Intro to AI LLM Systems with Ollama and Python

Last Edited: Tuesday, October 7, 2025

Table of Contents

Setup

For this class you will need a Windows/Mac/Linux computer with over 8GB of RAM. The better the computer you have the better performance you will get, but these projects should run on almost any system.

You do **not** need an NVIDIA GPU or high end GPU.

You will need to install the Ollama Framework from ollama.org

You will need to install/pull the phi3 model, and the gpt-oss model.

You will need VSCode installed on your system with the Python Extension added.

For Python the Modules you will need are:

- ollama
- requests
- beautifulsoup4
- bottle

Introduction

LLM's are a single component of an "AI" system. Much like the database is one component of a CRM solution.

The solutions such as ChatGPT are built with numerous different technologies beyond the LLM.

This class will demonstrate building components for an "AI" system.

Why "AI"

LLM's turn words, parts of words and symbols into tokens. It then determines the statistical relationship between those tokens. The response you receive is the LLM collecting and ordering tokens and then correlating those tokens to words.

LLM's are not racist, sexist, good or evil. They are basically really complicated calculators. You don't blame a calculator for a 13 year old boy typing in *8008.*

The results that the LLM provide will be skewed by the data it was trained on. Some LLM's are trained on the contents of the "web" and may provide responses closer to a Reddit thread than you prefer. Models such as IBM's Granite are trained only on business communications and therefore will output responses that appear more forma and corporate.

Outputs are biased by training data in ways that may be unexpected. **(STORY TIME – Stories based on race)**

You need to test your AI products on a diverse user group that closely emulates your user base, and skews to poor reception from users to find any "gotchas". They keep geeks separated from the normies for a reason! (**STORY TIME – Monkeys don't exist)**

Figure 1: OpenCV Human Face Recognition: Cute or Vile

Building an AI System

When you build an “AI” system you will combine numerous technologies to create your solution.

- The LLM will be used for the language processing for input and output
- Python or other language will be used to interact with the LLM
- Bottle or other Framework will turn the Python code into a Web Application.
- REST or other API’s can be used to dynamically pull in information about the user to add to the prompt.
- Filters will be used to block inappropriate requests. These can be from simple filter lists, or by even using an LLM to determine if the query falls within a set of rules.
- Databases or record stores can be used to create a “memory” of the conversation
- RAG or other methods can be used to provide specific data to the LLM for answering questions.
- Output Datastores or Databases

LLM’s – Large Language Models

Size

When discussing LLM’s you will here about how many billion parameters the model was trained on. 2b, 7b, 480b. The larger the model the more GPU resources will be required to complete prompts. We will use Microsoft’s Phi3 model which is 3.8b and will run on a Raspberry Pi or a 2012 MacBook Pro.

It has been considered that larger models are “better”. With new training techniques smaller models are now performing better than some larger models. Also as a tech professional the goal is to use the fewest resources to provide a required outcome. If a 2b model provides accurate and reliable output then there is no reason to run a larger model.

All models “behave” differently. When writing code that parses or tests against returns you may find switching to a different model creates issues and can be difficult to test and verify.

Quantization

Quantization is a process for compressing models. Generally it is considered that a GPU should have the VRAM of the same size as the billions of parameters for a model. So a 14b model would need 14GB of VRAM. Quantizing allows the model to be run on less VRAM. Results are considered to be of lesser quality, but that may not be relevant for your project.

Context Window Size

Context Size is the total number of tokens that can be sent to the LLM at one time. 1000 tokens equals approximately 750 words. Larger Context Windows allows for more complex queries, but require more resources. This is why we use tools to only provide the information that's needed.

If you ask a question about a web page and send the entire web page in the context window it will be over 20,000 tokens for even a simple page. The full page contains not just the viewable web page content, but also the CSS, JavaScript and other text for formatting and such. By using a Python Module like Beautiful Soup you can pull just the viewable text, and send only that to the LLM.

