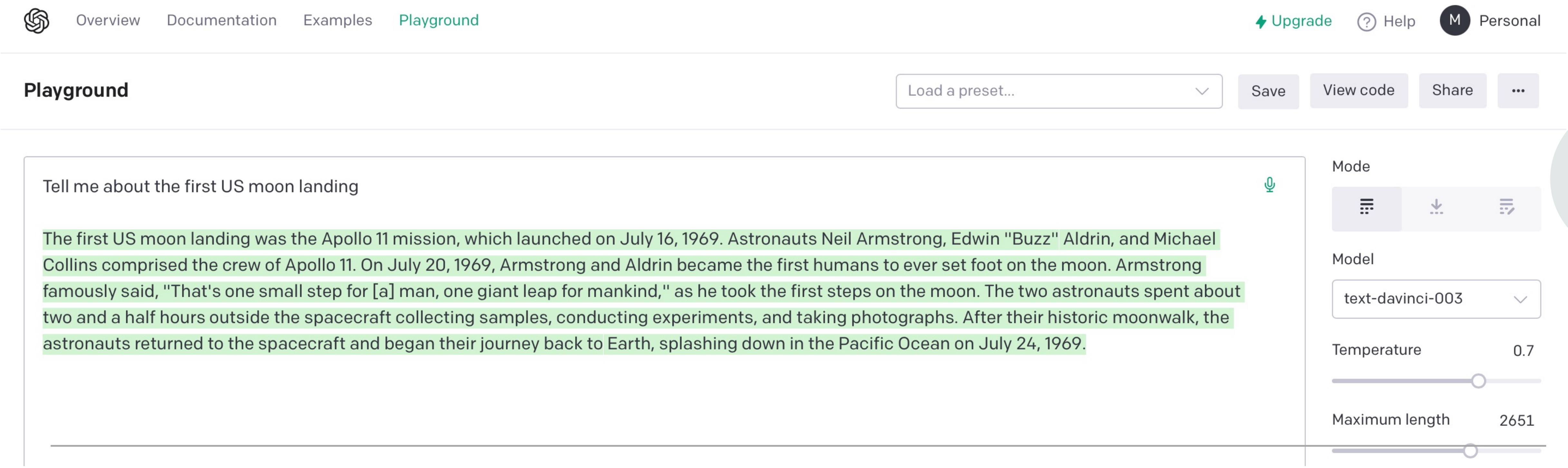
Generative AI Tools

First ideas and experiences

Michael A. Heroux

What are Generative AI Tools?

- Produce content by
  - Accepting an input prompt
  - Producing human-like output in response to the prompt
- Example:
  - Prompt: *Tell me about the first US moon landing*
  - https://beta.openai.com



Tell me about the first US moon landing

The first US moon landing was the Apollo 11 mission, which launched on July 16, 1969. Astronauts Neil Armstrong, Edwin "Buzz" Aldrin, and Michael Collins comprised the crew of Apollo 11. On July 20, 1969, Armstrong and Aldrin became the first humans to ever set foot on the moon. Armstrong famously said, "That's one small step for [a] man, one giant leap for mankind," as he took the first steps on the moon. The two astronauts spent about two and a half hours outside the spacecraft collecting samples, conducting experiments, and taking photographs. After their historic moonwalk, the astronauts returned to the spacecraft and began their journey back to Earth, splashing down in the Pacific Ocean on July 24, 1969.

