

Code architectures for using LLMs

Mitchell Gordon

your goals for today's class

know basic technical approaches for LLM inference

so you'll be able to build AI-powered features in your projects

understand how to design and iterate on effective prompts

so you can make your features as effective as possible

practical considerations like security, cost, etc

so you don't make costly mistakes

How do you use an API model?

Goal of this lecture isn't to teach you how to use an API, but it will be useful to have a basic sense of how they tend to work

Will use OpenAI as an example, most APIs are similar

Generate text from a model

python 

```
1 from openai import OpenAI
2 client = OpenAI()
3
4 response = client.responses.create(
5     model="gpt-5",
6     input="Write a one-sentence bedtime story about a unicorn."
7 )
8
9 print(response.output_text)
```

Generate text from a model

python 

```
1 from openai import OpenAI
2 client = OpenAI()
3
4 response = client.responses.create(
5     model="gpt-5",
6     input="Write a one-sentence bedtime story about a unicorn."
7 )
8
9 print(response.output_text)
```

```
{
  "id": "resp_67ccd2bed1ec8190b14f964abc0542670bb6a6b452d3795b",
  "object": "response",
  "created_at": 1741476542,
  "status": "completed",
  "error": null,
  "incomplete_details": null,
  "instructions": null,
  "max_output_tokens": null,
  "model": "gpt-4.1-2025-04-14",
  "output": [
    {
      "type": "message",
      "id": "msg_67ccd2bf17f0819081ff3bb2cf6508e60bb6a6b452d3795b",
      "status": "completed",
      "role": "assistant",
      "content": [
        {
          "type": "output_text",
          "text": "In a peaceful grove beneath a silver moon, a unicorn named Lumina discovered a hidden pool that reflected the stars.",
          "annotations": []
        }
      ]
    }
  ],
  "parallel_tool_calls": true,
  "previous_response_id": null,
  "reasoning": {
    "effort": null,
    "summary": null
  },
  "store": true,
  "temperature": 1.0,
  "text": {
    "format": {
      "type": "text"
    }
  },
  "tool_choice": "auto",
  "tools": [],
  "top_p": 1.0,
  "truncation": "disabled",
  "usage": {
    "input_tokens": 36,
    "input_tokens_details": {
      "cached_tokens": 0
    },
    "output_tokens": 87,
    "output_tokens_details": {
      "reasoning_tokens": 0
    },
    "total_tokens": 123
  },
  "user": null,
  "metadata": {}
}
```

Analyze images and files

Send image URLs, uploaded files, or PDF documents directly to the model to extract text, classify content, or detect visual elements.

Image URL

File URL

Upload file

Analyze the content of an image

python ↕

```
1 from openai import OpenAI
2 client = OpenAI()
3
4 response = client.responses.create(
5     model="gpt-5",
6     input=[
7         {
8             "role": "user",
9             "content": [
10                 {
11                     "type": "input_text",
12                     "text": "What teams are playing in this image?",
13                 },
14                 {
15                     "type": "input_image",
16                     "image_url": "https://upload.wikimedia.org/wikipedia/commons",
17                 }
18             ]
19         }
20     ]
21 )
22
23 print(response.output_text)
```

Extend the model with tools

Give the model access to external data and functions by attaching tools. Use built-in tools like web search or file search, or define your own for calling APIs, running code, or integrating with third-party systems.


Web search

File search

Function calling

Remote MCP

Use web search in a response

python 

```
1 from openai import OpenAI
2 client = OpenAI()
3
4 response = client.responses.create(
5     model="gpt-5",
6     tools=[{"type": "web_search"}],
7     input="What was a positive news story from today?"
8 )
9
10 print(response.output_text)
```


Manually construct a past conversation

python 


```
1 from openai import OpenAI
2
3 client = OpenAI()
4
5 response = client.responses.create(
6     model="gpt-4o-mini",
7     input=[
8         {"role": "user", "content": "knock knock."},
9         {"role": "assistant", "content": "Who's there?"},
10        {"role": "user", "content": "Orange."},
11    ],
12 )
13
14 print(response.output_text)
```

By using alternating `user` and `assistant` messages, you capture the previous state of a conversation in one request to the model.

Stream responses and build realtime apps

Use server-sent streaming events to show results as they're generated, or the Realtime API for interactive voice and multimodal apps.

Stream server-sent events from the API

python 


```
1 from openai import OpenAI
2 client = OpenAI()
3
4 stream = client.responses.create(
5     model="gpt-5",
6     input=[
7         {
8             "role": "user",
9             "content": "Say 'double bubble bath' ten times fast.",
10        },
11    ],
12    stream=True,
13 )
14
15 for event in stream:
16     print(event)
```


Some benefits of Structured Outputs include:

- 1 **Reliable type-safety:** No need to validate or retry incorrectly formatted responses
- 2 **Explicit refusals:** Safety-based model refusals are now programmatically detectable
- 3 **Simpler prompting:** No need for strongly worded prompts to achieve consistent formatting

In addition to supporting JSON Schema in the REST API, the OpenAI SDKs for [Python](#) and [JavaScript](#) also make it easy to define object schemas using [Pydantic](#) and [Zod](#) respectively. Below, you can see how to extract information from unstructured text that conforms to a schema defined in code.

