

Intro to Prompt Engineering

David Lee | TIM | 75

Today

- Intro to Generative AI
- Prompt engineering + Prelab
- Upcoming deliverables
 - Homework 2 (prelab) due TODAY at 11:59pm
 - Homework 2 (individual) due SATURDAY at 11:59pm
 - Homework 2 (team) due NEXT MONDAY at 11:59pm

Reminder: Late policy

- Automatic extensions for active contributors and engaged teams, 90% after that, 0% after next session (this is when we sometimes may discuss the submission)

	Due	Automatic extension (for active contributors)	90% by this deadline (0% afterwards)
Prelab	Tuesday 11:59pm	Wednesday 11:59pm	Before Thursday lecture
Lab (individual)	Saturday 11:59pm	Sunday 11:59pm	Before Monday section
Lab (team)	Monday 11:59pm	Tuesday 11:59am	Before Tuesday lecture

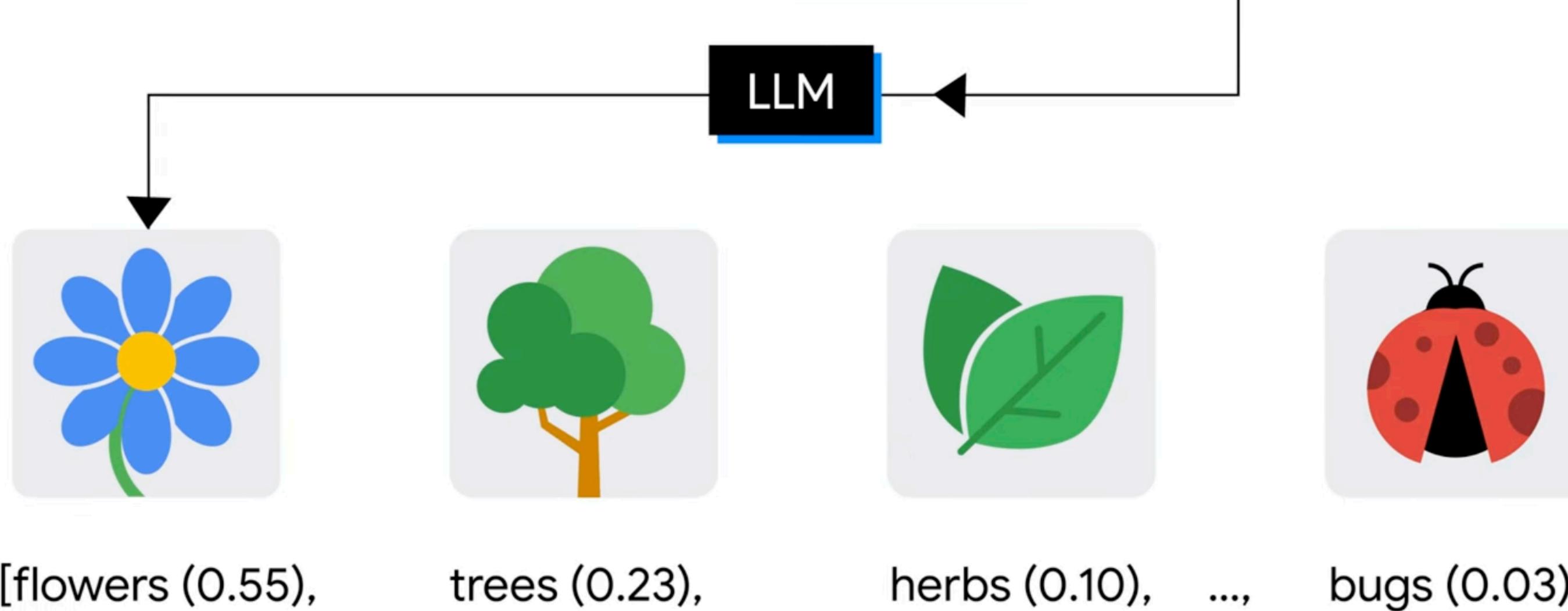
Intro to Generative AI

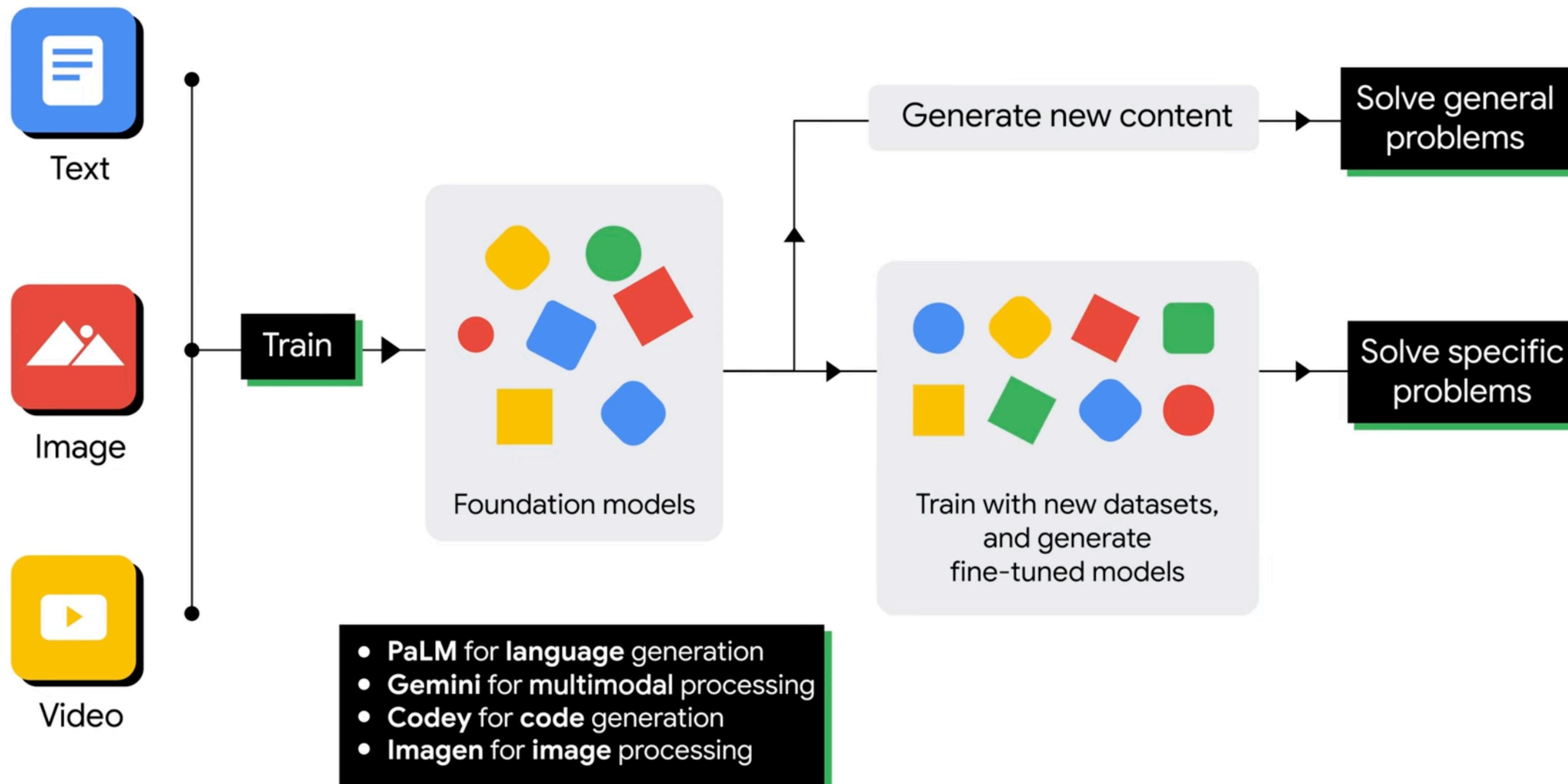
There are TONS of online resources

- I've shared some readings/videos each week
- I'll also be drawing from them in my lectures

