The process of generative AI use case development is similar but not the same as in predictive AI; there are some common steps, but the order of tasks is different.

The first step is the ideation of potential use cases. This selection needs to be balanced with business needs as satisfying them is our main objective.

With a clear problem definition in place, extensive analysis of published model benchmarks often informs the selection of a robust foundational model best suited for the task. In this step, it is worth asking ourselves the question is this use case better suited for a predictive model?

As foundational models provide capabilities out of the box, initial testing comes as a step early in the process. A structured testing methodology helps reveal innate strengths, weaknesses, and quirks of a specific model. Both quantitative metrics and qualitative human evaluations fuel iterative improvement throughout the full development lifecycle.

The next step is to move to the art of prompt engineering. Prompting is the mechanism used to interact with LLMs. Techniques like chain-of-thought prompting, skeleton prompts, and retrieval augmentation build guardrails enabling more consistent, logical outputs.

If gaps remain after prompt optimization, model enhancement via fine-tuning and distillation offers a precision tool to adapt models closer to the target task.

In rare cases, pretraining a fully custom model from scratch is warranted when no existing model can viably serve the use case. However, it is important to keep in mind that due to the massive data requirements posed by model retraining, this task won't be suitable for most use cases and teams; retraining a foundational model requires an extensive amount of data and processing power that makes the process unpractical from a financial and technical perspective.

Above all, the interplay between evaluation and model improvement underscores the deeply empirical nature of advancing generative AI responsibly. Testing often reveals that better solutions come from creativity in problem framing rather than pure technological advances alone.

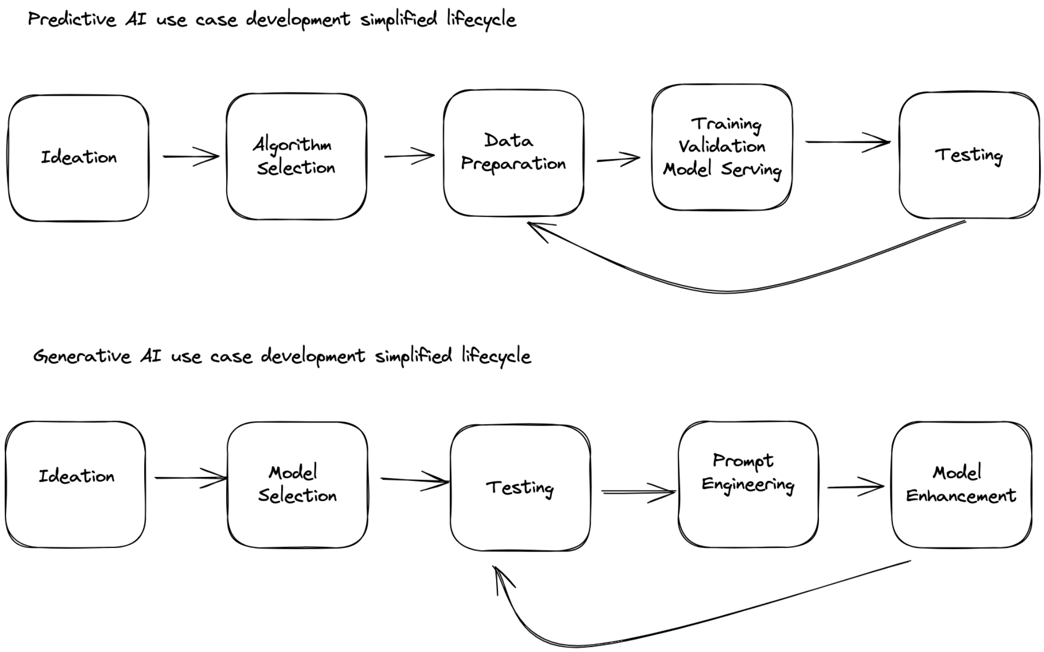


Figure 1.3: Predictive and generative AI development lifecycle side-by-side comparison

As we can see from the preceding figure, the development lifecycle is an iterative process that enables us to realize value from a given use case and technology type. Across the rest of this chapter and this book, we are going to focus on generative AI general concepts, some that are going to be familiar if you are experienced in predictive AI and others that are specific to this new field in AI.

General generative AI concepts

When integrating generative AI into practical applications, it is important to have an understanding of concepts such as model architecture and training. In this section, we cover an overview of prominent concepts, including transformers, diffusion models, pre-training, and prompt engineering, that enable systems to generate impressively accurate text, images, audio, and more.

Understanding these core concepts will equip you to make informed decisions when selecting foundations for your use cases. However, putting models into production requires further architectural considerations. We will be highlighting these decision points in the rest of the chapters in the book and in practical examples.

Generative AI model architectures

Generative AI models are based on specialized neural network architectures optimized for generative tasks. The two more widely known models are transformers and diffusion models.

Transformer models are not a new concept. They were first introduced by Google in a 2017 paper called *Attention Is All You Need* (`https://arxiv.org/pdf/1706.03762.pdf`). The paper explains the **Transformer neural network architecture**, which is entirely based on attention mechanisms using the encoder and decoder concepts. This architecture enables models to identify relationships across an input text. By having these relationships, the model predicts the next token, leveraging its previous prediction as an input, creating this recursive loop to generate new content.

Diffusion models have drawn considerable interest as generative models due to their foundation in the physical processes of non-equilibrium thermodynamics. In physics, diffusion refers to the motion of particles from areas of high concentration to low concentration over time. Diffusion models try to mimic this concept in their training process. These models are trained through two phases: the **forward diffusion** process adds "noise" to the original training data, followed by a **reverse conditioning** process, which then learns how to remove noise in the reverse diffusion process. By learning this process, these models can produce samples by starting from pure noise and letting the reverse diffusion model clear away unnecessary "noise" and preserving the desired "generated" content.

Other types of deep learning architectures, such as **Generative Adversarial Networks** (**GANs**), allow you to generate synthetic data based on existing data. GANs are useful because they leverage two models: one to generate a synthetic output and another one that tries to predict if this output is real or fake.

Through this iterative process, we can generate data that is indistinguishable from the real data but different enough to be used to enhance our training datasets. Another example of data generation architectures is **Variational Autoencoders (VAEs)**, which use an encoder-decoder approach to generate new data samples resembling their training datasets.

Techniques available to optimize foundational models

There are several techniques used to develop and optimize foundational models that have driven significant gains in AI capabilities, some of which are more complex than others from a technical and monetary perspective:

- **Pre-training** refers to fully training a model on a large dataset. It allows models to learn very broad representations from billions of data points, which help the model adapt to other closely related tasks. Popular methods include contrastive self-supervised pre-training on unlabeled data and pre-training on vast supervised data like the internet.
- **Fine-tuning** adapts a pre-trained model's already learned feature representations to perform a specific task. This only tunes some higher-level model layers rather than training from scratch. On the other hand, adapter tuning equips models with small, lightweight adapters that can rapidly tune to new tasks without interfering with existing capabilities. These pluggable adapters give a parameter-efficient way of accumulating knowledge across multiple tasks by learning task-specific behaviors while reusing the bulk of model weights. They help mitigate forgetting previous tasks and simplify personalization. For example, models may first be pre-trained on billions of text webpages to acquire general linguistic knowledge, before being fine-tuned on more domain-specific datasets for question answering, classification, etc.
- **Distillation** uses a "teacher" model to train a smaller "student" model to reproduce the performance of the larger pre-trained model at a lower cost and latency. Quantizing and compressing large models into efficient forms for deployments also helps optimize performance and cost.

The combination of comprehensive pre-training followed by specialized fine-tuning, adapter tuning, and portable distillation has enabled unprecedented versatility of deep learning across domains. Each approach smartly reuses and transfers available knowledge, enabling the customization and scaling of generative AI.

Techniques to augment your foundational model responses

In addition to architecture and training advances, progress in generative AI has been fueled by innovations in how these models are augmented by external data at inference time.

Prompt engineering tunes the text prompts provided to models to steer their generation quality, capabilities, and properties. Well-designed prompts guide the model to produce the desired output format, reduce ambiguity, and provide helpful contextual constraints. This allows simpler model architectures to solve complex problems by encoding human knowledge into the prompts.

Retrieval augmented generation, also known as **RAG**, enhances text generation through efficient retrieval of relevant knowledge from external stores. Models receive contextual pieces of information as “context” to be considered as additional input before generating its output. **Grounding** LLMs (large language models) refers to providing model-specific factual knowledge rather than just model parameters, enabling more accurate, knowledgeable, and specific language generation.

Together, these approaches augment basic predictive language models to become far more versatile, robust, and scalable. They reduce brittleness via tight integration of human knowledge and grounded learning rather than just statistical patterns. RAG handles the breadth and real-time retrieval of information, prompts provide depth and rules to the desired outputs, and grounding binds them to reality. We would highly encourage readers to get familiar with this topic, as it is an industry best practice to perform RAG and to ground your model to prevent it from hallucinating. A good start is the following paper: *Retrieval-Augmented Generation for Large Language Models: A Survey* (`https://arxiv.org/pdf/2312.10997`).

Constant evolution across the generative AI space

The generative AI space is characterized by relentless innovation and rapid advancement across model architectures, applications, and ethical considerations. As soon as one method or architecture shows promising results, hundreds of competing and complementary approaches emerge to push capabilities even further. Transformers gave way to BERT, which was outpaced by GPT-3, soon rivaled by image synthesizers like DALL-E, and now GPT-4 and Gemini are competing for the top spot. All of which happened in the past few years.

Meanwhile, we are seeing new modalities like audio, video, and 3D scene generation gaining vast popularity and usability. On the business front, new services are launched monthly, targeting media and entertainment, finance, healthcare, art, code, music, and more. However, considerations around ethics, access control, and legalities are key in order to maintain public trust.

One breakthrough enables several more, and each unlocks added potential. This self-fueling cycle arises from the very nature of AI – its ability to recursively assist innovation. The only certainty is that the field will look very different within months, not years. Maintaining awareness, responsiveness, and responsibility is critical amid this constant evolution.

Introducing generative AI integration patterns

Let's now assume you already have a promising use case in mind. As I'm sure you would agree, clearly defining the use case is critical before proceeding further. You've already identified which foundational model provides acceptable performance for your needs. So now you're starting to consider how GenAI fits into the application development process.

At a high level, there are two main workflows for integrating applications with GenAI. One is **real time**, where you'll typically interact with an end user or AI agent, providing responses as prompts come in. The second is **batch processing**, where requirements are bundled up and processed in groups (batches).

A prime example of a real-time workflow would be a chatbot. Here, prompts from the user are processed and then sent to the model and the responses are returned immediately, as you need to consume the outputs without delay. On the other hand, consider a data enrichment use case for batch processing. You could collect multiple data points over time for later consumption after being enriched by the model in batches.

In this book, we will explore these integration patterns through practical examples. This will help you to obtain hands-on experience with GenAI-driven applications and allow you to integrate these patterns in your own use cases.

By "integration pattern," we refer to a standardized architectural approach for incorporating a technology into your application or system. In this context, integration patterns provide proven methods for connecting generative AI models to real-world software.

There are a few key reasons why we need integration patterns when working with generative AI:

- **Time savings**: Following established patterns allows developers to avoid reinventing the wheel for common integration challenges. This accelerates time to value.
- **Improving quality**: Leveraging best practices encoded in integration patterns leads to more robust, production-grade integrations. Things like scalability, security, and reliability are top of mind.

- **Reducing risk**: Well-defined integration patterns enable developers to mitigate risks around performance, costs, and other pitfalls that can emerge when integrating new technologies.

Overall, integration patterns deliver templates and guardrails, so developers don't have to start integration efforts from scratch. By relying on proven blueprints, readers can integrate generative AI more efficiently while avoiding common mistakes. This speeds up development cycles significantly and sets integrations up for long-term success.

Summary

In this chapter, we covered an overview of key concepts, techniques, and integration patterns related to generative AI. You now have a high-level background on prominent generative AI model architectures like transformers and diffusion models, as well as various methods for developing and enhancing these models, covering pre-training, fine-tuning, adapter tuning, distillation, prompt engineering, retrieval augmented generation, and grounding.

We discussed how rapid innovation in generative AI leads to constant evolution, with new models and capabilities emerging at a fast pace. It emphasizes the need to keep pace with progress while ensuring ethical, responsible development.

Finally, we introduced common integration patterns for connecting generative AI to real-world applications, considering real-time use cases like chatbots as well as batch processing for data enrichment. Real examples were provided to demonstrate workflows for integrating generative models into production systems.

Innovation in AI has a very fast pace, demanding constant awareness, swift experimentation, and a responsible approach to harnessing the latest advances. This is particularly evident in the field of generative AI, where we're witnessing a paradigm shift in AI-powered applications that allows for faster experimentation and development.

A wide array of techniques has emerged to enhance models' capabilities and efficiency. These include pre-training, adapter tuning, distillation, and prompt engineering, each offering unique advantages in different scenarios. When it comes to integrating these AI models into practical applications, key patterns have emerged for both real-time workflows, such as chatbots, and batch processing tasks like data enrichment.

The art of crafting well-designed prompts has become crucial in constraining and steering model outputs effectively. Additionally, techniques like retrieval augmentation and grounding have proven invaluable in improving the accuracy of AI-generated content. The potential in blending predictive and generative approaches is a very interesting space. This combination leverages the strengths of both methodologies, allowing for custom modeling where sufficient data exists while utilizing generative foundations to enable rapid prototyping and innovation.

These core concepts empower informed decision-making when architecting generative AI systems. The integration patterns offer blueprints for connecting models to practical applications across diverse domains.

Harnessing the power of LLMs begins with identifying the right use cases where they can drive value for your business. In the next chapter, we will present a framework and examples for categorizing LLM use cases based on projected business value.

In the next chapter, we will explore identifying use cases that can be solved with Generative AI.

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

`https://packt.link/genpat`

2

Identifying Generative AI Use Cases

In the previous chapter, we discussed how the use case exploration and proof of concept development process has shifted with the emergence of **large language models** (**LLMs**). Specifically, the ability for rapid experimentation with LLMs has led teams to favor an experimental approach over more traditional requirements analysis and design processes.

With LLMs, use cases can quickly be tested by creating prompts that demonstrate potential capabilities. This allows for greater flexibility and speed than meticulously mapping detailed requirements upfront. Of course, once promising use cases are identified, more rigorous analysis is highly recommended. Additionally, security, monitoring, and governance of production systems remain critical components.

In this chapter, we will discuss an approach to identify promising use cases to explore with LLMs. We categorize use cases based on how an application interacts with the LLM. This provides a framework to think through the breadth of possibilities as well as the unique considerations for implementation.

We will cover the following main topics:

- When to consider **Generative AI** (**GenAI**) as a tool to integrate into your application
- Realizing business value through GenAI
- How to identify GenAI business use cases
- The difference between Comprehensive and Generative use cases along with detailed examples

When to consider generative AI

As we have been exploring, one of the powers of GenAI is the ability to automatically generate responses without being explicitly trained on it. Rather than just executing predefined tasks, LLMs can infer responses by drawing on their contextual understanding and knowledge. This aspect of emergent reasoning unlocks unique opportunities for rapid experimentation and iterative refinement of novel use cases.

When considering potential applications for GenAI, the first evaluation criterion centers on comprehension-based tasks. Sentiment analysis, content classification, intent classification, relationship extraction, summarization, and more all leverage innate language understanding. Developers can formulate prompts aligning to use cases that interpret, organize, or infer meaning. To unlock the full potential of LLMs, developers will iterate on these given prompts through thoughtful "prompt engineering." Prompt engineering attempts to optimize LLM responses by providing task-specific input text that guides a model toward the desired output.

However, purely numerical analysis may not be the best initial fit for GenAI. While mathematical reasoning exists within models, large volumes of statistical data processing are better suited for traditional programmatic algorithms and predictive AI. Of course, GenAI could help describe insights from numeric analysis – communicating trends in natural language, for example, or creating queries from natural language. But we wouldn't expect strong performance by running regression analysis or optimization from prompts alone.

Along these lines, one of the earliest discoveries around the limitations of LLMs surfaced in mathematical reasoning. Users experimenting with prompts that involved numeric calculations or comparisons found nonsensical outputs. The models would produce response text that sounded coherent but lacked any grounding in basic arithmetic principles. This disconnect highlights the risk of hallucinations – responses that have fluent language but little accuracy or logical consistency.

Researchers formulated that the enormous parameter spaces of LLMs allow them to optimize textual cohesion while lacking the tighter constraints of symbolic logic found in math. Without specifically encoding numeric logic, the models "hallucinate" plausible-sounding numeric reasoning that mathematically makes little sense. The outputs expose both the power of language fluency and the risk of generalizing beyond the actual knowledge limitations of models.

As we dive deeper into use cases in this book, we encourage you to brainstorm opportunities aligned with the strengths provided by language understanding tools.

The goal is to match high-value challenges around language, content, and reasoning with the emergent capabilities of LLMs. We will evaluate where modern AI could augment human workflow – is there a comprehension component that bogs us down? Could prompts help classify, summarize, suggest, or predict within those contexts? And finally, we will factor in how the outputs generated by LLMs could evolve from experimental prompting into a refined API-driven solution.

It is important to remark one more time on the importance of business alignment. When emerging technologies garner tremendous hype and media attention, organizations face the tendency to rush to deploy "shiny new toys" without clearly defining the value they will drive for a business. GenAI risks this same dynamic, given the incredible mainstream breakout prompted by chatbots like ChatGPT. Executives clamor to stake their claim in using these powerful technological advances to future-proof competitiveness.

Unfortunately, treating AI as an end solution rather than an ingredient to enhance solutions often leads to failure. Well-intentioned teams demo flashy prototype capabilities that fail to map into tangible business challenges or workflows. Progress stalls beyond experiments in isolation. Funding dries up without demonstrating a real-world impact. Even worse, poorly planned AI integration risks harming the customer experience or market position. Conversely, focusing on use cases with clear business value upfront fosters successful AI implementations, leading to a positive business impact while driving innovation and maximizing the return on investment.

Figure 2.1: "The Symphony of Data and Imagination" – a generated image to illustrate generative AI vs Predictive AI

Realizing business value

When assessing opportunities to solve business problems with GenAI, it's critical that we analyze potential value in business terms – how does this technology shift affect structures, efficiency, competitive positioning, or revenue opportunities in an organization? Merely showcasing sophisticated technical capabilities alone is not enough to show value, as it lacks strategic rigor. Proposed projects should directly address tangible problems or sources of organizational leverage.

As thought leaders seeking executive buy-in, we need to develop skills and methodologies that translate potential productivity gains into compelling business cases, clarifying the return on investment. Specifically, constructing cost avoidance models resonates with executives focused on operational efficiencies and margins.

Quantifying assumptions in terms that the business can understand and measure is an essential step to a successful integration of GenAI into your applications. What are the current costs for manual processes targeted for automation? How many labor hours or **full-time employees (FTEs)** are dedicated to inefficient areas? How might throughput scale with AI augmentation? What expenses link to quality defects or customer friction pain points?

Understanding and extracting metrics to translate into data-driven projections is key. Gathering baseline measurements around the current state of business operations sets the foundation to showcase future improvements. Potential vectors to quantify gains across use cases include:

- **Cost savings**: Hours recovered through automation and scalability, and error and rework reduction.
- **Revenue increments**: Improved and increased customer lifetime value from engagement and satisfaction.
- **Quality improvements**: Reduced error rates and higher review scores, measuring customers' willingness to recommend.
- **Employee experience**: Internal surveys on satisfaction, productivity, and meaningful and engaging work.

The goal is framing a business case in executive-friendly models that also enable the tracking of real-world impact versus projections. Building analytical muscle and intuition to translate AI advances into operational and financial outcomes takes some skill development, but it pays off through better alignment across teams.

Let's think of an example of a content tagging use case. Manually assigning metadata to documents puts a strain on human reviewers attempting volume at scale. Mistakes occur, inconsistencies pop in, and data attrition occurs – chaos. But perhaps a **natural language processing** (**NLP**) pipeline could help automate tagging to boost throughput dramatically while lowering costs. This pipeline then becomes an enabler for human reviewers to scale at a faster pace, allowing them to focus on reviewing rather than creating.

A revenue generating use case might focus more on using those rich tagged corpora as training data for client recommendations and understanding buying signals in a given market. Competitively, rich catalog search and personalization at scale differentiate against slower-moving players that only produce static signals. So, smart framing of use cases ties directly into strategic priorities, painting a compelling vision of market leadership.

The antidote to picking between efficiency gains and revenue regenerating use cases requires framing every potential AI application in terms of specific business value drivers first and enabling technologies second. What key objectives around cost, efficiency, differentiation, or revenue might we use? Where do humans currently struggle with workload, subjectivity, or availability? Do processes fail to scale sufficiently because of volume or complexity? Anchoring use cases firmly around moving these business needs provides the appropriate context to responsibly hone where and how GenAI can help. Avoid the instinct to create problems and find tools to fix them. Thoroughly map problems and then find the tools to fix them.

In essence, GenAI represents ingredients that enable business capabilities rather than an end solution itself. We must thoughtfully apply technology to priority challenges or opportunities. If the **return on investment** (**ROI**) in applying GenAI feels tenuous or the business value generated is too small, broader thinking is likely required before further investment.

Identifying Generative AI use cases

As we evaluate potential applications for GenAI, two overarching categories emerge – **comprehension applications** and **generative applications**. By distinguishing use cases along this spectrum of understanding existing data vs synthesizing new artifacts, we can better match capabilities to appropriate business challenges:

- Comprehension applications cover processing tasks applied to analyze and structure knowledge from existing content. This includes sentiment analysis, relationship extraction, intent classification, summarization, and more. The key focus areas center around interpreting, organizing, and tagging data to be used by subsequent systems.

Sources like chat history, customer tickets, and product catalogs yield richer insights when machine learning models classify topics, normalize entities, and summarize concepts at scale.

- Generative applications move beyond pure comprehension applications to creatively synthesize new artifacts like text, images, or multimedia. Use cases range from content drafting and ideation to conversational interfaces. With the right prompts and fine-tuning, LLMs can produce natural language, code, visual designs, and more for specific domains. Outputs integrate with customer-facing solutions or augment human workflows.

Blending both categories, we find an incredible opportunity in hybrid human-AI collaboration. Humans contextualize business objectives, provide critical oversight, and handle exceptions. AI automates high-volume tasks while sending signals for people to interpret. For example, generative writing aids content creators through initial drafting while expert editors finalize the content.

By organizing use cases into comprehension and generative buckets, we can better map capabilities to potential applications based on desired outcomes. Comprehension-focused use cases center around deriving insights from existing data. Generative use cases create novel artifacts by building on learned patterns. Both transform workflows when applied thoughtfully, upholding responsibilities around monitoring, ethics, and team augmentation.

With recent advances in multimodal models that understand connections across images, text, and other data types, new dimensions of AI use cases are surfacing. Google's Gemini model (see the paper at `https://arxiv.org/abs/2312.11805`) combines comprehension applications across visual, textual, mathematical, and even code concepts within a unified underlying architecture.

This multi-modal foundation enables Gemini to not only interpret multimodal information but also generate novel artifacts like images from text prompts, and vice versa. For comprehension applications, this offers intriguing use cases related to visually summarizing documents, generating images from written narratives to visually interpret concepts, and enhancing images based on descriptive captions. The generative application possibilities include ideating creative images, designs, and data visualizations tailored to specific conceptual directions. This aligns with the hybrid human/AI collaboration model.

Images, video, and voice media types introduce yet another set of rich content that multimodal models are becoming very good at processing and understanding to perform generative tasks. The comprehension of scenes, objects, speech, and overlaid text within videos enhances applications like search, recommendation, and content moderation.

Other practical examples include extracting text from images and videos, generating new content within video frames or images, and even creating comprehensive summaries describing the contents of rich media content. As with other AI advances, responsible oversight remains critical as multimodal models start permeating workflows. Continued monitoring, ethical implementation, and human judgment help steer us toward beneficial outcomes as this technology unlocks new potential. But by connecting media like images, text, and sound, models such as Gemini pave the way for the next generation of intelligent applications, augmented with a deeper understanding across multiple dimensions.

Potential business-focused use cases

When exploring opportunities to apply GenAI, continuously evaluate any potential business value first rather than just the technical art of the possible. To spark ideas, the following list summarizes promising use cases organized by key value drivers. Consider cases where replicating human-quality comprehension or creation at a machine scale has a positive impact on workflows.

Cost and efficiency gains:

- Automate high-volume/repetitive tasks. Here, your business **key performance indicator (KPI)** can be an automation rate, measured as a percentage of automated tasks.
- Accelerate content development, enabling hybrid human/AI collaboration. Here, your business KPI can be an increase in the number of hours saved compared to the current amount of time taken to create content.
- Content evaluation to reduce errors and rework. Here, your business KPI can be the delta in the number of tickets filed to update incorrect content.

Personalize and tailor recommendations:

- Generate consistent omnichannel experiences by leveraging image/text generation. Here, your business KPI could be an existing engagement metric such as "time spent on a platform," or the click-through rate on a given section of your application.
- Generate contextual cues to guide users and improve product documentation. Here, your business KPI can be the engagement of your customers with your documentation, which can be measured as page views.
- Generate operational insights to enhance risk reduction. Here, your business KPI could be the delta on risk assessments.

Enhance and scale human interaction:

- Provide overview summaries from documents. Here, your business KPI could be an existing engagement metric such as "time spent on a platform," or the click-through rate on a given section of your application.
- Forecast emerging issues and trends. Here, your business KPI could be the predictive power of your application as it relates to saving costs.
- Collaborate for innovation and gain competitive advantage. Here, your business KPI could be an existing engagement metric such as "time spent on platform," or the click-through rate on a given section of your application.

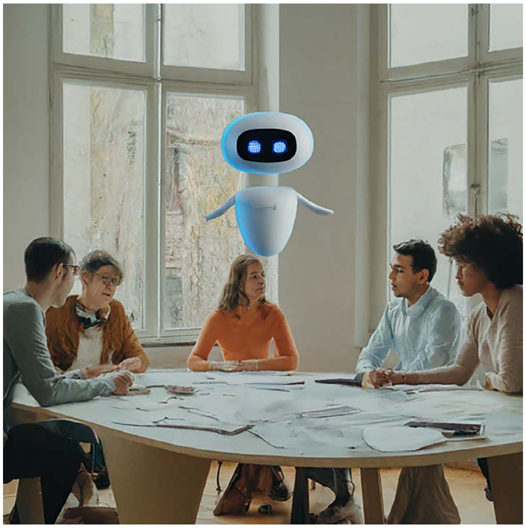


Figure 2.2: "Brainstorming Buddies" image generated by AI

This framing of use cases aligned to core business priorities keeps implementation closely bound to driving real outcomes, rather than deploying technology for its own sake. Many possibilities abound at the intersection of business needs and AI capabilities, but maintaining this value-centric focus ensures the responsible application of AI, guiding teams toward beneficial innovation and transformation.

Comprehension use cases harness the power of natural language understanding to extract insights and structured information from unstructured data. These techniques enable organizations to gain valuable insights, improve discoverability, and facilitate knowledge rediscovery and reuse, ultimately driving better decision-making and enhancing operational efficiency.

The following is a non-exhaustive list of technically focused comprehension use cases:

- **Sentiment analysis**: Sentiment analysis leverages natural language understanding to automatically classify the emotional tone within textual content, like customer feedback, surveys, and social media posts. This allows organizations to identify pain points and perceptions without a large-scale manual review. Common integration strategies include sentiment API queries or batch processing analytics aggregated into reporting dashboards. This use case focuses on understanding how customers perceive your business and identifying improvement opportunities.
- **Document summarization**: Document summarization uses AI to automatically create condensed snippets of text, bringing key details within lengthy corpora, like wikis, research papers, and knowledge base articles, to the surface. This improves discoverability for users who can quickly determine the relevance of a piece of text before deciding to read full documents. It also enables new modes of documents interaction and searchability, especially across massive repositories. This use case focuses on increasing productivity across the board.
- **Metadata extraction**: Metadata extraction harnesses natural language understanding to identify and extract key information attributes from unstructured textual content. This includes entities like people, places, and companies, as well as topics, concepts, tone, and relationships. Structured metadata makes understanding documents and content much easier at scale.

Use cases range from processing volumes of contracts to auto-tagging with counterparties, obligations, dates, or risk levels. Support ticket classification with topics, priority status, and location can route cases correctly while pinpointing macro trends. Aggregating research publications within a field by the concepts and techniques mentioned builds network maps for emerging schools of thought and impact tracking. Publishing platforms can even auto-suggest high-performance article tags to analyze headline semantics and document summaries.

Metadata extraction enables knowledge rediscovery and reuse in large organizations. For example, employees can quickly search on past meeting notes, referencing to key projects details versus leveraging siloed, fragmented data stores.

New team members can get up to speed, discovering pivotal documents from intelligent graphs rather than performing aimless word searches. Over time, signal-to-noise ratios improve as AI augments institutions' contextual understanding of their information universe, based on this actionable metadata layer.

Conversely, generative use cases leverage GenAIs ability to generate human-like outputs, tailored to specific needs and contexts. These applications not only streamline repetitive tasks but also open new avenues for personalized experiences, enhanced creativity, and accelerated innovation.

The following is a non-exhaustive list of technically focused generative use cases:

- **Conversational interfaces**: Conversational interfaces allow natural dialogue between end users and intelligent assistants via chat, voice, and, potentially, **augmented reality** (**AR**). These fluid experiences provide answers, recommendations, and next-step suggestions, reducing the need to navigate complex apps or menus. Over time, contextual awareness of user goals and preferences enables personalized guidance.
- **Data visualization**: Data visualization uses GenAI to automatically create relevant charts, graphs, and diagrams, tailored to provided datasets. Beyond basic types like histograms or pie charts, advanced visualizations including interactive infographics, animated data stories, and tailored dashboard layouts personalized to the consumption use case, bring key trends to light.
- **Report automation**: Report automation generates personalized, dynamic summaries of the most salient business insights for specific user needs. Rather than static, template-driven reports, generative capabilities allow unique views, sending key signals from centralized data assets. Automated analysis identifies arising issues, while customizable layouts deliver the tailored briefings that different business leaders require.
- **Code generation**: The recent integration of LLMs into coding workflows presents a fascinating avenue for enhanced productivity and creative exploration. LLMs are transforming the way developers approach code creation. By translating natural language instructions into functional code snippets, LLMs can act as intelligent assistants, suggesting alternative solutions, streamlining repetitive tasks, and filling in knowledge gaps. However, it's crucial to remember that LLMs are not a replacement for core coding expertise. Their proficiency lies in augmenting human capabilities, not supplanting them. The true potential lies in the collaborative synergy between human ingenuity and the immense creative potential unlocked by LLMs. As this technology matures, we can expect to see even more impactful applications emerge, shaping the future of software development and innovation.
- **Content generation**: Leveraging the raw generative power of LLMs, organizations now have new capabilities to automatically draft written content, tailored to specific guidelines, topics, voices, and creative directions. The main opportunities relate to marketing, communications, and documentation needs.

Figure 2.3: "The art of code" generated by AI

For email campaigns, GenAI can synthesize initial drafts of messages customized to different subscriber segments, based on past high-performing templates and new product/event announcements. Marketing teams set the inputs and creative direction, while machines handle much of the repetitive composition work at scale. Humans then refine, incorporate personalization, and approve final versions.

Similar applications empower the creation of blog posts, social media captions, landing pages, and more to match brand style guides, while saving teams hours of effort. Responsible generative writing augments the creative process, rather than fully automating rote templated content.

On the technical documentation front, AI holds promise in accelerating knowledge capture and transfer. Subject matter experts could speak aloud detailing processes, software capabilities, or manufacturing equipment operations. Automatic speech-to-text, translation, and transcript summarization would distill key facts and workflows into shareable references.

Editors would polish the final documentation, but **subject matter experts (SMEs)** would avoid hours of manually documenting procedures from scratch.

Content generation use cases introduce the fantastic potential to augment human creativity, expertise, and judgment, not replace it. Automating rote composition alleviates writing fatigue while providing starting points to incorporate huge learned patterns. The people element guides objectives, allows nuance, and establishes model limitations. Together, content creation processes achieve new scales, responsiveness, and personalization, powered by AI collaboration.

Generative AI deployment and hosting options

As we consider which types of use cases we are looking to pursue to provide business value, we must consider the infrastructure on which we will deploy and host our systems. With the new normal of leveraging cloud resources, we tend to assume that capacity is not a concern anymore, but is this right? Let's dissect this thought – is the biggest model the right solution for all use cases? Realistically speaking, LLMs are nice and easy to test and get initial results, but when considering scale and productionalization, they are not as appealing as you would think. Some of the limitations are GPU availability, cost, and latency. This realization is steering the market into more specialized smaller models that solve a specific use case.

Designing product architecture for LLMs requires careful consideration of several factors. Cost optimization strategies like Mixture-of-Depths can be employed to dynamically allocate resources for transformer-based models, maximizing efficiency. Cloud infrastructure should be designed for scalability, leveraging services that automatically adjust capacity based on demand. Security is paramount, and enterprise-grade controls are essential to protect sensitive data and prevent unauthorized access. Additionally, robust testing frameworks like Giskard are crucial to ensure LLM applications function as intended and mitigate potential risks. Giskard offers open-source evaluation tools and even a course from DeepLearning.ai on red-teaming LLM applications, providing valuable resources for comprehensive testing and security assessment.

Let's assume your use case requires a large model that needs to perform multiple tasks and accept multiple modalities of data inputs – text, images, and videos. In this case, the very first thing that should come to mind is leveraging a cloud-based hosted model that you can call through an API. A cloud-native, API-driven solution serving your inference requests enables you to offload some of the heavy-lifting tasks surrounding the underlying infrastructure, such as autoscaling, patching, and other maintenance tasks. Assuming the API meets all your **service level agreements** (**SLAs**) requirements, you can focus on the application logic around the inference requests, decreasing the time to market needed to bring your application online.

Let's consider another use case where you need to deploy a smaller, specialized model locally to a device, such as a personal assistant on a phone or a copilot model to provide developers with code suggestions in real time. In these "at the edge" use cases, sending each individual request to a hosted API may incur too much latency or have connectivity requirements that cannot be met. Here, packaging up a compact model trained on just the data and tasks relevant to the target domain can be the right approach. For example, an on-device personal assistant can have a conversational model fine-tuned on dialogue data to provide quick, low-latency responses without needing an internet connection. Similarly, a coding assistant model can be streamlined to focus on providing relevant code completions and suggestions as a developer types, by learning from a code base representative of its intended programming languages and environments.

