Google Cloud

A platform-centric approach to scaling generative AI in the enterprise

September 17, 2024

Table of contents

Introduction

This document describes the development and deployment of production-grade generative AI solutions, with a detailed technical comparison of different deployment approaches. Selection of a deployment approach is a critical choice point for solution builders. The document provides insights and recommendations to help you shape this decision.

Lessons learned from the field

Since 2022, large enterprises have prioritized experimentation with generative artificial intelligence (AI). Their experimentation and subsequent pilot deployments have shown the significant potential of generative AI to drive efficiency gains, enhance quality, and launch innovative offerings. However, despite their interest in deploying these solutions, only 10% of enterprises have matured from pilot to production[1].

During this period, enterprises discovered that foundation models are prone to hallucination, struggle to adapt to edge cases like jargon, and amplify biases. As a result, their highest priority use cases, such as customer support automation, demand a solution beyond the capabilities of foundation models alone. In fact, an executive at a large healthcare provider noted how their company avoided "high-value clinical use cases, and only felt confident in deploying generative AI solutions on efficiency-boosting administrative tasks."

However, recent advances enable more accurate, grounded answers, creating opportunities for organizations to improve patient outcomes. Those advances require a suite of capabilities to provide the model with domain-specific context, validate outputs, and ensure consistent performance. For example, monitoring and evaluation are required to validate performance, and retrieval augmented generation (RAG)[2] or function calling extends model knowledge and enables workflows. Collectively, capabilities like these are critical in deriving value from a generative AI solution.

To assemble these capabilities, AI solution builders, including AI engineers, product managers, data scientists, and line of business (LOB) stakeholders, can select services for their specific use case without full consideration for broader enterprise implications. Use cases that are developed through this approach can accumulate redundant components and they have limited governance and weaker cost controls. Further, as new services and capabilities are launched and updated, organizations with custom implementations can find it challenging to integrate these advancements. For instance, a technology executive at a B2B software company described a scenario where their team needed to replace a vector database due to evolving

[1] Gartner: Let Generative AI Flourish Through Bottom-Up Innovation
[2] Google: What is Retrieval Augmented Generation

needs. This replacement required significant rework, led to missed milestones, and ultimately delayed their project momentum for months. Such practices contribute to slower enterprise adoption due to mounting technical debt, escalating costs, and poor governance.

To enable deployments at scale, generative AI platforms—like Vertex AI—combine infrastructure, models, and tooling like orchestration and evaluation. Generative AI platforms integrate models from multiple providers, infrastructure, developer tools, and agent workflow frameworks into a single integrated and open platform. Platforms can accelerate development and deployment, scale to meet demand, and help realize tangible benefits.

Framework to deploy a generative AI solution

Deploying a generative AI solution requires planning for capabilities that support the model lifecycle to deliver enterprise-grade performance, experience, and management capabilities. These capabilities are typically deployed in five steps—discover, develop, deploy, operate, and govern—which are shown in the following diagram:

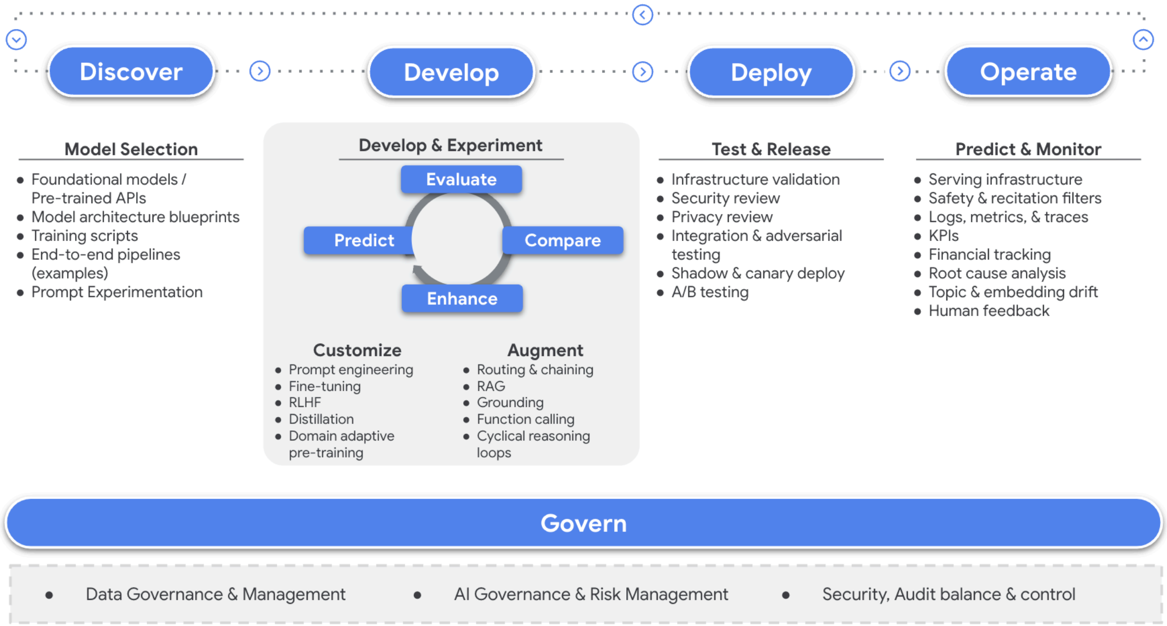

Figure 1: Gen AI use case deployment framework.

The following sections provide details about each step.

Discover

Before you develop a generative AI solution, you typically shortlist models for further evaluation. However, creating a shortlist can be challenging due to a lack of standardized model metrics, comparison factors, or readily available evaluation tooling. Vendor-provided metrics or third-party metrics like MMLU, GSM8K, and BIZ-Bench-Hard are useful for pre-filtering, but they don't directly translate to specialized use cases. For more human-driven comparisons, AI solution builders must set up sandboxes with access to multiple models that allow them to compare responses and prompts side-by-side. Initially, they start with basic prompts to test the feasibility of a use case and then do minor prompt engineering to enhance functionality.

Develop

After a use case has been deemed feasible and a candidate model is selected, the development phase begins. Generative AI use cases typically require small test datasets like customer intents and corresponding actions to drive solution evaluation and optimization. However, at scale, use cases like customer service or knowledge management require access to often distributed and non-standard enterprise data to generate grounded responses, which often makes building a dataset the most challenging step of implementation.

For example, consider an executive at a global software company that's developing a contract search use case. They faced significant challenges, saying "My team encountered difficulty accessing the contracts. We were unfamiliar with Salesforce, where a portion of the data was stored. Further, many contracts were located in a separate Oracle solution, complicating the process. Once we retrieved the contract data, the next challenge was ensuring accurate capture of the legal context within each contract."

The preceding experience is common and it reveals three key obstacles: sourcing data, assembling a representative dataset, and finding adequate labeled data. To source data, you need data connectors to access enterprise systems like CRM and HRIS, prebuilt connections to databases[3] like MySQL, and feature stores[4] like Vertex AI Feature Store, Feast, and Feathr to enable the reuse of data. To fill gaps, you could also generate synthetic data from existing, high-quality datasets. Finally, labeling services like Snorkel or Labelbox can use semi-supervised learning to label at scale to ensure proper data labels.

[3] Google: Integration Connectors

[4] Google: Introduction to Feature Management

After the dataset is ready, you can start building a solution. Building on lessons learned from experimentation and prototyping, leading enterprises follow a four-step iterative process to refine a candidate model for production:

1. **Infer**: After the solution has been improved, it can generate inferences and produce input-output pairs, associated metadata like timestamps and confidence intervals, and technical performance such as throughput and utilization of resources like CPU, memory, and GPU.

2. **Evaluate**: Using data points from Predict, AI solution builders compute comparative scores like the following:

   - Output metrics like perplexity, precision, recall, and BERT score
   - Domain-specific metrics like BLEU[5]
   - Technical metrics like response time
   - Business metrics like CSAT, adoption rate, and TCO

3. **Compare**: With an evaluation framework for each model, candidate solutions can be compared to support a decision. If the solution's performance meets expectations, you move on to deployment. However, if the results are inconclusive, you typically iterate on enhancements until you develop a solution that meets target outcomes.

4. **Enhance**: After AI solution builders have identified gaps in model performance like accuracy or latency, they can improve outcomes by following customization or augmentation processes. Customization refers to modifying model behavior by changing inputs, such as through prompt engineering, or updating parameters, such as through fine-tuning, adapter tuning, and reinforcement learning from human feedback. In contrast, augmentation adds context to the model in order to improve outputs through techniques like RAG, customizable re-rankers[6], function calling[7], and grounding[8].