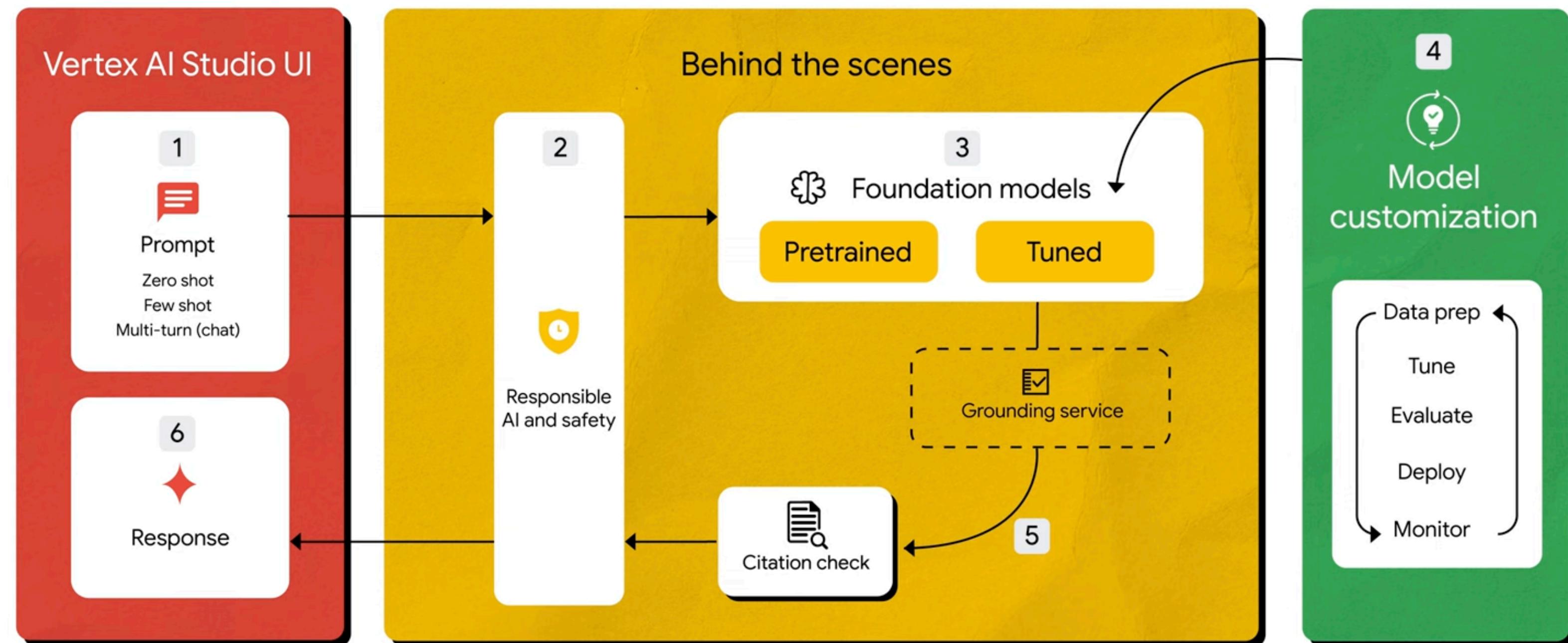
The screenshot shows the Google Cloud Skills Boost landing page. At the top, there's a navigation bar with links for 'Google Cloud', 'Paths', 'Explore', and 'Subscriptions'. On the right side of the header are 'Sign in' and 'Join' buttons. Below the header, the main title 'Google Cloud Skills Boost' is displayed, followed by a subtitle: 'Choose your learning path, build your skills, and validate your knowledge. All in one place.' A call-to-action button 'Create an account today' is present. To the right of the text is a decorative graphic of overlapping colored squares (blue, yellow, white) containing various icons related to cloud computing and AI. Below this section, the heading 'Discover Google Cloud training your way' is shown, followed by three tabs: 'Learning Path' (which is selected), 'Course', and 'Lab'. Under the 'Learning Path' tab, three cards are listed: 'Beginner: Introduction to Generative AI Learning Path' (with a brain icon), 'Intermediate: Gemini for Google Cloud Learning Path' (with a circuit board icon), and 'Advanced: Generative AI for Developers Learning Path' (with a hexagonal icon).

The garden was full of beautiful...





Generative AI workflow on Vertex AI





L

You

how do you describe the main idea/innovation in the transformer model compared to previous generative models like RNNs and LSTMs?



ChatGPT

The main innovation of the transformer model, introduced in the paper "Attention is All You Need" by Vaswani et al. in 2017, is its reliance solely on an attention mechanism to handle dependencies between input and output, which marks a departure from the recurrent architectures used in previous models like RNNs and LSTMs.

Here are the key innovations and advantages of transformers compared to RNNs and LSTMs:

- 1. Attention Mechanism:** The transformer uses a mechanism known as self-attention, which allows it to weigh the importance of different words in a sentence, irrespective of their positions. This is in contrast to RNNs and LSTMs, which process data sequentially and therefore have difficulty accessing information from earlier or later in the sequence.
- 2. Parallelization:** Unlike RNNs and LSTMs, which inherently require sequential processing (i.e., the output at one timestep depends on the previous output), transformers process all words or tokens in parallel during training. This parallelization significantly improves training speed and efficiency.
- 3. Global Context:** Transformers can capture dependencies between any two words in a sentence, regardless of their distance. RNNs and LSTMs, although designed to handle long-range dependencies, often struggle with this due to issues like vanishing gradients. Transformers address this with multi-head attention, allowing the model to focus on different parts of the sequence simultaneously.
- 4. Scalability:** Due to their parallel nature and efficient use of the attention mechanism, transformers are highly scalable with increasing data and model size, which has enabled training on much larger datasets and model architectures than was feasible with RNNs or LSTMs.



