Google Cloud

Improving LLM reliability and performance: Prompt engineering, fine-tuning, RAG, and long context window techniques

September 17, 2024

Table of contents

Introduction

This paper discusses techniques that AI solution builders can use to improve the reliability and performance of large language models (LLMs).

Common techniques for model enhancements

In a 2024 Deloitte survey about generative AI use by executives, 33% cited a lack of confidence in results as a significant risk to adoption[1]. To mitigate this, 73% of high-expertise organizations are implementing processes to improve input data, while 67% are focused on improving the reliability of generative AI outputs[2]. These focus areas are crucial for advancing more pilots into production, because generative AI models require customization to enhance accuracy and align behavior with expectations.

In this context, inaccurate outputs refers to when a foundation model generates a response that differs from a user's target response (for example, content, format, or factuality). A user might use a foundation model to summarize a legal contract and important clauses, expecting a 2000-word summary that includes key details. Instead, the model might provide a short 200-word summary with basic information about the parties involved without including the necessary key clauses to augment workflows and capture value adequately. Shortcomings like this can slow scaling or adoption in an enterprise. However, by modifying the query or underlying model, solution builders can adapt the solution's behavior to generate desired responses.

Similarly, out-of-the-box models often lack the proprietary or domain-specific data to generate accurate responses to complete a task, because they rely on the parametric memory formed during training. This means that the models can only generate responses based on the data they were trained on, which might not include information relevant to a particular use case. For instance, if a customer support chatbot using just a foundation model is not trained on a corporation's unique policies, it will not be able to answer specific policies and risk misinforming the end user. Further, even though foundation models exhibit reasoning and knowledge capabilities to pass specialized exams like the United States Medical Licensing Exam, they often struggle with novel problems that are outside their training data[2,3]. However, by making relevant data accessible, models can retrieve the information they need to ensure accuracy for new or existing problems.

[1] Deloitte State of Generative AI in the Enterprise Quarter 2 Report
[2] Large Language Models Encode Clinical Knowledge
[3] Reasoning or Reciting? Exploring the Capabilities and Limitations of Language Models Through Counterfactual Tasks

To improve solution performance, AI solution builders – product managers, developers, AI engineers, and IT managers – can incorporate a combination of *prompt engineering*, *retrieval-augmented generation (RAG)*, *fine-tuning*, and *long context window* techniques into the solution design. Each technique has its tradeoffs and limitations which can vary by use case, meaning solution builders need to evaluate solution quality on a per-use-case basis to understand what works in their situation. By understanding these techniques and adopting an evaluation-centric development approach, solution builders can effectively address technical challenges, enabling the deployment of more solutions into production.

This paper focuses on the following techniques for enhancing generative AI performance::

- **Prompt engineering[4]:** Designing and optimizing prompts to improve the likelihood of completing a task accurately and in the style or format the user or designer intended. By carefully crafting prompts, you can provide the model context, instructions, and examples that help it understand their intent and respond meaningfully.
- **RAG[5]:** Combining the strengths of traditional information retrieval systems (such as databases) with the capabilities of foundation models. By merging extra knowledge that might not have been in the models' training data, for example, proprietary enterprise data, with models' language skills, you enable the system to create more accurate, up-to-date, and relevant outputs for your specific needs.
- **Fine-tuning[6]:** Adapting foundation models to perform specific tasks with greater precision and accuracy. This technique works by focusing the model on specific downstream tasks by introducing tailored datasets, which might include both domain-specific and task-specific schemas, thereby which can enhance model performance for those needs.
- **Long-context-windows[7]:** Exploiting a model's ability to handle and process large amounts of information within a single prompt, allowing for the consideration of more detailed and varied content to generate better responses.

These techniques can be combined to address the challenges faced by enterprise use cases to create applications ranging from writing assistants[8] to fully autonomous customer support agents[9]. For example, to build fully autonomous customer support agents, a solution builder might choose RAG for accessing guidelines, prompt engineering for formatting responses, and fine-tuning to enhance tool utilization. Selecting the most appropriate technique or combination

[4] Google Cloud: What is Prompt Engineering
[5] Google Cloud: What is Retrieval-Augmented Generation?
[6] Google Cloud: Introduction to Tuning
[7] Google Cloud: Long Context
[8] Google Cloud: Everyday Case Study
[9] Google Cloud: Call Center Studio Case Study

of techniques involves making a series of tradeoffs between cost, model performance, and technical performance. To effectively build solutions, you can follow an evaluation-centric development approach by implementing a proof-of-concept (PoC), evaluating it, identifying gaps, and iterating until your requirements are met. This deep dive will provide an overview of each technique, discuss an approach for selecting techniques, and offer recommendations on which techniques are best suited for typical use cases.

Prompt engineering

Prompt engineering involves optimizing text to obtain better responses from a model[10]. This process often relies on trial and error to refine the prompts until the desired outcome is achieved[9]. Solution builders can integrate relevant data, or implement various techniques to enhance the model's accuracy, reasoning, and usability[9]. By making even these small changes to a prompt, you can have a large impact on the outputs[11]. Thus, prompt engineering requires an iterative approach to identify what works for your solution.

The following image describes the process for optimizing a Generative AI model through prompt engineering.

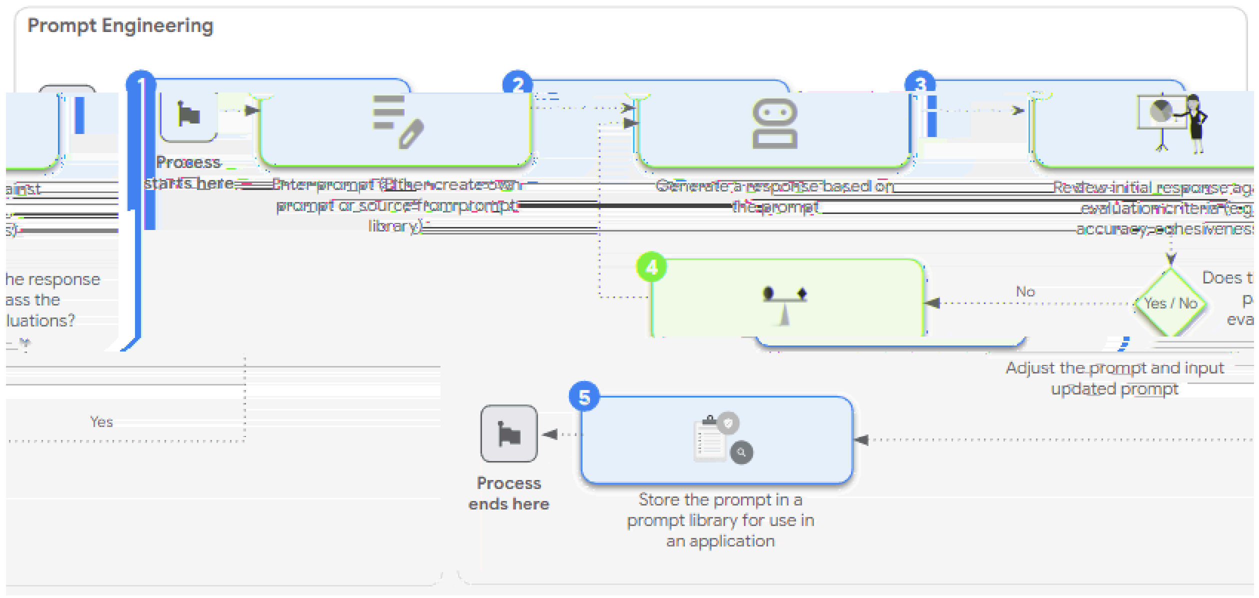

Prompt Engineering
1
Process starts here
Enter prompt (Either create own prompt or source from prompt library)
2
Generate a response based on the prompt
3
Review initial response ag
evaluation criteria (e.g
accuracy, cohesivenes
Does t
p
eva
Yes / No
No
4
Adjust the prompt and input updated prompt
he response
ass the
luations?
Yes
5
Store the prompt in a prompt library for use in an application
Process ends here

