

Operationalizing Generative AI on Vertex AI using MLOps

Authors: Anant Nawalgaria,
Gabriela Hernandez Larios, Elia Secchi,
Mike Styer, Christos Aniftos,
Onofrio Petragallo, and
Sokratis Kartakis



Google

Acknowledgements

Content contributors

Nenshad Bardoliwalla

Warren Barkley

Mikhail Chrestkha

Chase Lyall

Lakshmanan Sethu

Erwan Menard

Alan Blount

Curators and Editors

Antonio Gulli

Anant Nawalgaria

Grace Mollison

Technical Writer

Joey Haymaker

Designer

Michael Lanning

Table of contents

Introduction	6
What are DevOps and MLOps?	7
Lifecycle of a gen AI system	8
Discover	10
Develop and experiment	11
The foundational model paradigm	12
The core component of LLM Systems: A prompted model component	14
Chain & Augment	17
Tuning & training	21
Continuous Training & Tuning	22
Data Practices	24
Evaluate	28
Deploy	31
Deployment of gen AI systems	32
Version control	32
Continuous integration of gen AI systems	33
Continuous delivery of gen AI systems	34
Deployment of foundation models	35

Infrastructure validation	35
Compression and optimization	36
Deployment, packaging, and serving checklist	37
Logging and monitoring	38
Govern	43
Extend MLOps for gen AI to Agents	44
Agent Lifecycle	45
Tool Orchestration	47
Tool Types & Environments	48
Tool Registry	50
Tool Selection Strategies at Scale	51
Agent Evaluation & Optimization	53
Observability and Memory	55
Deploying an Agent to Production	58
Operations: People & Processes	60
The role of an AI platform for gen AI operations	65
Key components of Vertex AI for gen AI	66
Discover: Vertex Model Garden	67
Prototype: Vertex AI Studio & Notebooks	70
Customize: Vertex AI training & tuning	71
Train	72
Tune	72

Orchestrate	74
Chain & Augment: Vertex AI Grounding, Extensions, and RAG building blocks	75
Evaluate: Vertex AI Experiments, Tensorboard, & evaluation pipelines	79
Experiment	80
Evaluation	81
Predict: Vertex AI endpoints & monitoring	81
Govern: Vertex AI Feature Store, Model Registry, and Dataplex	83
Summary	85
Endnotes	87



Emergence of foundation models and generative AI (gen AI) has introduced a new era for building AI systems.

Introduction

The emergence of foundation models and generative AI (gen AI) has introduced a new era for building AI systems. Selecting the right model from a diverse range of architectures and sizes, curating data, engineering optimal prompts, tuning models for specific tasks, grounding model outputs in real-world data, optimizing hardware - these are just a few of the novel challenges that large models introduce.

This whitepaper delves into the fundamental tenets of MLOps and the necessary adaptations required for the domain of gen AI and Foundation Models. We also examine the diverse range of Vertex AI products, specifically tailored to address the unique demands of foundation models and gen AI-based applications. Through this exploration we uncover how Vertex AI, with its solid foundations of AI infrastructure and MLOps tools, expands its capabilities to provide a comprehensive MLOps platform for gen AI.

What are DevOps and MLOps?

DevOps is a software engineering methodology that aims to bridge the gap between development (Dev) and operations (Ops). It promotes collaboration, automation, and continuous improvement to streamline the software development lifecycle, introducing practices such as continuous integration and continuous delivery.

MLOps builds upon DevOps principles to address the unique challenges of operationalizing Machine Learning systems rapidly and reliably. In particular, MLOps tackles the experimental nature of ML through practices like:

- Data validation: Ensuring the quality and integrity of training data.
- Model evaluation: Rigorously assessing model performance with appropriate metrics.
- Model monitoring: Tracking model behavior in production to detect and mitigate drift.
- Tracking & reproducibility: Maintaining meticulous records for experiment tracking and result reproduction.

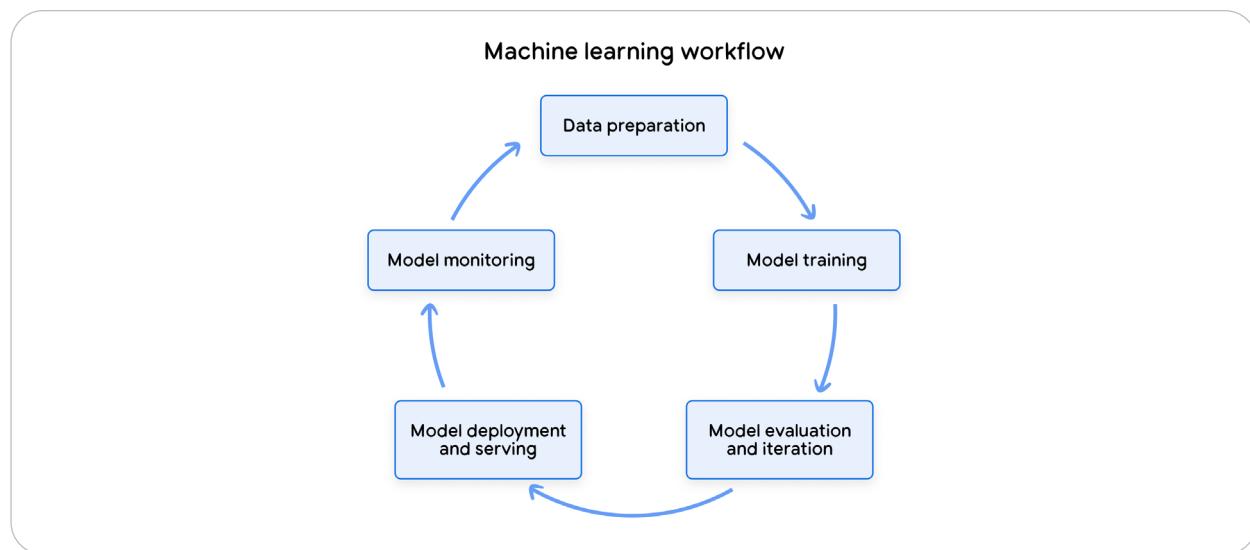


Figure 1. Machine learning workflow

Lifecycle of a gen AI system

Imagine deploying your first chatbot after months of dedicated work, and it's now interacting with users and answering questions. Behind this seemingly simple interaction lies the complex and fascinating life cycle of a gen AI System, which can be broken down into five key moments.

First in the **discovery phase**, developers and AI engineers must navigate the expanding landscape of available models to identify the most suitable one for their specific gen AI application. They must consider each model's strengths, weaknesses, and costs to make an informed decision.

Next, **development and experimentation** become paramount, with prompt engineering playing a crucial role in crafting and refining input prompts to elicit desired outputs based on an understanding of the model's intricacies. Few-shot learning, where examples are provided, can further guide model behavior, while additional customization may involve parameter-efficient fine-tuning (PEFT). Most gen AI systems also involve model chaining, which refers to orchestrating calls to multiple models in a specific sequence to create a workflow.

Data engineering practices have a critical role across all development stages, with factual grounding (ensuring the model's outputs are based on accurate, up-to-date information) and recent data from internal and enterprise systems being essential for reliable outputs. Tuning data is often needed to adapt models to specific tasks, styles, or to rectify persistent errors.

Deployment needs to manage many new artifacts in the deployment process, including prompt templates, chain definitions, embedding models, retrieval data stores, and fine-tuned model adapters among others. These artifacts each have unique governance requirements,

necessitating careful management throughout development and deployment. Gen AI system deployment also needs to account for the technical capabilities of the target infrastructure, ensuring that system hardware requirements are fulfilled.

Continuous monitoring in production ensures improved application performance and maintains safety standards through responsible AI techniques, such as ensuring fairness, transparency, and accountability in the model's outputs.

Continuous Improvement as a concept is still key for Gen AI-based applications, though with a twist. For most Gen AI applications, instead of training models from scratch, we're taking foundation models (FMs) and then adapting them to our specific use case. This means constantly tweaking these FMs through prompting techniques, swapping them out for newer versions, or even combining multiple models for enhanced performance, cost efficiency, or reduced latency. Traditional **continuous training** still holds relevance for scenarios when recurrent fine-tuning or incorporating human feedback loops are still needed.

Naturally, this lifecycle assumes that the foundational model powering the gen AI system is already operationalized. It's important to recognize that not all organizations will be directly involved in this part of the process. In particular, the operationalization of foundational models is a specialized set of tasks that is typically only relevant for a select few companies with the necessary resources and expertise.

Because of that, this whitepaper will focus on practices required to operationalize gen AI applications using and adapting existing foundation models, referring to other whitepapers in the book should you want to deep dive into how foundational models are operationalized.

This includes active areas of research such as model pre-training, alignment (ensuring the model's outputs align with the desired goals and values), evaluation or serving.

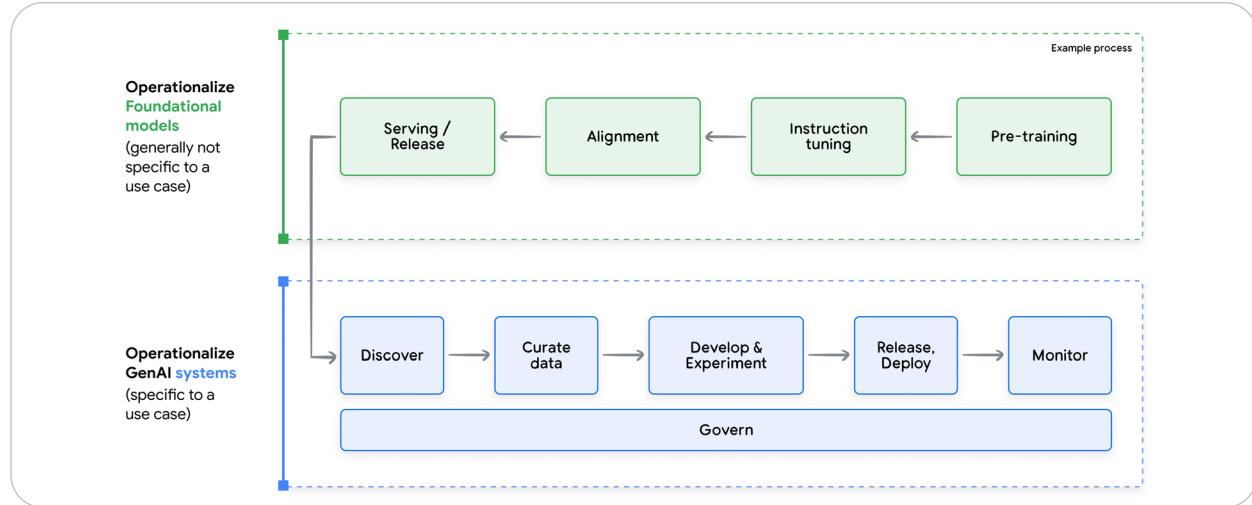


Figure 2. Lifecycle of a Foundational Model & gen AI system and relative operationalization practices

Discover

As mentioned before, building foundational models from scratch is resource-intensive. Training costs and data requirements are substantial, pushing most practitioners towards adapting existing foundation models through techniques like fine-tuning and prompt engineering. This shift highlights a crucial need: efficiently discovering the optimal foundation model for a given use case.

These two characteristics of the gen AI landscape make model discovery an essential MLOps practice:

- 1. An abundance of models:** The past year has witnessed an explosion of open-source and proprietary foundation models. Navigating this complex landscape, each with varying architectures, sizes, training datasets, and licenses, requires a systematic approach to identify suitable candidates for further evaluation.

2. **No one-size-fits-all solution:** Each use case presents unique requirements, demanding a nuanced analysis of available models across multiple dimensions.

Here are some factors to consider when exploring models:

1. **Quality:** Early assessments can involve running test prompts or analyzing public benchmarks and metrics to gauge output quality.
2. **Latency & throughput:** These factors directly impact user experience. A chatbot demands lower latency than batch-processed summarization tasks.
3. **Development & maintenance time:** Consider the time investment for both initial development and ongoing maintenance. Managed models often require less effort than self-deployed open-source alternatives.
4. **Usage cost:** Factor in infrastructure and consumption costs associated with using the chosen model.
5. **Compliance:** Assess the model's ability to adhere to relevant regulations and licensing terms.

Because the activity of discovery has become so important for gen AI systems, many model discoverability platforms were created to support this need. An example of that is Vertex Model Garden,¹ which is explored later in this whitepaper.

Develop and experiment

The process of development and experimentation remains iterative and orchestrated while building gen AI applications. Each experimental iteration involves a tripartite interplay between data refinement, foundation model(s) selection and adaptation, and

rigorous evaluation. Evaluation provides crucial feedback, guiding subsequent iterations in a continuous feedback loop. Subpar performance might call for gathering more data, augmenting data, or further curating the data. Similarly, the adaptation of the foundation model itself might need tweaking - optimizing prompts, applying fine-tuning techniques, or even swapping it out for a different one altogether. This iterative refinement cycle, driven by evaluation insights, is just as critical for optimizing gen AI applications as it's always been for traditional machine learning.

The foundational model paradigm

Foundation models differ from predictive models most importantly because they are multi-purpose models. Instead of being trained for a single purpose, on data specific to that task, foundation models are trained on broad datasets, and therefore can be applied to many different use cases. This distinction brings with it several more important differences between foundation models and predictive models.

Foundation models also exhibit what are known as 'emergent properties',² capabilities that emerge in response to specific input without additional training. Predictive models are only able to perform the single function they were trained for; a traditional French-English translation model, for instance, cannot also solve math problems.

Foundation models are also highly sensitive to changes in their input. The output of the model and the task it performs are strongly affected, indeed determined, by the input to the model. A foundation model can be made to perform translation, generation, or classification tasks simply by changing the input. Even insignificant changes to the input can affect its ability to correctly perform that task.

These new properties of foundation models have created a corresponding paradigm shift in the practices required to develop and operationalize Gen AI systems. While models in the predictive AI context are self-sufficient and task-specific, gen AI models are multi-purpose and need an additional element beyond the user input to function as part of a gen AI Application: a prompt, and more specifically, a prompt template, defined as a set of instructions and examples along with placeholders to accommodate user input. A prompt template, along with dynamic data such as user input, can be combined to create a complete prompt, the text that is passed as input to the foundation model.

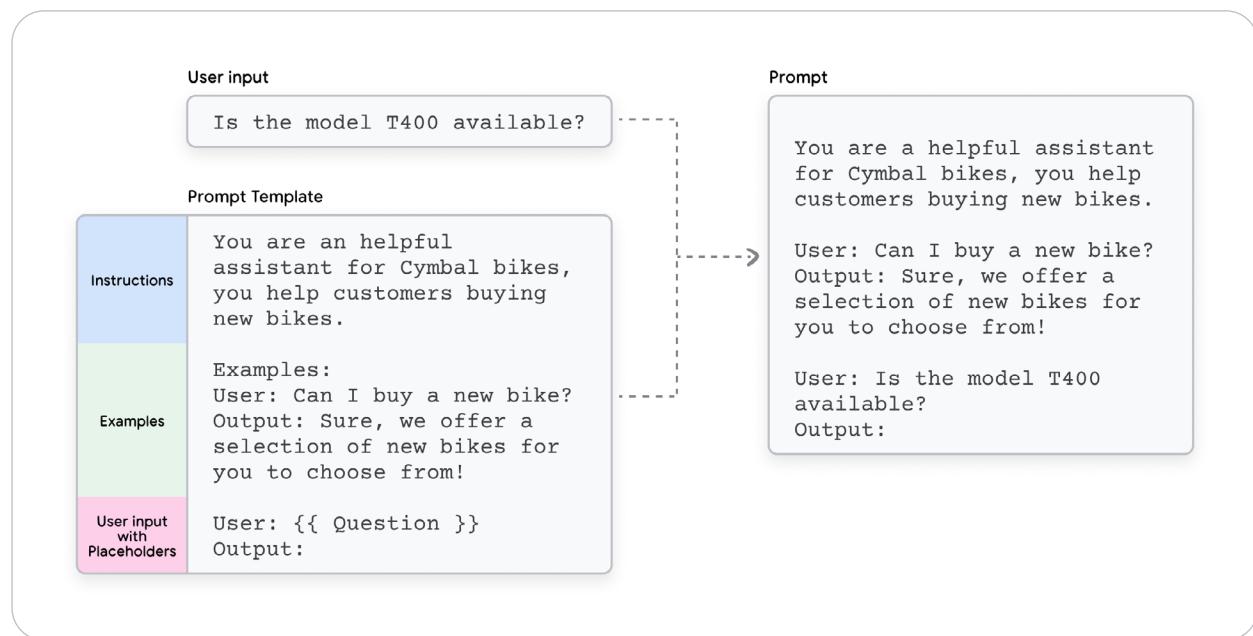


Figure 3. How Prompt Template and User input can be combined to create a prompt

The core component of LLM Systems: A prompted model component

The presence of the prompt element is a distinguishing feature of gen AI applications. Neither the model nor the prompt is sufficient for the generation of content; gen AI needs the combination of both. We refer to the combination as a ‘prompted model component’. This is the smallest independent component sufficient to create an LLM application. The prompt does not need to be very complicated. It can be a simple instruction, such as “translate the following sentence from English to French”, followed by the sentence to be translated. Without that preliminary instruction, though, a foundation model would not perform the desired translation task. So a prompt, even just a basic instruction, is necessary along with the input to get the foundation model to do the task required by the application.