Model: text-davinci-003

Temperature: 0.7

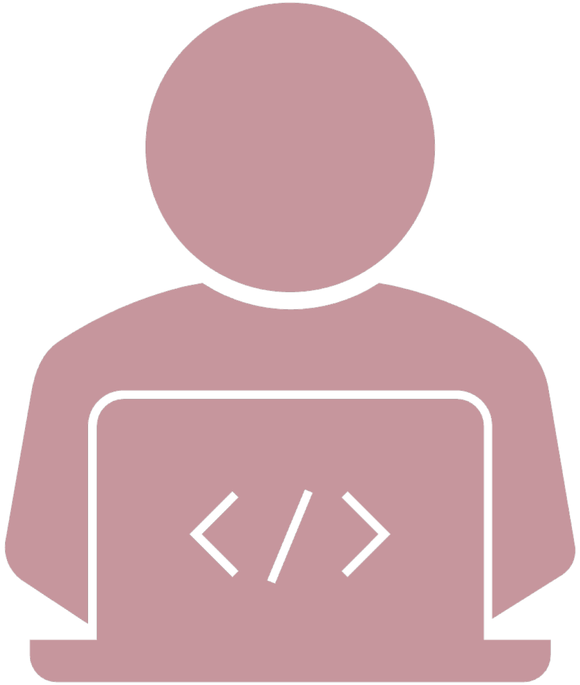
Maximum length: 2651

Tools I have used

- OpenAI: Company producing most-used tools
  - GitHub Copilot:
    - Have been tracking since Dec 2021
    - Based on Codex: special OpenAI model for programming
    - Available as a plugin for VSCode (how I use it)
  - OpenAI Playground:
    - Like ChatGPT with more flexible, complicated API (to GPT3)
    - Less known, more available
  - ChatGPT:
    - Latest API (to GPT3.5)
    - Raised broad awareness.
- Not tried: Jasper (also based on GPT3), Code Whisperer (Amazon)

Some ways to think about these tools

- College roommate English major who lives to help you with your writing assignments
- Programmer to produce code, scripts, templates as starting point for you to consider
- Administrative assistant to help with detailed content formatting tasks
- Elaborate auto-complete feature
- An on-demand intelligence



English major roommate

- Authoring
  - Brainstorm topics
  - Generate outline
  - Recursively provide more detail
- Assessment
  - Request AI review of content
- Improvement
  - Steel-manning other perspectives
  - Refine content – understandable by a novice, TL;DR, etc.

What does it mean to be the author?

- Student essay due by midnight:
  - Written with copy-and-paste and “lite” editing after dinner
  - Is the student the author?
- Definition of author here:
  - Ability to explain, defend, and elaborate on written content?
  - True novelty seems unlikely
- How to assess knowledge:
  - Assessing quality of writing OK but not content knowledge
  - Assessment via dialogue seems essential

Skills impact

- Low-level skills replaced by tools
  - Programming:
    - Syntax, formatting
    - Programming standards
    - Using APIs
- What to produce and how it's designed are relatively more important

Trends

- Soft skills become relatively more important
  - Understanding the individual, teams, communities become more important that technical skills
- Domain knowledge relatively more important
  - What to produce – Requirements
- Design become more important & easier
  - Rapid prototyping easier
- AI tools won't replace people
  - People who use AI tools will replace those who don't

https://beta.openai.com/playground
Bootstrapping a scripting tool – Step 1: Ask for advice

What is a good way to extract metadata from a GitHub repo?

The best way to extract metadata from a GitHub repo is to use the GitHub API. It provides a range of endpoints that allow you to access information about the repo, such as the number of commits, pull requests, releases, and other metadata. You can also use the GitHub GraphQL API to query more specific information.

Write a script to extract basic activity information from the Trilinos GitHub repo

```
#!/usr/bin/env python3

import requests

# Enter your GitHub personal access token here
token = '<YOUR_TOKEN_HERE>'

# The GitHub repository to query
repo = 'trilinos/Trilinos'

# Get the list of commits
url = f'https://api.github.com/repos/{repo}/commits'
headers = {'Authorization': f'token {token}'}
response = requests.get(url, headers=headers).json()

# Print the commit count
print(f'Commit count: {len(response)}')

# Get the list of pull requests
url = f'https://api.github.com/repos/{repo}/pulls'
headers = {'Authorization': f'token {token}'}
response = requests.get(url, headers=headers).json()

# Print the pull request count
print(f'Pull request count: {len(response)}')

# Get the list of releases
url = f'https://api.github.com/repos/{repo}/releases'
headers = {'Authorization': f'token {token}'}
response = requests.get(url, headers=headers).json()

# Print the release count
print(f'Release count: {len(response)}')
```

https://beta.openai.com/playground

Step 2: Generate script

https://beta.openai.com/playground
Step 3: Learn about token

How do I determine my github personal access token?