Figure 1: Process for optimizing a generative AI model through prompt engineering

[10] Google Cloud: Prompt Engineering
[11] Fantastically Ordered Prompts and Where to Find Them: Overcoming Few-Shot Prompt Order Sensitivity

Some of the most common prompt engineering techniques include the following:

- **System prompting:** Providing instructions to the model before any user input is processed to adjust the output, style (for example, corporate writing guidelines), or tone of the response[12].
- **Few-shot prompting:** Including example inputs and outputs (for example, clauses with corresponding labels, or product reviews with sentiment) within the prompt to guide the model on how to respond to similar inputs[10].
- **Chain-of-thought prompting:** Asking the model to explain its reasoning when answering to enhance its reasoning abilities[13].
- **Self-consistency prompting**: Asking the model to generate multiple responses to solve a particular problem and then prompting it again to find the most consistent answer from those outputs[14].

AI solution builders can apply best practices learned from other implementations by using prompt-design heuristics, such as Vertex Prompt Engineering. Although these techniques, practices, and guidelines are generally applicable, each model has its own style and sensitivities [15]. Thus, when changing models or new model versions are released, each prompt in a solution needs to be re-evaluated for performance changes.

Prompt engineering techniques can improve performance and may even outperform other techniques like fine-tuning, but they can be constrained by the context window size[16]. Even though context windows are expanding - for instance, 2 million tokens in Gemini 1.5 Pro - leading models can only process a fraction of enterprises' inherently large datasets.

Even when following a trial-and-error prompt engineering process, as described in "Enabling Generative AI Value: Creating an Evaluation Framework for Your Organization," to properly evaluate the success of each technique, solution builders must create datasets that represent their use cases. They should use sample inputs and outputs to minimize performance gaps. The datasets allow you to confirm that prompting techniques generalize and enable the use of techniques like *automatic prompt optimization*. Vertex AI Prompt Optimizer tests multiple strategies—such as chain-of-thought or self-consistency—and selects the best-performing approach based on one or more metrics[17]. Similar to hyperparameter tuning for predictive models, these techniques can reduce the number of iterations and expertise needed to find an

[12] Introduction to prompting
[13] Best practices for prompt engineering
[14] Self-Consistency Improves Chain Of Thought Reasoning In Language Models
[15] The Unreasonable Effectiveness of Eccentric Automatic Prompts
[16] Many-Shot In-Context Learning
[17] Teach Better or Show Smarter? On Instructions and Exemplars in Automatic Prompt Optimization

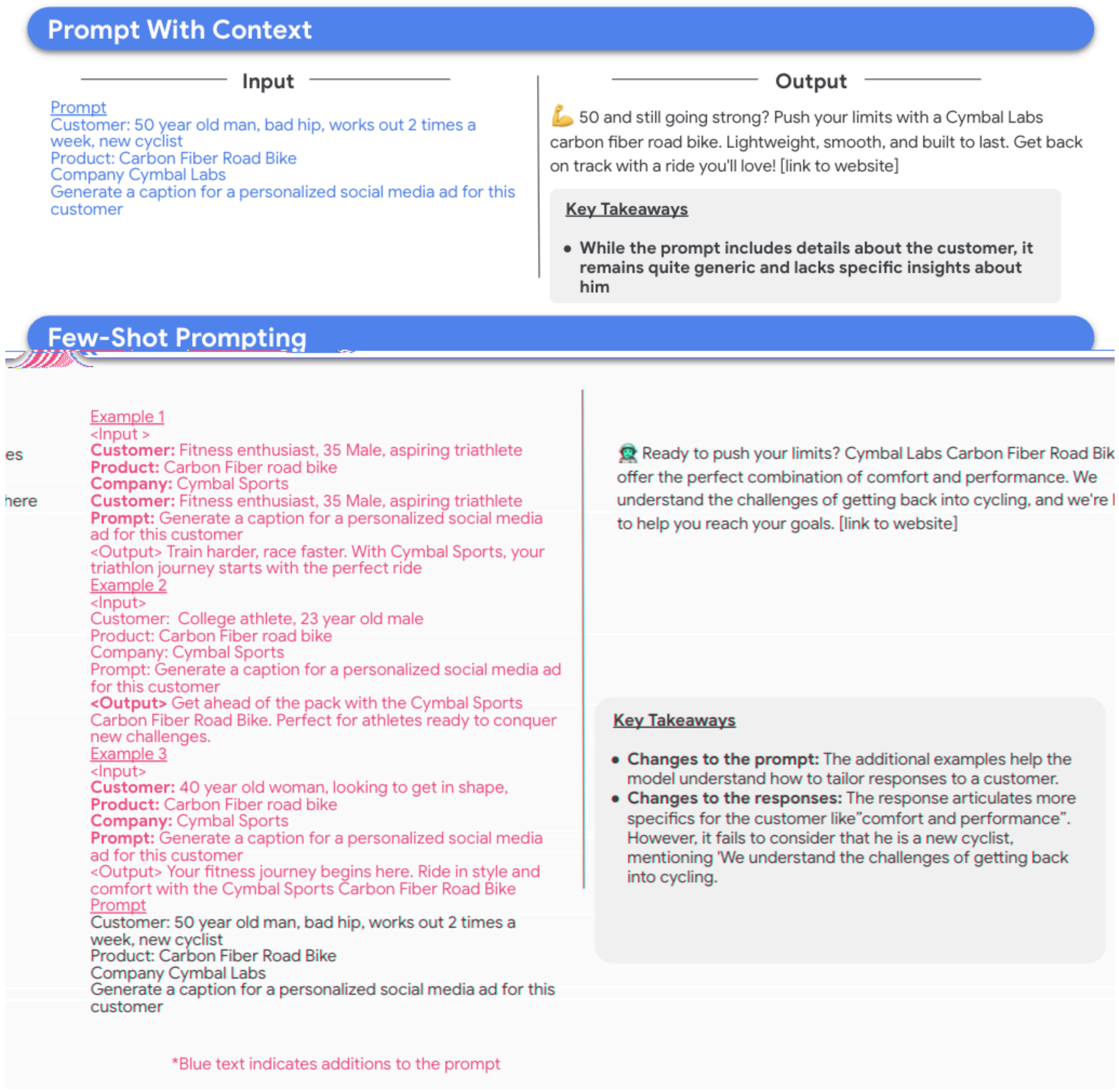
optimal solution.

An executive at a large health insurer illustrated the importance of evaluation for prompting, saying, “We use evaluations in every step of our prompt engineering process. This helps our engineers understand what is working and what isn’t. For instance, in our customer service applications, our engineers build generic prompts, evaluate the results, and create more specific prompts as needed.” This telescopic approach will allow you to test a single prompt for multiple intents, identify performance gaps, and refine the prompts until you achieve your desired performance.

Example: Advertising copy generation for enhanced marketing efficiency

A marketing agency wanted to improve its margins by reducing the amount of time and effort required to create copy for television adverts. The agency decided to build an advertising copy generator that incorporated its client’s brand guidelines, inspirations, and prior campaign copy to generate ideas and create content. The development team selected a foundation model suitable for their needs and began the process of prompt engineering to tailor the model’s outputs for their use case. Initially, they designed a prompt that incorporated the client’s style guide, campaign details, target audience preferences, and simple requirements for the advertisement.