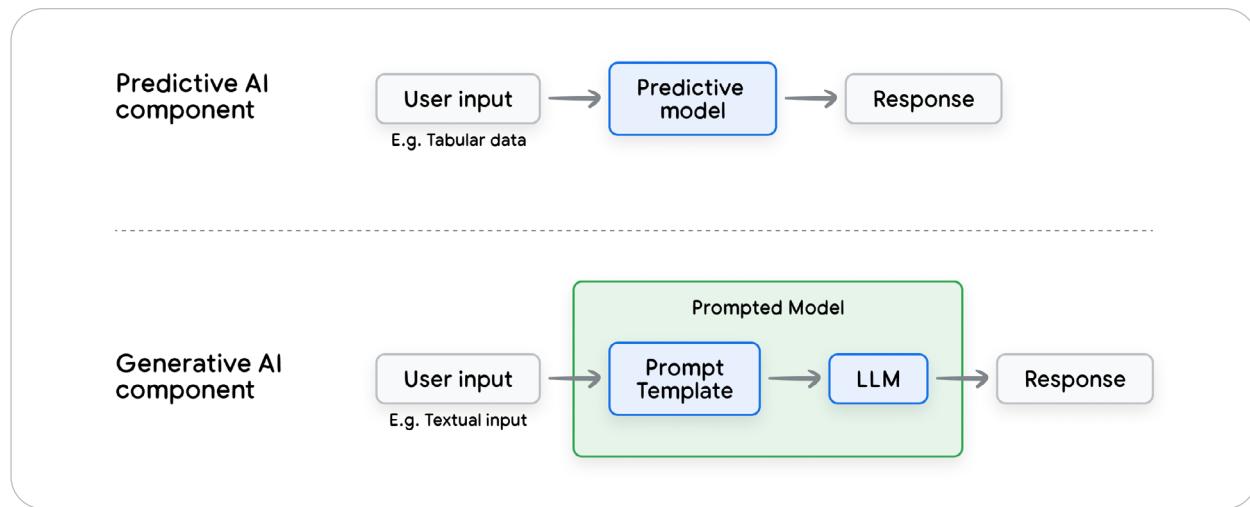


Figure 4. Predictive AI unit compared with the gen AI unit

This introduces an important distinction when it comes to MLOps practices for gen AI. In the development of a gen AI System, experimentation and iteration need to be done in the context of a *prompted model component*, the combination of a model and a prompt. The Gen

AI experimentation cycle typically begins with testing variations of the prompt – changing the wording of the instructions, providing additional context, or including relevant examples, etc., and evaluating the impact of those changes. This practice is commonly referred to as prompt engineering.

Prompt engineering involves two iterative steps:

1. **Prompting:** Crafting and refining prompts to elicit desired behaviors from a foundational model for a specific use case.
2. **Evaluation:** Assessing the model's outputs, ideally programmatically, to gauge its understanding and success in fulfilling the prompt's instructions.

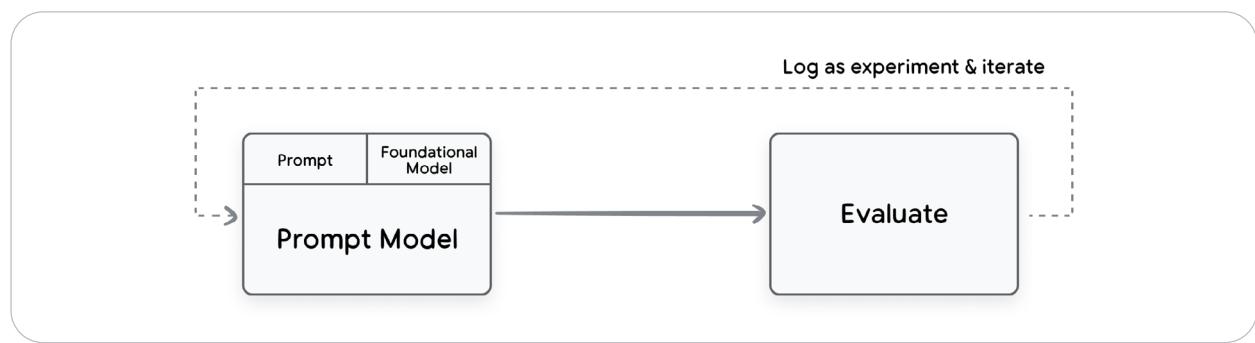


Figure 5. The activity of prompt engineering

Results of an evaluation can be optionally registered as part of an experiment, to allow for result tracking. Since the prompt itself is a core element of the prompt engineering process, it becomes a first class citizen within the artifacts part of the experiment.

However, we need to identify which type of artifacts they are. In the good old days of Predictive AI, we had clear lines – data was one thing, pipelines and code another. But with the “Prompt” paradigm in gen AI, those lines get blurry. Think about it: prompts can include anything from context, instructions, examples, guardrails to actual internal or external data pulled from somewhere else. So, are prompts data? Are they code?

To address these questions, a hybrid approach is needed, recognizing that a prompt has different components and requires different management strategies. Let's break it down:

Prompt as Data: Some parts of the prompt will act just like data. Elements like few-shot examples, knowledge bases, and user queries are essentially data points. For these components, we need data-centric MLOps practices such as data validation, drift detection, and lifecycle management.

Prompt as Code: Other components such as context, prompt templates, guardrails are more code-like. They define the structure and rules of the prompt itself. Here, we need code-centric practices such as approval processes, code versioning, and testing.

As a result, when applying MLOps practices to gen AI, it becomes important to have in place processes that give developers easy storage, retrieval, tracking, and modification of prompts. This allows for fast iteration and principled experimentation. Often one version of a prompt will work well with a specific version of the model and less well with a different version. In tracking the results of an experiment, both the prompt and its components version, and the model version must be recorded and stored along with metrics and output data produced by the prompted model.

The fact that development and experimentation in gen AI requires working with the prompt and the model together introduces changes in some of the common MLOps practices, compared to the predictive AI case in which experimentation is done by changing the model alone. Specifically, several of the MLOps practices need to be expanded to consider the prompted model component together as a unit. This includes practices like **evaluation**, **experiment tracking**, **model adaptation and deployment**, and **artifact management**, which will be discussed below in this whitepaper.

Chain & Augment

Gen AI models, particularly large language models (LLMs), face inherent challenges in maintaining recency and avoiding hallucinations. Encoding new information into LLMs requires expensive and data-intensive pre-training, posing a significant hurdle. Additionally, LLMs might be unable to solve complex challenges, especially when step-by-step reasoning is required. Depending on the use case, leveraging only one prompted model to perform a particular generation might not be sufficient. To solve this issue, leveraging a divide and conquer approach, several prompted models can be connected together, along with calls to external APIs and logic expressed as code. A sequence of prompted model components connected together in this way is commonly known as a chain.

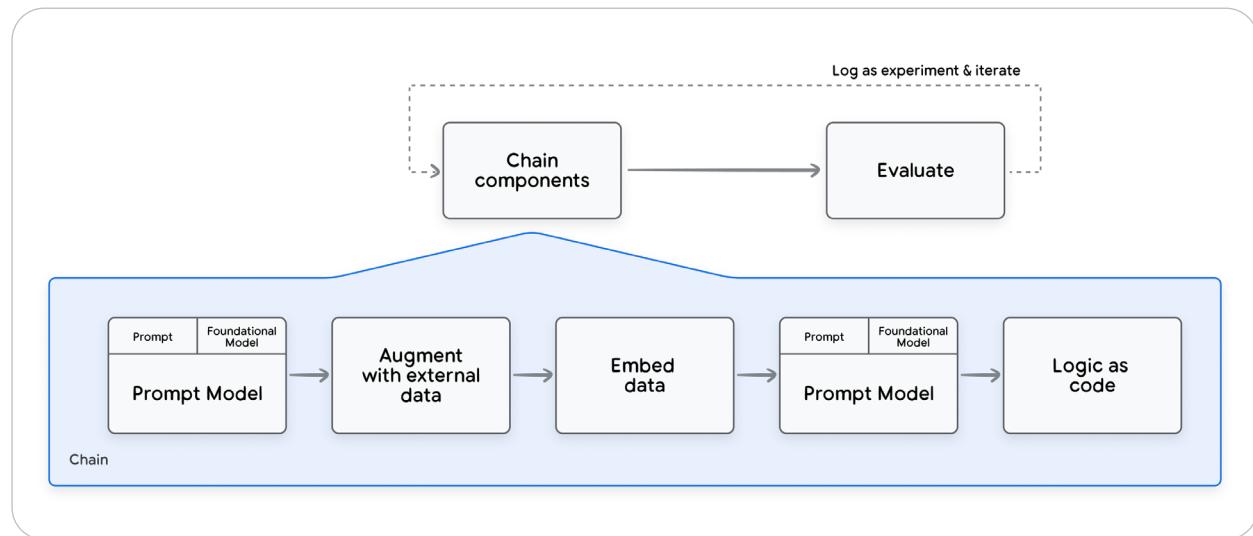


Figure 6. Components of a chain and relative development process

Two common chain-based patterns that have emerged to mitigate recency and hallucinations are retrieval augmented generation (RAG)³ and Agents.

- RAG addresses these challenges by augmenting pre-trained models with “knowledge” retrieved from databases, bypassing the need for pre-training. This enables grounding and reduces hallucinations by incorporating up-to-date factual information directly into the generation process.
- Agents, popularized by the ReAct prompting technique,⁴ leverage LLMs as mediators interacting with various tools, including RAG systems, internal or external APIs, custom extensions, or even with other agents. This enables complex queries and real-time actions by dynamically selecting and utilizing relevant information sources. The LLM, acting as an agent, interprets the user’s query, decides which tool to utilize, and how to formulate the response based on the retrieved information.

RAG and Agents approaches can be combined to create multi-agent systems connected to large information networks, enabling sophisticated query handling and real-time decision-making.

The orchestration of different models, logic and APIs is not a novelty of gen AI Applications. For example, recommendation engines have long combined collaborative filtering models, content-based models, and business rules to generate personalized product recommendations for users. Similarly, in fraud detection, machine learning models are integrated with rule-based systems and external data sources to identify suspicious activities.

What makes these chains of gen AI components different, is that, we can't a priori characterize or cover the **distribution** of component inputs, which makes the individual components much harder to evaluate and maintain in isolation.

This results in a paradigm shift in how AI applications are being developed for gen AI.

Unlike Predictive AI where it is often possible to iterate on the separate models and components in isolation to then chain in the AI application, in gen AI it's often easier to develop a chain in integration, performing experimentation on the chain end-to-end, iterating over chaining strategies, prompts, the underlying foundational models and other APIs in a coordinated manner to achieve a specific goal. No feature engineering, data collection, or further model training cycles is often needed; just changes to the wording of the prompt template.

The shift towards MLOps for gen AI, in contrast to predictive AI, brings forth a new set of demands. Let's break down these key differences:

1. **Evaluation:** Because of their tight coupling, chains need end-to-end evaluation, not just on a per-component basis, to gauge their overall performance and the quality of their output. In terms of evaluation techniques and metrics, evaluating chains is not dissimilar to evaluating prompted models. Please refer to the below segment on evaluation for more details on these approaches.
2. **Versioning:** A chain needs to be managed as a complete artifact in its entirety. The chain configuration should be tracked with its own revision history for analysis, reproducibility, and understanding the impact of changes on output. Logging should also include the inputs, outputs, and intermediate states of the chain, and any chain configurations used during each execution.
3. **Continuous Monitoring:** Establishing proactive monitoring systems is vital for detecting performance degradation, data drift, or unexpected behavior in the chain. This ensures early identification of potential issues to maintain the quality of the generated output. The activity of monitoring Chains is discussed in detail in the section 'Logging and Monitoring'.

4. **Introspection:** The ability to inspect the internal data flows of a chain (inputs and outputs from each component) as well as the inputs and outputs of the entire chain is paramount. By providing visibility into the data flowing through the chain and the resulting content, developers can pinpoint the sources of errors, biases, or undesirable behavior.

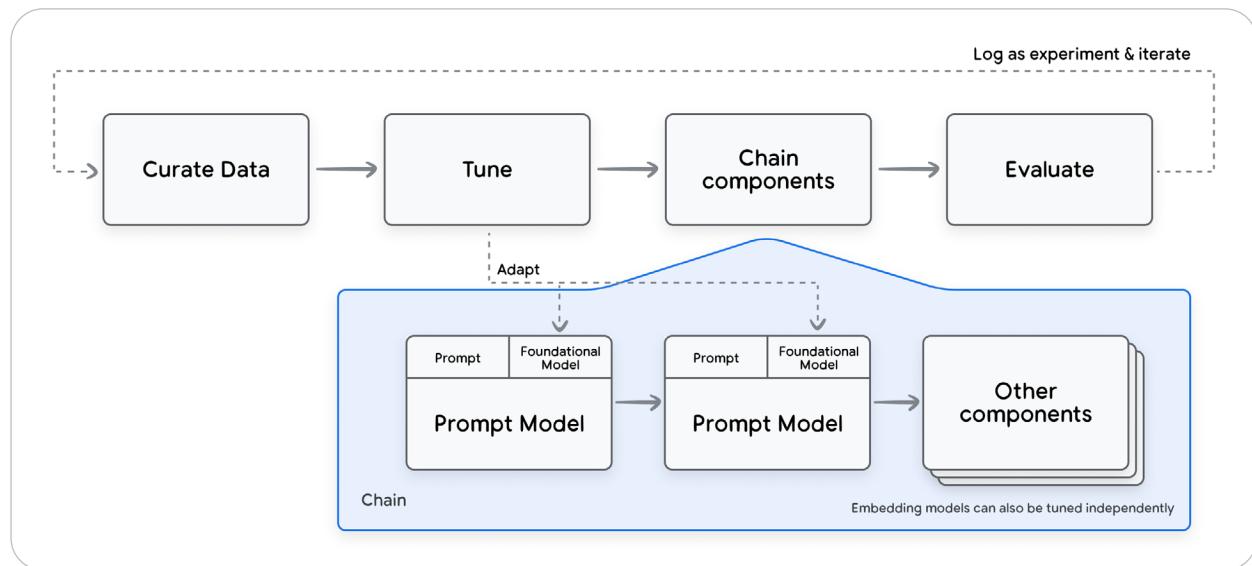


Figure 7. Putting together chains, prompted models and model tuning

There are several products in Vertex AI that can support the need for chaining and augmentation, including Grounding as a service,⁵ Extensions,⁶ Vector Search,⁷ Agent Builder,⁸ and more. We discuss the products in the section “Role of an AI Platform”. LangChain⁹ is also integrated with the Vertex SDK,¹⁰ and can be used alongside the core Vertex products to define and configure gen AI chained applications.

Tuning & training

When developing a gen AI use case and a specific task that involves LLMs, it can be difficult, especially for complex tasks, to rely on only prompt engineering and chaining to solve it. To improve task performance practitioners often also need to fine-tune the model directly. Fine-tuning lets you actively change the layers or a subset of layers of the LLM to optimize the capability of the model to perform a certain task. Two of the most common ways of tuning a model are:

1. Supervised fine-tuning: This is where we train the model in a supervised manner, teaching it to predict the right output sequence for a given input.
2. Reinforcement Learning from Human Feedback (RLHF): In this approach, we first train a reward model to predict what humans would prefer as a response. Then, we use this reward model to nudge the LLM in the right direction during the tuning process. Like having a panel of human judges guiding the model's learning.

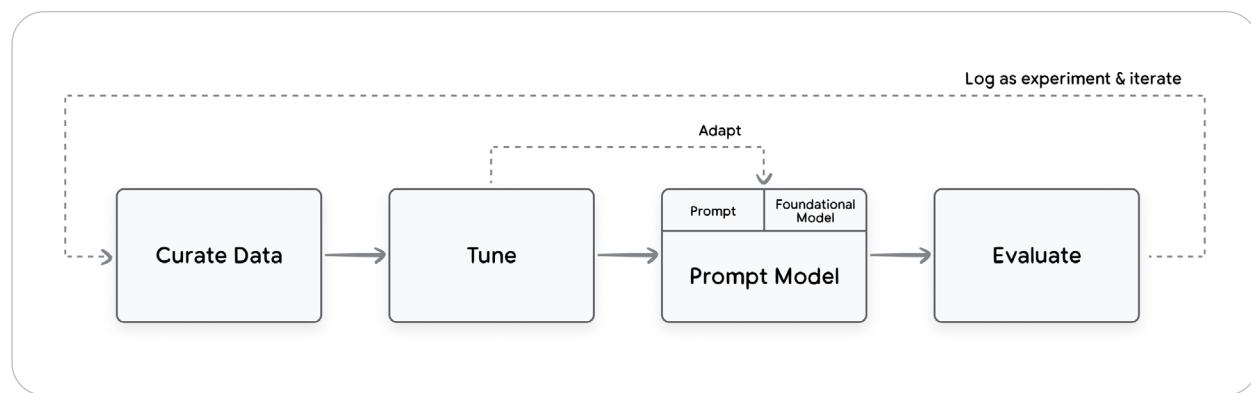


Figure 8. Putting together chains, prompted models and model tuning

When viewed through the MLOps lens, fine-tuning shares similar requirements with model training:

1. The capability to track artifacts being part of the tuning job. This includes for example the input data or the parameters being used to tune the model.
2. The capability to measure the impact of the tuning. This translates into the capability to perform evaluation of the tuned model for the specific tasks it was trained on and to compare results with previously tuned models or frozen models for the same task.

Platforms like Vertex AI¹¹ (and the Google Cloud platform more broadly) provide a robust suite of services designed to address these MLOps requirements: Vertex Model Registry,¹² for instance, provides a centralized storage location for all the artifacts created during the tuning job, and Vertex Pipelines¹³ streamlines the development and management of these tuning jobs. Dataplex,¹⁴ meanwhile, provides an organization-wide data fabric for data lineage and governance and integrates well with both Vertex AI and BigQuery.¹⁵ What's more, these products provide the same governance capability for both predictive and gen AI applications, meaning customers do not need separate products or configurations to manage generative versus AI development.

Continuous Training & Tuning

In machine learning operations (MLOps), continuous training is the practice of repeatedly retraining machine learning models in a production environment. This is done to ensure that the model remains up-to-date and performs well as real-world data patterns change over time. For gen AI models, continuous tuning of the models is often more practical than retraining from scratch due to the high data and computational costs involved.

The approach to continuous tuning depends on the specific use case and goals. For relatively static tasks like text summarization, the continuous tuning requirements may be lower. But for dynamic applications like chatbots that need constant human alignment, more frequent tuning using techniques like RLHF based on human feedback is necessary.

To determine the right continuous tuning strategy, AI practitioners must carefully evaluate the nature of their use case and how the input data evolves over time. Cost is also a major consideration, as the compute infrastructure greatly impacts the speed and expense of tuning. We discuss in detail monitoring of GenAI systems in the Logging and Monitoring section of this whitepaper.

Graphics processing units (GPUs) and tensor processing units (TPUs) are key hardware for fine-tuning. GPUs, known for their parallel processing power, are highly effective in handling the computationally intensive workloads and often associated with training and running complex machine learning models. TPUs, on the other hand, are specifically designed by Google for accelerating machine learning tasks. TPUs excel in handling large matrix operations common in deep learning neural networks.

To manage costs, techniques like model quantization can be applied. This represents model weights and activations using lower-precision 8-bit integers rather than 32-bit floats, which reduces computational and memory requirements.

We discuss in detail the support for tuning in Vertex AI in the Customize: Vertex AI Training & Tuning section.

Data Practices

Traditionally, ML model behavior was dictated solely by its training data. While this still holds true for foundation models – trained on massive, multilingual, multimodal datasets – gen AI applications built on top of them introduce a new twist: model behavior is now determined by how you adapt the model using different types of input data (Figure. 9).

