Getting Started With Generative AI in Your Application Architecture

Published 9 August 2023 - ID G00794269 - 38 min read

By Analyst(s): Gary Olliffe

Initiatives: Application Architecture and Integration for Technical Professionals; Software Development for Technical Professionals

Application and product owners are seeking to include generative AI features to enhance user experience, productivity and deliver differentiating outcomes. Application technical professionals driving application architecture must learn how to integrate these capabilities into their solutions.

Overview

Key Findings

- Generative AI is a broad category of machine learning (ML) models and services that produce data (i.e., text, images, code, audio and other complex data structures) based on input prompts. This contrasts with more established discriminative and predictive ML models which categorize, label or assess the "fit" of input data.
- Large language models (LLMS) such as OpenAI's GPT models, Google's LaMDA and PaLM2 and Meta's LLaMa are at the center of current hype (and exploration/innovation) due to their ability to be prompted to generate a wide variety of text-based output (including prose, code and configuration data).
- A fast moving ecosystem of models, tools and techniques means that what good looks like is constantly evolving and it is difficult to effectively match generative AI technologies to application use cases.

Recommendations

Technical professionals responsible for integrating generative AI into their applications should:

- Learn the core patterns for integrating generative AI to help them assess its feasibility and impact on their application architecture.

- Implement a continuous and comparative evaluation approach to find the best combination or capability, price and performance of generative AI models and services.
- Keep your generative-AI-dependent capabilities loosely coupled from core application capabilities to minimize the effort of changing models or providers as new capabilities continue to emerge.
- Use third-party services and their models to learn and evaluate generative AI capabilities and opportunities without the overhead of training and hosting your own models.

Analysis

Generative AI has become a mainstream trend due to recent advancements in transformer-based LLMS and diffusion-based image generation models. Google's Bard, Microsoft Bing Chat, Midjourney, OpenAI's ChatGPT and DALL·E 2, Stable Diffusion and others have allowed the public to experiment with, and use these tools to learn, create and often simply to play. This exposure is driving product owners, architects and developers to explore how the ML models and technologies behind these tools can be used to improve their own applications and processes.

This research answers the question:

How can we add generative AI capabilities to our applications and products?

The detailed answer to this question is provided later in this research in Fundamental Steps for Integrating Generative AI-Based Capabilities Into Your Application, but the key steps are:

- Step 1 — Define your goals and constraints
- Step 2 — Choose generative AI models using key selection criteria including model source, capability and integration approach
- Step 3 — Define your operational process to protect your organization and monitor model behavior
- Step 4 — Prepare and test your chosen model(s) including engineering prompts and model ensembles where necessary

- Step 5 — Integrate model interactions into your application favoring loose coupling to support future evolution and iteration

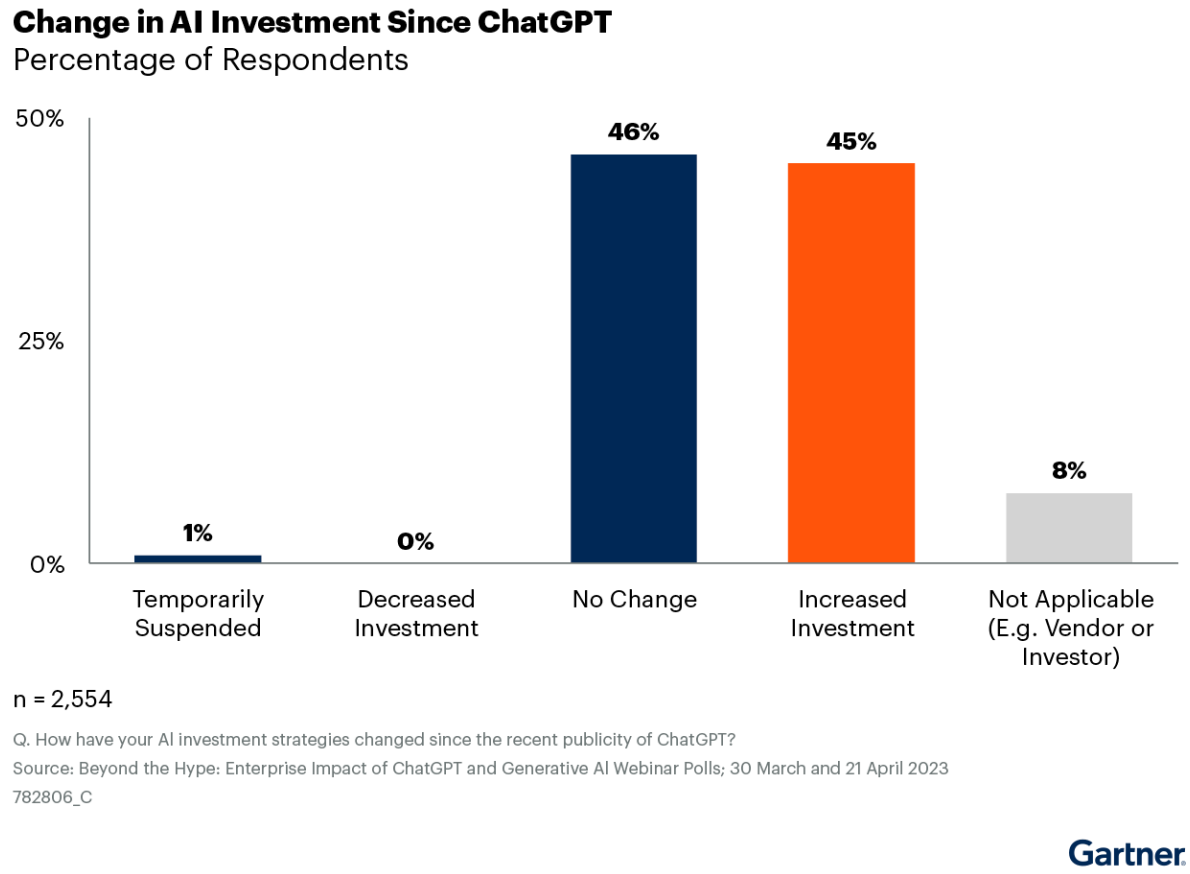
The remainder of this research explores the drivers for integration of generative AI capabilities into your applications, the key logical components of a generative-AI-based solution, the challenges you will face and the emerging patterns and practices you should learn.

Why Generative AI Matters

ML has been easily accessible to application architects and software engineers for many years, for example through the use of vendor-trained models exposed as APIs and services for image classification, sentiment analysis, or online fraud detection. Or, through integration with models trained and managed within the organization for predictive maintenance, payment fraud or customer personalization. However, until recently, most of these capabilities were largely focused on classification and prediction tasks. With the recent publicity for LLMs and generative AI we've seen a huge swing in focus and investment of resources to explore AI capabilities. When asked in the webinar, Beyond the Hype: Enterprise Impact of ChatGPT and Generative AI, how their organization's AI investment strategy has changed since the publicity of ChatGPT, 45% reported that it had increased (see Figure 1). [1]

ChatGPT, Google Bard, Microsoft Bing, You.com and others have raised the expectations of millions of users, and business stakeholders will turn to their application architects to translate those expectations into new features and capabilities in their applications and products.