This prompt is shown in the image on the following page:



Prompt With Context

Input

Prompt
Customer: 50 year old man, bad hip, works out 2 times a week, new cyclist
Product: Carbon Fiber Road Bike
Company Cymbal Labs
Generate a caption for a personalized social media ad for this customer

Output

💪 50 and still going strong? Push your limits with a Cymbal Labs carbon fiber road bike. Lightweight, smooth, and built to last. Get back on track with a ride you'll love! [link to website]

Key Takeaways

- **While the prompt includes details about the customer, it remains quite generic and lacks specific insights about him**

Few-Shot Prompting

Example 1
<Input >
Customer: Fitness enthusiast, 35 Male, aspiring triathlete
Product: Carbon Fiber road bike
Company: Cymbal Sports
Customer: Fitness enthusiast, 35 Male, aspiring triathlete
Prompt: Generate a caption for a personalized social media ad for this customer
<Output> Train harder, race faster. With Cymbal Sports, your triathlon journey starts with the perfect ride
Example 2
<Input>
Customer: College athlete, 23 year old male
Product: Carbon Fiber road bike
Company: Cymbal Sports
Prompt: Generate a caption for a personalized social media ad for this customer
<Output> Get ahead of the pack with the Cymbal Sports Carbon Fiber Road Bike. Perfect for athletes ready to conquer new challenges.
Example 3
<Input>
Customer: 40 year old woman, looking to get in shape,
Product: Carbon Fiber road bike
Company: Cymbal Sports
Prompt: Generate a caption for a personalized social media ad for this customer
<Output> Your fitness journey begins here. Ride in style and comfort with the Cymbal Sports Carbon Fiber Road Bike
Prompt
Customer: 50 year old man, bad hip, works out 2 times a week, new cyclist
Product: Carbon Fiber Road Bike
Company Cymbal Labs
Generate a caption for a personalized social media ad for this customer

🚴 Ready to push your limits? Cymbal Labs Carbon Fiber Road Bik offer the perfect combination of comfort and performance. We understand the challenges of getting back into cycling, and we're l to help you reach your goals. [link to website]

Key Takeaways

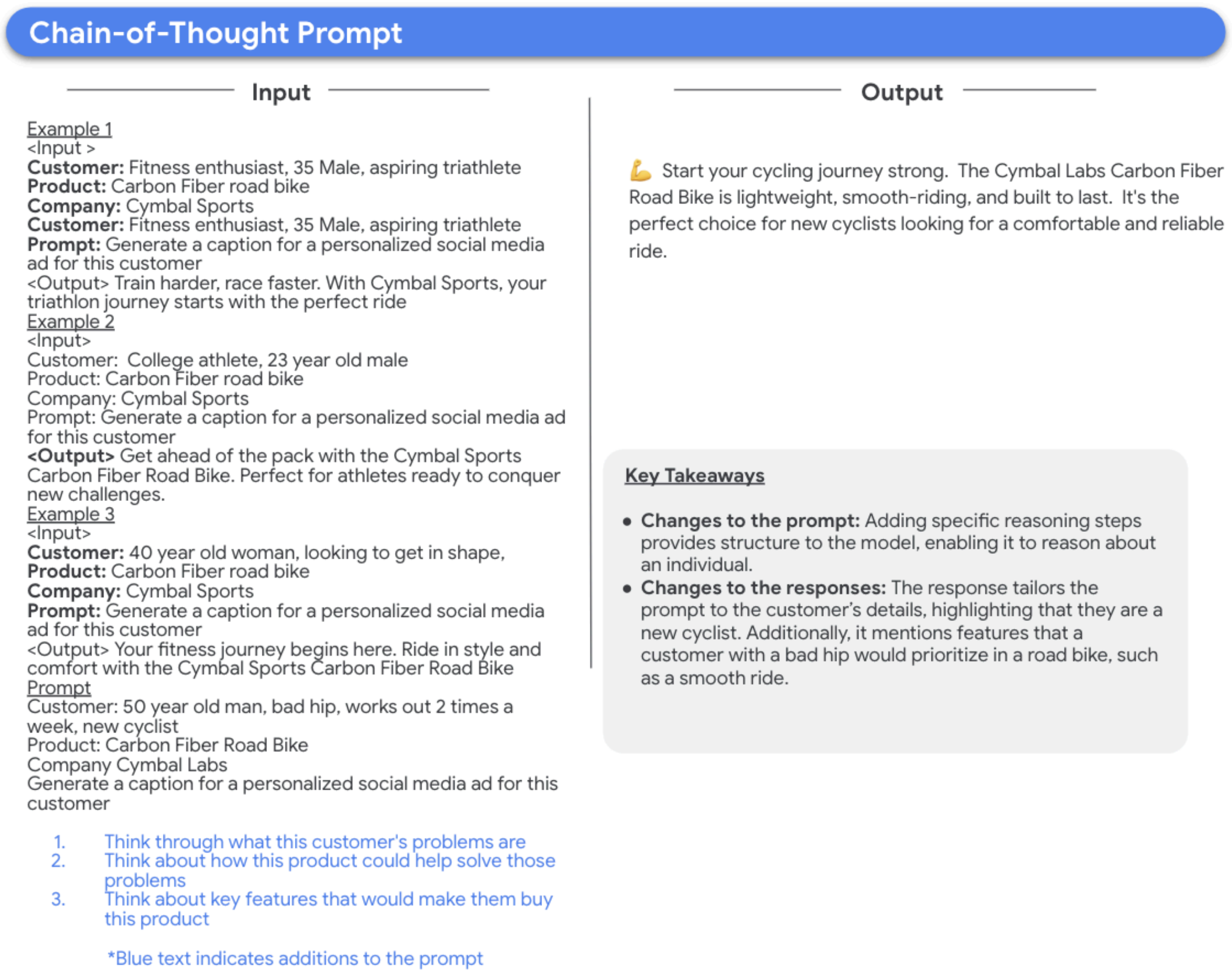
- **Changes to the prompt:** The additional examples help the model understand how to tailor responses to a customer.
- **Changes to the responses:** The response articulates more specifics for the customer like"comfort and performance". However, it fails to consider that he is a new cyclist, mentioning 'We understand the challenges of getting back into cycling.

*Blue text indicates additions to the prompt

Figure 2: Examples of a prompt with context and a few-shot prompt to personalize an advertising caption to a customer based on their needs

However, evaluations indicated that the solution lacked the creativity, accuracy, or cohesiveness specified in the requirements. To address this, the AI engineer used few-shot prompting, using five examples of input-output pairs within the context window. In subsequent evaluations, the model's creativity and accuracy improved but still lacked cohesiveness.

As shown in the following image, the solution builders then applied chain-of-thought prompting to enhance the model's reasoning and select the most consistent answer, thereby improving cohesiveness.