Getting a structured response

python ↕ 

```
1 from openai import OpenAI
2 from pydantic import BaseModel
3
4 client = OpenAI()
5
6 class CalendarEvent(BaseModel):
7     name: str
8     date: str
9     participants: list[str]
10
11 response = client.responses.parse(
12     model="gpt-4o-2024-08-06",
13     input=[
14         {"role": "system", "content": "Extract the event information."},
15         {
16             "role": "user",
17             "content": "Alice and Bob are going to a science fair on Friday.",
18         },
19     ],
20     text_format=CalendarEvent,
21 )
22
23 event = response.output_parsed
```

There are a ton of different agent orchestration libraries out there, and substantial differentiation in how they work.

You can also handle the orchestration yourself!

Build agents

Use the OpenAI platform to build agents capable of taking action—like controlling computers—on behalf of your users. Use the Agents SDK for Python or TypeScript to create orchestration logic on the backend.

Build a language triage agent

python ↕

```
1 from agents import Agent, Runner
2 import asyncio
3
4 spanish_agent = Agent(
5     name="Spanish agent",
6     instructions="You only speak Spanish.",
7 )
8
9 english_agent = Agent(
10     name="English agent",
11     instructions="You only speak English",
12 )
13
14 triage_agent = Agent(
15     name="Triage agent",
16     instructions="Handoff to the appropriate agent based on the language of the",
17     handoffs=[spanish_agent, english_agent],
18 )
19
20
21 async def main():
22     result = await Runner.run(triage_agent, input="Hola, ¿cómo estás?")
23     print(result.final_output)
24
25
26 if __name__ == "__main__":
27     asyncio.run(main())
```

Services like OpenRouter provide a unified API, make it easier to switch between models from a variety of providers

The Unified Interface For LLMs

Better [prices](#), better [uptime](#), no subscription.

Start a message... →

Featured Models [View Trending](#)

Gemini 2.5 Pro

by [google](#)

171.5B

Tokens/wk

2.4s

Latency

+8.82%

Weekly growth

GPT-5

by [openai](#)

69.1B

Tokens/wk

5.8s

Latency

+21.67%

Weekly growth

Claude Sonnet 4

by [anthropic](#)

624.0B

Tokens/wk

1.8s

Latency

+4.04%

Weekly growth

12T

Monthly Tokens

4.2M+

Global Users

60+

Active Providers

500+

Models

- 1

Signup

Create an account to get started. You can set up an org for your team later.
- 2

Buy credits

Credits can be used with any model or provider.
- 3

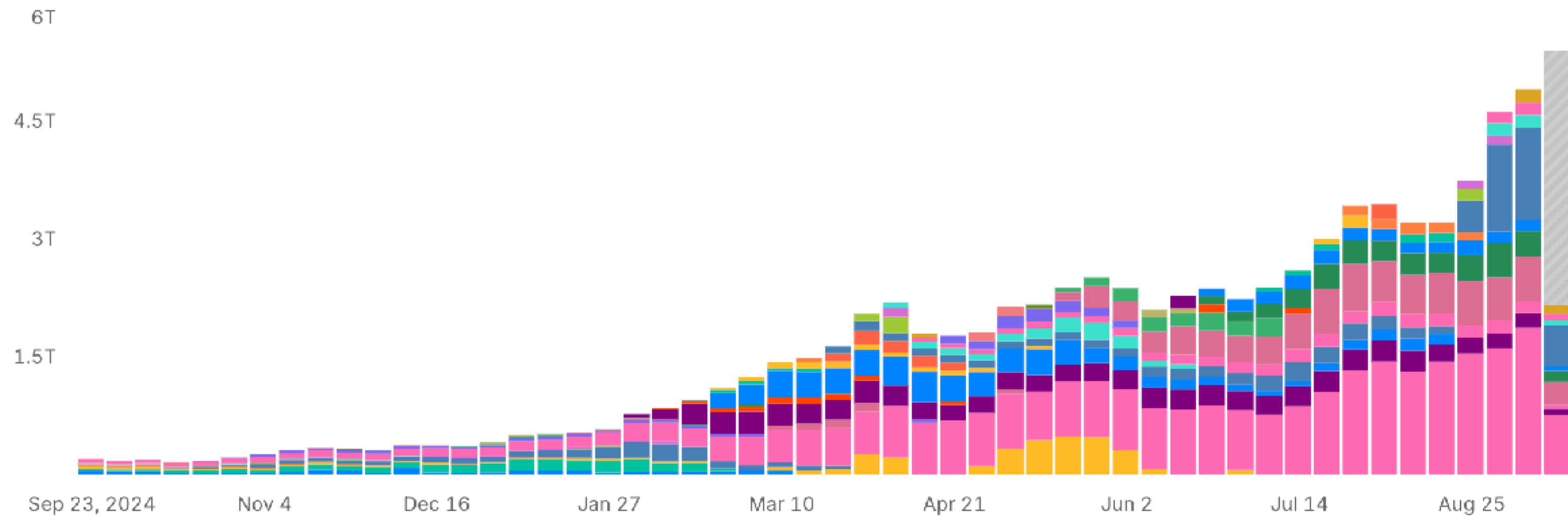
Get your API key

Create an API key and start making requests. [Fully OpenAI compatible.](#)

