Voice Dev 2

CS571: Building User Interfaces

Cole Nelson & Yaxin Hu

Today's Warmup

- Clone today's code to your machine.
  - Run the command `npm install` inside of the `starter` and `solution` folders.

Learning Objectives

1. Be able to work with generative AI
2. Be able to work with streamed responses
3. Be able to explain the event loop

Notes on LLMs/AI/Thinking Machines

1. This is new and experimental! Please use Piazza to report any issues.
2. There are no guarantees in what an AI agent generates.
3. Please be appropriate in your requests; all traffic is linked back to your `wisc.edu` account.
4. Please use our AI API for this exercise and HW11 only; do not use it for personal use.

Voice Dev: A Comparison

Voice Dev 1: Use *Wit.AI* to develop a *command-and-control* agent that matches *utterances* to *intents*.

Voice Dev 2: Use *OpenAI** to develop a general-purpose *conversational* agent that digests and produces *tokens*.

* Our AI API wraps GPT-4o-mini with a similar (but simplified) API.

Generative AI - Background

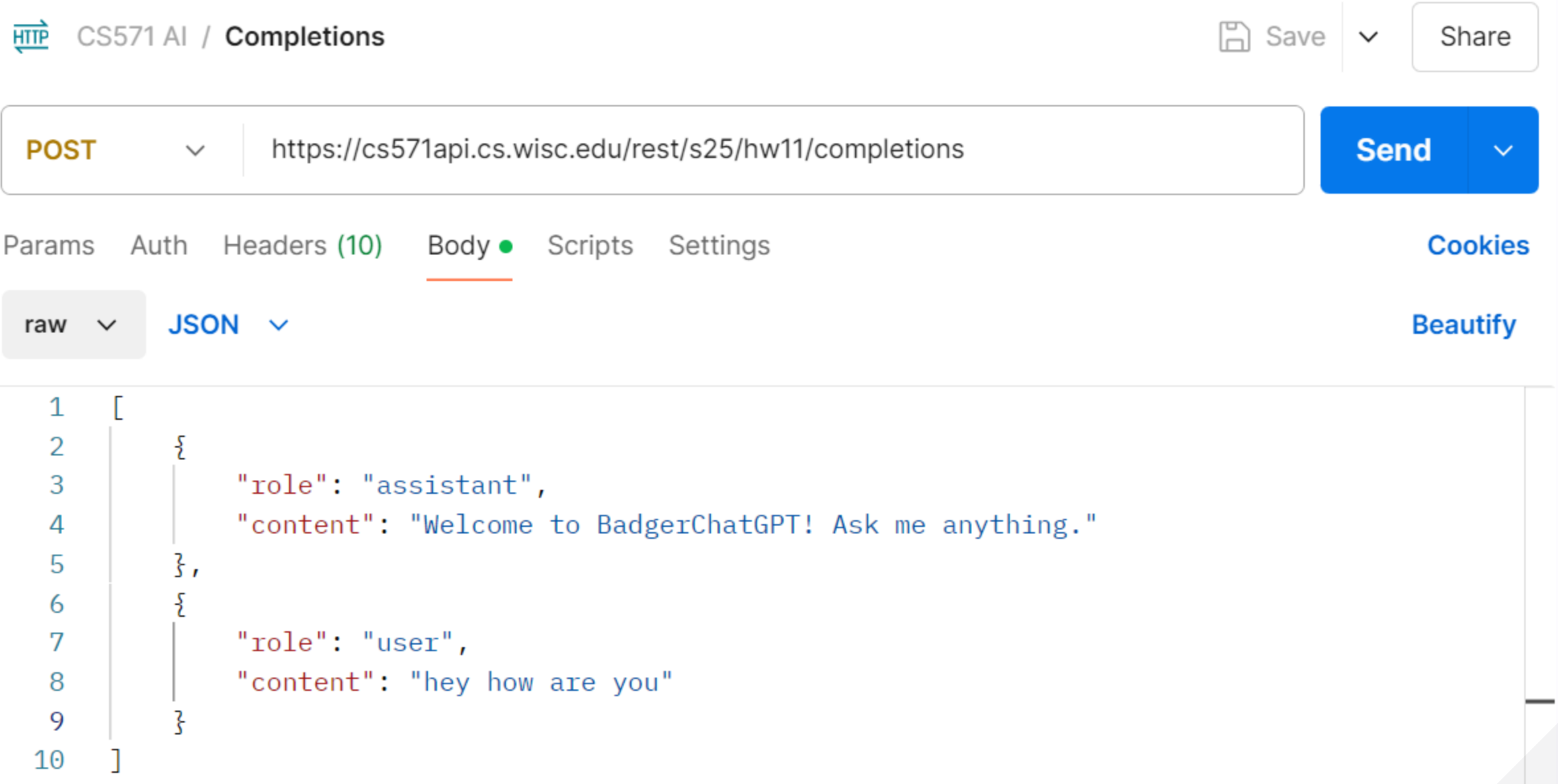
A model is trained on some data. This takes **a lot** of time, resources, and energy! This training gives it the *parameters* it needs to perform its generation