Chain-of-Thought Prompt

Input

Example 1
<Input >
Customer: Fitness enthusiast, 35 Male, aspiring triathlete
Product: Carbon Fiber road bike
Company: Cymbal Sports
Customer: Fitness enthusiast, 35 Male, aspiring triathlete
Prompt: Generate a caption for a personalized social media ad for this customer
<Output> Train harder, race faster. With Cymbal Sports, your triathlon journey starts with the perfect ride
Example 2
<Input>
Customer: College athlete, 23 year old male
Product: Carbon Fiber road bike
Company: Cymbal Sports
Prompt: Generate a caption for a personalized social media ad for this customer
<Output> Get ahead of the pack with the Cymbal Sports Carbon Fiber Road Bike. Perfect for athletes ready to conquer new challenges.
Example 3
<Input>
Customer: 40 year old woman, looking to get in shape,
Product: Carbon Fiber road bike
Company: Cymbal Sports
Prompt: Generate a caption for a personalized social media ad for this customer
<Output> Your fitness journey begins here. Ride in style and comfort with the Cymbal Sports Carbon Fiber Road Bike
Prompt
Customer: 50 year old man, bad hip, works out 2 times a week, new cyclist
Product: Carbon Fiber Road Bike
Company Cymbal Labs
Generate a caption for a personalized social media ad for this customer

1. Think through what this customer's problems are
2. Think about how this product could help solve those problems
3. Think about key features that would make them buy this product

*Blue text indicates additions to the prompt

Output

💪 Start your cycling journey strong. The Cymbal Labs Carbon Fiber Road Bike is lightweight, smooth-riding, and built to last. It's the perfect choice for new cyclists looking for a comfortable and reliable ride.

Key Takeaways

- **Changes to the prompt:** Adding specific reasoning steps provides structure to the model, enabling it to reason about an individual.
- **Changes to the responses:** The response tailors the prompt to the customer's details, highlighting that they are a new cyclist. Additionally, it mentions features that a customer with a bad hip would prioritize in a road bike, such as a smooth ride.

Figure 3: Example of a few-shot prompt with chain-of-thought prompting to personalize an advertising caption to a customer based on their needs

By using a structured prompt engineering and evaluation approach, this agency was able to identify gaps and adjust its strategy. Prompt engineering practices like this can achieve considerable performance improvements in a short period of time. As a result, prompt engineering is typically the first technique used to improve solution performance and is generally considered for all use cases.

RAG and grounding

RAG is the process of bringing specific information into the context window to generate more accurate, specific, and grounded responses[18]. It combines generative AI technologies, such as embedding models, with data stores, like vector databases or knowledge graphs[15]. Because foundation models are often trained on public data with training cutoffs, they lack access to internal knowledge (for example, customer data) and recent events[19]. By incorporating up-to-date data, RAG enables these models to generate responses grounded in relevant data from the enterprise, recent public data source such as the internet, and 3rd party data sets[20].

An effective mix of data stores for enterprises involves combining public search engines[21], enterprise systems, and a blend of structured and unstructured data sources. The specific use of each data store will depend on the solution, whether it's tapping into real-time internet data, accessing enterprise-specific information, or retrieving insights from relational databases and knowledge graphs. To maintain the reliability of the solution's outputs, it's essential that these data stores are continuously updated, ensuring that the system always accesses the most current and grounded information when generating responses.

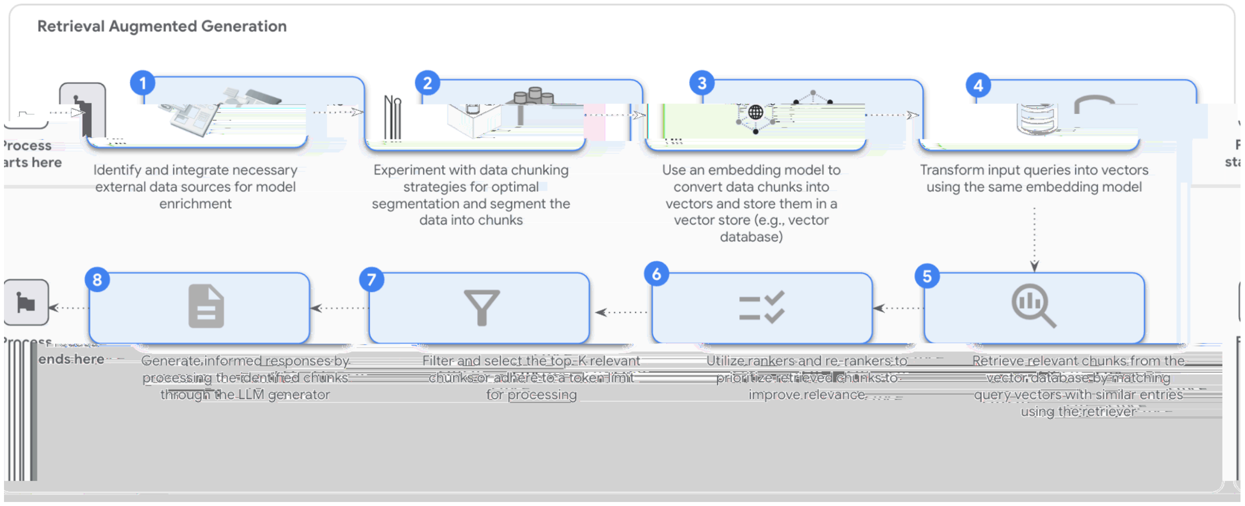

Retrieval Augmented Generation
rocess
arts here
1
Identify and integrate necessary external data sources for model enrichment
2
Experiment with data chunking strategies for optimal segmentation and segment the data into chunks
3
Use an embedding model to convert data chunks into vectors and store them in a vector store (e.g., vector database)
4
Transform input queries into vectors using the same embedding model
5
Retrieve relevant chunks from the vector database by matching query vectors with similar entries using the retriever
6
Utilize rankers and re-rankers to prioritize retrieved chunks to improve relevance
7
Filter and select the top-K relevant chunks or adhere to a token limit for processing
8
Generate informed responses by processing the identified chunks through the LLM generator
ends here

Figure 4: Process for optimizing a generative AI model through RAG

[18] Google Cloud: What is Retrieval Augmented Generation?
[19] Google DeepMind: Gemini A Family of Highly Capable Multi-Modal Models
[20] Google Cloud: RAG and Grounding on Vertex AI
[21] Google Cloud: RAGs powered by Google Search technology, Part 1

There are several types of RAG approaches suited to different use cases. Some of the common ones include the following:

- **Unstructured RAG:** Retrieving information from unstructured data such as free-text or web pages[19]**.**
- **Structured RAG:** Accessing data from structured storage, such as relational databases, tables, or data frames[22].
- **Knowledge graph RAG:** Utilizing relationships between datasets to improve relevance for comprehensive answers[23].
- **Multimodal RAG:** Searching across multiple types of data including text, images, or videos[24].
- **Public search RAG:** Using public search engines to ground references in dynamic data from the internet beyond a model's cutoff for latest developments on topics (for example, latest model releases) and events.
- **Blended RAG**: Combining several of the above approaches at the same time, with reranking across retrievals, ensuring both relevance and coverage[21].

While RAG improves response accuracy and quality, its implementation is more challenging than prompt engineering. This additional complexity is due to the need for additional infrastructure, increased performance variables, persistent data management, system access, and agentic controls. Similar to traditional search technologies, RAG requires tuning of variables like chunk size, parsing strategy, and distance metrics to optimize relevance and speed, which can be resource-intensive[21]. This tuning process requires iterative development and thorough evaluation to ensure that search results and generated outputs consistently meet performance and relevance standards, allowing for necessary adjustments and optimizations.

To simplify this time-consuming tuning, solution builders can use Retrieval-Augmented-Generation-as-a-Service (RAGaaS) offerings, like Vertex AI Search. These services use automatic optimization procedures to improve search performance, though they might limit the fine-grained control some solutions require[25].