Message ChatGPT





ⓘ See the latest updates to the [Gemini Apps Privacy Hub](#)



can you explain reinforcement learning with human feedback (RLHF) and how it is used to train generative models?



Absolutely! Reinforcement learning with human feedback (RLHF) is a technique for training AI models to better align with human preferences. Here's a breakdown of how it works and how it's used with generative models:

Traditional Reinforcement Learning (RL) Challenges

In standard reinforcement learning, an AI agent learns by trial and error in an environment. It receives rewards for good actions and penalties for bad ones. This feedback helps the agent develop a policy, essentially a set of rules for navigating the environment and maximizing rewards.

However, defining a clear reward function that captures human goals can be difficult, especially for complex tasks. Imagine training a language model to write funny jokes. It's hard to translate "funny" into a mathematical reward system.

How RLHF bridges the gap

This is where RLHF comes in. Instead of a pre-defined reward function, RLHF uses direct human feedback to train a separate "reward model." Here's the process:

1. **The model generates outputs:** The AI model (like a text generator) creates

Enter a prompt here



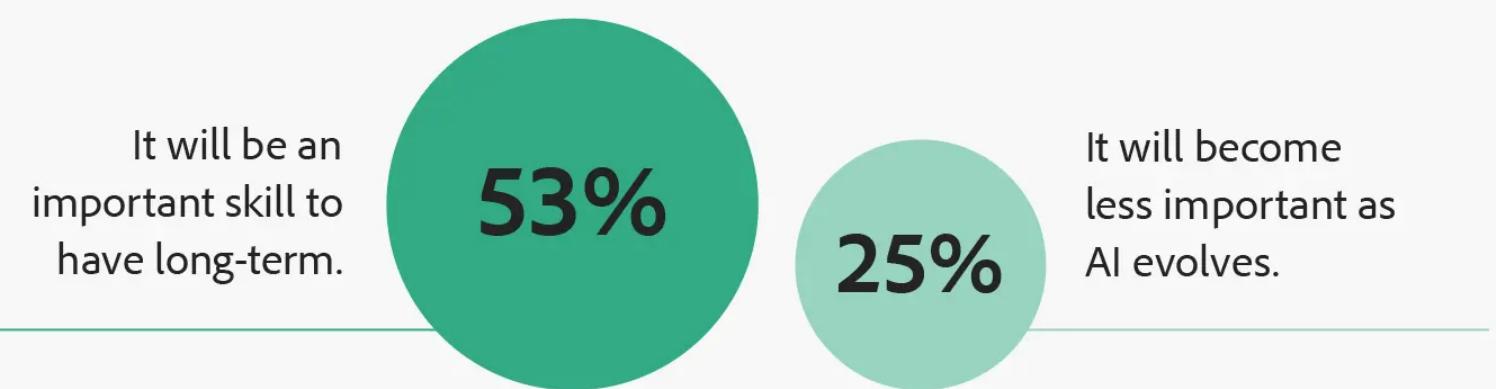
The Impact of Prompt Engineering on Small Businesses



Over 1 in 4 small business owners (27%) said that **prompt engineering is a very important skill** for their business.

Future Importance of Prompt Engineering Skills

According to Small Business Owners



38% of small business owners said they'd consider offering **prompt engineering training** to current employees.



More than 1 in 3 small business owners (36%) said they're **more likely to hire** someone skilled at prompt engineering.

Prompt engineering
+ Prelab overview

Prompt Engineering Techniques

TIM 175 WEEK 2 PRELAB

Brief Task Overview

Set up [LastMile AI](#) and create a copy of this document to work through the activities. For each activity:

- Capture screenshots of the prompts you used and the outputs generated by the model. Place these images directly **above** the analysis tables of each activity.
- Use the table to write a detailed analysis of your reflections from the activity.
- **ALL NEW TEXT YOU ADD SHOULD BE IN RED TO MAKE IT EASY FOR US TO SEE.**

Complete 4 activities to understand different popular Prompt Engineering Techniques

1. Compare Zero Shot vs Few Shot Prompting
2. Practice Basic Prompting Techniques
3. Chain of Thought Techniques
4. Chain of Verification Techniques
5. (Optional) Meta-Prompting



VPN



Update



Workbooks



David Lee



Workbooks

A workbook is a quick way to experiment with LLMs and other inference models, compare results, and share work with your team.