Figure 1: Change in AI Investment Since ChatGPT



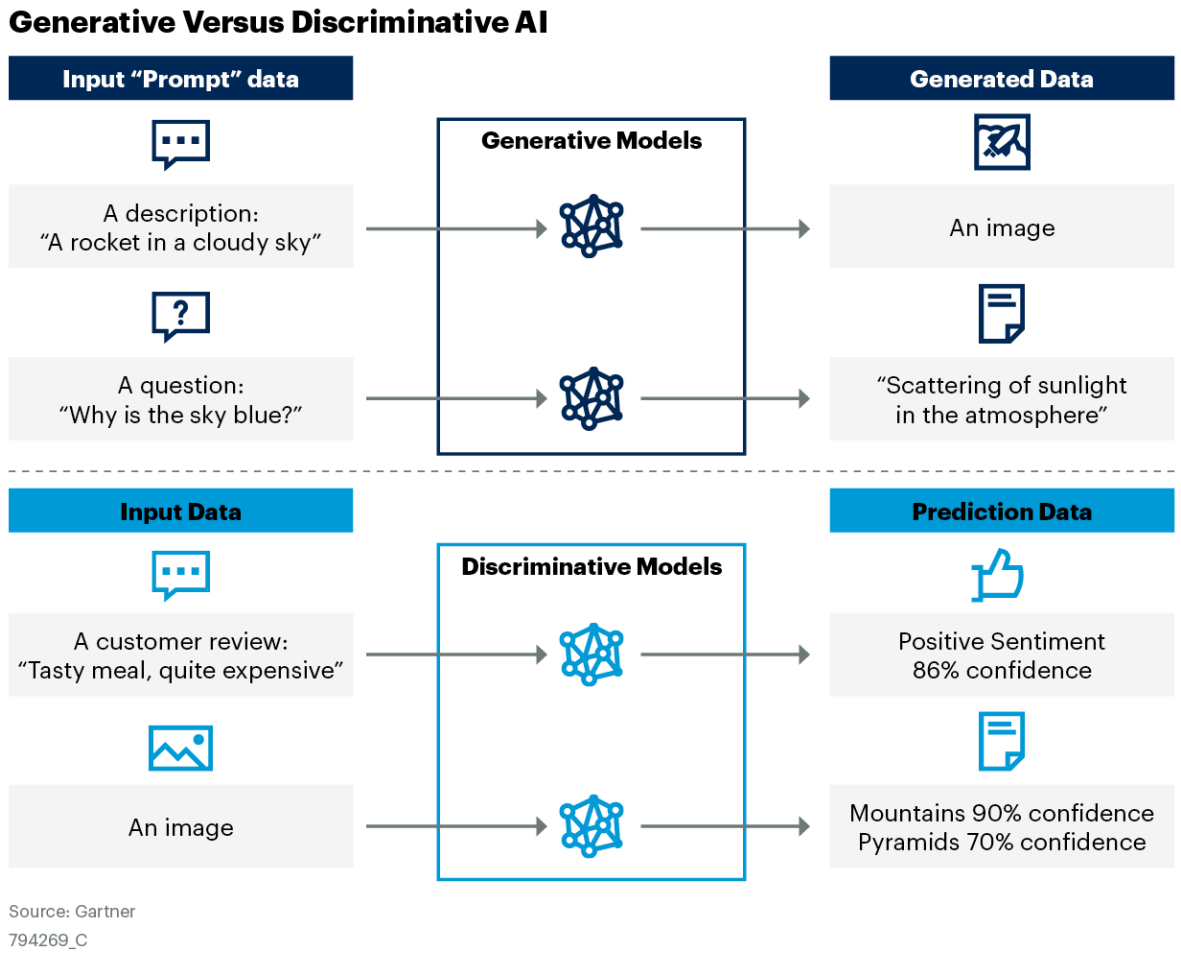
Generative AI Versus Discriminative AI

Generative AI uses ML models to generate new and unique outputs based on user or program input. These models are trained on representative (and often vast) data and, depending on the model and training data, they can generate text, code, images, audio, video or other data.

Most AI/ML approaches used today are “discriminative” (or “predictive.”) This means that the output of the ML model is an analytical prediction or classification based on the input — for example classifying or labeling images, assessing the sentiment of some text, or predicting the next best action for a customer. This distinction is illustrated in Figure 2.

The process of using a trained ML model to produce an output is commonly referred to as inference, for all kinds of AI.

Figure 2: Generative Versus Discriminative AI



To achieve these capabilities, generative AI models use new ML model architectures including generative adversarial networks (GANs), variational autoencoders (VAEs), diffusion models and transformer-based models. Transformer-based models are at the core of the LLMs such as BERT, Google's PaLM2, Meta's LLaMa and Open AI's GPT3.5/GPT4, which have been the trigger for recent hype and innovation in the market. These models are trained on large corpuses of text-based content; ranging from hundreds of billions to a few trillion tokens. This trained knowledge is then captured in the LLM parameters, which can range from a few billion to hundreds of billions and more.

Examples of Generative AI Use Cases

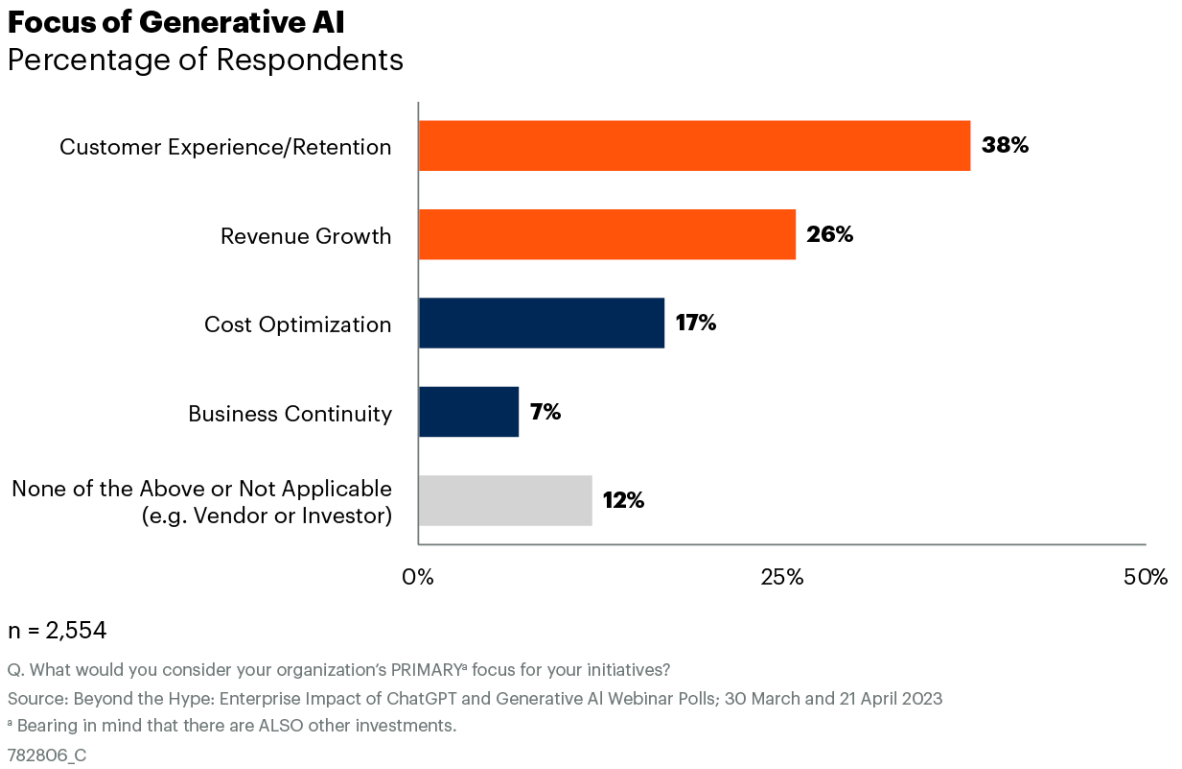
The most common use cases for generative AI can be categorized as follows:

