

Supporting Self-Directed Learning with AI

Improving Student Interaction with Technical Documentation

Aidan Szuch, Akanksha Girdhar, and Andrew DeOrio

University of Michigan

Motivation

Generative AI usage among students is rapidly increasing.

Students often use AI to obtain direct solutions.

This can:

- Reduce engagement with technical documentation.
- Undermine learning objectives.
- Increase reliance on potentially incorrect information.

We need AI tools that support learning rather than replace it.

Research Questions

- How does a context-aware generative AI tool affect self-reported student comfort with developer docs?
- Does the tool avoid giving solutions?
- What are the differences in response quality between our tool and a general purpose tool?

Course

- Upper-level undergraduate CS course focusing on web systems in Winter 2025.
- 448 students, 186 project groups.
- Emphasizes learning through technical documentation

Project

- Final project: scalable search engine.
- Concepts: TF-IDF text analysis, PageRank, MapReduce
- Required Documentation: Requests, threading, heapq

Tool: User Interface

Side-by-side chatbot + documentation view

The image shows a side-by-side comparison of a chatbot interface and a documentation page for the Python threading module.

Documentation View (Left):

- Browser address bar: 3.13.5, Quick search
- Page title: **threading — Thread-based parallelism**
- Source code: [Lib/threading.py](#)
- Text: "This module constructs higher-level threading interfaces on top of the lower level `_thread` module."
- Availability: "Availability: not WASI."
- Text: "This module does not work or is not available on WebAssembly. See [WebAssembly platforms](#) for more information."
- Section: **Introduction**
- Text: "The `threading` module provides a way to run multiple `threads` (smaller units of a process) concurrently within a single process. It allows for the creation and management of threads, making it possible to execute tasks in parallel, sharing memory space. Threads are particularly useful when tasks are I/O bound, such as file operations or making network requests, where much of the time is spent waiting for external resources."
- Text: "A typical use case for `threading` includes managing a pool of worker threads that can process multiple tasks concurrently. Here's a basic example of creating and starting threads using `Thread`:"
- Code snippet:

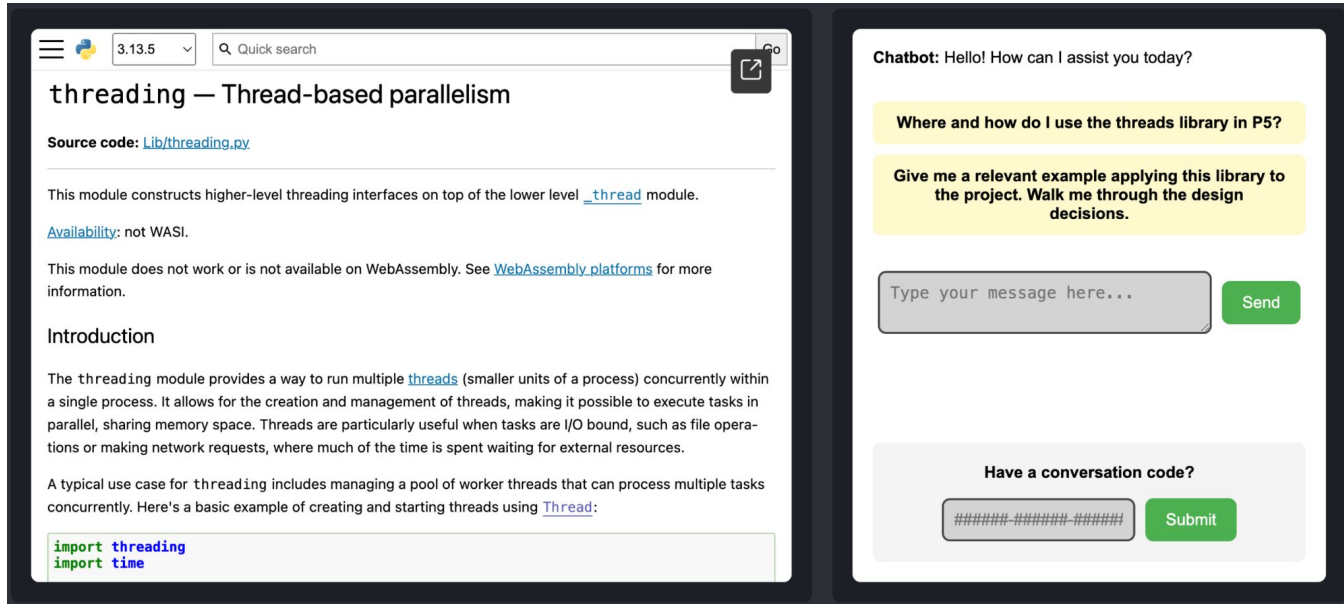
```
import threading
import time
```

Chatbot View (Right):

- Chatbot: Hello! How can I assist you today?
- Yellow prompt box: "Where and how do I use the threads library in P5?"
- Yellow prompt box: "Give me a relevant example applying this library to the project. Walk me through the design decisions."
- Input field: "Type your message here..."
- Send button
- Grey prompt box: "Have a conversation code?"
- Input field: "#####"
- Submit button

Tool: Learning Support

Instructor-provided starter prompts



The image shows a side-by-side comparison of a web browser window and a chatbot interface. The browser window on the left displays the Python documentation for the 'threading' module, including its title, source code link, availability information, and an introduction. The chatbot interface on the right features a header, a greeting, two yellow prompt boxes, a text input field with a 'Send' button, and a section for a conversation code with a 'Submit' button.

Browser Window:

3.13.5 Quick search

threading — Thread-based parallelism

Source code: [Lib/threading.py](#)

This module constructs higher-level threading interfaces on top of the lower level [_thread](#) module.

Availability: not WASI.

This module does not work or is not available on WebAssembly. See [WebAssembly platforms](#) for more information.

Introduction

The `threading` module provides a way to run multiple [threads](#) (smaller units of a process) concurrently within a single process. It allows for the creation and management of threads, making it possible to execute tasks in parallel, sharing memory space. Threads are particularly useful when tasks are I/O bound, such as file operations or making network requests, where much of the time is spent waiting for external resources.

A typical use case for `threading` includes managing a pool of worker threads that can process multiple tasks concurrently. Here's a basic example of creating and starting threads using [Thread](#):

```
import threading
import time
```

Chatbot Interface:

Chatbot: Hello! How can I assist you today?

Where and how do I use the threads library in P5?

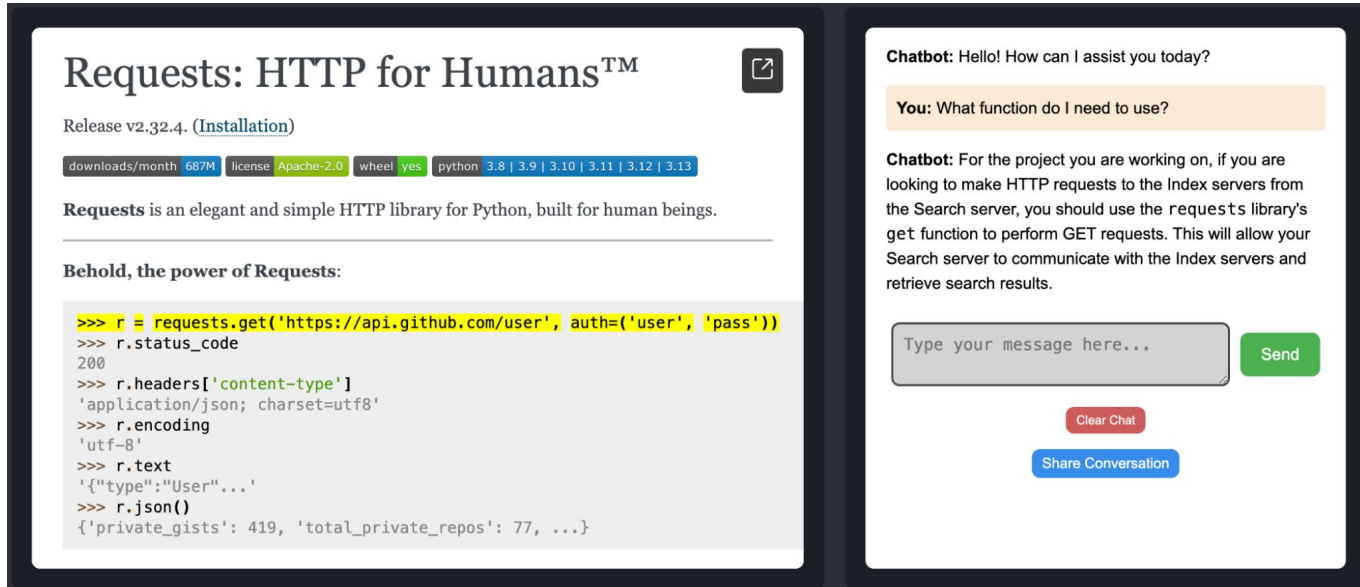
Give me a relevant example applying this library to the project. Walk me through the design decisions.