[+ New workbook](#)

Templates



GPT4 Coding Assistant

Summarize Interview Audio

Stable Diffusion Image Generator

Standup Meeting Summarizer

Customer Notes by GPT4

Learn a Language with AI Stories

 Search workbooks

Workbook Name

Created

No data for this table right now.

Best practices for prompt design

- ✓ Be concise.
- ✓ Be specific and well-defined.
- ✓ Ask one task at a time.
- ✓ Ask to classify instead of generate.
- ✓ Include examples.



Filter

Release notes

Capabilities

- ▶ Model tuning
- ▶ Function calling
- ▶ Embeddings
- ▶ Safety

Guides

- ▼ Prompting
 - Intro to prompting
 - Prompting with media files
 - Prompting strategies
 - File prompting strategies

System instructions

Semantic retrieval

OAuth authentication

Firebase extensions

Migrate to Cloud

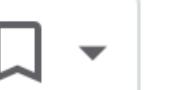
Tutorials

- ▶ Function calling
- ▶ Embeddings
- ▶ Applications

Troubleshooting

Troubleshooting guide

Prompt design strategies

[Send feedback](#)

This page introduces you to some general prompt design strategies that you can employ when designing prompts. While there's no right or wrong way to design a prompt, there are common strategies that you can use to affect the model's responses. Rigorous testing and evaluation remain crucial for optimizing model performance.

Large language models (LLM) are trained on vast amounts of text data to learn the patterns and relationships between units of language. When given some text (the prompt), language models can predict what is likely to come next, like a sophisticated autocomplete tool. Therefore, when designing prompts, consider the different factors that can influence what a model predicts comes next.

Give clear and specific instructions

Giving the model instructions on what to do is an effective and efficient way to customize model behavior. Ensure that the instructions you give are clear and specific. Instructions can be as simple as a list of step-by-step instructions or as complex as mapping out a user's experience and mindset.

Define the task to perform

Describe in detail the task that you want the model to perform. The task can be as simple as a step-by-step list of instructions or as complex as mapping out a user's experience and mindset. The following prompt tells the model to summarize a block of text:

Prompt:

Summarize this text.



On this page

[Give clear and specific instructions](#)

Define the task to perform

Specify any constraints

Define the format of the response

Include few-shot examples

Zero-shot vs few-shot prompts

Find the optimal number of examples

Use examples to show patterns instead of antipatterns

Use consistent formatting across examples

Add contextual information

Add prefixes

Let the model complete partial input

Break down prompts into simple components

Break down instructions

Chain prompts

Aggregate responses

Experiment with different parameter values

Max output tokens

Temperature

Top-K

Top-P

Prompt iteration strategies

Use different phrasing

Switch to an analogous task

 Search

⌘ K

Prompt engineering

GET STARTED

[Introduction](#)[Quickstart](#)[Models](#)[Tutorials](#)[Changelog](#)

CAPABILITIES

[Text generation](#)[Function calling](#)[Embeddings](#)[Fine-tuning](#)[Image generation](#)[Vision](#)[Text-to-speech](#)[Speech-to-text](#)[Moderation](#)

ASSISTANTS

[Overview](#)[How Assistants work](#)[File Search](#)[Code Interpreter](#)

This guide shares strategies and tactics for getting better results from large language models (sometimes referred to as GPT models) like GPT-4. The methods described here can sometimes be deployed in combination for greater effect. We encourage experimentation to find the methods that work best for you.

Some of the examples demonstrated here currently work only with our most capable model, `gpt-4`. In general, if you find that a model fails at a task and a more capable model is available, it's often worth trying again with the more capable model.

You can also explore example prompts which showcase what our models are capable of:



Prompt examples

Explore prompt examples to learn what GPT models can do

Six strategies for getting better results

Write clear instructions

These models can't read your mind. If outputs are too long, ask for brief replies. If outputs are too simple, ask for expert-level writing. If you dislike the format, demonstrate the format you'd like to see. The less the model has to guess at what you want, the more likely you'll get it.