Integrating enterprise data from various solutions, such as a CRM can be complex and often requires custom development, leading to slower implementation times. However, platforms like Vertex AI offer out-of-the-box integrations with widely-used enterprise software streamlining the implementation process. Additionally, for initial builds or smaller applications, data frameworks

[22] Google Cloud: Build enterprise gen AI apps with Google Cloud databases
[23] Google Cloud: RAGs powered by Google Search technology, Part 2
[24] Google Cloud: Multimodal Retrieval Augmented Generation (RAG) using the Vertex AI Gemini API
[25] Vertex AI Search

like LlamaIndex and LangChain can simplify data processing and orchestration, making it easier to manage necessary customizations.

RAG architectures involve multiple steps, including converting a query into an embedding, querying a database, and generating a response. Each of these steps requires specific features or services to be integrated effectively and optimized for a solution, which can add complexity and increase development time. Additionally, these services and features must be rigorously evaluated to provide feedback for tuning and, crucially, to help identify the best architecture for an application. Solution builders can use services from platforms like Vertex AI to help simplify this development. These platforms offer customizable services like vector databases, embedding models, ranking, grounded generation, and check grounding APIs. Builders can also opt for an out-of-the-box solution like Vertex AI Search and customize it.

As enterprise RAG systems connect to more data sources, the risk of unauthorized access increases. For instance, a general employee querying a system linked to HR data could inadvertently access sensitive pay information. To prevent such breaches, RAG systems need Role-Based Access Controls (RBAC) to ensure that data used in responses is accessible only to authorized individuals[26].

In agentic RAG applications, agents can use automated function calls to execute workflows, but without a human-in-the-loop, additional safeguards are needed to ensure effectiveness. For instance, if an agent retrieves incorrect information, it can cascade to subsequent tasks, leading to an inaccurate final output. To prevent this, these agentic applications need additional controls like check grounding to ensure accurate retrieval and effective workflows[27].

Example: Enhancing legal search with RAG architecture

A law firm wanted to enhance the quality of its legal briefs by finding relevant cases and generating draft language for these briefs. Initially, the firm utilized a foundation model that produced acceptable writing and content, but it often miscited cases, jeopardizing the credibility of the briefs.

To improve citations, the firm incorporated a RAG architecture into its existing solution. Through experimentation, they found that a simple chunking strategy—where the chunk size matched a paragraph—yielded the best performance. The solution successfully retrieved complete arguments by aligning the chunk size with the argument structure. Given that cases reference each other and similar case names can address entirely different topics, the RAG architecture needed a knowledge graph to ensure exhaustive retrieval and grounding in all related case details.

[26] Google Cloud: Access Control
[27] Google Cloud: Check Grounding

After these additions, the solution builders used an evaluation dataset of legal queries and their corresponding documents to measure the relevance of returned articles, using precision@10[28], and the system's accuracy and cohesiveness. Their analysis identified that the solution still produced inaccurate results and cited irrelevant cases. A deeper analysis found that although the search precision was high, the retriever passed irrelevant cases to the generator. To address this, the solution builders reduced the number of chunks included in the context from 15 to 5. This increased the relevance of the chunks provided to the model improving accuracy.

Using RAG, organizations like this law firm can source critical data points and generate richer outputs grounded in the underlying datasets. This approach improves output quality and also expands the capabilities of generative AI models by finding relevant data and enabling the models to reference multiple enterprise systems. It also allows them to generate up-to-date content, and to make decisions based on multiple data points.

Fine-tuning

Fine-tuning involves adapting foundation models to perform specific downstream tasks with greater precision and accuracy[29]. It involves updating a foundation model's parameters using a high-quality, task-specific or domain-specific dataset, which allows the model to become more effective at targeted tasks like function calling. However, fine-tuning can be costly. It requires careful execution to ensure that the process is cost-effective, that performance is optimized, and that potential issues like overfitting or reduced generalization are avoided.

[28] precision@10 - the proportion of relevant recommended items in a recommendation list of size 10.
[29] Google Cloud: Introduction to Tuning

The following image outlines the steps used to fine tune large language models (LLMs):

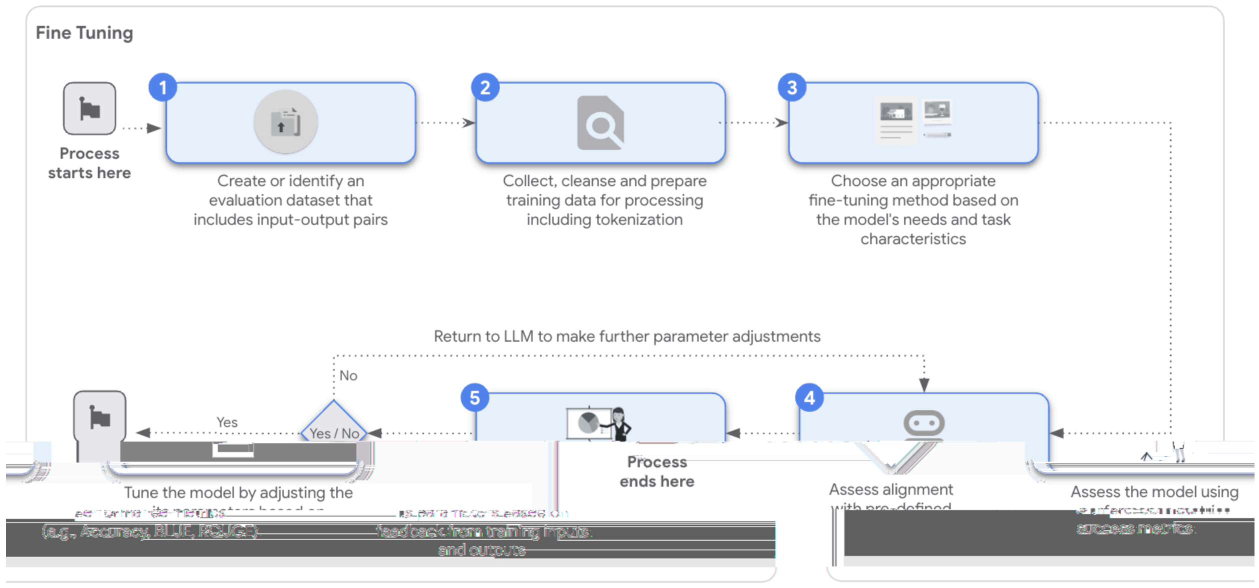

Fine Tuning
Process starts here
1
Create or identify an evaluation dataset that includes input-output pairs
2
Collect, cleanse and prepare training data for processing including tokenization
3
Choose an appropriate fine-tuning method based on the model's needs and task characteristics
Return to LLM to make further parameter adjustments
No
Yes
Yes / No
5
4
Process ends here
Tune the model by adjusting the its parameters based on feedback from training inputs and outputs
Assess alignment with pre-defined
Assess the model using performance metrics (e.g., Accuracy, BLUE, ROUGE)
success metrics

Figure 5: Process for optimizing a generative AI model through fine-tuning

Fortunately, several established fine-tuning techniques streamline the implementation process. To reduce costs, parameter-efficient fine-tuning techniques (PEFT) update only a subset of parameters, which has the additional benefit of limiting unintended changes to the model[31]. The most common of these techniques include the following:

- **Low-rank adaptation (LoRA):** Fine-tuning a model efficiently involves updating only a small portion of the model's parameters instead of adjusting the entire model, which makes the tuning process faster and less computationally intensive[30].

- **QLoRA**: Improving a model like LoRA by reducing memory usage, typical from changing 32-bit floating-point numbers to lower bit representations (for example, 4-bit or 8-bit)[31].