- **Text generation** — The output of the model is text based on (but not limited to) natural language. This is the primary use case for LLMs based on Transformer-models such as GPT. Use cases include:
  - Creating and redlining code and configuration
  - Answering questions
  - Translating between languages (human or computer)
  - Generating prose
  - Summarizing text
  - Describing the content of an image
- **Image generation** — The output of the model is digital image data or 3D model representation data. This is a common use case for GANs and diffusion models. The use cases include:
  - Image generation from text description and/or image-based input
  - Synthetic image data generation for training discriminative models
- **Generative design** — The output of the model (or the software that uses it) is a design or engineering artifact, for example a 3D model, circuit diagram or user interface definition. For example, neural radiance field (NeRF) models can generate 3D renderings from collections of 2D images. Use cases include:
  - Designing web and mobile app user interfaces
  - Generating 3D environments and modeling assets from text descriptions
  - Generating 3D environments and modeling assets from 2D inputs

- **Audio generation** — The output of the model is digital audio, for example voice, music or recognizable sounds. The model types used include GANs, transformer models and VAEs. Use cases include:
  - Converting text to speech
  - Mapping or cloning voices from an audio sample
  - Generating music from text or an audio sample
- **Structured data generation** — The output of the model is data-specific to the intended domain. Use cases include:
  - Synthesizing data for testing, or ML training
  - Generating potentially useful protein structures and drug formulations for scientific research

In the Beyond the Hype webinar poll, participants identified the primary focus for their organizations as they evaluate and adopt Generative AI, with the top priority being customer experience and/or retention, as shown in Figure 3. [1]

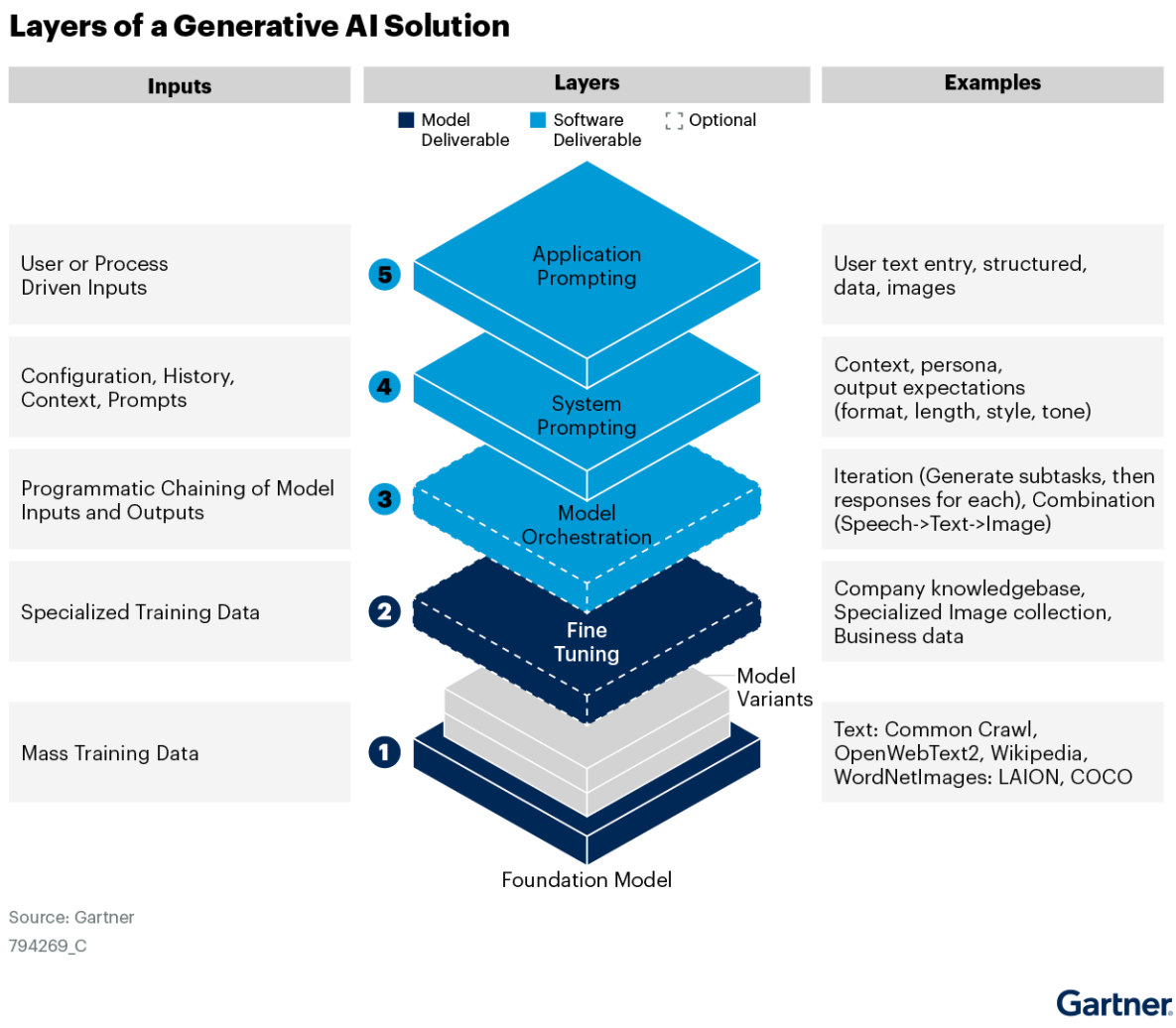
Figure 3: Focus of Generative AI



The Layers of a Generative AI Solution

To shed some light on how generative AI applications are implemented it is useful to unpack the logical layers of a simple solution, as shown in Figure 4.

Figure 4: Layers of a Generative AI Solution



Working from the bottom up, Figure 4 shows the key implementation decisions that need to be made to weave these tools into an application.

1. **Foundation models** are trained on the source dataset, often a vast corpus of information. Model variants give users a choice of capability balanced against cost and performance. [2, 3, 4] Multimodal models are also starting to emerge, with the ability to support combinations of text, images, audio and other data as inputs or outputs. [13, 14, 15]

2. **Fine tuning (optional)** can be used to specialize an existing model to work well for a specific task or domain without having to retrain the whole model. This requires you to provide additional training data organized and structured to suit the type of model (Note: this capability is not specific to generative AI — discriminative AI models can also be fine-tuned).