To generate a GitHub personal access token, you must first log in to your GitHub account. Once logged in, go to your Settings page, select “Developer settings” from the left sidebar, and then “Personal access tokens”. On the next page, click “Generate new token”. You will then be prompted to give your token a name and select the scopes for which the token will be granted access. Once you have set your token’s name and scopes, click “Generate token”. Your token will then be displayed on the next page.

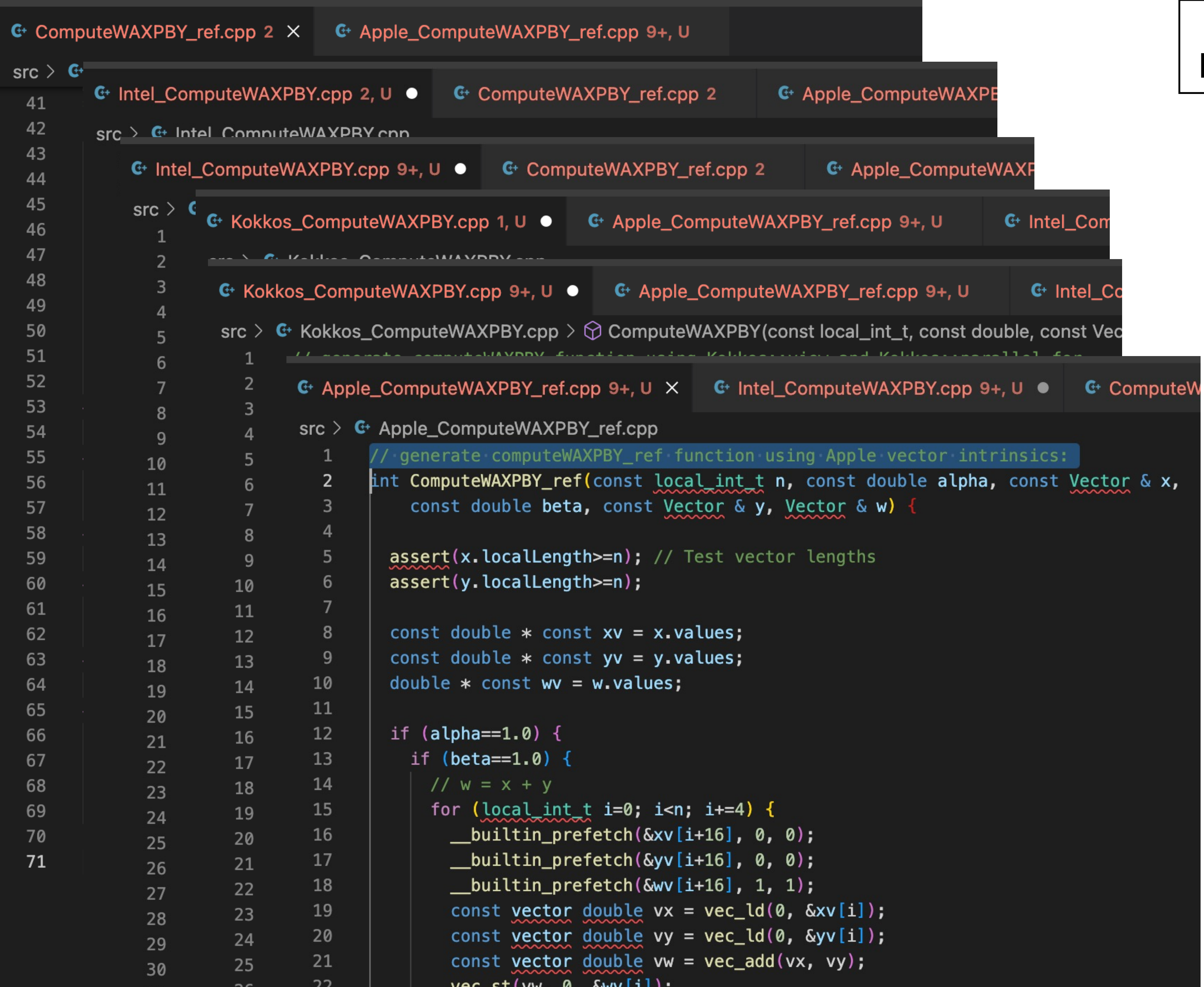
https://beta.openai.com/playground

Step 4: Experiment