These techniques can dramatically improve task-specific performance. For instance, by using PEFT to fine-tune Flan-PaLM for medical question answering, Med-PaLM improved performance on clinician evaluations of the original model from 61.9% to 92.6%[32]. While fine-tuning will not always result in such a large improvement, it can still yield substantial gains, unlocking new applications.

---

[30] Low-Rank Adaptation of Large Language Models
[31] QLORA: Efficient Fine Tuning of Quantized LLMs
[32] Google DeepMind: Large Language Models Encode Clinical Knowledge

Distillation, a complementary technique, can further improve the performance of smaller models by transferring knowledge from a larger model. Unlike LoRA and QLoRA, which focus on efficient fine-tuning, distillation uses a larger model (teacher) to train a smaller model (student) to replicate the teacher's outputs. This process, often used by model developers, results in a smaller, more resource-efficient model that can perform comparably to the larger one.

For generative AI applications, balancing performance with resource efficiency is essential. Larger models, like Gemini 1.5 Flash and Gemini 1.5 Pro, offer excellent performance but can be costly due to higher computational demands per query[33]. Smaller models are more cost-effective but less performant out-of-the-box. Techniques like fine-tuning and distillation can enhance the performance of smaller models, enabling developers to reduce costs. Fine-tuning optimizes smaller models, while distillation trains them to replicate larger models' outputs[34]. Typically, developers start with an out-of-the-box large model, such as Gemini 1.5 Pro, and optimize it. If this model doesn't balance performance and resource efficiency effectively, then developers can replace this model with a distilled model or fine tuned smaller model. By carefully evaluating task requirements and measuring performance, developers can optimize performance and manage costs effectively.

Fine-tuning can enhance performance on specific tasks but is less effective at introducing new information, such as corporate policies[35]. Additionally, fine-tuning can reduce performance on other tasks; for example, research has shown that it can diminish the effectiveness of the built-in safety features of foundation models[36]. As a result, fine-tuned models need additional evaluations, such as safety and bias testing, before deployment to production. To address these needs, solution builders can use a generative AI platform, such as Vertex AI. These platforms typically offer automated evaluation tools to monitor safety and additional guardrails to manage any unexpected behavior[37].

An executive of a large health plan discussed the benefits of fine-tuning, saying, “Our team has been collecting data over the last 5 to 6 years for AI and ML investments. For generative AI, we have fine-tuned BERT on a clinical dataset to improve the accuracy and speed of our retrievers for RAG. As a result, we minimized costs for our business units.” By fine-tuning their retrievers, this health plan reduced latency and developed a solution that balanced cost and speed, enabling scalability across their enterprise.

[33] Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes
[34] LLMs: Fine-tuning, distillation, and prompt engineering
[35] Does Fine-Tuning LLMs on New Knowledge Encourage Hallucinations?
[36] LoRA Fine-tuning efficiently Undoes Safety Training In LLama 2-CHAT 70B
[37] Google Cloud: A Platform Approach to Scaling Generative AI in the Enterprise

Example: Enhancing customer interaction with fine-tuning

A financial services company aimed to improve the accuracy and relevance of responses provided by its automated customer interaction system. Initially, the system used a foundation model enhanced with RAG and prompt engineering to handle customer inquiries. However, the system struggled with complex, multi-step queries, often providing generic or incomplete responses that led to customer dissatisfaction and escalations to human agents.

To address these issues, the team decided to fine-tune the model to better handle specific, nuanced interactions typical in financial services. They began by creating a comprehensive dataset of annotated customer interactions, focusing on complex scenarios like account disputes, loan processing, and regulatory inquiries. This dataset was designed to capture the specific language and detailed steps required for accurate resolution in these cases.

Using this tailored dataset, the team benchmarked the model's initial performance, identifying significant gaps in response specificity and accuracy. They then applied PEFT techniques, particularly LoRA, to fine-tune the model. The fine-tuning process was focused on adjusting the model to better understand the context and deliver precise, step-by-step guidance for complex customer interactions.

After fine-tuning, the model was rigorously evaluated using a set of complex, multi-step customer queries. The results showed a marked improvement in the model's ability to generate accurate, contextually relevant responses that reduced the need for human intervention. Finally, the fine-tuned solution was deployed in a live environment, where it significantly improved customer satisfaction scores and reduced escalation rates by enabling the automated system to resolve complex queries more effectively.

Fine-tuning allows organizations to refine and adapt models to meet their specific operational needs, ensuring that solutions not only meet but exceed performance expectations. By leveraging finetuning techniques, enterprises can transform generic models into specialized tools that handle complex tasks with precision and reliability. For any organization looking to deploy at scale, fine-tuning is an essential strategy to achieve a high level of customization and success in real-world solutions.

Long context windows

A long context window allows a model to handle a larger amount of information, increasing the detail and range of content that can be considered when generating a response. This expanded capacity unlocks additional techniques that might offer alternatives to fine-tuning or RAG[38,39].

Unlike these methods,long contextwindow techniques are special cases of prompt engineering and benefit from prompt engineering's lower implementation costs and straightforward use[38,38,39].

To leverage these capabilities, solution builders can use *many-shot prompting*[39], providing several – often 20 or more – examples of input-output pairs to the model. Although these examples are similar to those used in fine-tuning, they are fed into the model at inference time rather than during training. Depending on the use case, many-shot prompting can sometimes approach or even surpass the performance of fine-tuning[35]. Additionally, it can reduce the trial-and-error process associated with finding an optimal set of examples, which is more common in few-shot prompting[40,37].

Another capability of long context models is *corpus-in-context reasoning*[41], where engineers can include an entire corpus—such as all maritime case law from the last 10 years—into the context window. For instance, Gemini 1.5 Pro, with its 2M context window, can store approximately 1.4 million words of text or up to two hours of video, significantly expanding the range of data that can be processed in a single pass[41]. This increased access to relevant information can enable results comparable to, or even better than, those achieved with RAG. It does so without the need for additional infrastructure and optimizations associated with tweaking embeddings, retrievers, and rerankers[36].

Additionally,long context models experience less loss compared to RAG because they do not rely on vector representations for retrieval[42]. Furthermore, RAG solutions face challenges in retrieving and correlating relevant content across multiple document chunks. Unlike long context models that process the entire document, RAG needs to retrieve the right chunks and connect content across chunks, complicating the ability to connect ideas across different sections. As a result, these RAG solutions often require multi-step reasoning to synthesize information effectively.

Long-context models are constrained by the size of their context window, meaning that for use cases involving corpora exceeding this limit, RAG remains necessary. In these scenarios, shifting the optimization focus from retrieving chunks to entire documents can simplify the process and improve efficiency, allowing solution builders to enhance performance while reducing development effort and costs.

Long-context windows have advantages, however, they require additional optimizations to manage costs and improve performance. Implementing strategies like *context caching*, supported by platforms such as Vertex AI, allows developers to reuse context from previous

[38] Prompt Engineering v Fine-tuning v RAG
[39] Many-Shot In-Context Learning
[40] Canlong contextLanguage Models Subsume Retrieval, RAG, SQL, and More?
[41] Unlocking Multimodal Understanding Across Millions of Tokens of Context
[42] Seven Points of Failure When Engineering a Retrieval Augmented Generation System

queries[43], potentially reducing costs by up to 75%[44]. This cost reduction enables achieving the performance benefits of many-shot prompting at costs closer to those of few-shot prompting. Additionally, as the volume of input data increases, adding structure—such as document IDs—helpslong context models identify relevant content more effectively, leading to better outputs.