[5] BLEU: a Method for Automatic Evaluation of Machine Translation, Papineni et al, 2002

[6] Re-rankers recompute search results ordering to tailor them for a particular situation or user.

[7] Function calling enhances models with delegation capabilities for tasks like multiplication.

[8] Grounding is connecting model output to verifiable sources of information.

Deploy

After the solution is developed, it must be tested for risks like hallucinations, data leakage, and real-world performance. You typically start by conducting adversarial safety testing[9] to expose security vulnerabilities and customer data. This testing also helps you to assess adherence to privacy standards through legal reviews. In addition, the solution's efficacy should be compared to existing processes through shadow deployments[10] or A/B testing[11]. After you have completed security, privacy, and performance reviews, you should deploy to production by using a CI/CD process.

Operate

After a solution is in production, it can undergo significant performance changes due to variations in input data. Performance changes necessitate comprehensive monitoring to track safety metrics, solution performance like task completion rate, and trace logs. Often, evaluations that are used for solutions are reused and combined with real-world feedback to generate alerts when solution performance deviates from an expected range. Similarly, as solutions scale to answer more queries, their technical performance and costs can substantially change. At this point, usage limits or cost tracking can be implemented to prevent overruns. If you observe consistent cost issues or performance deviations, you should consider restarting the iterative development phase to improve outcomes.

Govern

Between the Discover and Operate phases, you will need rigorous governance to manage risks of bias, harm, and improper use, and to maintain legal and regulatory compliance. To manage risks, solutions need safety filters to prevent generating harmful or inappropriate answers. Solutions also need guardrails to limit the scope and prevent unplanned situations, such as offering clinical advice in a customer service application.

To maintain legal and regulatory compliance, solutions must leverage security technology like Identity and Access Management (IAM) for permissions, authorization to securely connect external systems, role-based access controls (RBAC) to regulate information access, and logging to keep audit records. Additionally, to ensure the traceability and accountability of models, solutions need to leverage systems for model lineage, experimentation tracking, and data lineage to create auditable records and enable rollbacks if needed.

[9] Adversarial safety testing is a family of techniques that simulate attacks or challenging conditions to expose vulnerabilities and assess a system's defenses.
[10] Shadow deployment is a technique where a model or solution is deployed simultaneously with an existing model, but its outputs do not affect system behavior. Instead they are recorded for future comparison.
[11] A/B Testing is a family of techniques to compare the performance of one or more candidates against an existing champion in a real world setting.

Approaches to generative AI solution development

A typical use case for generative AI begins with aligning around a business outcome like cost reduction, revenue expansion, or experience improvements. As AI solution builders, you can explore two primary approaches to implementing a generative AI solution: decentralized and centralized (referred to as *platform*). In the decentralized approach, you select individual services from providers to build an end-to-end solution. Alternatively, you might select a platform like Vertex AI, which typically includes infrastructure, models, and curated tooling in an integrated package that can serve multiple use cases. The next sections discuss both approaches in detail.

Decentralized approach

In the decentralized approach, solution builders make selections for every service in the AI solution stack. For example, some solution builders begin by selecting a model that's fit for their target use case, typically through a combination of qualitative metrics and quantitative metrics like LMsys ELO and MMLU. Other solution builders might prefer to begin with the optimal vector database for their use case or to select an orchestration engine like LangChain as their first service. The following diagram shows the types of services that are required to enable a decentralized approach:

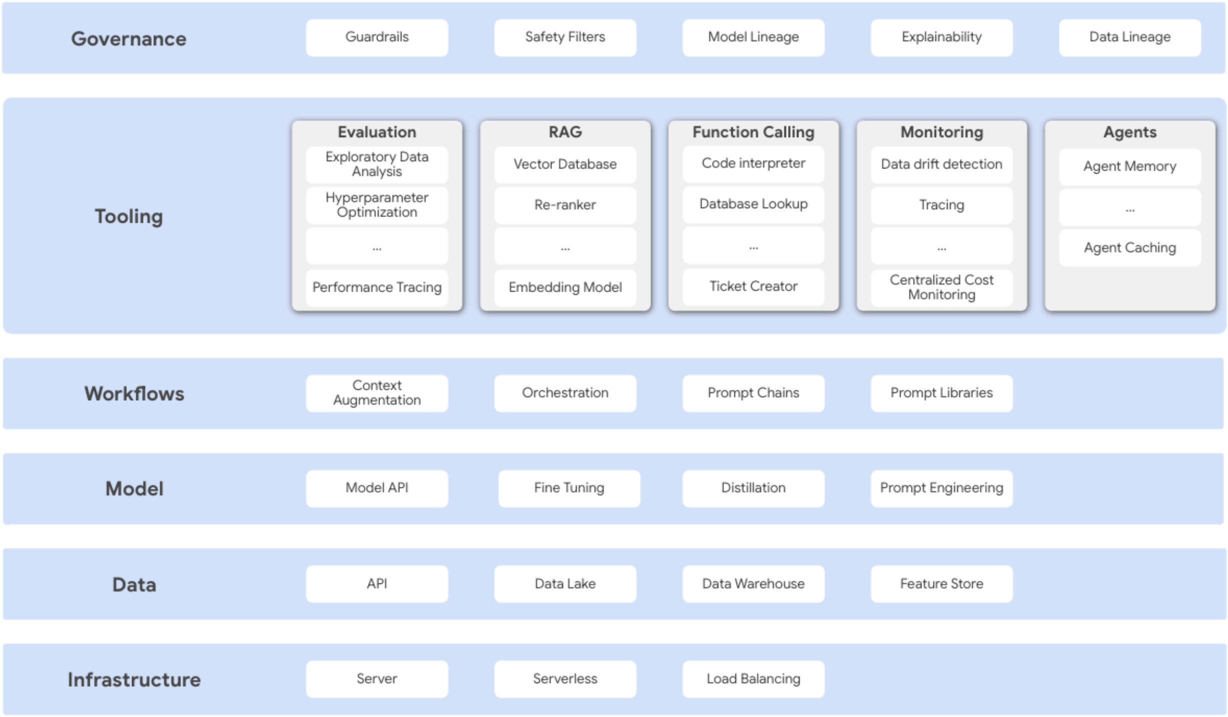

Figure 2: Services that are required to enable a decentralized approach (along with open-source tools).

The preceding diagram shows the following service requirements:

- **Governance**: Provides services like guardrails, safety filters, model lineage, explainability, and data lineage.
- **Tooling**: Performs services like evaluation, RAG, function calling, monitoring, and agent services like memory and caching.
- **Workflows**: Provides services like context augmentation, orchestration, prompt chains, and prompt libraries.
- **Model**: Provides services like a model API, fine tuning, distillation, and prompt engineering.
- **Data**: Provides services like an API, data lake, data warehouse, and feature store.
- **Infrastructure**: Provides services like servers, serverless implementation, and load balancing.

After solution builders select the first service or set of services, they iterate through the enhance, infer, evaluate, and compare steps that are described in the preceding "Deploy" section. The solution builders then augment or customize model services like re-rankers. With each service, you have to understand the API, configure, integrate, and deploy it, and optimize performance factors like response time. To illustrate these steps, consider a document-search use case. To build a document search feature, you might take the following steps:

1. Choose a vector database like Pinecone or Pgvector, design a schema, and deploy the database.
2. Select an embedding model based on performance and cost considerations.
3. Fine-tune the embedding model and use use-case data to optimize performance.
4. Select a chunking[12] strategy to segment the data into smaller, digestible pieces.
5. Populate the vector database with embeddings and mappings to source content.
6. Select a re-ranker to improve underlying retriever performance and to contextualize results.
7. Optimize the re-ranker by fine-tuning with labeled data sets.
8. Deploy an end-to-end RAG pipeline to production.

The decentralized approach of selecting and combining individual services lets builders have fine-grained control over every service. For example, you can adjust re-ranker configurations, fine-tune the model, or augment context. By combining these optimizations, you can maximize performance. Therefore, the decentralized approach is well suited for use cases with demanding performance requirements or organizations that can manage complex deployments.

However, building custom connections and deploying services can be time consuming, whether for RAG, or other services like function calling. Consequently, when builders use multiple

[12] Chunking strategy is a technique for segmenting larger documents into small, manageable pieces like 50 characters that can be vectorized and stored to power more accurate search results.

external services, development time often compounds. Similarly, switching services is a significant undertaking and doing so can create more potential failure points. An executive highlighted this challenge: "In our pilot, we selected a vector database that enabled us to get going quickly; later, we discovered latency issues when we scaled to the 100,000+ documents for our use case. We had to start over by setting up a new database and refactoring connections between the database and model."