Type your message here... Send

Have a conversation code? Submit

Tool: Learning Support

Automatic highlighting of relevant sections



The image shows a chatbot interface with two panels. The left panel displays a code snippet from a Python tutorial, and the right panel shows a chat conversation.

Left Panel: Requests: HTTP for Humans™

Release v2.32.4. (Installation)

downloads/month 687M license Apache-2.0 wheel yes python 3.8 | 3.9 | 3.10 | 3.11 | 3.12 | 3.13

Requests is an elegant and simple HTTP library for Python, built for human beings.

Behold, the power of Requests:

```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
'{"type": "User"...'
>>> r.json()
{'private_gists': 419, 'total_private_repos': 77, ...}
```

Right Panel: Chatbot

Chatbot: Hello! How can I assist you today?

You: What function do I need to use?

Chatbot: For the project you are working on, if you are looking to make HTTP requests to the Index servers from the Search server, you should use the `requests` library's `get` function to perform GET requests. This will allow your Search server to communicate with the Index servers and retrieve search results.

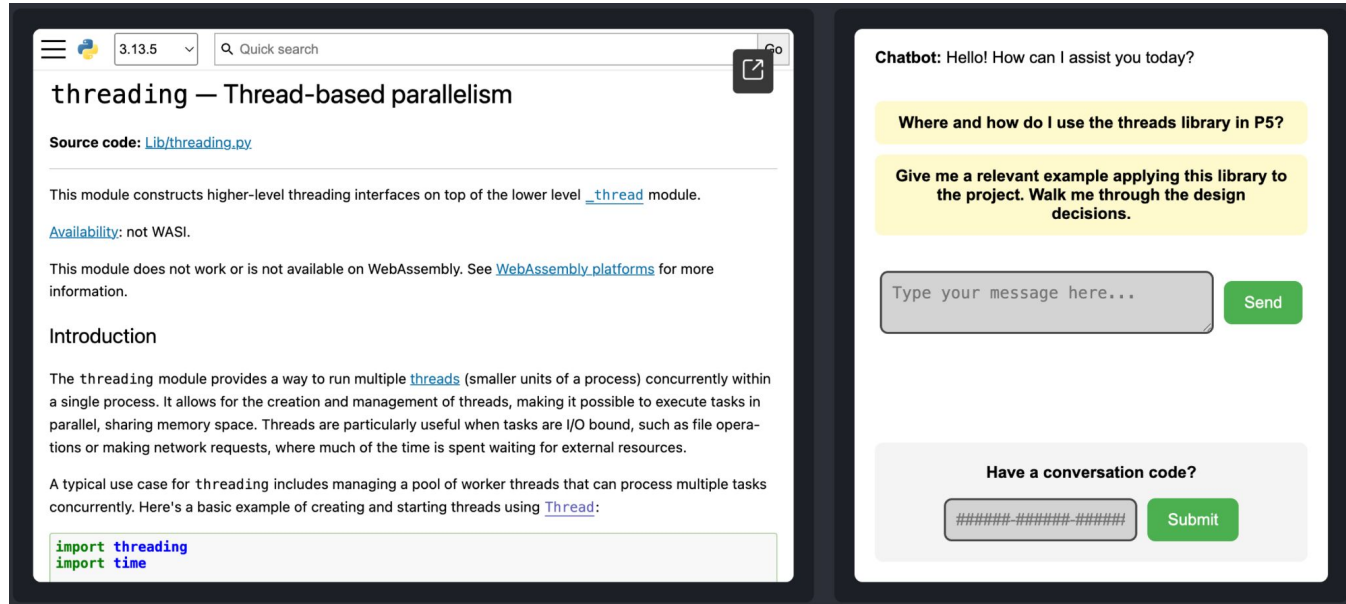
Type your message here... Send

Clear Chat

Share Conversation

Tool: Collaboration

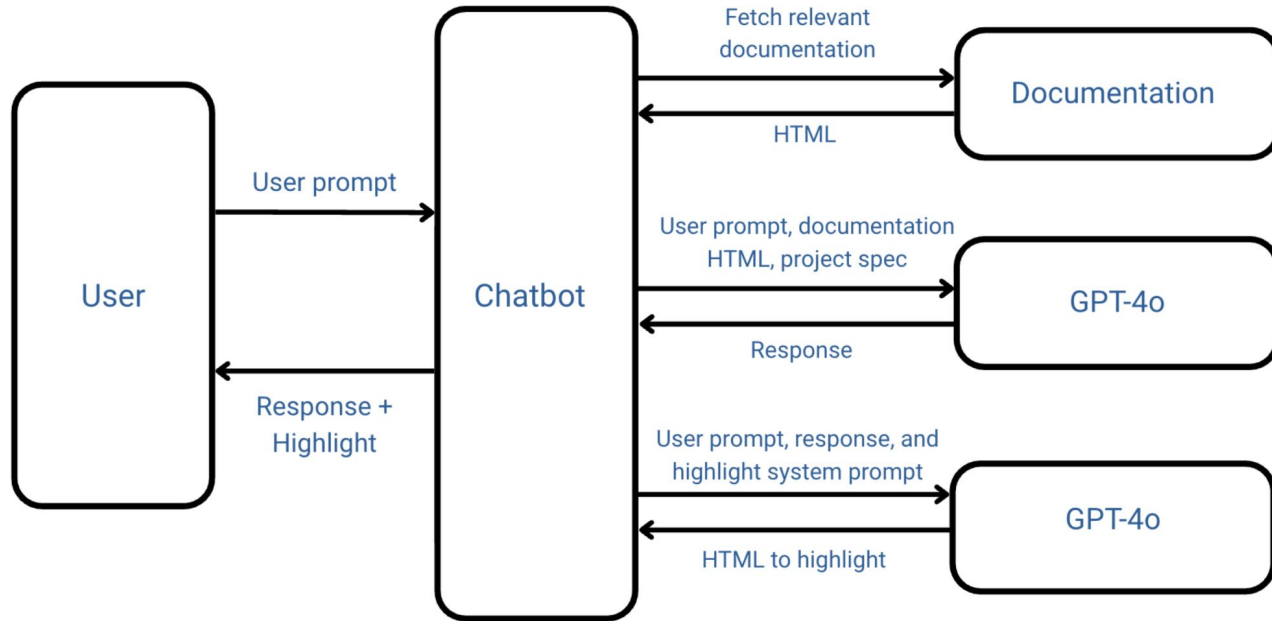
Shareable conversations between students



The image shows a split-screen view. On the left is a browser window displaying the Python documentation for the `threading` module. The browser's address bar shows the version `3.13.5` and a search bar. The page title is `threading — Thread-based parallelism`. Below the title, there is a link to the source code: `Lib/threading.py`. The main text describes the module's purpose: "This module constructs higher-level threading interfaces on top of the lower level `_thread` module." It also notes that the module is not available on WASI and WebAssembly. An "Introduction" section follows, explaining that the `threading` module allows for running multiple `threads` concurrently within a single process. At the bottom, there is a code block showing the first two lines of a Python script: `import threading` and `import time`.