3. **Model orchestration (optional).** It may be necessary to combine models, for example discriminative AI model and generative AI models or multiple generative AI models (e.g., LLM and diffusion-based image generation) to solve the problem. This orchestration is typically coded using your preferred programming language. The outputs of one model are used as inputs to one or more others.

4. **System prompting.** Besides parameter-based configuration, generative AI models that accept natural language input can be guided by providing text-based prompts. These prompts can be a combination of natural language and structured data, and are combined with application prompts at runtime.

5. **Application prompting.** Your application code will send prompts to the model. These prompts could be direct input from a user or composed from application or process data. They are combined with the system prompts to help guide the output to meet the system requirements.

For more detailed exploration of prompting patterns and practices see Step 4 — Prepare and Test Your Chosen Model.

The Key Challenges of Generative AI Solutions

Generative AI approaches share many of the challenges associated with more established ML when they are integrated into production applications and processes:

- Bias in the training data creates bias in output that affects user outcomes
- Model life cycle management is critical to deal with changes in model behavior between versions
- Model drift means that models become less relevant over time (e.g., OpenAI's GPT 3 and 4 models are trained on datasets where the most recent data is currently from late 2021)
- Training, fine-tuning and inference costs remain at a premium for the best performance.

For more details of these challenges see Incorporate, Test, Deploy and Maintain Machine Learning Models in Production Applications.

Generative AI also introduces new challenges that application architects must take into account as they assess how it can be integrated into business solutions, these include:

- **Unresolved copyright and intellectual property concerns** [5]
  - Models are frequently trained on training data that includes copyrighted material, including text, images and code.
  - Vendors of ML services or LLMs do not indemnify users against plagiarism or copyright infringement.
  - Many open-source models use licenses that limit or prevent their usefulness in commercial solutions, including the GNU General Public License (GPL). Some models, including the recently released Falcon model, are licensed using permissive open-source licenses such as Apache 2.0 and MIT. The Responsible AI Licenses (RAIL) initiative is working to define licenses to specifically address AI use cases.
  - Users of generated content may be held responsible for validating that generated text or code is not plagiaristic and/or a copy of protected material.

- **Training and inference costs**
  - Training AI models is a compute-intensive activity, and has unpredictable outcomes. Application architects will need to collaborate with the data scientists and engineers that are responsible for managing the training data and orchestrating the training process. Because training is commonly done on a low-frequency cadence, most organizations will rely on cloud services for training resources so that they only pay for what they use. The cost of these resources creating and iterating on a model's configuration is closely related to the size, complexity and, if well designed, the capability of the resulting model. See How to Securely Design and Operate Machine Learning for more details.
  - Deploying models for inference is less resource intensive than training, but models must be loaded into RAM and the best performance is offered by using GPUs or AI-optimized processors. Delivering high performance from larger more capable models associated with Generative AI requires these high performance compute resources and the associated hardware or cloud service costs.
  - Operations expenditure (opex) for model inference is commonly proportional to demand, meaning that it can be challenging to predict the running costs of a solution before implementation. A successful implementation that drives high usage will also drive high costs, which could force a review of architecture and choice of model or model provider.

- **“Trial and error” design processes**
  - AI and ML development remains a relatively unpredictable activity. Variables including the dataset, the processing of the dataset (including filtering or tokenizing), the design of the ML model itself, the parameterization of the model and the number training iterations all affect the resulting model. Add to this the variables affecting inference, including configuration, prompt engineering and the nondeterministic(*) nature of the output from inference, and the result is a design and delivery process that relies on a lot of iterative “trial and error” to refine deliverables to meet acceptance criteria.

- **New testing challenges**
  - Software testing typically relies on validating the output of a set of test cases that match the expected results. The nondeterministic nature of AI, and the unstructured and often “creative” nature of AI generated text, images and audio make testing a much more subjective undertaking. Testing strategies need to detect the likelihood and impact of errors in output (hallucinations) and the quality of the resulting content.

- **New security challenges**
  - As new technologies become popular they present a more interesting target to attackers eager to disrupt or subvert applications built with them. Generative-AI-based tools have already been attacked to expose their underlying configuration. [6] The unstructured nature of system and application prompts in generative AI that uses natural language prompting makes this particularly hard to defend against (see 4 Ways Generative AI Will Impact CISOs and Their Teams). [7]

(* nondeterministic — providing the same input does not result in the same output)

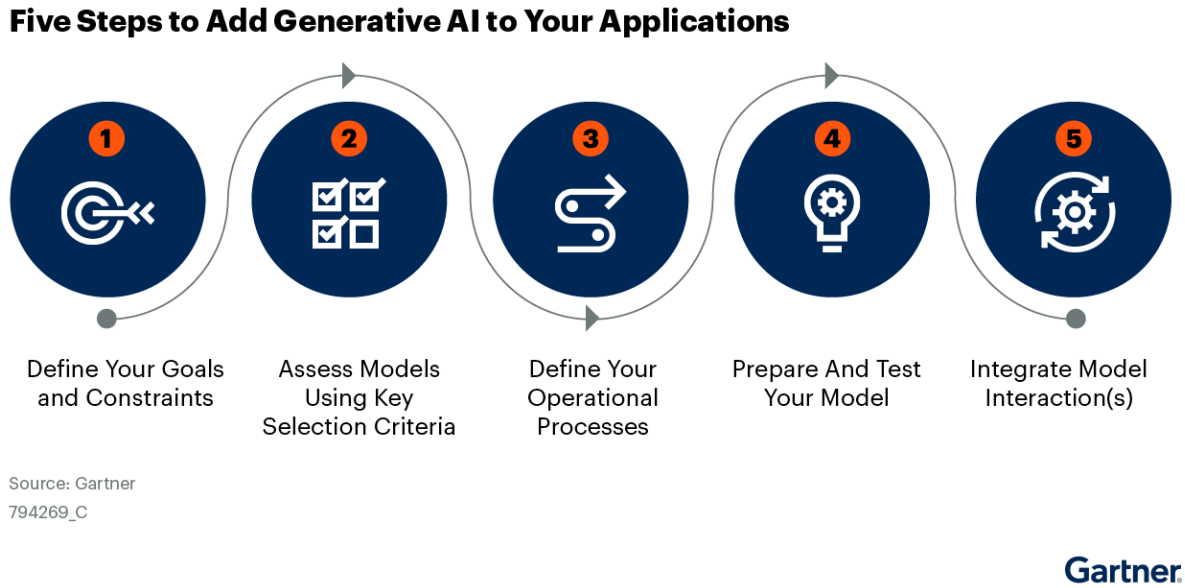
As you assess how generative AI could be applied to the needs of your customers, users and stakeholders, you should balance the promise of new capabilities against the complexity, costs and constraints of dealing with these challenges.