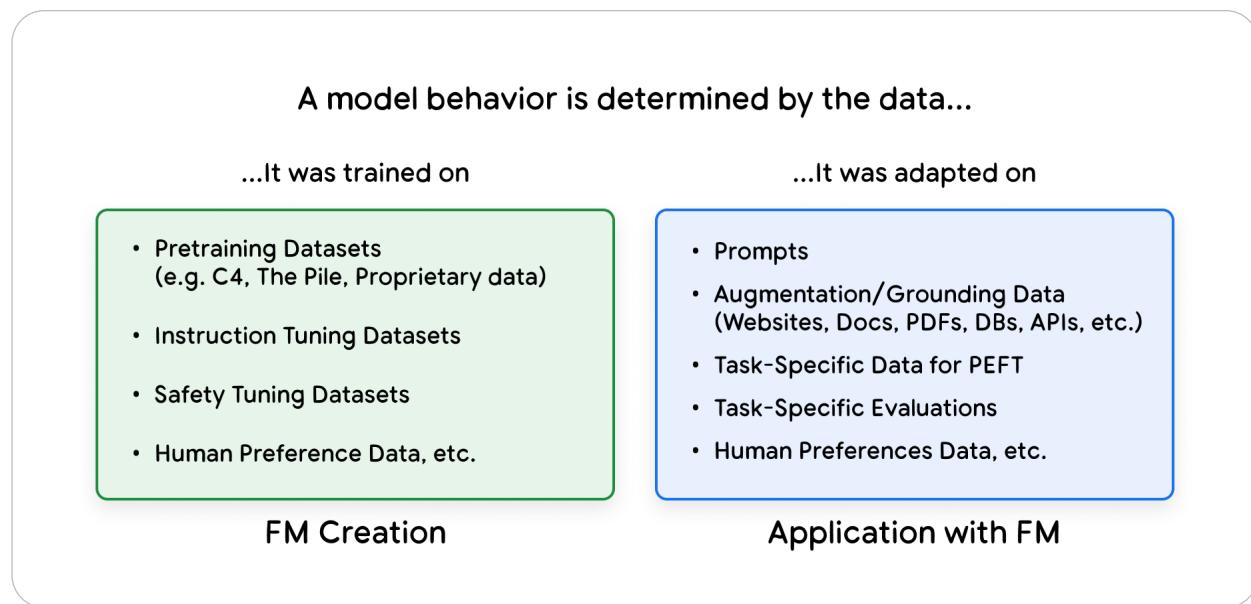


Figure 9. Examples of data spectrum for foundation models – creation (left) vs. adaptation (right)

The key difference between traditional predictive ML and gen AI lies in where you start. In predictive ML, the data is paramount. You spend a lot of time on data engineering, and if you don't have the right data, you cannot build an application. Gen AI takes a unique approach to this matter. You start with a foundation model, some instructions and maybe a few example inputs (in-context learning). You can prototype and launch an application with surprisingly little data.

This ease of prototyping, however, comes with a challenge. Traditional predictive AI relies on apriori well-defined dataset(s). In gen AI, a single application can leverage various data types, from completely different data sources, all working together (Figure 10). Let's explore some of these data types:

- **Conditioning prompts:** These are essentially instructions given to the Foundation Model (FM) to guide its output, setting boundaries of what it can generate.
- **Few-shot examples:** A way to show the model what you want to achieve through input-output pairs. This helps the model grasp the specific task(s) at hand, and in many cases, it boosts performances.
- **Grounding/augmentation data:** Data coming from either external APIs (like Google Search) or internal APIs and data sources. This data permits the FM to produce answers for a specific context, keeping responses current, relevant without retraining the entire FM. This type of data also supports reducing hallucinations.
- **Task-specific datasets:** These are used to fine-tune an existing FM for a particular task, improving its performance in that specific area.
- **Human preference datasets:** These capture feedback on generated outputs, helping refine the model's ability to produce outputs that align with human preferences.
- **Full pre training corpora:** These are massive datasets used to initially train foundation models. While application builders may not have access to them nor the tokenizers, the information encoded in the model itself will influence the application's output and performance.

This is not an exhaustive list. The variety of data used in gen AI applications is constantly growing and evolving.

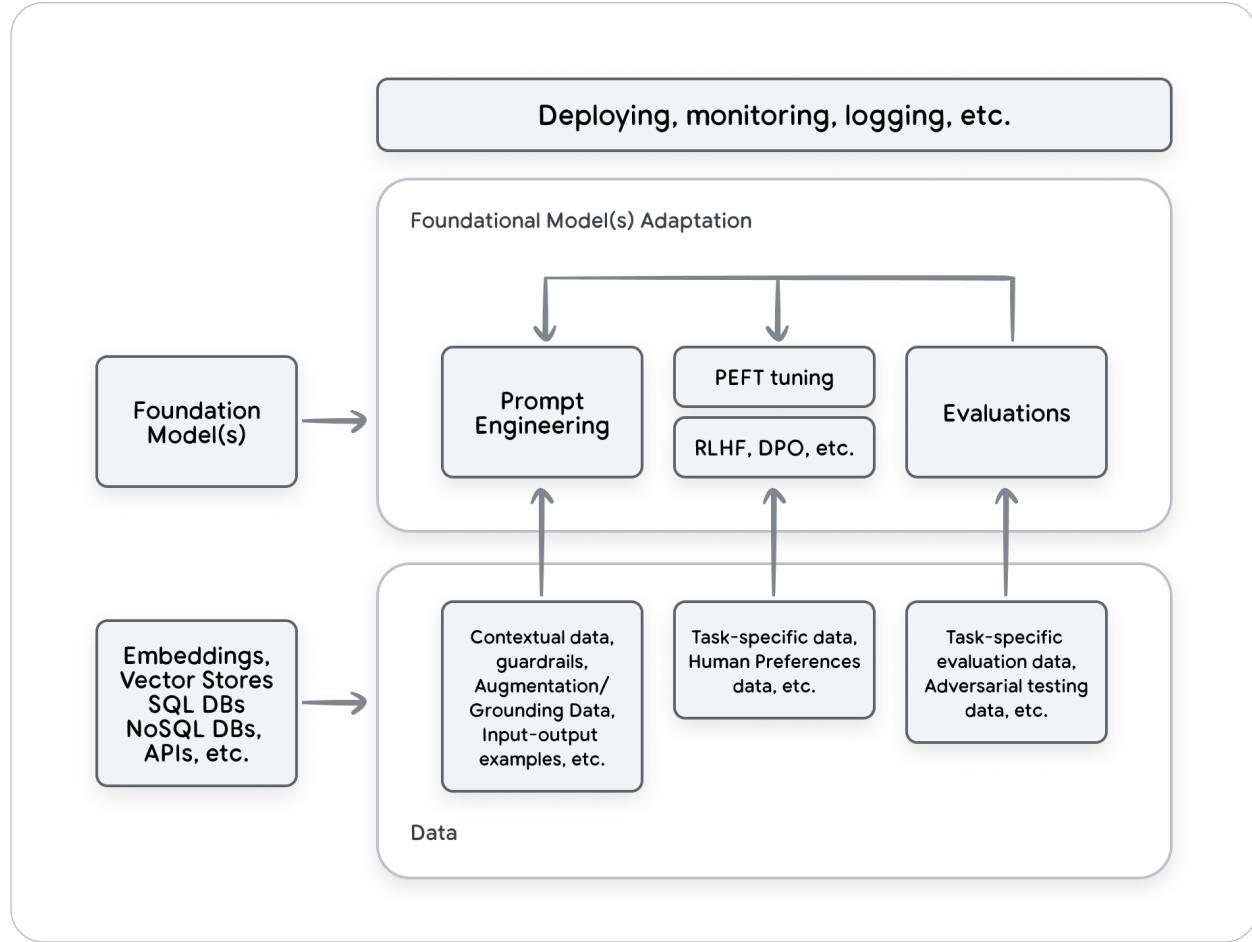


Figure 10. Example of high-level data and adaptations landscape for developing gen AI applications using existing foundation models

This diverse range of data adds another complexity layer in terms of data organization, tracking and lifecycle management. Take a RAG-based application as an example: it might involve rewriting user queries, dynamically gathering relevant examples using a curated set of examples, querying a vector database, and combining it all with a prompt template. This involves managing multiple data types: user queries, vector databases with curated few-shot examples and company information, and prompt templates.

Each data type needs careful organization and maintenance. For example, the vector database requires processing data into embeddings, optimizing chunking strategies, and ensuring only relevant information is available. The prompt template itself needs versioning and tracking, the user queries need rewriting, etc. This is where traditional MLOps and DevOps best practices come into play, with a twist. We need to ensure reproducibility, adaptability, governance, and continuous improvement using all the data required in an application as a whole but also individually. Think of it this way: in predictive AI, the focus was on well-defined data pipelines for extraction, transformation, and loading. In gen AI, it's about building pipelines to manage, evolve, adapt and integrate different data types in a versionable, trackable, and reproducible way.

As mentioned earlier, fine-tuning foundation models (FMs) can boost gen AI app performance, but it needs data. You can get this data by launching your app and gathering real-world data, generating synthetic data, or a mix of both. Using large models to generate synthetic data is becoming popular because it speeds things up, but it's still good to have a human check the results for quality assurance. Here are few ways to leverage large models for data engineering purposes:

1. **Synthetic data generation:** This process involves creating artificial data that closely resembles real-world data in terms of its characteristics and statistical properties, often being done with a large and capable model. This synthetic data serves as additional training data for gen AI, enabling it to learn patterns and relationships even when labeled real-world data is scarce.
2. **Synthetic data correction:** This technique focuses on identifying and correcting errors and inconsistencies within existing labeled datasets. By leveraging the power of larger models, gen AI can flag potential labeling mistakes and propose corrections, improving the quality and reliability of the training data.

3. **Synthetic data augmentation:** This approach goes beyond simply generating new data. It involves intelligently manipulating existing data to create diverse variations while preserving essential features and relationships. Thus, gen AI can encounter a broader range of scenarios during training, leading to improved generalization and ability to generate nuanced and relevant outputs.

Evaluating gen AI, unlike predictive AI, is tricky. You don't usually know the training data distribution of the foundational models. Building a custom evaluation dataset reflecting your use case is essential. This dataset should cover essential, average, and edge cases. Similar to fine-tuning data, you can leverage powerful language models to generate, curate, and augment data for building robust evaluation datasets.

Evaluate

Even if only prompt engineering is performed, as any experimental process, it does require evaluation in order to iterate and improve. This makes the evaluation process a core activity of the development of any gen AI systems.

In the context of gen AI systems, evaluation might have different degrees of automation: from entirely driven by humans to entirely automated by a process.

In the early days of a project, when you're still prototyping, evaluation is often a manual process. Developers eyeball the model's outputs, getting a qualitative sense of how it's performing. But as the project matures and the number of test cases balloons, manual evaluation becomes a bottleneck. That's when automation becomes key.

Automating evaluation has two big benefits. First, it lets you move faster. Instead of spending time manually checking each test case, you can let the machines do the heavy lifting. This means more iterations, more experiments, and ultimately, a better product. Second, automation makes evaluation more reliable. It takes human subjectivity out of the equation, ensuring that results are reproducible.

But automating evaluation for gen AI comes with its own set of challenges.

For one, both the inputs (prompts) and outputs can be incredibly complex. A single prompt might include multiple instructions and constraints that the model needs to juggle. And the outputs themselves are often high-dimensional - think a generated image or a block of text. Capturing the quality of these outputs in a simple metric is tough.

There are some established metrics, like BLEU for translations and ROUGE for summaries, but they don't always tell the full story. That's where custom evaluation methods come in. One approach is to use another foundational model as a judge. For example, you could prompt a large language model to score the quality of generated texts across various dimensions. This is the idea behind techniques like AutoSxS.¹⁶

Another challenge is the subjective nature of many evaluation metrics for gen AI. What makes one output 'better' than another can often be a matter of opinion. The key here is to make sure your automated evaluation aligns with human judgment. You want your metrics to be a reliable proxy for what people would think. And to ensure comparability between experiments, it's crucial to lock down your evaluation approach and metrics early in the development process.

Lack of ground truth data is another common hurdle, especially in the early stages of a project. One workaround is to generate synthetic data to serve as a temporary ground truth, which can be refined over time with human feedback.

Finally, comprehensive evaluation is essential for safeguarding gen AI applications against adversarial attacks. Malicious actors can craft prompts to try to extract sensitive information or manipulate the model's outputs. Evaluation sets need to specifically address these attack vectors, through techniques like prompt fuzzing (feeding the model random variations on prompts) and testing for information leakage.

Automating the evaluation process ensures speed, scalability and reproducibility

An automation of the evaluation process can be considered a proxy for the human judgment

Depending on the use case, the evaluation process will require a high degree of customization.

To ensure comparability it is essential to stabilize the evaluation approach, metrics, and ground truth data as early as possible in the development phase.

It is possible to generate synthetic ground truth data to accommodate for the lack of real ground truth data.

It is important to include test cases of adversarial prompting as part of the evaluation set to test the reliability of the system itself for these attacks.

Table 1. Key suggestions to approach evaluation of gen AI systems

Deploy

It should be clear by this point that production gen AI applications are complex systems with many interacting components. Some of the common components discussed include multiple prompts, models, adapter layers and external data sources. In deploying a gen AI system to production, all these components need to be managed and coordinated with the previous stages of gen AI system development. Given the novelty of these systems, best practices for deployment and management are still evolving, but we can discuss observations and recommendations for these components and indicate how to address the major concerns.

Deploying gen AI solutions necessarily involves multiple steps. For example, a single application might utilize several large language models (LLMs) alongside a database, all fed by a dynamic data pipeline. Each of these components potentially requires its own deployment process.

For clarity, we distinguish between two main types of deployment:

1. **Deployment of gen AI systems:** This focuses on operationalizing a complete system tailored for a specific use case. It encompasses deploying all the necessary elements - the application, chosen LLMs, database, data pipelines, and any other relevant components - to create a functioning end-user solution.
2. **Deployment of foundational models:** This applies to open-weight models, where the model weights are publicly available on platforms like Vertex Model Garden or Hugging Face, or privately trained models. Deployment in this scenario centers around making the foundational model itself accessible to users. Given their multipurpose nature, these deployments often aim to support various potential use cases.

Deployment of gen AI systems

Deployment of gen AI systems is broadly similar to deployment of any other complex software system. Most of the system components – databases, Python applications, etc. – are also found in other non-gen AI applications. As a result, our general recommendation is to manage these components using standard software engineering practices such as version control¹⁷ and Continuous Integration / Continuous Delivery (CI/CD).¹⁸

Version control

Gen AI experimentation is an iterative process involving repeated cycles of development, evaluation, and modification. To ensure a structured and manageable approach, it's crucial to implement strict versioning for all modifiable components. These components include:

- **Prompt templates:** Unless leveraging specific prompt management solutions, version them through standard version control tools like Git.
- **Chain definitions:** The code defining the chain (including API integrations, database calls, functions, etc.) should also be versioned using tools like Git. This provides a clear history and enables easy rollback if needed.
- **External datasets:** In retrieval augmented generation (RAG) systems, external datasets play a key role. It's important to track these changes and versions of these datasets for reproducibility. You can do that by leveraging existing data analytics solutions such as BigQuery, AlloyDB, Vertex Feature Store.
- **Adapter models:** The landscape of techniques like LoRA tuning for adapter models is constantly evolving. . You can leverage established data storage solutions (e.g. cloud storage) to manage and version these assets effectively.

Continuous integration of gen AI systems

In a continuous integration framework, every code change goes through automatic testing before merging to catch issues early. Here, unit and integration testing are key for quality and reliability. Unit tests act like a microscope, zooming in on individual code pieces, while integration testing verifies that different components work together.

The benefits of continuous integration in traditional software development are well-understood. Implementing a CI system helps to do the following:

1. **Ensure reliable, high-quality outputs:** Rigorous testing increases confidence in the system's performance and consistency.
2. **Catch bugs early:** Identifying issues through testing prevents them from causing bigger problems downstream. It also makes the system more robust and resilient to edge cases and unexpected inputs.
3. **Lower maintenance costs:** Well-documented test cases simplify troubleshooting and enable smoother modifications in the future, reducing overall maintenance efforts

These benefits are applicable to gen AI Systems as much as any software product. Continuous Integration should be applied to all elements of the system, including the prompt templates, the chain and chaining logic, and any embedding models and retrieval systems.

However, applying CI to gen AI comes with challenges:

1. Difficult to generate comprehensive test cases: The complex and open-ended nature of gen AI outputs makes it hard to define and create an exhaustive set of test cases that cover all possibilities.

2. Reproducibility issues: Achieving deterministic, reproducible results is tricky since generative models often have intrinsic randomness and variability in their outputs, even for identical inputs. This makes it harder to consistently test for specific expected behaviors.

These challenges are closely related to the broader question of how to evaluate gen AI systems. Many of the same techniques discussed in the Evaluation section above can also be applied to the development of CI systems for gen AI. This is an ongoing area of research, however, and more techniques will undoubtedly emerge in the near future.

Continuous delivery of gen AI systems

Once the code is merged, a continuous delivery process begins to move the built and tested code through environments that closely resemble production for further testing before the final deployment.

As mentioned in the "Develop and Experiment" segment, chain elements become one of the main components to deploy, as they fundamentally constitute the gen AI application serving users.

The delivery process of the gen AI application containing the chain may vary depending on the latency requirements and whether the use case is batch or online:

1. Batch use cases require deploying a batch process executed on a schedule in production. The delivery process should focus on testing the entire pipeline in integration in an environment close to production before deployment. As part of the testing process, developers can assert specific requirements around the throughput of the batch process itself and checking that all components of the application are functioning correctly (e.g., permissioning, infrastructure, code dependencies).

2. Online use cases require deploying an API, in this case, the application containing the chain, capable of responding to users at low latency. The delivery process should involve testing the API in integration in an environment close to production, with tests to assert that all components of the application are functioning correctly (e.g., permissioning, infrastructure, code dependencies). Non-functional requirements (e.g., scalability, reliability, performance) can be verified through a series of tests, including load tests.

Deployment of foundation models

Because foundation models are so large and complex, deployment and serving of these models raises a number of issues – most obviously, the compute and storage resources needed to run these massive models successfully. At a minimum, a foundation model deployment needs to include several key considerations: selecting and securing necessary compute resources, such as GPUs or TPUs; choosing appropriate data storage services like BigQuery or Google Cloud Storage that can scale to deal with the large datasets; and implementing model optimization or compression techniques.