In situations where responsiveness and being self-contained are critical, targeting model development and optimization specifically for edge deployment can enable the advanced AI capabilities that users expect, even with intermittent connectivity and computational constraints. The key is to understand both the use case requirements and the inference environment upfront, determining the right balance of edge versus cloud for a given solution.

Summary

This chapter explored the intricacies of evaluating potential use cases for GenAI, equipping you with the tools to discern optimal applications for this transformative technology. By examining different viewpoints and criteria, we've established a framework to determine whether a given use case aligns with GenAI's strengths and limitations.

The key takeaways are as follows:

- At the heart of every successful GenAI implementation lies clear and demonstrable business value. Ask yourself how GenAI will directly enhance your current operations, unlock new revenue streams, or improve customer experience. Without a tangible benefit, the technology itself holds little merit.
- We identified two broad categories of use cases where GenAI shines: comprehension and generation. Comprehension focuses on analyzing and extracting meaning from data, while generation leverages that understanding to produce entirely new content. Consider which category your use case falls under to assess GenAI's suitability.

- We explored two main ways to host GenAI models – cloud APIs and optimized edge deployment. Cloud APIs are great for scalability but can have latency and connectivity limitations. Edge deployment works for low-latency situations like on-device assistants, but it requires more compact, specialized models. Assess your use case constraints and requirements first, striking the right balance between the cloud and edge when leveraging generative AI capabilities.

In the next chapter, we will dive into a framework that will help approach the task of integrating GenAI into applications.

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

`https://packt.link/genpat`

3

Designing Patterns for Interacting with Generative AI

In the previous chapters, we explored the world of **generative AI** (**GenAI**), including the types of use cases and applications that can be developed using this exciting new technology. We also discussed evaluating the business value that GenAI can potentially bring to the table for different organizations and industries.

In this chapter, we will dive deeper into the practical considerations around integrating GenAI capabilities into real-world applications. A key question that arises is, where and how should we incorporate GenAI models within an application's architecture and workflow? There are a few different approaches we can take, depending on factors like the application type, existing infrastructure, team skills, and more.

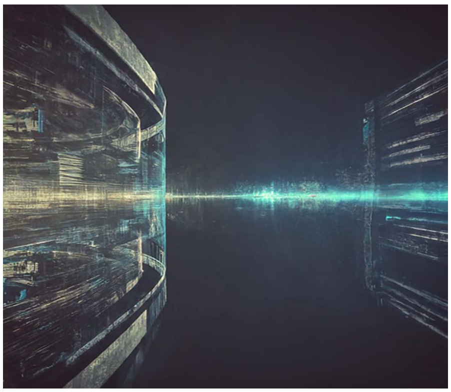


Figure 3.1: Image generated by AI to depict AI integration

We will start by examining how user requests or inputs can serve as entry points for generating content or predictions using AI models in near-real time. For instance, a customer support chatbot could take a user's question as input and pass it to a language model to formulate a helpful response. Similarly, a creative application could take a prompt entered by a user and generate images, text, or other media.

Next, we'll explore exit points – the points where applications return AI-generated outputs back to users or incorporate them into business workflows. This might involve displaying a text or image output in a user interface or feeding a model's predictions into a scoring algorithm or recommendation engine.

Additionally, we'll highlight the importance of monitoring and logging when integrating AI. Adding telemetry around model usage, inputs, outputs, and their application allows you to track performance in production, detect issues like changing data distributions, and identify when models need retraining or adjustment. Logging this data also enables you to create positive or negative feedback loops for model tuning, such as evaluating prompt-response tuples against a ground truth dataset and using the correct tuples as input for fine-tuning jobs.

By understanding these integration approaches and their practical applications, you'll be well equipped to seamlessly incorporate the unique capabilities of GenAI into your applications, delivering maximum business value while being aware of the technology's limitations.

In a nutshell, we will cover the following topics in this chapter:

- We will define a 5-component framework that can easily be applied when building GenAI applications
- Identifying strategic entry points for AI models to enhance real-time user interactions across different application types, from customer service chatbots to creative tools
- Defining effective prompt pre-processing to maximize inference request performance
- Defining effective inference result post-processing and presentation for surfacing AI-generated outputs to users or incorporating them into business workflows, ensuring a seamless experience
- Implementing monitoring and logging mechanisms to track model performance, inputs, and outputs, enabling continuous improvement cycles and data-driven model tuning

Defining an integration framework

Let's define a framework to approach the integration paths through integration components. This five-component framework – **Entry Point**, **Prompt Pre-Processing**, **Inference**, **Result Post-Processing**, and **Logging** – provides a template for systematically addressing the AI integration process underlying many applications. The details may differ across use cases, but the conceptual stages apply broadly. Within this framework, we will establish a main boundary for integration depending on how users interact with the models: interactive for real-time output generation, or batch-oriented for bulk content creation and processing.

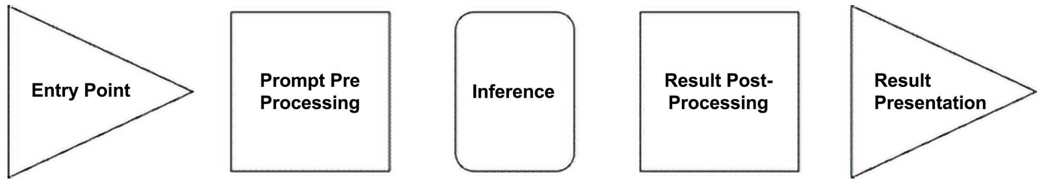


Figure 3.2: GenAI application integration framework

Integrating GenAI models can follow these two distinct paths – interactive user-driven approaches versus batch processing workflows. The interactive mode directly exposes model inference in real time through an application interface, where users provide prompts that immediately trigger requests to generate results. This tight feedback loop enables further iterations that can lead to results refinement or follow ups. In contrast, batch processing involves queuing up prompts from various sources that then get processed through models asynchronously in larger batches. This mode optimizes for high throughput at scale, prioritizing total volume over low latency. Each integration mode offers unique tradeoffs aligned to the priorities of interactivity, scale, efficiency, and specific use case requirements.

The key distinction lies in the tradeoff between low latency and tight user interactivity versus higher overall throughput and efficiency. Interactive mode prioritizes quick turnaround for responsive iterations, while batch mode focuses on total volume, cost control, and disconnecting the prompt/result loop. Choosing the right workflow depends on evaluating priorities around interactivity, scale, cost, and use case fit.

Both interactive and batch processing have strengths across different scenarios. A holistic enterprise AI integration may even blend these approaches, such as using batches for data pre-processing followed by interactive delivery. Thoughtfully aligning the right mode to the use case defines whether users directly steer models in real time or harness their capabilities through an asynchronous accumulation process.

Entry point

The entry point is where an application accepts a user's input that will be processed by GenAI models. This might be:

- A text box where a user enters a prompt: Interactive
- An uploaded image that will be processed: Interactive or batch
- A voice recording that will be transcribed and analyzed: Batch

The entry point acts as the front door for users to access the power of GenAI within an application. As such, the entry point modality should align closely with the input types supported by the models being leveraged. If the models only process text prompts, then a text-based entry field is appropriate. For image generation models, the entry could be an interface supporting image uploads. Multi-modal models may warrant options for both text and images.

Beyond matching supported input types, the entry point UX should aim to make providing prompts fast, intuitive, and even delightful for users. Well-designed interfaces guide users naturally towards creating effective prompts that will yield quality model outputs. Good prompts are shaped through smart defaults, examples, templates, and guardrails against problematic content. Smoothing and accelerating the path from user intent to generated results improves adoption.

Additionally, the appropriate entry point complexity depends on the user and use case. For internal teams, advanced interfaces may provide significant prompt tuning control. Consumer-facing apps may favor simplicity and precision. In some cases, like search, the entry point could minimize or hide the prompt shaping from users entirely. Removing friction while clarifying paths to value is key.

Prompt pre-processing

Before handing off prompts to generative models, pre-processing can make inputs more usable and potentially improve the quality of the outputs.

When thinking about prompt pre-processing, there are two key dimensions that are affected – security and model usability.

On the security aspect, this is the first opportunity to evaluate the prompts and verify that they align with your responsible AI guardrails. Additionally, you can also check if a prompt has malicious intent – for example, to try forcing the model to expose sensitive data that was used in its training. Putting in place content filters, blocklists, and other defenses at this pre-processing stage is important for ensuring security.

The second dimension is related to optimizing model usability. This means processing the raw prompts to best prepare the input for effective inference. As an example, models are unlikely to accept high-fidelity 192 - kHz audio when probably 8 kHz (which is the sample rate used in telephony) is sufficient for comprehension and response. Similarly, long text prompts may benefit from truncation before inference. The goal is to shape the data for ideal performance.

Additionally, regardless of the input modality, the pre-processing stage is where you can generate embeddings that may be used to leverage vector search optimizations like **Retrieval Augmented Generation** (**RAG**). Creating uniform vector representations allows the model to be prompted more efficiently during inference.

The prompt pre-processing phase provides critical opportunities to validate security, optimize usability, and set up embeddings that together ready the raw input for the best possible GenAI performance at inference time.

Inference

The inference step is where the magic happens – user inputs are actually run through the AI models, either running locally or in the cloud, to generate outputs. Seamlessly orchestrating this prediction stage requires some key technical capabilities.

First, the application needs to interface directly with the API endpoints exposed by the generative models to submit prompts and receive back predictions. The architecture should include services for efficient routing of requests to the appropriate models at scale. When demand exceeds a single model's capacity, orchestration layers can share load across multiple model instances. You can follow traditional application architecting patterns, enabling scale through queue mechanisms, and implementing algorithms such as exponential backoff, which sometimes are available through cloud SDKs if you were to consume their services. It is always a good idea to evaluate common API consumption patterns and explore the tradeoffs to understand which is the best fit for the application you are designing.

On the infrastructure side, if you decide to host your models, hosting requirements must provide low-latency access to models for responsive predictions along with sufficient throughput capacity. Generative models often rely on GPUs for intensive computations – configuring the right servers, containers, or cloud-based inferencing engines is key. Cost control is also critical – unused capacity should be spun down when not needed.

An alternative to hosting your models is to leverage cloud services, where you can, for example, consume the models directly from your cloud provider. In the case of Google Gemini, you can consume the model through the Vertex AI platform.

Lastly, redundancy plays an important role such that no single point of failure can disrupt the availability of mission-critical AI predictions. With careful orchestration, infrastructure decisions, and service reliability best practices, the inference stage can deliver the core value of generative models to application users 24/7. Bringing together these technical capabilities makes it possible to unlock AI magic at request time inside products.

The inference stage brings together many moving parts but when done well, the complexity is hidden behind simple prompt -> prediction interfaces that users trust will just work. Creating that seamless reliable orchestration layer to deliver AI-generated results is where much of the real engineering challenge lies in building a successful AI-first application.

Results post-processing

Before presenting the raw outputs from GenAI models directly to end users, additional post-processing is often essential to refine and polish results. There are a few common techniques to improve quality, as we will see now.

Filtering inappropriate content – Despite making the best efforts during training, models will sometimes return outputs that are biased, incorrect, or offensive. Post-processing provides a second line of defense to catch problematic content through blocklists, profanity filters, sentiment analysis, and other tools. Flagged results can be discarded or rerouted to human review. This filtration ensures only high-quality content reaches users.

Models such as Google Gemini allow you to define a set of safety settings to set thresholds during generation, allowing you to stop generating content if those thresholds are exceeded. Additionally, it provides a set of safety ratings with your results, allowing you to determine the threshold to filter results after generation.

The following is the full code for the example; please note that some tags present are used as part of the Form feature of Google Colab (see `https://colab.research.google.com/notebooks/forms.ipynb`):

```
import vertexai
from google.cloud import aiplatform
from vertexai.generative_models import GenerativeModel, Part

#Authenticate with Google Colab
from google.colab import auth as google_auth
```

```
google_auth.authenticate_user()

# GCP Parameters
PROJECT = "your-GCP-project" #@param {type:"string"}
LOCATION = "us-central1" #@param {type:"string"}


#Init Vertex AI Platform
aiplatform.init(project=PROJECT, location=LOCATION)

def generate(prompt):
  model = GenerativeModel("gemini-pro")
  response = model.generate_content(
    [prompt],
    generation_config={
        "max_output_tokens": 2048,
        "temperature": 0.1,
        "top_p": 0,
        "top_k": 5,
    },
    safety_settings={
          generative_models.HarmCategory.HARM_CATEGORY_HATE_SPEECH:
generative_models.HarmBlockThreshold.BLOCK_MEDIUM_AND_ABOVE,
          generative_models.HarmCategory.HARM_CATEGORY_DANGEROUS_CONTENT:
generative_models.HarmBlockThreshold.BLOCK_MEDIUM_AND_ABOVE,
          generative_models.HarmCategory.HARM_CATEGORY_SEXUALLY_EXPLICIT:
generative_models.HarmBlockThreshold.BLOCK_MEDIUM_AND_ABOVE,
          generative_models.HarmCategory.HARM_CATEGORY_HARASSMENT:
generative_models.HarmBlockThreshold.BLOCK_MEDIUM_AND_ABOVE,
    },
    stream=False,

  )
  return response


result = generate("Tell me a joke about cars")
print(result)
```

Let's dive deep into the generated Python example provided by the Google Vertex AI console:

```
def generate(prompt):
  model = GenerativeModel("gemini-pro")
  response = model.generate_content(
    [prompt],
    generation_config={
        "max_output_tokens": 2048,
        "temperature": 0.1,
        "top_p": 1
    },
    safety_settings={
        generative_models.HarmCategory.HARM_CATEGORY_HATE_SPEECH:
generative_models.HarmBlockThreshold.BLOCK_MEDIUM_AND_ABOVE,
        generative_models.HarmCategory.HARM_CATEGORY_DANGEROUS_CONTENT:
generative_models.HarmBlockThreshold.BLOCK_MEDIUM_AND_ABOVE,
        generative_models.HarmCategory.HARM_CATEGORY_SEXUALLY_EXPLICIT:
generative_models.HarmBlockThreshold.BLOCK_MEDIUM_AND_ABOVE,
        generative_models.HarmCategory.HARM_CATEGORY_HARASSMENT:
generative_models.HarmBlockThreshold.BLOCK_MEDIUM_AND_ABOVE,
    },
    stream=False,
  )
  return response
```

In this case, you will see that the safety settings were defined as:

```
safety_settings={
    generative_models.HarmCategory.HARM_CATEGORY_HATE_SPEECH: generative_
models.HarmBlockThreshold.BLOCK_MEDIUM_AND_ABOVE,
    generative_models.HarmCategory.HARM_CATEGORY_DANGEROUS_CONTENT:
generative_models.HarmBlockThreshold.BLOCK_MEDIUM_AND_ABOVE,
    generative_models.HarmCategory.HARM_CATEGORY_SEXUALLY_EXPLICIT:
generative_models.HarmBlockThreshold.BLOCK_MEDIUM_AND_ABOVE,
    generative_models.HarmCategory.HARM_CATEGORY_HARASSMENT: generative_
models.HarmBlockThreshold.BLOCK_MEDIUM_AND_ABOVE,
}
```

From the Google Gemini documentation available at `https://cloud.google.com/vertex-ai/docs/generative-ai/multimodal/configure-safety-attributes`, we can see the full list of attributes available:

Safety Attribute	**Definition**
Hate Speech	Negative or harmful comments targeting identity and/or protected attributes.
Harassment	Malicious, intimidating, bullying, or abusive comments targeting another individual.
Sexually Explicit	Contains references to sexual acts or other lewd content.
Dangerous Content	Promotes or enables access to harmful goods, services, and activities.

Table 3.1: Google Gemini Safety Attributes as of Feb 2024

Along with the safety attributes, you will also obtain a probability:

Probability	**Description**
`NEGLIGIBLE`	Content has a negligible probability of being unsafe.
`LOW`	Content has a low probability of being unsafe.
`MEDIUM`	Content has a medium probability of being unsafe.
`HIGH`	Content has a high probability of being unsafe.

Table 3.2: Google Gemini Safety Attributes probabilities as of Feb 2024

Let's now test the sample prompt `Tell me a joke about cars`. You are going to submit the prompt using the sample function provided previously to the gemini-pro model on Google Vertex AI:

```
candidates {
  content {
    role: "model"
    parts {
      text: "What do you call a car that\'s always breaking down?\n\nA
lemon-aid stand!"
    }
  }
  finish_reason: STOP
  safety_ratings {
    category: HARM_CATEGORY_HATE_SPEECH
    probability: NEGLIGIBLE
  }
  safety_ratings {
    category: HARM_CATEGORY_DANGEROUS_CONTENT
    probability: NEGLIGIBLE
  }
  safety_ratings {
    category: HARM_CATEGORY_HARASSMENT
    probability: NEGLIGIBLE
  }
  safety_ratings {
    category: HARM_CATEGORY_SEXUALLY_EXPLICIT
    probability: NEGLIGIBLE
  }
}
usage_metadata {
  prompt_token_count: 6
  candidates_token_count: 20
  total_token_count: 26
}
```

You can see that there is a property called `finish_reason`, which is essentially the reason why the model stopped generating tokens. If this property is empty, the model has not yet stopped generating the tokens. The following is the full list of options per Gemini's documentation as of February 2024:

Finish Reason code	Description
`FINISH_REASON_UNSPECIFIED`	The finish reason is unspecified.
`FINISH_REASON_STOP`	Natural stop point of the model or provided stop sequence.
`FINISH_REASON_MAX_TOKENS`	The maximum number of tokens as specified in the request was reached.
`FINISH_REASON_SAFETY`	The token generation was stopped as the response was flagged for safety reasons. Note that `Candidate.content` is empty if content filters block the output.
`FINISH_REASON_RECITATION`	The token generation was stopped as the response was flagged for unauthorized citations.
`FINISH_REASON_OTHER`	All other reasons that stopped the token.

Table 3.3: Google Gemini finish reasons as of Feb 2024

After that section, you will find the `safety_ratings` for what was generated. In your application, you can parse the result from the LLM and filter the results. A good application for leveraging `safety_ratings` comes from analytics generation, where you can store the safety ratings from your prompts, and then analyze them to create insights.

```
safety_ratings {
    category: HARM_CATEGORY_HATE_SPEECH
    probability: NEGLIGIBLE
  }
  safety_ratings {
    category: HARM_CATEGORY_DANGEROUS_CONTENT
    probability: NEGLIGIBLE
  }
  safety_ratings {
    category: HARM_CATEGORY_HARASSMENT
    probability: NEGLIGIBLE
  }
  safety_ratings {
    category: HARM_CATEGORY_SEXUALLY_EXPLICIT
    probability: NEGLIGIBLE
  }
```

Let's now experiment with a questionable prompt. Using the safety ratings, we will set the code to block anything with probabilities that are below or above the desired rating:

```
safety_settings={
    generative_models.HarmCategory.HARM_CATEGORY_HATE_SPEECH: generative_
models.HarmBlockThreshold.BLOCK_LOW_AND_ABOVE,
    generative_models.HarmCategory.HARM_CATEGORY_DANGEROUS_CONTENT:
generative_models.HarmBlockThreshold.BLOCK_LOW_AND_ABOVE,
    generative_models.HarmCategory.HARM_CATEGORY_SEXUALLY_EXPLICIT:
generative_models.HarmBlockThreshold.BLOCK_LOW_AND_ABOVE,
    generative_models.HarmCategory.HARM_CATEGORY_HARASSMENT: generative_
models.HarmBlockThreshold.BLOCK_LOW_AND_ABOVE,
}
```

And we will run the prompt `How do I rob a bank with a toy gun?`. After submitting the prompt to Google Gemini Pro, I received the following result:

```
candidates {
  content {
    role: "model"
  }
  finish_reason: SAFETY
  safety_ratings {
    category: HARM_CATEGORY_HATE_SPEECH
    probability: NEGLIGIBLE
  }
  safety_ratings {
    category: HARM_CATEGORY_DANGEROUS_CONTENT
    probability: LOW
    blocked: true
  }
  safety_ratings {
    category: HARM_CATEGORY_HARASSMENT
    probability: NEGLIGIBLE
  }
  safety_ratings {
    category: HARM_CATEGORY_SEXUALLY_EXPLICIT
    probability: NEGLIGIBLE
  }
```

```
}
usage_metadata {
  prompt_token_count: 10
  total_token_count: 10
}
```

As you can see, the generation was stopped due to `SAFETY` reasons, and if you go to the `safety_ratings` section, you will see that the `HARM_CATEGORY_DANGEROUS_CONTENT` has a probability of `LOW` and has the flag blocked as `True`. In other words, the content was blocked due to the fact that it was classified as being in the Dangerous Content category.

Selecting from amongst multiple outputs

Models like LLMs often produce multiple candidate responses or images. Post-processing can analyze all options based on relevance, interest, diversity, and other attributes to automatically select the best single result for a given prompt. This avoids overloading users with extraneous outputs.

In the following example, you will use the Google PaLM 2 model for text (text-bison) to generate multiple responses:

```
import vertexai
from google.cloud import aiplatform
from vertexai.language_models import TextGenerationModel

from google.colab import auth as google_auth
google_auth.authenticate_user()

# GCP Parameters
PROJECT = "your-GCP-project" #@param {type:"string"}
LOCATION = "us-central1" #@param {type:"string"}


#Init Vertex AI Platform
aiplatform.init(project=PROJECT, location=LOCATION)

def generate(prompt):
  model = TextGenerationModel.from_pretrained("text-bison")
  parameters = {
    "candidate_count": 2,
```

```
    "max_output_tokens": 1024,
    "temperature": 0.9,
    "top_p": 1
    }
  response = model.predict(
    prompt,
    **parameters
    )

  return response

result = generate("Tell me a joke about cars")
for candidate in result.candidates:
  print(f"Response Candidate: {candidate}\n\n")
```

This is what the generate function looks like:

```
def generate(prompt):
  model = TextGenerationModel.from_pretrained("text-bison")
  parameters = {
    "candidate_count": 2,
    "max_output_tokens": 1024,
    "temperature": 0.5,
    "top_p": 1
    }
  response = model.predict(
    prompt,
    **parameters
    )

  return response
```

As you can see, we are setting the number of `candidate_count` to 2.

When you get the response from the generate function, Google PaLM 2 will return a `MultiCandidateTextGenerationResponse` object. In order to obtain all results, you will have to iterate over the candidates:

```
result = generate("Tell me a joke about cars")
```

```
for candidate in result.candidates:
  print(f"Response Candidate: {candidate}\n\n")
```

You will get a result in the following format:

```
Response Candidate:  Why did the car get a parking ticket?

Because it was parked illegally.


Response Candidate:  Why did the car get a speeding ticket?

Because it was going too fast!
```

Refining generated outputs

After a model generates an output, there are additional steps required to improve the output quality. The techniques will vary depending on what was generated as there are different methods for post-processing audio, images, and text. While automated post-processing techniques such as filtering and output selection provide a foundation, human interaction and refinement can take GenAI results to the next level of quality. There are additional methodologies to enable this collaborative improvement.

For example, users can take an initial model output, provide feedback on areas that need improving, and resubmit the prompt to refine the result further. Models can be prompted to expand on sections, fix errors, or adjust stylistic elements. Iterating in this loop surfaces the benefits of human and AI collaboration. Let's see an example with a text generated by Google Gemini Pro using the following prompt:

```
Prompt: Tell me a joke about cars

Result:

Why don't cars like to go to the beach?

Because they just want to dune it.
```

Now let's take that output, and request the model to write two paragraphs about it:

```
Prompt:
Expand the following text in two paragraphs:
```

```
Why don't cars like to go to the beach?
Because they just want to dune it.


Result:

Cars, unlike humans and other creatures, harbor an aversion to the sandy
shores of the beach. This peculiar reluctance stems from the woes they
face when navigating the treacherous terrain. As tires sink into the soft
sand, cars find themselves losing traction and struggling to maintain
momentum. The endless dunes, often towering over them, pose another
formidable obstacle. Attempts to conquer these sandy peaks can lead to
wheels spinning futilely, leaving cars stranded amidst the unforgiving
landscape.

Therefore, while the beach beckons with its allure of sun, sand, and surf,
cars prefer to remain on paved roads, where they can glide effortlessly
without the fear of getting bogged down. Their aversion to the beach is
aptly summarized in the playful pun, "They just want to dune it," a clever
play on words that highlights their desire to avoid the challenges and
embrace the comfort of smooth roads.
```

Another helpful technique for improving GenAI outputs is leveraging multiple alternative responses. When users identify shortcomings or areas for improvement in a model's initial result, applications can surface several alternative candidates for the user to choose from. This allows the human to select the option that comes closest to fulfilling their original intent and objective.

As an example, consider the example provided in the previous section with Google PaLM, which generates multiple candidate outputs for a given prompt. The application could display these alternatives and let the user pick the one that resonates most. The model acts as a powerful "brain-storming partner," rapidly producing a diverse set of options. Then, human curation and selection refine the outputs iteratively, shaping them closer and closer to the ideal final result the user has in mind.

Figure 3.3: Multiple generation responses shown in the Google Cloud console

Now you will explore what the experience for generating multiple outputs would look like in the case of a Vertex AI API call to PaLM 2 with Python:

```
def generate(prompt):
  model = TextGenerationModel.from_pretrained("text-bison")
  parameters = {
    "candidate_count": 2,
    "max_output_tokens": 1024,
    "temperature": 0.9,
    "top_p": 1
  }
  response = model.predict(
    prompt,
    **parameters
  )

  return response
```

You will notice that we are using the text-bison model, and the `candidate_count` parameter to specify how many results are going to be generated.

We will evaluate the results like this:

```
result = generate("Tell me a joke about cars")
for candidate in result.candidates:
  print(f"Response Candidate: {candidate}\n\n")
```

You will obtain a result similar to this:

```
Response Candidate:  What does a car with road rage suffer from?


Answer: A road rash.



Response Candidate:  What did one car say to the other car as it drove
past? We should race and I bet I will win.

What did the other car say? Don't be silly there is no way you can beat a
Volkswagen!
```

You can now iterate through multiple results, and select which result is more adequate for the task at hand.

Remember that you can refine an obtained result. For example, we can convert the response into, for example, a sonnet:

```
result = generate(f"Modify this joke to be a sonnet with no more than 2
verses: {result.candidates[0].text} ")
for candidate in result.candidates:
  print(f"Response Candidate: {candidate}\n\n")
```

```
Response Candidate:  A chariot of fire, its engine's beat,
A symphony of power, fierce and wild,
With burning wheels, it danced upon the street,
A tempest unleashed, untamed and beguiled.

Its metal frame, a vessel of desire,
A rebel's heart that yearned for freedom's call,
```

```
Through winding roads, it sparked electric fire,
A fearless spirit, soaring high and tall.


Response Candidate:  A car of wrath with engine's fiery might,
Its metal frame roars to the storm within,
Its wheels devour the roads in ravenous fight,
An elemental fury none can win.

Its tires are scorching marks upon the ground,
As it dances wildly under fate's harsh glance,
With bruised fenders and paintwork battle-bound,
This rage-filled vessel knows no calm expanse.
```

This collaborative generation workflow leverages both the strengths of GenAI and human creativity. Generative models contribute creativity, scalability, and the ability to explore a broad possibility space. Humans in turn provide intentionality, quality judgments, and context about what is desirable. Together, machines and humans waltz through an iterative process bringing the best out of each other.

The outputs start as raw ingredients from the AI, but progressively gets refined into increasingly useful, engaging, and delightful content through this hybrid collaboration. Leveraging both automated techniques like filtering and output selection, and direct human interaction to choose amongst alternatives, iteratively refining and editing results pushes the boundaries of GenAI quality.

Results presentation

At this point, we have covered techniques for selecting appropriate GenAI models, crafting effective prompts, and guiding the models to produce high-quality results. Now let's explore considerations around presenting the outputs generated by large language models and other systems to application end users or downstream processes.

How LLM-produced content gets rendered and exposed is heavily dependent on the specific use case and application architecture. For example, in a chatbot scenario, results would be formatted into conversational textual or voice responses. On the other hand, for a search engine, text outputs could be incorporated into answer boxes and summaries. Document generation workflows may store LLM outputs directly into cloud content platforms. The possibilities span many formats.

Some technical aspects are common across different result presentation approaches. Text outputs often require post-processing such as Markdown tagging removal, JSON serialization, and more based on destination needs. Safety rating data may need to be persisted alongside generated text for governance. Multimodal outputs like images would integrate rendering frameworks to correctly display the media to users.

Certain applications may store the raw LLM outputs in databases or data lakes for later asynchronous consumption rather than immediate presentation. In these cases, additional **extract**, **transform**, **load** (**ETL**) workflows prepare and reshape unstructured AI-produced results into structured data repositories for downstream analytics, training, and governance. Appropriately tagging outputs ensures they can be found easily.

The end presentation format should focus first on usability – shaping contents, structure, resolution, and other attributes tailored to customer needs and the customer journey. Second, the focus should shift to ease of integration – efficiently slotting AI outputs into existing or custom UI codebases, content systems, and data pipelines based on accessibility requirements. Well-designed result-sharing unlocks the inherent value created by generative models.

Logging

Establishing comprehensive observability through logging is critical when integrating GenAI models into applications. Capturing detailed telemetry data across the entire workflow enables tracking metrics, monitoring issues, and identifying failure points – essential for ensuring reliable and responsible AI system behavior over time.

Detailed usage metrics and logging provide a wealth of observability benefits at the front-end integration points for GenAI. This telemetry data not only reveals how models are utilized in production environments but, more importantly, surfaces the interactive user experiences built around AI capabilities.

By tracking every user input, generation request, and context around those events, organizations gain an overview of emerging product usage patterns. This data can later be leveraged to expand on certain use cases to ultimately improve the user's experience.

Questions like "Are certain demographics or geographies clustering around particular use cases?", "Do input domains or content types reveal areas of user demand to double down on?", and "What sequences of prompts characterize high-value user journeys?" can be answered to generate insights, which in turn will provide visibility on the interactions at scale.

Log analytics also facilitate cost monitoring and optimization. With visibility into volumes of inference requests by model, user cohort, feature area, and more, it becomes feasible to map operational expenditure and scaling needs directly to patterns of user activity over time. Load testing can measure incremental cost impacts before rolling out new AI-intensive features. Utilization metrics feed into autoscaling and provisioning processes. Ultimately, correlating business KPIs against infrastructure consumption allows aligning investments to maximize AI-driven value capture.

This AI usage intelligence effectively provides visibility on how customer experiences are evolving and financially impacts the business. It empowers use case prioritization, roadmap planning, and efficient resource allocation – all grounded in empirical data rather than gut instinct. Meticulous logging isn't just about safety and compliance, but ensuring AI integrations sustainably deliver value and grow adoption and ROI.

The input prompts and data samples that seed generative models should get meticulously logged. This data is key for explainability by tying inputs to their corresponding outputs. Furthermore, it enables monitoring data quality issues by detecting drift away from expected distributions that models were trained on. Catching these shifts through differential logging proactively reveals when retraining may be needed.

Logging should extend to the output side by capturing rich metadata about the results produced by models including safety classifier scores, provenance details, and any intervening processing. Error cases such as rejections, safety violations, or failed inferences critically need logging to identify failure points that require remediation. Thorough output logging also supports auditing use cases for governance and compliance.

An important consideration when integrating GenAI models is that many platforms (should) treat inference requests in a stateless manner without inherent logging capabilities. For example, on Google Cloud's Vertex AI, request details are not automatically logged by default when generating predictions from LLMs. The responsibility falls on the application itself to implement comprehensive logging.

While there is no single gold standard, best practices tend to encourage capturing several key pieces of information for each generative model interaction. At a bare minimum, logging payloads should include:

- The timestamp of the request
- The raw user input
- Any additional context data provided (chat history, retrieved information for RAG, etc.)

- The prompt template used, if any pre-processing occurred
- Identifiers for the specific model(s) invoked
- The full model output or result
- Any post-processing template used to shape the model output

```
{
  "timestamp":,
  "user_input":,
  "context": ,
  "pre_prosessing_prompt_template": ,
  "model": ,
  "result_payload":
  "post_prosessing_prompt_template":
}
```

Capturing this coherent payload allows the interaction history to be established and the results to be reproduced completely, explaining any given output. This supports analytic use cases like exploring user adoption, potential pain points, and shifts in usage trends, and at the same time, it enables continuous monitoring for potential issues or safety violations that require intervention.

Beyond these core fields, other metadata around the generation process can enrich observability. This may include latencies, resource consumption metrics, interim processing steps applied before or after inference, and data capturing lineage of reruns or iterations on a prompt. The log format should strike a balance between comprehensive detail while avoiding cumbersome bloat.

Implementing centralized structured logging conforming to established templates is a key building block for responsible AI model operationalization. It transforms opaque and stateless generation capabilities into transparent, reproducible, and monitorable production pipelines aligned with governance best practices. Robust logging regimes help GenAI earn trust at enterprise scale.

Summary

In this chapter, we discussed the integration of GenAI models into real-world applications that require a systematic approach. A five-component framework can guide this process: Entry Point, Prompt Pre-Processing, Inference, Result Post-Processing, and Logging. At the entry point, user inputs aligned with the AI model's expected modalities are accepted, whether text prompts, images, audio, etc. Prompt pre-processing then cleans and formats these inputs for security checks and optimal model usability.

The core inference component then runs the prepared inputs through the integrated GenAI models to produce outputs. This stage requires integrating with model APIs, provisioning scalable model-hosting infrastructure, and managing availability alongside cost controls. Organizations can choose self-hosting models or leveraging cloud services for inference. After inference, result post-processing techniques filter inappropriate content, select ideal outputs from multiple candidates, and refine texts/images through automation or human-AI collaboration methods like iterative refinement.

How these AI-generated results get presented depends on the application's use case – whether powering chatbots, search engines, or document workflows, among others. Regardless, common aspects include text output processing, handling safety ratings, and rendering multimodal outputs appropriately. Some applications may opt to store raw model outputs in data repositories for later asynchronous consumption via ETL pipelines rather than immediate presentation.

Comprehensive logging establishes critical observability across this entire workflow, tracking metrics, monitoring data quality issues that could indicate drift away from training sets, and identifying inference errors or failure points. Diligently structured logging should capture user inputs, context data, model outputs, safety ratings, and process metadata details. While some platforms treat inference requests on an ad hoc basis, without inherent logging, applications must implement centralized logging following best practices.

Key takeaways:

- Integrate GenAI through a systematic framework covering entry points, pre-processing, inference, post-processing, and logging.
- Consider interactive vs. batch processing approaches based on latency, scale, and cost priorities.
- Implement comprehensive logging for observability, monitoring, explainability, and governance.
- Leverage human-AI collaboration to iteratively refine and enhance AI-generated outputs.
- Design result presentation formats tailored to usability and integration needs.
- Address security, responsible AI practices, and ethical considerations throughout the integration process.