Write a github API python script to list the contributors and number of commits to a github repo with user provided name

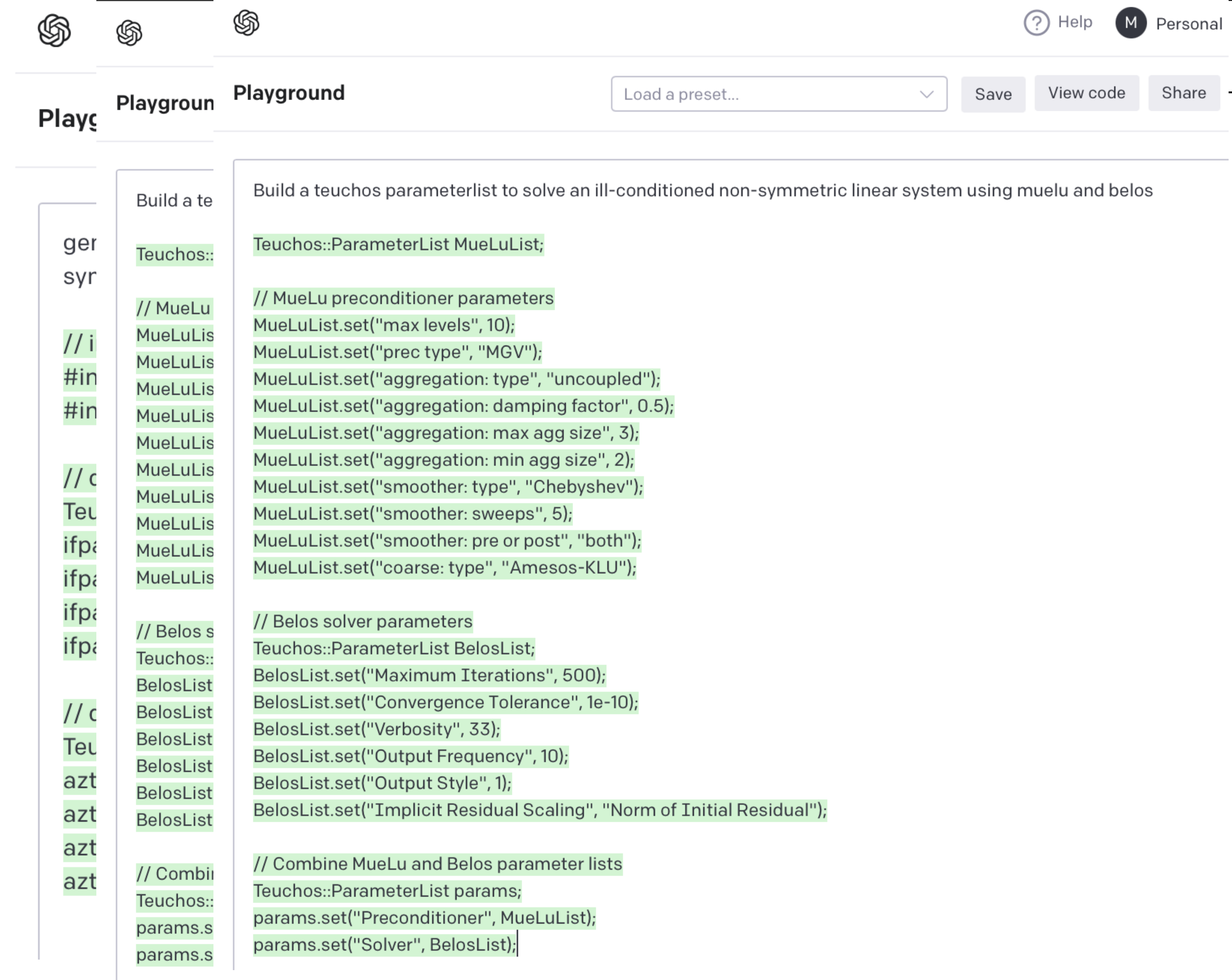
```
import requests
import json


# Get the user input
user_input = input("Please enter the GitHub repo name: ")


# API call to get the list of contributors
url = "https://api.github.com/repos/{}/contributors".format(user_input)
r = requests.get(url)


# Parse the response
data = json.loads(r.text)


# Print the result
print("List of Contributors and Number of Commits to {}".format(user_input))
for contributor in data:
    print("{}: {}".format(contributor['login'], contributor['contributions']))
```