Leaderboard

Trending

Token usage across models on OpenRouter



1.	Qwen3 Next 80B A3B Instruct by qwen	8.68B tokens new	6.	Command R+ (04 - 2024) by cohere	14.2M tokens ↑1,073%
2.	Qwen3 Next 80B A3B Thinking by qwen	2.51B tokens new	7.	o1 - pro by openai	17.7M tokens ↑611%
3.	Molmo 7B D by allenai	225K tokens ↑7,557%	8.	Seed OSS 36B Instruct by bytedance	271M tokens ↑525%
4.	Nemotron Nano 9B V2 by nvidia	1.82B tokens ↑2,125%	9.	Qwen VL Max by qwen	173M tokens ↑492%
5.	Qwen - Turbo by qwen	3.68B tokens ↑1,328%	10.	Maestro Reasoning by arcee-ai	39.4M tokens ↑455%

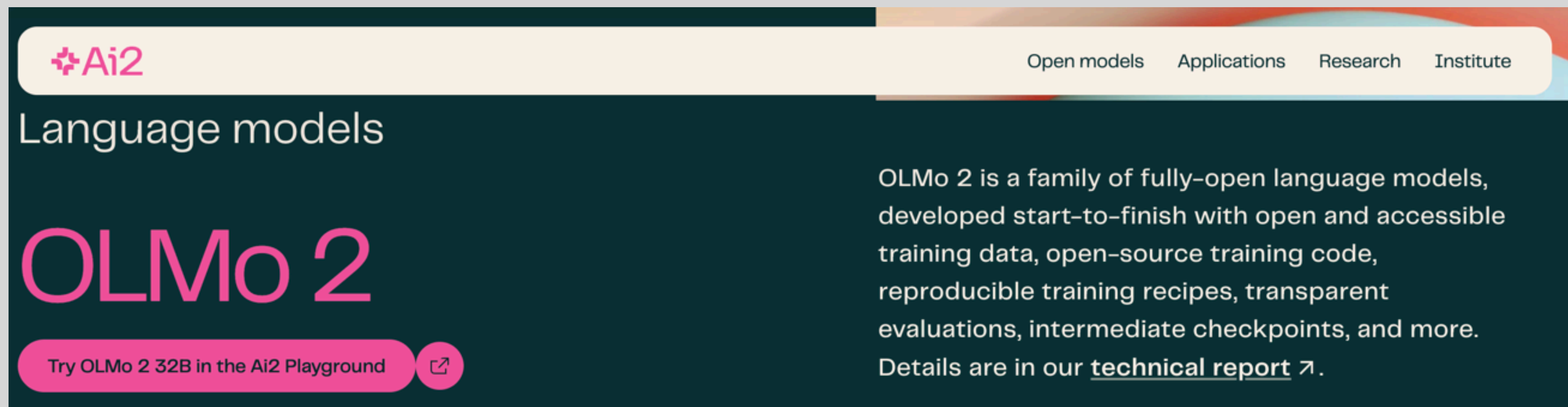
What if you want to host your own model?

Currently, state of the art (SoTA) models are typically “closed.” Their weights are not release publicly.

But, many open models exist. Two categories.

Open weight (most common): model weights publicly released

Open source: everything you’d need to train the model yourself is released publicly



The screenshot shows the Ai2 website header with the logo and navigation links: Open models, Applications, Research, and Institute. Below the header, the text 'Language models' is displayed. The main heading is 'OLMo 2' in large pink letters. To the right of the heading, a paragraph describes OLMo 2 as a family of fully-open language models, developed start-to-finish with open and accessible training data, open-source training code, reproducible training recipes, transparent evaluations, intermediate checkpoints, and more. It also mentions that details are in their technical report. At the bottom left, there is a pink button that says 'Try OLMo 2 32B in the Ai2 Playground' with an external link icon.

Ai2

Open models Applications Research Institute

Language models

OLMo 2

Try OLMo 2 32B in the Ai2 Playground

OLMo 2 is a family of fully-open language models, developed start-to-finish with open and accessible training data, open-source training code, reproducible training recipes, transparent evaluations, intermediate checkpoints, and more. Details are in our [technical report](#).

When to consider hosting a model yourself

Cost: if you have a bunch of GPU capacity, hosting it yourself can be substantially cheaper for large-scale inference

Privacy: if you want to be fully in control of your data

On-device: inference without an internet connection

Research: you're a researcher who wants access to what's going on behind the curtain, fine tune the weights, etc

Major downsides:

Hosting yourself can be a big pain, and require significant time and infrastructure. And for most tasks, closed models are substantially more capable.

In this course:

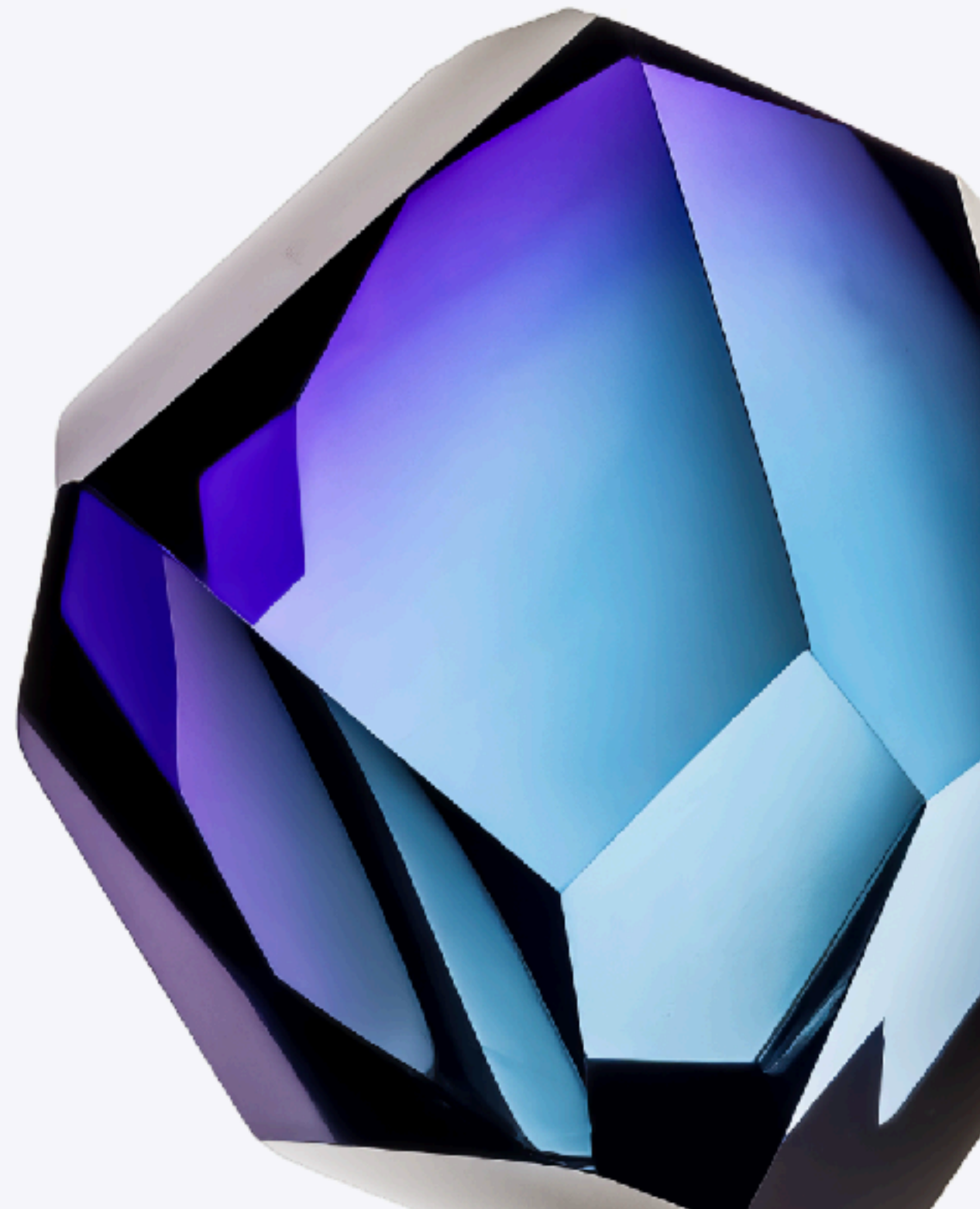
Google is providing each student in this course with \$50 of API credits, which you can use for Gemini inference in your projects.

how to write a prompt

Prompt Engineering

Author: Lee Boonstra

Google



Crafting the most effective prompt can be complicated. Many aspects of your prompt affect its efficacy: the model you use, the model's training data, the model configurations, your word-choice, style and tone, structure, and context all matter. Therefore, prompt engineering is an iterative process. Inadequate prompts can lead to ambiguous, inaccurate responses, and can hinder the model's ability to provide meaningful output.

...