Fundamental Steps for Integrating Generative AI-Based Capabilities Into Your Application

The following steps apply to integrating all types of Generative AI described above to meet a relevant use case (see Figure 5). These steps have strong similarities to the steps for adopting ML in general (including discriminative AI models) as described in Incorporate, Test, Deploy and Maintain Machine Learning Models in Production Applications.

The steps are not a complete guidance framework. Rather, they are intended to highlight the key considerations application architects need to build into their planning should they embark on evaluating and using Generative AI capabilities.

Figure 5: Five Steps to Add Generative AI to Your Applications



Step 1 — Define Your Goals and Constraints

Informal experimentation is valuable as a learning process, but when you intend to integrate generative AI into applications for production use it is important to be clear what your goals and constraints are. This will help you set expectations, avoid wasting time on "cool" solutions that don't deliver value, and stay on track as you navigate an already diverse ecosystem that is evolving at a dizzying pace.

Start by answering these questions:

- **What problem are you trying to solve?** What blocker of value delivery are you trying to remove? What new value are you trying to unlock? Any hyped and potentially disruptive technology can lead to the "solution looking for a problem" situation. Ensure you are focused on delivering a business outcome, not implementing a technology.
- **Who will be affected by the solution?** Generative AI solutions frequently manifest in the user experience or in automation of processes. Understanding which user roles will be impacted by the features is critical to ensure that you ensure the proper change management is in place. Understanding the user population and their likely usage patterns of the solutions' generative AI features will also help you estimate the opex for model inference.
- **What constraints do you have?** For example, do your goals involve sensitive data or intellectual property that needs to be protected, impacting architecture and service selection? Performance or scale requirements also need to be balanced against budget constraints and ROI.
- **How will generative AI influence your user experience (UX)?** Having generative AI capabilities in your application is pointless unless the UX incorporates them in a way that aligns to users' needs. For example, a chat-based interface may be "cool" but may not be as efficient for users accessing certain types of information; additionally, presenting information using generated natural language may be less intuitive than a structured format. Also, not all generative AI use cases will be exposed directly via the user interface, but UX may be impacted by changes to processes and decision making in the back end of the application.
- **Is a generative AI model the best approach?** While generative AI capabilities have massive hype and momentum, they come with costs and complexity that may not make them the best choice for use cases where alternative proven solutions exist. Avoid using a generative-AI-based solution when there are better, less challenging solutions to the problem. For example, you should not use generative AI to summarize articles when simply using the first 300 characters of the article body provides good enough results.

Step 2 — Choose Models Using Key Selection Criteria

There are an increasing number of generative AI model sources, capabilities, structures and operating models that you will need to assess as you select the technology to integrate into your applications.

Model Sourcing

There are many ways to access these capabilities and integrate them into your applications. Your goals and constraints from step 1 will influence how you source your generative AI models. The most common sources of generative AI models are:

- **Embedded in platforms** — Software vendors are adding generative AI capabilities to their products. For use cases that relate to specific application markets, sourcing generative AI capabilities from a vertical platform provider may be the simplest and optimal solution. For example, if your goal is to streamline call-center operations and improve call handling, generative AI features from your call-center technology provider could be a good solution rather than trying to train and integrate your own models. These capabilities may be accessible via the provider's APIs, allowing you to integrate them into other applications or processes without taking full responsibility for model integration and maintenance.
- **Cloud providers** — The major cloud providers have long provided support for developers using more traditional styles of AI and ML (see Magic Quadrant for Cloud AI Developer Services). These are being augmented with generative AI capabilities including: API access pretrained models, support for fine-tuning models, support for inference hosting of customer trained models or back-end party models, and model marketplaces (sometimes referred to as "model gardens" or "model zoos"). The cloud providers differentiate from pure-play providers by offering integration with their platforms, including identity services, encryption key management, private networking, monitoring, billing and support for deployment to more cloud regions. In some cases they also provide additional assurances about data locality and access policies.

- **Pure-play providers** — As with any emerging market, much of the innovation can be found among providers focused on the market. Pure-play providers may offer more differentiated, focused or specialized solutions, and may offer lower costs or more flexible pricing models. Pure play vendors include:
  - AI21 Labs (text generation and comprehension)
  - Anthropic (AI assistant)
  - Cohere (text search, summarization and generation)
  - MosaicML (model training and inference services)
  - OpenAI (AI assistant, text and image generation)
  - RunwayML (image and video generation)
  - Tonic.ai (fake data generation)

- **Open source and commercial models** — Since trained ML models are static data they are relatively easy to package and distribute. Various organizations publish open-source models that can be found through model marketplaces such as Hugging Face, PyTorch Hub and TensorFlow. Cloud providers are also beginning to offer access to OSS and commercial generative AI models within their marketplaces, for example AWS Bedrock, AWS SageMaker and Google Vertex AI Model Garden.

- **Fine-tuned models** — An existing OSS model (with appropriate licensing) or commercially provided model may be used as a basis for creating a new more specialized model without the need to completely retrain them. Fine-tuning allows you to reuse and benefit from the core capabilities learned by a model trained on a larger, but more general dataset. The process uses your own more limited and specialized training data to create a new specialized model more cheaply and quickly than retraining the model from scratch. For LLMs, the fine-tuning data required is typically in the form of a set of additional prompts, inputs and sample responses (often in JSON format). When considering fine-tuning a model you must assess the costs of the fine-tuning process (based on the amount of training data used and/or compute time required) and hosting the resulting model for inference. Note that generative AI platforms may charge a premium for inference using fine-tuned models. For example, at the time of writing, OpenAI charges approximately between four times and six times the cost per thousand tokens for inference using a fine-tuned model when compared to the equivalent base model (Pricing). As another example, Microsoft charges the same for inference using pretrained or fine-tuned models, but fine-tuned models incur an additional per-hour hosting fee of between $0.05 USD and $3.00 USD per hour depending on the base model used.

- **Training your own model** — While outside the scope of this research your organization may have training data and specific requirements that justify training its own generative AI models from scratch. Data science team(s) typically manage the training data, model design and training. The resulting model will be hosted for inference in the same way as a pretrained or fine-tuned model. An example of this approach is the BloombergGPT model. [12]

Model Capability

Generative AI models are large data structures that need to be held in memory for efficient inference, so the size of the model dictates the minimum system memory required. Inference is a computationally intensive process that is highly dependent on the processing performance of the host environment (e.g., CPU and GPU performance).

This means that choosing the right models involves a balance between model capabilities good enough to meet your requirements and the cost of hosting that model.

Finding this balance is more art than science. Some providers recommend starting with the most capable, and hence most costly, model first to confirm the type of model that meets your needs. [8] You can then try less capable models to optimize cost or performance. Alternatives include running side by side comparisons (for example using tools like Prompt Compare) to assess how different models and configuration affect the output.

The nondeterministic nature of generative AI models makes this assessment more challenging, and it is important that any assessment is made by assessing multiple responses for each set of inputs. For example, a small empirical study comparing the responses of ChatGPT and humans for a text classification task discovered not only some statistical disagreement between humans and ChatGPT, but also surprisingly wide disagreement between multiple runs of ChatGPT for the same task. [9]

> **Do not take one good output as evidence that a generative AI model will consistently generate good output**

Selecting the right model is further complicated by the importance of prompt engineering — a simpler (and hence cheaper) model may be able to provide suitable output if provided with the right prompts.