VSCode+Copilot
HPCG Benchmark WAXPBY Kernel


ComputeWAXPBY_ref.cpp 2
Apple_ComputeWAXPBY_ref.cpp 9+, U
Intel_ComputeWAXPBY.cpp 2, U
src > Intel_ComputeWAXPBY.cpp
Intel_ComputeWAXPBY.cpp 9+, U
Kokkos_ComputeWAXPBY.cpp 1, U
src > Kokkos_ComputeWAXPBY.cpp > ComputeWAXPBY(const local_int_t, const double, const Vec
Kokkos_ComputeWAXPBY.cpp 9+, U
Apple_ComputeWAXPBY_ref.cpp 9+, U
Intel_ComputeWAXPBY.cpp 9+, U
src > Apple_ComputeWAXPBY_ref.cpp

```
// generate computeWAXPBY_ref function using Apple vector intrinsics:
int ComputeWAXPBY_ref(const local_int_t n, const double alpha, const Vector & x,
    const double beta, const Vector & y, Vector & w) {

  assert(x.localLength>=n); // Test vector lengths
  assert(y.localLength>=n);

  const double * const xv = x.values;
  const double * const yv = y.values;
  double * const wv = w.values;

  if (alpha==1.0) {
    if (beta==1.0) {
      // w = x + y
      for (local_int_t i=0; i<n; i+=4) {
        __builtin_prefetch(&xv[i+16], 0, 0);
        __builtin_prefetch(&yv[i+16], 0, 0);
        __builtin_prefetch(&wv[i+16], 1, 1);
        const vector double vx = vec_ld(0, &xv[i]);
        const vector double vy = vec_ld(0, &yv[i]);
        const vector double vw = vec_add(vx, vy);
```

OpenAI Playground
AztecOO Belos Parameters



Build a teuchos parameterlist to solve an ill-conditioned non-symmetric linear system using muelu and belos

```
Teuchos::ParameterList MueLuList;

// MueLu preconditioner parameters
MueLuList.set("max levels", 10);
MueLuList.set("prec type", "MGV");
MueLuList.set("aggregation: type", "uncoupled");
MueLuList.set("aggregation: damping factor", 0.5);
MueLuList.set("aggregation: max agg size", 3);
MueLuList.set("aggregation: min agg size", 2);
MueLuList.set("smoother: type", "Chebyshev");
MueLuList.set("smoother: sweeps", 5);
MueLuList.set("smoother: pre or post", "both");
MueLuList.set("coarse: type", "Amesos-KLU");

// Belos solver parameters
Teuchos::ParameterList BelosList;
BelosList.set("Maximum Iterations", 500);
BelosList.set("Convergence Tolerance", 1e-10);
BelosList.set("Verbosity", 33);
BelosList.set("Output Frequency", 10);
BelosList.set("Output Style", 1);
BelosList.set("Implicit Residual Scaling", "Norm of Initial Residual");

// Combine MueLu and Belos parameter lists
Teuchos::ParameterList params;
params.set("Preconditioner", MueLuList);
params.set("Solver", BelosList);
```