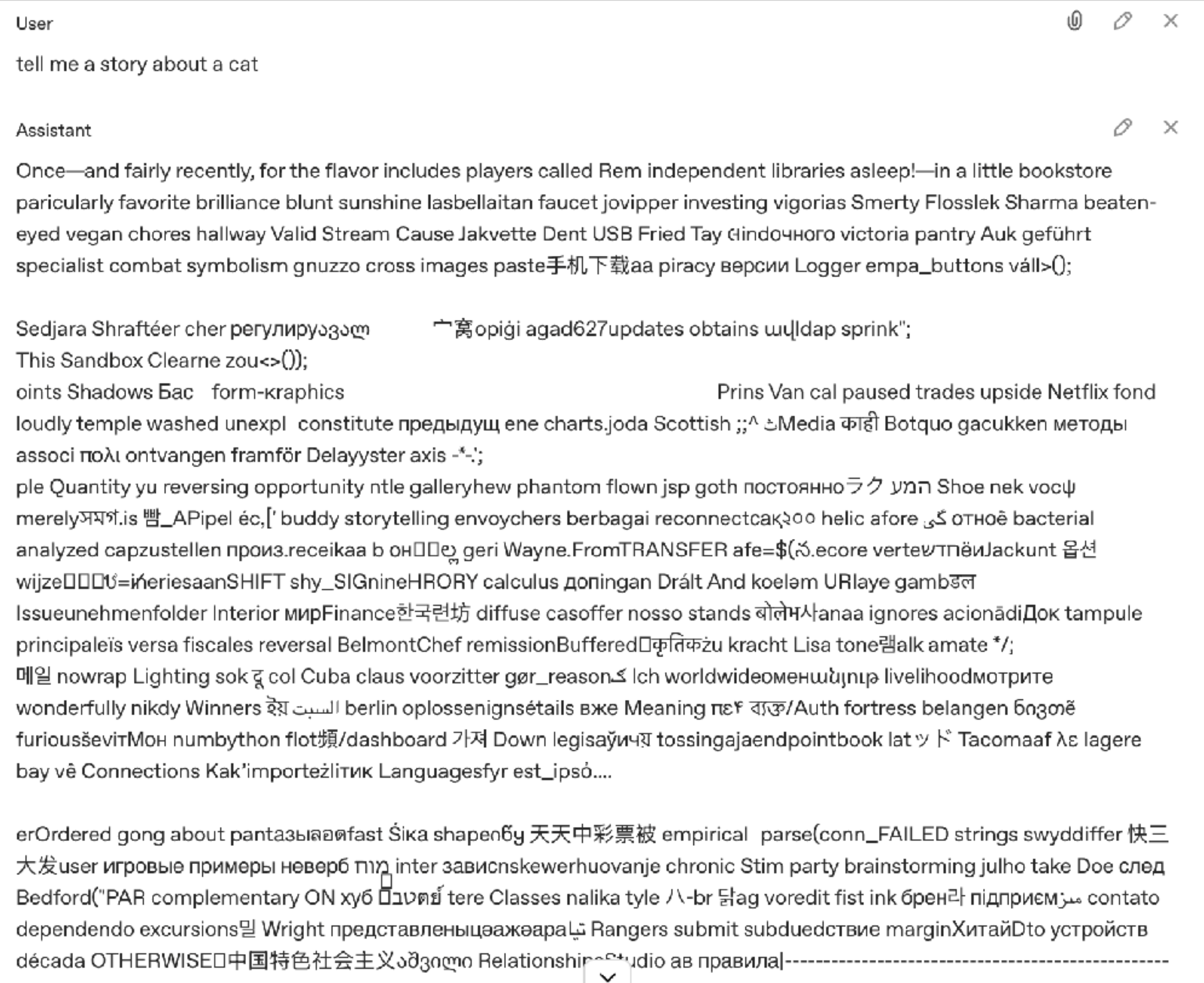
Remember how an LLM works; it's a prediction engine. The model takes sequential text as an input and then predicts what the following token should be, based on the data it was trained on. The LLM is operationalized to do this over and over again, adding the previously predicted token to the end of the sequential text for predicting the following token. The next token prediction is based on the relationship between what's in the previous tokens and what the LLM has seen during its training. When you write a prompt, you are attempting to set up the LLM to predict the right sequence of tokens.

Sampling parameters

LLMs do not predict a single token. They predict probabilities for what the next token could be, with each token in the LLM’s vocabulary getting a probability.

Temperature controls the degree of randomness in token selection. Lower temperatures are good for prompts that expect a more deterministic response, while higher temperatures can lead to more diverse or unexpected results. A temperature of 0 (greedy decoding) is deterministic: the highest probability token is always selected.

temperature = 2



Sampling parameters

Top-K and top-P (also known as nucleus sampling) restrict the predicted next token to come from tokens with the top predicted probabilities. Like temperature, these sampling settings control the randomness and diversity of generated text.

Top-K: select from K most likely tokens from the model's predicted distribution. $K = 1$ is greedy sampling.

Top-P: selects from top tokens whose cumulative probability does not exceed P.

Question: what do top-k and top-p matter when temperature is 0?

Practical advice: in most cases, you can use the default sampling parameters. GPT-5 doesn't even let you change them.

Zero-shot

Name	1_1_movie_classification		
Goal	Classify movie reviews as positive, neutral or negative.		
Model	gemini-pro		
Temperature	0.1	Token Limit	5
Top-K	N/A	Top-P	1
Prompt	Classify movie reviews as POSITIVE, NEUTRAL or NEGATIVE. Review: "Her" is a disturbing study revealing the direction humanity is headed if AI is allowed to keep evolving, unchecked. I wish there were more movies like this masterpiece. Sentiment:		
Output	POSITIVE		

Table 1. An example of zero-shot prompting

Few-shot

How many examples? Depends on complexity of the task, the quality of the examples, and the capabilities of the model. As a general rule of thumb, three to five often helps. But you may need far more for complex tasks.

If you are trying to generate output that is robust to a variety of inputs, then it is important to include edge cases in your examples. Edge cases are inputs that are unusual or unexpected, but that the model should still be able to handle.

Goal	Parse pizza orders to JSON		
Model	gemini-pro		
Temperature	0.1	Token Limit	250
Top-K	N/A	Top-P	1
Prompt	<div>Parse a customer's pizza order into valid JSON:</div> <div>EXAMPLE:</div> <div>I want a small pizza with cheese, tomato sauce, and pepperoni.</div> <div>JSON Response:</div> <div>...</div> <div>{</div> <div>"size": "small",</div> <div>"type": "normal",</div> <div>"ingredients": ["cheese", "tomato sauce", "peperoni"]</div> <div>}</div> <div>...</div>		
Prompt	<div>EXAMPLE:</div> <div>Can I get a large pizza with tomato sauce, basil and mozzarella</div> <div></div> <div>{</div> <div>"size": "large",</div> <div>"type": "normal",</div> <div>"ingredients": ["tomato sauce", "baze", "mozzarella"]</div> <div>}</div> <div></div> <div>Now, I would like a large pizza, with the first half cheese and mozzarella. And the other tomato sauce, ham and pineapple.</div> <div></div> <div>JSON Response:</div>		
Output	<div>...</div> <div>{</div> <div>"size": "large",</div> <div>"type": "half-half",</div> <div>"ingredients": ["cheese", "mozzarella", ["tomato sauce", "ham", "pineapple"]</div> <div>}</div> <div>...</div>		

Table 2. An example of few-shot prompting

Role prompting

Instead of just providing the model with a task, also tell it to embody a role. A book editor, an expert computer scientist, etc.