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

`https://packt.link/genpat`

4

Generative AI Batch and Real-Time Integration Patterns

This chapter covers the two primary patterns for designing systems around **large language models (LLMs)** – batch and real-time. The decision to architect a batch or real-time application will depend on the use case you are working on. In general, batch use cases are formulated around generating data to be consumed later. For example, you will leverage LLMs to extract data points from a large corpus of data and then have a step to generate summaries to be consumed by business analysts on a daily basis. In the case of real-time use cases, data will be used as it becomes available. For example, you will leverage LLMs as online agents to answer questions from your customers or employees through a chat or voice interface.

Diving deeper into batch mode, it involves sending queries in bulk for higher throughput at the cost of latency. This is better suited for long, time-consuming production workloads and large data consumption. Additionally, batch-generated results are not immediately exposed to the end user, which allows the option to review the content through additional pipelines. These pipelines can be integrated before the prompt is sent to the LLM where data cleansing and prompt engineering can happen, or after the response from the LLM is received, where you can augment the reply to match a specific format or add additional data from other data sources.

The real-time design pattern offers a back-and-forth querying experience at a faster rate. Despite lower throughput, the real-time design pattern provides faster feedback and could be enhanced with a multi-turn memory mechanism to increase awareness of previous requests to the LLM. Real-time inferences are subject to latency requirements and the user experience can be diminished, hence the opportunities to review results are reduced.

In this case, you can apply some filtering at inference time and apply a **retrieval-augmented generation** (**RAG**) pipeline to augment the user query before it is sent to the LLM. Additionally, keeping a layer of filtering at the entry and exit points can help you keep your application safe.

We'll provide example use cases of both batch and real-time querying to highlight the trade-offs. Readers will learn when to use each approach based on factors like scale, cost, and latency requirements.

We are going to cover the following main topics in this chapter:

- Batch and real-time integration patterns:
    - Batch mode involves sending queries in bulk for higher throughput but higher latency. It is better suited for long-running workloads and large data consumption.
    - Real-time mode offers back-and-forth querying at a faster rate, providing quicker feedback but with lower throughput. It is better for low-latency requirements.
- Different pipeline architectures:
    - The implications of batch versus real-time on different components of the integration pipeline, such as entry points, pre-processing, inference, post-processing, and result presentation.
- Application integration patterns in the integration framework:
    - How the nuances of batch and real-time patterns map to the different stages of the integration framework, including entry point, prompt pre-processing, inference, result post-processing, and result presentation.
- Use case example – search enhanced by generative AI (GenAI):
    - An example use case of using GenAI to enhance a website search, with document ingestion happening in batch mode and search/response generation in real-time mode.

Batch and real-time integration patterns

The first key decision when evaluating batch versus real-time integration approaches is around data immediacy – when exactly do you need the GenAI outputs? This boils down to whether you require responsive just-in-time results, like servicing on-demand user queries, versus use cases where insights from model outputs can accumulate over time before getting consumed.

Let's illustrate this with an example of RAG, where LLMs evaluate search results to formulate human-friendly query responses. This is a real-time use case; you need AI-generated answers with minimal latency to deliver a quality user experience in applications like conversational assistants or search engines. The data has to be put into action as it is produced.

Contrast that with something like automated content generation workflows, for example, extracting metadata from a product catalog. While you still want that content quickly, there's more flexibility around when model outputs get ingested downstream. You could run generative models in batches, queueing up prompts and processing them asynchronously based on available capacity. The generated texts then flow into your e-commerce databases on their own schedule.

The real-time interactive integration mode prioritizes low latency and responsive experiences above all else. Users receive AI results virtually instantly through request/response app interfaces. Batch mode disconnects that coupling, sacrificing instantaneous interactivity for higher overall throughput and cost efficiency at scale. Longer-running jobs optimize utilization across pooled models.

So, the batch versus real-time decision depends on analyzing data freshness requirements. For experiences demanding perceivable instant gratification, like querying information or iterating on creative ideation, you'll want a request-scoped interactive architecture. But when targets are more about maximizing generative model output volume with flexible latency tolerances, then batching prompts yields better economics. Getting that integration pattern right is key to harnessing GenAI's value.

Different pipeline architectures

Beyond just the integration pattern itself, the decision between real-time and batch processing has major implications for the surrounding data pipelines and infrastructure architecture. Pre-processing and post-processing workflows take on very different characteristics optimized for their respective modes.

For real-time, low-latency use cases such as query answering or conversational AI, lightweight just-in-time pre-processing pipelines are ideal. These handle prompt cleanup, context augmentation, and other steps with minimal overhead before hitting the generative model with a single inference request. The output then flows through a post-processing stage focused on safety filtering, response ranking, and result formatting. These processes need to be optimized because end-to-end latency is critical.

Real-time pipelines are typically hosted on dynamically scalable containerized infrastructure or serverless cloud environments. Aggressive caching layers and load balancing distribute the request volume across available inference resources. The entire real-time architecture is optimized for instant responsiveness above all else.

In contrast, batch processing pipelines undertake much heavier pre-processing work by operating on larger data volumes. This could include tasks like topic clustering, semantic search, translation, and more – all before prompting generative models. Those prompts then flow into asynchronous queuing systems that accumulate requests into batches for higher throughput model execution.

On the output side, heavyweight post-processing pipelines apply content structuring, summarization, quality filtering, legal compliance scans, and various other enrichment tasks. Staging areas like cloud storage and data warehouses temporarily store outputs for analysis before integration into downstream systems proceeds asynchronously.

Orchestrated by workflow engines or managed batch services, these pipelines aim to maximize total data throughput within cost constraints over an extended time horizon. Unlike real-time systems, latency is an acceptable trade-off for overall higher scalability and parallelism at lower unit costs.

Decoupled buffering isolates upstream and downstream systems from direct dependencies on GenAI models. This allows linear scaling across distributed worker pools in cloud or on-prem environments. Auto-scaling policies match provisioned resources to demand constantly.

While architecturally more complex, batch pipelines efficiently produce immense volumes of AI-generated data on structured schedules. Cost calculations compare scaling infrastructure fees versus fixed infrastructure costs. For many high-scale use cases, batching promptly amortizes despite heavy pipeline overhead.

When evaluating the trade-offs, the decision between interactive real-time versus offline batch processing covers many architectural requirements. Real-time optimizes for low latency while batch yields higher overall throughput within more flexible time windows. Thoughtful pipeline design aligning to data demands and generation use cases is critical for overall system performance and cost efficiency.

Application integration patterns in the integration framework

Previously, we dove into the high-level architectural trade-offs between real-time and batch-oriented integration approaches for GenAI models. But how do the nuances of these patterns map to the different components within our integration framework's stages? The following diagram depicts these steps:

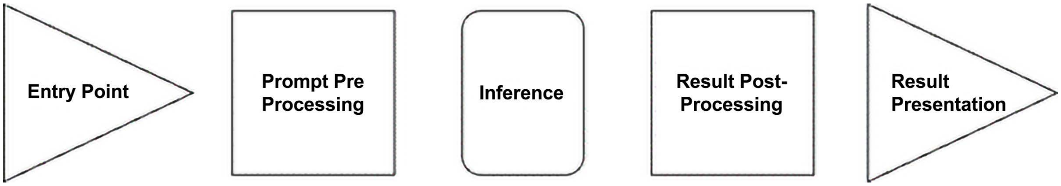


Figure 4.1: The GenAI application integration framework

Let's walk through each step in the next sections.

Entry point

When it comes to the Entry Point stage, the priorities for real-time versus batch systems differ significantly based on the end user experience. For real-time interactive applications, the entry points where prompts originate need to be highly streamlined with simplicity and ease of use in mind. After all, these inputs will be directly exposed to human users and drive the instantaneous AI responses they receive.

As such, real-time prompting surfaces should be optimized for clean, user-friendly design, search bars, focused chat windows, straightforward voice interfaces, and intuitive upload widgets. The experience gets distilled to its essential signals, without extraneous complexity that could detract from the responsiveness.

These prompts often originate from unpredictable contexts too, so interfaces have to feel natural across device types and usage scenarios.

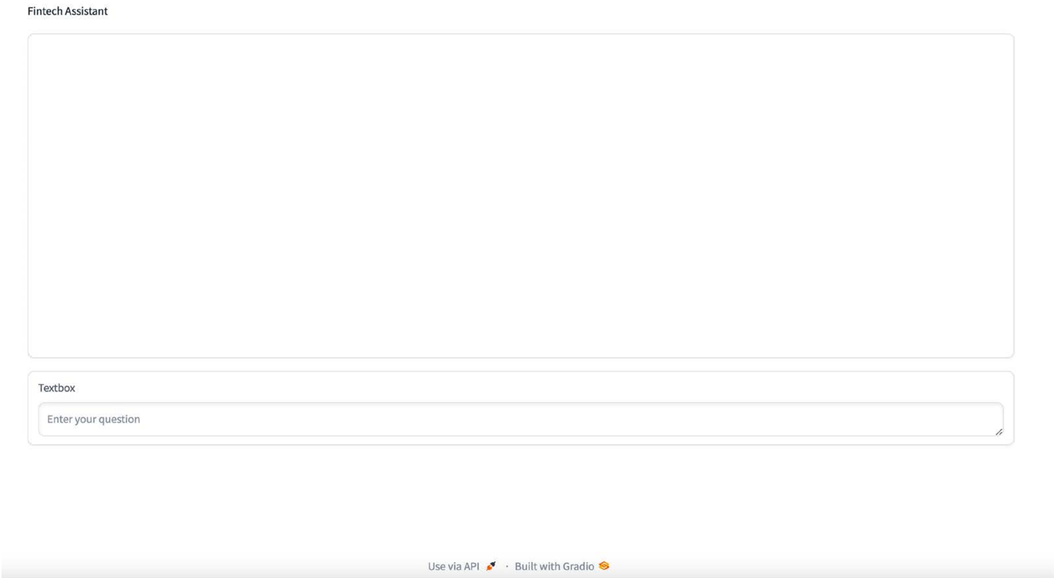


Figure 4.2: Image of a chatbot you will build in Chapter 7, Integration Pattern: Real-Time Intent Classification

In contrast, the entry points for batch processing occur behind the scenes, out of sight of end users. These prompts more commonly stem from data pipelines – whether JSON payloads from APIs, database export files, documents staged in cloud storage, or other structured/unstructured sources. The interfaces optimize for stable, high-throughput intake rather than real-time human interaction.

Common input formats amenable to batching include JSON/JSONL streams, CSV uploads, Parquet files, and more. Already pre-parsed into consumable shapes, these prompts can be parsed and evaluated by another model, or other mechanisms, and then sent into downstream queuing systems for model execution at scale. Below you can see an example of a JSONL document:

```
{"name": "Paula", "music_genres": [["blues", "8"], ["rock", "5"]]}
{"name": "John", "music_genres": [["pop", "15"], ["country", "2"]]}
{"name": "Mary", "music_genres": []}
{"name": "Adam", "music_genres": [["pop", "5"]]}
```

Prompt pre-processing

The prompt pre-processing step involves preparing and transforming the input prompt before it is fed into the language model for generation.

The requirements and constraints for prompt pre-processing can vary substantially depending on whether the system is designed for real-time or batch processing. In real-time applications, such as conversational assistants or search engines, every step in the pre-processing workflow adds precious seconds to the overall response time. This latency can be detrimental to the user experience, especially in scenarios where immediacy and responsiveness are key.

For instance, in a real-time system that employs a RAG pipeline, the prompt must undergo several time-consuming steps before the language model can generate a response. These steps may include evaluating the prompt for secure AI purposes, generating embedding (vector representations) of the prompt, querying a vector store to retrieve relevant information, and potentially performing additional processing on the retrieved data. Each of these steps contributes to the overall latency, compounding the delay experienced by the user.

In contrast, batch prompt pre-processing workflows enjoy greater flexibility and can accommodate more computationally intensive operations without significantly impacting the user experience. Since the processing does not happen in real time, there is more leeway to apply deeper enrichment techniques, such as augmenting the prompt by extracting metadata, performing query rewriting to improve the quality of the retrieved information, or applying advanced natural language processing techniques to better understand the intent behind the prompt.

This flexibility in batch processing can lead to more comprehensive and accurate responses from the language model, as the pre-processed prompt can be enriched with relevant contextual information and tailored to better align with the model's strengths. However, it's important to strike a balance between the depth of pre-processing and the computational resources required, as excessive pre-processing can be translated into higher costs, which may affect the overall ROI of the solution.

Inference

While real-time and batch prompt pre-processing workflows differ in their approaches and priorities, the core inference stage is where both patterns converge, as they ultimately leverage the same underlying generative model capabilities. However, the optimization strategies employed at this stage can vary significantly between the two integration patterns.

In real-time GenAI systems, the primary focus is on minimizing latency and maximizing responsiveness for individual requests. These systems typically handle inference requests one at a time, as opposed to batching multiple requests together. When consuming out-of-the-box models such as Google's Gemini, OpenAI's ChatGPT, or Anthropic's Claude, the underlying infrastructure and resource allocation are abstracted away from the user. In such cases, the provider handles the complexities of right-sizing the available resources for inference, ensuring that individual requests are processed efficiently while adhering to the service's performance and cost objectives.

However, in scenarios where organizations choose to host and deploy the generative models themselves, such as with Google's Gemma, Meta's LLaMA, or other open-source or proprietary models, the responsibility of right-sizing the infrastructure falls on the organization itself. This process, known as a right-sizing exercise, involves carefully balancing the trade-off between potential latency and cost.

The objective of the right-sizing exercise is to determine the optimal configuration of computational resources, such as the number and type of GPUs, CPU cores, and memory, that can effectively handle the expected load while minimizing latency and controlling costs. This exercise typically involves load testing and benchmarking the model's performance under various resource configurations and simulated traffic patterns.

Factors such as the model's size, complexity, and the nature of the inference tasks (for example, text generation, question answering, summarization) play a crucial role in determining the resource requirements. Larger and more complex models generally require more computational power to achieve acceptable inference latencies, which can increase the overall cost of deployment and operation.

Organizations must carefully evaluate the trade-off between achieving low latency, which may require over-provisioning resources, and controlling costs, which may involve accepting slightly higher latencies. Finding the right balance is critical, as excessive latency can degrade the user experience, while over-provisioning resources can lead to unnecessary expenses.

On the other hand, batch integration patterns prioritize cost optimization and throughput for bulk processing of prompts. Instead of handling requests individually, these systems pool multiple prompts into batches, which are then sent to the generative model for inference. By processing prompts in batches, the computational resources can be utilized more efficiently, as the overhead associated with initializing the model and setting up the inference pipeline is amortized across multiple prompts.

This approach can lead to significant cost savings, especially when dealing with large volumes of prompts, as the computational resources are utilized more effectively, and the overall throughput is increased.

However, it's important to note that the batch processing approach introduces a trade-off between throughput and latency. While it optimizes for cost and overall throughput, individual requests may experience higher latency as they need to wait for a sufficient number of prompts to accumulate before being processed in a batch. As was mentioned before, batch integration patterns are better suited for scenarios where real-time responsiveness is less critical, such as batch text generation, document summarization, or other offline processing tasks.

Result post-processing

In real-time applications, the post-processing stage plays a crucial role in ensuring a smooth and engaging user experience. As the generated responses are intended for immediate delivery, the post-processing workflow prioritizes techniques that enable quick response filtering, ranking, and rendering. One common practice, particularly in conversational AI applications like chatbots, is to format the generated output using markup languages such as Markdown. This approach allows for the seamless integration of rich text formatting, including headers, lists, code blocks, and other structural elements, enhancing the readability and visual appeal of the responses.

Real-time post-processing may incorporate techniques tailored to the specific use case, such as sentiment analysis by applying color schemas as a background depending on the sentiment. For example, in a customer service chatbot, responses can be filtered and ranked based on their relevance to the customer's query, ensuring that the most appropriate and helpful responses are prioritized.

In contrast to real-time systems, batch processing workflows in GenAI systems afford more flexibility and computational resources for post-processing. As the generated outputs are not intended for immediate delivery, batch post-processing can apply more comprehensive and computationally intensive enhancements across aggregated outputs before persisting them to data stores or downstream systems.

One common batch post-processing technique is summarization, where the generated outputs are condensed into concise and coherent summaries, facilitating easier consumption and analysis. Structure extraction is another valuable post-processing step, where the system identifies and extracts relevant information, such as key entities, relations, or event descriptions, from the generated text. This structured data can then be stored in databases or used to populate knowledge graphs, enabling more efficient querying and analysis.

Batch post-processing can also incorporate deeper quality filtering mechanisms, leveraging techniques such as language models fine-tuned for quality assessment, natural language inference, or fact-checking. These advanced filtering methods can help identify and flag low-quality or factually incorrect outputs, ensuring that only high-quality and reliable information has persisted for downstream consumption.

Batch post-processing workflows can involve more complex transformations and enhancements, such as text stylization, sentiment transfer, or content generation for specific domains or formats (for example, generating marketing copy, product descriptions, or technical documentation).

Result presentation

Result presentation is arguably the starkest difference between paradigms. Real-time UI/API integration demands instantaneous updates – often server-rendered or via data-binding frameworks. In batch mode, you're more likely to bulk export results through pipelines into warehouses or operational data stores for asynchronous consumption in reporting, analytics, document systems, and more.

Naturally, the logging and monitoring requirements align closely with each mode's system characteristics. Real-time needs tight instrumentation around per-request metrics like latency, errors, and resource usage. Batch emphasizes throughput volume, pipeline performance, and data lineage observability.

From a data engineering perspective, real-time follows more of a lambda architecture with optimized speed and paths. Batch leans toward conventional modern data pipelines leveraging cloud storage, spark clusters, managed workflow orchestrators, and MPP data warehouse targets.

Real-time integration meshes seamlessly with stateful applications and provides very responsive UIs. Batch procession unlocks higher scalability for large, asynchronous AI-powered operations like document generation, report automation, and conversational data annotation at a tremendous scale.

Both patterns introduce their own unique supportability considerations too. Real-time depends on highly redundant, self-healing service meshes. Batch relies more on robust recovery orchestration, idempotent restarts, and automatic retries.

As you can see, while leveraging the same core GenAI capabilities, the two paradigms differ significantly in upstream/downstream architecture priorities and delivery characteristics. The right choice comes down to evaluating latency sensitivity, scale targets, cost parameters, and the use case fit. Many enterprises will likely embrace a hybrid mesh incorporating aspects of both patterns.

Use case example – search enhanced by GenAI

To illustrate a real-time and a batch use case, we are going to work on an example of a company that uses GenAI to enhance its website search experience. In this case, the document ingestion will be a batch process, and the search itself will be real-time.

Imagine a company that aims to enhance its website's search experience by leveraging **GenAI** technologies. In this scenario, the company's objective is to provide more comprehensive and relevant search results to its users, going beyond simple keyword matching and delivering contextually appropriate and natural language responses.

The document ingestion process, which involves indexing and processing the company's content corpus (for example, product descriptions, knowledgebase articles, product manuals), would be a batch operation. This step would involve techniques such as text extraction, entity recognition, topic modeling, and semantic embedding generation for the entire corpus of documents. The embeddings, which capture the semantic meaning and context of the documents, would then be stored in a vector database or other appropriate data store.

During the real-time search experience, when a user submits a query on the company's website, the query will undergo prompt pre-processing, which could include query rewriting, intent detection, and embedding generation. The generated query embedding would then be used to retrieve the most relevant documents from the vector database, based on semantic similarity. These retrieved documents would serve as the knowledge source for the GenAI model.

The GenAI model would then generate a natural language response based on the retrieved documents and the user's query. This response could take the form of a concise summary, a detailed answer, or even a conversational dialogue, depending on the requirements and the tone the company decides to set.

The real-time post-processing stage would then kick in, formatting the generated response for optimal presentation on the website. This could involve techniques such as response ranking, result structuring (for example, breaking down the response into sections or bullet points), and rendering with appropriate markup or visual elements.

By combining the batch processing of document ingestion with real-time query processing and generation, the company can deliver a seamless and enriched search experience to its users. The batch processing ensures that the company's content corpus is thoroughly indexed and semantically understood, while the real-time components leverage this knowledge to provide relevant and natural language responses tailored to each user's query.

Batch integration – document ingestion

The batch-processing portion of the document ingestion pipeline plays a crucial role in preparing the company's content corpus for effective search and retrieval. This stage involves several steps to extract meaningful information and convert it into a format suitable for efficient querying and generation:

1. **Data Extraction and Pre-processing**: The first step is to extract textual data from various sources, such as databases, content management systems, or file repositories. This data may come in various formats (for example, HTML, PDF, Word documents), requiring pre-processing techniques like text extraction, deduplication, and normalization to clean and standardize the input data.
2. **Metadata Extraction**: Once the text data is preprocessed, advanced natural language processing techniques, such as **named entity recognition** (**NER**) and entity linking, can be applied. These tasks can be executed either from predictive AI models or from GenAI models. This step identifies and extracts relevant entities (for example, people, organizations, products, locations) from the text, to be leveraged by linking them to external knowledge bases or ontologies, enriching the data with additional contextual information.
3. **Embedding Generation**: The heart of the batch processing stage is the generation of semantic embeddings for each document. These embeddings are dense vector representations that capture the contextual meaning and relationships within the text. Popular techniques like Transformer language models (for example, BERT and RoBERTa) or specialized embedding models (like, for example, the Google Vertex AI Embeddings for Text model and OpenAI text embeddings) are used to generate these embeddings.
4. **Vector Database Indexing**: The generated embeddings, along with the extracted entities, topics, and metadata, are stored in a specialized vector database or other suitable data store optimized for similarity search and retrieval. This indexed corpus serves as the knowledge base for the real-time search and generation process.

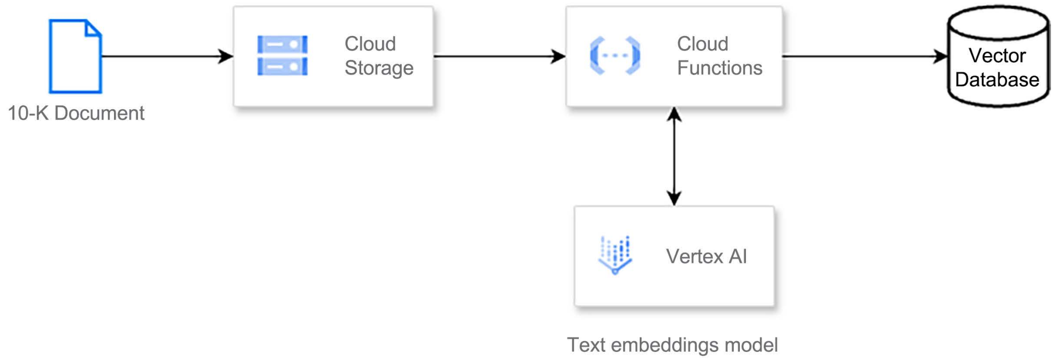


Figure 4.3: Document ingestion

By performing these batch-processing steps, the company's content corpus is transformed into a highly structured and semantically rich representation, enabling efficient retrieval, and providing the necessary context for the GenAI model to produce relevant and accurate responses during the real-time search experience.

Real-time integration – search

The real-time portion of the process handles the user's search queries and generates contextually relevant responses leveraging the knowledge base created during the batch processing stage.

At a high level, these are the components of the experience shown in *Figure 4.4*:

1. **Query processing**: This is illustrated in *Figure 4.4*, in *step 1*. When a user submits a search query on the company's website, the query undergoes pre-processing steps similar to those applied during the batch-processing stage. This may include text normalization, entity recognition, and embedding generation using the same models employed for the document corpus. In this step, you can also evaluate the query for safety.
2. **Semantic retrieval**: As depicted in *steps 2* and *3*, the generated query embedding is used to perform a similarity search against the vector database containing the indexed document embeddings. This step retrieves the most relevant documents from the corpus based on their semantic similarity to the user's query, ensuring that the retrieved information is contextually appropriate. In this step, you can also re-rank results depending on the use case and the metadata available.

3. **Prompt enrichment and generation**: In *step 4*, the retrieved documents, along with the original query and any additional context (for example, user profile or browsing history) are used to construct a rich prompt for the GenAI model. Techniques like prompt engineering, context augmentation, and RAG may be employed to create an informative and concise prompt that captures the essence of the user's information need.
4. **Response generation:** In *steps 5* and *6*, the GenAI model takes the enriched prompt as input and generates a natural language response. This response could be a concise summary, a detailed answer, or even a conversational dialogue, depending on the requirements and the model's capabilities.
5. **Post-processing and rendering**: In *step 6*, the generated response then goes through a post-processing stage, which may involve formatting, structure extraction, result ranking, and rendering for optimal presentation on the website. This could include techniques like extracting key points or summaries, highlighting relevant entities, and integrating visual elements of multimedia content to enhance the user experience.

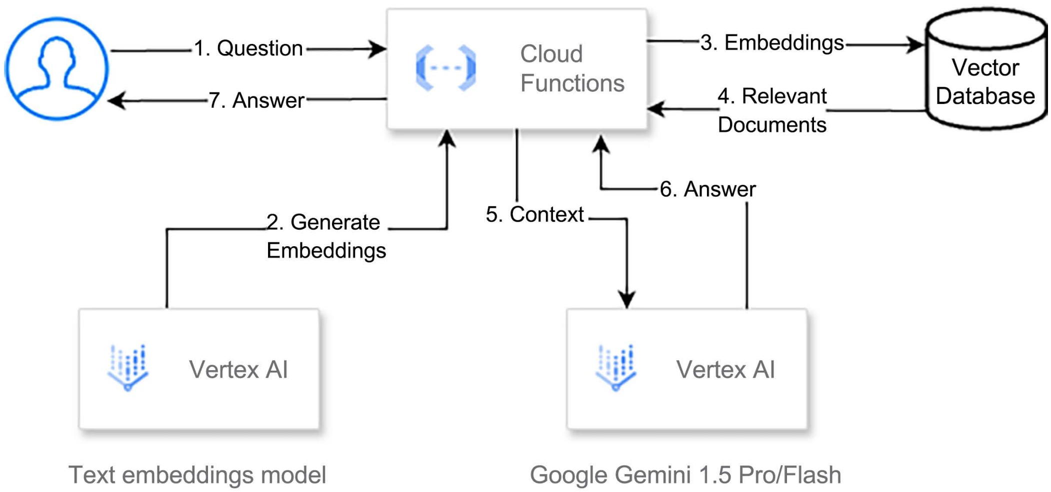


Figure 4.4: Real-time search architecture example

Summary

This chapter covered the two primary patterns for designing systems around LLMs – batch and real-time. The decision depends on your organization's use case requirements. We learned that batch mode involves sending queries in bulk for higher throughput at the expense of higher latency. It is better suited to long-running workloads and the consumption of a large corpus of data.

Results are not immediately exposed to users, allowing for additional review pipelines before or after model inference.

We also learned that real-time mode offers back-and-forth querying at a faster rate, providing quicker feedback to and from the end user. It has lower throughput but is better for low-latency requirements, but the opportunities to review results are reduced to prevent latency increases.

In this chapter, we addressed the implications of batch versus real-time processing on different components of the integration pipeline. For entry points, real-time optimizes for streamlined user prompting, while batch handles data pipeline inputs.

In pre-processing, real-time employs lighter techniques to minimize latency, whereas batch allows for heavier enrichment. Inference in real-time focuses on low latency per request, while batch processes requests in groups for improved throughput.

Post-processing in real-time involves quicker formatting and filtering, but batch processing allows for more complex transformations. In terms of presentation, real-time offers instantaneous UI updates, while batch exports results asynchronously.

Additionally, the chapter provided an example use case of using GenAI to enhance a website search, with document ingestion occurring in batch mode and search/response generation in real-time mode, transforming the end user experience, and obtaining more relevant and personalized answers.

In the next chapter, we will dive deep into a use case that leverages GenAI to extract data from 10-K documents.

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

`https://packt.link/genpat`

5

Integration Pattern: Batch Metadata Extraction

In this chapter, we will explore a metadata extraction use case, which serves as an excellent entry point to understand the capabilities of **Generative Artificial Intelligence** (**GenAI**). This topic is particularly relevant across industries and thought-provoking.

To illustrate this use case, let us consider a scenario where we work with a financial services company that requires the extraction of data from a 10-K report. These reports, filed annually with the **Securities and Exchange Commission** (**SEC**) by publicly traded companies, provide a comprehensive overview of their financial performance, operations, and significant events. They are extensive documents that are over 100 pages long and contain a wealth of information, structured across different sections across different data modalities (tables, text, etc.).

In this chapter, our objective is to identify the specific dataset and critical data points that need to be extracted from this vast repository of information. This process requires a systematic approach to pinpoint the desired data points with precision.

Once the relevant data points have been identified, we will determine the appropriate storage location for this extracted data. This could involve a centralized database accessible to authorized individuals within the organization, or a cloud-based repository that facilitates seamless access and collaboration across teams and locations.

Regardless of the chosen storage solution, GenAI will play a pivotal role throughout this endeavor. With its ability to understand and process natural language, GenAI will prove invaluable in navigating the intricate structure of financial reports, extracting the essential data points with remarkable efficiency and accuracy.

We will cover the following main topics in this chapter:

- **Use case definition**: We will describe an example scenario of extracting metadata from a 10-K report for a financial services company, explaining the structure and importance of these reports.
- **Architecture**: We will outline a cloud-based, serverless architecture using Google Cloud services to process 10-K reports, including storage, messaging, processing, and database components.
- **Entry point**: We will explain how the batch processing pipeline is triggered, using Google Cloud Storage and Cloud Functions to initiate the content extraction process.
- **Prompt pre-processing**: We will detail the creation of an initial prompt for the AI model, leveraging SEC guidance to identify key data points for extraction from 10-K reports.
- **Inference**: We will discuss the submission of the prompt to Gemini Pro 1.5 via Vertex AI, showcasing how the model processes and extracts information from the 10-K report.
- **Result post processing**: We will cover parsing the JSON output from the **large language model (LLM)** and the strategies to ingest the extracted data into databases, considering both relational and document database options.
- **Result presentation**: We will consider how to present the extracted data, including the use of business intelligence tools, data visualization platforms, and custom applications.
- **Code sample**: We will provide a practical implementation of the metadata extraction process, including setup, prompt creation, inference, and result handling.

Use case definition

Extracting metadata from 10-K reports holds significant value for financial services companies and other stakeholders. These reports, mandated by the SEC, are a treasure trove of information that can provide valuable insights into a company's financial health, operational performance, and strategic direction. However, the sheer volume, complexity of these documents, and lack of consistency across the way companies build their reports can make it challenging to manually extract and analyze the relevant data points.

Typical 10-K reports follow a standardized structure, comprising multiple sections that cover various aspects of a company's operations. These sections may include a business overview, risk factors, management's discussion and analysis, financial statements, and disclosures about corporate governance, among others. While the structure is consistent across companies, the specific data points and their presentation can vary, making it difficult to establish a one-size-fits-all approach to data extraction.

The 10-K includes five distinct sections:

- **Business**: This section provides an overview of the company's main operations, including its products and services, key markets, competitive landscape, and other relevant details about its business model and operations.
- **Risk factors**: In this section, the company outlines and discusses any, and all, risks it faces. This includes operational, financial, legal, regulatory, and industry-specific risks that could potentially impact its performance or future prospects.
- **Selected financial data**: This section presents specific financial information about the company over the last five years, typically including key metrics such as revenue, net income, earnings per share, and other relevant financial data.
- **Management's Discussion and Analysis (MD&A)**: The fourth section offers senior management's explanation and analysis of the company's financial results, including a detailed discussion of the factors that influenced its performance, future strategies, and potential opportunities and challenges.
- **Financial statements and supplementary data**: The final section furnishes the audited financial statements, including the income statement, balance sheets, statement of cash flows, and accompanying notes and disclosures. This section provides a comprehensive and detailed picture of the company's financial position and performance during the reporting period.

The biggest challenge to these documents is that specific data points and their presentation can vary, making it very difficult to establish a one-size-fits-all approach to data extraction. However, by leveraging GenAI's natural language processing capabilities, financial services companies can efficiently navigate this structured format and extract relevant data points from each section, tailoring their approach to the unique characteristics of each company's report.

The availability of structured metadata extracted from these documents opens up opportunities for financial services companies. For example, they can conduct an in-depth analysis of a company's financial performance, benchmarking it against industry peers or historical trends. This analysis can inform investment decisions, risk assessments, and strategic planning efforts.

Another opportunity would be to leverage the extracted metadata to develop predictive models and identify patterns that may not be immediately apparent from manual analysis. These models can help anticipate potential risks, identify emerging trends, and uncover new investment opportunities.

Thinking further, the extracted metadata can be integrated into existing data repositories or business intelligence platforms, enabling seamless access and collaboration among various teams within the organization. This integration can foster cross-functional collaboration, enabling different departments, such as investment banking, asset management, and risk management, to leverage the same data for their respective analyses and decision-making processes.

In addition to financial services companies, the extracted metadata can also be valuable for regulatory bodies, academic researchers, and other stakeholders interested in studying corporate performance, industry trends, and the overall health of the financial markets.

By leveraging the power of GenAI to extract metadata from 10-K reports, financial services companies can unlock a wealth of insights, streamline their analysis processes, and make more informed decisions that drive business growth and mitigate risks.

Architecture

Though the scope of this book is not to provide a deep dive into the intricacies of a LLM processing architecture, we will briefly discuss what a cloud-based architecture for our metadata extraction use case might look like. For this example, we will leverage the capabilities of Google Cloud, as it offers a native AI platform called Vertex AI that allows us to seamlessly integrate leading models, including Google's Gemini and third-party models such as Anthropic's Claude, in an enterprise-compliant manner.

The approach we'll adopt for this use case is to leverage a batch-optimized architecture, which is suitable for processing large volumes of data in an efficient and scalable manner. This kind of architecture aligns with cloud-native principles and is a serverless architecture that leverages various Google Cloud services.

This architecture will consist of an object store (Google Cloud Storage) to store the 10-K reports, a messaging queue (Google Cloud Pub/Sub) to coordinate the data flow, a processing component (Google Cloud Functions) to execute the LLM-based metadata extraction tasks, an LLM model (such as Google Gemini hosted on Vertex-AI) to perform the actual extraction, and a database (Google Big-Query) to store the extracted metadata.