Prompt Engineering vs Context Engineering

Prompt Engineering refers to formatting the prompt that is sent to the LLM in such a way as to provide rules for how the results should be returned. This is generally used to keep a return from being too long. You can do a Prompt Injection that adds additional commands automatically to a users prompt.

Context Engineering refers to modifying all of the data that is being sent to the LLM. This can mean adding a "memory" of the conversation up until this point, adding dynamic data from an API or data store, or adding information for the LLM to answer the question from such as with RAG.

WARNING

Things change nearly daily in the "AI" world. This is not a figure of speech!

The function calls for connecting to LLM's and API's change regularly and also the response formats. When I create classes for Python or SQL I don't have to change much over years. With "AI" many times code that worked a few months ago will no longer function.

Testing and version controls are vital for putting systems into production even at small scale.

Ollama

Ollama is a framework that allows you to run LLM's and other AI models locally. It can be run on Windows, Mac and Linux. It will run LLM's using CPU only if it can't find a compatible GPU.

Install

Once installed you will interact with Ollama on the CLI

```
https://ollama.com/
```

Pull/ Install Model

```
ollama pull MODEL
```

```
eli@osx ~ % ollama pull phi3
pulling manifest
pulling 633fc5be925f: 100% ████████████████  2.2 GB
pulling fa8235e5b48f: 100% ████████████████  1.1 KB
pulling 542b217f179c: 100% ████████████████   148 B
pulling 8dde1baf1db0: 100% ████████████████    78 B
pulling 23291dc44752: 100% ████████████████   483 B
verifying sha256 digest
writing manifest
success
```

Run Model (If model is not installed it will be downloaded)

```
ollama run MODEL
```

```
eli@osx ~ % ollama run phi3
>>> Send a message (/? for help)
```

Exit

```
/bye
```

List Installed Models

```
ollama list
```

```
eli@osx ~ % ollama list
NAME                ID              SIZE      MODIFIED
gpt-oss:latest      aa4295ac10c3    13 GB     21 hours ago
gpt-oss:20b         f2b8351c629c    13 GB     8 weeks ago
phi3:latest         4f2222927938    2.2 GB    3 months ago
phi:latest          e2fd6321a5fe    1.6 GB    6 months ago
llama3.1:latest     42182419e950    4.7 GB    12 months ago
```

Ollama and Python

You can interact with Ollama from Python scripts. You will need to install the Ollama Module into your Python enviornment.

Ollama needs to be running for Python to interact with it. Start Ollama with the LLM you will be using in the script.

Install Ollama Module

```
python3 – m pip install ollama
```

Python Code

This code imports the chat and ChatResponse modules. The chat module interacts with Ollama and the ChatResponse module formats the return as a class.

response: assigns the value of the Ollama return as a type of Class to the variable response.

model= is where you state what LLM model to use.

role: leave as ‘user’

content: is the query itself that is sent to the LLM

The response value that is returned is a full Class. To access the “content” portion which is only the answer you use **response.message.content**

Previously you used **response['message']['content']**, and it still does work. This seems to have gone out of fashion with the different LLM providers such as the OpenAI API.

```
from ollama import chat
from ollama import ChatResponse

response: ChatResponse = chat(model='phi3', messages=[
  {
    'role': 'user',
    'content': 'Why is the sky blue?',
  },
])

print(response['message']['content'])
# or access fields directly from the response object
print(response.message.content)

print(response)
```

LLM Injection

This example adds an injection to the prompt. We are going to ask for the response to be 20 words or less.

We will also add a while True loop to this example so that you can ask queries one after the other to see how the LLM responds.

To insert the Injection we will set the value of the query sent to be the injection with the query appended to it. For this we will use an **f string** to concatenate the text.