Tactics:

- [Include details in your query to get more relevant answers](#)
- [Ask the model to adopt a persona](#)
- [Use delimiters to clearly indicate distinct parts of the input](#)
- [Specify the steps required to complete a task](#)
- [Provide examples](#)
- [Specify the desired length of the output](#)

Write clear instructions

These models can't read your mind. If outputs are too long, ask for brief replies. If outputs are too simple, ask for expert-level writing. If you dislike the format, demonstrate the format you'd like to see. The less the model has to guess at what you want, the more likely you'll get it.

Tactics:

- [Include details in your query to get more relevant answers](#)
- [Ask the model to adopt a persona](#)
- [Use delimiters to clearly indicate distinct parts of the input](#)
- [Specify the steps required to complete a task](#)
- [Provide examples](#)
- [Specify the desired length of the output](#)

Provide reference text

Language models can confidently invent fake answers, especially when asked about esoteric topics or for citations and URLs. In the same way that a sheet of notes can help a student do better on a test, providing reference text to these models can help in answering with fewer fabrications.

Tactics:

- [Instruct the model to answer using a reference text](#)
- [Instruct the model to answer with citations from a reference text](#)

Give clear and specific instructions

Define the task to perform

Specify any constraints

Define the format of the response

Include few-shot examples

Zero-shot vs few-shot prompts

Find the optimal number of examples

Use examples to show patterns instead of antipatterns

Use consistent formatting across examples

Add contextual information

Add prefixes

Let the model complete partial input

Test changes systematically

Improving performance is easier if you can measure it. In some cases a modification to a prompt will achieve better performance on a few isolated examples but lead to worse overall performance on a more representative set of examples. Therefore to be sure that a change is net positive to performance it may be necessary to define a comprehensive test suite (also known as an "eval").

Tactic:

- Evaluate model outputs with reference to gold-standard answers

Give the model time to "think"

If asked to multiply 17 by 28, you might not know it instantly, but can still work it out with time. Similarly, models make more reasoning errors when trying to answer right away, rather than taking time to work out an answer. Asking for a "chain of thought" before an answer can help the model reason its way toward correct answers more reliably.

Tactics:

- Instruct the model to work out its own solution before rushing to a conclusion
- Use inner monologue or a sequence of queries to hide the model's reasoning process
- Ask the model if it missed anything on previous passes

Experiment with different parameter values

Max output tokens

Temperature

Top-K

Top-P

Prompt iteration strategies

Use different phrasing

Switch to an analogous task

Change the order of prompt content

Split complex tasks into simpler subtasks

Just as it is good practice in software engineering to decompose a complex system into a set of modular components, the same is true of tasks submitted to a language model. Complex tasks tend to have higher error rates than simpler tasks. Furthermore, complex tasks can often be re-defined as a workflow of simpler tasks in which the outputs of earlier tasks are used to construct the inputs to later tasks.

Tactics:

- Use intent classification to identify the most relevant instructions for a user query
- For dialogue applications that require very long conversations, summarize or filter previous dialogue
- Summarize long documents piecewise and construct a full summary recursively