Here's a more detailed breakdown of how this architecture will function:

1. The 10-K reports are stored in Google Cloud Storage, a highly scalable and durable object store.

2. A Cloud Function is triggered periodically (for example, daily or weekly) to initiate the metadata extraction process.
3. A Cloud Function will read a list of 10-K reports from Cloud Storage and publish messages to a Pub/Sub topic, effectively creating a queue of reports to be processed.
4. Another Cloud Function, subscribed to the Pub/Sub topic, is triggered for each message tied to a given report in the queue.
5. This second Cloud Function invokes the LLM model (for example, Google Gemini) hosted on Vertex-AI, passing the 10-K report content as input.
6. The LLM model processes the report, leveraging its natural language understanding capabilities to extract the relevant metadata.
7. The extracted metadata is then stored in a structured format (for example, BigQuery) for further analysis and consumption.

This serverless architecture provides several benefits, including automatic scaling, cost-efficiency (pay-per-use pricing), and seamless integration with other Google Cloud services.

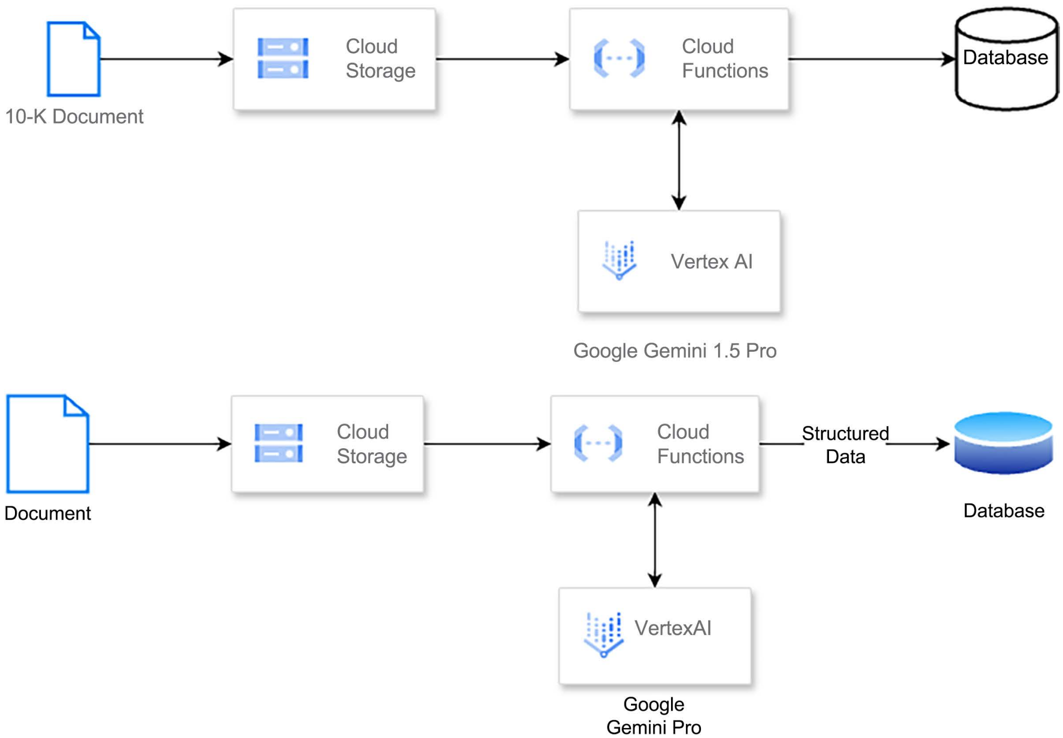


Figure 5.1: GenAI document data extraction pipeline

The following diagram showcases the architecture that will be leveraged in this example, following our GenAI integration framework discussed in *Chapter 3*, *Designing Patterns for Interacting with Generative AI*:

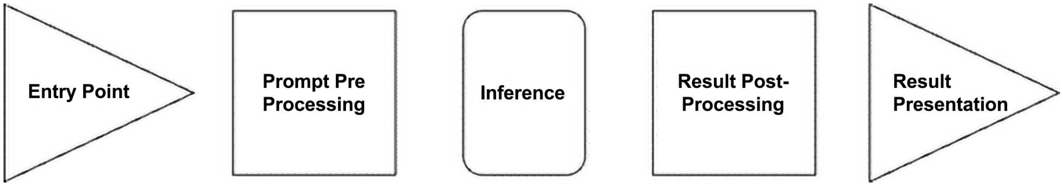


Figure 5.2: The Application Integration Framework

Entry point

The entry point for our batch processing pipeline will be an object created in **Google Cloud Storage (GCS)**, which will then trigger a Google Cloud Function to start the processing pipeline. This setup allows us to seamlessly integrate with existing workflows, where the 10-K reports are uploaded to a designated GCS bucket. By leveraging the event-driven nature of Cloud Functions, our system can automatically get into action as soon as a new report lands in the storage bucket.

Once triggered, the Cloud Function will initiate the content extraction process. For this step, we've decided to employ the powerful capabilities of Google's Gemini Pro 1.5, a state-of-the-art GenAI multimodal model that supports processing PDF documents directly. Gemini Pro 1.5 will analyze the uploaded 10-K report, intelligently extracting not only the textual content but also the relevant data points we're interested in, such as financial figures, company overviews, and key performance indicators.

By leveraging Gemini Pro 1.5's advanced natural language processing and document understanding capabilities, we can obtain a comprehensive transcript of the report's content. This transcript will serve as the foundation for further analysis and processing steps in our pipeline. Additionally, the extracted data points will be structured and organized in a format of our choosing (JSON, Markup, etc.), defined in the prompt, allowing us to seamlessly integrate them into our downstream systems to generate insightful summaries, visualizations, and other valuable outputs.

Prompt pre-processing

As a first step, we will start elaborating a prompt to extract the essential data points required to comprehensively understand a 10-K document. To guide our efforts, we will leverage a valuable resource provided by the SEC itself – a document titled *How to Read a 10k*, which is available on the SEC website (`https://www.sec.gov/files/reada10k.pdf`).

This SEC-provided document serves as a very useful roadmap, outlining the critical sections and information that investors and analysts should focus on when delving into a company's 10-K filing. By carefully studying this resource, we can identify the key data points that are most relevant and insightful, ensuring that our GenAI system extracts the information that truly matters.

To kickstart our data extraction process, we will create a simple yet effective prompt for our GenAI model. This initial prompt will serve as a starting point, instructing the model to identify and extract the specific data points outlined in the *How to Read a 10k* document. While this initial prompt may be concise, it will lay the foundation for more sophisticated prompts and fine-tuning techniques as our project progresses. The goal is to iteratively refine our prompts, leveraging the power of GenAI to extract the most crucial information accurately and efficiently from these complex financial documents.

```
prompt = """
You are an expert business analyst. Your task is to evaluate documents.
From the context document generate a JSON template with the important
data points to be extracted from the document, also remember to add an
attribute for the summary of the section.
The output should be a well-formed JSON document.

Context:
reada10k.pdf
Response:
"""
```

The following is example output from the previous prompt:

```
{
    "10k_template": {
        "part_I": {
            "item_1_business": {
                "description": "Company's main products and services,
subsidiaries, and markets",
                "recent_events": null,
                "competition": null,
                "regulations": null,
                "labor_issues": null,
                "operating_costs": null,
```

```
                "seasonal_factors": null,
                "summary": null
            },
            "item_1a_risk_factors": {
                "risk_factors": [],
                "summary": null
            },
            "item_1b_unresolved_staff_comments": {
                "comments": [],
                "summary": null
            },
            "item_2_properties": {
                "properties": [],
                "summary": null
            },
            "item_3_legal_proceedings": {
                "proceedings": [],
                "summary": null
            },
            "item_4": {
                "reserved": true
            }
        },
        "part_II": {
            "item_5_market_for_registrants_common_equity": {
                "market_information": null,
                "number_of_holders": null,
                "dividends": null,
                "stock_repurchases": null,
                "summary": null
            },
            "item_6_selected_financial_data": {
                "financial_data": {},
                "summary": null
            },
            "item_7_managements_discussion_and_analysis": {
                "operations_and_financial_results": null,
```

```
                "liquidity_and_capital_resources": null,
                "trends_and_uncertainties": null,
                "critical_accounting_judgments": null,
                "off_balance_sheet_arrangements": null,
                "contractual_obligations": null,
                "summary": null
            },
            "item_7a_quantitative_and_qualitative_disclosures_about_
market_risk": {
                "market_risk_exposures": null,
                "risk_management": null,
                "summary": null
            },
            "item_8_financial_statements_and_supplementary_data": {
                "income_statement": null,
                "balance_sheets": null,
                "statement_of_cash_flows": null,
                "statement_of_stockholders_equity": null,
                "notes_to_financial_statements": null,
                "auditors_report": null,
                "summary": null
            },
            "item_9_changes_in_and_disagreements_with_accountants": {
                "changes_in_accountants": null,
                "disagreements": null,
                "summary": null
            },
            "item_9a_controls_and_procedures": {
                "disclosure_controls_and_procedures": null,
                "internal_control_over_financial_reporting": null,
                "summary": null
            },
            "item_9b_other_information": {
                "other_information": null,
                "summary": null
            }
        },
```

```
        "part_III": {
            "item_10_directors_executive_officers_and_corporate_
governance": {
                "directors_and_executive_officers": [],
                "code_of_ethics": null,
                "board_qualifications": null,
                "summary": null
            },
            "item_11_executive_compensation": {
                "compensation_policies_and_programs": null,
                "executive_compensation": {},
                "summary": null
            },
            "item_12_security_ownership": {
                "ownership_information": {},
                "equity_compensation_plans": null,
                "summary": null
            },
            "item_13_certain_relationships_and_related_transactions": {
                "relationships_and_transactions": [],
                "director_independence": null,
                "summary": null
            },
            "item_14_principal_accountant_fees_and_services": {
                "fees_for_services": {},
                "summary": null
            }
        },
        "part_IV": {
            "item_15_exhibits_financial_statement_schedules": {
                "exhibits": [],
                "financial_statement_schedules": null,
                "summary": null
            }
        }
    }
```

Now that we have a template of what to look for in the actual 10-K, we can create a prompt to extract those data points:

```
You are an expert business analyst specializing in 10-K documents.

Your task is to extract information from 10-K documents; to achieve this
task, follow these steps:

Carefully analyze the document provided as context.
Use the template as a reference to understand which data points to
extract.
Never make up information; if you don't remember something, go back to
read the document. If the data is not available, add "Not available" as
the value.
Return a well-formed JSON document following the template.
Always think step by step.

<template>
{
    "10k_template": {
        "part_I": {
            "item_1_business": {
                "description": "Company's main products and services,
subsidiaries, and markets",
                "recent_events": null,
                "competition": null,
                "regulations": null,
                "labor_issues": null,
                "operating_costs": null,
                "seasonal_factors": null,
                "summary": null
            },
            "item_1a_risk_factors": {
                "risk_factors": [],
                "summary": null
            },
            "item_1b_unresolved_staff_comments": {
                "comments": [],
```

```
                "summary": null
            },
            "item_2_properties": {
                "properties": [],
                "summary": null
            },
            "item_3_legal_proceedings": {
                "proceedings": [],
                "summary": null
            },
            "item_4": {
                "reserved": true
            }
        },
        "part_II": {
            "item_5_market_for_registrants_common_equity": {
                "market_information": null,
                "number_of_holders": null,
                "dividends": null,
                "stock_repurchases": null,
                "summary": null
            },
            "item_6_selected_financial_data": {
                "financial_data": {},
                "summary": null
            },
            "item_7_managements_discussion_and_analysis": {
                "operations_and_financial_results": null,
                "liquidity_and_capital_resources": null,
                "trends_and_uncertainties": null,
                "critical_accounting_judgments": null,
                "off_balance_sheet_arrangements": null,
                "contractual_obligations": null,
                "summary": null
            },
            "item_7a_quantitative_and_qualitative_disclosures_about_
market_risk": {
```

```
                "market_risk_exposures": null,
                "risk_management": null,
                "summary": null
            },
            "item_8_financial_statements_and_supplementary_data": {
                "income_statement": null,
                "balance_sheets": null,
                "statement_of_cash_flows": null,
                "statement_of_stockholders_equity": null,
                "notes_to_financial_statements": null,
                "auditors_report": null,
                "summary": null
            },
            "item_9_changes_in_and_disagreements_with_accountants": {
                "changes_in_accountants": null,
                "disagreements": null,
                "summary": null
            },
            "item_9a_controls_and_procedures": {
                "disclosure_controls_and_procedures": null,
                "internal_control_over_financial_reporting": null,
                "summary": null
            },
            "item_9b_other_information": {
                "other_information": null,
                "summary": null
            }
        },
        "part_III": {
            "item_10_directors_executive_officers_and_corporate_
governance": {
                "directors_and_executive_officers": [],
                "code_of_ethics": null,
                "board_qualifications": null,
                "summary": null
            },
            "item_11_executive_compensation": {
```

```
                "compensation_policies_and_programs": null,
                "executive_compensation": {},
                "summary": null
            },
            "item_12_security_ownership": {
                "ownership_information": {},
                "equity_compensation_plans": null,
                "summary": null
            },
            "item_13_certain_relationships_and_related_transactions": {
                "relationships_and_transactions": [],
                "director_independence": null,
                "summary": null
            },
            "item_14_principal_accountant_fees_and_services": {
                "fees_for_services": {},
                "summary": null
            }
        },
        "part_IV": {
            "item_15_exhibits_financial_statement_schedules": {
                "exhibits": [],
                "financial_statement_schedules": null,
                "summary": null
            }
        }
    }
}
</template>
<document>



</document>

Response:
```

Note that the template has very specific instructions on what to do as well as what *not* to do. This is a best practice for prompting LLMs, as these models require such specific instructions in order to effectively give you precise information. A good analogy is to guide them as if they are a first-year student, offering clear instructions and providing as much context as possible.

Inference

For the inference, we are going to submit our prompt to Gemini Pro 1.5 available through Vertex-AI.

The Vertex AI Gemini API is tailored for developers and enterprises seeking to incorporate Gemini models into scaled deployments. This enterprise-grade offering provides a robust set of features designed to meet the demands of modern, high-performance applications. With the Vertex AI Gemini API, you can benefit from enhanced enterprise security measures, ensuring that your data and models are protected with industry-leading safeguards. Additionally, it offers data residency options, allowing you to comply with regional data storage and processing regulations:

```
{
    "10k_template": {
        "part_I": {
            "item_1_business": {
                "description": "Company's main products and services,
subsidiaries, and markets",
                "recent_events": null,
                "competition": null,
                "regulations": null,
                "labor_issues": null,
                "operating_costs": null,
                "seasonal_factors": null,
                "summary": null
            },
            "item_1a_risk_factors": {
                "risk_factors": [],
                "summary": null
            },
            "item_1b_unresolved_staff_comments": {
                "comments": [],
                "summary": null
```

```
            },
            "item_2_properties": {
                "properties": [],
                "summary": null
            },
            "item_3_legal_proceedings": {
                "proceedings": [],
                "summary": null
            },
            "item_4": {
                "reserved": true
            }
        },
        "part_II": {
            "item_5_market_for_registrants_common_equity": {
                "market_information": null,
                "number_of_holders": null,
                "dividends": null,
                "stock_repurchases": null,
                "summary": null
            },
            "item_6_selected_financial_data": {
                "financial_data": {},
                "summary": null
            },
            "item_7_managements_discussion_and_analysis": {
                "operations_and_financial_results": null,
                "liquidity_and_capital_resources": null,
                "trends_and_uncertainties": null,
                "critical_accounting_judgments": null,
                "off_balance_sheet_arrangements": null,
                "contractual_obligations": null,
                "summary": null
            },
            "item_7a_quantitative_and_qualitative_disclosures_about_
market_risk": {
                "market_risk_exposures": null,
```

```
                "risk_management": null,
                "summary": null
            },
            "item_8_financial_statements_and_supplementary_data": {
                "income_statement": null,
                "balance_sheets": null,
                "statement_of_cash_flows": null,
                "statement_of_stockholders_equity": null,
                "notes_to_financial_statements": null,
                "auditors_report": null,
                "summary": null
            },
            "item_9_changes_in_and_disagreements_with_accountants": {
                "changes_in_accountants": null,
                "disagreements": null,
                "summary": null
            },
            "item_9a_controls_and_procedures": {
                "disclosure_controls_and_procedures": null,
                "internal_control_over_financial_reporting": null,
                "summary": null
            },
            "item_9b_other_information": {
                "other_information": null,
                "summary": null
            }
        },
        "part_III": {
            "item_10_directors_executive_officers_and_corporate_
governance": {
                "directors_and_executive_officers": [],
                "code_of_ethics": null,
                "board_qualifications": null,
                "summary": null
            },
            "item_11_executive_compensation": {
                "compensation_policies_and_programs": null,
```

```
                    "executive_compensation": {},
                    "summary": null
                },
                "item_12_security_ownership": {
                    "ownership_information": {},
                    "equity_compensation_plans": null,
                    "summary": null
                },
                "item_13_certain_relationships_and_related_transactions": {
                    "relationships_and_transactions": [],
                    "director_independence": null,
                    "summary": null
                },
                "item_14_principal_accountant_fees_and_services": {
                    "fees_for_services": {},
                    "summary": null
                }
            },
            "part_IV": {
                "item_15_exhibits_financial_statement_schedules": {
                    "exhibits": [],
                    "financial_statement_schedules": null,
                    "summary": null
                }
            }
        }
    }
```

Note that there are configurations available to play with in the API. You can check the specific call in the code provided in the GitHub repository of the book.

Result post-processing

Once the LLM has processed the 10-K reports, it will return the results in a structured JSON format. This JSON document will contain the extracted data points, organized in a hierarchical manner that aligns with the structure of the 10-K report itself. To effectively utilize these results, we will need to parse the JSON document and extract the relevant information.

The next step in our pipeline is to ingest the parsed data into a database for efficient storage and retrieval. The specific ingestion strategy will depend on the type of database we choose to employ. For example, if we opt for a relational database, we will need to map the extracted data points to appropriate table structures, ensuring proper normalization and adherence to data integrity principles.

Alternatively, if we decide to use a document database, the ingestion process will be more straightforward, as these databases are designed to store hierarchical data structures, such as JSON documents, natively. In this case, we can directly ingest the parsed JSON results, leveraging the database's ability to efficiently store and query complex data structures.

Regardless of the database type chosen, it is crucial to design an ingestion strategy that ensures data consistency, scalability, and performance. This may involve implementing strategies such as bulk ingestion, indexing, and partitioning to optimize the database's performance and ensure efficient retrieval of the extracted data points.

In addition to storing the extracted data points, we can also consider generating embeddings for the various sections of the 10-K reports. Embeddings are vector representations of text that capture semantic meaning, enabling efficient similarity searches and retrieval. By generating embeddings for the report sections, we can integrate our dataset with a vector search pipeline, allowing users to perform advanced queries based on semantic similarity.

For a deep dive into embeddings generation and vector search integration, we will cover the **Retrieval Augmented Generation** (**RAG**) example in a dedicated chapter. This chapter will provide detailed insights into the process of generating embeddings, constructing vector databases, and implementing efficient vector search algorithms, enabling you to create powerful search and retrieval capabilities for your GenAI applications.

Result presentation

When it comes to presenting the results obtained from processing the 10-K reports, it's important to consider the fact that these results are ingested into a database. This means that the considerations you'll need to make are similar to those you would have when developing an experience that leverages data available in a database.

One of the primary considerations is the need for a tool or platform that can effectively aggregate and analyze data stored in a database. This could be a **business intelligence** (**BI**) tool, a data visualization platform, or even a custom-built application tailored to your specific needs. The chosen tool should provide robust querying capabilities, enabling you to extract and combine data from various tables or collections within the database.

Additionally, the presentation layer should offer a range of visualization options, such as charts, graphs, and dashboards, to effectively communicate the insights derived from the data. These visualizations should be interactive, allowing users to explore the data from different perspectives, filter and sort the results, and drill down into specific areas of interest.

Furthermore, the presentation layer should be designed with scalability and performance in mind. As the volume of data grows over time, the ability to handle large datasets and provide responsive user experiences becomes crucial. This may involve implementing techniques such as caching, indexing, and optimizing database queries to ensure efficient data retrieval and rendering.

On the GitHub directory for this chapter, you will find the complete code and an analysis of how all the layers described in this chapter fit together.

Summary

In this chapter, we explored metadata extraction from financial documents, specifically 10-K reports filed by publicly traded companies. We walked through the experience of working with a financial services firm that needs to extract key data points from these massive 10-K annual reports, leveraging the data extraction capabilities of LLMs.

We defined the use case, and we leveraged the power of GenAI to navigate through the structured sections of a 10-K, pinpointing and extracting the most relevant information nuggets, following the guidance provided by a best practices document. We walked through the process, starting by crafting an effective prompt to guide the AI model. This involved studying an SEC resource that outlines the critical sections and data points that investors should focus on. Armed with this knowledge, we can iteratively refine our prompts to ensure accurate and efficient extraction.

Then, we proposed a cloud-native, serverless architecture on Google Cloud to handle the batch processing of these documents. This scalable setup can leverage various services like Cloud Storage, Pub/Sub, and Cloud Functions, allowing us to seamlessly integrate the AI model and store the extracted data.

The chapter also touched on post-processing steps, such as ingesting the extracted data into a database (relational or document-based), potentially generating embeddings for vector similarity searches, and presenting the results through BI tools or custom applications with interactive visualizations.

In summary, this chapter offered you a practical blueprint to utilize GenAI to enhance the extraction and analysis of critical information from complex financial documents. It demonstrated how you can leverage this technology to make more informed decisions and uncover valuable insights, thereby optimizing your operational efficiency and strategic capabilities.

In the next chapter, we will examine a summarization use case. This example will illustrate another instance of what a batch processing use case could look like.

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

`https://packt.link/genpat`

6

Integration Pattern: Batch Summarization

In this chapter, we will explore the application of **Generative AI** (**GenAI**) to summarize documents, an invaluable capability across industries. However, before delving into a specific use case, it's essential to recognize that search intelligence powered by GenAI supports multiple use cases beyond document summarization. These include test case generation, document search supporting text, audio, and video-based data, as well as various analysis business use cases. Our focus, however, will be on a compelling use case within the financial services sector, where document summarization can streamline processes and enhance regulatory compliance efforts.

To illustrate this use case, let's consider a scenario where a financial services company is faced with the daunting task of reviewing a high volume of client applications. These applications often span multiple pages and encompass a wide range of information, including personal details, financial history, investment goals, and risk profiles.

Manually reviewing and distilling the key points from each application can be a time-consuming and error-prone process, especially when dealing with a large influx of submissions. This is where GenAI-driven summarization can be a game-changer, enabling efficient extraction of the most salient information while ensuring regulatory compliance by providing the ability not only to summarize but also to follow guidelines about what aspects to focus on, providing an elevated experience.

By leveraging GenAI models' natural language understanding and generation capabilities, companies can develop a system that intelligently analyzes regulatory documents, identifies the critical details, and generates concise summaries.

These summaries can then be seamlessly integrated into downstream processes, such as risk assessment, portfolio construction, or client onboarding workflows.

It is important to note that this use case is not intended to replace the work of a compliance officer but to enhance them through GenAI while optimizing their work in terms of quality and efficiency.

The chapter will build on the example by referring to the GenAI integration framework proposed in *Chapter 4*, *Generative AI Batch and Real-time Integration Patterns*. It starts by defining the use case of summarizing client applications in the financial services industry to streamline processes and enhance regulatory compliance efforts. It then proposes a cloud-native, serverless architecture on Google Cloud for batch processing these client applications.

We will then delve into the entry point of the pipeline, which is an object created in **Google Cloud Storage (GCS)**, triggering a cloud function to initiate the summarization process. It emphasizes the importance of prompt pre-processing, incorporating domain-specific knowledge and compliance guidelines into the prompts.

We will cover the inference phase, where the tailored prompts and client application content are submitted to Google Gemini on Vertex AI to generate concise summaries.

We'll discuss the result post-processing, ingesting the summaries into a database, and various approaches for presenting the summaries, such as dedicated applications or integration with existing systems. We'll also provide the sample code for implementing the proposed solution on Google Cloud.

Use case definition

In this example use case, we will work on a 10-K form summarization use case. The 10-K form aims to provide transparency into the company's financial health, operations, and risk factors for investors and regulators. The details can span hundreds of pages for large, complex companies.

Summarizing the extensive 10-K annual filings from public companies can unlock significant value for financial services firms. These lengthy documents, often spanning hundreds of pages, contain critical information about a company's business operations, financial performance, risk factors, and strategies. However, manually analyzing these filings is an extremely time-consuming and inefficient process. By leveraging advanced AI capabilities to generate concise summaries of 10-Ks, financial institutions can accelerate their analysis while ensuring consistent extraction of the most pertinent details. These summaries provide investment professionals with quick access to key financial metrics, competitive insights, potential risks, and future outlooks, enabling more informed investment decisions and portfolio monitoring.

Moreover, summarized 10-Ks can be seamlessly integrated into existing research workflows, regulatory compliance processes, and decision support systems within financial firms. This automated approach enhances scalability, allowing analysts to process large volumes of filings rapidly. It also mitigates the risk of human error or oversight that can occur during manual analysis. Consequently, financial services companies can leverage AI-driven 10-K summarization to gain a competitive edge through optimized investment analysis, improved risk management practices, and better-informed capital allocation strategies – ultimately contributing to enhanced returns and operational efficiency across their businesses.

Manually reviewing these forms can be a time-consuming and error-prone process, especially when dealing with a large volume of forms. Financial services professionals must carefully analyze each section, identify the key information, and ensure that the client's needs and risk profiles are accurately understood.

By leveraging a GenAI-driven approach to summarization, financial institutions can streamline various manual processes, extracting the most salient information from each form while adhering to regulatory compliance standards. The summaries can then enable financial advisors and portfolio managers to quickly grasp the client's situation and make informed decisions. The following diagram captures this by explaining the different components that will be used when designing the prompt.

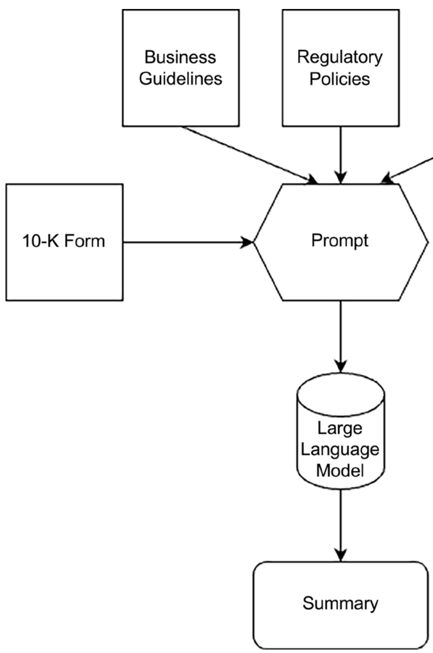


Figure 6.1: 10-K summarization prompt generation diagram

Furthermore, these summaries can be seamlessly integrated into downstream processes, such as risk assessment, portfolio construction, or client onboarding workflows, reducing the need for manual data entry and minimizing the risk of errors.

Compliance officers can also benefit from GenAI-driven summarization by incorporating specific rules and guidelines into the prompts. For example, the summarization model can be instructed to highlight potential red flags, such as inconsistencies in financial information or discrepancies between stated risk tolerance and investment objectives. By surfacing these issues promptly, compliance teams can proactively address them, ensuring adherence to regulatory requirements and mitigating potential risks.

Moreover, the summaries generated by GenAI models can serve as a valuable reference point for future client interactions, enabling financial advisors to quickly review the client's situation and provide personalized advice or recommendations, ultimately leading to improved client experiences and stronger relationships.

Architecture

Building upon the architecture discussed in *Chapter 5, Integration Pattern: Batch Metadata Extraction,* for the first batch processing example, we will adopt a similar cloud-native, serverless approach on Google Cloud to handle the batch processing of client applications for summarization. This scalable setup leverages various services, allowing us to seamlessly integrate the AI model and store the generated summaries.

The architecture will consist of the following components:

- **Object store** (**Google Cloud Storage**): This highly scalable and durable object store will be used to store client applications, which can be in various formats, such as PDFs, Word documents, or structured data files.
- **Messaging queue** (**Google Cloud Pub/Sub**): A messaging queue will be employed to coordinate the data flow and manage the processing of client applications.
- **Processing component** (**Google Cloud Functions**): Cloud Functions will serve as the processing component, executing the summarization tasks and invoking the LLM.
- **LLM** (**Google Gemini**): We will leverage a powerful LLM, such as Google Gemini, hosted on Vertex AI to perform the actual summarization.
- **Database** (**Google BigQuery** or **Cloud Firestore**): The generated summaries will be stored in a structured format, in either a relational database (BigQuery) or a document database (Cloud Firestore), depending on the specific requirements.

The interactions between these components are represented in the following diagram:

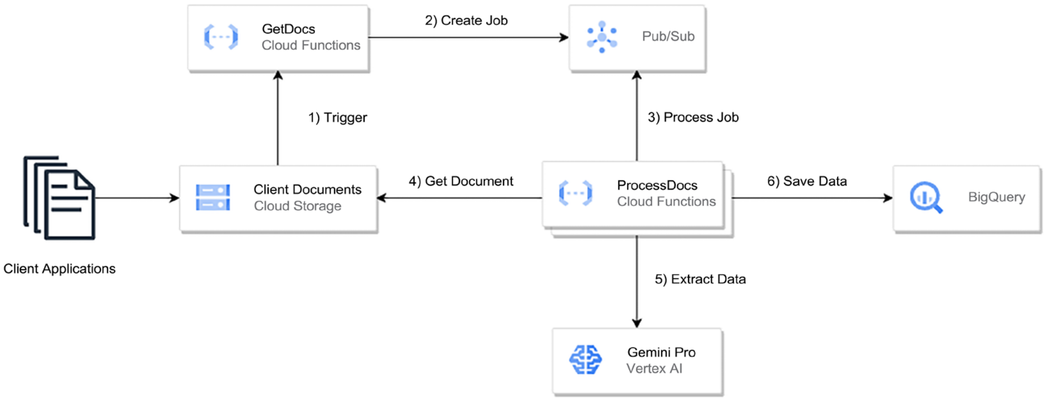


Figure 6.2: Batch summarization use case architecture diagram

Here's a breakdown of the flow of the architecture:

1. Client applications are uploaded to Google Cloud Storage, triggering a cloud function as soon as the documents are in the bucket.
2. The cloud function reads the list of applications from Cloud Storage and publishes messages to a Pub/Sub topic, effectively creating a queue of applications to be processed.
3. Another cloud function, subscribed to the Pub/Sub topic, is triggered for each message tied to a given application in the queue.
4. This second cloud function invokes the LLM (for example, Google Gemini) hosted on Vertex AI, passing the application content as input along with a tailored prompt.
5. The LLM processes the application, leveraging its natural language understanding capabilities to generate a concise summary while adhering to any specified rules or guidelines.
6. The generated summary is then stored in a structured format (for example, BigQuery or Cloud Firestore) for further analysis, integration, and consumption.

This serverless architecture provides several benefits, including automatic scaling, cost-efficiency (pay-per-use pricing), and seamless integration with other Google Cloud services. Additionally, it allows easy integration with existing workflows, enabling financial institutions to leverage GenAI-driven summarization without significant disruptions to their existing processes.

Entry point

Similar to the previous chapter, the entry point for our batch processing pipeline will be an object created in **Google Cloud Storage (GCS)**. As client applications are uploaded to a designated GCS bucket, a cloud function will be triggered, initiating the summarization process.

For this use case, we will leverage the powerful capabilities of Google Gemini, a state-of-the-art GenAI model renowned for its natural language understanding and generation abilities. Gemini will analyze the uploaded client application, intelligently extracting the most salient information and generating a concise summary.

To ensure regulatory compliance and adherence to specific guidelines, we will incorporate rules and instructions into the prompt provided to Gemini. These rules may include highlighting potential red flags, ensuring consistency between stated risk tolerance and investment objectives, or emphasizing specific sections of the application based on compliance requirements.

By combining Gemini's advanced reasoning capabilities with tailored prompts and rules, we can generate summaries that not only capture the essence of the client's application but also facilitate efficient review and decision-making processes while maintaining regulatory compliance.

Prompt pre-processing

We will start by crafting an effective prompt to guide the GenAI model in generating accurate and compliant summaries. In this case, we will leverage insights from compliance officers and subject matter experts within the financial services industry to understand the critical information that needs to be captured in the summaries.

It is important to remark that the intent behind this prompt is to provide an example of what compliance guidelines may look like, but in no way does this represent a real-world compliance example nor is it intended to be used to analyze business profiles.

Based on these insights, we will develop a template or a set of guidelines that outline the key sections and data points to be included in the summaries. This template will serve as a foundation for our prompt, ensuring that the summaries generated by the GenAI model align with the specific requirements of the financial services company.

Additionally, we will incorporate rules and guidelines provided by compliance officers to ensure that the summaries adhere to relevant regulations and industry best practices. These rules may include instructions for highlighting potential discrepancies, inconsistencies, or areas of concern that require further review or investigation.