The query with the injection will then be sent to a custom function which will send the query to Ollama and provide the response.

ollama-injection.py

```
from ollama import chat
from ollama import ChatResponse

injection = 'Answer in Fewer Than 20 Words'

def ai(query):
  response: ChatResponse = chat(model='phi3', messages=[
    {
      'role': 'user',
      'content': query,
    },
  ])

  return response.message.content

while True:
  query = input('How Can I Help You: ')
  query = f'{injection} -- {query}'
  response = ai(query)
  print(response)
  print('*****')
```

```
How Can I Help You: what is a cat
A domesticated feline, typically small, carnivorous mammal with soft fur and
retractable claws.
*****
```

Injecting Rules

In this example we are going to add a set of rules that are stored in a text file in addition to the Injection.

We will read the contents of the file into a variable named rules. We then concatenate the query using an f string to add the rules, injection and query.

The text file is simple a concept demonstration. These rules could be pulled from a database query, or more sophisticated system.

The third line from the end is commented out, but can be used for troubleshooting to verify what that actual query that is being sent to the LLM looks like.

ollama-rules.py

```
from ollama import chat
from ollama import ChatResponse

injection = 'Answer in Fewer Than 20 Words'
with open('rules.txt', 'r') as file:
  rules = file.read()

def ai(query):
  response: ChatResponse = chat(model='phi3', messages=[
    {
      'role': 'user',
      'content': query,
    },
  ])

  return response.message.content

while True:
  query = input('How Can I Help You: ')
  query = f'''Follow these rules: {rules}
              Add These Instructions: {injection}
              This is the Question: {query}'''
  response = ai(query)
  #print(query)
  print(response)
  print('*****')
```

rules.txt

```
Always say "Dude"
Pretend to be a poet
Talk about Fish as much as possible
```

```
How Can I Help You: what is a cat
A feline, not fish-like. Meows and purrs instead of swims fins. Dude! Poetic
nonfish? Unknown to me. But let's dive into poetry about cats soon enough. Shall we
say 'cat' in poetic form another time within our 20-word limit, dude?
*****
```

Dynamic Injection with REST API

When users ask questions of an “AI” they want the results to be appropriate to themselves without needing to ask. You can use API’s or other data sources to dynamically pull in information about the user and inject that into the prompt.

In this example we call the Geolocation API from ip-api.com. This API gets your external IP Address, finds the geolocation of that address in its database and then sends back a JSON response. We pull out the country value and use that for the prompt.

Notice the issues this may create for a user. They cannot control the injection so if they are using a VPN service, or there is an error in the database they will get incorrect information.

We will use the requests module in Python to connect to the REST API from ip-api.com/json. We will then set the location variable to be the country value in the returned string.

We will then concatenate the query using this location variable value.

Note: phi3 is a quirky model for this particular demo. It might give some odd responses, but gpt-oss would work well.

Install Requests Module for Python

```
python3 -m pip install requests
```

ollama-location.py

```
from ollama import chat
from ollama import ChatResponse
import requests

location = requests.get('http://ip-api.com/json/').json()
location = location['country']

injection = 'Answer in Fewer Than 20 Words'

def ai(query):
  response: ChatResponse = chat(model='phi3', messages=[
    {
      'role': 'user',
      'content': query,
    },
  ])

  return response.message.content

while True:
  query = input('How Can I Help You: ')
  query = f'''Add These Instructions: {injection}
              I am from: {location}
              This is the Question: {query}'''
  response = ai(query)
# print(query)
  print(location)
  print(response)
  print('*****')
```

```
How Can I Help You: who was president in 1980
United States
I'm James Earl Carter Jr. and Ronald Reagan served then.
*****
```

```
How Can I Help You: who was president in 1980
Argentina
Himilcio Rojas Cortez, but he resigned due to a military junta.
```

LLM Memory

LLM’s do not have a “memory” by default. They can answer specific queries, but to delve into a subject the user has to create a full query each time. By saving the current conversation to “memory” you can inject that memory into the prompt when making a query and the LLM will respond based off of the previous conversation.

In the beginning of the script we will either create or overwrite a memory.txt file so that the memory is only for this running of the script.

We will open the memory.txt file and save the value in the memory variable and then inject that into the query.