Infrastructure validation

One technique that can be applied to address the resource requirements of gen AI systems is infrastructure validation. This refers to the introduction of an additional verification step, prior to deploying the training and serving systems, to check both the compatibility of the model with the defined serving configuration and the availability of the required hardware. There are a number of optional infrastructure validation layers that can perform some of these checks automatically. For instance, TFX¹⁹ has an infrastructure validation layer that checks

whether the model will run correctly on a specified hardware configuration, which can help catch configuration issues before deployment. Nevertheless, the availability of the required hardware still needs to be verified by hand by the engineer or the system administrator.

Compression and optimization

Another way of addressing infrastructure challenges is to optimize the model itself. Compressing and/or optimizing the model can often significantly reduce the storage and compute resources needed for training and serving, and in many cases can also decrease the serving latency.

Some techniques for model compression and optimization include quantization, distillation and model pruning. Quantization reduces the size and computational requirements of the model by converting its weights and activations from higher-precision floating-point numbers to lower-precision representations, such as 8-bit integers or 16-bit floating-point numbers. This can significantly reduce the memory footprint and computational overhead of the model. Model Pruning is a technique for eliminating unnecessary weight parameters or by selecting only important subnetworks within the model. This reduces model size while maintaining accuracy as high as possible. Finally, distillation trains a smaller model, using the responses generated by a larger LLM, to reproduce the output of the larger LLM for a specific domain. This can significantly reduce the amount of training data, compute, and storage resources needed for the application.

In certain situations, model distillation can also improve the performance of the model itself in addition to reducing resource requirements. This happens because the smaller model can combine the knowledge of the larger model with labeled data, which can help it to generalize better to new data on a limited use case. The process of distillation usually involves training a large foundational LLM (teacher model) and having it generate responses to certain tasks,

and then having the smaller LLM (student model) use a combination of the LLMs knowledge as well as task specific supervised dataset to learn. The size and complexity of the smaller LLM can be adjusted to achieve the desired trade-off between performance and resource requirements. A technique known as step-by-step distillation²⁰ has proven to achieve great results.

Deployment, packaging, and serving checklist

Following are the important steps to take when deploying a model on Vertex AI.

- Configure version control:** Implement version control practices for LLM deployments. This allows you to roll back to previous versions if necessary and track changes made to the model or deployment configuration.
- Optimize the model:** Perform any model optimization (distillation, quantization, pruning, etc.) before packaging or deploying the model.
- Containerize the model:** Package the trained LLM model into a container.
- Define target hardware requirements:** Ensure the target deployment environment meets the requirements for optimal performance of the LLM model, such as GPUs, as well as TPUs and other specialized hardware accelerators.
- Define model endpoint:** Define the endpoint configuration using Vertex AI's endpoint creation interface or the Vertex AI SDK. Specify the model container, input and output formats, and any additional configuration parameters.
- Allocate resources:** Allocate the appropriate compute resources for the endpoint based on the expected traffic and performance requirements.

- **Configure access control:** Set up access control mechanisms to restrict access to the endpoint based on authentication and authorization policies. This ensures that only authorized users or services can interact with the deployed LLM.
- **Create model endpoint:** Create a Vertex AI endpoint to deploy²¹ the LLM as a REST API service. This allows clients to send requests to the endpoint and receive responses from the LLM..
- **Configure monitoring and logging:** Establish monitoring and logging systems to track the endpoint's performance, resource utilization, and error logs.
- **Deploy custom integrations:** Integrate the LLM into custom applications or services using the model's SDK or APIs. This provides more flexibility for integrating the LLM into specific workflows or frameworks.
- **Deploy Real-time Applications:** For real-time applications, consider using Cloud Functions and Cloud Run in combination with LLMs hosted in Vertex AI to create a streaming pipeline that processes data and generates responses in real time.

Logging and monitoring

Monitoring gen AI applications and, as a result, their components, presents unique challenges that require additional techniques and approaches on top of those in traditional MLOps. The use of gen AI requires the chaining of components in order to produce results for practical applications. Additionally, to your application user, all the components are hidden. Therefore, the interface they have to your application is their input and the final output. This creates the need to log and monitor your application end-to-end: that is, logging and monitoring the input and output of your application overall as well as the input and output of every single component.

Logging is necessary for applying monitoring and debugging on your gen AI system in production. An input to the application triggers multiple components. Imagine the output to a given input is factually inaccurate. How can you find out which of the components are the ones that didn't perform well? To answer this question it is necessary to apply logging on the application level and at the component level. We need lineage in our logging for all components executed. For every component we need to log their inputs and outputs. We also need to be able to map those with any additional artifacts and parameters they depend on so we can easily analyze those inputs and outputs.

Monitoring can be applied to the overall gen AI application and to individual components. We prioritize monitoring at the application level. This is because if the application is performant and monitoring proves that, it implies that all components are also performant. You can also apply the same practices to each of the prompted model components to get more granular results and understanding of your application.

Skew detection in traditional ML systems refers to training-serving skew that occurs when the feature data distribution in production deviates from the feature data distribution observed during model training. In the case of Gen AI systems using pretrained models in components chained together to produce the output, we need to modify our approach. We can measure skew by comparing the distribution of the input data we used to evaluate our application (the test set as described under the Data Curation and Principles section above) and the distribution of the inputs to our application in production. Once the two distributions drift apart,further investigation is needed. The same process can be applied to the output data as well.

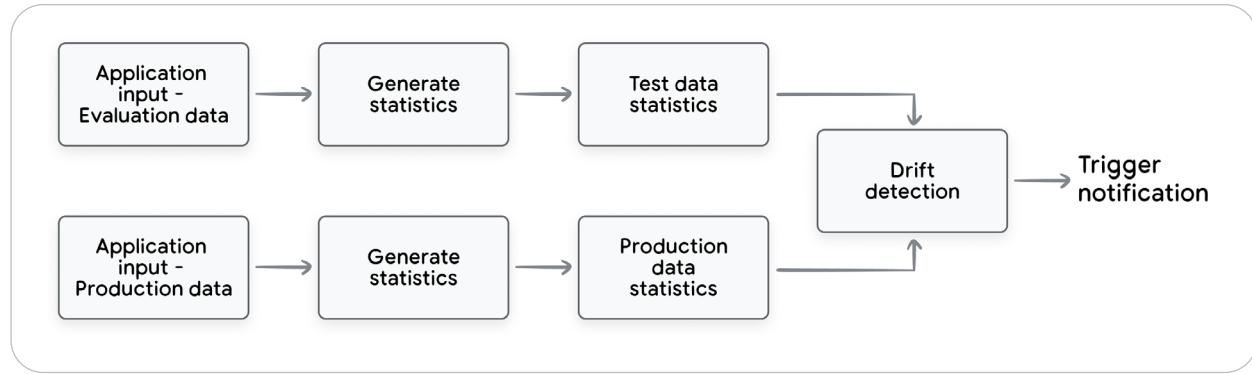


Figure 11. Drift/skew detection process overview

Like skew detection, the drift detection process checks for statistical differences between two datasets. However, instead of comparing evaluations and serving inputs, drift looks for changes in input data. This allows you to check how the inputs and therefore the behavior of your users changed over time. This is the same as traditional MLOps.

Given that the input to the application is typically text, there are a few approaches to measuring skew and drift. In general all the methods are trying to identify significant changes in production data, both textual (size of input) and conceptual (topics in input), when compared to the evaluation dataset. All these methods are looking for changes that could potentially indicate the application might not be prepared to successfully handle the nature of the new data that are now coming in. Some common approaches are calculating embeddings and distances, counting text length and number of tokens, and tracking vocabulary changes, new concepts and intents, prompts and topics in datasets, as well as statistical approaches such as least-squares density difference,²² maximum mean discrepancy (MMD),²³ learned kernel MMD,²⁴ or context-aware MMD.²⁵ As gen AI use cases are so diverse, it is often necessary to create additional custom metrics that better capture abnormal changes in your data.

With the rise of multimodal models, both dedicated image or video generation ones like Imagen and Veo, along with Gemini's ability to process and output other modalities than only text, there's a need for additional techniques to monitor and align these outputs to the creator's intents. In addition to prompt alignment - assuring that the output from the generative model matches the text description provided, there's also organizational policy alignment that can be tracked, logged, and monitored. As an analog to drift, subjective output can be measured by reviewing the original prompt and utilizing a multimodal generative AI's capabilities of assessment to determine whether the prompt and the output are similar, or have alignment. This can be done in multiple ways - a zero-shot, single prompt process to get a quick classification - or using the reasoning process of a generative model to decompose the prompt into related component parts, assess each one of these component part questions, and aggregate a score, yielding a more explainable or interpretable score. This process, often called "LLM as a Judge" or a generative Autorater, can bring some of the organization's potentially subjective policy guidance into assessable scores. Vertex AI's Generative AI Evaluation service provides primitives for constructing Custom Metrics and using these as autoraters.

Continuous evaluation is another common approach to GenAI application monitoring. In a continuous evaluation system, you capture the model's production output and run an evaluation task using that output, to keep track of the model's performance over time. One approach is collecting direct user feedback, such as ratings (for example thumbs up/down), which provides immediate insight into the perceived quality of outputs. In parallel, comparing model-generated responses against established ground truth, often collected through human assessment or as a result of an ensemble AI Model approach, allows for deeper analysis of performance. Ground truth metrics can be used to generate evaluation metrics as described in the Evaluation section. This process provides a view on how your evaluation metrics changed from when you developed your model to what you have in production today.

As with traditional monitoring in MLOps an alerting process should be deployed for notifying application owners when a drift, skew or performance decay from evaluation tasks is detected. This can help you promptly intervene and resolve issues. This is achieved by integrating alerting and notification tools into your monitoring process.

Monitoring expands beyond drift, skew and evaluation tasks. Monitoring in MLOps includes efficiency metrics like resources utilization and latency. Efficiency metrics are as relevant and important in gen AI as they are in any other AI application.

Vertex AI provides a set of tools that can help with monitoring. The Vertex AI Eval Service for gen AI²⁶ can be used for classification, summarization, question answering, and text generation tasks as well as Agent evals based on rubrics and more detailed trajectory evaluations. Vertex Pipelines can be used to allow the recurrent execution of evaluation jobs in production as well as running pipelines for skew and drift detection processes.

One form of logging commonly employed in application development is “tracing”, which sends out logged events to an aggregator from internal steps in a complex system - allowing detailed representations of a complicated application. This same approach can be utilized in developing agents, instrumenting steps by emitting traces, and ensuring that when you look at the logs you can understand which step for which agent for which user session. OpenTelemetry is the commonly adopted standard for traces with many possible aggregators, including Cloud Observability. The spec for traces has been changing recently to support larger payloads as needed to describe the behaviors of agents and LLMs.

Govern

In the context of MLOps governance encompasses all the practices, and policies that establish control, accountability, and transparency over the development, deployment, and ongoing management of machine learning (ML) models, including all the activities related to the code, data and models lifecycle.

As mentioned in the Develop & Experiment section the chain element and the relative components become a new type of assets that need to be governed over the full lifecycle from development to deployment, to monitoring.

The governance of the chain element lifecycle extends to lineage tracking practices as well.

While for predictive AI systems lineage focuses on tracking and understanding the complete journey of a machine learning model, in gen AI, lineage goes beyond the model artifact extending to all the components in the chain. This includes the data and models used and their lineage, the code involved and the relative evaluation data and metrics. This can help auditing, debugging and improvements of the models

Along with these new practices, existing MLOps and DevOps practices still apply to MLOps for gen AI:

1. The need to govern the data lifecycle; see “**Data Practices**”.
2. The need to govern the tuned model lifecycle; see “**Tuning and Training**”.
3. The need to govern the code lifecycle; see “**Deployment of GenAI System components**”.

The next segment will introduce a set of products that allow developers to perform governance of the data, model and code assets. We will discuss products like Google Cloud Dataplex, which centralizes the governance of model and data, Vertex ML Metadata and Vertex Experiment, which allows developers to register experiments, their metrics and artifacts.

Extend MLOps for gen AI to Agents

The development and operationalization of genAI systems involve intricate workflows, often chaining together multiple models, APIs, and data sources. A particularly compelling and increasingly important facet of this landscape is the rise of Agents. These intelligent systems, capable of interacting with their environment/tools and making decisions, represent a significant advancement in genAI capabilities. While we briefly introduced the concept of agents within the context of chains and augmentation, their unique characteristics and operational demands warrant a deeper dive. This section unfolds the specific MLOps considerations for building, deploying, and managing agent-based systems, addressing their distinct lifecycle, tooling requirements, and the crucial aspects of observability, safety, and continuous improvement. For the fundamental knowledge on Agents we recommend to read the Agent whitepaper by Google.⁹³

Agent Lifecycle

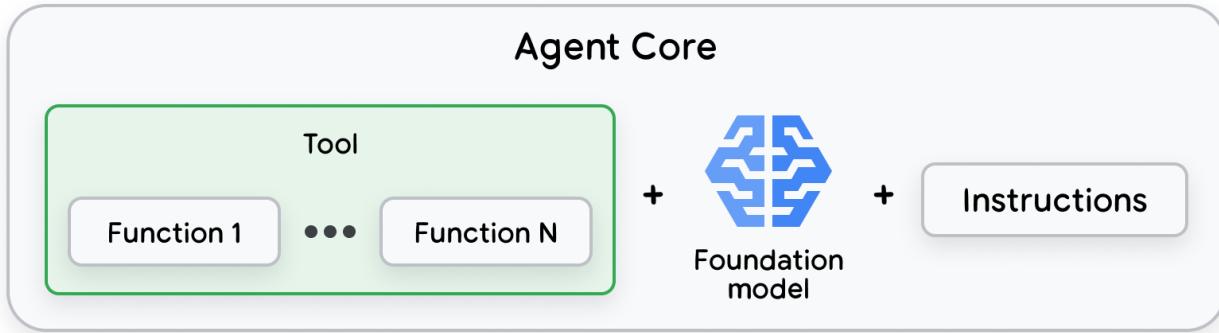


Figure 12. Agent Core: Tools, model, and instruction prompt

The core of an agent system comprises three fundamental elements: a Foundation Model, Instructions, and a Tool. The foundation model serves as the cognitive engine, providing the agent with reasoning and language processing capabilities. Instructions represent the guiding directives or goals that the agent is designed to achieve. These instructions could range from simple tasks to complex, multi-step objectives. The Tool, rather than being the actual executable code, consists of descriptions of the available functions and their required parameters. These descriptions provide the foundation model with the information it needs to reason about which tool is most appropriate for a given situation and how to use it. The foundation model analyzes the instructions and the descriptions of the available functions, determining which function to call and what parameter values to use. The actual execution of the chosen function is then handled separately (by the developer's code or, in some advanced systems, directly by the model itself). Together, these three components work in concert, allowing the agent to intelligently interpret instructions, leverage the Foundation Model's knowledge of available tools, and utilize the appropriate tool (via its description) to accomplish its objectives.

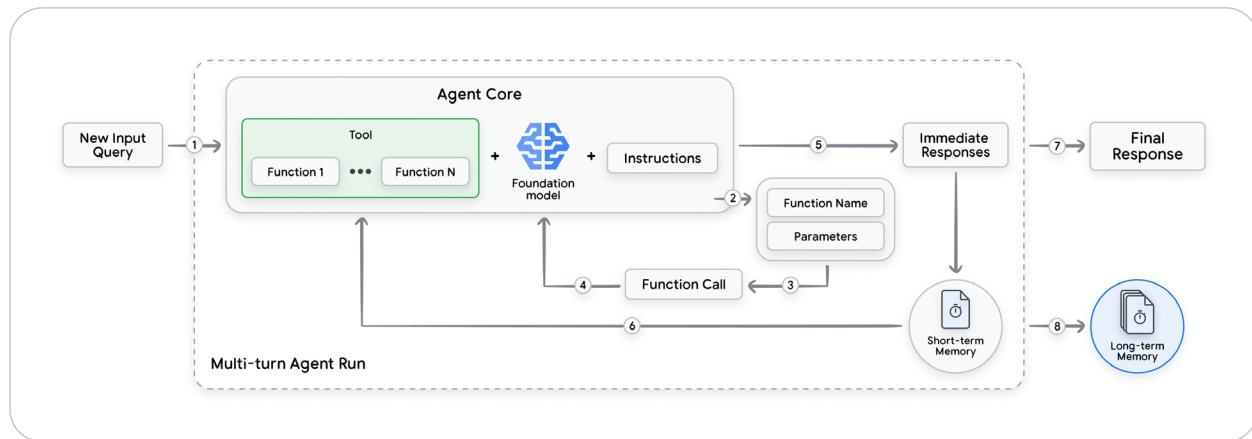


Figure 13. Multi-turn interaction of an agent with tools and memory until the final response

These core components – the Foundation Model, Instructions, and Tool – work in dynamic interplay throughout the agent's operational lifecycle. The following steps detail how these elements are orchestrated to process information, interact with the environment, and ultimately achieve the agent's objectives.

- 1. New Query:** The user initiates the interaction by providing a new query or question
- 2. Function Identification:** The foundation model, along with the available tools (e.g., APIs, databases, specialized software) and instructions, analyzes the query and determines if a function call is necessary. If so, it identifies the appropriate function name and the required parameters.
- 3. Function Call Preparation:** The foundation model generates a structured function call request, specifying the function name and the parameters to be passed.
- 4. Function Call Execution:** This step is performed by the developer's code (or, in advanced foundation models, automatically by the model). The code receives the function call request, executes the corresponding function (e.g., making an API call), and retrieves the result.

5. **Intermediate Response:** The result of the function execution (the data retrieved by the function) is sent back to the foundation model that produces an intermediate response.
6. **Iterative Context Update (Short-Term Memory):** The developer's code updates the conversation history (short-term memory) with the intermediate response and provides the updated context to the foundation model (returning to step 2).
7. **Final Response:** Once the foundation determines it has all the necessary information (or a maximum number of steps is reached to prevent infinite loops), it generates the final response to the user, incorporating the information gathered from all the function calls.

The preceding summary of the core components and operational lifecycle of an agent provides a foundation for examining the crucial considerations for productionizing these intelligent systems.

Tool Orchestration

As established, tools are essential for enabling generative AI agents to interact with the world and execute complex tasks. These tools vary significantly in implementation, accessibility, and capabilities. Understanding these distinctions is crucial for building effective and robust agents.