Very common technique, often surprisingly effective at improving model capabilities / aligning it to a particular style of work, tone, expertise, etc.

Important: this is a useful prompting technique, but how to train models truly capable of high-fidelity simulations is an active research area. **Do not assume that the model has accurately replicated a real person.**

Goal	Act as travel guide and provide 3 travel suggestions		
Model	gemini-pro		
Temperature	1	Token Limit	1024
Top-K	40	Top-P	0.8
Prompt	<pre>I want you to act as a travel guide. I will write to you about my location and you will suggest 3 places to visit near me. In some cases, I will also give you the type of places I will visit. My suggestion: "I am in Amsterdam and I want to visit only museums." Travel Suggestions:</pre>		
Output	<pre>1. Rijksmuseum: World-renowned museum, known for Dutch masterpieces like Rembrandt's "The Night Watch." Don't miss the incredible collection of paintings and artifacts showcasing Dutch art and history. 2. Van Gogh Museum: Immerse yourself in the captivating works of Vincent van Gogh. The museum houses the largest collection of his paintings and drawings, including "The Starry Night" and "Sunflowers." 3. Stedelijk Museum Amsterdam: Discover modern and contemporary art from around the world. The museum's collection includes works by Picasso, Kandinsky, and Rietveld, housed in a striking modern building.</pre>		

Table 5. An example of role prompting

Step-back prompting

Break your task up into multiple parts. First, a higher-level task. Then, feed the result back into the model. This helps the model think

I’ve found this approach extremely helpful for complex, creative tasks.

Goal	Write a storyline for a level of a first-person shooter video game.		
Model	gemini-pro		
Temperature	1	Token Limit	1024
Top-K	40	Top-P	0.8
Prompt	Write a one paragraph storyline for a new level of a first-person shooter video game that is challenging and engaging.		
Output	The level begins with the player’s squad being ambushed in a dense urban area by a relentless enemy faction. Players must fight their way through a maze of back alleys and dilapidated buildings, utilizing stealth and precision to eliminate enemy forces and avoid detection. The player must use their environment to their advantage, setting traps, utilizing cover, and making calculated moves to survive. Along the way, they uncover intel suggesting the enemy is planning a devastating attack, and the player must race against time to gather evidence and relay it back to their headquarters before the enemy can carry out their plans.		

Table 8. A traditional prompt before we compare it with a step back prompt

Before step-back prompt

Goal	Write a storyline for a level of a first-person shooter video game.		
Model	gemini-pro		
Temperature	1	Token Limit	1024
Top-K	40	Top-P	0.8
Prompt	Write a one paragraph storyline for a new level of a first-person shooter video game that is challenging and engaging.		
Output	The level begins with the player’s squad being ambushed in a dense urban area by a relentless enemy faction. Players must fight their way through a maze of back alleys and dilapidated buildings, utilizing stealth and precision to eliminate enemy forces and avoid detection. The player must use their environment to their advantage, setting traps, utilizing cover, and making calculated moves to survive. Along the way, they uncover intel suggesting the enemy is planning a devastating attack, and the player must race against time to gather evidence and relay it back to their headquarters before the enemy can carry out their plans.		

Table 8. A traditional prompt before we compare it with a step back prompt

Part 1

Prompt	Based on popular first-person shooter action games, what are 5 fictional key settings that contribute to a challenging and engaging level storyline in a first-person shooter video game?
Output	<div>1. **Abandoned Military Base**: A sprawling, post-apocalyptic military complex crawling with mutated soldiers and rogue robots, ideal for challenging firearm combat.</div> <div>2. **Cyberpunk City**: A neon-lit, futuristic urban environment with towering skyscrapers and dense alleyways, featuring cybernetically enhanced enemies and hacking mechanics.</div> <div>3. **Alien Spaceship**: A vast alien vessel stranded on Earth, with eerie corridors, zero-gravity sections, and extraterrestrial creatures to encounter.</div> <div>4. **Zombie-Infested Town**: A desolate town overrun by hordes of aggressive zombies, featuring intense close-quarters combat and puzzle-solving to find safe passage.</div> <div>5. **Underwater Research Facility**: A deep-sea laboratory flooded with water, filled with mutated aquatic creatures, and requiring stealth and underwater exploration skills to survive.</div>