Furthermore, research has progressed beyond the largely solved "needle-in-a-haystack" tests and lost-in-the-middle problem, focusing on enhancing performance and unlocking novel techniques[45]. While these benchmarks are solved in academic settings, real-world applications (for example, legal search) require multi-needle extraction across the document. Solving these scenarios requires experimentation withlong contextprompting as well as improvements in model quality. These advancements are enabling long context windows to be applied to common enterprise use cases, such as market research, where the ability to process large, complex datasets in a single pass is invaluable.

Example: Streamlining market research with long context AI

A consulting firm specializing in market research sought to enhance the efficiency and accuracy of its report generation process. The firm upgraded to a new foundation model, Gemini 1.5 Pro, which features a 2M token context window. Thislong contextcapability allowed the model to ingest and analyze vast amounts of market data, including entire industry reports, financial statements, and historical performance metrics, all within a single query.

To leverage this capability, the firm implemented corpus-in-context reasoning by loading entire datasets of market research reports from the past decade into the model's context window. This approach enabled the model to draw upon a comprehensive corpus of industry knowledge, allowing it to identify trends, compare historical data, and generate more contextually grounded insights. The model could now provide detailed, comparative analyses that integrated both current and historical data, leading to richer and more accurate market forecasts.

By also implementing many-shot prompting with 30 examples of detailed market analysis queries and corresponding insights, the firm saw further improvements in the model's ability to generate nuanced and actionable reports. The model could now synthesize a broader range of data points, leading to deeper market insights and more precise predictions.

To optimize costs while maintaining high performance, the firm employed context caching, which enabled the reuse of previously loaded data across multiple market research projects. This approach effectively reduced the cost-per-query and allowed the consulting firm to offer high-quality, data-driven insights at a lower operational cost. As a result, the firm could deliver

[43] Google Cloud: Context caching overview
[44] Medium: Vertex AI Context Caching with Gemini
[45] Long Context Prompting For Claude 2.1

more thorough and timely reports to clients, significantly enhancing their competitive edge in the market research space.

By expanding the capabilities of prompt engineering,long contextwindow techniques offer potential performance improvements that might approach those achieved through RAG and fine-tuning in certain scenarios. However, the effectiveness of these methods can vary, and they require careful experimentation to determine their optimal application. As best practices for leveraging long context models continue to evolve, techniques like many-shot prompting and corpus-in-context reasoning could improve performance and streamline development. While these approaches might offer a lower technical lift, they should be evaluated on a case-by-case basis to determine whether they serve as a complementary tool or a full alternative.

Selecting a technique or set of techniques

AI solution builders often face challenging design decisions when choosing the optimal technique for enhancing a generative AI solution. Beyond prompt engineering, these techniques can increase development costs but are often required to realize value, making the choice crucial for maximizing ROI. We recommend a two-step process:

1. Select a starting point based on your use case's needs
2. Evaluate the results, add services or features as needed, and refine the solution until it meets the desired performance criteria.

The following image provides a decision tree to decide the right starting point for customizing an LLMs responses for your use cases:

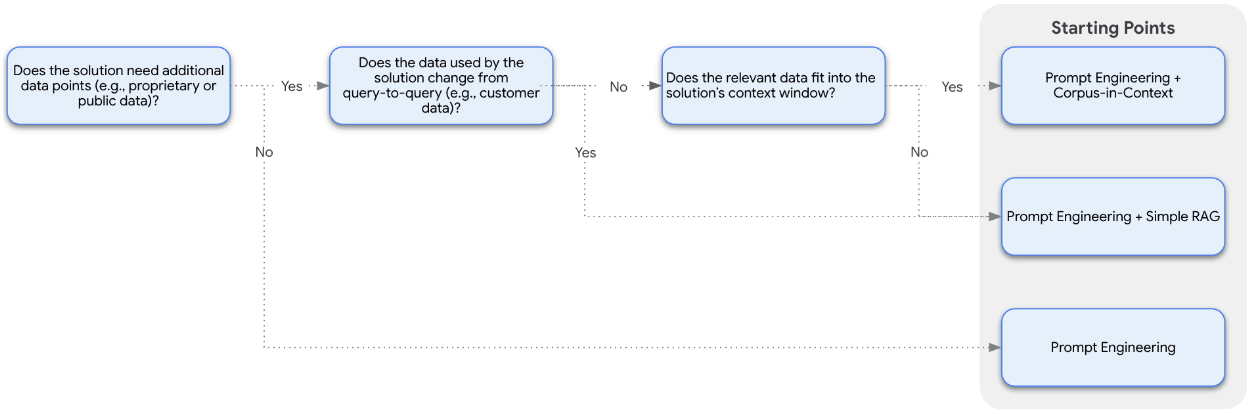

Starting Points
Does the solution need additional data points (e.g., proprietary or public data)?
Yes
No
Does the data used by the solution change from query-to-query (e.g., customer data)?
No
Yes
Does the relevant data fit into the solution's context window?
Yes
No
Prompt Engineering + Corpus-in-Context
Prompt Engineering + Simple RAG
Prompt Engineering

Figure 6: Decision tree for selecting a starting point

To identify the optimal starting point for your solution, begin by clearly defining your success criteria and specific requirements. Next, evaluate the characteristics of the data used in your solution. Determine whether additional data, either public or proprietary, is necessary. If no extra data is needed, a straightforward prompt engineering approach might be sufficient. If additional data is required, consider whether this data is dynamic (changing with each query) or static (consistent across queries) and whether it fits within the context window. For static data that fits within the context window, combining prompt engineering with corpus-in-context techniques is recommended. If static data exceeds the context window, consider integrating prompt engineering with a simple RAG approach. For dynamic data, start with a prompt engineering and RAG approach.

When considering fine-tuning, it's important to recognize that this technique, while powerful, is best reserved for later stages due to its higher costs and complexity. Fine-tuning should generally not be the starting point unless the task is highly specialized or demands a level of precision that cannot be achieved through prompt engineering or RAG alone. If fine-tuning is necessary, it's often more effective to begin with a smaller, more focused model. This approach allows for quicker, cost-effective specialization before potentially scaling up to a larger model if further improvements are needed. By carefully evaluating your task requirements and selecting the appropriate starting point, you can optimize performance while managing resources effectively.

After deciding on the starting approach, you can then proceed to build and refine a working prototype. To determine the most effective starting point, begin by identifying a use case archetype that closely aligns with your project goals.

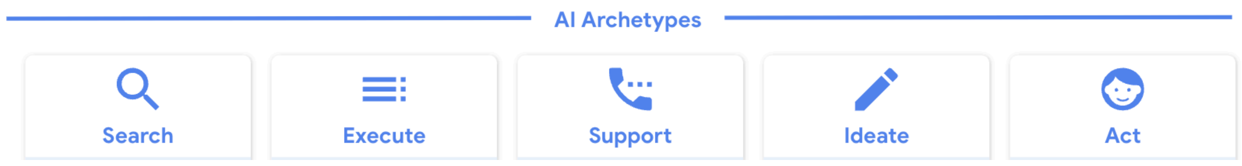


Figure 7: AI Use case archetypes and starting points.

The following table describes how you can prioritize features in your PoC.