Tool Types & Environments

In a cloud environment, tools can be categorized based on origin and access method:

- **Code Functions:** Functions implemented directly within the agent's codebase, granting direct access to local resources and logic. These can be implemented in various programming languages (e.g., Python, Java). On Google Cloud, services like Artifact Registry (for storing code libraries) and Cloud Code (for repository management) can be used.
- **Private REST APIs:** APIs hosted within a Virtual Private Cloud (VPC), providing secure access to internal services and data. Google Cloud offers services like Cloud Run (for deploying containerized applications, including APIs), API Gateway (for managing and securing APIs), and Apigee API Management (for advanced API management).
- **Public REST APIs:** Publicly available APIs from third-party services, offering diverse functionalities accessible over the internet. A NAT Gateway can be used to provide secure outbound internet access for resources within a VPC without direct public exposure.

Both Code Functions and Private REST APIs enable secure interactions with internal databases, storage systems, other internal services, or even other agents within the organization's infrastructure. A comparison of the tool types follows:

Feature \ Tool Type	Code Functions	Private REST APIs	Public REST APIs
Latency	Very low	Medium	Potentially High
Implementation	Easy	Medium	Relatively Simple
Ownership/Control	Full	Full within the VPC	None
Authentication	None	Enabled	Requires API keys, OAuth
Shareability	Limited	Sharable within VPC	Accessible to all
Monitoring	Custom	Standard APIs	Limited
Version Control	Repositories (e.g. Git)	Repositories (e.g. Git)	API provider dependent
Security	Agent environment	Strong within VPC	Careful consideration
Internal Systems	Direct access	Access via network	Generally not direct

Table 2. Different types of tools and their feature comparison

As described, tools can be implemented in various ways, from code functions within the agent itself to APIs hosted in VPCs or publicly available services. Furthermore, these tools often reside in different locations within a cloud environment. Data sources, for example, are frequently managed within dedicated data environments, for example data lake or mesh projects, while APIs and agents used as tools might be deployed within application production environments. Code repositories, housing code functions and other artifacts, are typically maintained in central artifact government environments for auditability purpose. This distributed nature of tool deployment presents a significant challenge: how do we effectively manage and utilize this diverse and dispersed collection of tools in a production setting?

To address this challenge, and given the variety of tool types and their locations, a centralized catalog becomes essential for managing tools in a production environment. This is where the concept of a Tool Registry comes into play.

Tool Registry

A Tool Registry, a centralized catalog of all available tools, provides a standardized way to discover, access, and manage these essential components. This centralized approach offers several key advantages. First, it promotes **reusability** by enabling easy discovery and reuse of tools by different agents, significantly reducing development time and effort. Second, it enhances **shareability and visibility**, making tools readily available to all authorized developers, fostering collaboration and knowledge sharing across teams. Third, it strengthens **security and accessibility** by enforcing access control and ensuring that only authorized agents utilize specific tools. Fourth, the registry promotes **standardization** in tool implementation and usage, which improves code maintainability and agent interoperability. Fifth, it contributes to the **robustness** of the system through centralized management that allows for better monitoring, evaluation, and version control. Finally, the Tool Registry facilitates **auditability** by providing a clear record of tool usage, which is crucial for compliance and accountability.

To deliver these advantages, the Tool Registry incorporates several key features. The Tool Registry stores essential information about each tool, including its name, description, parameters, and output format, effectively serving as a comprehensive tool catalog. It also provides robust version control, tracking different versions of tools to ensure compatibility with various agents. The registry further simplifies the discovery process through effective search and discovery functionalities, allowing developers to easily find the tools they need. Finally, it manages permissions through access control mechanisms, ensuring that only authorized agents can access specific tools.

The Tool Registry mirrors the Model Registry in MLOps, highlighting its importance for operationalizing generative AI agents, just as Model Registries are essential for operationalizing machine learning models.

Tool Selection Strategies at Scale

The Tool Registry serves as a comprehensive catalog of all available tools within an enterprise, potentially housing hundreds or even thousands of options. However, simply providing a foundation model with access to this entire catalog isn't always the most effective approach. Presenting the model with an excessively long list of tools can actually be counterproductive. The model might become overwhelmed or confused, especially if tools have overlapping functionalities or similar descriptions. This can lead to incorrect tool selection, reduced performance, and unpredictable agent behavior. Imagine trying to choose the right tool from a massive, disorganized toolbox – it becomes a time-consuming and error-prone process. Therefore, strategic tool selection is crucial for optimizing agent performance and reliability.

To address this challenge, we can draw a parallel with microservices architecture. Just as in microservices, where each service is designed for specific tasks, we can view generative AI agents as specialized microservices. Each agent should ideally be equipped with only the subset of tools directly relevant to its specific responsibilities. This focused collection of tools, a carefully curated "Tool List," is a subset of the broader Tool Registry. Providing agents with a limited toolset offers several key benefits:

- Improves performance by reducing the search space for tool selection.
- Increases predictability, making the agent's behavior more understandable and easier to anticipate.
- Simplifies testing and debugging, as the agent's scope is more clearly defined.

- Enhances security by limiting the agent's potential impact on sensitive data or systems.

Three general strategies can be employed: providing full toolset access (creating a generalist agent), providing a limited toolset (creating a specialist agent), or dynamic tool selection. A generalist agent is granted access to the entire Tool Registry, relying on the foundation model's reasoning abilities to select the appropriate tool. While this approach offers greater flexibility, it can lead to the performance and predictability issues mentioned earlier. A specialist agent, conversely, is equipped with a carefully curated Tool List containing only the tools necessary for its specific task. This strategy promotes performance, predictability, and security, but it requires more upfront design effort to define each agent's toolset.

The third strategy, dynamic tool selection, represents a more emergent approach. In this model, the agent doesn't have a predefined tool list. Instead, for each incoming task, the agent queries the Tool Registry to identify and select the most relevant tools. This dynamic approach allows the agent to adapt to a wider range of tasks without requiring a massive, pre-configured toolset. However, this increased flexibility comes with its own set of challenges. Because the tool selection process occurs at runtime and is dependent on the specific task, predicting the agent's behavior and ensuring consistent performance can be more difficult.

The optimal strategy, whether to create generalist, specialist, or dynamically selecting agents, depends on the specific needs and context of the enterprise. Factors to consider include the total number of tools in the registry, the complexity of the tasks the agents are designed to perform, the desired level of control over agent behavior, and the available resources for agent design and maintenance. Future work will involve releasing experimental results comparing these three strategies quantitatively to provide clearer guidance. In particular, the dynamic selection strategy, being relatively new, requires further research to develop robust testing and evaluation methodologies.

Agent Evaluation & Optimization

As we have discussed, agents are a new and emerging type of AI system with the potential to revolutionize human-computer interaction. A crucial aspect of realizing this potential lies in effectively evaluating and optimizing agent performance. While previous sections of this white paper have addressed general GenAI evaluation, this section focuses specifically on agent-centric evaluation and optimization techniques.

Agent evaluation can be structured into five key stages:

- 1. Tool Unit Testing and Refinement:** During development, each tool available to the agent undergoes rigorous unit testing. This process validates the tool's functionality and ensures it performs as expected. Crucially, this stage also involves refining the descriptions of the available functions and their parameters, which are what the foundation model uses for reasoning and selection. Thorough unit testing and well-defined tool descriptions are foundational for reliable agent behavior.
- 2. Evaluation Dataset Creation:** A representative evaluation dataset of potential agent (multi-turn) scenarios is constructed. This dataset can be created manually, by carefully crafting specific interaction sequences, or automatically, by recording real user interactions with the agent and then incorporating these recorded sessions into the evaluation set. This automatic approach allows for continuous enrichment of the evaluation dataset with real-world usage patterns.
- 3. Tool Selection Evaluation:** Tool selection evaluation focuses on assessing the agent's proficiency in choosing and utilizing the correct tools for given tasks. For an individual tool, this involves measuring the percentage of successful tool selections, ensuring the accuracy of parameter structuring and population for function calls, and confirming the appropriate handling of scenarios where no tool selection is necessary. Collectively we use a trajectory evaluation, where we compare the set of steps we expect the agent to

take, including the tools the agent should use, against the steps an agent actually took. From a trajectory evaluation we can identify when an agent hallucinates (by not looking up knowledge) or uses the wrong tool or getting stuck in a cul-de-sac. These metrics collectively provide a comprehensive view of the agent's effectiveness in leveraging its available tools.

- 4. Reasoning and Groundedness Evaluation:** Beyond the standard gen AI end-to-end evaluation (i.e., assessing the correctness and groundedness of the final answer given an input), agent evaluation also involves assessing the reasoning process. This includes evaluating how the agent handles situations with missing data, how it recovers from errors, and its ability to maintain context across multiple turns. This stage examines the agent's "cognitive" abilities and its capacity to handle complex, multi-step interactions.
- 5. Operational Metric Evaluation:** Finally, the agent's performance is evaluated from an operational perspective. This involves measuring metrics such as latency (the time taken to generate a response or interact with specific tools) and cost (the computational resources consumed). These metrics are essential for determining the practical viability and scalability of the agent in a production environment.

By systematically addressing these five stages, developers can gain a comprehensive understanding of their agent's strengths and weaknesses, enabling targeted optimization and improvement. This multi-faceted evaluation approach is critical for building robust and reliable agent-based applications.

Agent optimization is an iterative process that focuses on refining the agent's components and configuration to enhance its performance and effectiveness. One key aspect of optimization involves selecting the most appropriate set of tools for the agent's tasks and ensuring that the function and parameter definitions within those tools are clear, concise, and well-structured. This clarity enables the foundation model to effectively reason about and utilize the available tools. Another crucial factor is selecting the right foundation model

for the agent's specific needs. Different foundation models may exhibit varying levels of proficiency in reasoning and tool selection, so choosing a model that excels in these areas is essential. Finally, optimizing the instruction prompt provided to the agent can significantly improve its performance. A well-crafted prompt provides clear guidance, sets expectations, and helps the agent effectively navigate complex tasks. By iteratively refining these three elements—tools, foundation model, and instruction prompt—developers can significantly enhance the agent's capabilities and overall performance.

Observability and Memory

In the realm of agent-based systems, observability and explainability are paramount. These qualities are essential for understanding agent behavior, building trust, and ensuring that agents operate as intended. Observability refers to the ability to gain insights into an agent's internal workings and its interactions with the environment. Explainability, on the other hand, focuses on understanding the why behind an agent's decisions—why it selected a particular tool or took a specific action.

Achieving observability and explainability requires careful consideration of the agent's memory and decision-making processes. The agent's short-term and long-term memory play a crucial role in providing context and enabling traceability. Being able to see events, organized on a timeline, as related to the steps an agent takes is paramount.

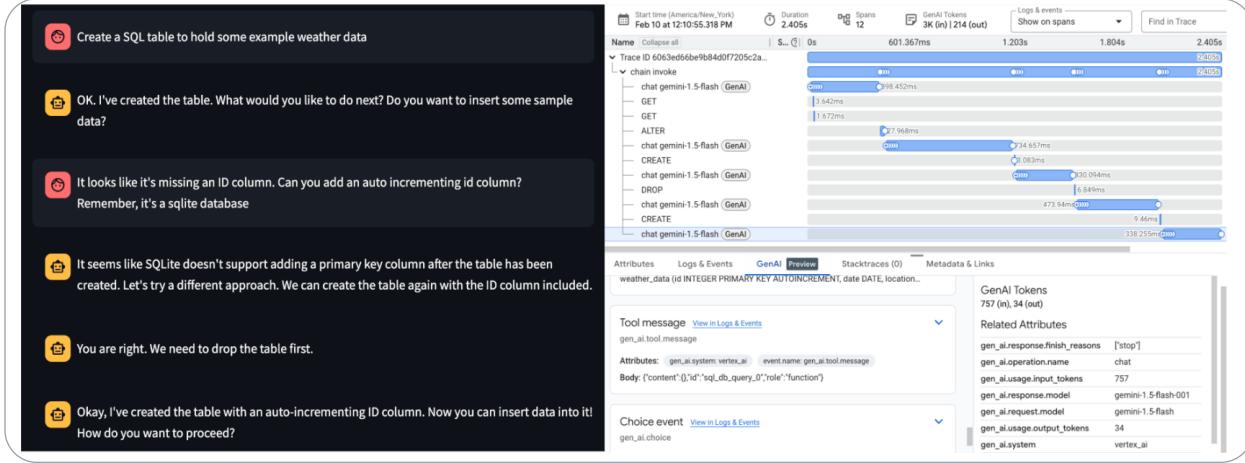


Figure 14. Example of cloud trace for agents

Short-term memory (or conversation history) stores the ongoing conversation within a single user session. This includes the user's queries, the model's function calls, and the responses from those function calls. This context is essential for the model to understand follow-up questions and maintain coherence throughout the interaction. By examining the short-term memory, we can trace the sequence of interactions and understand how the agent arrived at a particular state or decision.

Long-term memory, on the other hand, stores information about past user interactions across multiple sessions. This allows the agent to learn user preferences, provide personalized recommendations, and offer more efficient service over time. Long-term memory provides valuable insights into the agent's learning process and how its behavior evolves over time.

Several implementation options exist for both short-term and long-term memory in agent productionization, each with its own trade-offs:

Short-Term Memory Implementation Options:

- **Logs:** Storing interaction history as plain text logs can be sufficient for simple applications.
- **Cloud Storage/Database:** Offers more structured storage for complex applications.
- **API Session:** Managing conversation history on the client-side reduces server-side storage needs.
- **Combination:** A hybrid approach combining different mechanisms for optimized performance.

Long-Term Memory Implementation Options:

- **Vector Databases:** Well-suited for storing and retrieving information based on semantic similarity.
- **Metadata Storage/Graphs:** Used to store session IDs, timestamps, and other relevant metadata.
- **Cloud Storage/Databases:** Provides a full record of all interactions.
- **Combination:** Combining different mechanisms for efficient storage and retrieval.

The choice of implementation depends on the specific needs of the application and the complexity of the agent's interactions.

By effectively leveraging both short-term and long-term memory, developers can gain a deeper understanding of the agent's behavior, trace its decision-making process, and identify areas for improvement. This granular level of observability and explainability is crucial for building trust in agent-based systems and ensuring their responsible deployment.

Deploying an Agent to Production

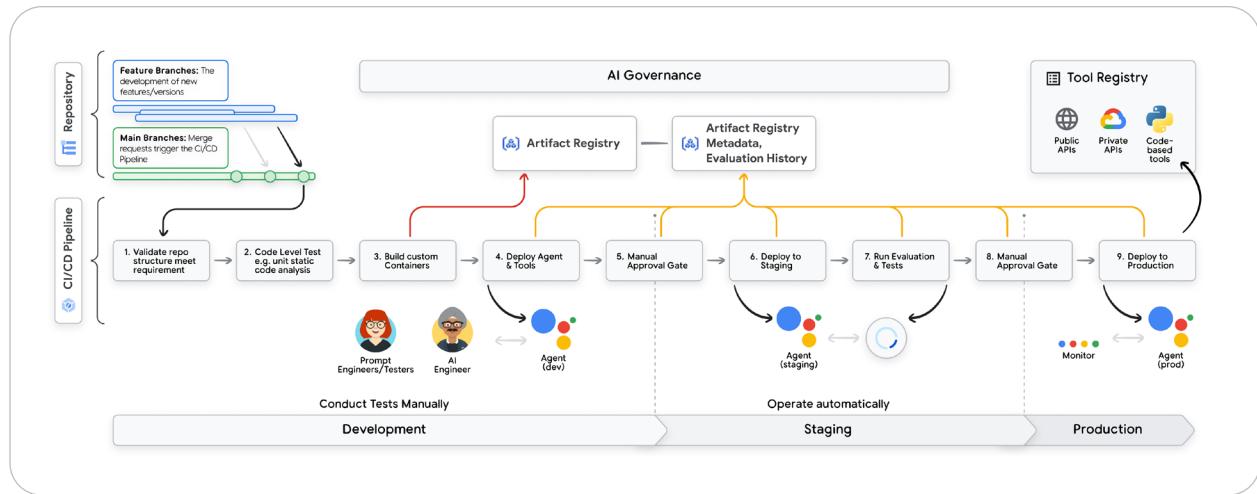


Figure 15. Agent and tool productionization using CI/CD pipelines

Deploying an agent to production requires a robust and automated process to ensure consistency, reliability, and scalability. Building upon a standardized repository structure, we can implement CI/CD pipelines that automate the tool registration process and streamline the deployment workflow. Such a pipeline typically involves several key stages. Initially, the pipeline validates that the repository structure adheres to defined standards, including the presence of required directories and files. It then executes code-level tests, such as unit tests and static code analysis, to verify code quality and correctness. If necessary, the pipeline builds custom container images for the tool or agent, ensuring consistent deployment across environments.

The pipeline then deploys the agent and its associated tools to a development environment for early testing and integration. This deployment can leverage infrastructure-as-code (IaC) scripts within the agent's repository for fine-grained control over the environment.

or utilize a pre-defined, standardized infrastructure configuration for consistency and security. A manual approval gate at this stage allows for human oversight and verification before proceeding.

After approval, the pipeline deploys the agent and tools to a staging environment that closely mirrors production, enabling more rigorous testing under realistic conditions. Here, human testers can engage with the agent at scale, further validating its performance and potentially augmenting the evaluation data with real-world usage patterns. Automated evaluation scripts and various tests, including integration and stress tests, are also executed, and the results are stored centrally for analysis and governance. Another manual approval gate follows, allowing for final verification based on the comprehensive testing results.

Once approved, the pipeline deploys the agent to the production environment. Crucially, during this process, the tool registration occurs implicitly. Metadata extracted from the repository structure is automatically registered in the Tool Registry, eliminating manual effort and ensuring consistency. Post-deployment, the agent's performance is continuously monitored using predefined scripts and configurations, enabling rapid detection and resolution of any issues.

This entire deployment lifecycle, from initial development to production deployment and monitoring, forms a continuous loop of improvement. Insights gained from monitoring and evaluation inform further refinements to the agent's components, tools, and configuration, leading to iterative enhancements and updates that are then redeployed through the same pipeline. This iterative approach ensures that agents remain adaptable, effective, and aligned with evolving needs and objectives.

This automated approach, with its standardized structure, rigorous testing, and implicit tool registration, ensures a robust and efficient deployment process. The CI/CD pipeline, coupled with the centralized Tool Registry, provides a solid foundation for deploying and managing agents in production, enabling organizations to leverage the full potential of these intelligent systems.

Operations: People & Processes

Productionizing machine learning or gen AI solutions, requires more than just technology and services. It demands a well-orchestrated interplay of diverse individuals, each contributing specialized skills and expertise, working together seamlessly under standardized processes. Like a well-oiled machine, the effective operation of ML solutions hinges on the coordinated efforts of various personas, forming a cohesive and efficient workflow.