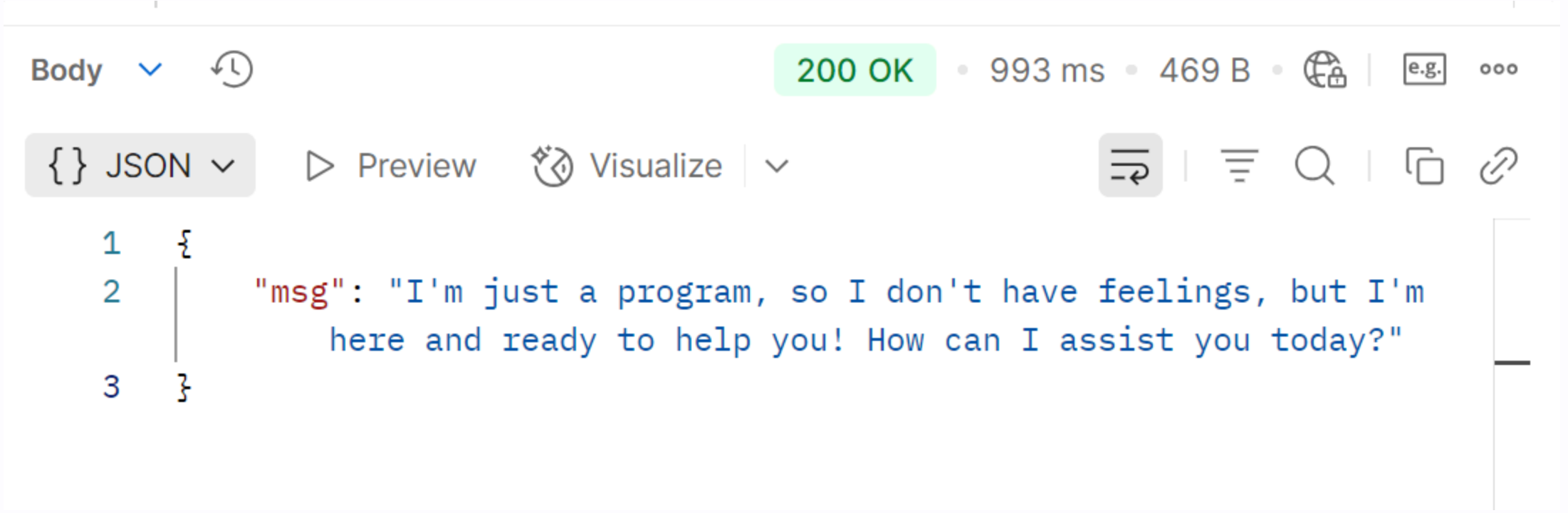
Given some input tokens and these parameters, it produces the most likely output tokens. These can be generated *relatively* quickly.

We will *not* focus on *the* model, we will focus on interacting with *a* model.