AI archetype	**Search**	**Execute**	**Support**	**Ideate**	**Act**
Description	Find and summarize information for a user enabling subsequent business processes	Tailor standard language, implementation, or content to a specific prompt	Answer a wide variety of questions and provide guidance to resolve issues	Create novel ideas for content, research topics, or frameworks to facilitate further thinking	Complete end-to-end business processes to reduce resource requirements
Starting points	• Prompt engineering • RAG	Prompt engineering	• Prompt engineering • RAG	Prompt engineering	• Prompt engineering • RAG
Common use cases	• Legal Search Assistant • Medical Search Assistant • Claims Benefit Extraction • Supplier Invoice Investigator	• Code Assistant • Contract Generation • Enterprise Co-pilot	• Customer Support • IT Support • Agent Assistant • FAQ Chatbot • Claims Agent (FSI, HCLS)	• Ad Copy Generator • Interactive Content Development • Personalized Content Generator	• Ad Copy Generator • Interactive Content Development • Personalized Content Generator
Illustrative use case	Search is time intensive and open-ended, creating long, uncertain timelines. While AI models have impressive knowledge	Businesses have lots of repetitive tasks that require small customizations. LLMs can make these customizations, but occasional inaccuracies and indeterminate	High inquiry volumes drive costs. Using LLMs to answer queries reduces costs and frees up resources. However, their latency and occasional	Creating new ideas or frameworks is a time-consuming and iterative process. LLMs can accelerate this process by generating potential ideas. But	Executives rate automation as a key goal of AI investments, but lots of AI solutions still require a human touch. LLMs can execute these end-to-end

	out-of-the-box, models occasionally generate inaccurate answers, raising doubts about results.	outputs can limit the use of LLMs.	inaccuracies can prevent adoption	with more creativity they tend to have inaccuracies.	processes, but struggle with multi-step tasks and accessing information.

With a working prototype, you can begin iterating by following these steps:

- **Evaluate:** Calculate and track performance metrics using input-output pairs, technical performance data, and human evaluations. For a detailed discussion, refer to the "Evaluation Paper Link."
- **Compare:** Identifying performance gaps in areas such as accuracy, style, or tone relative to your solution requirements.
- **Enhance:** Choosing appropriate techniques to improve services and features, such as prompt engineering, RAG architecture, fine-tuning, or long context window methods, to boost system performance (see Figure 8 for common options).
- **Predict:** Using the updated solution to generate new predictions, applying an evaluation dataset to assess its effectiveness.

The following image describes the various various customization that may be applied for a RAG architecture:

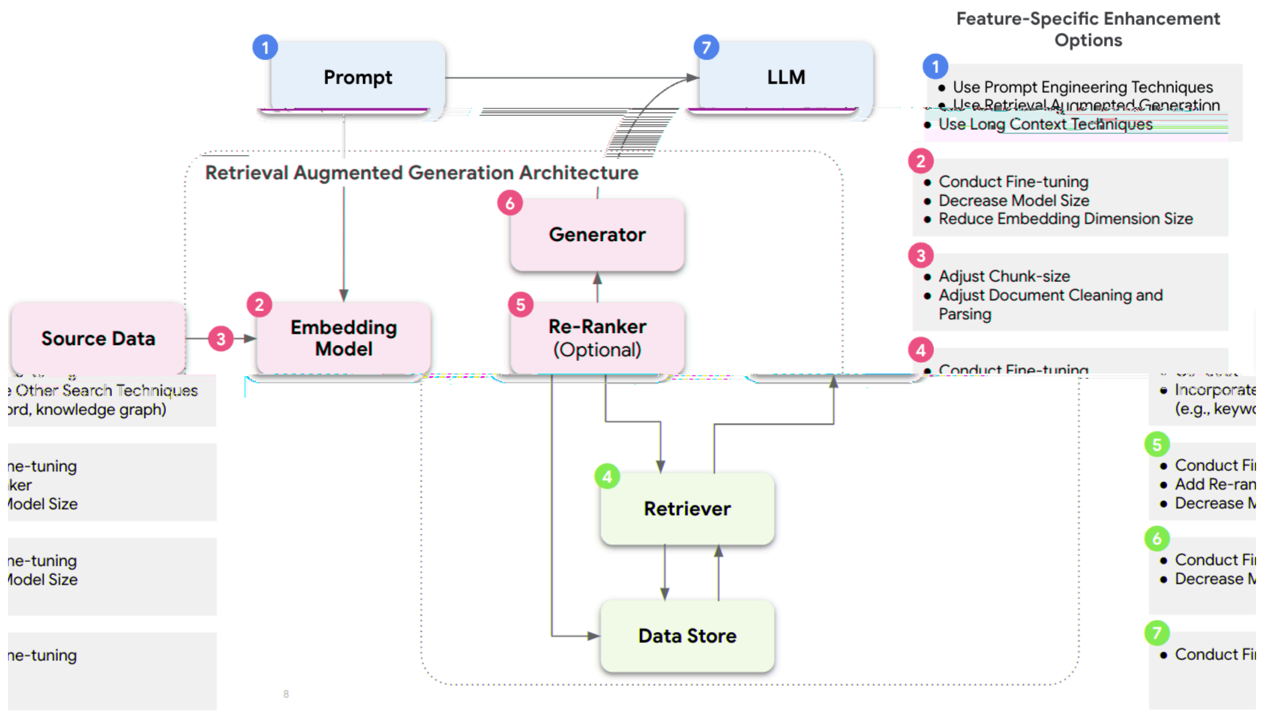


Figure 8 Common generative AI use case architectures incorporating RAG with feature-specific enhancement options to improve end-to-end solution performance.

Continue to repeat the steps described in this section until the solution meets your requirements and defined success criteria. By iterating and making incremental adjustments, you can steadily improve performance while keeping development costs to a minimum.

Example: Iterative development of a customer support chatbot

A clothing retailer aimed to boost call center efficiency by deploying a generative AI chatbot. The team defined success as achieving 99% accuracy in answering customer questions or seamlessly connecting customers to a live agent when necessary. To meet these goals, the chatbot needed to accurately reference a corporate knowledge base to retrieve relevant policies and how-to manuals. The team initially implemented a RAG architecture combined with prompt engineering using few-shot examples.

During the first round of testing, the chatbot fell short of the 99% accuracy target, often providing irrelevant or incomplete responses. The team diagnosed the issue as stemming from the RAG architecture, which was retrieving results that were not closely aligned with the specific customer queries.

In the next iteration, the team introduced a re-ranker to refine the relevance of the retrieved results while maintaining minimal latency to ensure a smooth customer experience. This adjustment led to improved accuracy, but the results still did not meet the high standards set by the team.

To address this, the team decided to fine-tune the BERT-based re-ranker using a more focused dataset that included edge cases and complex queries often encountered by the chatbot. This fine-tuning process significantly enhanced the model's ability to prioritize the most relevant information, leading to a noticeable improvement in accuracy.

In the final round of evaluations, the chatbot not only achieved the desired 99% accuracy but also demonstrated a reliable ability to direct customers to live agents when necessary. The iterative process of refining the RAG architecture, introducing a re-ranker, and fine-tuning the system ultimately resulted in a highly effective customer support tool that exceeded initial performance expectations.

Building high-performing generative AI solutions begins with understanding and applying prompt engineering, RAG, fine-tuning, andlong contextwindow techniques. Each of these methods offers unique strengths: prompt engineering tailors model responses to specific needs, RAG enriches contextual understanding by pulling in relevant data from diverse sources, fine-tuning hones model precision for domain-specific tasks, andlong contextwindows enable deeper analysis by accommodating larger datasets.

Most solutions derive value from a blend of these approaches. Integrating them within an evaluation-driven framework creates an iterative process that optimizes performance and adapts the solution to your evolving needs. By systematically refining your solution and integrating additional features as needed, you can develop AI systems that not only meet but exceed performance expectations, setting new standards for enterprise efficiency.