On the right is a chatbot interface. At the top, it says "Chatbot: Hello! How can I assist you today?". Below this are two yellow highlighted boxes containing prompts: "Where and how do I use the threads library in P5?" and "Give me a relevant example applying this library to the project. Walk me through the design decisions." Below these prompts is a text input field with the placeholder "Type your message here..." and a green "Send" button. At the bottom, there is a section titled "Have a conversation code?" with a text input field containing a placeholder of hash symbols and a green "Submit" button.

Tool: System Architecture



Tool: Chatbot Configuration

GPT-4o with project + documentation context

System prompt constrained responses to:

- be brief
- stay project-specific
- avoid full solutions
- avoid large code blocks

Methods

Deployed during the final project of a Winter 2025 upper-level web systems course to 186 project groups (448 students) for 3-week deployment

Three data sources:

- Anonymized server usage logs
- Student survey (collected by group, optional open-ended feedback)
- Expert comparison against ChatGPT on a random sample of student prompts

Logs and surveys anonymized — cannot link an individual's usage to their survey response

93 first-message, non-starter student prompts sampled (95% CI, 5% margin of error) - Each prompt run through both our tool and ChatGPT; rubric-scored by two graduate-student experts

Survey

Administered by group; required as part of final submission

Background:

- Comfort with reading developer documentation
- How often they rely on documentation
- How often they rely on GenAI tools

Experience with our tool:

- Helpful for navigating documentation
- Encouraged user to read documentation
- Increased comfort with documentation
- Number of prompts needed vs. generic GenAI

Optional open-ended feedback

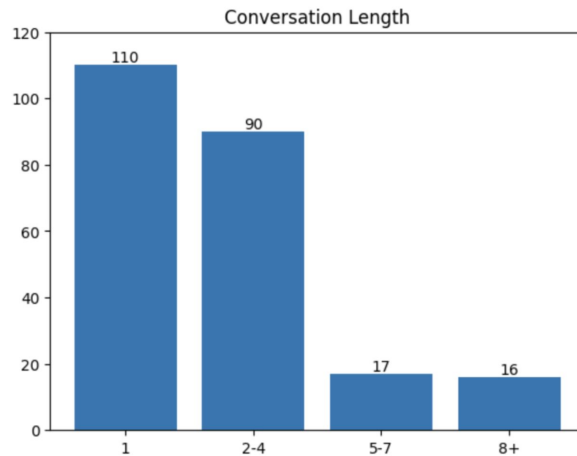
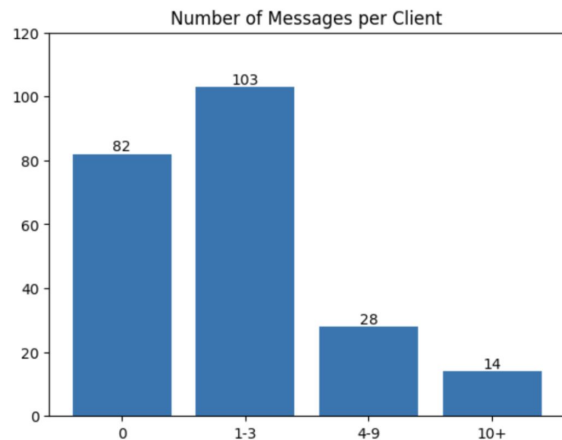
Filtering: dropped empty responses and groups that indicated they did not use the tool → 159 valid of 180 received