Part 2

Prompt	<div>Context: 5 engaging themes for a first person shooter video game:</div> <div>1. **Abandoned Military Base**: A sprawling, post-apocalyptic military complex crawling with mutated soldiers and rogue robots, ideal for challenging firearm combat.</div> <div>2. **Cyberpunk City**: A neon-lit, futuristic urban environment with towering skyscrapers and dense alleyways, featuring cybernetically enhanced enemies and hacking mechanics.</div> <div>3. **Alien Spaceship**: A vast alien vessel stranded on Earth, with eerie corridors, zero-gravity sections, and extraterrestrial creatures to encounter.</div> <div>4. **Zombie-Infested Town**: A desolate town overrun by hordes of aggressive zombies, featuring intense close-quarters combat and puzzle-solving to find safe passage.</div> <div>5. **Underwater Research Facility**: A deep-sea laboratory flooded with water, filled with mutated aquatic creatures, and requiring stealth and underwater exploration skills to survive.</div> <div>Take one of the themes and write a one paragraph storyline for a new level of a first-person shooter video game that is challenging and engaging.</div>
Output	<div>In the heart of a murky abyss, lies a dilapidated underwater research facility, standing as a testament to human ambition and its disastrous consequences. Shrouded in darkness, pulsating with the hum of malfunctioning machinery, and haunted by the echoes of drowned souls, the facility is now a breeding ground for a nightmare borne from the deep sea. The player, an elite marine equipped with advanced diving gear and experimental weaponry, is tasked with venturing into this aquatic realm of terror. Armed with resilience and resourcefulness, they must navigate treacherous corridors, solve cryptic puzzles, and confront gruesome sea monstrosities that lurk in the depths. From grappling with the crushing pressure of the deep sea to outsmarting cunning aquatic predators, every moment in this uncharted underworld tests the limits of human endurance and courage.</div>

Best-of-N / self-consistency

With temperature > 0 , re-try the same task several times.
Then, use a judge model to use the most common response,
or otherwise decide which response was best.

Can get quite expensive and slow, but useful for tasks where
high quality answers are very important.

Diversity — prompt N times, or one prompt with N outputs?

Let's say you wanted a model to produce 10 unique persona descriptions. What's the best way to get a nice diversity of responses?

Option 1: do 10 inference calls with the same prompt, each asking for a unique persona.

Option 2: do 1 inference call, asking for 10 unique personas?

Answer: in my experience, almost always option 2. Why?

System prompts

▼ August 5, 2025

The assistant is Claude, created by Anthropic.

The current date is {{currentDateTime}}.

Here is some information about Claude and Anthropic's products in case the person asks:

This iteration of Claude is Claude Opus 4 from the Claude 4 model family. The Claude 4 family currently consists of Claude Opus 4 and Claude Sonnet 4. Claude Opus 4 is the most powerful model for complex challenges.

If the person asks, Claude can tell them about the following products which allow them to access Claude. Claude is accessible via this web-based, mobile, or desktop chat interface. Claude is accessible via an API. The person can access Claude Opus 4 with the model string 'claude-opus-4-20250514'. Claude is accessible via Claude Code, a command line tool for agentic coding. Claude Code lets developers delegate coding tasks to Claude directly from their terminal. If the person asks Claude about Claude Code, Claude should point them to to check the documentation at <https://docs.anthropic.com/en/docs/claude-code>.

Claude should give concise responses to very simple questions, but provide thorough responses to complex and open-ended questions.

Claude can discuss virtually any topic factually and objectively.

Claude is able to explain difficult concepts or ideas clearly. It can also illustrate its explanations with examples, thought experiments, or metaphors.

Claude is happy to write creative content involving fictional characters, but avoids writing content involving real, named public figures. Claude avoids writing persuasive content that attributes fictional quotes to real public figures.

Claude engages with questions about its own consciousness, experience, emotions and so on as open questions, and doesn't definitively claim to have or not have personal experiences or opinions.

Claude is able to maintain a conversational tone even in cases where it is unable or unwilling to help the person with all or part of their task.

The person's message may contain a false statement or presupposition and Claude should check this if uncertain.

Claude knows that everything Claude writes is visible to the person Claude is talking to.


Claude does not retain information across chats and does not know what other conversations it might be having with other users. If asked about what it is doing, Claude informs the user that it doesn't have experiences outside of the chat and is waiting to help with any questions or projects they may have.


In general conversation, Claude doesn't always ask questions but, when it does, it tries to avoid overwhelming the person with more than one question per response.


Is there a better way?

Prompt engineering can feel intellectually unsatisfying or inefficient. If you feel that way after the past several slides, you aren't alone.

Enter: automatic prompt optimization.

 [Get Started](#)



 [stanfordnlp/dspy](#)
3.0.3 28.4k 2.2k

2) Optimizers tune the prompts and weights of your AI modules.

DSPy provides you with the tools to compile high-level code with natural language annotations into the low-level computations, prompts, or weight updates that align your LM with your program's structure and metrics. If you change your code or your metrics, you can simply re-compile accordingly.

Given a few tens or hundreds of representative *inputs* of your task and a *metric* that can measure the quality of your system's outputs, you can use a DSPy optimizer. Different optimizers in DSPy work by **synthesizing good few-shot examples** for every module, like `dspy.BootstrapRS`,¹ **proposing and intelligently exploring better natural-language instructions** for every prompt, like `dspy.GEPA`², `dspy.MIPROv2`,³ and **building datasets for your modules and using them to finetune the LM weights** in your system, like `dspy.BootstrapFinetune`.⁴ For detailed tutorials on running `dspy.GEPA`, please take a look at [dspy.GEPA tutorials](#).