We recommend you assess models by:

- Classifying your problem and choosing the right type of model (e.g., a chat model, a text completion model, an image generation model, etc.)
- Defining evaluation test cases that are representative of the tasks you need the generative AI to deliver as part of your integrated solution. This includes defining "ground truth" to capture what the correct or optimal output of the model is in specific situations.
- Execute these tests against candidate models using multiple runs of each test combination to expose the impact of nondeterministic output. Test cases should also include adjusting model parameters and system prompt content for each test prompt.

Model Integration

Generative AI models are ultimately hosted by some code, typically using a model serving framework, that provides inputs to and receives output from the model. Service providers may own and host this code, providing APIs or you may need to build and host it yourself. This gives you a variety of choices for integrating generative AI models into your applications and processes:

- **"Model as a service"** — These services provide API-based access to generative AI models that they train, maintain and operate. Examples include AI21Labs, OpenAI, Microsoft Azure OpenAI Service, Google Vertex AI, IBM Watsonx.ai (preview)
- **"ML platform as a service"** — These services provide API-based access to platform services that streamline the hosting and operation of generative AI models, including models that have been sourced from the community, fine-tuned or trained outside the platform. These services also typically provide support for model training and management. Examples include Microsoft Azure Endpoints for Inference in Production, Amazon SageMaker Real-Time Inference Endpoints, Hugging Face Hosted Inference API, Google Vertex AI Model and Endpoint Components, MosaicML Inference and IBM Watsonx.ai (preview).
- **Self-hosted inference API** — If you want more control over the hosting location, connectivity and compute resources you can use a model serving framework (such as ONNX Runtime, TensorFlow, TensorRT) to wrap your models in an appropriate service implementation and expose it as an API. This gives you control over the API format and contract and allows the service to incorporate additional processing of the model inputs (e.g., to enrich them with data from other sources) and outputs (e.g., to transform, present or combine them as needed). Developers and integrators will use their preferred approach to calling APIs to invoke the model.
- **API developer frameworks and platforms** — Developers of generative AI applications often follow specific patterns. Frameworks and platforms are emerging to streamline the implementation of these patterns. These tools also provide an abstraction of the technical complexity in dealing with "model as a service" APIs, reduce the coupling to specific models and services and provide a simpler developer experience. Examples include Deepset.ai Haystack AnswerGenerator, Dust, Gooey.AI, GradientJ, Hugging Face Diffusers, LangChain, LlamaIndex, Microsoft Guidance, Microsoft Semantic Kernel (for search-based use cases), PromptChainer and Steamship.

- **Application embedding** — Inference frameworks such as Apple CoreML, Google ML Kit and TensorFlow Lite, as well as the same inference frameworks listed for the self-hosted inference API approach, can be used to embed the model invocation with your application code. However this creates a closer coupling between the model and your application code, and in a fast-evolving ecosystem like generative AI you should take care to ensure that you decouple the model invocation logic so that it can be more easily updated with minimal impact on the rest of the codebase. This model is most relevant in scenarios where you want to integrate generative AI models into software that is deployed into devices that do not have reliable network connectivity and cannot support process-level service isolation (i.e., to host the model as a local service).

See also Magic Quadrant for Cloud AI Developer Services and Critical Capabilities for Cloud AI Developer Services.

Step 3 — Define Your Operational Processes

It is crucial to define specific operational processes when integrating generative AI into your applications. The drivers for this are:

- **Cost management** — Generative AI inference comes at a high computational cost compared to traditional processing, often demanding large memory capacity and GPU or custom silicon support for performance. It is easy to create high value solutions that also have high costs as their usage is scaled up.
- **Nondeterminism** — Since you cannot accurately predict the output of a model for a given input, it is important that you are able to assess whether its results are within acceptable levels, and remain at acceptable levels over time (as the model is updated, prompts are changed, or user behavior adapts).
- **Rapid evolution** — The pace of innovation in the generative AI market is high, with both megavendors and startups vying for position and market share. With such evolution, the best solution at a point in time is likely to be superseded quickly and the applications you build around generative AI capabilities will have the opportunity to evolve quickly too.

- **Privacy and intellectual property** — Many generative AI solutions will present users with the opportunity to provide text or data as part of their input or prompt. You need to ensure that data submitted to third-party generative AI services meets your policies on data sharing, residency and processing. You may also need to validate that the output does not violate the copyright or other intellectual property rights of data that was included in the training data.

- **Security** — Preventing abuse of LLM-based applications has already proven to be challenging, with prompt injection attacks demonstrated against many of the leading LLM-based chat providers. This includes exposing the details of your system prompts and finding ways to coax the model into producing unintended outputs (for example, offensive content). Defending against prompt injection attacks has proven particularly difficult where the input is natural language. As a result, there are many variations of the language to achieve the attack so pattern matching to block malicious prompts is only a partial solution.

To address these drivers, you must ensure that your operational processes relating to applications with integrated generative AI include the following:

- **Active cost/capacity monitoring** — this is particularly true for "pay as you go" services where cost is directly proportional to demand. At low usage, your application may generate trivial levels of traffic to generative AI services, but at scale those costs may spiral beyond the costs of the traditional components.

- **Model version management** — Since generative AI models are evolving rapidly you will need to define processes to manage changes to the model. A model change may improve or reduce performance for a given task and configuration. Also, a model may need to be fine-tuned on more recent data introducing another variable to the model behavior. This means you need processes defined around your chosen model source to evaluate the impact of a change, and to use an improved model where appropriate (e.g., to reduce model drift). Your version management will need to account for the model source and hosting model, and you should review model providers' life cycle documentation. For example OpenAI models are "being continuously updated" and static (historical) versions of a model are only available for three months after the introduction of a new model version. [10]

- **Prompt and result logging** — Logging, monitoring and analyzing model inputs and outputs is necessary to assess and track the effectiveness of the solution over time, and to provide feedback on user behavior. Analysis of generative AI prompts can provide useful insight into user expectations and experience that can be fed into future system increments where it can influence model selection, model configuration and system prompting. [11] The same logging also provides a mechanism for auditing the flow of information to the model in support of data privacy and intellectual property policies. However, you must also ensure that the content of these logs is appropriately filtered or masked to prevent the unauthorized collection or sharing of sensitive data (e.g., personally identifiable information [PII] or personal health information [PHI]).

How you implement these operational processes will be dictated by the source of your model and the overall architecture of your generative AI application.

Step 4 — Prepare and Test Your Chosen Model

Once you have selected a model (step 2) you must design and validate the set of model inputs that give the best results for your use case. These inputs include user- or process-derived data, but also include the model's configuration parameters and system prompts.