In addition, as generative AI solutions scale, the decentralized approach can be a potential limiter. Consider the scenario where the RAG architecture, initially used for document search, is now used for a customer service use case. The chunking strategy for RAG, which might incorporate entire case descriptions in a chunk, now adds extraneous information to the summary for customer service use cases, which decreases accuracy[13]. Additionally, learnings from the document-search use case are only incrementally applicable to a customer service use case because the latter might require different services or customized optimizations. Different needs and optimizations limit the efficiency gain for subsequent use cases and might limit the speed at which generative AI can scale across an enterprise.

Further, when solution builders develop for the customer service use case, they will likely choose models or tools with a faster response time, leading to a different stack relative to RAG. This fracturing of the stack can complicate the integration of auditing, cost controls, and governance frameworks, and lead to partial coverage. Consequently, the decentralized approach can lead to higher-than-expected TCO, unforeseen risks, and ultimately slower scaling of generative AI.

Platform approach

In contrast to the decentralized approach, the platform approach begins by selecting a platform. The platform is usually comprised of infrastructure, models, and curated tools that can serve multiple use cases. Rather than integrating specific services for every capability like the decentralized approach, the platform approach provides solution builders with curated features. The features can be customized for each use case, along with support tools like governance frameworks to manage scaled implementations.

[13] Unveiling the Optimal Chunk Size in Retrieval-Augmented Generation

The following diagram shows the features that are provided by using a platform approach:

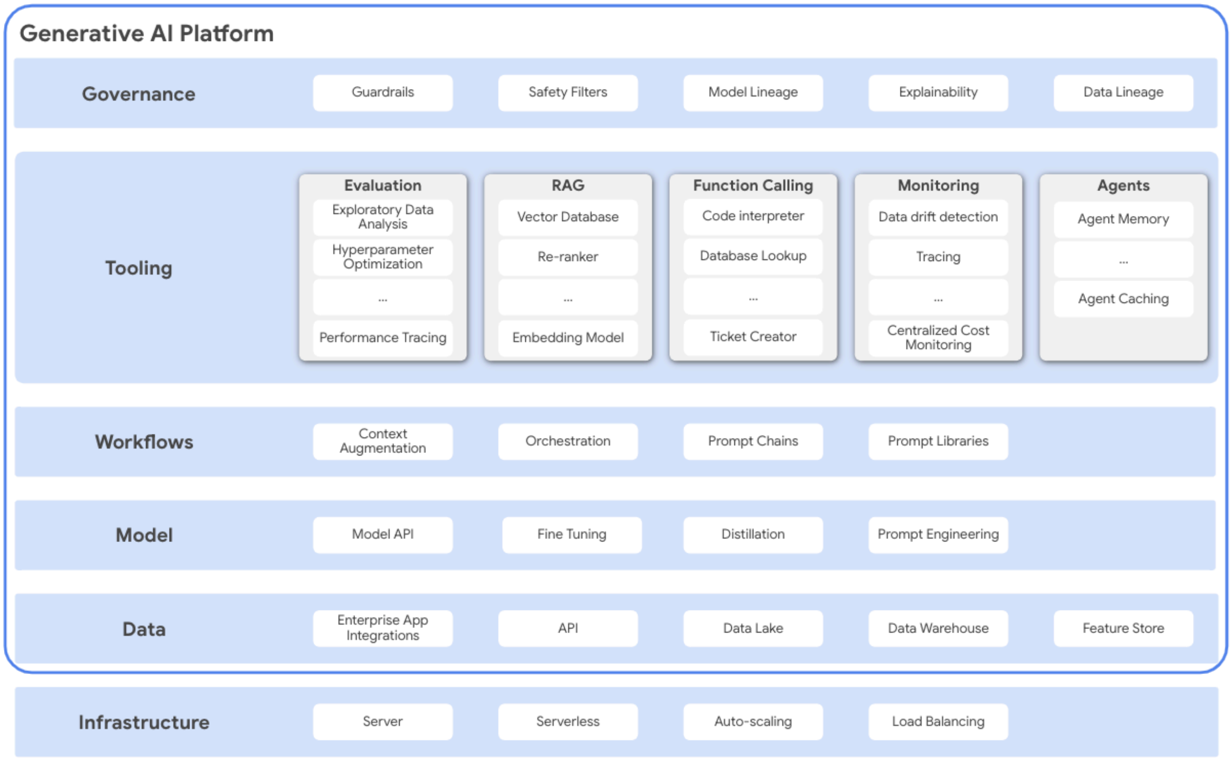

Figure 3: Features provided by a platform like Vertex-AI that enable a centralized approach.