Break down prompts into simple components

Break down instructions

Chain prompts

Aggregate responses

Use external tools

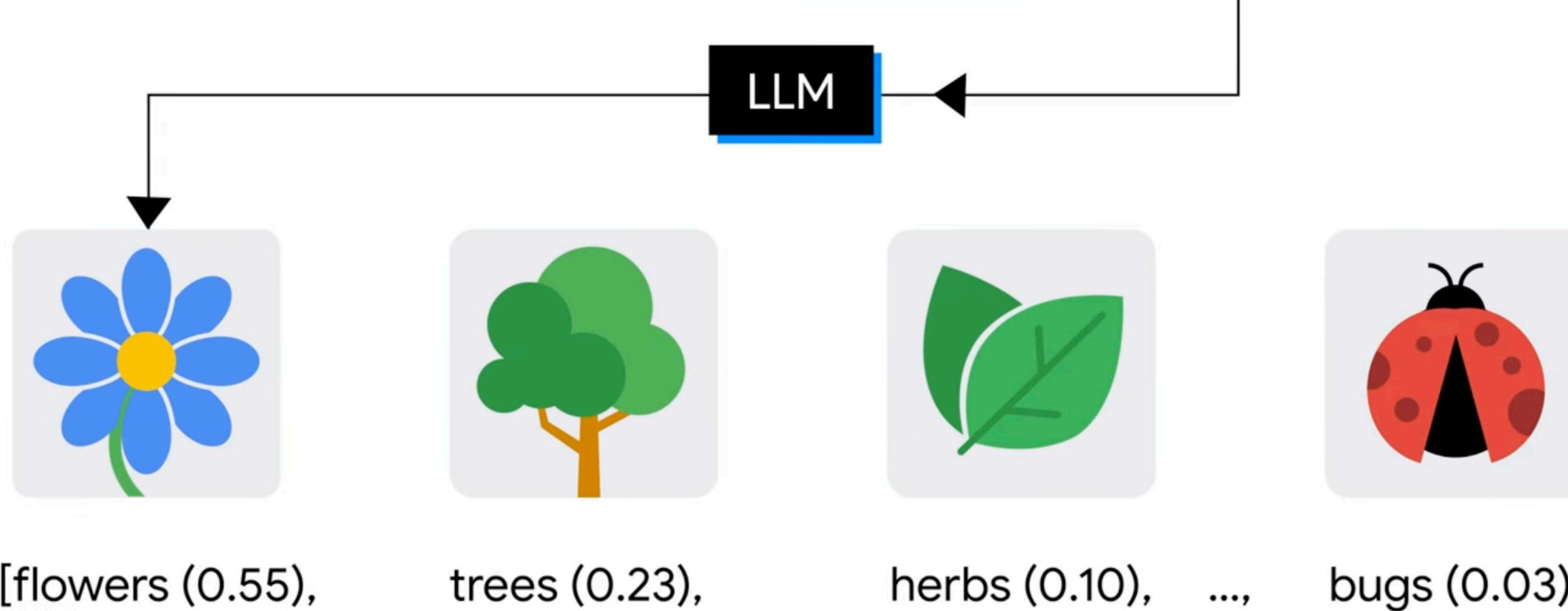
Compensate for the weaknesses of the model by feeding it the outputs of other tools. For example, a text retrieval system (sometimes called RAG or retrieval augmented generation) can tell the model about relevant documents. A code execution engine like OpenAI's Code Interpreter can help the model do math and run code. If a task can be done more reliably or efficiently by a tool rather than by a language model, offload it to get the best of both.

Tactics:

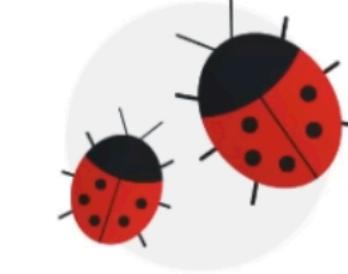
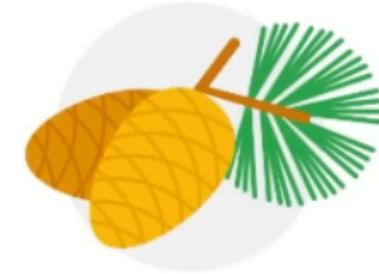
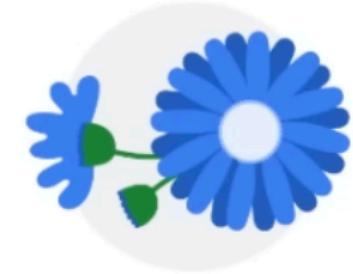
- Use embeddings-based search to implement efficient knowledge retrieval
- Use code execution to perform more accurate calculations or call external APIs
- Give the model access to specific functions

Parameter tuning

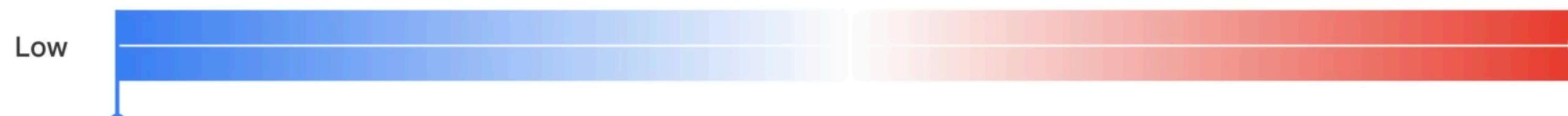
The garden was full of beautiful...



The garden was full of beautiful...