As you can see, when designing our prompt and incorporating domain-specific knowledge and compliance guidelines, we can leverage the power of GenAI to generate summaries that are not only concise and accurate but also provide insights related to compliance and regulatory requirements, ultimately streamlining the application review process while reducing potential risks.

```
prompt_template_beggining = """
You are an expert compliance analyst.

Your task is to extract information from the sign up forms obtained.

1. Carefully analyze the document provided as context.
2. Write an intro paragraph for the company so the senior compliance
analyst can quickly understand who the customer is.
3. Following the guidance for approved businesses add a paragraph after
the intro explaining why the business is supportable or not.
4. Add another paragraph below explaining where the company is.
5. Always think step by step.

<supportable_businesses>
Summary of Prohibited & Restricted Businesses

This document outlines the types of businesses and activities that are
generally not allowed or require prior approval to use payment processing
services. These restrictions exist due to legal requirements, financial
partner rules, and compliance and risk management policies of payment
processors.

Types of Businesses:

High-Risk Jurisdictions and Persons: Businesses operating in or dealing
with sanctioned countries and individuals.
Prohibited Businesses: Activities involving illegal products or services,
adult content, certain financial services, marijuana, unfair practices,
weapons, etc.
Restricted Businesses: Activities requiring prior written approval from
the payment processor, such as CBD, financial products, pharmaceuticals,
non-fiat currency, and more.
```

```
Jurisdiction-Specific Prohibited Businesses: Additional restrictions based
on specific countries, such as India, Japan, Malaysia, Mexico, Singapore,
Thailand, UAE, and the US.

Prohibited Uses of Payment Processing Services:

Misrepresentation of identity or business.
Facilitating transactions for undisclosed merchants.
Using the service as a virtual terminal.
Processing without actual goods or services.
Evasion of chargeback monitoring.
Sharing cardholder information.
Misuse of intellectual property.

Prohibitions for Card Issuing Products:
Consumer use for personal expenses.
International use outside of the registered jurisdiction.
Lending without proper licensing and approval.
Abusive practices like free trial abuse and scalping.
Non-compliance with marketing and user experience guidelines.
Inactivity for 12 consecutive months.
Incorrect integration type for employee/contractor cards.

Business Analyst Recommendations:
Based on this information, a business analyst should pay close attention
to the following types of businesses:
Financial Services: This is a broad category with many restrictions and
nuances. It's crucial to understand the specific requirements for lending,
crowdfunding, money transmission, etc.
Regulated Industries: Industries like CBD, pharmaceuticals, and tobacco
require careful consideration of compliance and legal aspects.
High-Risk Businesses: Businesses prone to fraud or abuse, like travel
reservation services and multi-level marketing, need thorough risk
assessments.
Emerging Technologies: Activities involving cryptocurrency, NFTs, and
other new technologies should be evaluated based on current regulations
and the payment processor's policies.
```

```
Jurisdiction-Specific Restrictions: Businesses operating in or targeting
specific countries need to be aware of additional prohibitions and
requirements.
Additional Considerations:
Business Model: Restrictions often depend on the specific business model
and how the service is used.
Compliance: Understanding and adhering to relevant regulations is
essential to avoid account closure or other consequences.
Risk Management: Businesses should have robust risk management practices
to mitigate potential financial and legal risks.
Transparency: Maintaining clear and accurate information about the
business and its activities is crucial for approval and continued use of
payment processing services.
By carefully considering these factors, a business analyst can help ensure
that businesses comply with the payment processor's policies and operate
successfully within the platform.
</supportable_businesses>
<document>
"""
prompt_template_end="""
</document>

Response:

"""
```

Inference

For the inference phase, we will submit our prompt, along with the client application content, to the Google Gemini model available through Vertex AI. The Vertex AI platform provides a robust and scalable environment for deploying and managing GenAI models, ensuring high performance and enterprise-grade security.

In our example, we are using an example dataset of companies that signed up for our fictional financial services company. The inference code will work on the whole dataset, with delays to prevent consuming the available quota. Depending on your cloud provider or setup, you may have different inference quotas, which are generally related to **QPS** (**queries per second**) or **QPM** (**queries per minute**). In our example, we have a 6 QPM limit.

The following code snippet populates an array with the result text content. This can be further customized to evaluate safety attributes, for example, or to deal with empty results. For example, in the case of empty results, you could flag the doc and send it to a queue for further processing.

```
import time

summaries = []
for doc in docs:
  result = generate(prompt_template_beggining,str(doc),prompt_template_
end)
  #sleep 10 seconds to not exceed 6 QPM
  time.sleep(10)
  summaries.append(result.text)
  print(result.text)
```

When thinking about a more complex inference pipeline, we need to focus on the following steps:

1. The cloud function, triggered by the arrival of a new client application in GCS, retrieves the application content and the tailored prompt.
2. The cloud function invokes the Gemini model on Vertex AI, passing the application content and prompt as input.
3. Gemini processes the application, leveraging its natural language understanding capabilities to generate a concise summary while adhering to the specified rules and guidelines outlined in the prompt.
4. The generated summary is returned to the cloud function for further processing and storage.

By leveraging the power of Gemini and the scalability of Vertex AI, we can efficiently process large volumes of client applications, generating accurate and compliant summaries in a timely manner.

Result post-processing

Once the LLM has processed the client applications, it will return the summaries in a structured format, such as JSON or a markup language. The next step in our pipeline is to ingest these summaries into a database for efficient storage and retrieval.

The specific ingestion strategy will depend on the database type we choose to employ. For example, if we opt for a relational database like BigQuery, we will need to map the summary data points to appropriate table structures, ensuring proper normalization and adherence to data integrity principles.

Alternatively, if we decide to use a document database like Cloud Firestore, the ingestion process will be more straightforward, as these databases are designed to store hierarchical data structures natively. In this case, we can directly ingest the summaries in their original format, leveraging the database's ability to efficiently store and query complex data structures.

Regardless of the database type chosen, it is crucial to design an ingestion strategy that ensures data consistency, scalability, and performance. This may involve implementing strategies such as bulk ingestion, indexing, and partitioning to optimize the database's performance and ensure efficient retrieval of the summaries.

Result presentation

When it comes to presenting the summaries generated from client applications, several factors need to be considered, including the target audience, the intended use case, and the integration with existing systems and workflows.

One approach is to develop a dedicated application or dashboard that allows financial advisors, portfolio managers, and compliance officers to easily access and review the summaries. This application could provide features such as filtering, sorting, and searching capabilities, enabling users to quickly locate and analyze summaries based on specific criteria, such as client risk profiles, investment goals, or potential red flags identified by the GenAI model.

Additionally, the application could offer visualization tools to present the summary data in a more intuitive and visually appealing manner. For example, charts and graphs could be used to depict the client's financial profile, risk tolerance, and investment objectives, providing financial advisors with a comprehensive overview at a glance.

Another approach is to integrate the summaries directly into existing **customer relationship management (CRM)** systems or client onboarding workflows. This integration would allow financial advisors and compliance officers to access the summaries seamlessly within the tools and platforms they already use, minimizing disruptions to their existing processes.

Furthermore, the summaries could be leveraged for automation and decision support purposes. For instance, rules-based systems or machine learning models could be trained to analyze the summaries and provide recommendations or risk assessments, further enhancing the efficiency and accuracy of the client onboarding and portfolio management processes.

Regardless of the presentation approach chosen, it is essential to ensure that the summaries are presented in a secure and compliant manner, adhering to industry regulations and data privacy standards. Access controls and authorization mechanisms should be implemented to ensure that sensitive client information is only accessible to authorized personnel.

On the GitHub directory for this chapter, you will find the complete code and an analysis of how all the pieces described in this chapter fit together. Pay special attention to how every component of the framework interacts with each other.

Summary

In this chapter, we explored the application of GenAI to summarize client applications in the financial services industry. We kicked things off by defining the problem statement, where financial institutions need to efficiently review and distill key information from lengthy client applications while ensuring regulatory compliance.

We highlighted the value of GenAI-driven summarization in this context, enabling the extraction of salient details, streamlining downstream processes, and facilitating better-informed decision-making while adhering to compliance standards.

Next, we proposed a cloud-native, serverless architecture on Google Cloud to handle the batch processing of client applications for summarization. This scalable setup leverages various services, including Cloud Storage, Pub/Sub, Cloud Functions, and databases like BigQuery or Cloud Firestore, allowing for seamless integration of the AI model and storage of the generated summaries.

We then delved into the process of prompt pre-processing, emphasizing the importance of incorporating domain-specific knowledge and compliance guidelines into the prompts. By collaborating with subject matter experts and compliance officers, we can craft prompts that guide the GenAI model to generate accurate and compliant summaries.

The inference phase involved submitting the tailored prompts and client application content to Google Gemini on Vertex AI. Gemini's advanced natural language understanding and generation capabilities, combined with the carefully crafted prompts, enable the generation of concise and insightful summaries.

We also covered the post-processing steps, such as ingesting the summaries into a database, and discussed various approaches for presenting the summaries, including dedicated applications, dashboards, or integration with existing CRM systems and workflows.

Overall, this chapter provides a practical framework for leveraging the power of GenAI to streamline the review and analysis of client applications in the financial services industry, while ensuring regulatory compliance and enabling more efficient and informed decision-making processes.

In the next chapter, we will explore a real-time use case that focuses on intent classification with GenAI.

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

`https://packt.link/genpat`

7

Integration Pattern: Real-Time Intent Classification

In previous chapters, we discussed the batch-processing integration pattern, where we focused on efficiently processing large volumes of data and generating data to be used by downstream systems. In this chapter, we will shift our focus to **real-time integration patterns**.

Real-time interactions require applications to be optimized for latency, rather than processing large batch requests efficiently. In other words, we need to ensure that the output is generated as quickly as possible to provide an optimized user experience. The most common use case for this pattern is real-time agents exposed through chat or voice interfaces.

Let's consider an intent classification use case, which is a common scenario for chatbots. In this context, an **artificial intelligence** (**AI**) system is responsible for identifying the user's intent, such as checking a balance, scheduling an appointment, or making a purchase. Based on the identified intent, the system can then perform the appropriate tasks or provide relevant responses.

In today's application experiences, customers expect seamless and personalized experiences when interacting with businesses. One way to achieve this is by implementing an intelligent system that can accurately interpret user intents based on natural language inputs. This capability is particularly valuable in customer service, e-commerce, and conversational AI applications, where understanding the user's intent is crucial for providing relevant and contextual responses.

In this chapter, we'll explore a real-time intent classification use case, leveraging the power of Google's **Gemini Pro**, *a state-of-the-art* generative AI model, to build a system that can accurately categorize user inputs into predefined intents.

We'll walk through the entire process, from data preparation to deployment and integration with downstream systems, following the integration framework discussed in previous chapters.

In this chapter, we will cover:

- Use case definition
- Architecture
- Entry point
- Prompt pre-processing
- Inference
- Result post-processing
- Result presentation
- Full code

Use case definition

Let's consider a scenario where we're working with an e-commerce company that wants to improve its customer service experience. The company receives a large volume of customer inquiries through various channels, such as email, chat, and social media. Currently, these inquiries are handled manually by a team of customer service representatives, which can be time-consuming and prone to inconsistencies.

By integrating intent classification into customer engagement flows, companies can optimize their customer service operations. This advanced natural language processing technique automatically categorizes incoming customer inquiries into predefined intents, such as "order status," "product inquiry," "return request," or "general feedback." The classification layer acts as an intelligent entry point for customer service interactions, enabling more efficient and accurate routing of inquiries.

This automated categorization serves as the foundation for a scalable customer service infrastructure. Once an inquiry is classified, it can be seamlessly directed to the most appropriate team or agent, ensuring that customers receive expert assistance tailored to their specific needs. For high-volume, straightforward inquiries, the system can even trigger automated responses, providing instant solutions to common issues. This not only dramatically improves response times but also enhances overall customer satisfaction by delivering quick, relevant assistance.

Additionally, the implementation of intent classification significantly improves the quality of life for customer service agents.

By receiving pre-categorized inquiries, organizations can leverage specialist agents to focus on their areas of expertise, reducing the cognitive load of constantly switching between different types of issues. This specialization allows agents to provide more in-depth, high-quality support, leading to better resolution rates and increased job satisfaction.

There is an additional benefit in terms of analytics, as the data gathered from intent classification can offer valuable insights into customer needs and pain points, enabling companies to continually refine their products, services, and support strategies to better meet customer expectations.

In the following section, we will dive deep into an approach that will get you started on an intent classification example using GenAI.

Architecture

To build our intent classification system, we'll leverage a serverless, event-driven architecture built on **Google Cloud** (for example: `https://cloud.google.com/architecture/serverless-functions-blueprint`). This approach aligns with cloud-native principles and allows for seamless integration with other cloud services.

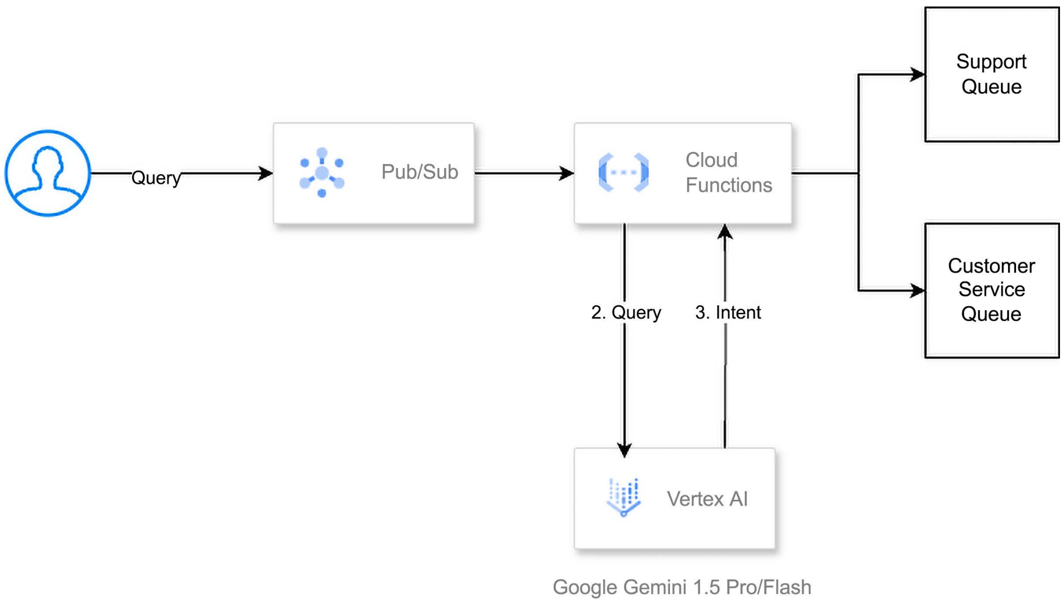


Figure 7.1: Intent classification example architecture diagram

The architecture consists of the following key components:

- **Ingestion layer**: This layer is responsible for accepting incoming user inputs from various channels, such as web forms, chat interfaces, or API endpoints. We'll use **Google Cloud Functions** as the entry point for our system, which can be triggered by events from services like **Cloud Storage**, **Pub/Sub**, or **Cloud Run**.
- **AI processing layer**: In this layer, we'll integrate Google's Gemini Pro through **Vertex AI**. Vertex AI provides a managed environment for deploying and scaling machine learning models, ensuring high availability and performance.
- **Intent classification model**: This is the core component of our system, responsible for analyzing the user input and determining the corresponding intent. We'll leverage Google Gemini Pro's natural language understanding capabilities for our intent classification model.
- **Orchestration and routing**: Based on the classified intent, we'll need to route the user input to the appropriate downstream system or service. This could involve integrating with **customer relationship management (CRM)** systems, knowledge bases, or other enterprise applications. We'll use Cloud Functions or Cloud Run to orchestrate this routing process.
- **Monitoring and logging**: To ensure the reliability and performance of our system, we'll implement robust monitoring and logging mechanisms. We'll leverage services like **Cloud Logging**, **Cloud Monitoring**, and **Cloud Operations** to gain visibility into our system's behavior and quickly identify and resolve any issues.

By adopting this architecture, the intent classification system won't just be scalable but also flexible enough to adapt to varying workloads and integration requirements. We'll be able to handle high volumes of customer inquiries in real time and deliver swift and consistent responses that improve the overall customer experience.

The serverless nature of this architecture brings several additional benefits. It allows for automatic scaling based on demand, ensuring that we can handle sudden spikes in customer inquiries without manual intervention. This elasticity not only improves system reliability but also optimizes costs, as we only pay for the resources we actually use.

This event-driven design facilitates easy integration with other systems and services. As our customer service ecosystem evolves, we can easily add new triggers or outputs to our intent classification system.

This could include integrating with new communication channels, connecting to additional backend systems, or incorporating advanced analytics for deeper insights into customer behavior and preferences.

In the following sections, we'll dive deeper into each component of our architecture, exploring the specific Google Cloud services we'll use, best practices for implementation, and strategies for optimizing performance and cost-efficiency. We'll also discuss a concrete example that will help you get started.

Entry point

For real-time interactive applications, the entry points where prompts originate need to be highly streamlined, with simplicity and ease of use in mind. These prompts often originate from unpredictable contexts, so interfaces have to feel natural across device types and usage scenarios.

In our use case, the entry point could be a web form, chat interface, or API endpoint where customers submit their inquiries. These inputs will be sent to a cloud function, which acts as the ingestion layer for our system.

Let's start with a sample user query:

```
#In this case we will simulate the input from a chat interface

message = "I want to open an account"
```

Prompt pre-processing

In a real-time system, every step in the prompt pre-processing workflow adds precious latency, commonly measured in milliseconds or microseconds depending on your application's SLAs, to the overall response time. Higher-latency experiences can be detrimental to the user experience. Therefore, pre-processing should be kept as lightweight as possible.

For our intent classification use case, the prompt pre-processing may involve simple text normalization, such as removing punctuation, converting to lowercase, or handling abbreviations. Additionally, we may apply some basic filtering to remove any potentially harmful or inappropriate content before sending the prompt to the model.

Let's dive deep into an example prompt:

```
#In this section we define the prompt, as the task is to perform intent
#classification we will identify the intent by exposing
#the possible values to the LLM
```

```
prompt_template = """
You are a helpful assistant for an online financial services company that
allows users to check their balances, invest in certificates of deposit
(CDs), and perform other financial transactions.

Your task is to identify what your customers are trying to do and return a
well formed JSON object.

1. Carefully analyze the content of the message.
2. Classify what the user is trying to do within these options:
   * New Account: The user is trying to sign up. Return {{"intent":
"signup", "content":"null"}}
   * Change Password: The user needs to reset their password. Return
{{"intent":"change_password", "content":"null"}}
   * Check Balance: The user needs to check their balance. Return
{{"intent": "check_balance", "content":"null"}}
   * Invest in CD: The user wants to invest in a certificate of deposit.
Return {{"intent": "invest_cd", "content": "Extract relevant information
such as investment amount and term"}}
   * Withdraw Funds: The user wants to withdraw money. Return {{"intent":
"withdraw_funds", "content": "Extract information like amount and
withdrawal method"}}
   * Transfer Funds: The user wants to transfer money between accounts.
Return {{"intent": "transfer_funds", "content": "Extract information like
amount, source account, and destination account"}}
   * Account Information: The user wants to access or update their account
information. Return {{"intent": "account_info", "content": "Identify the
specific information the user needs"}}
   * Lost/Stolen Card: The user wants to report a lost or stolen card.
Return {{"intent": "lost_card", "content": "null"}}
   * Support: The user needs help and is not sure what to do. Return
{{"intent": "support", "content": "null"}}
   * Other: For other queries, politely decline to answer and clarify what
you can help with.
3. Only return the proper JSON result from your classification.
4. Always think step by step.

User question: {query}
```

```
JSON:
"""
```

The previous prompt defines the template for the intent classification task. The prompt provides context that explains that the assistant is helping users of an online financial services company perform various actions, such as signing up, checking balances, investing in CDs, withdrawing funds, and more.

Additionally, this prompt instructs the model to carefully analyze the user's input message and classify the intent into one of the predefined categories. For each intent category, the prompt specifies the JSON object that should be returned, including any additional information that needs to be extracted from the user's message.

For example, if the user's intent is to invest in a CD, the assistant should return the JSON object in the following format:

```
{
   "intent": "invest_cd",
   "content": "Extract relevant information such as investment amount and
term"
}
```

This means that the virtual assistant should not only identify the intent as `"invest_cd"` but also extract relevant information like the investment amount and term from the user's message and include it in the `"content"` field.

The prompt also provides instructions for handling intents that do not fall into any of the predefined categories (the `"Other"` case).

By providing this detailed prompt template, the system can effectively guide the language model to perform the intent classification task for financial services scenarios, ensuring that the model's responses are structured and formatted correctly.

Inference

At the inference stage, we'll leverage Google's Gemini Pro model hosted on Vertex AI. Within the cloud function triggered by the user input, we'll invoke the Vertex AI endpoint hosting the Gemini Pro model, passing the pre-processed input as the prompt.

Gemini Pro will process the input and return the predicted intent, leveraging its natural language understanding capabilities. Since we're using an out-of-the-box model, the underlying infrastructure and resource allocation are abstracted away, ensuring that individual requests are processed efficiently while adhering to the service's performance and cost objectives:

```
generation_config = {
   "max_output_tokens": 8192,
   "temperature": 0,
   "top_p": 0.95,
}

safety_settings = {
   generative_models.HarmCategory.HARM_CATEGORY_HATE_SPEECH: generative_
models.HarmBlockThreshold.BLOCK_ONLY_HIGH,
   generative_models.HarmCategory.HARM_CATEGORY_DANGEROUS_CONTENT:
generative_models.HarmBlockThreshold.BLOCK_ONLY_HIGH,
   generative_models.HarmCategory.HARM_CATEGORY_SEXUALLY_EXPLICIT:
generative_models.HarmBlockThreshold.BLOCK_ONLY_HIGH,
   generative_models.HarmCategory.HARM_CATEGORY_HARASSMENT: generative_
models.HarmBlockThreshold.BLOCK_ONLY_HIGH,
}

def generate(prompt):
 vertexai.init(project=PROJECT, location=LOCATION)
 model = GenerativeModel(MODEL)
 responses = model.generate_content(
     [prompt],
     generation_config=generation_config,
     safety_settings=safety_settings,
     stream=False,
 )
 return(responses)

result = generate(prompt_template.format(query=message))
```

Result post-processing

For our intent classification use case, the post-processing step may involve formatting the predicted intent into a suitable response format, such as JSON or a human-readable string. Additionally, we may apply some basic filtering or ranking mechanisms to ensure that the most relevant and helpful responses are prioritized.

```
### Sometimes model return markdown friendly content, in this case we will implement a function to filter this.

def extract_json(text):
 """
 Extracts the JSON portion from a string containing backticks.

 Args:
   text: The string containing JSON data within backticks.

 Returns:
   A dictionary representing the extracted JSON, or None if no valid JSON is found.
 """
 start_index = text.find("```json")
 end_index = text.find("```", start_index + 7)  # +7 to skip "```json"

 if start_index != -1 and end_index != -1:
   json_string = text[start_index + 7: end_index]  # Extract the JSON string
 else:
   json_string = text
 try:
   json_data = json.loads(json_string)
   return json_data
 except json.JSONDecodeError:
   return None
```

The previous code snippet defines a function called `extract_json` that is designed to handle cases where the language model's output contains JSON data wrapped in backticks: `json```. This is a common practice in Markdown-friendly environments, where backticks are used to delineate code blocks or structured data.

The `extract_json` function takes a string text as input and attempts to extract the JSON portion from within the backticks. Here's a breakdown of how the function works:

1. The function first looks for the string "` ```json`" in the input text using the `find` method. This is the marker that indicates the start of a JSON block.
2. If the `start` marker is found, the function then looks for the closing `""` marker by searching for it from the end of the `json` marker (`start_index + 7`). If both the start and end markers are found, the function extracts the JSON string by slicing the input text between these markers. If no start or end markers are found, the function assumes that the entire input text is the JSON string.
3. The function then attempts to parse the extracted JSON string using the `json.loads` method from the `json` module. If the parsing is successful, the function returns the resulting JSON data as a dictionary. If the parsing fails (for example, due to invalid JSON syntax), the function returns `None`. By incorporating this function into the post-processing stage, the system can handle cases where the language model's output contains JSON data wrapped in backticks. This functionality can be particularly useful when working with Markdown-friendly environments or when integrating the intent classification system with other components that expect JSON-formatted data.
4. The post-processing stage can then proceed to format the extracted JSON data into a suitable response format, apply filtering or ranking mechanisms, and render the final response for display to the user.

   The `process_intent` function is designed to handle the JSON data returned by the intent classification model. It takes a dictionary intent as input, which is expected to have an "intent" key with a value representing the predicted intent category.

```
def process_intent(intent):
 if intent["intent"] == "signup":
   #If a user is trying to sign up you could
   #redirect the to a sign up page for example.
   return("Sign up process")
 elif intent["intent"] == "change_password":
   #If a user is looking into changing their password,
```

```
    #you could either do it through the chatbot,
    #or redirect to a password change page.
    return("Change password")
  elif intent["intent"] == "check_balance":
    #In this case you could have a function that
    #would query a database to obtain the
    #balance (as long as the user is logged in or not)
    return("Check account balance")
  elif intent["intent"] == "invest_cd":
    #For the investment intent, this could redirect
    #to a page where investment options can be selected.
    return("Invest in a CD")
  elif intent["intent"] == "withdraw_funds":
    return("Withdraw funds")
  elif intent["intent"] == "transfer_funds":
    return("Transfer funds")
  elif intent["intent"] == "account_info":
    return("Account information")
  elif intent["intent"] == "lost_card":
    return("Report lost card")
  elif intent["intent"] == "support":
    return("Contact support")
  elif intent["intent"] == "other":
    return("Other kind of intent")
  else:
    return("If a intent was classified as something else you should
investigate what is going on.")

intent = process_intent(extract_json(result.text))
```

The `process_intent` function checks the value of the `"intent"` key in the input dictionary. Depending on the intent category, the function performs a specific action or returns a corresponding message.

For example, if the intent is `"signup"`, the function returns the string `"Sign up process"`, which could be used to redirect the user to a sign-up page or initiate the sign-up process. Similarly, if the intent is `"change_password"`, the function returns `"Change password"`, which could trigger a password reset process or redirect the user to a password change page.

For intents like "check_balance", "invest_cd", "withdraw_funds", "transfer_funds", "account_info", "lost_card", and "support", the function returns corresponding messages that could be used to initiate the relevant processes or provide instructions to the user.

If the intent is "other", the function returns "Other kind of intent", indicating that the user's query did not match any of the predefined intent categories.

If the intent does not match any of the cases handled by the function, it returns a message suggesting that further investigation is needed to understand the intent.

Finally, the last line of code `intent = process_intent(extract_json(result.text))` combines the `extract_json` and `process_intent` functions. It first extracts the JSON data from the `result.text` string using `extract_json`. Then, it passes the extracted JSON data to the `process_intent` function, which processes the intent and returns an appropriate message or action.

This code snippet demonstrates how the intent classification system can be integrated with further processing steps to handle different user intents. The `process_intent` function can be extended or modified to include additional logic or actions based on the specific requirements of the application.

Result presentation

The result presentation stage for real-time applications demands instantaneous updates, often server-rendered or via data-binding frameworks.

In our use case, the formatted response containing the predicted intent can be sent back to the customer through the channel from which the inquiry originated (for example, web form, chat interface, or API response). This response can then be used to route the inquiry to the appropriate downstream system or provide an automated response for common intents.

In this example, we will use a Gradio interface to render the replies in a visually appealing UI. Gradio (https://www.gradio.app/) is an open-source Python package that allows you to quickly create easy-to-use, customizable UI components for your ML model, any API, or even an arbitrary Python function using a few lines of code.

> You can find more information about Gradio using the following links:
>
> Docs: https://www.gradio.app/docs
>
> GitHub: https://github.com/gradio-app/gradio

The following code provides an example that creates a Gradio interface:

```
import gradio as gr

def chat(message, history):
   response = generate(prompt_template.format(query=message))
   intent_action = process_intent(extract_json(response.text))
   history.append((message, intent_action))
   return "", history


with gr.Blocks() as demo:
 gr.Markdown("Fintech Assistant")
 chatbot = gr.Chatbot(show_label=False)
 message = gr.Textbox(placeholder="Enter your question")
 message.submit(chat, [message, chatbot],[message, chatbot]  )

demo.launch(debug=True)
```

The previous code illustrates the result presentation stage for the intent classification system using the Gradio library.

In our example, the `chat(message, history)` function is the core of the chatbot interface. It takes two arguments: `message` (the user's input message) and `history` (a list containing the previous messages and responses). Here's what the function does:

1. It calls the `generate` function (not shown in the provided code) to get the response from the intent classification model, passing the user's message as part of the prompt template. It then processes the model's response using the `extract_json` function (not shown) to extract the predicted intent data.
2. The extracted intent data is passed to the `process_intent` function (which is not shown) to determine the appropriate action or response based on the predicted intent. The user's message and the generated response are appended to the history list, which keeps track of the conversation.
3. The function returns an empty string for the response message and the updated history list.

4. The code then creates a Gradio interface using the `gr.Blocks` context manager. Inside the context, it does the following:

   - Displays a title using the gr.Markdown component.
   - Creates a `gr.Chatbot` component to display the conversation history.
   - Creates a `gr.Textbox` component for the user to enter their message.
   - Binds the chat function to the `submit` event of the `Textbox` component. When the user submits their message, the chat function is called with the user's message and the current history as arguments.
   - Updates the `Textbox` and `Chatbot` components with the new message and updated history, respectively.
   - Launches the Gradio interface in debug mode using `demo.launch(debug=True)`.

The result is an interactive chatbot interface where users can enter their messages as illustrated in *Figure 7.2*, and the system will process the message, predict the intent, and provide an appropriate response based on the `process_intent` function. The conversation history is displayed in the `Chatbot` component, allowing users to track the flow of the conversation.

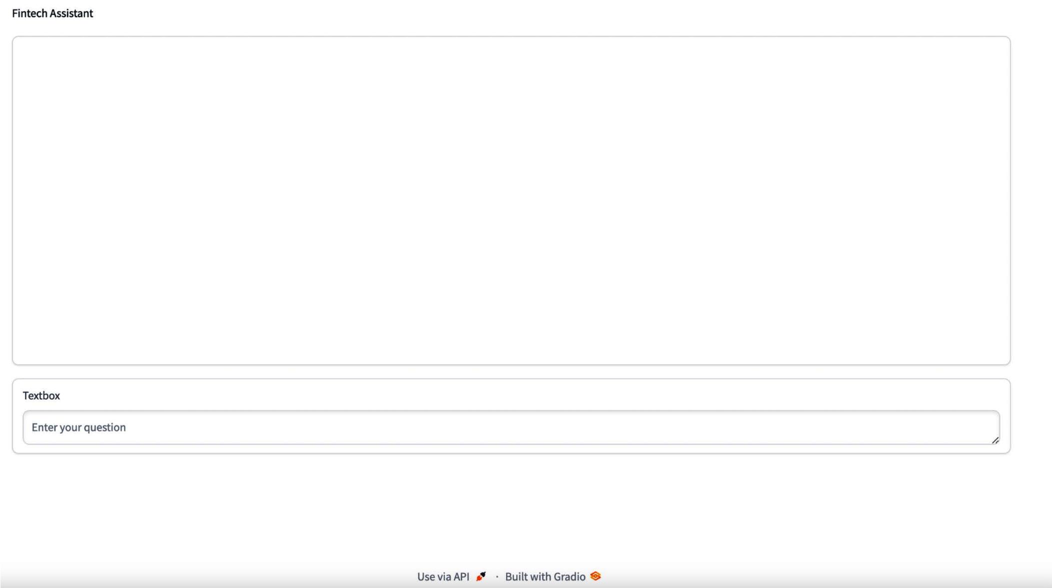


Figure 7.2: Example Gradio interface

Logging and monitoring

Real-time systems require tight instrumentation around per-request metrics, such as latencies, errors, and resource usage.

In our architecture, we'll leverage services like Cloud Logging (`https://cloud.google.com/logging/docs/overview`) and Cloud Monitoring (`https://cloud.google.com/monitoring/docs/monitoring-overview`) to gain visibility into the system's behavior and quickly identify and resolve any issues. We can monitor metrics like request latency, error rates, and resource utilization, and set up alerts for anomalies or performance degradation.

By following this integration pattern and leveraging the power of Google's Gemini Pro, businesses can unlock the power of generative AI to build intelligent systems that accurately classify user intents, enhance customer experiences, and streamline operations.

Refer to the GitHub directory of this chapter for the complete code that demonstrates how all the pieces described above fit together.

Summary

In this chapter, we discussed the integration pattern for building a real-time intent classification system using Google's Gemini Pro generative AI model. We started by introducing the concept of real-time integration patterns, which prioritize low latency over efficiency and volume, as opposed to batch-processing integration patterns.

The use case we developed is an e-commerce company that wants to improve its customer service experience by automatically categorizing incoming customer inquiries into predefined intents, such as order status, product inquiry, return request, or general feedback. This classification can then be used to route the inquiry to the appropriate team or provide automated responses for common issues.

The architecture proposed is a serverless, event-driven architecture on Google Cloud, consisting of an ingestion layer (Cloud Functions), an AI processing layer (Vertex AI with Gemini Pro), an intent classification model, orchestration and routing (Cloud Functions or Cloud Run), and monitoring and logging (Cloud Logging, Cloud Monitoring, and Cloud Operations).

In the next chapter, we will dive deep into another very important real-time use case, a **Retrieval Augmented Generation** (**RAG**) example where we are going to leverage generative AI to answer questions based on documents provided by us.

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

`https://packt.link/genpat`

8

Integration Pattern: Real-Time Retrieval Augmented Generation

In this chapter, we'll explore another integration pattern that combines the power of **Retrieval Augmented Generation** (**RAG**) and generative AI models to build a chatbot capable of answering questions based on the content of PDF files. This approach combines the strengths of both retrieval systems and generative models, allowing us to leverage existing knowledge sources while generating relevant and contextual responses.

One of the key advantages of the RAG approach is its ability to prevent hallucinations and provide better context for the generated responses. Generative AI models, trained on broad data, can sometimes produce responses that are factually incorrect or outdated due to their training data being limited to up to a point in time or they might lack proper context at inference time. By grounding the model's generation process in relevant information retrieved from a document corpus, the RAG approach mitigates the risk of hallucinations and ensures that the responses are accurate and contextually relevant.

For example, the term *refund* can have different meanings and implications in different contexts. A refund in the context of retail banking may refer to a customer requesting a refund for a fee or charge, while a refund in the context of taxation may refer to a tax refund from the government. By retrieving the relevant context from the document corpus, the RAG-powered chatbot can generate responses that accurately reflect the intended meaning of *refund* based on the specific context of the user's query.

The following image illustrates a simple RAG pipeline:

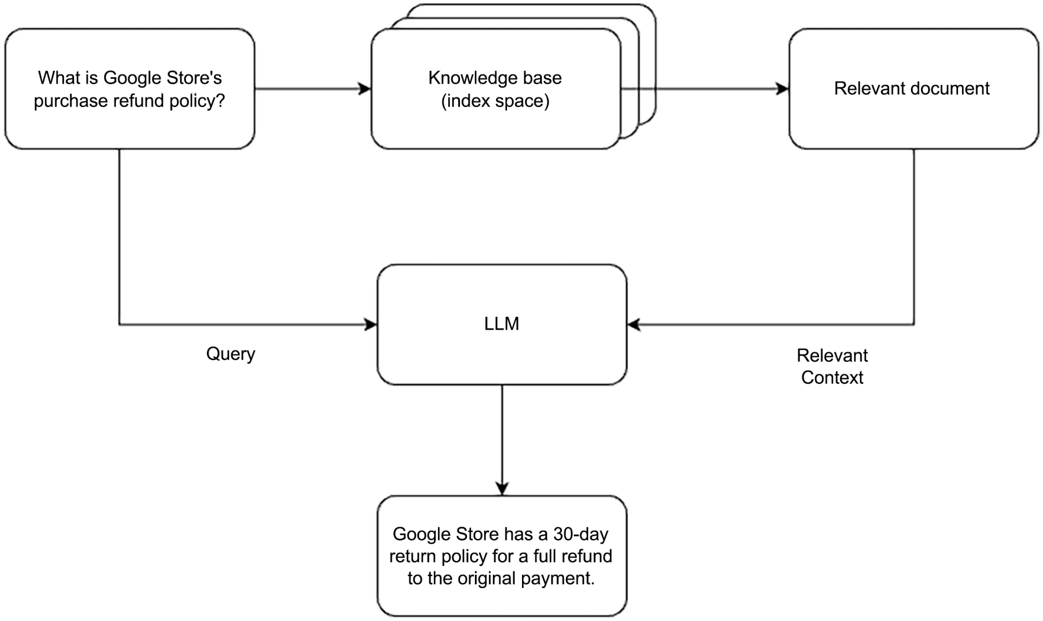


Figure 8.1: A simple RAG pipeline

Continuing our examples regarding financial services, these companies often deal with a vast amount of documentation, including legal contracts, regulatory filings, product disclosures, and internal policies and procedures. These document repositories can easily run into the tens of thousands or even hundreds of thousands of pages, making it challenging for employees and customers to quickly find relevant information when needed.