- **Model configuration** — generative AI models and services typically support one or more configuration options that influence the generated output. For example, these include:
  - **Output limits:** These constrain the output, for example the number of tokens for an LLM or the image resolution for an image generator.
  - **Temperature:** This drives the "randomness" of the result produced by the model. A value of 0 provides the lowest randomness but does not always mean the model will provide repeatable output for a given input. The general advice that the more creative you want the output to appear the higher the temperature parameter should be set (within reason, as setting it too high can lead to incomprehensible output in the case of an LLM).
  - **Top_p:** This influences the diversity of LLM output, a low value for top_p forces the model to select each output token from a smaller set of those with the highest probability of being a good fit.
  - **Top_k:** This influences the diversity of LLM output by defining the number of highest probability tokens from which the next token will be selected. Smaller values result in less variation.
  - **Seed:** Models commonly use a random number to initiate the generation process. When configurable, using the same seed with the same prompt will generate the same output. This can be used to create similar output (e.g., a set of similar images with slight changes to the prompt) "lock" in aspects of the output such as composition of an image while other stylistic aspects of the prompt are modified.

For LLMs, Hugging Face Transformer API Documentation provides a detailed set of parameters and explanations. For Image generation Midjourney Documentation and Stability.ai Documentation provide more detailed parameter explanations

- **System prompt engineering** — System prompts are part of the input to the model that you control. Their purpose is to guide, shape and limit the output of the behavior of the model for a given application or user prompt. System prompts are often static and used across multiple interactions to drive output consistency, but can also be dynamically generated, for example based on session, user or process context. Full coverage of prompt engineering and design is beyond the scope of this research but key elements of an LLM system prompt can include:
  - **Personality** — for LLMs providing a description of the persona the model should imitate in its response, for example their profession and experience but also more personal characteristics such as age or temperament.
  - **Context** — Providing information about the scenario, whether real or hypothetical, about the context the model should assume for the input. This can include specific information about the application, processes or data specific to the session or user that may influence or be included in the responses generated. An example of using context is to use it for few-shot learning in LLMs, where a small number of examples are used to introduce data that would not have been included in the models training data (e.g., proprietary data).
  - **Response format** — If you require the model to produce output of a specific layout or structure every time you can include examples of the format in the system prompt. This can be text layout examples, for example an email or article format, or more structured data formats such as examples of comma-separated data, JSON or XML documents. If requesting output in a structured format, be prepared for the output to be "almost right" some of the time, due to the nondeterministic nature of the model. You will need to implement format validation and correction or retry logic to detect and resolve these issues.
  - **Response level** — Describe the level of detail (or brevity) required for the output, if there is a specific length constraint that can also be defined, but it may not be strictly enforced so you should still anticipate receiving longer responses.
  - **Stylistic instructions** — Describe any additional requirements for the output, for example the tone of language,
  - **Negative prompts** — Explicitly describe the things you do not want the model to output. This could include not asking for sensitive information, or not generating images of a specific style.

When defining your system and application prompts you will have to take the overall prompt size into account. The maximum prompt size for a model is called its context window. Even when working within the context window of a model the number of tokens in the prompt affects both the performance and cost of inference.

In simple terms, you can view the entire prompt as the available "memory" available to the model. If your use case involves accumulating prompt context over a number of steps, iterations or turns, you may hit the token limits of the model and earlier context information may be "forgotten." Evaluate Emerging Patterns and Practices lists some of the patterns that are being used to work around these model limitations.

Caution: *Earlier prompts can be overridden by later ones. This opens up the possibility for a user prompt to accidentally or deliberately (a "prompt injection attack") instruct the model to output something different from that described in your system prompts. If your system takes user input (directly or indirectly) and combines it into the model prompt, you should take steps to check for language that might subvert the behavior of your model.*

Designing your applications interactions with the model is an iterative process that will involve testing alternative system and application prompt content and model configuration. Each iteration needs to be tested multiple times to evaluate its impact on the quality and content of the output of the model. The qualitative assessment typically requires statistical analysis of the outputs across multiple runs and configurations to identify the "best" design.

Prompt engineering practices are continuing to mature with various vendor and community tools and guidance for refining the process including [alphabetical]:

- DAIR.AI's Prompt Engineering Guide — An open-source collection of guidance and patterns for prompt engineering.
- DeepLearning.ai ChatGPT Prompt Engineering for Developers — A free online training tool that outlines an iterative approach to refining prompts of different types.
- Galileo's LLM Studio (Preview at time of writing) — A tool for managing the prompt development and testing process.
- Jina.ai's PromptPerfect — A tool for refining and iterating on, and comparing results during prompt engineering.

- Microsoft Azure Machine Learning Prompt Flow — Tools provided as part of the Azure platform
- Microsoft Guidance — An-open source framework for codifying prompts based on a prompt templating language.

We recommend you approach this as a discrete activity as part of the over design of your generative AI applications.

Step 5 — Integrate Model Interaction(s)

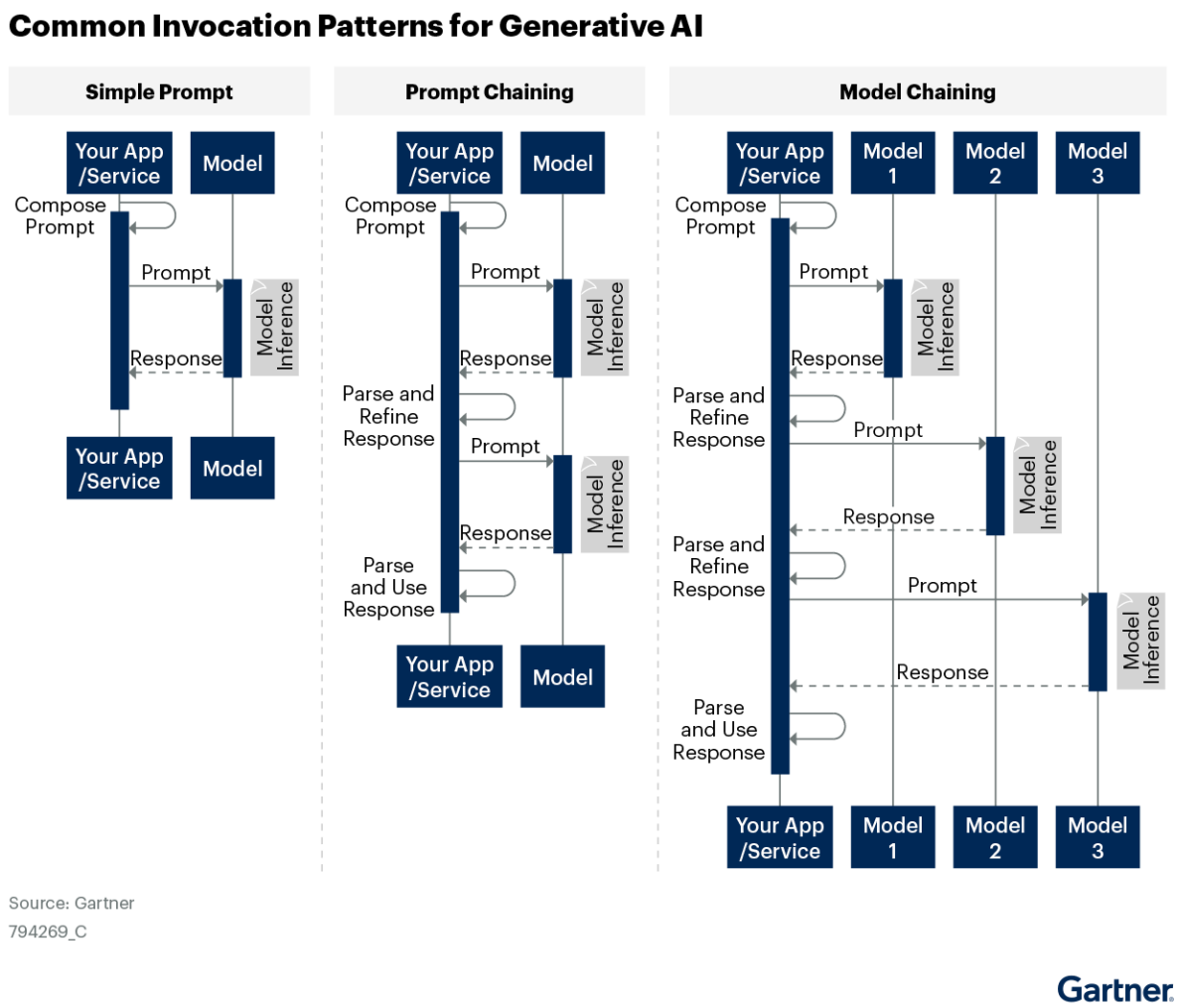
Delivering useful business outcomes using Generative AI means you need to integrate the interactions with the model into your applications, services and processes. The integration pattern will be strongly influenced (if not dictated) by the model source selected in step 2. The model and its API or framework abstractions will dictate the technical design of the integration.