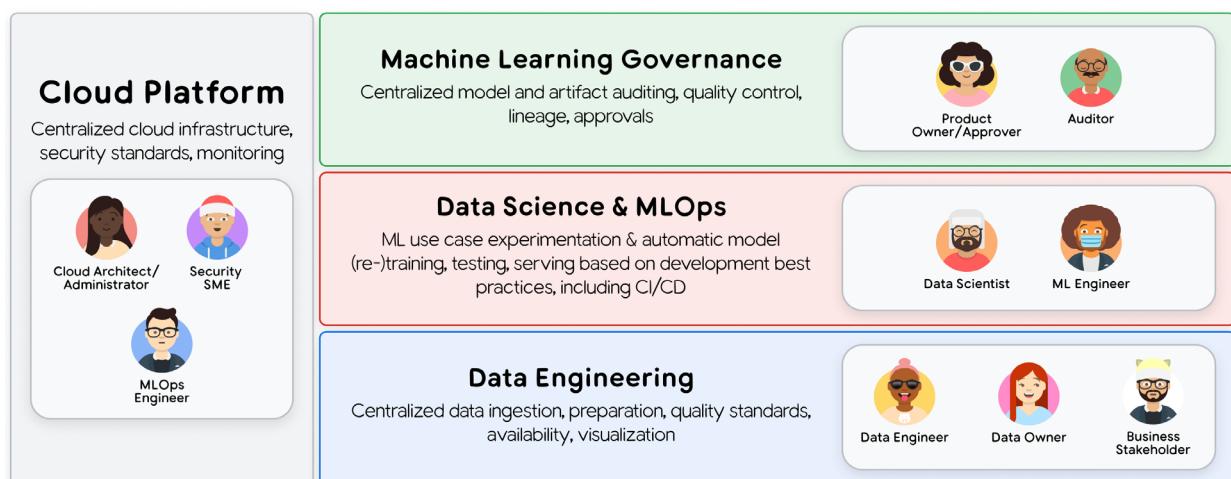


Figure 16. Traditional MLOps landscape of people, process, and environments

In the traditional predictive ML landscape, a typical setup involves several interconnected teams, each with distinct roles and responsibilities:

- **Cloud Platform Team:** This foundational team comprises cloud architects, administrators, and security specialists. They are responsible for providing and managing the essential cloud infrastructure, ensuring security, defining access control, and overseeing the overall cloud strategy.
- **Data Engineering Team:** This team focuses on building and maintaining the data pipelines that fuel the AI models and solutions. The data engineers handle data ingestion, preprocessing, and preparation, ensuring data quality, while data owners are responsible for the data accessibility for data scientists and other business stakeholders.
- **Data Science and MLOps Team:** This team includes data scientists who experiment with and train models, as well as ML engineers who collaborate with data scientists to automate the ML pipeline, encompassing data preprocessing, model training, and post-processing at scale. They leverage CI/CD pipelines for seamless model deployment and management. Within this team, the MLOps Engineer plays a vital role in building and maintaining the infrastructure (IAM roles and networking layer) for the Cloud Platform team that supports and standardizes these pipelines.
- **Machine Learning Governance:** This centralized function oversees the entire ML lifecycle, acting as a repository for model metadata, development artifacts, and performance metrics. It ensures transparency, accountability, and compliance in AI development and deployment. This is the single point of truth for product owner and auditors to assess the ML solutions and models respectively.

This collaborative structure ensures a smooth and controlled path from initial experimentation to production-ready ML solutions. However, gen AI introduces a new layer of complexity to the enterprise organizational landscape. This new layer, focused on building and deploying **GenAI applications**, necessitates specialized roles and expertise to effectively harness the power of these advanced models.

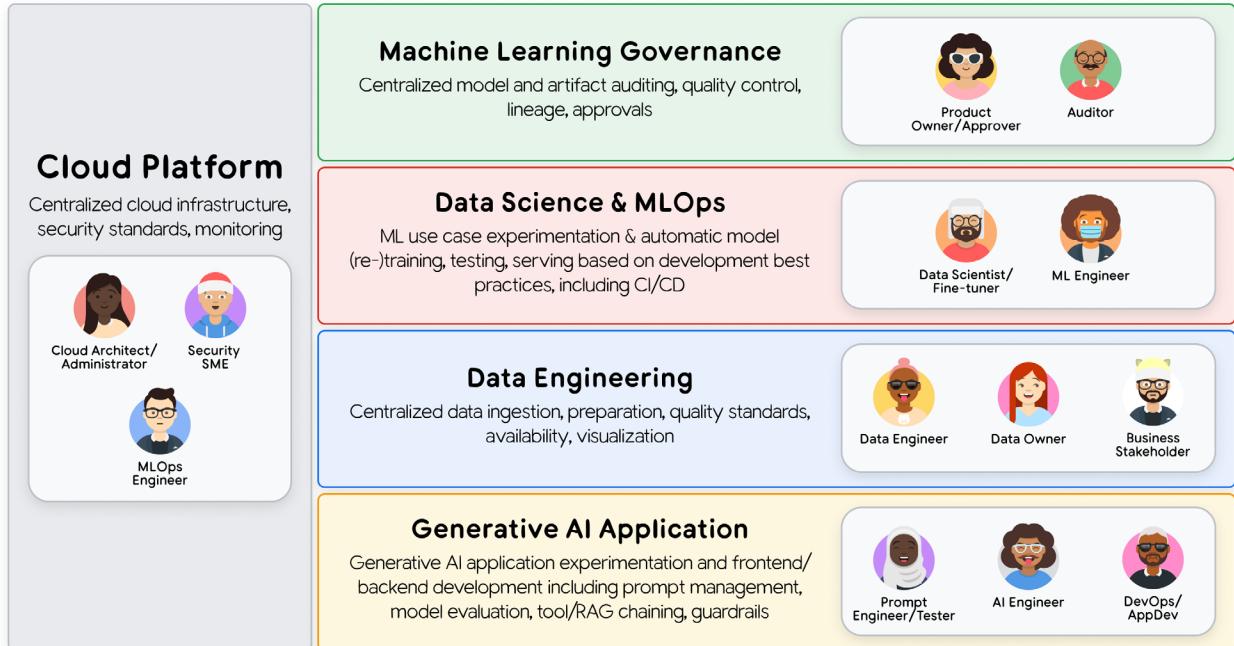


Figure 17. From MLOps to GenAI application development landscape of people, process, and environments

- **Prompt Engineers:** These individuals wear two hats, blending technical expertise with domain knowledge. First, they are skilled in the art of crafting and refining prompts, understanding how different inputs influence model behavior and optimizing prompt design for specific applications. Second, they often possess deep subject matter expertise, allowing them to define the right questions and expected answers for eliciting desired knowledge from the model. This dual role enables them to bridge the gap between the technical intricacies of gen AI models and the specific needs of various domains. We have seen data scientists, ML engineers, and even business analysts with deep domain knowledge successfully transition into prompt engineering roles.
- **AI Engineers:** AI engineers are responsible for scaling the use of generative models and transitioning gen AI solutions into production. They focus on building robust backend systems that incorporate functionalities like evaluation at scale, guardrails to ensure

responsible AI, and seamless integration with data sources (including RAG, data lakes, and tools). Their expertise lies in creating a stable and scalable foundation for gen AI applications.

- **DevOps/App Developers:** These individuals leverage their software engineering skills to develop the front-end components of gen AI applications. They work closely with AI engineers to integrate the backend capabilities, ensuring a user-friendly and efficient interface for interacting with GenAI models. Their focus is on delivering a seamless and engaging user experience.

The introduction of these specialized roles highlights the growing demand for expertise in applying and deploying gen AI solutions effectively. Prompt engineers, AI engineers, and DevOps/App developers work in concert, forming a new operational layer dedicated to building and deploying the next generation of AI-powered solutions.

It's important to acknowledge that the scale and structure of an organization will influence the specific roles and responsibilities within these teams. In smaller organizations, individuals may wear multiple hats, fulfilling the duties of several personas. As companies grow and their GenAI initiatives mature, more specialized roles and dedicated teams may emerge to handle the increased complexity.

The diagram below provides a more detailed illustration of the key operations and interactions within this organizational structure. A deeper dive into this diagram can provide further insights into the specific processes and responsibilities associated with each role.

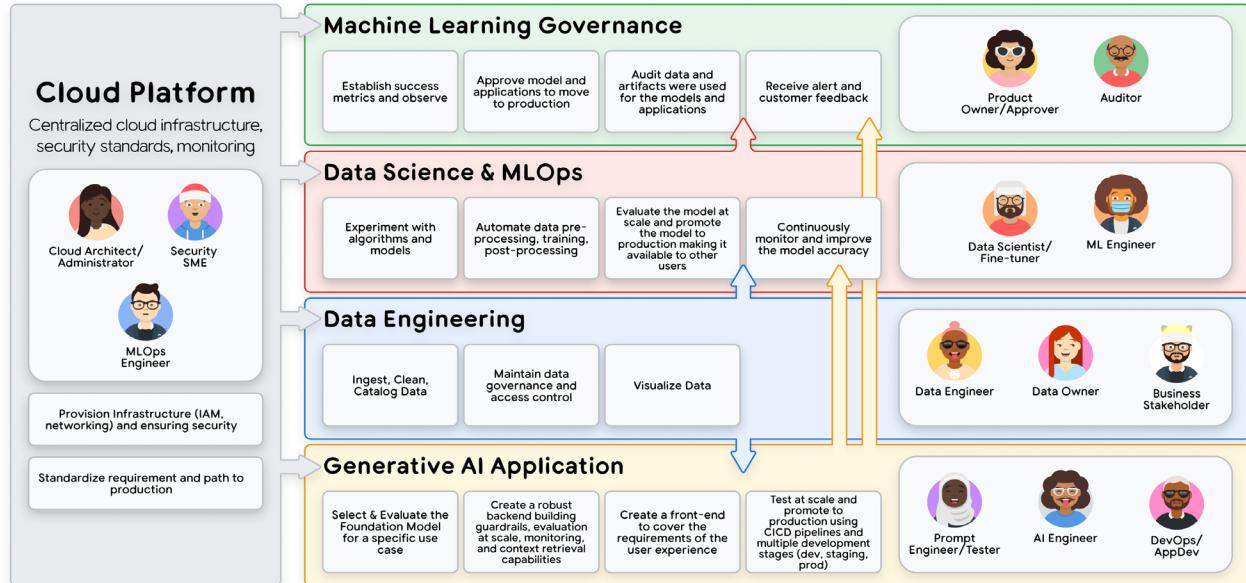


Figure 18. How multiple team collaborate to operationalize both models and GenAI applications

By effectively coordinating these diverse roles and streamlining their interactions, organizations can establish a robust operational foundation for both traditional ML and generative AI initiatives. This collaborative approach, combined with well-defined processes, is essential for successfully navigating the complexities of AI productionization and maximizing its potential benefits.

The role of an AI platform for gen AI operations

Alongside the explosion of both predictive and gen AI applications, AI platforms, like Vertex AI,¹¹ have emerged as indispensable tools for organizations seeking to leverage the power of Artificial Intelligence (AI). These comprehensive platforms provide a unified environment that streamlines the entire AI lifecycle, from data preparation and model training to deployment, automation, continuous integration/continuous delivery (CI/CD), governance, and monitoring.

At the heart of an AI platform lies its ability to support diverse AI development needs. Data scientists and model builders need to create, evaluate, and tune predictive and generative AI tools. Whether you seek to utilize pre-trained AI solutions, adapt existing models through tuning or transfer learning, or embark on training your own large models, AI platforms provide the infrastructure and tools necessary to support these journeys. Application developers and agent builders also need centralized and scalable access to models and services which support them, like RAG, and observability support to bring MLOps to gen AI. The advent of these platforms has revolutionized the way organizations approach AI, enabling them to productionize AI applications in a secure, enterprise-ready, responsible, controlled and scalable manner. These platforms accelerate innovation as well as foster reproducibility and collaboration while reducing costs and maximizing Return on Investment (ROI).

The new gen AI paradigm discussed in prior sections demands a robust and reliable AI platform that can seamlessly integrate and orchestrate a wide range of functionalities. These functionalities include model tuning for specific tasks; leveraging paradigms like retrieval augmented generation³ (RAG) to connect to internal and external data sources; and pre-training or instruction fine-tuning large models from scratch. Complex applications

also often require chaining with other models, such as classifiers to route inputs to the appropriate LLM/ML model, extraction of customer information from a knowledge base, inclusion of safety checks, or even creation of caching systems for cost optimization.

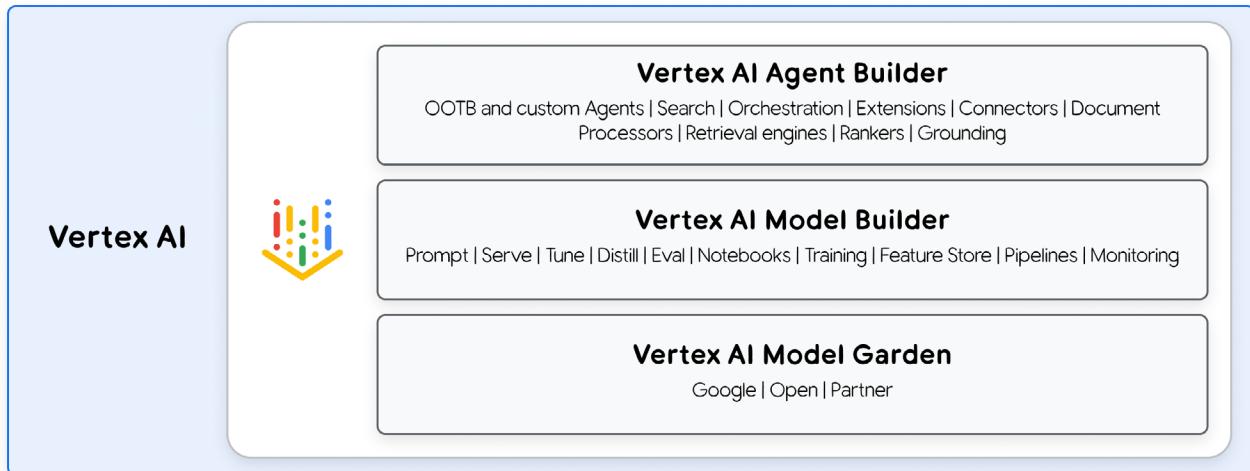


Figure 19. Key components of Vertex AI for gen AI

Key components of Vertex AI for gen AI

Vertex AI eliminates the complexities of managing the entire infrastructure required for AI development and deployment. Instead, Vertex AI offers a user-centric approach, providing on-demand access to the needed resources. This flexibility empowers organizations to focus on innovation and collaboration, rather than infrastructure management, and up-front hardware purchase. The features of Vertex AI that support gen AI development can be grouped into eight areas.

Discover: Vertex Model Garden

As discussed before, there is already a wide variety of available foundation models, trained on a broad range of datasets, and the cost of training a new foundation model can be prohibitive. Thus it often makes sense for companies to adapt existing foundation models rather than creating their own from scratch. As a result, a platform facilitating seamless discovery and integration of diverse model types is critical.

Vertex AI Model Garden¹ supports these needs, offering a curated collection of over 150 Machine Learning and gen AI models from Google, Google partners, and the open-source community. It simplifies the discovery, customization, and deployment of both Google's proprietary foundational models and diverse open-source models across a vast spectrum of modalities, tasks, and features. This comprehensive repository permits developers to leverage the collective research on artificial intelligence models within a single streamlined environment.

Model Garden encompasses a diverse range of modalities such as Language, Vision, Tabular, Document, Speech, Video, and Multimodal data. This broad coverage enables developers to tackle a multitude of tasks, including generation, classification, regression, extraction, recognition, segmentation, tracking, translation, and embedding. Model Garden houses Google's proprietary and foundational models (like Gemini,²⁷ PaLM 2,²⁸ Imagen²⁹) alongside numerous popular open source and third-party partner models like Llama 3,³⁰ T5 Flan,³¹ BERT,³² Stable Diffusion,³³ Claude 3 (Anthropic),³⁴ and Mistral AI.³⁵ Additionally, it offers task-specific models for occupancy analysis, watermark detection, text-moderation, text-to-video, hand-gesture recognition, product identification, and tag recognition, among others. Every model³⁶ in Vertex Model Garden has a model card which includes a description of the model, the main use cases that can cover, and the option (if available) to tune the model or deploy it directly.

Model Garden fosters experimentation by facilitating access to Google's proprietary foundational models through the Vertex AI Studio UI,³⁷ a playground where you can play around with prompts, models, and open-source models using provided Colab notebooks. One-click deployment is available for some external models, and there are more than 40 models available for fine-tuning for specific needs. Furthermore, the platform allows users to leverage technologies like vLLM³⁸ and quantization techniques for optimizing deployments for efficiency and reduced costs. We present below an overview of some of the models in Model Garden. For an up-to-date list, please visit.³⁶

Model Type	Description	Details
First-party models	<p>Foundation models</p> <p>Leverage multimodal models from Google across vision, dialog, code generation, and code completion.</p>	<p>Gemini³⁹ family of models supporting text, image, audio, video streaming, giant cacheable context windows, grounding and function calling Imagen for text-to-image⁴¹</p> <p>Chip for high-quality speech-to-text⁴³</p> <p>Veo for high quality text-to-video⁴³</p> <p>Codey for code generation and completion⁴²</p>
First-party models	<p>Pre-trained APIs</p> <p>Build and deploy AI applications faster with our pre-trained APIs powered by the best Google AI research and technology.</p>	<p>Text-to-Speech⁴⁴</p> <p>Natural Language processing⁴⁵</p> <p>Translation⁴⁶</p> <p>Vision⁴⁷</p>
Open models	<p>Open source models</p> <p>Access a wide variety of enterprise-ready open source models</p>	<p>Google's Gemma,⁴⁸ family of models including PaliGemma,¹⁶ CodeGemma⁴⁹, etc</p> <p>Meta's Llama³⁰</p> <p>TII's Falcon⁵⁰</p> <p>Mistral AI⁵¹</p> <p>BERT,³² T-5 FLAN,³¹ ViT,⁵² EfficientNet⁵³</p>
Third-party models	<p>Third-party models</p> <p>Model Garden will support third-party models from partners with foundation models.</p>	<p>Anthropic's Claude 3 Haiku, Sonnet and Opus^{54,55}</p>

Table 3. An overview of some of the models in Model Garden [Last Updated: March 18th, 2024]

Prototype: Vertex AI Studio & Notebooks

Rapid development and prototyping capabilities are also essential for developing gen AI applications. Vertex AI prioritizes inclusivity and flexibility in its development environments, catering to a wide range of developer preferences and proficiency levels. This platform provides options for both console-driven and programmatic development workflows. Users can leverage the intuitive web interface for end-to-end application creation or utilize various APIs for deeper customization and control. These include the REST API⁵⁶ and dedicated SDKs for Python,⁵⁷ NodeJS⁵⁸ and Java,⁵⁹ ensuring compatibility with diverse programming languages and ecosystems. Developers can choose to use the tools and IDEs of their choice for interacting with the platform, or take advantage of Vertex-native tools like Vertex Colab Enterprise or Vertex Workbench to explore and experiment with code within familiar notebook environments.