The diagram shows how a platform can combine tooling to provide many of the services that are selected individually in the decentralized approach. In the diagram, Vertex AI provides services for governance, tooling, workflows, model, and data, while the infrastructure services are part of the Google Cloud platform that Vertex AI is built on.

Typically, a platform has several features built in, including the following:

- **Model garden**: A set of APIs to access models, both open-source and proprietary, allowing solution builders to explore, select, and integrate models into solutions. A model garden also facilitates APIs to third-party hosted proprietary models.
- **Evaluation**: Test datasets, dataset creation tools, experimentation frameworks, and tracing tools to perform evaluations on multiple models for their use case.
- **Infrastructure abstraction**: Scale compute, data, and networking on demand and automatically load balance across services and endpoints.

- **Developer tools**: Features to customize and train models, including collaborative workbenches like Jupyter notebooks, prompt engineering tools, fine-tuning capabilities, and CI/CD for deployment.

- **Workflows**: Orchestration of end-to-end processes, combining features like RAG and function calling to execute complex tasks and processes.

- **Agents**: Development and management of autonomous software to tackle goals, combining workflow tooling with memory management and prompt libraries.

- **Connectors**: Prebuilt integrations with commonly used applications, databases, and systems of record.

The built-in features can range from turn-key features like end-to-end search to AI primitives like an embedding model. The breadth and built-in support of these features eliminate the need for integration, testing, extensive deployment, or custom code. To illustrate, consider the document search use case that was described in the "Decentralized approach" section. In contrast to the decentralized approach, you can expect to use built-in features and turn-key features and do the following:

1. Access a RAG feature.
2. Configure the vector database by using a drop-down list to select a vector dimensionality and a similarity metric like euclidean.
3. Select the embedding model and fine-tune by using a web-based UI.
4. Configure the re-ranker by connecting user data through a no-code interface to add context.
5. Deploy the feature to production.

As solution builders, you can leverage the platform approach to add features, configure them, and test them by using a combination of no-code, low-code, or a unified console. Built-in platform features are easier to use compared to decentralized services, and they're also interoperable, eliminating the need for custom connections. Using the platform approach shifts the focus from creating integrations and tuning features to basic configuration and minor optimization, which lets you get to production faster. However, the typical tradeoff compared to the decentralized approach is the loss of fine-grained control over the deployment.

Unlike the decentralized approach, the platform approach lets you quickly learn how to set up features by leveraging use case templates and reusing common tools, patterns, and processes, which accelerates time to market. Consider the additional customer service use case that was described in the "Decentralized approach" section. With the platform approach, instead of selecting completely different services, an implementation team selects from a set of curated services, which helps to ensure standardization. Further, using platform tools, solution builders might configure RAG architectures with a different chunking strategy and make adjustments to the re-ranker, all while delivering adequate performance. As low-code or no-code tools are frequently available within a platform, solution builders need fewer specialized resources, which accelerates time-to-market.

Platforms can also make integrations easier. For example, an executive at a large health insurer said, "Our platform integrates directly with our governance process, enabling teams to easily report standard metrics and manage the approval process from their development platform." This easy-to-use process fosters developer buy-in and streamlines reviews. Further, this executive mentioned, "Our most popular feature is our centralized billing. Teams get a single bill for each project, making budgeting and cost control straightforward." By consolidating financial information, platforms make cost controls easier and give an enterprise confidence in financial projections. These deep integrations, which are built in to the platform approach, provide flexible features, simpler implementation, and integrated governance, which enables enterprises to implement faster and deploy more use cases to production.

Comparison of decentralized and platform approaches

The decentralized and platform approaches are two proven approaches to implementing generative AI solutions. This section contrasts the strengths and weaknesses of each.

