Figure 1-10. Photo by Jason Goodman on Unsplash

CAUTION

These examples demonstrate the capabilities of image generation models, but we would exercise caution when uploading base images for use in prompts. Check the licensing of the image you plan to upload and use in your prompt as the base image, and avoid using clearly copyrighted images. Doing so can land you in legal trouble and is against the terms of service for all the major image generation model providers.

---

4. Evaluate Quality

As of yet, there has been no feedback loop to judge the quality of your responses, other than the basic trial and error of running the prompt and seeing the results, referred to as *blind prompting*. This is fine when your prompts are used temporarily for a single task and rarely revisited. However, when you're reusing the same prompt multiple times or building a production application that relies on a prompt, you need to be more rigorous with measuring results.

There are a number of ways performance can be evaluated, and it depends largely on what tasks you're hoping to accomplish. When a new AI

model is released, the focus tends to be on how well the model did on *evals* (evaluations), a standardized set of questions with predefined answers or grading criteria that are used to test performance across models. Different models perform differently across different types of tasks, and there is no guarantee a prompt that worked previously will translate well to a new model. OpenAI has open-sourced its evals framework for benchmarking performance of LLMs, and encourages others to contribute additional eval templates.

In addition to the standard academic evals there are also more headline-worthy tests like GPT-4 passing the bar exam. Evaluation is difficult for more subjective tasks, and can be time-consuming or prohibitively costly for smaller teams. In some instances researchers have turned to using more advanced models like GPT-4 to evaluate responses from less sophisticated models, as was done with the release of Vicuna-13B, a fine-tuned model based on Meta's LLaMA open-source model (see Figure 1-11).

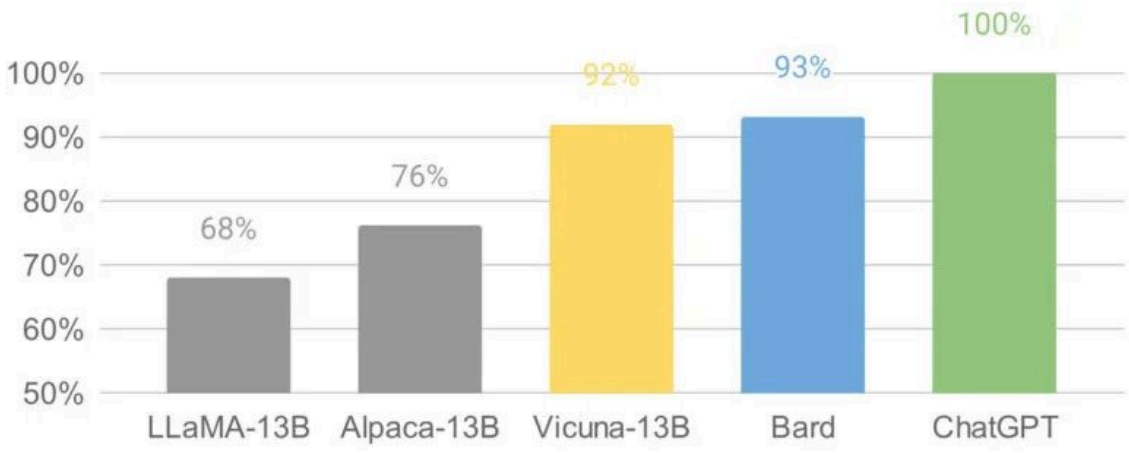

Figure 1-11. Vicuna GPT-4 Evals

More rigorous evaluation techniques are necessary when writing scientific papers or grading a new foundation model release, but often you will only need to go just one step above basic trial and error. You may find that a simple thumbs up / thumbs down rating system implemented in a Jupyter Notebook can be enough to add some rigor to prompt optimization, without adding too much overhead. One common test is to see whether providing examples is worth the additional cost in terms of prompt length, or whether you can get away with providing no examples in the prompt. The first step is getting responses for multiple runs of each

prompt, and storing them in a spreadsheet, which we will do after setting up our environment.

You can install the OpenAI Python package with `pip install openai`. If you're running into compatability issues with this package, create a virtual environment and install our *requirements.txt* (instructions in the preface).

To utilize the API, you'll need to create an OpenAI account and then navigate here for your API key.

> **IMPORTANT**
>
> Hardcoding API keys in scripts is not recommended due to security reasons. Instead, utilize environment variables or configuration files to manage your keys.