Vertex AI Studio⁶⁰ provides a unified console-driven entry point to access and leverage the full spectrum of Vertex AI's gen AI services. It facilitates exploration and experimentation with various Google first party foundation models (for example, PaLM 2, Gemini, Codey, Imagen, and Universal Speech Model). Additionally, it offers prompt examples and functionalities for testing distinct prompts and models with diverse parameters. It's also possible to adapt existing models through various techniques like supervised fine-tuning (SFT), reinforcement learning tuning techniques, and Distillation, and deploy gen AI applications in just a few clicks. Vertex AI Studio considerably simplifies and democratizes gen AI adoption, catering to a variety of users, from business analysts to machine learning engineers. You can see the homepage of Vertex AI Studio in Figure 13.

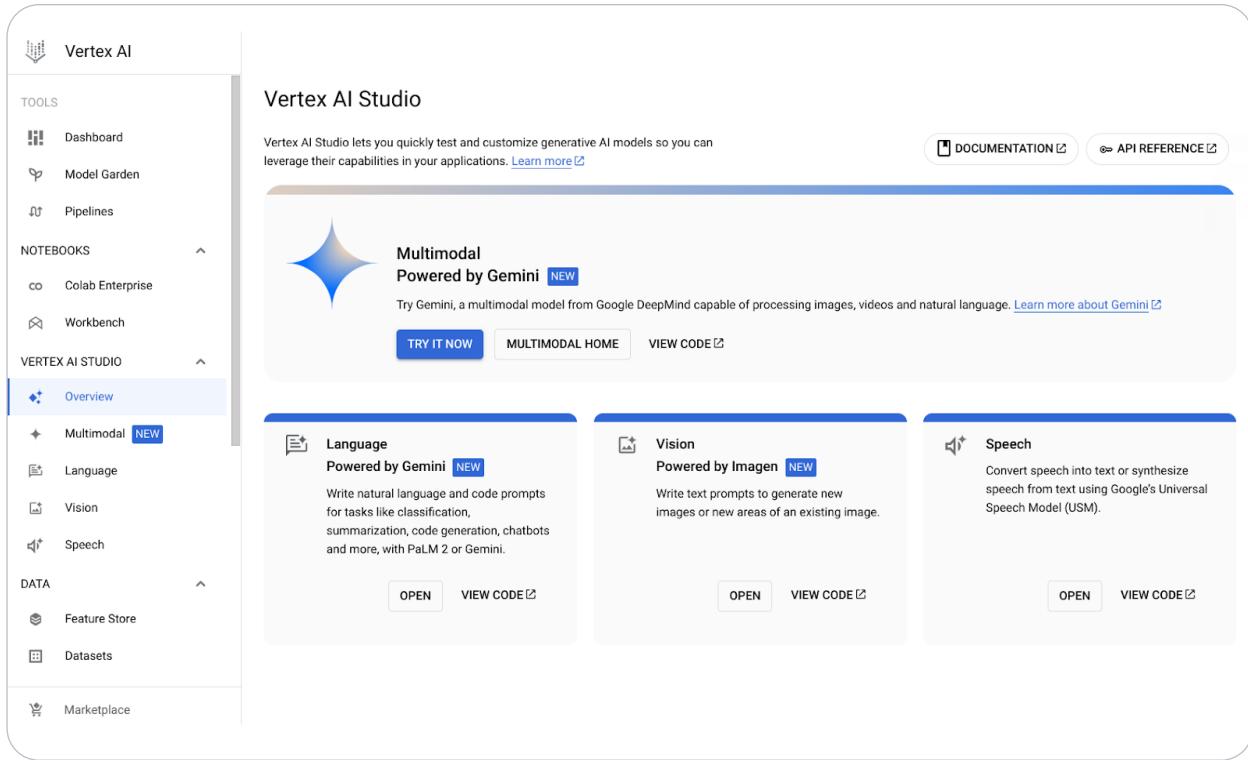


Figure 20. Vertex AI Studio - Homepage

Customize: Vertex AI training & tuning

While prompt engineering and augmentation are sufficient for some gen AI use cases, other cases require training, tuning and adapting the models to get the best results. Vertex AI provides a comprehensive platform for training and adapting LLMs, supporting a range of techniques and approaches from prompt engineering to training models from scratch.

Train

For full-scale LLM training, TPUs and GPUs are vital because of their superior processing power and memory capacity compared to CPUs. GPUs excel at parallel processing, enabling faster model training. TPUs, specifically designed for machine learning tasks, offer even faster processing and higher energy efficiency. This makes them ideal for large-scale, complex models. Google Cloud provides a range of offerings to support LLM training, including TPU VMs with various configurations, pre-configured AI platforms like Vertex AI, and dedicated resources like Cloud TPU Pods for scaling up training. These offerings allow users to choose the right infrastructure for their needs, accelerating LLM development and enabling cutting-edge research and applications.

Tune

Vertex AI also provides a comprehensive solution for adapting pre-trained LLMs. It supports a spectrum of techniques from a non-technical prompt engineering playground at inference time, to data-driven approaches involving tuning, reinforcement learning and distillation methods during the development or adaptation phase. The following five techniques – many of which are unique to Vertex AI – enable users to explore and implement them effectively. This applies to both proprietary and open-source LLMs, allowing you to achieve superior results while optimizing for costs and latency requirements.

- **Prompt engineering**⁶¹ leverages carefully crafted natural language prompts, potentially chained and enriched with external knowledge and examples, to nudge the LLM towards desired outputs without necessitating further training. Vertex AI through Vertex AI Studio offers a dedicated playground for crafting, testing, comparing and managing diverse prompts and techniques. Users can access various pre-built prompt templates within the platform and leverage public prompting guidelines⁶² for Google's proprietary large models.

- **Supervised fine-tuning (SFT)**⁶³ on Vertex AI facilitates model adaptation by leveraging a set of labeled examples (even a few hundred is enough) to tune a model on specific tasks and contexts within domain-specific datasets. The required examples resemble the one-shot example structure employed in the construction of a prompt. This effectively extends the few-shot learning approach for enhanced optimization. This focused tuning enables the model to encode additional parameters in the model necessary for mimicking desired behaviors such as improved complex prompt comprehension, adaptation to specific output formats, correcting errors, and learning new tasks. The SFT tuning approach on Vertex AI, minimizes computational overhead and time while yielding an updated model that integrates the newly acquired parameters with the original model's core parameters.
- **Reinforcement learning with human feedback (RLHF)**,⁶⁴ available on Vertex AI for foundational models like PaLM 2, and open-source models like T5 (s-xxl) and Llama2, leverages human feedback to train large models to align with human preferences. This technique is well-suited in complex tasks involving preference modeling and optimizes LLMs on intricate, sequence-level objectives not easily addressed by traditional supervised fine-tuning. The process involves first training a reward model using a human preference dataset, then utilizing it to score the output from the LLM, and finally applying reinforcement learning to optimize the LLM. This approach is recognized as a key driver of success in conversational large language models.
- **Distillation step-by-step**²⁰ is an advanced distillation technique transferring knowledge from a significantly larger model (known as teacher model) to a smaller task-specific model (known as student model), preserving important information while reducing model size. Step-by-Step Distillation²⁰ surpasses common techniques by requiring significantly less data. This method, accessible on Vertex AI,⁶⁵ significantly reduces inference costs and latencies while minimizing performance impact in the resulting smaller LLM.⁶⁶

Orchestrate

Any training or tuning job you run can be orchestrated and then operationalized using Vertex Pipelines,¹³ a service that aims to simplify and automate the deployment, management, and scaling of your ML workflows.

It provides a platform for building, orchestrating, scheduling and monitoring complex and custom ML pipelines, enabling you to efficiently translate your models from prototypes to production.

Vertex Pipelines is also the platform behind all the managed tuning and evaluation services for the Google Foundation Models on Vertex AI. This ensures consistency as you can consume and extend those pipelines easily, without having to familiarize yourself with many services.

Getting started with Vertex Pipelines is simple: you define the pipeline's step sequence in a Python file utilizing Kubeflow SDK.⁶⁷ For further details and comprehensive onboarding, consult the official documentation.⁶⁸

Chain & Augment: Vertex AI Grounding, Extensions, and RAG building blocks

Beyond training, tuning and adapting models and prompts directly, Vertex AI offers a comprehensive ecosystem for augmenting LLMs, to address the challenges of factual grounding and hallucination. The platform incorporates emerging techniques like RAG and agent-based approaches.

LLM function calling⁶⁹ empowers users by enhancing the capabilities of language models (LLMs). It enables LLMs to access real-time data and interact with external systems, providing users with more accurate and up-to-date information. To do that, users need to provide function definitions such as description, inputs, outputs to the gen AI model. Instead of directly executing functions, the LLM intelligently analyzes user requests and generates structured data outputs. These outputs propose which function to call and what arguments to use. Gemini models regularly top function calling leaderboards and support multi-modal and compositional function calling. Not all LLMs support function calling.

LLM sandbox & code interpreter allow LLMs to write code (usually python) and execute that code in a sandbox runtime on the server. This can allow a model to build its own tools on the fly and execute tools in a different fashion than the function calling feature. In some cases this may outperform function calling.

LLM structured output allows LLMs to return outputs in JSON or adhering to a specific schema of fields and allowed value types. This is extremely important to application and agent developers who are stitching together multiple LLM outputs across the steps of an application, to enforce a data model and have the LLM respect it.

LLM context caching and context window are extremely important aspects of LLM selection for any agent builder. A large context window allows a lot of flexibility, bringing more information to the LLM, about the session or task in scope, and about the skill or knowledge in scope. Caching a large context window allows a developer to provide knowledge to LLM without having to consume all of the input tokens at runtime for every execution - resulting in faster and cheaper execution. If you need more information than can fit into your context window, or need speed or pricing optimizations beyond context caching, you need to consider RAG and Grounding.

RAG overcomes limitations by enriching prompts with data retrieved from vector databases, circumventing pre-training requirements and ensuring the integration of up-to-date information. Agent-based approaches, popularized by ReAct prompting, leverage LLMs as mediators interacting with tools like RAG systems, APIs, and custom extensions. Vertex AI facilitates this dynamic information source selection, enabling complex queries, real-time actions, and the creation of multi-agent systems connected to vast information networks for sophisticated query processing and real-time decision-making.

Vertex AI Grounding⁵ helps users connect large models with verifiable information by grounding them to internal data corpora on Vertex AI Agent Builder⁷⁰ or external sources using Google Search. This enables two key functionalities: verifying model-generated outputs against internal or external sources and creating RAG systems using Google's advanced search capabilities that produce quality content grounded in your own or web search data.

Vertex AI extensions⁶ manage the whole lifecycle of tools for agents and foundation models. Instead of function calling where the developer invokes an API as recommended by the LLM, using Vertex AI extensions allows the model to do so automatically. Google offers a set of 1st party extensions like Vertex AI Search⁷ (RAG) and Code Interpreter,⁷¹ (sandbox). Apigee Cloud Hub allows you to automatically turn any API spec into an extension.

Vertex AI Agent Builder⁷⁰ is a suite of products that allows you to quickly build gen AI applications and agents, streamlining setup of Google quality search on your data wherever it is stored, providing RAG to agents and applications. Agent Builder is integrated with Conversational Agents (playbooks) which is a no-code, natural language agent building product. All agents, no matter how they are built, will require knowledge provided by RAG and search.

Vertex AI Search is an out-of-the-box, fully managed search and RAG provider built on Google Search technologies, for your data. With it you are be able to easily ground your agents by indexing a diverse range of data sources, including structured datastores such us BigQuery, Spanner, Cloud SQL, unstructured sources like website content crawling and cloud storage as well as connectors to Google drive and dozens of third party APIs. Vertex AI Search brings together a robust foundation of Google Search technologies, encompassing semantic search, hybrid search, semantic chunking, embedded image & table annotations, ranking, algorithms, query rewriting and user intent understanding. Under the hood it optimizes each of these steps, embedding models, and ranking strategies. It abstracts away these complexities and allows users to simply specify their data source to initiate the gen AI-powered agent. This approach is ideal for organizations seeking to build robust search experiences for standard use cases without extensive technical expertise, and this product implements search ready for any scale.

RAG Engine is a companion product which can orchestrate any set of components using LlamaIndex like syntax which should be familiar for many developers. This can assemble a bespoke RAG pipeline to build your own search engine, and still utilize many of the same underlying components which Vertex AI Search uses like the Document AI Layout Parser or the Ranking API or Check Grounding API. This approach is much more flexible, but still provides the orchestration and runtime for a DIY RAG implementation on any components in any configuration.

Vector databases are specialized systems for managing multi-dimensional data. This data, encompassing images, text, audio, video, and other structured or unstructured formats, is represented as vectors capturing its semantic meaning. Vector databases accelerate searching and retrieval within these high-dimensional spaces, enabling efficient tasks like finding similar images from billions or extracting relevant text snippets based on various inputs. For a deeper dive into these topics, refer to 4 and 19. Vertex AI offers three flexible solutions for storing and serving embeddings at scale, catering to diverse use cases and user profiles.

Vertex AI Vector Search⁷ is a highly scalable low-latency similarity search and fully managed vector database scaling to billions of vector embeddings with auto-scaling. This technology, built upon ScaNN⁷² (a Google-developed technology used in products like Search, YouTube, and Play), allows you to search from billions of semantically similar or related items within your stored data. In the context of gen AI, the most common use cases where Vertex Vector Search can be used are:

1. Finding similar items (either text or image) based solely on their semantic meaning, in conjunction with an embedding model.
2. Creating a hybrid search approach that combines semantic and keyword or metadata search to refine the results.
3. Extracting relevant information from the database to feed into LLMs, enabling them to generate more accurate and informed responses.

Vertex AI Vector Search primarily functions as a vector database for storing pre-generated embeddings. These embeddings must be created beforehand using separate models like Vertex Embedding models⁷³ (namely `textembedding-gecko`, `text-embedding-gecko-multilingual`, or `multimodalembedding`). Choosing Vertex Vector Search is optimal when you require control over aspects like the chunk, retrieval, query and models strategy.

This includes fine-tuning an embedding model for your specific data. However, if your use case is a standard one requiring little customization, a readily available solution like Vertex Search might be a better choice.

Vertex AI Feature Store⁷⁴ is a centralized and fully managed repository for ML features and embedding. It enables teams to share, serve, and reuse machine learning features and embeddings effortlessly alongside other data. Its native BigQuery²³ integration eliminates duplication, simplifies lineage tracking and preserves data governance. Vertex AI Feature Store supports offline retrieval and an easy and fast online serving for machine learning features and embeddings. Vertex AI Feature Store is a good choice when you want to iterate and maintain different embedding versions alongside other machine learning features in a single place.

Vertex AI offers the flexibility to seamlessly create and connect various products to build your own custom grounding, RAG, and Agent systems. This includes utilizing diverse embedding models (multimodal, multilingual), various vector stores (Vector Search, Feature Store) and search engines like Vertex AI Agent Builder, extensions, grounding, and even SQL query generation for complex natural language queries. Moreover, Vertex AI provides SDK integration with LangChain⁹ to easily build and prototype applications using the umbrella of Vertex AI products. For further details and integration information, consult the official documentation⁷⁵ and official examples.⁷⁶

Evaluate: Vertex AI Experiments, Tensorboard, & evaluation pipelines

In the dynamic world of gen AI, experimentation and evaluation are the cornerstones of iterative development and continuous improvement. With a multitude of variables influencing Gen AI models (prompt engineering, model selection, data interaction, pretraining,

and tuning), evaluation goes hand-in-hand with experimentation. The more seamlessly experiments and evaluations can be integrated into the development process, the smoother and more efficient the overall development becomes. Vertex AI provides cohesive experimentation and evaluation products permitting connected iterations over applications and models alongside their evaluations.

Experiment

The process of selecting, creating, and customizing machine learning (including large models) and its applications involves significant experimentation, collaboration, and iteration.

Vertex AI seamlessly integrates experimentation and collaboration into the development lifecycle of AI/ML and gen AI models and applications. Its **Workbench Instances**⁷⁷ provide Jupyter-based development environments for the entire data science workflow, connected to other Google Cloud services and with GitHub synchronization capabilities. **Vertex Colab Enterprise**⁷⁸ accelerates the AI workflow by enabling collaborative coding and leveraging code completion and generation features.

Vertex AI also provides two tools for tracking and visualizing the output of many experiment cycles and training runs. **Vertex AI Experiments**⁷⁹ facilitates meticulous tracking and analysis of model architectures, hyperparameters, and training environments. It logs experiments, artifacts, and metrics, enabling comparison and reproducibility across multiple runs. This comprehensive tracking permits data scientists to select the optimal model and architecture for their specific use case. **Vertex AI TensorBoard**⁸⁰ complements the experimentation process by providing detailed visualizations for tracking, visualizing, and sharing ML experiments. It offers a range of visualizations, including loss and accuracy metrics tracking, model computational graph visualization, and weight and bias histograms,

which - for example - can be used for tracking various metrics pertaining to training and evaluation of gen AI models with different prompting and tuning strategies. It also projects embeddings to lower-dimensional space, and displays image, text, and audio samples.

Evaluation

Vertex AI also provides a comprehensive set of evaluation tools for gen AI, from ground truth metrics to using LLMs as raters.

For Ground Truth-based metrics, **Automatic Metrics in Vertex AI**⁸¹ lets you evaluate a model based on a defined task and “ground truth” dataset. For LLM-based evaluation, pairwise and pointwise model-based evaluation in Vertex AI⁸² uses a large model to evaluate the output of multiple models or configurations being tested, helping to augment human evaluation at scale.