Business factors

Consideration	Decentralized approach	Platform approach
Time-to-market	**Longer**, due to the manual selection, integration, and testing of individual services.	**Shorter**, through the use of pre-integrated features that enable rapid prototyping and accelerate deployment.
Scalability	**Limited**, due to custom services integrations and deployment, resulting in higher development and maintenance costs, and slower deployments.	**High,** due to built-in, configurable features that simplify management and reduce the need for custom development.
Flexibility	**Higher**, allows for best-in-class services, such as model monitoring from one vendor with model registry from an alternate vendor.	**Lower**, due to pre-selected, built-in features, constraining last-mile optimization opportunities.
Customer support	**Inconsistent**, because diverse vendor relationships can result in varying levels of service quality.	**Consistent**, because of the customer support experience from a single vendor.

Technical factors

Consideration	Decentralized approach	Platform approach
 Consistency & repeatability	**Lower**, because solution builders can select, develop, and configure services, which leads to a lack of consistency across different solutions.	**Higher**, because solution builders choose from curated features.
 Ease of deployment	**More complex**, due to the deployment of individual services that might not be built for interoperability.	**Easier**, due to standardized features that are designed and maintained to be interoperable.
 Development complexity	**Higher**, because services require custom development, configuration, and connections.	**Lower**, due to prebuilt, interoperable, plug-and-play features.

Risk and cost management factors

Consideration	Decentralized approach	Platform approach
Total cost of ownership (TCO)	**Higher and unpredictable**, due to the technical and operational overhead of developing, maintaining, integrating, and upgrading duplicative models and services.	**Lower and predictable**, due to fewer vendor relationships, interoperable standardized features, and lower maintenance costs.
Governance	**Manual**, including deployment of guardrails, safety features, and model management through individual services.	**Integrated**, because guardrails, safety features, model management, and data lineage are typically prebuilt.
Vendor lock-in	**Minimal**, because solution builders can select services from multiple vendors, which limits the extent of migration and reliance on a single vendor.	**Moderate**, because solution builders create most features on a single vendor's platform, and therefore migrating to another platform requires refactoring.
Security	**Higher risk**, due to the need for standalone security services and greater attack surface area from custom integrations and data in motion.	**Lower risk,** due to integrated solution with out-of-the-box security features and less attack surface area due to fewer integration and less data transit.

In summary, the decentralized approach maximizes flexibility by allowing solution builders to choose different vendors for each service. This flexibility potentially enables better performance but can also lead to higher costs and reduced standardization. By contrast, the strength of the platform approach is the abstractions in accessing models, integrating capabilities, and building end-to-end workflows. The platform approach increases speed and improves scalability, enabling more solutions to get to production. However, the platform approach might also limit the tools, capabilities, and control that experienced solution builders seek. These limitations highlight the importance of selecting the right platform that offers the flexibility to meet your needs.

Recommendations for approaching generative AI solution deployment

The decentralized and platform approaches are distinct ways to implement enterprise generative AI. The following guidelines can help you to shape this decision.

● **Deep technical expertise**: Suited for organizations with experienced AI/ML teams developing custom solutions. ● **Best-in-class performance**: Suited to organizations that seek extensive customization to improve last-mile performance and gain a competitive edge. ● **Specialized vendors**: Favors organizations that seek to integrate specific services and vendors that might not be compatible with platform vendors. ● **Flexibility**: Fits organizations that are looking to avoid vendor lock-in to preserve optionality between services, tools, and vendors**.**	● **Rapid time to market**: Suitable when prioritizing fast implementations and deployments with prebuilt features. ● **Enterprise-wide deployments**: Favors a standard, central framework for use across multiple business lines. ● **Lower TCO**: Supports long-term cost-efficiency with lower maintenance costs and flexible pricing. ● **Scalability**: Enables rapid growth, using features like auto-scaling to meet demand. ● **Large, distributed teams**: Fits geographically dispersed teams by encouraging standardization and collaboration. ● **Proof-of-concept deployments**: Suits a quick pilot, using low-code or no-code features for rapid MVP development.

Organizations that are aiming to maximize performance for a few use cases, and that have experience implementing AI solutions, should consider the decentralized approach. For most organizations, particularly those that are prioritizing rapid deployments and scaling, the platform approach is likely more suitable. Platforms like Vertex AI offer models, infrastructure, and tools that simplify implementations and accelerate the deployment of use cases. By adopting the platform approach, organizations can rapidly move proof of concepts to production and realize the benefits of generative AI.