For simple data generation scenarios, the interaction model will be a simple two-step prompting process.

- Push input data to the model (e.g., user prompts, application data prompts, contextual data)
- Process response (including metadata)
  - Parse, validate and manipulate the response programmatically (if required)
  - Present, store or combine the response with other data within your application

More advanced use cases will involve more sophisticated patterns including prompt chaining, model chaining as shown in Figure 6.

Figure 6: Common Invocation Patterns for Generative AI



Evaluate Emerging Patterns and Practices

Building solutions using generative AI is a rapidly evolving practice, with new tools, new patterns and new ideas emerging almost daily. During the course of this research we have identified the following emerging patterns and practices that warrant investigation and your time to understand the problems they solve so that you can assess their relevance to your use cases:

- **Zero-, one- and few-shot learning** — These are common prompt engineering patterns to guide general-purpose LLMs for specific use cases. They can be used as a simpler (and less accurate) alternative to model fine-tuning or custom model training:
  - Zero-shot learning relies on the LLM to provide an appropriate response based entirely on its training.
  - One-shot learning uses prompts that provide a single example of the request/response format the model should expect and can enrich this with a description of the logic the model is expected to use. This guides the model to offer responses that more closely fit the users expectations.
  - Few-shot learning uses prompts that include a small number of examples to provide stronger guidance to the model and introduce more data the model may not have seen in its training dataset.

- **Model abstractions** — These frameworks decouple your code from specific models and supporting tools allowing your applications to switch models with minimal change to code and configuration. As the market is rapidly evolving this can be a powerful way of isolating your functional logic from the specific models and services, and support more rapid evaluation and evolution of solutions.
- **Multimodal model chaining** — Model chaining (see Figure 6) does not need to use the same type of model at each step, multimodal model chaining combines models using different input formats at each step. For example, an image can be input into a model to generate an English text description. This text can then be input into a translation model converting it to Spanish text, which can then be used as the prompt for a text-to-voice model generating an audio description. The user sees a solution that generates Spanish audio descriptions of their images.
- **Programmatic prompt engineering** — Rather than using a purely natural language prompt, include technical details of the structure of input the model should expect and output it should generate. Application code processing the structured output must be ready to deal with deviations from the requested format.
- **Retrieval augmented generation** — An alternative to fine-tuning or prompt-based contexts. This approach avoids token limits and content limitation LLMs, and provides a way to use the LLM to more naturally present your own data and content. [16]

- **Chain of thought** — A task-oriented decomposition pattern used in prompt engineering. Commonly used with prompt chaining to first direct the LLM to generate a task breakdown for solving a broader request, then using each task as an input to get a more detailed response.

Recommendations

As you evaluate Generative-AI-based technologies and consider how to integrate them into your applications you should:

- Spend time to learn how generative AI models of different types work and which capabilities they can offer to your applications. A solid understanding of the technical fundamentals is essential to avoid misaligned expectations during planning and execution of your generative AI projects. Use one or more ML services to create proof-of-concepts to reinforce your understanding of these capabilities.
- Pay close attention to how you source generative AI capabilities to meet both your functional and nonfunctional requirements, including cost, data protection and residency, security and performance. New models and new vendors emerge almost daily and services from larger vendors are advancing rapidly.
- Keep your applications loosely coupled from whatever source of Generative AI models you choose. Due to the rapid evolution in the market, you need to avoid being deeply locked into a specific model, API or vendor if your application has an intended life span of more than six months.
- Invest time and resources into learning prompt engineering practices and patterns. The basics that users learn through interacting with user-facing services such as ChatGPT, Bard, Bing, DALL·E 2 and Midjourney are a useful basis, but are only scratching the surface of the sophistication needed for advanced system prompting.

Conclusion

Integrating generative AI capabilities into your software solutions involves learning a lot of new terminology, tools and skills. The well-established principles of modern software architecture and cloud services can help you embrace and evolve these capabilities as the market, and new patterns and practices emerge. However, the probabilistic and nondeterministic nature of AI models introduces new challenges to engineering robust and sustainable solutions, including prompt engineering, testing and monitoring.

Evidence

[1] Beyond the Hype: Enterprise Impact of ChatGPT and Generative AI

[2] Models, OpenAI.

[3] Introducing PaLM 2, Google.

[4] Introducing LLaMA: A Foundation, 65-Billion Parameter Large Language Model, Meta.

[5] Generative AI Has An Intellectual Property Problem, Harvard Business Review.

[6] AI-powered Bing Chat Spills Its Secrets via Prompt Injection Attack [Updated], Ars Technica.

[7] Prompt Injection Attacks Against GPT-3, Simon Willison.

[8] Azure OpenAI Service Models, Microsoft.

[9] Can ChatGPT Replace UX Researchers? An Empirical Analysis of Comment Classifications, MeasuringU.

[10] Continuous Model Upgrades, OpenAI.

[11] A Prompt Log Analysis of Text-to-Image Generation Systems, arXiv.

[12] BloombergGPT: A Large Language Model for Finance, arXiv.

[13] ImageBind: Holistic AI Learning Across Six Modalities, Meta. [14] GPT4, OpenAI. [15] RadImageGAN: Multi-Modal Generative AI for Medical Imaging, NVIDIA.

[16] Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks, arXiv.

Recommended by the Author

Some documents may not be available as part of your current Gartner subscription.

Incorporate, Test, Deploy and Maintain Machine Learning Models in Production Applications

Magic Quadrant for Cloud AI Developer Services

Critical Capabilities for Cloud AI Developer Services

How to Securely Design and Operate Machine Learning