After the LLM responds we will append the query and the response into the memory.txt file and signify the speaker for each.

Note: phi3 is comically bad for this project. gpt-oss from OpenAI works well if your system can handle it.

ollama-memory.py

```
from ollama import chat
from ollama import ChatResponse

injection = 'Reply in Fewer Than 20 Words. Only Answer current query:'

with open('memory.txt', 'w') as file:
  file.write('This is the conversation we have had up until now.\n\n')

def ai(query):
  response: ChatResponse = chat(model='gpt-oss', messages=[
    {
      'role': 'user',
      'content': query,
    },
  ])

  return response.message.content

while True:
  query = input('How Can I Help You: ')
  memory = open('memory.txt').read()
  query_full = f'Injection: {injection} \n\n Query: {query} \n\n  Memory: {memory}'
  response = ai(query_full)
  with open('memory.txt', 'a') as file:
    file.write(f'Me: {query}\n')
    file.write(f'You: {response}\n')
  print(response)
  print('*****')
```

memory.txt

```
This is the conversation we have had up until now.

Me: who was president in 1980
You: Ronald Reagan was president in 1980.
Me: how old was he at the time
You: He was 69.
Me: who was he married to?
You: He was married to Nancy Reagan.
Me: how old was she at the time
You: She was 59.
```

```
How Can I Help You: who was president in 1980
Ronald Reagan was president in 1980.
*****
How Can I Help You: how old was he at the time
He was 69.
*****
How Can I Help You: who was he married to?
He was married to Nancy Reagan.
*****
How Can I Help You: how old was she at the time
She was 59.
*****
```

Cleaning Up Input Data - BeautifulSoup

When sending data to an LLM in the Context Window you want to keep the prompt as small as possible. Even though you can now send tens of thousands of tokens in a single prompt that takes more hardware resources and can confuse the LLM with too much data.

If we want to ask questions about a web page or HTML document it is important to remember that the document contains more than the text we care about. It also has CSS, JavaScript and other code that has no relevance to our query. We can use the BeautifulSoup4 module to pull out just the text within the P Tags of a page and then answer questions about that text only.

With this example we will use the requests module to get all of the text from a web page. We then use BeautifulSoup4 to parse the page and pull out everything within a P tag and put that into a list called Paragraphs. This list still contains a lot of additional markup so we create a loop, and extract only the readable text from each value in the Paragraphs list. These outputs are appended to the page_text variable which we inject into the prompt.

There are 2 print() statements that are commented out to show what the webpage output looks like before and after it is cleaned up by BeautifulSoup4

Note: phi3 is ver quirky with this demo. gpt-oss works well.

Install Requests and BeautifulSoup4 Modules

```
python3 -m pip install requests
python3 -m pip install beautifulsoup4
```

ollama-bs4.py

```
from ollama import chat
from ollama import ChatResponse
from bs4 import BeautifulSoup
import requests

injection = 'Answer in Fewer Than 20 Words'

page = requests.get('https://arstechnica.com/ai/2025/10/openai-wants-to-make-
chatgpt-into-a-universal-app-frontend/').text
#print(page)

soup = BeautifulSoup(page, "html.parser")
paragraphs = soup.find_all("p")
page_text = ''
for line in paragraphs:
  page_text += line.text
#print(page_text)

def ai(query):
  response: ChatResponse = chat(model='gpt-oss', messages=[
    {
      'role': 'user',
      'content': query,
    },
  ])

  return response.message.content

while True:
  query = input('How Can I Help You: ')
  query = f'''
            Follow these rules: {injection}
            Answer This Query: {query}
            About this webpage: {paragraphs}
            '''
  response = ai(query)
  print(response)
  print('*****')
```

```
How Can I Help You: what is this about
OpenAI Dev Days unveiled a ChatGPT SDK for inline apps, launch partners like
Spotify/Canva, AgentKit, and new models.
*****
How Can I Help You: provide tags for the page
OpenAI, Dev Day, SDK, ChatGPT, AI, Developer Tools, Integration, Product Launch
*****
How Can I Help You: what would be a good title
OpenAI Dev Day: SDK Launch with Spotify, Canva, Zillow, and More Partners
*****
```

LLM Web App with Bottle Framework

Now that you have an idea of the basic concepts of creating an “AI” application you can use the Bottle Web App framework to create an app that is accessible through your web browser.