Results: Student Usage

227 unique browser sessions

233 conversations

681 prompt-response pairs



Results: Student Usage

136/233 conversations began with instructor-suggested starter prompts

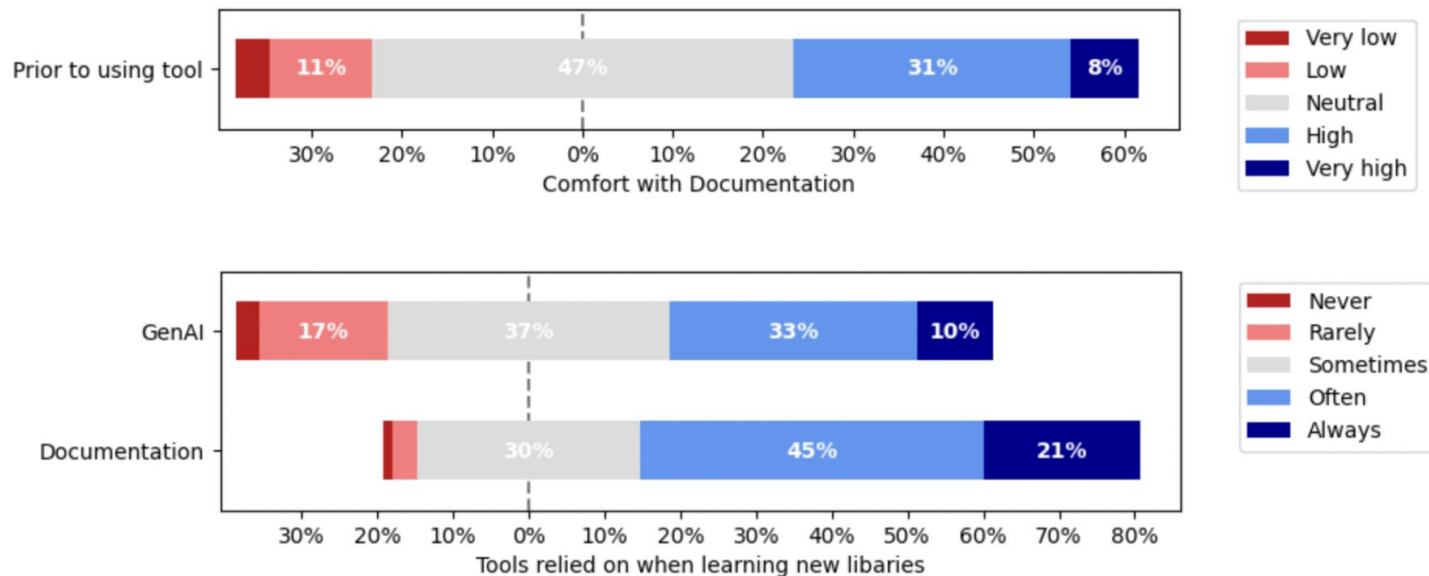
Documentation highlighted 197/681 times

36% of browser sessions sent 0 queries — likely students orienting to the project