Once you have an API key, it's crucial to assign it as an environment variable by executing the following command, replacing `api_key` with your actual API key value:

```
export OPENAI_API_KEY="api_key"
```

Or on Windows:

```
set OPENAI_API_KEY=api_key
```

Alternatively if you'd prefer not to pre-set an API key then you can manually set the key while initializing the model, or load it from an *.env* file using python-dotenv. First, install the library `pip install python-dotenv`, then load the environment variables with the following code at the top of your script or notebook:

```
from dotenv import load_dotenv

load_dotenv()  # take environment variables from .env.
```

The first step is getting responses for multiple runs of each prompt, and storing it in a spreadsheet.

Input:

```
# Define two variants of the prompt to test zero-shot
# vs few-shot
prompt_A = """Product description: A pair of shoes that can
fit any foot size.
Seed words: adaptable, fit, omni-fit.
Product names:"""

prompt_B = """Product description: A home milkshake maker.
Seed words: fast, healthy, compact.
Product names: HomeShaker, Fit Shaker, QuickShake, Shake
Maker

Product description: A watch that can tell accurate time in
space.
Seed words: astronaut, space-hardened, eliptical orbit
Product names: AstroTime, SpaceGuard, Orbit-Accurate,
EliptoTime.

Product description: A pair of shoes that can fit any foot
size.
Seed words: adaptable, fit, omni-fit.
Product names:"""

test_prompts = [prompt_A, prompt_B]

import pandas as pd
from openai import OpenAI
import os

# Set your OpenAI key as an environment variable
# https://platform.openai.com/api-keys
client = OpenAI(
```

```
  api_key=os.environ['OPENAI_API_KEY'],  # Default
)

def get_response(prompt):
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=[
            {
                "role": "system",
                "content": "You are a helpful assistant."
            },
            {
                "role": "user",
                "content": prompt
            }
        ]
    )
    return response.choices[0].message.content

# Iterate through the prompts and get responses
responses = []
num_tests = 5

for idx, prompt in enumerate(test_prompts):
    # prompt number as a letter
    var_name = chr(ord('A') + idx)

    for i in range(num_tests):
        # Get a response from the model
        response = get_response(prompt)

        data = {
            "variant": var_name,
            "prompt": prompt,
            "response": response
            }
        responses.append(data)

# Convert responses into a dataframe
df = pd.DataFrame(responses)

# Save the dataframe as a CSV file
df.to_csv("responses.csv", index=False)
```

```
print(df)
```

Output:

```
  variant                                             prompt
  \
0       A  Product description: A pair of shoes that can ...
1       A  Product description: A pair of shoes that can ...
2       A  Product description: A pair of shoes that can ...
3       A  Product description: A pair of shoes that can ...
4       A  Product description: A pair of shoes that can ...
5       B  Product description: A home milkshake maker.\n...
6       B  Product description: A home milkshake maker.\n...
7       B  Product description: A home milkshake maker.\n...
8       B  Product description: A home milkshake maker.\n...
9       B  Product description: A home milkshake maker.\n...

                                            response
0  1. Adapt-a-Fit Shoes \n2. Omni-Fit Footwear \n...
1  1. OmniFit Shoes\n2. Adapt-a-Sneaks \n3. OneFi...
2  1. Adapt-a-fit\n2. Flexi-fit shoes\n3. Omni-fe...
3  1. Adapt-A-Sole\n2. FitFlex\n3. Omni-FitX\n4. ...
4  1. Omni-Fit Shoes\n2. Adapt-a-Fit Shoes\n3. An...
5  Adapt-a-Fit, Perfect Fit Shoes, OmniShoe, OneS...
6       FitAll, OmniFit Shoes, SizeLess, AdaptaShoes
7       AdaptaFit, OmniShoe, PerfectFit, AllSizeFit.
8  FitMaster, AdaptoShoe, OmniFit, AnySize Footwe...
9        Adapt-a-Shoe, PerfectFit, OmniSize, FitForm
```

Here we're using the OpenAI API to generate model responses to a set of prompts and storing the results in a dataframe, which is saved to a CSV file. Here's how it works:

1. Two prompt variants are defined, and each variant consists of a product description, seed words, and potential product names, but `prompt_B` provides two examples.
2. Import statements are called for the pandas library, openai library, and os library.