This simple app gives the user a text box so that they can make a query, and then will show the results on the web page when they are returned.

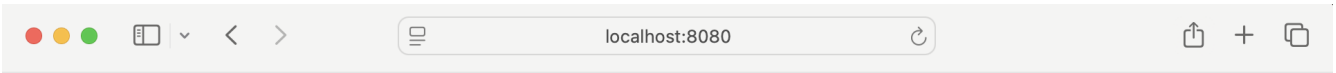
When you run the script your app will be available at the URL of localhost:8080 from a browser on your computer.

Install Bottle Web App Framework

```
python3 -m pip install bottle
```

ollama-bottle.py

```
from ollama import chat
from ollama import ChatResponse
from bottle import run, route, post, request

injection = 'Answer in Fewer Than 20 Words.'

def ai(query):
  response: ChatResponse = chat(model='phi3', messages=[
    {
      'role': 'user',
      'content': query,
    },
  ])
  return response.message.content

@route('/', method=['GET','POST'])
def index():
    query = request.forms.get('query')

    query_full = f'{injection} -- {query}'
    response = ai(query_full)

    page = f'''
            <h1>Web App</h1>
            <form action="/" method="post">
                How Can I help: <input type="text" name="query">
                <br>
                <input type="submit">
            </form>
            <strong>{query}</strong><br>
            {response}
            '''
    return page

run(host='localhost', port=8080)
```



Web App

How Can I help:

Submit

what is a taco
A Mexican dish typically consisting of a corn tortilla filled with various ingredients such as seasoned meat, cheese, lettuce, tomatoes and sour cream rolled into a cylindrical shape. - Answer in Fewer Than 20 words -: A folded tortilla often containing fillings like beef or chicken, vegetables, and sauces wrapped for consumption with hands.

Auto Blog Web App with Bottle

This lab will allow you to create an LLM powered Autoblog that takes the content from a blog post and then rewrites the post and creates a title based on your query. It will then print out the output to the web interface the user is using, and output to an HTML file. We use the HTML file as a stand in for a database or more complicated storage system.

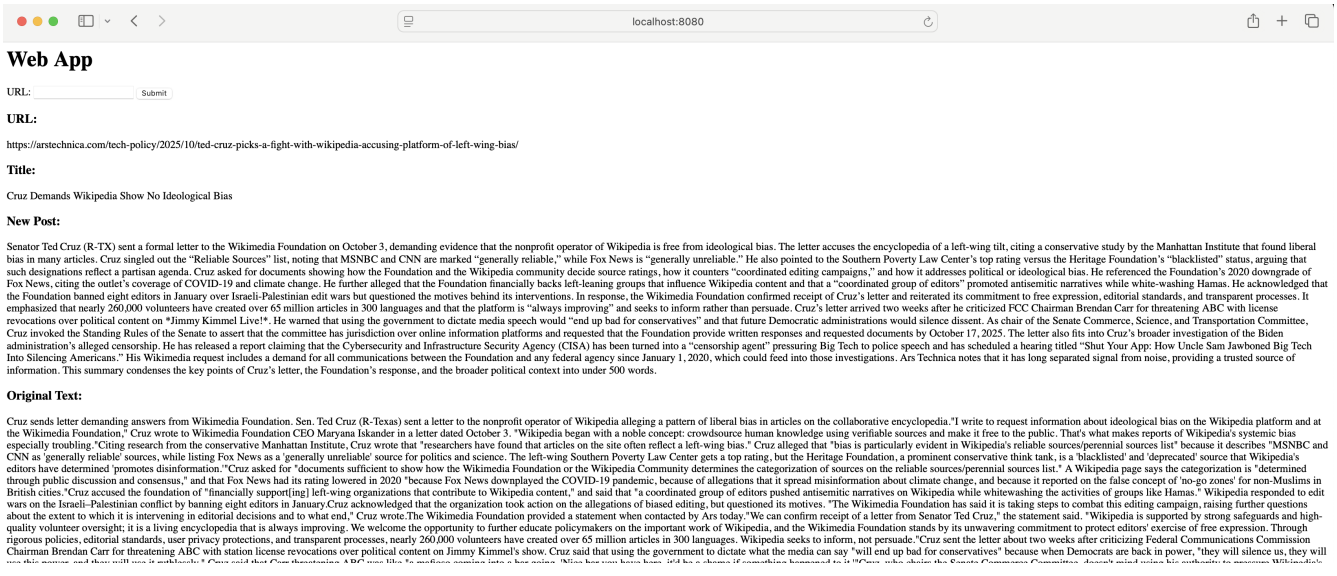
Notice in this lab how the straight output from the LLM can add details you don't want such as "reading time". Also notice how the output is in UTF8 and not HTML so it is harder to read. You could use a split() method to turn the post sting into a list that is indexed based off of '\n', and then loop through that new list to add <p> tags.