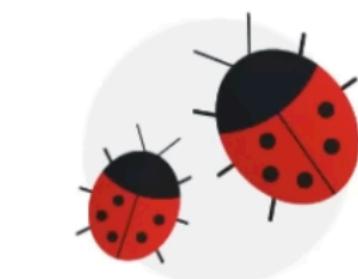
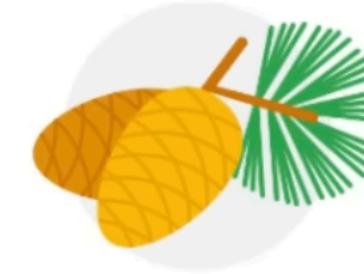
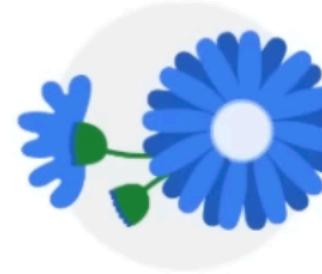


[flowers (0.5), trees (0.23), herbs (0.05), ..., bugs (0.03)]

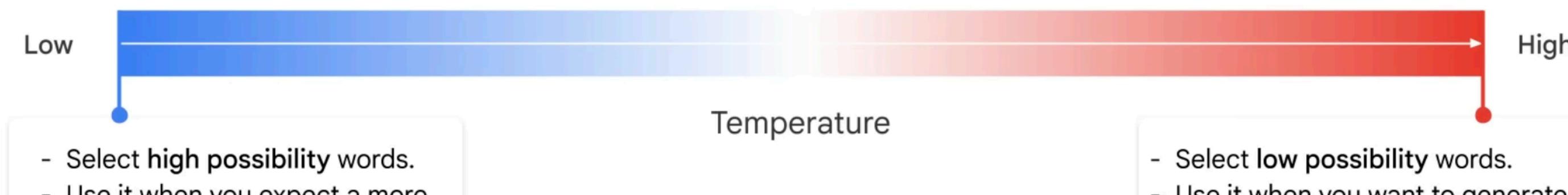


- Select high possibility words.
- Use it when you expect a more "predictable" answer.

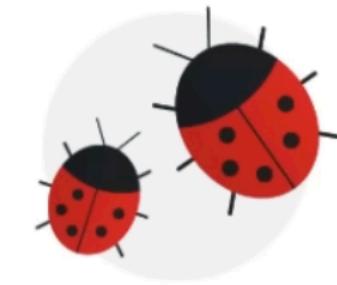
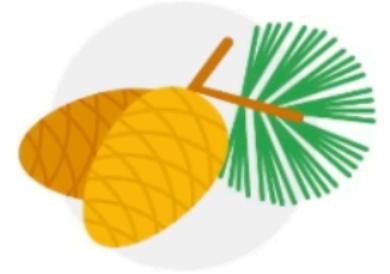
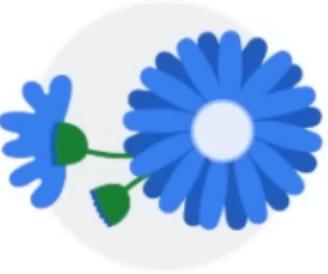
The garden was full of beautiful...



[flowers (0.5), trees (0.23), herbs (0.05), ..., bugs (0.03)]



The garden was full of beautiful...



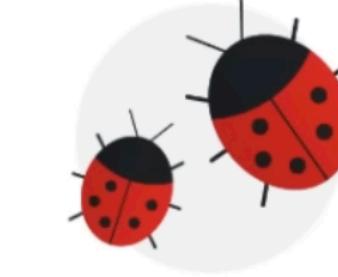
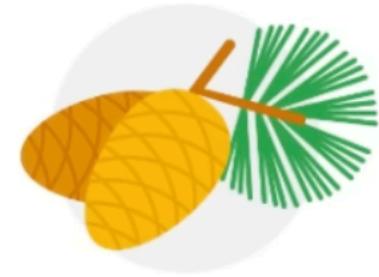
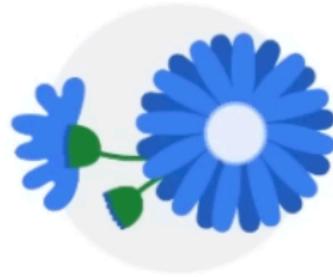
[flowers (0.5), trees (0.23), herbs (0.05), ..., bugs (0.03)]

Top K

The model returns a random word from a set of top K possible words.

For example: K = 2

The garden was full of beautiful...



[flowers (0.5), trees (0.23), herbs (0.05), ..., bugs (0.03)]

Top P

The model returns a random word from a set of words with the sum of the likelihoods not exceeding P.

For example: $P = 0.75$ Flowers (0.5)+trees (0.23)+herbs (0.05)=0.78 (> 0.75)