In addition to that, users can also leverage a set of pre-built metrics for evaluating gen AI applications and relative SDK, integrated into the Vertex AI Python SDK for rapid and flexible, notebook-based, prototyping. To get started with Evaluation Vertex AI SDK see example in the official documentation.⁸³

Predict: Vertex AI endpoints & monitoring

Once developed, a production gen AI application must be deployed, including all its model components. If the application uses any models that have been trained or adapted, those models need to be deployed to their own serving endpoints. You can serve any model in the Model Garden through Vertex AI Endpoints²¹, which acts as the gateway for deploying your trained machine learning models. They allow you to serve online predictions with low latency,

manage access controls, and monitor model performance easily through Model Monitoring. Endpoints also offer scaling options to handle varying traffic demands, ensuring optimal user experience and reliability.

Along with the prediction service, Vertex AI offers the following features for all Google managed models:

- **Citation checkers:** Gen AI on Vertex performs Citation checks⁷¹. Citations are important for LLMs and gen AI for several reasons. Citing sources ensures proper acknowledgment of sources and prevents plagiarism and demonstrates transparency and accountability. Citing sources is essential for LLMs and gen AI also because they help identify, understand potential biases, and enable reproducibility and verification. For example in Google Cloud,⁸⁴ the gen AI models are designed to produce original content, limiting the possibility of copying existing contents. If this happens, Google Cloud provides quotes for websites and code repositories.
- **Safety scores:** Safety attributes are crucial for LLMs and gen AI to mitigate potential risks like bias, lack of explainability, and misuse. These attributes help detect and mitigate biased outputs and mitigate misuse, enabling these tools to be used responsibly. As LLMs and gen AI evolve, incorporating safety attributes will be increasingly essential for responsible and ethical use. For example, Google Cloud added safety scores in Vertex AI PaLM API and Vertex AI Gemini API⁸⁵: content processed through the API is checked against a list of safety attributes, including "harmful categories" and sensitive topics. Each attribute has a confidence score between 0.0 and 1.0, indicating the likelihood of the input belonging to that category. These safety filters can be used in conjunction with all models: be it proprietary ones like Palm2 and Gemini or OSS ones like the ones available in Model garden.

- **Watermarking:** With AI-based tools becoming increasingly popular for creation of content, it's very important to identify if an image has been created using AI. Vertex AI offers digital watermarking and verification for AI-generated images⁸⁶ using the algorithm SynthID⁸⁷ developed by Google DeepMind.
- **Content moderation and bias detection:** By using the Content moderation⁸⁸ and Bias⁸⁹ detection tools on Vertex AI, you can add an extra layer of security on the responses of the LLMs to mitigate the risk that the model training and tuning may sway a model to generate outputs that aren't fair or appropriate for the task.

Govern: Vertex AI Feature Store, Model Registry, and Dataplex

Addressing the multifaceted requirements of data and model lineage and governance in gen AI requires a comprehensive strategy that tackles both conventional challenges and novel regulatory or technical complexities associated with large models. By adopting robust governance, observability, and lineage practices in the development of gen AI solutions, organizations can ensure comprehensive tracking, iteration, and evolution of data. They can also track the large models used, prompt adaptations, tuning, and other artifacts. This facilitates reproducibility of results, transparency and understanding of generated content sources, troubleshooting, compliance enforcement, and enhanced reliability and security. These practices collectively enable the ethical and responsible development and deployment of gen AI solutions. This fosters internal and external trust and fairness in gen AI models and practices. Vertex AI and Google Cloud offer the following comprehensive suite of tools for unified lineage, governance and monitoring, effectively addressing these critical concerns.

In the context of governance and lineage, **Vertex AI Feature Store**⁷⁴ offers:

- Track feature and embeddings versions and lineage, ensuring transparency

- Monitor feature (prompt) and embedding, response drift, and identify potential issues proactively
- Store feature formulas and discover relevant features or embeddings for different use cases
- Utilize feature selection algorithms to optimize model performance
- Consolidate and unify all machine learning data within a singular repository encompassing numerical data, categorical data, textual data, and embeddings representations

Vertex AI Model Registry¹² serves as a centralized repository for comprehensive lifecycle management of both Google proprietary foundational and open-source Machine Learning models. This includes gen AI models in addition to predictive models. This unified platform enables registration, storage, and version control of diverse model types, including various iterations of tuning for large models. Vertex AI Model Registry seamlessly integrates with **Vertex Pipelines**,¹³ facilitating orchestration and management of training and tuning jobs while leveraging lineage capabilities for recording and documenting the lineage from datasets to models and associated artifacts. It also couples with **Vertex AI Experiments**⁷⁹ and **Vertex AI Model Evaluation**,⁹⁰ enabling performance monitoring and comparison of different model versions alongside their artifacts – all within a single interface. Furthermore, Vertex AI Model Registry bolsters observability by providing integrated configuration and access to **Vertex AI Model Monitoring**⁹¹ and logging functionalities. This enables proactive identification and mitigation of both training-serving skew and prediction drift, ensuring reliability and accuracy of deployed models. Users can directly assign desired model versions to endpoints for one-click deployment from Vertex Model Registry or leverage aliases for simplified deployment.

Google Cloud Dataplex¹⁴ provides an organization-wide lineage across product boundaries in Google Cloud. Within the domains of AI and gen AI (and more broadly across data analytics and AI/ML) Dataplex seamlessly integrates with BigQuery and Vertex AI. Dataplex facilitates

the unification, management, discovery, and governance of both data and models. Through comprehensive data lineage, quality, and metadata management capabilities it provides actionable insights for comprehensive data and model understanding. This promotes compliance, facilitates data analysis, and guarantees the training of machine learning models on trusted data sources. This in turn leads to enhanced accuracy and reliability. This integration permits users across an organization to identify ‘champion models’ and ‘golden datasets and features’ across projects and regions in a secure way by adhering to identity access management (IAM)⁹² boundaries. In short, Dataplex encapsulates a framework within an organization that governs the interaction between people, processes and technology across all the products in Google Cloud.

Summary

The explosion of gen AI in the last several years introduced fundamental changes in the way AI applications are developed – but far from upending the MLOps discipline, these changes have only reinforced its basic principles and processes. As we have seen, the principles of MLOps that emphasize reliability, repeatability, and dependability in ML systems development are comfortably extended to include the innovations of gen AI. Some of the necessary changes are deeper and more far-reaching than others, but nowhere do we find any change that MLOps cannot accommodate.

As a result, many tools and processes built to support traditional MLOps can also support the requirements of gen AI. Vertex AI, for instance, is a powerful platform that can be used to build and deploy machine learning models and AI applications. It provides a comprehensive suite of functions for developing both Predictive and gen AI systems, encompassing data preparation, pre-trained APIs, AutoML capabilities, training and serving hardware, advanced fine-tuning techniques and deployment tools, and a diverse selection of proprietary and

open-source foundation models. It also offers evaluation methods, monitoring capabilities, and governance tools, all unified within a single platform to streamline the AI development lifecycle. It's built on Google Cloud Platform, which provides a scalable, reliable, secure and compliant infrastructure for machine learning. It's a good choice for organizations that want to build and deploy machine learning models and AI applications.

The next few years will undoubtedly see gen AI extended in directions that today are unimaginable. Regardless of the direction these developments take, it will continue to be important to build on solid engineering processes that embody the basic principles of MLOps. These principles support the development of scalable, robust production AI applications today, and no doubt will continue to do so into the future.

Endnotes

1. Model Garden on Vertex AI. Available at: <https://cloud.google.com/model-garden>
2. Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, William Fedus. 2022. Emergent Abilities of Large Language Models. Available at: <https://arxiv.org/pdf/2206.07682.pdf>
3. Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, Douwe Kiela. 2022. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Available at: <https://arxiv.org/pdf/2005.11401.pdf>
4. Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, Yuan Cao, Department of Computer Science, Princeton University, Google Research, Brain team, REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS. Available at: <https://arxiv.org/pdf/2210.03629.pdf>
5. Grounding in Vertex AI. Available at: <https://cloud.google.com/vertex-ai/docs/generative-ai/grounding/ground-language-models>
6. Vertex Extensions. Connect models to APIs by using extensions. Available at: <https://cloud.google.com/vertex-ai/docs/generative-ai/extensions/overview>
7. Overview of Vertex AI Vector Search. Available at: <https://cloud.google.com/vertex-ai/docs/vector-search/overview>
8. What is Vertex AI Agent Builder? Available at: <https://cloud.google.com/generative-ai-app-builder/docs/introduction>
9. LangChain. Get your LLM application from prototype to production. Available at: <https://www.langchain.com/>
10. Introduction to the Vertex AI SDK for Python. Available at: <https://cloud.google.com/vertex-ai/docs/python-sdk/use-vertex-ai-python-sdk>
11. Introduction to Vertex AI. Available at: <https://cloud.google.com/vertex-ai/docs/start/introduction-unified-platform>
12. Introduction to Vertex AI Model Registry. Available at: <https://cloud.google.com/vertex-ai/docs/model-registry/introduction>

13. Introduction to Vertex AI Pipelines. Available at: <https://cloud.google.com/vertex-ai/docs/pipelines/introduction>
14. Dataplex. Available at: <https://cloud.google.com/dataplex>
15. BigQuery. Available at: <https://cloud.google.com/bigquery?hl=en>
16. PaLi-Gemma model card. Available at: <https://ai.google.dev/gemma/docs/paligemma/model-card>
17. Version Control. Available at: https://en.wikipedia.org/wiki/Version_control
18. Continuous integration. Available at: https://wikipedia.org/wiki/Continuous_integration
19. TFX is an end-to-end platform for deploying production ML pipelines. Available at: <https://www.tensorflow.org/tfx>
20. Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, Tomas Pfister. 2023. Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes. Available at: <https://arxiv.org/pdf/2305.02301.pdf>
21. Vertex Endpoints. Use private endpoints for online prediction. Available at: <https://cloud.google.com/vertex-ai/docs/predictions/using-private-endpoints>
22. Tuan Duong Nguyen, Marthinus Christoffel du Plessis, Takafumi Kanamori, Masashi Sugiyama, 2014. Constrained Least-Squares Density-Difference Estimation. Available at: <https://www.ms.k.u-tokyo.ac.jp/sugi/2014/CLSDD.pdf>
23. Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, Alexander Smola, 2012. A Kernel Two-Sample Test. Available at: <https://jmlr.csail.mit.edu/papers/v13/gretton12a.html>
24. Oliver Cobb, Arnaud Van Looveren, 2022. Context-Aware Drift Detection. Available at: <https://arxiv.org/pdf/2203.08644.pdf>
25. Google Gemma Model. Available at: <https://gemini.google.com/>
26. Perform metrics-based evaluation. Available at: <https://cloud.google.com/vertex-ai/docs/generative-ai/models/evaluate-models>
27. Gemini Team, Google, 2023. Gemini: A Family of Highly Capable Multimodal Models. Available at: https://storage.googleapis.com/deepmind-media/gemini/gemini_1_report.pdf
28. Anil, Dai et al., 2023. PaLM 2 Technical Report. Available at: <https://arxiv.org/abs/2305.10403>

29. Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamvar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J Fleet, Mohammad Norouzi, 2022. Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding. Available at: <https://arxiv.org/abs/2205.11487>
30. Build the future of AI with Meta Llama 3. Available at: <https://llama.meta.com/llama3>
31. Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, Jason Wei. 2022. Scaling Instruction-Finetuned Language Models. Available at: <https://arxiv.org/abs/2210.11416>
32. Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Available at: <https://arxiv.org/abs/1810.04805>
33. Stable Diffusion. Available at: <https://github.com/CompVis/stable-diffusion>
34. Vertex AI Function Calling. Available at: <https://cloud.google.com/vertex-ai/generative-ai/docs/multimodal/function-calling>
35. Mistral AI. Available at: <https://mistral.ai/>
36. Models available in Model Garden. Available at: <https://cloud.google.com/vertex-ai/docs/start/explore-models#available-models>
37. Vertex AI Studio. Customize and deploy generative models. Available at: <https://cloud.google.com/generative-ai-studio>
38. vLLM. Easy, fast, and cheap LLM serving for everyone. Available at: <https://github.com/vllm-project/vllm>
39. Overview of multimodal models. Available at: <https://cloud.google.com/vertex-ai/docs/generative-ai/multimodal/overview>
40. Text models. Available at: <https://cloud.google.com/vertex-ai/docs/generative-ai/model-reference/text>
41. Imagen on Vertex AI | AI Image Generator. Available at: <https://cloud.google.com/vertex-ai/docs/generative-ai/image/overview>
42. Code models overview. Available at: <https://cloud.google.com/vertex-ai/docs/generative-ai/code/code-models-overview>

43. Convert speech to text. Available at: <https://cloud.google.com/vertex-ai/docs/generative-ai/speech/speech-to-text>
44. Text-to-Speech AI. Available at: <https://cloud.google.com/text-to-speech>
45. Natural Language AI. Available at: <https://cloud.google.com/natural-language>
46. Translate docs, audio, and videos in real time with Google AI. Available at: <https://cloud.google.com/translate>
47. Vision AI. Available at: <https://cloud.google.com/vision>
48. Git. Available at: <https://git-scm.com/>
49. CodeGemma model card. Available at: https://ai.google.dev/gemma/docs/codegemma/model_card
50. TII's Falcon. Available at: <https://falconllm.tii.ae/>
51. Mistral AI. Available at: <https://mistral.ai/>
52. Hugging Face, 2024. Vision Transformer (ViT) Documentation. Hugging Face, [online] Available at: https://huggingface.co/docs/transformers/en/model_doc/vit
53. Mingxing Tan, Quoc V. Le, 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Available at: <https://arxiv.org/abs/1905.11946>
54. Anthropic Claude 3. Available at: <https://www.anthropic.com/news/clause-3-haiku>
55. Anthropic Claude 3 on Google Cloud Model Garden. Available at: <https://cloud.google.com/blog/products/ai-machine-learning/announcing-anthropic-claude-3-models-in-google-cloud-vertex-ai>
56. Vertex AI API. Available at: <https://cloud.google.com/vertex-ai/docs/reference/rest>
57. Vertex AI: Python SDK. Available at: <https://cloud.google.com/python/docs/reference/aiplatform/latest/vertexai>
58. Vertex AI: Node.js Client. Available at: <https://cloud.google.com/nodejs/docs/reference/aiplatform/latest/overview>
59. Vertex AI for Java. Available at: <https://cloud.google.com/java/docs/reference/google-cloud-aiplatform/latest/overview>
60. Customize and deploy generative models. Available at: <https://cloud.google.com/generative-ai-studio>

61. Design text prompts. Available at: <https://cloud.google.com/vertex-ai/docs/generative-ai/text/text-prompts>
62. Introduction to prompt design. Available at: <https://cloud.google.com/vertex-ai/docs/generative-ai/learn/introduction-prompt-design>
63. Supervised tuning. Available at: <https://cloud.google.com/vertex-ai/docs/generative-ai/models/tune-models#supervised-tuning>
64. RLHF model tuning. Available at: <https://cloud.google.com/vertex-ai/generative-ai/docs/models/tune-text-models-rlhf>
65. Vertex AI Distillation. Available at: <https://cloud.google.com/vertex-ai/generative-ai/docs/models/tune-text-models>
66. Create distilled text models. Available at: <https://cloud.google.com/vertex-ai/docs/generative-ai/models/distill-text-models>
67. Pipeline Basics. Available at: <https://www.kubeflow.org/docs/components/pipelines/v2/pipelines/pipeline-basics/>
68. Build a pipeline. Available at: <https://cloud.google.com/vertex-ai/docs/pipelines/build-pipeline>
69. Vertex AI Search extension. Available at: <https://cloud.google.com/vertex-ai/generative-ai/docs/extensions/vertex-ai-search>
70. What is Vertex AI Agent Builder? Available at: <https://cloud.google.com/generative-ai-app-builder/docs/introduction>
71. Generative AI on Vertex AI, Citation Check. Available at: https://cloud.google.com/vertex-ai/generative-ai/docs/learn/overview#citation_check
72. Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar, 2020. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. Available at: <https://arxiv.org/pdf/1908.10396.pdf>
73. Get text embeddings. Available at: <https://cloud.google.com/vertex-ai/docs/generative-ai/embeddings/get-text-embeddings>
74. About Vertex AI Feature Store. Available at: <https://cloud.google.com/vertex-ai/docs/featurestore/latest/overview>

75. Google Cloud Vertex AI. Available at: https://python.langchain.com/docs/integrations/lms/google_vertex_ai_palm
76. Generative AI - Language - LangChain. Available at: <https://github.com/GoogleCloudPlatform/generative-ai/tree/main/language/orchestration/langchain>
77. Introduction to Vertex AI Workbench, Workbench Instances. Available at: <https://cloud.google.com/vertex-ai/docs/workbench/introduction>
78. Introduction to Colab Enterprise. Available at: <https://cloud.google.com/colab/docs/introduction>
79. Introduction to Vertex AI Experiments. Available at: <https://cloud.google.com/vertex-ai/docs/experiments/intro-vertex-ai-experiments>
80. Vertex AI TensorBoard Introduction to Vertex AI TensorBoard. Available at <https://cloud.google.com/vertex-ai/docs/experiments/tensorboard-introduction>
81. Perform metrics-based evaluation. Available at: <https://cloud.google.com/vertex-ai/docs/generative-ai/models/evaluate-models>
82. Perform automatic side-by-side evaluation. Available at: <https://cloud.google.com/vertex-ai/docs/generative-ai/models/side-by-side-eval>
83. Rapid Evaluation Vertex AI. Available at: <https://cloud.google.com/vertex-ai/generative-ai/docs/models/rapid-evaluation>
84. Citation metadata. Available at: https://cloud.google.com/vertex-ai/docs/generative-ai/learn/responsible-ai#citation_metadata
85. Responsible AI. Available at: <https://cloud.google.com/vertex-ai/docs/generative-ai/learn/responsible-ai#filters-palm-api>
86. Imagen on Vertex AI | AI Image Generator. Available at: <https://cloud.google.com/vertex-ai/docs/generative-ai/image/overview>
87. SynthID. Identifying AI-generated content with SynthID. Available at: <https://deepmind.google/technologies/synthid/>
88. Moderate text. Available at: <https://cloud.google.com/natural-language/docs/moderating-text>
89. Model bias metrics for Vertex AI. Available at: <https://cloud.google.com/vertex-ai/docs/evaluation/model-bias-metrics>

90. Model evaluation in Vertex AI. Available at: <https://cloud.google.com/vertex-ai/docs/evaluation/introduction>
91. Introduction to Vertex AI Model Monitoring. Available at: <https://cloud.google.com/vertex-ai/docs/model-monitoring/overview>
92. Identity and Access Management (IAM). Available at: <https://cloud.google.com/iam/docs>
93. Agents. Available at: <https://www.kaggle.com/whitepaper-agents>