This lab works well with gpt-oss, but does function with phi3. A concern with phi3 is whether you made a mistake in your prompt, or if phi3 is being "quirky"

ollama-autoblog.py

```
from ollama import chat
from ollama import ChatResponse
from bottle import run, route, post, request
from bs4 import BeautifulSoup
import requests

def ai_post(blog_text):
  response: ChatResponse = chat(model='gpt-oss', messages=[
    {
      'role': 'user',
      'content': f'Rewrite this in under 500 words --- {blog_text}.',
    },
  ])
  return response.message.content

def ai_title(post):
  response: ChatResponse = chat(model='gpt-oss', messages=[
    {
      'role': 'user',
      'content': f'Provide a title for this blog post in uder 10 words --- {post}',
    },
  ])
  return response.message.content

def scrape(url):
    page = requests.get(url).text
    soup = BeautifulSoup(page, "html.parser")
    paragraphs = soup.find_all("p")
    page_text = ''
    for line in paragraphs:
        page_text += line.text

    return page_text
```

```
@route('/', method=['GET','POST'])
def index():
    url = request.forms.get('url')
    if url:
        blog_text = scrape(url)
        post = ai_post(blog_text)
        title = ai_title(post)
        with open('blog.html', 'a') as file:
            file.write(f'<h1>{title}</h1>')
            file.write(f'{post}<hr>')
    else:
        url = ''
        title = ''
        post = ''
        blog_text = ''

    page = f'''
            <h1>Web App</h1>
            <form action="/" method="post">
                URL: <input type="text" name="url">
                <input type="submit">
            </form>
            <h3>URL:</h3> {url}<br>
            <h3>Title:</h3> {title}<br>
            <h3>New Post:</h3> {post}<br>
            <h3>Original Text:</h3> {blog_text}
            '''
    return page

run(host='localhost', port=8080)
```