Working with GenAI

We typically interact with an AI agent through an *API*.
The CS571 AI API is based loosely on the OpenAI API.


CS571 AI / Completions
Save
Share
POST
https://cs571api.cs.wisc.edu/rest/s25/hw11/completions
Send
Params
Auth
Headers (10)
Body
Scripts
Settings
Cookies
raw
JSON
Beautify
[
{
"role": "assistant",
"content": "Welcome to BadgerChatGPT! Ask me anything."
},
{
"role": "user",
"content": "hey how are you"
}
]


Body
200 OK
993 ms
469 B
e.g.
{ } JSON
Preview
Visualize
1 {
2 "msg": "I'm just a program, so I don't have feelings, but I'm here and ready to help you! How can I assist you today?"
3 }

Generative AI - Messages

A message has a `role` and `content`.

Roles include...

- `platform`: By OpenAI; content restrictions and moderation. Cannot be changed.
- `developer`: By you; directive and purpose of agent.
- `assistant`: By agent; response to user queries.
- `user`: By client; queries and questions.

Generative AI - `developer` Message

```
{
    "role": "developer",
    "content": "You are a helpful agent
        working with students at UW-Madison.
        They are trying to enroll in college courses.
        You are there to help them with their college
        course enrollment. Do not help with any
        enrollment that is not UW-Madison."
},
```

This is just an example! Set the `developer` directive to be relevant to your specific agent.

CS571 HW11 AI API

Explore today's Postman Collection!

Generative AI - Working With

Our goal: Maintain an ongoing conversation, appending the `developer`, `assistant`, and `user` messages as they happen.

```
[
  {
    "role": "developer",
    "content": "You are a helpful agent..."
  }
];
```

Generative AI - Working With

When we want an AI response, we simply send the current conversation...

```
fetch("https://cs571api.cs.wisc.edu/rest/s25/hw11/completions", {
    method: "POST",
    headers: {
        "X-CS571-ID": `ENTER_YOUR_BID`,
        "Content-Type": "application/json"
    },
    body: JSON.stringify(CONVERSATION)
})
```

Generative AI - Working With

... and deal with the response like any other `fetch`

```
fetch(.......)
.then(res => res.json())
.then(data => {
  console.log("The AI sent back...");
  console.log(data);
});
```

Your turn!

Add generative AI to today's starter code.

Markdown

Generative AI often produces markdown in its response; we can use react-markdown to handle this!

GenAI - Streaming

This can take time! What do we do while we wait?

Instead, let's stream the response. **See Postman.**

Generative AI - Streaming

Finally, not doing `resp.json()` ! Instead...

```
const reader = resp.body.getReader();
const decoder = new TextDecoder("utf-8");
```

As the response body is streamed, read the data that is coming across!

Generative AI - Streaming

```
let done = false;
while (!done) {
    // respObj is an object containing two properties:
    // - value: the value read (encoded)
    // - done: if was last value
    const respObj = await reader.read();
}
```

Generative AI - Streaming

```
let done = false;
while (!done) {
    const respObj = await reader.read();
    const value = respObj.value;
    done = respObj.done;
    if (value) {
        // Chunk can contain many lines, see docs!
        const chunk = decoder.decode(value, { stream: true });
    }
}
```

Generative AI - Streaming

```
let done = false;
while (!done) {
    const respObj = await reader.read();
    const value = respObj.value;
    done = respObj.done;
    if (value) {
        const chunk = decoder.decode(value, { stream: true });
        const lines = chunk.split("\n").filter(line => line.trim() !== "");
        for (const line of lines) {
            let deltaObj = JSON.parse(line);
            console.log(deltaObj);
        }
    }
}
console.log("Done!");
```

Your turn!

Stream the generative AI response instead.

Is this performant?

Do we *really* wait in an infinite loop?

Is this performant?

Yes, it is performant.

Just like Promises, `await` is non-blocking. But, thanks for asking! 🙂 Let's discuss **the event loop.**

The Event Loop

See attached presentation.

Questions?