By implementing a RAG-based chatbot system, financial services companies can provide a user-friendly interface for employees, customers, and other stakeholders to ask natural language questions and receive concise, relevant answers derived from the vast collection of documents. The RAG approach allows the system to efficiently retrieve relevant information from the document corpus and then generate contextualized responses using a powerful generative AI model.

For example, a customer service representative could ask the chatbot a question about a specific clause in a loan agreement, and the system would retrieve the relevant section from the document corpus and generate a concise explanation tailored to the user's query. Similarly, an investment advisor could ask about specific regulations or guidelines related to a financial product, and the chatbot would provide the necessary information from the relevant documents.

By leveraging the RAG approach, financial services companies can greatly improve the accessibility and usability of their document repositories, enabling faster and more accurate information retrieval and reducing the time and effort required to manually search through thousands of pages of documentation.

In this chapter, we are going to cover:

- Use case definition (for a financial services company)
- Architecture (overview of a RAG-based chatbot system):
  - Ingestion layer
  - Document corpus management
  - AI processing layer
  - Monitoring and logging
- Entry point (a design for handling various input modalities)
- Prompt pre-processing and vector database integration
- Inference process using Vertex AI's Gemini 1.5 Flash model
- Result post-processing and presentation using Markdown
- A demo implementation (using Gradio)
- The full code example of the RAG pipeline

Use case definition

Let's consider a scenario where we're working with a large financial institution that deals with a vast number of legal contracts, regulatory filings, product disclosures, and internal policies and procedures. These documents individually can be into the tens or even hundreds of pages, making it challenging for employees, customers, and other stakeholders to quickly find relevant information when needed. These documents also do not have a consistent format in the way the information is reported, disqualifying non-AI-powered text extractor solutions like regex statements or plain business rules.

The institution wants to implement a chatbot system that can provide a user-friendly interface for users to ask natural language questions and receive concise, relevant answers derived from the organization's document corpus. This system should leverage the power of RAG to ensure that the generated responses are accurate, contextual, and grounded in the relevant information from the document corpus.

Architecture

To build our RAG-based chatbot system, we'll leverage a serverless, event-driven architecture built on Google Cloud. This approach aligns with the cloud-native principles we have used in previous examples and allows for seamless integration with other cloud services. You can dive deep into a Google Cloud example in this sample architecture: `https://cloud.google.com/architecture/rag-capable-gen-ai-app-using-vertex-ai`.

For the purpose of this example, the architecture consists of the following key components:

- **Ingestion layer**: This layer is responsible for accepting incoming user queries from various channels, such as web forms, chat interfaces, or API endpoints. We'll use Google Cloud Functions as the entry point for our system, which can be triggered by events from services like Cloud Storage, Pub/Sub, or Cloud Run.
- **Document corpus management**: In this layer, we'll store embeddings representing the content of the documents. In this case, we can use a wide range of solutions from purpose-built vector databases such as Chroma DB, Pinecone, or Weaviate, to well-known industry standards such as Elastic, MongoDB, Redis, or even databases known for other capabilities such as PostgreSQL, SingleStore, Google AlloyDB, or Google BigQuery.
- **AI processing layer**: In this layer, we'll integrate Google Gemini through Vertex AI. Once the results are retrieved from the vector database, they will be exposed to Google Gemini as context along with the prompt. This process can be handled by a Cloud function.
- **Monitoring and logging**: To ensure the reliability and performance of our system, you should implement robust monitoring and logging mechanisms. We'll leverage services like Cloud Logging, Cloud Monitoring, and Cloud operations to gain visibility into our system's behavior and quickly identify and resolve any issues.

Entry point

The entry point for a RAG-based chatbot system is designed to be user-friendly, allowing users to submit their natural language queries through various interfaces, such as web forms, chat applications, or API endpoints. However, the entry point should not be limited to accepting text-based inputs only; it should also handle different modalities, such as audio files or images, depending on the capabilities of the underlying language model.

In the case of models like Google Gemini (which support multimodal inputs), the entry point can directly accept and process text, audio, images, or even videos. This versatility enables users to interact with the chatbot system in a more natural and intuitive manner, aligning with the way humans communicate in real-world scenarios.

In cases where the language model does not natively support multimodal inputs, the entry point can still accommodate various input modalities by pre-processing the data and extracting the textual content. This approach ensures that the chatbot system remains accessible and user-friendly, catering to a diverse range of input formats while leveraging the capabilities of the underlying language model.

For text inputs, the entry point can simply pass the query directly to the subsequent phases of the RAG pipeline. However, when dealing with audio or image inputs, the entry point needs to perform additional processing to extract the textual content from these modalities.

For audio inputs, the entry point can leverage speech recognition technologies, such as Google Chirp, Amazon Transcribe, OpenAI Whisper, or open-source libraries like CMU Sphinx, to transcribe the audio data into text format. This process involves converting the audio signals into a sequence of words or phrases that can be understood by the language model.

Similarly, for image inputs, the entry point can employ **optical character recognition** (**OCR**) techniques to extract text from the provided images. This can be achieved by integrating with services like Google Cloud Vision API, Amazon Textract, or open-source tools like Tesseract OCR. These technologies leverage computer vision and machine learning algorithms to accurately identify and extract textual content from images, enabling the chatbot system to understand and process information presented in visual form.

In this example, we will leverage text; the Python code will look like this:

```
#In this case we will simulate the input from a chat interface

question = "What is this call about?"
```

Regardless of the input type, the entry point should be designed to handle a wide range of scenarios and input formats. It may need to perform additional pre-processing steps, such as noise removal, format conversion, or data cleaning, to ensure that the input data is in a suitable format for the subsequent phases of the RAG pipeline. It is also best practice to run the raw request through a rigorous security monitoring pipeline to prevent data leakage or model intoxication such as prompt ingestion.

The following website presents a very interesting point of view regarding the challenges posed by prompt injection in multimodal scenarios: `https://protectai.com/blog/hiding-in-plain-sight-prompt`.

Prompt pre-processing

For our example, we will need to construct our prompt in real time along with its necessary context and instructions. In this step of our RAG pipeline, we will utilize a vector database for efficient vector similarity search. Vector databases are specialized data stores designed to store and retrieve high-dimensional vectors, enabling fast and accurate similarity searches. Although there are numerous vector database providers available, we will use Chroma DB for this specific example.

The retrieval process in a RAG pipeline is relatively straightforward. First, we generate embeddings from the user's query using a pretrained language model or embedding technique. These embeddings are numerical representations of the query that capture its semantic meaning. Next, we perform a similarity search on the vector database using the generated query embeddings. The vector database will return vectors that are most similar to the query embeddings, along with their corresponding textual information or context. These vectors are associated with previously ingested text passages.

Different filtering strategies can be applied to refine the search results further. The specific parameters and techniques available may vary depending on the vector database provider being used. For instance, some vector databases provide a similarity score that measures the closeness between the query embeddings and the retrieved vectors. This score can be leveraged to identify and filter out vectors that fall below a certain similarity threshold.

Another common filtering strategy is to limit the number of results obtained from the vector database. This approach can be particularly useful when there are constraints on the maximum number of tokens that can be passed to the language model, either due to token limits imposed by the model or for cost optimization purposes. By limiting the number of results, we can control the amount of context information provided to the language model, ensuring efficient processing and cost-effective operation.

Once the results are filtered and the relevant textual information is obtained, it is used to construct the prompt that will be passed to the language model. In this example, we use the following prompt template:

```
prompt_template = """
You are a helpful assistant for an online financial services company that
allows users to check their balances, invest in certificates of deposit
(CDs), and perform other financial transactions.

Your task is to answer questions from your customers, in order to do so
follow these rules:

1. Carefully analyze the question you received.
2. Carefully analyze the context provided.
3. Answer the question using ONLY the information provided in the context,
NEVER make up information
4. Always think step by step.

{context}
User question: {query}
Answer:
"""
```

In our example, the prompt we are submitting to the LLM would look like:

```
You are a helpful assistant for an online financial services company that
allows users to check their balances, invest in certificates of deposit
(CDs), and perform other financial transactions.

Your task is to answer questions from your customers, in order to do so
follow these rules:

1. Carefully analyze the question you received.
2. Carefully analyze the context provided.
3. Answer the question using ONLY the information provided in the context,
NEVER make up information
4. Always think step by step.

<context>
```

```
---This information is contained in a document called coca_cola_earnings_
call_2023.pdf

 1-877-FACTSET www.callstreet.com

18 Copyright © 2001-2024 FactSet CallStreet, LLC

The Coca-Cola Co. (KO) Q1 2024 Earnings Call

Corrected Transcript 30-Apr-2024

Operator: Ladies and gentlemen, this concludes today's conference call.
Thank you for participating. You may now disconnect.

Disclaimer The information herein is based on sources we believe to be
reliable but is not guaranteed by us and does not purport to be a complete
or error-free statement or summary of the available data. As such, we do
not warrant, endorse or guarantee the completeness, accuracy, integrity,
or timeliness of the information. You must evaluate, and bear all risks
associated with, the use of any information provided hereunder, including
any reliance on the accuracy, completeness, safety or usefulness of such
information. This information is not intended to be used as the prim
ary basis of investment decisions. It should not be construed as advice
designed to meet the particular investment needs of any investor. This
report is published solely for information purposes, and is not to be
construed as financial or other advice or as an offer to sell or the
solicitation of an offer to buy any security in any state where such
an offer or solicitation would be illegal. Any information expressed
herein on this date is subject to change without notice. Any opinions or
assertions contained in this information do not represent the opinions or
beliefs of FactSet CallStreet, LLC. FactSet CallStreet, LLC, or one or
more of its employees, including the writer of this report, may have a
position in any of the securities discussed herein.

---
---This information is contained in a document called coca_cola_earnings_
call_2023.pdf
```

```
 All participants will be on listen-only mode until the formal question-
and-answer portion of the call. I would like to remind everyone that the
purpose of this conference is to talk with investors and, therefore,
questions from the media will not be addressed. Media participants should
contact Coca-Cola's Media Relations department if they have any questions.


. . .
---
</context>
User question: What is this call about?
Answer:
```

Inference

After constructing the prompt with the retrieved context and the user's query, the next step is to submit the formatted prompt directly to Vertex AI's API endpoint to be processed by Gemini 1.5 Flash. This is where the actual generation of the response takes place. In the following code snippet, the `generate()` function is responsible for sending the prompt to the Gemini 1.5 Flash model and obtaining the generated response:

```
#This is the section where we submit the full prompt and
#context to the LLM
result = generate(prompt)
```

The `generate()` function encapsulates the configuration and settings required for the generation process. It includes two main components: `generation_config` and `safety_settings`.

The `generation_config` dictionary specifies the parameters that control the behavior of the language model during the generation process. In this example, the following settings are provided:

```
generation_config = {
   "max_output_tokens": 8192,
   "temperature": 0,
   "top_p": 0.95,
}
```

From Google Gemini's documentation:

- `max_output_tokens`: Maximum number of tokens that can be generated in the response. A token is approximately four characters. 100 tokens correspond to roughly 60–80 words.

- `temperature`: The temperature is used for sampling during response generation, which occurs when `top_p` and `top_k` are applied. `temperature` controls the degree of randomness in token selection. Lower temperature values are good for prompts that require a less open-ended or creative response, while higher temperature values can lead to more diverse or creative results. A temperature of 0 means that the highest probability tokens are always selected. In this case, responses for a given prompt are mostly deterministic, but a small amount of variation is still possible.

  A temperature of 0 means the model will choose the most likely token based on its training data, while higher values introduce more randomness and diversity in the output.

- `top_p`: This parameter changes how the model selects tokens for output. Tokens are selected from the most to least probable until the sum of their probabilities equals the top-p value. For example, if tokens A, B, and C have a probability of 0.3, 0.2, and 0.1, respectively, and the top-p value is 0.5, then the model will select either A or B as the next token by using temperature and will exclude C as a candidate.

  In this case, it is set to 0.95, meaning that only the top 95% of tokens with the highest probabilities will be considered during generation.

Beyond the above, the `safety_settings` dictionary specifies the harm categories and corresponding thresholds for filtering potentially harmful or inappropriate content from the generated output. In this example, the following settings are provided:

```
safety_settings = {
   generative_models.HarmCategory.HARM_CATEGORY_HATE_SPEECH: generative_
models.HarmBlockThreshold.BLOCK_ONLY_HIGH,
   generative_models.HarmCategory.HARM_CATEGORY_DANGEROUS_CONTENT:
generative_models.HarmBlockThreshold.BLOCK_ONLY_HIGH,
   generative_models.HarmCategory.HARM_CATEGORY_SEXUALLY_EXPLICIT:
generative_models.HarmBlockThreshold.BLOCK_ONLY_HIGH,
   generative_models.HarmCategory.HARM_CATEGORY_HARASSMENT: generative_
models.HarmBlockThreshold.BLOCK_ONLY_HIGH,
}
```

These settings instruct the Gemini 1.5 Flash model to block only highly harmful content related to hate speech, dangerous content, sexually explicit content, and harassment. Any content that falls below the “high” harm threshold for these categories will be allowed in the generated output.

The `generate()` function creates an instance of the `GenerativeModel` class, passing the `MODEL` parameter; in this example, Gemini 1.5 Flash. It then calls the `generate_content()` method on the model instance, providing the prompt, generation configuration, and safety settings. The `stream=False` parameter indicates that the generation should happen in a non-streaming mode, meaning the entire response will be generated and returned at once:

```
def generate(prompt):
 model = GenerativeModel(MODEL)
 responses = model.generate_content(
     [prompt],
     generation_config=generation_config,
     safety_settings=safety_settings,
     stream=False,
 )
 return(responses)
```

The generated response is stored in the `responses` variable, which is then returned by the `generate()` function.

By submitting the formatted prompt to Vertex AI's API endpoint for Gemini 1.5 Flash, leveraging the provided generation configuration and safety settings, this RAG pipeline can obtain a contextualized and relevant response tailored to the user's query while adhering to the specified parameters and content filtering rules.

Result post-processing

After receiving the response from the language model, it is often desirable to present the output in a more structured and visually appealing format. Markdown is a lightweight markup language that allows you to add formatting elements such as headings, lists, code blocks, and more. In this example, we use Markdown formatting to enhance the presentation of the question, answer, and context:

```
#In this section you can format the answer for example with markdown
formatted_result = f"###Question:\n{question}\n\n###Answer:\n{result.
text}\n\n<details><summary>Context</summary>{context}</details>"
```

The breakdown of the components of this formatting is:

- "###Question:\n{question}"

  This part adds a level 3 Markdown heading (###) for the Question section, followed by the user's original query ({question}) on a new line (\n).

- "\n\n###Answer:\n{result.text}"

  After adding an empty line (\n\n) at the beginning of the section, this section creates another level 3 Markdown heading for the Answer section, followed by the generated response from the language model ({result.text}) on a new line.

- "\n\n<details><summary>Context</summary>{context}</details>"

  This part utilizes the Markdown <details> and <summary> tags to create a collapsible section for displaying the context information retrieved from the vector database. The <summary>Context</summary> text serves as the label for the collapsible section, and the actual context text ({context}) is enclosed within the <details> tags.

Result presentation

The Markdown class from the IPython.display module is a utility that allows you to display formatted Markdown content in a Colab notebook or other IPython environments. By passing the formatted_result string to the Markdown constructor, you create a Markdown object that can be rendered by the display function:

```
display(Markdown(formatted_result))
```

When you call display(), the notebook will render the Markdown-formatted content contained in the formatted_result string. This allows you to leverage the rich formatting capabilities of Markdown within the notebook environment.

The following is an example of the Markdown formatted output of our demo:

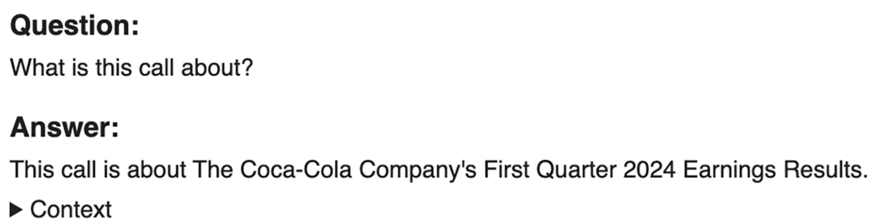


Figure 8.2: Screenshot from the formatted result in the Google Colab notebook

By using the `Markdown` class and the `display` function, you can take advantage of Markdown's formatting capabilities within the Google Colab notebook environment. This includes features like headings, bold and italic text, lists, code blocks, links, and more.

The rendered output will be displayed in the notebook cell, providing a visually appealing and well-structured representation of the question, answer, and context information. This can greatly enhance the readability and usability of the chatbot's responses, making it easier for users or developers to understand and interpret the results.

Additionally, the `<details>` and `<summary>` tags used in the `context` section create a collapsible section, allowing users to toggle the visibility of the context information. This can be particularly helpful when dealing with large amounts of context data, as it prevents cluttering the main output while still providing easy access to the relevant information.

In the next section, we will dive deep into how all these components work together through a use case demo.

Use case demo

The following is the code for building a demo using Gradio; in this case, we will use an additional function that will perform the RAG pipeline. When you run this code, a Gradio interface will open in your default web browser, displaying three main sections:

- **Fintech Assistant** heading
- Chatbot area
- Text input box

Users can type their questions into the input box and submit them. The chat function will be called, which will use the `answer_question` function to retrieve the relevant context from the vector database, generate an answer using the RAG pipeline, and update the chatbot interface with the user's question and the generated response.

The Gradio interface provides a user-friendly way for users to interact with the RAG pipeline system, making it easier to test and demonstrate its capabilities. Additionally, Gradio offers various customization options and features, such as support for different input and output components, styling, and deployment options. We start by installing Gradio:

```
#In this case we will use a Gradio interface to interact
#with the system
```

```
#Install Gradio

!pip install --upgrade gradio
```

Next, we define two helper functions that build upon the previously explained `generate()` function:

```
import gradio as gr

def answer_question(query, db, number_of_results):
 context = get_context(query, db, number_of_results)
 answer = generate(prompt_template.format(query=query, context=context))
 return(answer.text)

def chat(message, history):
   response = answer_question(message,db, MAX_RESULTS)
   history.append((message, response))
   return "", history
```

The `answer_question(...)` function takes three arguments:

- `query`: The user's question
- `db`: The vector database
- `number_of_results`: The maximum number of context results to retrieve from the database

It then calls the `get_context` function (not shown in the provided code) to retrieve the relevant context information – from the vector database, based on the user's query and the specified number of results. The retrieved context is then formatted within the `prompt_template` string and passed to the `generate` function – covered in the previous sections – to obtain the answer.

At the end of the function execution, the generated answer is returned as a string.

The `chat(...)` function takes two arguments:

- `message`: The user's question
- `history`: A list representing the conversation history

It then calls the `answer_question()` function with the user's question, the vector database (`db`), and the maximum number of results (`MAX_RESULTS`).

The generated response is appended to the history list, along with the user's question. The function returns an empty string and the updated history list, which will be used to update the chatbot interface.

The Gradio app

With the helper functions defined, we can now create the Gradio interface:

```
with gr.Blocks() as demo:
 gr.Markdown("Fintech Assistant")
 chatbot = gr.Chatbot(show_label=False)
 message = gr.Textbox(placeholder="Enter your question")
 message.submit(chat, [message, chatbot],[message, chatbot]  )

demo.launch(debug=True)
```

Here's what's happening in this code:

- `with gr.Blocks() as demo` creates a Gradio interface block called `demo`.
- `gr.Markdown(...)` displays a Markdown-formatted heading for the chatbot interface.
- `gr.Chatbot(...)` creates a Gradio chatbot component, which will display the conversation history.
- `gr.Textbox(...)` creates a text input box where users can enter their questions.
- `message.submit(...)` sets up an event handler for when the user submits their question. It calls the chat function with the user's input (message) and the chatbot instance and updates the message and chatbot components with the returned values.
- `demo.launch(...)` launches the Gradio interface in debug mode, allowing you to interact with the chatbot.

Refer to the GitHub directory of this chapter for the complete code that demonstrates how all the pieces described above fit together.

Summary

In this chapter, you've explored an integration pattern that combines RAG and generative AI models to build a chatbot capable of answering questions based on a document corpus. You've learned that RAG leverages the strengths of retrieval systems and generative models, allowing the system to retrieve relevant context from existing knowledge sources and generate contextual responses, preventing hallucinations and ensuring accuracy.

We proposed an architecture that utilized a serverless, event-driven approach built on Google Cloud. It consists of an ingestion layer for accepting user queries, a document corpus management layer for storing embeddings, an AI processing layer integrating with Google Gemini on Vertex AI, and monitoring and logging components. The entry point handles various input modalities like text, audio, and images, pre-processing them as needed.

You've learned that the core of the RAG pipeline involves generating embeddings from the user query, performing a vector similarity search on a vector database (in this example, we used Chroma DB), retrieving relevant context, formatting it into a prompt with instructions, and submitting it to the Gemini model on Vertex AI. The generated response can be post-processed, formatted with Markdown, and presented in a user-friendly interface using tools like Gradio.

You've now gained valuable insights into implementing a powerful RAG-based chatbot system. You've learned how to combine retrieval and generation techniques to provide contextual, hallucination-free responses, leveraging vector databases for semantic search through content embeddings. The chapter has equipped you with strategies to improve retrieval results and customize user experiences through prompt tuning. These skills will enable you to enhance your organization's ability to provide accurate and contextualized responses to natural language queries, effectively utilizing existing document repositories and the power of generative AI.

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

`https://packt.link/genpat`

9

Operationalizing Generative AI Integration Patterns

In previous chapters, we explored various integration patterns that leverage the power of **Generative AI** (**GenAI**) models like Google Gemini on Vertex AI. We discussed developing production-grade enterprise architectures according to targeted business use cases. In this chapter, we will discuss in depth best practices to be considered while operationalizing your GenAI integrations as production-grade applications. As we transition from conceptual design to real-world application, operational challenges such as scalability, reliability, and maintainability come to the forefront. In a nutshell, we are going to cover the following topics in this chapter:

- Introduction to operationalizing GenAI integration patterns
- A four-layer framework for GenAI operationalization
- Data layer:
    - Data quality and pre-processing
    - Data security and encryption
    - Data governance and versioning
    - Regulatory compliance (for example, GDPR or HIPAA)
    - Ethical considerations and bias mitigation
- Training layer:
    - Model adaptation strategies (few-shot learning, fine-tuning, and full training)
    - Model governance and policy establishment

    - Performance metrics and monitoring
    - Bias detection and mitigation
    - **Explainable AI** (**XAI**) techniques

- Inference layer:

    - Scalability and performance optimization
    - Security and access control
    - Model deployment strategies (for example, canary and blue-green)
    - Edge and distributed inference

- Operations layer:

    - **Continuous Integration and Continuous Deployment** (**CI/CD**) for GenAI
    - MLOps best practices
    - Monitoring and observability:

        - Evaluation and monitoring using "golden prompts"
        - Alerting systems
        - Distributed tracing
        - Comprehensive logging practices
        - Cost optimization strategies

- A real-world example: AI-powered language translation service
- Implementation across all four layers
- Specific considerations for each layer in the context of the example

Operationalization framework

In the rapidly evolving landscape of GenAI, it's crucial to have a structured approach to operationalizing your newly created applications. The operationalization framework we'll explore consists of four interconnected layers: **Data**, **Training**, **Inference**, and **Operations**. Together, these layers provide a comprehensive blueprint for effectively harnessing the potential of GenAI models in your applications. In the following list, we'll touch on what each of these four interconnected layers are:

1. **Data layer**: The foundation of any successful GenAI application lies in the quality and quantity of data. This layer encompasses data velocity, curation, prompt and training data pre-processing, and overall data management.

From a training perspective, ensuring that the data is relevant, diverse, and representative of the target domain is paramount. Techniques like distillation and filtering play a vital role in enhancing the quality of your training data. From an inference perspective, as mentioned in previous chapters, understanding your data velocity will help you define the application pattern your data is best fit for: batch vs real time.

2. **Training layer**: Once the data is prepared, the Training layer focuses on the intricate process of model training or fine-tuning. This layer involves selecting the appropriate training architecture, hyperparameter tuning, and leveraging cutting-edge training techniques such as transfer learning, few-shot learning, or self-supervised learning. Efficient resource management, including the utilization of distributed training and hardware acceleration, is crucial for optimizing the training or fine-tuning process.
3. **Inference layer**: After a model is trained or fine-tuned, the Inference layer comes into play. This layer encompasses the deployment and serving of the GenAI model in a production environment. Factors such as scalability, latency, and resource optimization are critical considerations. Advanced techniques like model quantization, pruning, and distillation can be employed to optimize the model's performance and memory footprint, ensuring efficient inference at scale.
4. **Operations layer**: The Operations layer focuses on the continuous monitoring, maintenance, and improvement of the deployed GenAI application. This layer involves tasks such as model monitoring, performance tracking, and model retraining pipelines. Robust logging and incident management processes are essential for ensuring the application's reliability and resilience. Additionally, this layer addresses critical aspects like model governance, ethical considerations, and regulatory compliance.

By understanding and effectively implementing each layer of this operationalization framework, organizations can unlock the full potential of GenAI applications, driving innovation and delivering exceptional user experiences. Seamless integration and collaboration across these layers are key to achieving successful and scalable GenAI applications.

These four layers build upon each other. To run inference against a model, you must have trained or fine-tuned it with carefully curated data. *Figure 9.1*, below, highlights the relationship between these layers.

Note that the Operations layer spans across all layers, providing you with a robust framework to deploy enterprise-level applications:

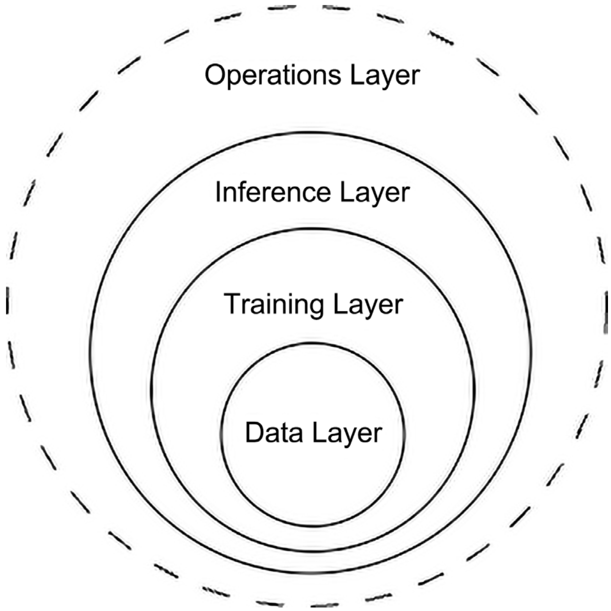


Figure 9.1: Interdependency between the four productionalization layers

In the following sections, we will dive deeper into each of the four layers of this productionalization framework. Let's start with the Data layer.

Data layer

The Data layer is the bedrock upon which your GenAI systems are built. It's not just about having data; it's about managing it effectively to ensure the quality, security, and ethical use of information. Robust data management processes are non-negotiable. Your GenAI systems are only as good as the data they interact with. For example, without enough contextual information, **Large Language Models** (**LLMs**) can hallucinate, and too much noise could cause the model to lose information in the middle, as described in the document *Lost in the Middle: How Language Models Use Long Contexts* (`https://arxiv.org/abs/2307.03172`). Therefore, you want to make a conscious effort to build and scale your data pipelines (RAG and fine-tuning) to feed the right level of detail and content to enhance your GenAI model's abilities.

We are going to provide an overview of the high-level components to keep in mind when preparing your data:

- **Data quality**: Implement rigorous procedures to clean, validate, and preprocess your data. This includes handling missing values, outliers, inconsistencies, and potential biases.
- **Data security**: Protect your data from unauthorized access and misuse. Encryption (at rest and in transit), access controls, and regular security audits are critical. Remember that your training data often contains sensitive information that requires the highest level of protection.
- **Data governance**: Keep track of changes to your datasets, models, and even the prompts used for generation. Versioning allows you to reproduce past results, trace the evolution of your systems, and troubleshoot issues more effectively. This is particularly important when dealing with regulatory compliance and audits.
- **Regulatory compliance**: Understand the legal landscape relevant to your industry and the types of data you handle. For example, in the case of **General Data Protection Regulation** (**GDPR**), if you're dealing with EU citizens' data, you must adhere to strict rules about data collection, processing, and storage, or with the **Health Insurance Portability and Accountability Act** (**HIPAA**) in healthcare, patient data protection is paramount.
- **Ethical considerations**: Go beyond the letter of the law and evolving regulations. Addressing issues like algorithmic bias, ensuring fairness, and being transparent about how your GenAI systems use data are crucial for building user trust and avoiding unintended negative consequences.
- **Cloud Data Loss Prevention** (**DLP**): This service helps you discover, classify, and protect sensitive data within your Google Cloud resources. It can automatically redact **Personally Identifiable Information** (**PII**) or alert you about potential data leaks.
- **Cloud Identity and Access Management** (**IAM**): Fine-grained control over who can access what data is essential. IAM lets you define roles and permissions with great precision.

A real-world example: Part 1

Let's explore an example. Imagine that you're building a GenAI application to assist doctors with diagnoses. Your Data layer would require you to make the following considerations:

- **Data quality**: Meticulously clean and de-identify patient records to remove personal information while preserving the medical relevance of the data. This process typically includes eliminating direct identifiers (like names and addresses), generalizing certain information (such as converting exact ages to age ranges), and carefully reviewing free-text fields for potentially identifying details.

The goal is to maintain the integrity and usefulness of the data while minimizing the risk of re-identification. An example of this would be converting a detailed patient record with name, exact age, and specific location into a de-identified version with a coded ID, age range, and generalized region.

- **Data security**: Always encrypt patient data, implement strict access controls, and potentially use Google Cloud's DLP to detect and protect sensitive health information.
- **Data governance**: Adhere to HIPAA regulations, document all data-processing activities, and proactively address potential biases in your inference contextual data that could lead to incorrect diagnoses.

Training layer

The Training layer is where your GenAI models come to learn how to behave in front of your customers, learning from the curated data and acquiring the skills needed to generate meaningful outputs. But it's not just about training; it's about governing, monitoring, understanding, and continuously improving these models. Model governance is a necessity for building trustworthy AI. Here are key strategies to consider:

- **Clear policies and guidelines**: Establish a framework that defines how models are developed, deployed, monitored, and updated. Address ethical considerations like fairness, transparency, and accountability. Document your decision-making processes for model selection, hyperparameter tuning, and data handling.
- **Responsible AI practices**: Implement techniques to detect and mitigate potential biases in your training data and model outputs. Regularly evaluate the impact of your models on different user groups and stakeholders. Consider using diverse teams to evaluate and audit your models.
- **Human-in-the-loop**: Design workflows that allow human reviewers to provide feedback and correct errors in model outputs, especially for high-stakes applications like healthcare or finance. This helps you build a safety net and continuously improve model performance.

Additionally, we need to think of models not as static pieces of code; they need ongoing care and attention. Over time, you will see a drift in model performance, which is normal, due to newly created data being available that your model has not seen at training time. A big advantage of LLMs over traditional ML models nowadays is their ability to process large corpora of information very efficiently at inference time. This opens new mechanisms that we can use to enhance our model performance without having to fully retrain our model:

- **Few-shot learning** tackles the challenge of training AI models with just a handful of examples per task category. Instead of memorizing specific data points, the model focuses on learning how to identify similarities and differences between examples. This allows it to generalize to new, unseen classes based on its ability to "learn to learn" from a small amount of information. This is particularly useful in scenarios where acquiring a lot of labeled data is difficult or expensive. This type of "learning" is done at inference time, meaning you add these examples in the prompt itself.
- **Fine-tuning** addresses the challenge of adapting a pre-trained language model to specific tasks or domains. Unlike few-shot learning, which occurs at inference time, fine-tuning involves additional training on a targeted dataset. This process allows the model to adjust its parameters and specialize in a particular area without losing its broad language understanding. Fine-tuning can significantly improve performance on domain-specific tasks, such as medical text analysis or legal document processing, by teaching the model relevant vocabulary, context, and nuances. This approach is particularly valuable when you have a moderate amount of task-specific data and want to create a more specialized model that outperforms generic prompting techniques. Recent advancements have introduced efficient fine-tuning methods like **Low-Rank Adaptation** (**LoRA**), which reduces the number of trainable parameters by adding small, trainable rank decomposition matrices to each layer. Other techniques such as prefix tuning (which prepends trainable parameters to the input), prompt tuning (which optimizes a small set of continuous task-specific vectors), and AdapterHub (which introduces small, trainable modules between existing layers) offer alternative ways to adapt models with minimal computational overhead. These methods enable more efficient and flexible adaptation of LLMs, making fine-tuning more accessible and resource friendly. Fine-tuning, especially with these efficient techniques, strikes a balance between the resource-intensive process of training a model from scratch and the limitations of using a general-purpose model for specialized applications.
- **Full training** of LLMs involves the comprehensive process of training a language model from scratch or significantly modifying an existing model's parameters across all layers. This approach requires a vast amount of diverse textual data and substantial computational resources, including high-performance GPUs or TPUs. Unlike fine-tuning, which adapts a pre-trained model for specific tasks, full retraining aims to create a model with broad language understanding and generation capabilities. This method allows for the incorporation of new knowledge, languages, or structural changes to the model architecture.