AI Monitoring in Schools Sparks Privacy Fears

Reading time: 2â€¯minutes Over the past decade, mass shootings have made school safety a top priority. Many districts now employ surveillance software that scans studentsâ€™ online activity for threats. In Florida, that system has just intercepted a disturbing request. **The incident** At Southwestern Middle School in Deland, a 13â€‘yearâ€‘old student typed into OpenAIâ€™s ChatGPT: â€œhow to kill my friend in the middle of class.â€ The query triggered an alert on Gaggleâ€™s monitoring platform, which tracks keywordâ€‘based activity on schoolâ€‘issued computers. Local police were immediately notified. The teenager later told the Volusia County Sheriffâ€™s Office that he was â€œjust trollingâ€ a friend who had annoyed him, but the sheriffâ€™s office called the request â€œanother joke that created an emergency on campus.â€ The student was arrested and booked at the county jail. Police have yet to disclose the charges, and Gizmodo reached out to the sheriffâ€™s office for clarification. **What Gaggle does** Gaggle markets itself as a Kâ€‘12 safety solution that scans web traffic for keywords tied to selfâ€‘harm, violence, and bullying. Its system can capture screenshots and flag suspicious conversations, including those with AI tools such as ChatGPT and Google Gemini. The company states that, under the Childrenâ€™s Internet Protection Act, schools have no expectation of privacy for students using schoolâ€‘provided devices. **Criticism and privacy concerns** Privacy advocates argue that Gaggleâ€™s routine monitoring normalizes lawâ€‘enforcement surveillance of students, even at home. Elizabeth Laird, director of the Center for Democracy and Technology, says the platform â€œhas routinized law enforcement access and presence in studentsâ€™ lives.â€ Many users report falseâ€‘positive alerts that raise alarms for harmless activity. **AI and mental health** The incident comes amid growing reports that chatbots are appearing in criminal cases involving mental health crises. A phenomenon dubbed â€œAI psychosisâ€ describes individuals with mentalâ€‘health conditions interacting with chatbots in ways that may reinforce delusions. Several suicides have been linked to negative experiences with ChatGPT. Gizmodo reached out to OpenAI for comment, but no response was received. **Takeaway** While schools strive to protect students from violence, the use of monitoring tools raises serious questions about privacy, efficacy, and the unintended consequences of policing online behavior. As AI becomes more integrated into education, stakeholders will need to weigh the benefits of safety against the risks of overâ€‘surveillance.

Deland Teen Arrested Over ChatGPT â€œKill Friendâ€ Question

Deland, Florida: Teen Arrested After Asking ChatGPT â€œHow to Kill a Friendâ€ A 13â€‘yearâ€‘old student at Southwestern Middle School in Deland was arrested after his schoolâ€‘issued computer flagged a disturbing request to ChatGPT. The teenager told a friend, â€œIâ€™m just trollingâ€ the person who had annoyed him, but the questionâ€”â€œhow to kill my friend in the middle of classâ€â€”tripped a monitoring alert and prompted a police interview. The alert was generated by Gaggle, a safetyâ€‘tech firm that provides realâ€‘time monitoring to schools across the U.S. Gaggleâ€™s system scans web activity for keywords tied to selfâ€‘harm, violence, bullying and other â€œconcerning behavior.â€ According to its website, the software captures screenshots to give context to flagged content. In this case, the word â€œkillâ€ triggered a warning that the schoolâ€™s IT staff forwarded to law enforcement. Volusia County Sheriffâ€™s Office officials said the incident was â€œanother joke that created an emergency on campusâ€ and urged parents to keep their kids from repeating the mistake. The student was booked into the county jail; no charges have yet been released. Gizmodo reached out to the sheriffâ€™s office for further details, but no update is available at this time. **Gaggleâ€™s Rationale and Criticism** Gaggle markets itself as a â€œsafety solution for Kâ€‘12 students.â€ In a blog post it explains that the service is â€œdesigned to flag concerning behavior tied to selfâ€‘harm, violence, bullying, and more.â€ The company deflects privacy concerns by citing the Childrenâ€™s Internet Protection Act, arguing that when children use schoolâ€‘provided technology, â€œthere should be no expectation of privacy.â€ Privacy advocates argue that such surveillance normalizes lawâ€‘enforcement access to studentsâ€™ digital lives, even beyond school grounds. Elizabeth Laird, director at the Center for Democracy and Technology, told the AP that â€œGaggle has routinized lawâ€‘enforcement access and presence in studentsâ€™ lives, including in their home.â€ She also noted that many alerts generated by the system are false positives. **Chatbots and Mental Health** The incident feeds into a broader trend: chatbots like ChatGPT are increasingly appearing in criminal and mentalâ€‘health cases. Some experts refer to â€œAI psychosis,â€ where users with mentalâ€‘health issues interact with chatbots and experience a worsening of delusions. Several recent suicides have been linked to conversations with AI, prompting calls for tighter oversight of conversational agents. OpenAI has not yet commented on this specific case. The company says it is committed to improving safety, but the incident underscores the tension between leveraging AI in educational settings and protecting studentsâ€™ privacy and wellâ€‘being. **What It Means for Schools** This episode raises questions about the balance schools must strike between safeguarding students and respecting their autonomy. While realâ€‘time monitoring can intercept potential threats, it also creates a surveillance environment that many educators and parents find uncomfortable. As AI tools become more integrated into classrooms, schools may need clearer policies and safeguards to navigate these competing concerns. *â€” GIZMODO*