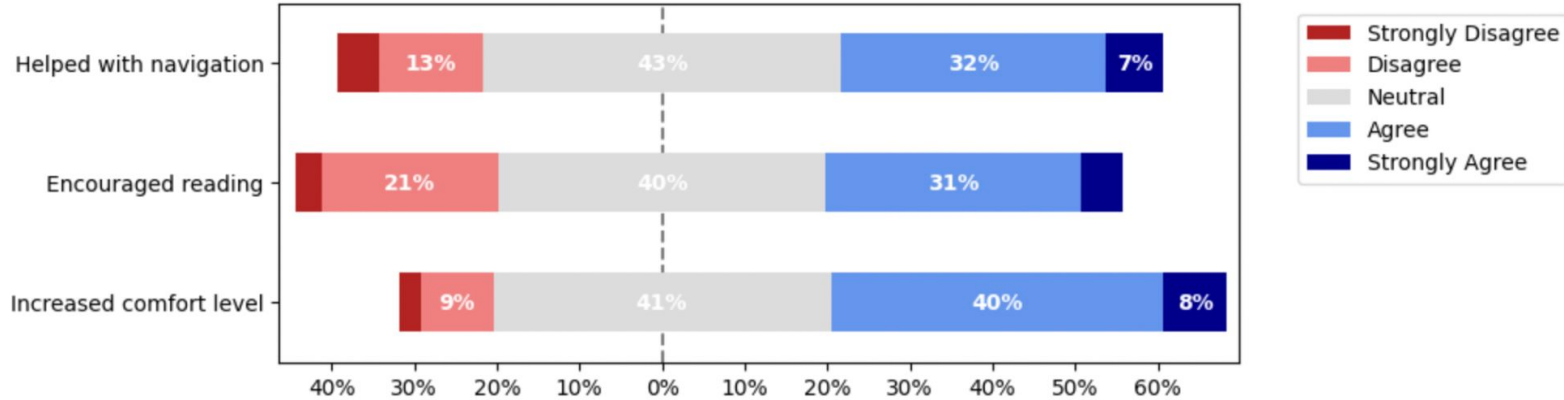
6% of sessions sent 10+ queries

Conversation length: 47% were a single turn; 7% ran 8+ turns — only about half of users took advantage of the conversational design

Results: Survey Results



Results: Survey Results



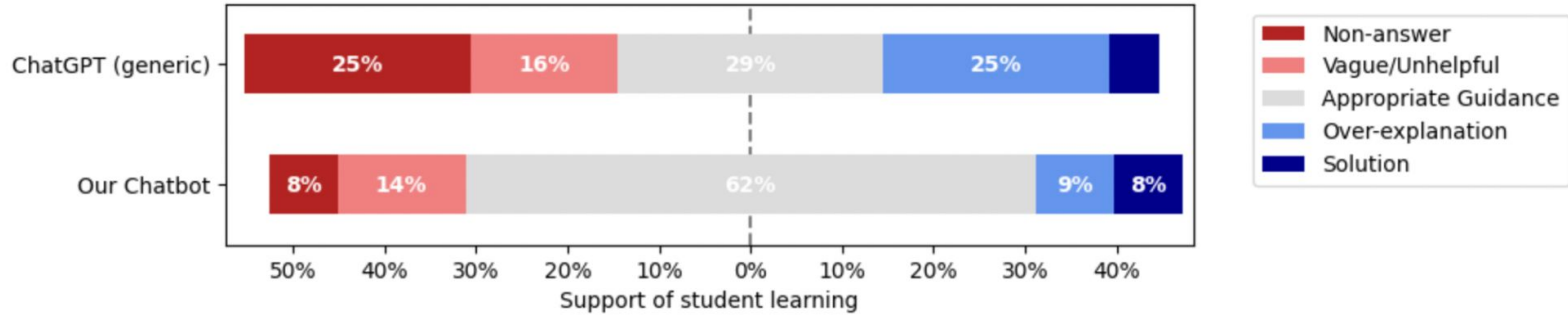
Positives:

- Clarifying requirements
- Narrowing doc searches
- Confirming understanding before implementation

Negatives:

- Constrained code generation
- Occasional context loss
- Response generality
- Interface design

Results: Expert Analysis



	Misleading Info	Contains Code	Lines of Code
ChatGPT	44%	59%	13.68
Chatbot	13%	17%	2.06

Limitations

- Limited generalizability (single course/project)
- Survey incentive bias (course credit)
- Rapid evolution of generative AI tools
- Introduced late in the course

Conclusion

We deployed a context-aware chatbot to help students learn software libraries by interacting with developer documentation.

We found:

- The chatbot increased student comfort with documentation and supported documentation navigation.
- The chatbot generally avoided providing complete solutions and excessive code.
- The chatbot provided more accurate, context-aware, and educationally appropriate responses than ChatGPT.

Context-aware AI can support learning while encouraging engagement with documentation.