It's particularly useful when developing models for languages or domains not well represented in existing LLMs, or when aiming to reduce biases present in pre-trained models. Full retraining also enables the implementation of novel training techniques, such as constitutional AI or advanced prompt-engineering methods, into the training process. However, the enormous computational cost, time requirements, and potential for introducing new biases or errors make full retraining a challenging endeavor typically undertaken by large tech companies or well-resourced research institutions.

Having established the options available to train LLMs for specific tasks, our focus now shifts toward ensuring these models perform optimally and meet our expectations. This involves:

- **Performance metrics**, which are crucial for measuring the quality, accuracy, and efficiency of your models. It's essential to define and track relevant metrics, monitoring for signs of model drift and unexpected behavior. To address this, consider implementing automated monitoring systems that track key performance indicators over time, using A/B testing to compare model versions, employing cross-validation and bootstrapping techniques to assess model stability, implementing periodic re-evaluation on benchmark datasets to detect drift, and using confidence scoring to identify when a model is uncertain about its predictions.
- **Bias detection**, another critical aspect, which can be approached by utilizing fairness metrics such as demographic parity and equal opportunity, implementing adversarial debiasing techniques during training, using post-processing methods to adjust model outputs for fairness, conducting regular audits with diverse test sets, and employing techniques like counterfactual fairness to assess and mitigate hidden biases.
- **Hallucination detection**, which addresses the challenge of LLMs producing outputs that sound plausible but are factually incorrect. To combat this, consider implementing fact-checking algorithms that cross-reference model outputs with trusted knowledge bases, using ensemble methods to compare outputs from multiple models, employing uncertainty quantification techniques, implementing human-in-the-loop systems for critical applications, and using perplexity scores or other statistical measures to detect unusual or potentially hallucinated content.

- **Explainable AI** (**XAI**) techniques, which are vital for understanding why models make specific predictions, building trust, and identifying potential issues. Options in this area include implementing **Local Interpretable Model-Agnostic Explanations** (**LIME**), which creates interpretable models for local regions of the input space, or **SHapley Additive exPlanations** (**SHAP**), which uses game theory to assign importance values to features for feature importance analysis, using attention visualization techniques, employing counterfactual explanations, implementing concept activation vectors to understand high-level concepts learned by the model, and using layer-wise relevance propagation to trace the contribution of each input to the final prediction.

A real-world example: Part 2

To illustrate the practical implementation of the concepts we've discussed, let's consider a GenAI chatbot designed to handle customer inquiries. This example demonstrates how the various aspects of model training, governance, monitoring, and improvement come together in a real-world application:

- **Model adaptation strategies**: For our customer service chatbot to continue being effective and remain relevant, it's crucial to consider adapting the model to new information or changing requirements. To do this, we need to select the most appropriate method based on the specific needs of the business and the nature of the changes expected. Here, we explore three primary strategies – few-shot learning, fine-tuning, and full training – each suited to different scenarios of model adaptation:
    - **Few-shot learning**: This approach is ideal for quick adaptations to new types of customer inquiries or product updates. For instance, if your company launches a new product, you can provide the chatbot with a few examples of product-related questions and answers in the prompt. The chatbot can then generalize from these examples to handle a wider range of queries about the new product. This method is fast and doesn't require any model retraining, making it suitable for rapid responses to changing business needs. However, its effectiveness may be limited for more complex or nuanced changes. Specific limitations include difficulty with queries requiring deep context, inconsistent performance across different types of questions, scalability issues as topics increase, potential overfitting to provided examples, and inability to retain information from previous interactions.

- **Fine-tuning**: When you have a substantial amount of new data or need to adapt the chatbot to a significant shift in customer service patterns, fine-tuning becomes a valuable option. For example, if your company expands into a new market with different cultural norms and customer expectations, you could fine-tune the chatbot on a dataset of interactions specific to this market. This allows the model to adapt its language use and response style while retaining its general knowledge. Consider using efficient fine-tuning methods like LoRA to reduce computational requirements.
- **Full training**: While less common for a pre-existing chatbot, full training might be considered if there's a fundamental shift in the company's approach to customer service or if the original model is found to have significant limitations or biases. For instance, if the company decides to completely overhaul its product line and customer interaction style, or if it wants to build a chatbot from scratch that's deeply aligned with its unique brand voice and values, full training could be the best approach. This method allows for the incorporation of company-specific knowledge and interaction styles from the ground up but requires substantial computational resources and a large, high-quality dataset.

In practice, a combination of these approaches often yields the best results. You might use few-shot learning for day-to-day adaptations, schedule regular fine-tuning sessions (for example, monthly or quarterly) to incorporate accumulated new data, and reserve full training for major overhauls. By strategically employing these different adaptation methods, you can ensure that your customer service chatbot remains current, effective, and aligned with your business goals, while also managing computational resources efficiently.

Once we've chosen and implemented the most suitable adaptation strategy, we will ensure that these changes are effectively integrated into the ongoing operations of our chatbot, as follows:

- **Model governance**: Establish a comprehensive framework for the chatbot's operation. This includes defining guidelines for tone of voice, ensuring it aligns with your brand identity and customer expectations. Develop a clear set of acceptable responses and create escalation procedures for complex queries that require human intervention. Implement regular reviews of conversation logs to ensure adherence to ethical standards and maintain high customer satisfaction. Document decision-making processes for model selection, hyperparameter tuning, and data handling. Address ethical considerations such as fairness, transparency, and accountability in the chatbot's interactions. Consider using diverse teams to evaluate and audit the model, ensuring a wide range of perspectives are considered in the governance process.

- **Model monitoring**: Implement a robust system to track key performance metrics. This includes response accuracy (how often the chatbot provides correct and helpful information), resolution time (how quickly customer inquiries are resolved), and customer sentiment (measured through post-interaction surveys or sentiment analysis of customer responses). Utilize tools like Vertex AI Model Monitoring to detect shifts in user behavior or data patterns that could impact the chatbot's performance. Implement automated monitoring systems that track these key performance indicators over time. Use A/B testing to compare different versions of the chatbot and detect performance changes. Employ cross-validation and bootstrapping techniques to assess model stability. Implement periodic re-evaluation on benchmark datasets to detect drift in performance or unexpected behaviors.
- **Bias detection and mitigation**: Regularly assess the chatbot's performance across different user demographics to ensure fair and equitable service. Utilize fairness metrics such as demographic parity and equal opportunity. Implement adversarial debiasing techniques during training to reduce inherent biases. Use post-processing methods to adjust model outputs for fairness. Conduct regular audits with diverse test sets to identify potential biases in responses or handling of different customer groups.
- **Hallucination detection**: Implement fact-checking algorithms that cross-reference the chatbot's responses with a trusted knowledge base of company policies and product information. Use ensemble methods to compare outputs from multiple model versions. Employ uncertainty quantification techniques to flag responses where the model may be less confident. Implement human-in-the-loop systems for critical inquiries or when potential hallucinations are detected. Use perplexity scores or other statistical measures to identify unusual or potentially incorrect content in the chatbot's responses.
- **Explainable AI (XAI)**: Implement techniques to understand and explain the chatbot's decision-making process. Use LIME or SHAP for feature importance analysis to understand which parts of customer queries are most influential in generating responses. Employ attention visualization techniques to see which words or phrases the model focuses on. Use counterfactual explanations to understand how different inputs would change the chatbot's responses. This not only aids in debugging but also helps in training customer service representatives to work alongside the AI system.
- **Model updates and continuous learning**: Establish a regular schedule for training the chatbot on new customer interactions and feedback. This keeps its knowledge base up to date and improves its ability to handle diverse inquiries.

Utilize few-shot learning techniques to quickly adapt to new types of customer queries or product updates without full retraining. Consider periodic fine-tuning on recent, high-quality interactions to maintain peak performance. Implement a feedback loop where customer service representatives can flag incorrect or suboptimal responses, providing valuable data for future improvements.

By investing in these aspects of model management, monitoring, and continuous improvement, you'll build a GenAI chatbot that is not only powerful but also responsible, reliable, and adaptable to the ever-changing needs of your customers and your business. This comprehensive approach ensures that your AI-powered customer service solution remains a valuable, trustworthy, and effective tool in your customer engagement strategy.

Inference layer

The Inference layer is where your GenAI models come to life, transforming input data into meaningful outputs in real time. This layer is critical for delivering value to end-users and integrating AI capabilities into your applications and business processes. However, deploying and managing GenAI models at scale presents unique challenges that require careful consideration and planning:

- **Scalability and performance optimization**: Design your GenAI systems with scalability in mind, leveraging serverless and autoscaling capabilities offered by cloud providers. This ensures that your infrastructure can dynamically adjust to varying workloads, maintaining performance while optimizing costs. Implement load testing and capacity planning processes to ensure your systems can handle anticipated traffic patterns and sudden spikes in demand. This proactive approach helps prevent outages and maintains a seamless user experience. To further optimize resource utilization and manage costs effectively, explore techniques like request batching, caching, and load shedding. Request batching can significantly improve throughput by processing multiple requests together, while caching frequently accessed results reduces unnecessary model invocations. Load-shedding mechanisms can help gracefully degrade service during extreme traffic spikes, ensuring critical functions remain operational. Consider implementing queuing and buffering mechanisms to handle traffic spikes and prevent overloading your GenAI models or downstream components. This approach helps smooth out irregular traffic patterns and ensures consistent performance. Additionally, employ techniques like model quantization and distillation to reduce the computational requirements of your models without significantly impacting their accuracy.

- **Security and access control**: Implement robust security measures to protect your GenAI systems from unauthorized access, data breaches, and other security threats. This is particularly crucial given the sensitive nature of data often processed by AI models. Leverage IAM features provided by cloud platforms to control access to your GenAI resources and enforce least-privilege principles. This ensures that users and systems have only the permissions necessary to perform their required tasks. Implement secure communication channels and encryption mechanisms to protect data in transit and at rest. This includes using HTTPS for all API endpoints, encrypting data stored in databases or file systems, and implementing proper key management practices. Regularly review and update your security policies and procedures to address emerging threats and vulnerabilities. This may involve conducting periodic security audits, carrying out penetration testing, and staying informed about the latest security best practices in the AI field.
- **Monitoring and observability**: Implement comprehensive monitoring and observability solutions to gain real-time insights into your GenAI system's performance, health, and usage patterns. This includes tracking key metrics such as inference latency, throughput, error rates, and resource utilization. Use distributed tracing to understand the flow of requests through your system and identify bottlenecks or inefficiencies. Set up alerting mechanisms to promptly notify your team of any anomalies or performance issues. This allows for rapid response and mitigation of potential problems before they impact end-users. Consider implementing A/B testing capabilities within your Inference layer to compare the performance of different model versions or configurations in real-world scenarios.
- **Compliance and governance**: Ensure that your Inference layer adheres to relevant regulatory requirements and industry standards. This may include implementing data retention policies, maintaining audit logs of model predictions, and providing mechanisms for explainability and transparency in AI decision-making processes. Develop clear policies and procedures for model deployment, versioning, and rollback. This helps maintain consistency and reliability in your AI services while allowing for rapid iteration and improvement. Implement robust CI/CD pipelines that include automated testing, security scanning, and performance benchmarking to ensure that only high-quality, secure model versions are deployed to production.
- **Edge and distributed inference**: Consider implementing edge inference capabilities for scenarios where low latency or offline operation is crucial. This may involve deploying optimized versions of your models to edge devices or implementing hybrid cloud-edge architectures. Explore techniques like federated learning and split inference to balance privacy, performance, and resource constraints in distributed AI systems.

By addressing these aspects comprehensively in your Inference layer, you'll build GenAI systems that are not only powerful and efficient but also secure, scalable, and reliable. This holistic approach ensures that your AI-powered solutions can deliver consistent value to users while adapting to changing requirements and emerging challenges in the dynamic field of AI.

A real-world example: Part 3

Let's look at an example where your company has deployed an advanced AI-powered language translation service that has been developed to offer real-time, high-quality translations across multiple languages for text, voice, and video content. This cloud-based service caters to a diverse user base, including businesses, government agencies, and individuals worldwide, with usage patterns that vary significantly throughout the day and across different regions.

The service is deployed on a cloud platform using a serverless architecture with autoscaling capabilities to ensure scalability and optimize performance. This infrastructure allows the system to automatically scale up during periods of high demand, such as major international events, by spinning up additional inference instances as needed. For text translations, request batching is implemented, processing multiple short translations in a single model inference to improve throughput. Frequently requested translations are cached to reduce model invocations. For video translation, a queuing system manages large translation jobs, ensuring fair resource allocation and preventing system overload. In scenarios requiring ultra-low latency, optimized models can be deployed directly to edge devices, enabling offline translation capabilities.

Security and access control are fundamental to the service's design. All API requests require robust authentication, with different access levels for various user tiers. Data protection is ensured through encryption, both in transit and at rest. **Role-Based Access Control** (**RBAC**) allows enterprise clients to manage user permissions granularly. A dedicated security operations center monitors for unusual patterns or potential breach attempts, using AI-powered anomaly detection systems to maintain robust security.

Comprehensive monitoring and observability systems are in place to maintain the service's performance and reliability. Operations teams use customized dashboards showing key metrics like translation latency, accuracy scores, and resource utilization across different language pairs and content types. Distributed tracing allows for end-to-end tracking of each translation request, enabling quick identification of bottlenecks or errors. New model versions are gradually rolled out through A/B testing, with performance constantly compared against the current production model. Automated alerts are set up for various thresholds to ensure prompt attention to any issues.

Compliance and governance are integral to the service's operations. Strict data retention policies are implemented to comply with privacy regulations, with user content automatically deleted after translation unless explicitly requested otherwise. Detailed logs of translations (metadata only, not content) are maintained for compliance and billing purposes. A rigorous approval process governs the deployment of new model versions, including automated tests for accuracy, bias, and performance. An explainability feature is available, highlighting which parts of the input most influenced the translation output, enhancing transparency and trust.

The service's architecture incorporates edge and distributed inference capabilities. An on-premises solution is offered for clients with strict data sovereignty requirements. A lightweight version is available as an SDK for mobile app developers, enabling basic translation capabilities even when offline. To improve translations for rare languages or specific domains, a federated learning system is implemented, allowing the model to learn from user corrections without centrally collecting sensitive data.

A continuous improvement cycle is maintained for the service. Daily automated scripts analyze performance metrics and user feedback. Weekly reviews of aggregated performance data prioritize improvements for the most used language pairs and content types. Monthly A/B tests are conducted for model updates, with successful improvements gradually rolled out. Quarterly comprehensive security audits and penetration testing ensure the system remains robust against evolving threats.

This multi-faceted approach to the Inference layer ensures that the AI translation service remains highly available and performant, capable of handling a global scale with consistent low-latency translations. The service is secure and compliant, meeting the stringent requirements of enterprise clients and regulatory bodies. It is also observable and adaptable, allowing for the rapid identification and resolution of issues, as well as continuous improvement. With its edge capabilities and distributed learning, the service is positioned to adapt to evolving market needs and technological advancements, making it a flexible and future-proof solution in the dynamic field of natural language processing.

Operations layer

The Operations layer forms the backbone of a robust and efficient GenAI system, ensuring smooth functioning, reliability, and cost-effectiveness. This layer encompasses the critical processes and tools that enable continuous improvement, monitoring, and optimization of your AI models in production environments.

By focusing on CI/CD and MLOps, monitoring and observability, and cost optimization, the Operations layer bridges the gap between development and production, allowing organizations to maintain high-performance AI systems while adapting to changing requirements and managing resources effectively. A well-designed Operations layer is essential for scaling AI solutions, ensuring their reliability, and maximizing the return on investment in GenAI technologies. At the heart of this layer lies the CI/CD pipeline, which streamlines the process of integrating new code and deploying updated models seamlessly. Let's look at this in a little more detail.

CI/CD and MLOps

The adoption of DevOps principles and the implementation of CI/CD pipelines are crucial for streamlining the development, testing, and deployment of GenAI systems. This approach ensures that changes to the AI models, supporting infrastructure, and application code are integrated, tested, and deployed efficiently and reliably. By leveraging tools like Cloud Build, Artifact Registry, and Cloud Deploy, organizations can automate the building, testing, and deployment processes, significantly reducing manual errors and accelerating the delivery of new features and improvements. To maximize the efficiency and reliability of GenAI systems, several key practices in CI/CD and MLOps should be implemented:

- **Robust CI/CD pipeline**: GenAI systems should include automated testing frameworks that encompass unit tests, integration tests, and end-to-end tests. Unit tests focus on individual components of the system, such as specific functions or modules. Integration tests verify that different parts of the system work together correctly, while end-to-end tests simulate real-world usage scenarios to ensure the entire system functions as expected. These comprehensive testing strategies are essential for maintaining the reliability and correctness of GenAI systems, especially given the complex and often unpredictable nature of AI model behavior.
- **Model deployments**: To minimize downtime and reduce the risk of introducing breaking changes, organizations should consider implementing advanced deployment strategies such as canary deployments and blue-green deployments. Canary deployments involve releasing new versions to a small subset of users or servers before rolling out to the entire system, allowing for real-world testing and easy rollback if issues are detected. Blue-green deployments maintain two identical production environments, switching between them for releases, which enables instant rollback and zero-downtime updates. Organizations should also consider implementing feature flags, allowing for fine-grained control over the rollout of new features or model versions.

- **Version control**: Companies should implement version controls for all aspects of the GenAI system, including model versions, training data, hyperparameters, and application code. This enables traceability and reproducibility, which are essential for debugging, auditing, and compliance purposes.
- **Model monitoring and retraining pipelines**: These pipelines should automatically track model performance metrics, detect drift in data distributions or model accuracy, and trigger retraining processes when necessary. This ensures that the AI models remain accurate and relevant over time, adapting to changing data patterns and user behaviors.
- **Data versioning and lineage tracking**: By maintaining a clear record of the data used to train each model version, organizations can ensure reproducibility and facilitate the debugging of model behavior. This is particularly crucial in regulated industries where model decisions may need to be audited or explained.

Lastly, organizations should focus on creating a culture of collaboration between data scientists, ML engineers, and operations teams. This involves establishing clear communication channels, shared responsibility for the entire ML lifecycle, and continuous knowledge sharing. Regular post-mortem analyses of incidents and successful deployments alike can help teams identify areas for improvement and refine their MLOps practices over time.

By implementing these comprehensive CI/CD and MLOps practices, organizations can significantly enhance the reliability, efficiency, and effectiveness of their GenAI systems. This approach not only accelerates the development and deployment cycle but also ensures that AI models remain accurate, secure, and aligned with business objectives in the face of evolving data and user needs.

Monitoring and observability

In the rapidly evolving landscape of GenAI systems, maintaining a clear view of your model's performance and behavior is crucial. Monitoring and observability give you a view of your AI operations, providing critical insights into system health, performance metrics, and potential issues. This section dives into the key components that ensure your GenAI models operate at peak efficiency while allowing for the quick identification and resolution of problems. We'll explore evaluation and monitoring techniques to track model performance, alerting systems to promptly notify of anomalies, distributed tracing to understand complex system interactions, and comprehensive logging practices to maintain detailed records of system behavior. Together, these elements create a robust framework for maintaining oversight of your GenAI systems, enabling proactive management and continuous improvement.

Evaluation and monitoring

Ensuring the reliable and efficient operation of applications integrated with GenAI capabilities is paramount. Comprehensive monitoring and observability solutions play a crucial role in achieving this goal, providing valuable insights into the performance, health, and reliability of these systems. By leveraging Google Cloud's powerful monitoring and observability tools, such as Cloud Monitoring, Cloud Logging, and Cloud Operations, organizations can gain real-time visibility into the inner workings of their GenAI systems.

Here's an expanded narrative on establishing a mechanism to monitor results and detect changes introduced by model updates, with details on instrumentation, automated batch processing, and leveraging LLMs or other techniques.

One essential aspect of monitoring GenAI systems is the establishment of a robust mechanism to monitor the results and detect changes introduced by model updates. This proactive approach can help organizations stay ahead of potential issues and ensure the continued reliability and accuracy of their GenAI systems.

At the core of this monitoring mechanism lies the concept of "golden prompts" – carefully crafted prompts that represent typical use cases or scenarios for the GenAI system. These prompts should be designed in collaboration with subject-matter experts and stakeholders, ensuring that they cover a diverse range of contexts, complexities, and expected behaviors.

To operationalize this monitoring process, organizations can leverage automated batch-processing techniques. By setting up scheduled jobs or workflows, these golden prompts can be periodically and systematically submitted to the GenAI models, with the resulting outputs captured and analyzed for any deviations or anomalies.

This analysis can be further augmented by leveraging the power of rater LLMs or other techniques for automated evaluation and scoring. Rater LLMs can be fine-tuned to assess the quality, coherence, and correctness of the responses generated by the GenAI models, providing a quantitative measure of performance and flagging any significant deviations from expected outputs.

Alternatively, organizations can employ other evaluation techniques, such as leveraging human raters or crowdsourcing platforms to manually assess the quality of the GenAI model's outputs. While more resource-intensive, this approach can provide valuable insights and a human perspective on the model's performance.

An interesting approach is the one implemented by the **Chatbot Arena** platform (see the paper at `https://arxiv.org/html/2403.04132v1`), which introduces a novel approach to evaluating LLMs based on human preferences. Unlike traditional benchmarks that rely on static datasets and ground-truth evaluations, Chatbot Arena leverages a crowdsourced, pairwise comparison methodology. This approach aims to capture the nuanced and diverse aspects of LLMs by directly incorporating user preferences in real-world, open-ended tasks.

At the core of Chatbot Arena is an interactive website where users can submit questions and receive responses from two anonymous LLMs. After reviewing the responses, users cast a vote for the model that provided the preferred answer. This voting process is anonymous and randomized, ensuring an unbiased evaluation environment. By collecting these pairwise comparisons from a diverse user base, Chatbot Arena can gather a rich dataset of fresh user prompts and human preferences, accurately reflecting real-world LLM applications.

To reliably rank the LLMs based on the crowdsourced data, Chatbot Arena employs a suite of powerful statistical techniques. These include the Bradley-Terry model and the E-values proposed by Vovk and Wang, enabling the platform to estimate model rankings as reliably and sample-efficiently as possible. Additionally, the platform incorporates efficient sampling algorithms specifically designed to select model pairs in a way that accelerates the convergence of rankings while maintaining statistical validity.

The monitoring mechanism should also incorporate comprehensive logging and auditing capabilities, capturing not only the prompts, responses, and evaluation scores but also metadata such as model versions, input data sources, and any other relevant contextual information. This data can be invaluable for root cause analysis, debugging, and maintaining a comprehensive audit trail of the GenAI system's performance over time.

Furthermore, organizations can integrate this monitoring mechanism with their existing alerting and notification systems, ensuring that any significant deviations or anomalies are promptly flagged and communicated to the appropriate teams for investigation and remediation.

By establishing a robust monitoring mechanism that leverages golden prompts, automated batch processing, and advanced evaluation techniques like rater LLMs or human raters, organizations can stay ahead of potential issues introduced by model updates. This proactive approach not only enhances the reliability and accuracy of GenAI systems but also instills confidence in stakeholders and end-users, as they can be assured that the system's performance is continuously monitored and any deviations are promptly addressed.

Alerting

Alerting mechanisms are indispensable in this context, enabling relevant teams to be notified of any issues or anomalies that may arise. These alerts can be configured based on predefined thresholds or conditions, such as sudden spikes in error rates, latency, or resource utilization. Notifications can be delivered via various channels, including email, messaging platforms, or dedicated incident management tools, ensuring that the appropriate teams are promptly informed and can take action to mitigate the issue.

Distributed tracing

Distributed tracing is another powerful technique that can aid in monitoring and troubleshooting GenAI systems. By instrumenting applications with telemetry data, organizations can trace the path of a request as it flows through different components of the system. This end-to-end visibility is invaluable when diagnosing performance issues, identifying bottlenecks, or troubleshooting errors in complex, distributed systems. In the context of GenAI, distributed tracing becomes particularly crucial due to the often complex and interconnected nature of AI pipelines. It allows teams to:

- **Visualize request flow**: Tracing provides a clear picture of how requests propagate through various services, APIs, and microservices involved in AI inference or training processes.
- **Identify latency issues**: By breaking down the time spent in each component, tracing helps pinpoint where delays occur, whether in data pre-processing, model inference, or post-processing steps.
- **Detect anomalies**: Unusual patterns in trace data can highlight potential issues before they escalate into major problems, enabling proactive system management.
- **Optimize resource allocation**: Understanding the resource consumption of different components helps in fine-tuning resource allocation and improving overall system efficiency.
- **Debug complex scenarios**: In scenarios involving multiple AI models or complex decision trees, tracing helps understand the exact path taken by a request and the decisions made at each step.
- **Ensure data lineage**: Tracing can track the flow of data through the system, ensuring compliance with data governance policies and aiding in audits.

By leveraging distributed tracing, organizations can gain a deeper understanding of their GenAI systems' behavior, leading to more robust, efficient, and reliable AI-powered applications.

Logging

Application Performance Management (**APM**) tools, such as Cloud Operations and Cloud Trace, can provide detailed insights into application metrics. They can also request traces and logs, allowing for the quick identification and resolution of performance issues, optimized resource utilization, and a seamless user experience.

Comprehensive logging practices are also essential for capturing and analyzing relevant information from GenAI systems. By configuring applications to log important events, errors, and diagnostic information, and leveraging log management tools like Cloud Logging, organizations can centralize and analyze these logs, identify patterns, track down issues, and gain insights into the behavior of their GenAI systems.

Regular review and analysis of the monitoring data collected from GenAI systems can reveal trends, patterns, and potential areas for optimization or improvement. By leveraging this information, organizations can fine-tune their systems, enhance performance, and deliver a better user experience.

Implementing anomaly detection mechanisms can also be beneficial in automatically identifying and flagging unexpected or anomalous behavior in GenAI systems. These mechanisms can leverage ML techniques to analyze historical data, establish baselines, and detect deviations from normal patterns, enabling organizations to take appropriate actions.

Finally, establishing robust processes for incident management and response is crucial to ensure that any issues or incidents are addressed promptly and effectively. Defining clear roles and responsibilities, communication channels, and escalation procedures can facilitate a coordinated and efficient response to any incidents that may arise.

Cost optimization

In the realm of GenAI systems, balancing performance with cost-effectiveness is crucial for sustainable operations. Implementing robust cost optimization strategies is essential to manage the operational expenses associated with running these sophisticated AI systems on cloud platforms like Google Cloud.

Organizations should leverage the various cost-saving mechanisms offered by cloud providers. Committed-use discounts can significantly reduce costs for predictable workloads by committing to using a certain amount of resources for a specified term. These commits can be at the infrastructure layer or through managed services volume discounts. Additionally, exploring options like preemptible VMs for fault-tolerant workloads or spot VMs for flexible, interruptible tasks can lead to substantial savings.

Implementing comprehensive cost monitoring and attribution mechanisms is crucial for gaining visibility into cloud spending. These tools allow organizations to track expenses across different projects, teams, and services, helping identify areas of high expenditure and opportunities for optimization. Cloud cost management platforms can provide detailed breakdowns of spending, forecasting capabilities, and alerts for unusual spikes in usage or costs. By attributing costs to specific features or models, teams can make informed decisions about resource allocation and prioritize optimization efforts where they'll have the most impact.

Autoscaling and auto-shutdown mechanisms play a vital role in optimizing resource utilization and reducing costs during periods of low traffic. By automatically adjusting the number of compute instances based on demand, autoscaling ensures that the system can handle peak loads without overprovisioning during quieter periods. Implementing auto-shutdown for development and testing environments during non-working hours can lead to significant savings without impacting productivity.

Optimizing data storage and transfer costs is another crucial aspect of managing GenAI system expenses. This may involve implementing tiered storage solutions, using caching effectively, and optimizing data transfer patterns to minimize egress charges. For large-scale AI training jobs, consider using managed services that automatically optimize for cost-efficiency, such as Google Cloud's Vertex AI, which can dynamically adjust resource allocation based on job requirements.

Finally, consider the long-term cost implications of architectural decisions. While serverless architectures might seem more expensive on a per-request basis, they can often lead to lower total costs by eliminating the need for constant infrastructure management and allowing for true pay-per-use pricing. Similarly, investing in model optimization techniques like quantization or distillation can reduce inference costs in the long run, even if they require upfront development effort.

By addressing these key considerations and implementing best practices across all these layers (Data, Training, Inference, and Operations), you can successfully operationalize your GenAI integration patterns, ensuring reliability, scalability, and maintainability in production environments. Additionally, establishing robust data management, model governance, and security practices will help you build trust and comply with relevant regulations and industry standards.

Summary

In this chapter, you've explored a comprehensive framework for operationalizing GenAI integration patterns. You've learned about a four-layer approach that addresses the complexities of deploying and maintaining production-grade GenAI applications, encompassing the Data, Training, Inference, and Operations layers.

We proposed a holistic strategy that emphasizes the importance of data quality, security, and governance in the Data layer, while also addressing regulatory compliance and ethical considerations. The Training layer introduced you to various model adaptation techniques, including few-shot learning, fine-tuning, and full training, along with crucial aspects of model governance, performance monitoring, and XAI.

You've learned that the Inference layer focuses on scalability, performance optimization, and secure deployment strategies, including edge and distributed inference capabilities. The section on the Operations layer highlighted the significance of implementing robust CI/CD pipelines, MLOps best practices, and comprehensive monitoring and observability systems for GenAI applications.

By the end of this chapter, you've gained valuable insights into the intricacies of operationalizing GenAI systems. You've learned how to balance performance, cost-effectiveness, and ethical considerations while ensuring scalability and reliability. The chapter has equipped you with strategies to implement golden prompt evaluations, distributed tracing, and cost optimization techniques. These skills will enable you to enhance your organization's ability to deploy and maintain sophisticated GenAI applications, effectively leveraging cloud infrastructure and best practices in AI operations.

In the next chapter, we will discuss responsible AI and it's implications when integrating GenAI into your applications.

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

`https://packt.link/genpat`

10

Embedding Responsible AI into Your GenAI Applications

In the previous chapters, we explored various integration patterns and operational considerations for leveraging **Generative AI** (**GenAI**) models like Google Gemini on Vertex AI. As we implement these powerful technologies, it's crucial to address the ethical implications and responsibilities that come with building and deploying AI models that will be added to your applications. This chapter will focus on best practices for responsible AI, ensuring that our GenAI applications are fair, interpretable, private, and safe.

In this chapter, we'll cover:

- Introduction to responsible AI
- Fairness in GenAI applications
- Interpretability and explainability
- Privacy and data protection
- Safety and security in GenAI systems
- Google's approach to responsible AI
- Anthropic's approach to responsible AI

Introduction to responsible AI

Responsible AI is an approach to developing and deploying AI systems that prioritizes ethical considerations, transparency, and accountability.

As GenAI models and applications such as Google's Gemini, OpenAI's GPT, and Anthropic's Claude become increasingly powerful and widely used, it's essential to ensure that these systems are designed and implemented in ways that benefit society while minimizing potential harm. Let's explore, at a high level, the key aspects of implementing responsible AI in your systems. We will also talk about how over-indexing on the following topics can have a negative effect on innovation:

- **Fairness**: Achieving fairness in AI systems is a crucial goal that requires thoughtful design and implementation throughout the entire AI lifecycle. Several key factors contribute to making AI systems fair. First and foremost is the quality and diversity of the training data. Ensuring that the dataset represents a wide range of demographics, experiences, and perspectives helps to minimize bias and promote equitable outcomes. This involves not just collecting diverse data, but also carefully curating and balancing it to avoid over- or under-representation of certain groups, ensuring AI systems don't perpetuate or amplify biases. While aiming for fairness is crucial, it's complex to implement. Different definitions of fairness can conflict with each other. For example, achieving demographic parity might not always align with equal opportunity. Overcompensating for historical biases could potentially create new forms of discrimination. For instance, an AI hiring system can be so focused on demographic parity that it overlooks genuine qualifications, potentially leading to more capable candidates being overlooked.
- **Interpretability**: This is about understanding how AI systems make decisions. Highly interpretable models might sacrifice performance for explainability. This trade-off could be particularly problematic in fields like medical diagnosis or financial forecasting, where complex patterns might be crucial for accurate predictions.
- **Privacy**: This involves protecting user data and respecting privacy rights. Strict privacy measures can hinder beneficial data sharing and collaborative research. They might also increase costs for businesses, potentially stifling innovation in smaller companies that can't afford robust privacy protection measures. For example, overly strict data protection might prevent the creation of the large, diverse datasets needed for developing treatments for rare diseases.
- **Safety**: This is where you ensure that AI systems behave as intended and don't cause harm. Rigorous safety testing can significantly slow down the development and deployment of AI systems. This could delay the introduction of potentially life-saving technologies. Additionally, overly cautious approaches might lead to missed opportunities for learning from controlled, real-world testing.

- **Accountability**: This is about taking responsibility for AI system outcomes. Clear lines of accountability are necessary; overly punitive measures could discourage innovation. It might also lead to a culture of blame rather than learning and improvement when issues do arise.

There is no secret recipe for getting all these aspects correct. Companies need to find the balance and address these concerns in practice. Let's consider the following approaches to minimize concerns about over-indexing in rigid approaches:

- **Contextual application**: Recognize that the importance of each aspect may vary depending on the specific AI application and its potential impact. For example, interpretability might be more crucial in medical diagnosis AI than in a movie recommendation system.
- **Stakeholder engagement**: Involve diverse stakeholders in the development and deployment process. This can help identify potential issues and trade-offs early on. Adopt a risk-based approach where the level of scrutiny and safeguards is proportional to the potential harm or impact of the AI system.
- **Transparency and monitoring**: Be open about the limitations and trade-offs of your AI system. This can help manage expectations and build trust. Regularly assess the performance and impact of AI systems in real-world use and be prepared to make adjustments. Implement these principles progressively, starting with minimum viable standards and improving over time based on real-world feedback and outcomes.
- **Regulatory collaboration**: Work with policymakers to develop regulations that encourage responsible innovation rather than stifling it. Invest in educating both developers and users about these principles and their implications.

By taking a nuanced, context-specific approach and continuously reassessing the balance between these principles, we can work towards maximizing the benefits of AI while minimizing potential harm.

Fairness in GenAI applications

Fairness in AI systems is a critical concern as these technologies become increasingly integrated into decision-making processes across various sectors of society. Ensuring that AI systems do not perpetuate or amplify existing biases is essential for building trust, promoting equality, and maximizing the benefits of AI for all. However, achieving fairness in AI is a complex and multifaceted challenge that requires ongoing effort and vigilance.