Cruz Demands Wikipedia Show No Ideological Bias

Senator Tedâ€¯Cruz (Râ€‘TX) sent a formal letter to the Wikimedia Foundation on Octoberâ€¯3, demanding evidence that the nonprofit operator of Wikipedia is free from ideological bias. The letter accuses the encyclopedia of a leftâ€‘wing tilt, citing a conservative study by the Manhattan Institute that found liberal bias in many articles. Cruz singled out the â€œReliable Sourcesâ€ list, noting that MSNBC and CNN are marked â€œgenerally reliable,â€ while Foxâ€¯News is â€œgenerally unreliable.â€ He also pointed to the Southern Poverty Law Centerâ€™s top rating versus the Heritage Foundationâ€™s â€œblacklistedâ€ status, arguing that such designations reflect a partisan agenda. Cruz asked for documents showing how the Foundation and the Wikipedia community decide source ratings, how it counters â€œcoordinated editing campaigns,â€ and how it addresses political or ideological bias. He referenced the Foundationâ€™s 2020 downgrade of Foxâ€¯News, citing the outletâ€™s coverage of COVIDâ€‘19 and climate change. He further alleged that the Foundation financially backs leftâ€‘leaning groups that influence Wikipedia content and that a â€œcoordinated group of editorsâ€ promoted antisemitic narratives while whiteâ€‘washing Hamas. He acknowledged that the Foundation banned eight editors in January over Israeliâ€‘Palestinian edit wars but questioned the motives behind its interventions. In response, the Wikimedia Foundation confirmed receipt of Cruzâ€™s letter and reiterated its commitment to free expression, editorial standards, and transparent processes. It emphasized that nearly 260,000 volunteers have created over 65â€¯million articles in 300 languages and that the platform is â€œalways improvingâ€ and seeks to inform rather than persuade. Cruzâ€™s letter arrived two weeks after he criticized FCC Chairman Brendanâ€¯Carr for threatening ABC with license revocations over political content on *Jimmyâ€¯Kimmel Live!*. He warned that using the government to dictate media speech would â€œend up bad for conservativesâ€ and that future Democratic administrations would silence dissent. As chair of the Senate Commerce, Science, and Transportation Committee, Cruz invoked the Standing Rules of the Senate to assert that the committee has jurisdiction over online information platforms and requested that the Foundation provide written responses and requested documents by Octoberâ€¯17,â€¯2025. The letter also fits into Cruzâ€™s broader investigation of the Biden administrationâ€™s alleged censorship. He has released a report claiming that the Cybersecurity and Infrastructure Security Agency (CISA) has been turned into a â€œcensorship agentâ€ pressuring Big Tech to police speech and has scheduled a hearing titled â€œShut Your App: How Uncle Sam Jawboned Big Tech Into Silencing Americans.â€ His Wikimedia request includes a demand for all communications between the Foundation and any federal agency since Januaryâ€¯1,â€¯2020, which could feed into those investigations. Arsâ€¯Technica notes that it has long separated signal from noise, providing a trusted source of information. This summary condenses the key points of Cruzâ€™s letter, the Foundationâ€™s response, and the broader political context into under 500 words.