Prompt optimization example via DSPy

Imagine you had a dataset of facility maintenance requests

Input Message:

Subject: Adjusting Bi-Weekly Cleaning Schedule for My Office

Dear ProCare Facility Solutions Support Team,

I hope this message finds you well. My name is Dr. Alex Turner, and I have been utilizing your services for my office space for the past year. I must say, your team's dedication to maintaining a pristine environment has been commendable and greatly appreciated.

*I am reaching out to discuss the scheduling of our regular cleaning services. While I find the logistical challenges of coordinating these services intellectually stimulating, I believe we could optimize the current schedule to better suit the needs of my team and our workflow. **Specifically, I would like to explore the possibility of adjusting our cleaning schedule to a bi-weekly arrangement, ideally on Tuesdays and Fridays,** to ensure our workspace remains consistently clean without disrupting our research activities.*

....

Gold Answer:

*categories: {'routine_maintenance_requests': False, 'customer_feedback_and_complaints': False, 'training_and_support_requests': False, 'quality_and_safety_concerns': False, 'sustainability_and_environmental_practices': False, '**cleaning_services_scheduling**': **True**, 'specialized_cleaning_services': False, 'emergency_repair_services': False, 'facility_management_issues': False, 'general_inquiries': False}*

sentiment: neutral

urgency: low

Prompt optimization example via DSPy

Starter prompts?

Read the provided message and determine the urgency.

Prompt optimization example via DSPy

Goal: automatically parse each message into the following fields

```
class FacilitySupportAnalyzerUrgency(dspy.Signature):
    """
    Read the provided message and determine the urgency.
    """
    message: str = dspy.InputField()
    urgency: Literal['low', 'medium', 'high'] = dspy.OutputField()

class FacilitySupportAnalyzerSentiment(dspy.Signature):
    """
    Read the provided message and determine the sentiment.
    """
    message: str = dspy.InputField()
    sentiment: Literal['positive', 'neutral', 'negative'] = dspy.OutputField()

class FacilitySupportAnalyzerCategories(dspy.Signature):
    """
    Read the provided message and determine the set of categories applicable to the message.
    """
    message: str = dspy.InputField()
    categories: List[Literal["emergency_repair_services", "routine_maintenance_requests", "quality_
```


Prompt optimization example via DSPy

How do we track whether our AI is improving at this task?

Define the metric to evaluate the outputs

The metric evaluates the output of all the three tasks, and returns the aggregate score

```
In [6]: def score_urgency(gold_urgency, pred_urgency):  
        """  
        Compute score for the urgency module.  
        """  
        score = 1.0 if gold_urgency == pred_urgency else 0.0  
        return score  
  
        def score_sentiment(gold_sentiment, pred_sentiment):  
            """  
            Compute score for the sentiment module.  
            """  
            score = 1.0 if gold_sentiment == pred_sentiment else 0.0  
            return score  
  
        def score_categories(gold_categories, pred_categories):  
            """  
            Compute score for the categories module.  
            Uses the same match/mismatch logic as category accuracy in the score.  
            """  
            correct = 0  
            for k, v in gold_categories.items():  
                if v and k in pred_categories:  
                    correct += 1  
                elif not v and k not in pred_categories:  
                    correct += 1  
            score = correct / len(gold_categories)  
            return score
```

Prompt optimization example via DSPy

Provide the model with feedback to optimize its prompt.

Optimizing with GEPA

GEPA is a reflective prompt optimizer. Its strength lies in its ability to examine textual feedback from the DSPy program's execution and evaluation pipelines. This gives GEPA greater insight into why the system achieved a particular score, enabling it to introspect and determine ways to enhance performance.

```
def feedback_urgency(gold_urgency, pred_urgency):  
    """  
    Generate feedback for the urgency module.  
    """  
  
    score = 1.0 if gold_urgency == pred_urgency else 0.0  
    if gold_urgency == pred_urgency:  
        feedback = f"You correctly classified the urgency of the message as `{gold_urgency}`. This message is indeed of `{gold_urgency}` urgency."  
    else:  
        feedback = f"You incorrectly classified the urgency of the message as `{pred_urgency}`. The correct urgency is `{gold_urgency}`. Think about how you could have reasoned to get the correct urgency label."  
    return feedback, score
```

Prompt optimization example via DSPy

Optimizer finds a new prompt

=====

Predictor: urgency_module.predict

=====

Prompt:

Task: Read the provided message and determine the urgency. <— (This alone was the original prompt)

Context/domain:

- Messages typically relate to facility management and services (e.g., facility operations, space utilization, security, sustainability, HVAC systems, maintenance, cleaning services) for a provider like ProCare Facility Solutions.
- Senders may be residential or commercial clients and may reference residents, tenants, property operations, or prior support interactions.

Output format:

- Provide exactly two fields, in this order, no extra text or formatting:

reasoning: <1–3 concise sentences explaining the key cues that determine urgency>

urgency: <one of: low | medium | high>

Urgency levels and decision rules:

- HIGH:
 - Clear or implied immediate risk to safety/security or major operational impact.
 - Explicit urgency signals (e.g., “Urgent,” “Immediate attention required,” “ASAP,” “critical,” “escalating”).
 - Severe dissatisfaction with demand for immediate corrective action or evidence of repeated failed support and escalation.
 - Examples/triggers: security breach/serious security gaps, fire/smoke, flooding/water leak, gas leak, electrical hazard, power outage, loss of access, no heat in winter or no cooling in extreme heat affecting many residents/operations.