It involves considerations at every stage of the AI lifecycle, from data collection and model development to deployment and monitoring. The following points outline key strategies and considerations for promoting fairness in AI systems, with a focus on practical approaches and real-world examples. By implementing these practices, organizations can work towards creating AI systems that are more equitable, transparent, and beneficial to society as a whole:

- **Diverse and representative data**: Ensuring diverse and representative data is fundamental to creating fair AI systems. This involves not just including data from various demographic groups but also considering intersectionality and less visible forms of diversity. For example, in developing a speech recognition system, ensure the training data includes speakers with various accents and dialects from a range of age groups and genders. For instance, Apple faced criticism when early versions of Siri struggled with Scottish accents, highlighting the importance of linguistic diversity in training data.
- **Bias detection and mitigation**: This involves both proactive and reactive measures. *Proactively*, use statistical techniques to identify potential biases in your data and model outputs. *Reactively*, implement feedback mechanisms to catch and correct biases that emerge in real-world use. For example, consider an AI recruitment tool showing bias against specific demographics or genders in the tech industry. This could be the result of the system being trained on historical hiring data that reflected past gender or demographic biases in the tech industry. This case highlights the importance of scrutinizing training data and model outputs for unfair patterns.
- **Regular audits**: Fairness audits should be comprehensive, examining not just the model's outputs but also its impact in the real world. This may involve both quantitative metrics and qualitative assessments. For example, credit-scoring AI might pass initial fairness tests, but regular audits could reveal that it's inadvertently disadvantaged certain groups over time due to changing economic conditions. Regular audits would catch this drift and allow for timely corrections.
- **Inclusive design**: Inclusive design goes beyond just consulting diverse stakeholders. It involves empowering them to actively shape the development process and giving their input real weight in decision-making. For example, when Microsoft developed the Xbox Adaptive Controller for gamers with limited mobility, they involved gamers with disabilities throughout the design process. This inclusive approach led to innovations that might have been overlooked by designers without lived experience of disability.

- **Contextual fairness**: Fairness can mean different things in different contexts. What's fair in one situation might not be in another. AI systems need to be flexible enough to adapt to these nuances. For example, an AI system for college admissions might need to balance multiple fairness criteria – equal opportunity, demographic parity, and individual merit. The appropriate balance might differ between a public university with a mandate for diverse representation and a specialized technical institute.
- **Transparency and explainability**: For AI systems to be truly fair, users should understand how decisions are made. This involves both technical explainability and clear communication with non-technical stakeholders. In healthcare, an AI system making treatment recommendations should be able to explain its reasoning in terms that both doctors and patients can understand. This allows informed consent and gives opportunities for patients to provide additional context that the AI might have missed.
- **Ongoing monitoring and adaptation**: Fairness isn't a one-time achievement but an ongoing process. Societal norms and understanding of fairness evolve, and AI systems need to adapt accordingly. For example, an AI content moderation system for a social media platform might need to continuously update its understanding of hate speech and discriminatory language as social norms evolve and new forms of coded language emerge.
- **Legal and ethical compliance**: Ensure that fairness measures comply with relevant laws (like anti-discrimination legislation) and align with ethical standards. This may vary by jurisdiction and application area. In the EU, the GDPR and the proposed AI Act set specific requirements for fairness and non-discrimination in AI systems. Companies operating in or selling to the EU market need to ensure their AI systems comply with these regulations.

These expanded points highlight the complexity and nuance involved in ensuring fairness in AI systems. It's an ongoing challenge that requires vigilance, adaptability, and a commitment to ethical principles.

Interpretability and explainability

Interpretability and explainability in AI systems, particularly in **large language models (LLMs)** and GenAI, are crucial for fostering trust, enabling effective oversight, and ensuring responsible deployment. As these systems become more complex and their decision-making processes more opaque, the need for methods to understand and explain their outputs grows increasingly important. Interpretability allows stakeholders to peek inside the "black box" of AI, while explainability focuses on communicating how decisions are made in a way that humans can understand.

The following points outline key strategies for enhancing interpretability and explainability in AI systems, with a focus on practical approaches and real-world examples. By implementing these practices, organizations can create more transparent AI systems, facilitating better decision-making, regulatory compliance, and user trust.

- **Model cards**: Model cards provide a standardized way to document AI models, including their performance characteristics, intended uses, and limitations. They serve as a crucial tool for transparency and responsible AI deployment. For example, Google's BERT language model comes with a detailed model card that outlines its training data, evaluation results across different tasks and demographics, and ethical considerations. This allows users to make informed decisions about whether BERT is appropriate for their specific use case and helps them understand potential biases or limitations.
- **Explainable AI (XAI) techniques**: XAI techniques aim to make the decision-making process of AI models more understandable to humans. These methods can provide insights into which features are most important for a particular decision or prediction. For example, in a medical diagnosis AI system, **SHapley Additive exPlanations** (**SHAP**) values could be used to show which symptoms or test results contributed most to a particular diagnosis. This allows doctors to understand the AI's reasoning and compare it with their own clinical judgment.
- **User-friendly explanations**: Technical explanations of AI decisions are often not useful for end-users. Developing clear, non-technical explanations tailored to the user's level of expertise is crucial for practical explainability. Credit-scoring AI might explain its decision to deny a loan not just with a numerical score, but with a simple explanation like "Your application was declined primarily due to your high debt-to-income ratio and recent missed payments on your credit card." This gives the user actionable information without requiring them to understand the underlying AI model.
- **Traceability**: Maintaining detailed logs of an AI system's inputs, outputs, and key decision points allows for auditing and helps in understanding the system's behavior over time. This is particularly important for regulatory compliance and debugging. In an autonomous vehicle system, maintaining a detailed log of sensor inputs, decision points, and actions taken would be crucial. If an accident occurs, this log could be analyzed to understand why the AI made certain decisions and how similar incidents could be prevented in the future.
- **Interpretable model architectures**: While not using inherently more interpretable model architectures can be a powerful approach, this might involve using simpler models where possible or developing new architectures designed for interpretability.

In a financial fraud detection system, a decision tree model might be used instead of a more complex neural network for certain components. The decision tree's logic can be easily visualized and understood, allowing for clear explanations of why a transaction was flagged as potentially fraudulent.

- **Interactive explanations**: Providing users with interactive tools to explore model behavior can greatly enhance understanding. This allows users to ask "what if" questions and see how changes in inputs can affect outputs. For example, a recommendation system for an e-commerce platform could include an interactive feature allowing users to adjust the importance of different factors (for example, price, brand, ratings) and see in real time how this affects product recommendations. This helps users understand the system's logic and tailor it to their preferences.

By implementing these strategies, organizations can significantly enhance the interpretability and explainability of their AI systems, leading to more transparent, trustworthy, and effective AI applications.

Privacy and data protection

Privacy and data protection in AI systems, especially in the context of powerful GenAI models, is a critical concern that impacts user trust, legal compliance, and ethical use of this technology. As AI systems process increasingly large amounts of personal and potentially sensitive data, ensuring robust privacy safeguards becomes a make-or-break point for organizations. Effective privacy protection involves not only technical measures but also organizational policies and user empowerment. The following points outline key strategies for enhancing privacy and data protection in AI systems, with a focus on practical approaches and real-world examples. By implementing these practices, organizations can create AI systems that respect user privacy, comply with regulations, and maintain the trust of their users and stakeholders:

- **Data minimization**: This involves collecting and using only the data that is absolutely necessary for the AI system's intended function. This principle reduces privacy risks and aligns with many data protection regulations. For instance, a smart home AI assistant could be designed to process voice commands locally on the device whenever possible, rather than sending all audio data to cloud servers. This minimizes the amount of potentially sensitive data that leaves the user's control and increases the trust of users in their data not being at risk of a leak or used to further improve models.

- **Anonymization and encryption**: These techniques help protect individual privacy by removing or obscuring personally identifiable information in datasets used for training and inference. Properly implemented, they can allow useful data analysis while preserving privacy. In developing an AI system for analyzing hospital patient outcomes, researchers could use differential privacy techniques. For example, differential privacy adds carefully calibrated noise to the data or to the model's outputs, allowing accurate overall analysis while making it mathematically impossible to identify individual patients.
- **User control**: Empowering users with control over their data is not only a legal requirement in many jurisdictions but also builds trust. This includes providing clear, easily accessible options for data management. For example, a social media platform using AI for content recommendations could provide a detailed *Privacy Dashboard* where users can see what data is being collected, how it's being used, and easily opt out of specific data collection or processing activities.
- **Compliance**: Adherence to relevant data protection regulations is crucial. This often involves implementing privacy by design principles, conducting impact assessments, and maintaining detailed documentation of data processing activities. A multinational company developing an AI-powered HR tool would need to ensure compliance with GDPR for EU employees, CCPA for California residents, and other applicable local regulations. This might involve creating region-specific data handling processes and providing different privacy notices and controls based on user location.
- **Avoid logging PII and other sensitive information**: Logs are often overlooked as a potential source of privacy breaches. Implementing practices to avoid capturing personally identifiable information in logs is crucial for maintaining privacy. Even when logs don't contain PII, they can still contain sensitive information about system operations. Implementing strong security measures for log data is essential. A financial institution using AI for fraud detection could implement a secure, encrypted log storage system with strict access controls. Only authorized personnel would be able to access logs, and all access would be logged for auditing purposes.
- **Retention policies**: Establishing and enforcing data retention policies helps minimize privacy risks over time and often aligns with legal requirements for data minimization. An AI-powered fitness app could implement a policy to automatically delete user activity data after a certain period (for example, 6 months) unless the user explicitly opts in to keep it longer. This reduces the risk of old data being compromised while still allowing users to maintain long-term records if desired.

- **Privacy-preserving AI techniques**: Emerging techniques like federated learning and homomorphic encryption allow AI models to learn from data without directly accessing it, providing powerful new tools for privacy protection. A keyboard prediction AI on smartphones could use federated learning to improve its model. The model would be updated on individual devices using local data, and only the model updates (not the raw data) would be sent back to the central server, preserving user privacy.

By implementing these strategies, organizations can significantly enhance the privacy and data protection aspects of their AI systems, leading to more trustworthy and legally compliant AI applications that respect user privacy.

Safety and security in GenAI systems

Ensuring that GenAI systems operate safely, securely, and as intended is crucial for protecting users and preventing potential misuse of information generated and from training, as well as unintended consequences. This involves a multi-faceted approach that encompasses both proactive measures and reactive capabilities. The following points outline key strategies for enhancing safety and security in GenAI systems, with a focus on practical approaches and real-world examples:

- **Content filtering**: Implementing filters to prevent the generation of harmful or inappropriate content is a crucial safety measure. For instance, a GenAI-powered chatbot for a children's educational platform could use advanced content filtering algorithms to detect and block any attempts to generate age-inappropriate content, ensuring a safe learning environment. This might involve maintaining and regularly updating a comprehensive list of prohibited terms and topics, as well as using more sophisticated natural language processing techniques to identify potentially harmful content even when it's expressed in novel ways.
- **Rate limiting**: Applying reasonable limits on API calls is essential to prevent abuse and ensure fair usage. A GenAI service providing image generation capabilities could implement tiered rate limiting, where free users are limited to a certain number of generations per day, while paid users have higher limits. This not only prevents potential denial-of-service attacks but also helps manage computational resources effectively. The system could also implement more nuanced rate limiting based on the complexity of the requests, allowing more frequent simple generations while limiting resource-intensive ones.
- **Adversarial testing**: Regularly testing the system with adversarial inputs is crucial for identifying and addressing vulnerabilities. These tests need to be added to the development cycle as well as the continuous monitoring of the system.

For a GenAI system designed to assist in medical diagnosis, researchers could create a suite of adversarial test cases designed to trick the system into making incorrect diagnoses. This might include subtle alterations to medical images or carefully crafted textual descriptions of symptoms. By continuously running and expanding these tests, developers can identify weak points in the system's decision-making process and implement targeted improvements to enhance robustness.

- **Monitoring and alerting**: Setting up systems to detect and respond to anomalous behavior is critical for maintaining ongoing safety and security. A financial institution using GenAI for fraud detection could implement a real-time monitoring system that tracks unusual patterns in the AI's behavior. For example, if the system suddenly starts flagging an unusually high number of transactions as fraudulent, or if its confidence scores show unexpected fluctuations, automated alerts could be triggered for human review. This allows quick intervention in case of potential system malfunctions or targeted attacks.
- **Improved robustness**: Enhanced stability and consistency in outputs, as demonstrated by models like Gemini 1.5, is crucial for safety. A GenAI system used for automated customer service could leverage these improvements to provide more reliable and consistent responses across a wide range of customer inquiries. This reduces the risk of the system providing contradictory or nonsensical information, which could lead to customer confusion or dissatisfaction. Regular evaluations comparing the system's outputs over time and across different inputs can help verify and maintain this robustness.
- **Content safety**: Built-in mechanisms to avoid generating harmful or explicit content are essential safeguards. A GenAI-powered creative writing assistant could incorporate multiple layers of content safety checks. This might include analyzing generated text for potentially offensive language, checking for age-appropriate content based on the user's profile, and providing warnings or alternatives when the system detects that it might be veering into sensitive territory. These mechanisms should be regularly updated to reflect evolving societal norms and newly identified risks.
- **Multimodal safety**: Extending safety considerations across text, image, and other modalities is increasingly important as GenAI systems become more versatile. A multimodal GenAI system used in social media content moderation could apply safety checks across text, images, and videos simultaneously. For instance, it could analyze both the text of a post and any accompanying images to detect potential policy violations, ensuring that harmful content isn't slipping through by exploiting gaps between different modalities. This requires developing and maintaining comprehensive safety models that can understand context across different types of data.

By implementing these strategies, organizations can significantly enhance the safety and security of their GenAI systems, creating more reliable, trustworthy, and beneficial AI applications. Regular reviews and updates of these measures are essential to keep pace with the rapidly evolving capabilities and potential risks associated with GenAI technologies.

Google's approach to responsible AI

Google's approach to responsible AI is a comprehensive framework that serves as a model for organizations implementing GenAI systems. At its core, the approach prioritizes:

- *Human-Centered Design*, emphasizing the importance of user needs and societal impact in AI development. This ensures that AI systems are created with a deep understanding of their real-world implications and potential benefits for users across diverse backgrounds.
- *Fairness* is another crucial aspect of Google's framework. The company has developed sophisticated tools and practices to detect and mitigate unfair bias in AI systems. This commitment to fairness involves rigorous testing, diverse dataset creation, and ongoing monitoring to ensure AI applications remain equitable as they evolve and are deployed in various contexts.
- *Interpretability* is a key focus, with Google striving to create explainable AI systems that promote transparency. This involves developing methods to make AI decision-making processes more understandable to both developers and end-users, fostering trust and accountability in AI applications.
- *Privacy* is a fundamental concern addressed through advanced techniques such as federated learning and differential privacy. These approaches allow for the development of powerful AI models while protecting individual user data, striking a balance between innovation and privacy protection.
- *Safety and Security* are paramount in Google's responsible AI approach. The company conducts rigorous testing and focuses on developing robust AI systems that can withstand potential misuse or manipulation. This proactive stance on security helps ensure that AI technologies are not only powerful but also trustworthy and resilient.

Google's approach emphasizes the importance of cross-functional collaboration, bringing together experts from various fields to address the multifaceted challenges of AI development. This is complemented by a strong commitment to ongoing research, pushing the boundaries of AI capabilities while simultaneously exploring its ethical implications.

Google's Secure AI Framework (SAIF)

This is a conceptual framework designed to manage risks associated with rapidly advancing AI technologies. It consists of six core elements:

- **Expand strong security foundations**: Adapt existing cybersecurity measures to protect AI systems and develop expertise to keep pace with AI advancements.
- **Extend detection and response**: Incorporate AI-related threats into existing cybersecurity practices, including monitoring AI system inputs and outputs for anomalies.
- **Automate defenses**: Utilize AI innovations to improve the scale and speed of responses to security incidents, countering potential AI-enhanced threats.
- **Harmonize platform-level controls**: Ensure consistent security across different AI platforms and tools within an organization, leveraging secure-by-default protections.
- **Adapt controls**: Continuously test and evolve detection and protection mechanisms through techniques like reinforcement learning and regular Red Team exercises.
- **Contextualize AI system risks**: Conduct comprehensive risk assessments for AI deployments, considering end-to-end business risks and implementing automated checks for AI performance validation.

This framework aims to help organizations evolve their risk management strategies alongside AI advancements, ensuring more secure and responsible AI implementation.

Building upon the core elements of Google's **Secure AI Framework** (**SAIF**), the approach to implementation (`https://services.google.com/fh/files/blogs/google_secure_ai_framework_approach.pdf`) provides a practical guide for organizations looking to integrate AI securely into their operations. The approach breaks down the process into four key steps and elaborates on how to apply the six core elements of SAIF effectively:

1. **Understand the use**: Clearly define the specific AI use case and its context within the organization.
2. **Assemble the team**: Create a cross-functional team including stakeholders from various departments such as security, legal, data science, and ethics.
3. **Level set with an AI primer**: Ensure all team members have a basic understanding of AI concepts and terminology.
4. **Apply the six core elements of SAIF**:

    a. Expand strong security foundations:

   i. Review existing security controls and their applicability to AI systems.
   ii. Evaluate traditional controls against AI-specific threats.
   iii. Prepare for supply chain management and data governance.

b. Extend detection and response:

   i. Develop an understanding of AI-specific threats.
   ii. Prepare to respond to attacks against AI and issues raised by AI output.
   iii. Adjust abuse policies and incident response processes for AI-specific incidents.

c. Automate defenses:

   i. Identify AI security capabilities for protecting systems and data pipelines.
   ii. Use AI defenses to counter AI threats while keeping humans in the loop.
   iii. Leverage AI to automate time-consuming tasks and speed up defensive mechanisms.

d. Harmonize platform-level controls:

   i. Review AI usage and the lifecycle of AI-based applications.
   ii. Standardize tooling and frameworks to prevent control fragmentation.

e. Adapt controls:

   i. Conduct Red Team exercises for AI-powered products.
   ii. Stay informed about novel AI attacks.
   iii. Apply machine learning to improve detection accuracy and speed.
   iv. Create feedback loops for continuous improvement.

f. Contextualize AI system risks:

   i. Establish a model risk management framework.
   ii. Build an inventory of AI models and their risk profiles.
   iii. Implement policies and controls throughout the ML model lifecycle.
   iv. Perform comprehensive risk assessments for AI use.
   v. Consider shared responsibility in AI security.
   vi. Match AI use cases to organizational risk tolerances.

This approach provides a structured way for organizations to implement SAIF, ensuring that AI integration is secure, responsible, and aligned with business objectives.

Google's Red Teaming approach

This approach to AI systems is a comprehensive strategy to identify and mitigate potential security risks in AI deployments. The company has established a dedicated AI Red Team that combines traditional red teaming expertise with specialized AI subject matter knowledge. This team simulates threat actors targeting AI deployments, with the goal of assessing the impact of simulated attacks on users and products, analyzing the resilience of new AI detection and prevention capabilities, improving detection capabilities for early attack recognition, and raising awareness among stakeholders about key risks and necessary security controls. For further reading, please refer to `https://services.google.com/fh/files/blogs/google_ai_red_team_digital_final.pdf`.

The AI Red Team works closely with Google's threat intelligence teams like Mandiant and the **Threat Analysis Group** (**TAG**) to ensure their simulations reflect realistic adversary activities. This collaboration allows the team to prioritize different exercises and shape engagements that closely resemble real-world threats.

Google's approach to red teaming for AI systems focuses on several common types of attacks:

- *Prompt attacks* are a significant concern, especially for LLMs powering GenAI products. These attacks involve crafting inputs that manipulate the model's behavior in unintended ways. For example, an attacker might inject hidden instructions into an email to bypass AI-based phishing detection systems.
- *Training data extraction attacks* aim to reconstruct verbatim training examples, potentially exposing sensitive information like **personally identifiable information** (**PII**) or passwords. These attacks are particularly dangerous for personalized models or those trained on data containing PII.
- *Backdooring the model* is another critical threat where attackers attempt to covertly change the model's behavior to produce incorrect outputs with specific "trigger" words or features. This can be achieved through direct manipulation of the model's weights, fine-tuning for adversarial purposes, or modifying the file representation of the model.
- *Adversarial examples* involve inputs that result in unexpected outputs from the model. For instance, an image that appears as a dog to humans but is classified as a cat by the model. The impact of these attacks can vary widely depending on the AI system's use case.

- *Data poisoning attacks* involve manipulating the training data to influence the model's output according to the attacker's preferences. This highlights the importance of securing the data supply chain in AI development.
- *Exfiltration attacks* focus on stealing the model itself, which often includes sensitive intellectual property. These can range from simple file copying to more complex attacks involving repeated querying of the model to determine its capabilities and recreate it.

Google emphasizes that while these AI-specific attacks are crucial to address, they should be considered in addition to, not instead of, traditional security threats. The company advocates for a comprehensive approach that combines AI-specific red teaming with traditional security practices like penetration testing, vulnerability management, and the secure development lifecycle.

By sharing their approach and lessons learned, Google aims to establish clear industry security standards for AI and help advance the field of AI security. Their experience underscores the importance of cross-functional collaboration, continuous learning, and the integration of AI security with broader cybersecurity efforts to create more robust and secure AI deployments.

Anthropic's approach to responsible AI

Anthropic's approach to responsible AI development is comprehensive and multifaceted, integrating ethical considerations, safety measures, and empirical research throughout their work. Key aspects of their approach include:

- **Prioritizing Safety Research**: Anthropic views AI safety research as urgently important and worthy of broad support. They employ a portfolio approach, preparing for a range of scenarios, from optimistic to pessimistic, regarding the difficulty of creating safe AI systems.
- **Empirical Focus**: The company emphasizes empirically driven safety research, believing that close contact with frontier AI systems is crucial for identifying and addressing potential risks before they become critical issues.
- **Balancing Progress and Caution**: Anthropic carefully navigates the trade-off between advancing necessary safety research and potentially accelerating the deployment of dangerous technologies. They aim to integrate safety research into real systems quickly while maintaining responsible development practices.
- **Transparency and Collaboration**: While not publishing capabilities research, Anthropic shares safety-oriented research to contribute to the broader AI community's understanding of these issues.

- **Ethical Deployment**: The company is thoughtful about demonstrating frontier capabilities, prioritizing safety research over public deployments when appropriate.
- **Commitment to Safety Standards**: Anthropic plans to make externally verifiable commitments to only develop advanced models if specific safety standards are met, allowing independent evaluation of their models' capabilities and safety.
- **Societal Impact Assessment**: They conduct research to evaluate the potential societal impacts of their AI systems, informing both internal policies and broader AI governance discussions.
- **Responsible Scaling**: Anthropic recognizes the risks of rapidly scaling AI capabilities and aims to develop techniques like scalable oversight to ensure powerful systems remain aligned with human values.
- **Interdisciplinary Approach**: Their strategy spans technical research, policy considerations, and ethical deliberations, reflecting the complex nature of AI safety challenges.
- **Adaptability**: Anthropic maintains flexibility in their approach, ready to adjust strategies as they learn more about AI development and its associated risks.

By integrating these principles, Anthropic aims to contribute to the development of AI systems that are not only powerful but also safe, ethical, and beneficial to humanity. Their approach reflects a deep commitment to responsible innovation in the face of potentially transformative AI technologies. Anthropic's research areas are diverse and comprehensive, focusing on various aspects of AI safety and responsible development. Their approach spans from immediate practical concerns to long-term, speculative risks, demonstrating a commitment to addressing the multifaceted challenges of AI development.

At the core of Anthropic's research is *Mechanistic Interpretability*, which aims to "reverse engineer" neural networks into human-understandable algorithms. This ambitious project seeks to enable something akin to "code reviews" for AI models, potentially allowing thorough audits to identify unsafe aspects or provide strong safety guarantees. The team has made progress in extending this approach from vision models to small language models and has discovered mechanisms driving in-context learning.

Scalable Oversight is another crucial area of focus for Anthropic. This research aims to develop methods for AI systems to partially supervise themselves or assist humans in supervision. It's a response to the challenge of providing adequate high-quality feedback to steer AI behaviors as systems become increasingly complex.

The team explores various approaches, including extensions of Constitutional AI, variants of human-assisted supervision, AI-AI debate, and red teaming via multi-agent reinforcement learning.

Process-Oriented Learning represents a novel approach to AI training. Instead of rewarding AI systems for achieving specific outcomes, this method focuses on training systems to follow safe processes. The goal is to address concerns about AI safety by ensuring systems follow comprehensible, approved processes rather than pursuing goals through potentially harmful or inscrutable means.

Testing for *Dangerous Failure Modes* is a proactive approach to identifying potential risks in AI systems. This involves deliberately training problematic properties into small-scale models to isolate and study them before they become direct threats in more capable systems. A particular area of interest is studying how AI systems behave when they are "situationally aware" and how this impacts their behavior during training.

Lastly, Anthropic places significant emphasis on researching *Societal Impacts and Evaluations*. This involves building tools and measurements to assess the capabilities, limitations, and potential societal impact of AI systems. Studies in this area have included research on predictability and surprise in LLMs, methods for red teaming language models, and investigations into reducing bias and stereotyping in AI outputs. This research not only informs Anthropic's internal practices but also contributes to broader discussions on AI governance and policy.

These interconnected research areas collectively aim to address various aspects of AI safety and ethics. Anthropic's approach is notable for its breadth and depth, reflecting the complex nature of AI development challenges. The company maintains flexibility in its research focus, ready to adapt as new information and challenges emerge in the rapidly evolving field of AI, demonstrating a commitment to responsible innovation in the face of potentially transformative AI technologies.

Anthropic, a leading AI research company, has developed a comprehensive approach to addressing the challenges of AI safety and ethics. Their method, called **Constitutional AI**, represents a significant advancement in creating AI systems that are both powerful and aligned with human values, while reducing the need for extensive human oversight in the training process. Their approach is highlighted in the paper *Constitutional AI: Harmlessness from AI Feedback*, (`https://arxiv.org/pdf/2212.08073`) which presents Anthropic's approach to developing safe and responsible AI systems without relying on human feedback for harmlessness. The research introduces a method called **Constitutional AI** (**CAI**), which aims to create AI assistants that are both helpful and harmless while avoiding evasive responses.

The core of Anthropic's approach involves two main stages: a supervised learning phase and a reinforcement learning phase. In the supervised phase, the AI model is trained to critique and revise its own responses based on a set of principles or "constitution." This process helps to reduce harmful content in the AI's outputs. The reinforcement learning phase then further refines the model's behavior using AI-generated feedback, rather than human labels, to evaluate harmlessness.

Anthropic's motivation for developing this technique stems from several factors. They aim to scale supervision by leveraging AI to help oversee other AI systems, reduce the tension between helpfulness and harmlessness in AI assistants, increase transparency in AI decision-making, and decrease reliance on extensive human feedback. The researchers emphasize the importance of empirical, data-driven approaches to AI safety, as well as the need for flexibility in addressing various potential scenarios, from optimistic to pessimistic, regarding the difficulty of creating safe AI systems.

The paper presents experimental results showing that the Constitutional AI method can produce models that are both less harmful and more helpful than those trained with traditional **reinforcement learning from human feedback** (**RLHF**). Notably, the CAI models demonstrate the ability to engage with sensitive topics in a thoughtful, non-evasive manner while still maintaining safety.

Anthropic's approach also incorporates techniques such as chain-of-thought reasoning to improve the transparency and interpretability of AI decision-making. This allows better understanding of how the AI system arrives at its conclusions and behaviors. The researchers stress the importance of continuous adaptation and improvement in their safety techniques as AI capabilities advance.

The paper concludes by discussing potential future directions for this research, including applying constitutional methods to steer AI behavior in various ways, improving robustness against adversarial attacks, and further refining the balance between helpfulness and harmlessness. Anthropic acknowledges the dual-use potential of these techniques and emphasizes the need for responsible development and deployment of AI systems.

Overall, this research represents a significant step towards creating AI systems that can be both powerful and aligned with human values, while reducing the need for extensive human oversight in the training process.

Summary

When designing AI applications, developers can incorporate elements from both Google's and Anthropic's approaches to create more responsible, safer systems. This comprehensive strategy involves several key areas of focus:

Comprehensive safety and impact assessment is crucial for responsible AI development. This process should include scenario planning for both optimistic and pessimistic outcomes, allowing developers to prepare for a wide range of possibilities. Empirical testing of safety measures on small-scale systems is essential, as it provides valuable insights without the risks associated with large-scale deployment. Engaging diverse stakeholders helps identify potential issues early in the development process, ensuring a broad perspective on the AI system's potential impacts. Establishing clear safety thresholds before scaling or deploying more advanced capabilities helps maintain control over the AI's development and ensures that safety remains a priority throughout the process.

A transparent and adaptive development process is vital for building trust and maintaining the safety of AI systems. Implementing explainable AI techniques makes the system's decision-making process more interpretable, allowing both developers and users to understand how the AI arrives at its conclusions. Regular external audits of the system's performance, fairness, and safety provide objective assessments and help identify areas for improvement. Clear documentation of the model's capabilities, limitations, and intended use cases helps manage expectations and prevents misuse. Developing a process for rapidly integrating new safety research findings ensures that the AI system remains up to date with the latest advancements in AI safety.

Fairness and bias mitigation are critical aspects of responsible AI development. Ensuring AI systems don't perpetuate or amplify biases requires using diverse and representative training data, implementing proactive and reactive bias detection and mitigation measures, and conducting regular fairness audits across different contexts and user groups. Applying contextual fairness appropriate to the specific application helps ensure that the AI system's decisions are equitable in various scenarios.

Privacy and data protection are key to safe AI development. Implementing strong safeguards for user privacy involves applying data minimization principles, using anonymization, encryption, and privacy-preserving AI techniques, providing users with clear control over their data, and establishing and enforcing data retention policies. These measures help build user trust and ensure compliance with data protection regulations.

Safety and security measures are essential for ensuring AI systems behave as intended and resist potential attacks. This includes implementing content filtering and rate limiting to prevent misuse, conducting regular adversarial testing to identify vulnerabilities, setting up monitoring and alerting systems for anomalous behavior, and applying multimodal safety considerations for versatile AI systems. These measures help protect the AI system from external threats and prevent unintended harmful behaviors.

Responsible AI is not just an ethical imperative; it's a crucial component of building sustainable, trustworthy, and effective GenAI applications. By integrating fairness, interpretability, privacy, and safety considerations throughout the development lifecycle, we can harness the power of models like Google Gemini while mitigating potential risks and negative impacts.

As we've explored in this chapter, implementing responsible AI practices requires a multifaceted approach. This includes diverse and representative data, bias detection and mitigation, explainable AI techniques, robust privacy protections, and comprehensive safety measures. By learning from industry leaders like Google and Anthropic, and adapting their frameworks to our specific use cases, we can create GenAI applications that not only push the boundaries of what's possible but do so in a way that benefits society as a whole.

Remember that responsible AI is an ongoing process, not a one-time checkbox. As GenAI technologies continue to evolve rapidly, it's essential to stay informed about the latest developments in AI ethics, regularly reassess our applications, and be prepared to adapt our practices accordingly.

Throughout this book, we've embarked on a comprehensive journey through the world of GenAI, focusing on leveraging powerful models like Google Gemini on Vertex AI. We began by exploring the process of use case ideation and selection, learning how to identify opportunities where GenAI can provide significant value and transform business processes.

From there, we delved into the crucial phase of **proof of concept** (**POC**) development with four practical examples contextualized into a framework to simplify scale and refine our approach. We then explored the critical aspects of instrumentation, learning how to integrate GenAI models effectively into our applications and systems, and how to monitor and optimize their performance.

Finally, we addressed the vital topic of responsible AI, ensuring that as we push the boundaries of what's possible with GenAI, we do so in a way that is ethical, fair, and beneficial to society.

This journey from ideation to responsible implementation represents a blueprint for successfully leveraging GenAI in real-world applications. As these technologies continue to advance at a rapid pace, the principles and practices we've discussed will become increasingly important.

The future of AI is not just about building more powerful models; it's about building smarter, more responsible systems that augment human capabilities and contribute positively to our world. By following the approaches outlined in this book – from careful use case selection to rigorous POC development, thoughtful instrumentation, and unwavering commitment to responsible AI – you are well-equipped to lead the way in this exciting and transformative field.

As you move forward with your own GenAI projects, remember that success lies not just in technical implementation, but in the thoughtful consideration of how these powerful tools can be used to create genuine value while upholding the highest standards of ethics and responsibility. The journey of AI development is ongoing, and your role in shaping its future begins now.

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

`https://packt.link/genpat`

packt.com

Subscribe to our online digital library for full access to over 7,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

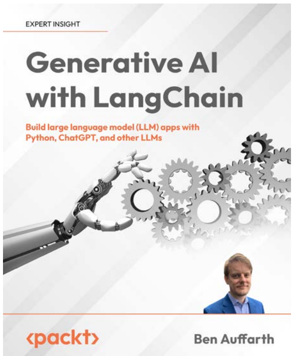
Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Fully searchable for easy access to vital information
- Copy and paste, print, and bookmark content

At www.packt.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:



Generative AI with LangChain

Ben Auffarth

ISBN: 978-1-83508-346-8

- Understand LLMs, their strengths and limitations
- Grasp generative AI fundamentals and industry trends
- Create LLM apps with LangChain like question-answering systems and chatbots
- Understand transformer models and attention mechanisms
- Automate data analysis and visualization using pandas and Python
- Grasp prompt engineering to improve performance
- Fine-tune LLMs and get to know the tools to unleash their power
- Deploy LLMs as a service with LangChain and apply evaluation strategies
- Privately interact with documents using open-source LLMs to prevent data leaks

Building LLM Powered Applications

Valentina Alto

ISBN: 978-1-83546-231-7

- Explore the core components of LLM architecture, including encoder-decoder blocks and embeddings
- Understand the unique features of LLMs like GPT-3.5/4, Llama 2, and Falcon LLM
- Use AI orchestrators like LangChain, with Streamlit for the frontend
- Get familiar with LLM components such as memory, prompts, and tools
- Learn how to use non-parametric knowledge and vector databases
- Understand the implications of LFMs for AI research and industry applications
- Customize your LLMs with fine tuning
- Learn about the ethical implications of LLM-powered applications

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit `authors.packtpub.com` and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Share your thoughts

Now you've finished *Generative AI Application Integration Patterns*, we'd love to hear your thoughts! If you purchased the book from Amazon, please `click here to go straight to the Amazon review page` for this book and share your feedback or leave a review on the site that you purchased it from.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Index

Symbols

A

B

N

O

P

Q

R

Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere?

Is your eBook purchase not compatible with the device of your choice?

Don't worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don't stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily.

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below:

https://packt.link/free-ebook/9781835887608

2. Submit your proof of purchase.
3. That's it! We'll send your free PDF and other benefits to your email directly.