CGTA: An OpenAI-Powered Virtual Course Assistant

MONIQUE LEGASPI
University of Pennsylvania

Teaching assistants have an enormous load to handle. In addition to holding office hours and grading assignments, they must also answer questions at all hours of the day on Piazza and Ed, providing quality feedback to posts made by students of all levels of expertise. In an effort to alleviate the amount of effort needed from TAs during this particular task, and having taken inspiration from previous researchers who created similar solutions to this problem, I sought to create CGTA, an OpenAI-powered chat-bot that takes archival information about a course (specifically, hundreds of old Piazza and Ed posts for said course) and uses that specialized, scope-appropriate knowledge to provide detailed answers to any questions posed by students.

CGTA is a Python-based, locally-run program that utilizes sentence encoding and the OpenAI API to generate answers to CG@Penn questions. Specifically, when given the course, assignment tag, and question from a student's Piazza/Ed post, CGTA first accesses an archive of Piazza/Ed posts from previous semesters and uses sentence encoding to find the five most similar posts to the current question. Then, through an OpenAI model with customized settings and system prompt, CGTA uses these most similar posts to generate an answer to the student's question, which is thorough while remaining within the scope of what the student should be learning in the course. Finally, CGTA provides a "confidence score" for its answer based on the question's similarity to other posts in its archive, along with suggestions to the human TA running the program for whether the answer should be modified or rewritten.

Since CGTA's archive consists mainly of specific questions about assignments and not the assignment code/specs itself, CGTA performs poorly when asked questions that do not have any similar questions that appear in its archive. CGTA does, however, perform better with generic C++ questions, perhaps due to the fact that OpenAI has access to the broader Internet and can more easily answer with information that can be found in, say, C++ documentation. User testing suggests that CGTA is also hindered by vagueness in how students word their questions; a more specific question will usually produce a more specific answer.

Additional Keywords: Chatbot, Education, Question Answering, Information Retrieval, Embeddings, OpenAI, ChatGPT

1. INTRODUCTION

Being a teaching assistant for a Computer Graphics course requires an enormous amount of effort. In an endeavor to give one's students the greatest chance of success, one must hold multiple office hours per week, carefully grade assignments, and stay ever-vigilant in online forums such as Piazza to deftly answer any technical or conceptual questions asked at odd hours. A human eye will always be necessary to debug code in person or fairly judge the amount of points awarded for an assignment, but given the current state of technology, some improvements can be made to the Piazza environment to lighten the load of TAs. For example, in CIS 4600: Interactive Computer Graphics, there are hundreds of questions asked in Piazza every semester, many of which are common questions that appear across multiple semesters, or are even asked by different students in the same semester.

Rather than require TAs to answer these questions every time – and relying on them to answer quickly at all hours of the day – it would be easier to pull from the existing archive of knowledge and have some kind of bot answer them instead. Additionally, with the advent of ChatGPT, it is now much easier to create a chatbot that can interpret questions asked in natural human language and give a custom, coherent answer. Through independent study, the goal of this project was to create a virtual assistant using the OpenAI API, trained on past archives of CIS 4600/5600 (Interactive Computer Graphics) and CIS 4620/5620 (Computer Animation) Piazza questions, which would easily, effectively, and seamlessly be able to answer questions asked by students in these courses in future semesters.

2. RELATED WORK

There has already been some interest in the area of utilizing chatbots in academia, especially so with the rise of ChatGPT, mostly for the purposes of aiding teaching assistants. In terms of speculation, many scholars have ruminated on the place that chatbots have in academic spaces, such as higher education. In particular, scholars at the University of Rochester note that ChatGPT is now a mainstay of education that has forced instructors to evolve their teaching practices, for better or for worse [Man23], with reporters at Forbes saying much of the same [Rav23].

In terms of implementation, there are already a few iterations of chatbots designed expressly for the purpose of being teaching assistants. Percy, a chatbot created for an artificial intelligence class at Stanford University, most closely mirrors what I am trying to achieve, being a chatbot that operates through Piazza and is trained on Piazza data [CGS16]. However, this chatbot performs poorly in many cases, and it was created in 2016, and as such does not utilize ChatGPT API. Other examples of chatbots in academia include one created for Virginia Polytechnic University [AK20], and ATOB, which was created for University of Seattle [CYC21]. My independent study will draw inspiration from all of these chatbots in conjunction with ChatGPT API.

There is also much critical reception on the use of AI for teaching assistance. Stanford's Percy chatbot was inspired by a TA bot created for an artificial intelligence course at Georgia Tech [Mad16]. Scholars reflect on the impact of using AI to assist in Ghanaian higher education [EVT22]. And, of course, students take to Reddit to complain of ChatGPT being used to answer their

Piazza questions, indicating that, even when armed with ChatGPT's seemingly endless knowledge, they still come to TAs for more accurate and specific answers [Bre23].

Given the tendency for today's students to take to ChatGPT for answers on assignments of all subjects, ChatGPT's very existence makes the question of virtual TAs all the more relevant. While ChatGPT is an excellent resource, it has been trained on data that is far beyond the scope of anything that CG@Penn students would need to understand for their courses, and is especially known to give out incorrect information with an astounding level of confidence. It would be much wiser to craft a course-specific ChatGPT-like bot that can give answers within the scope of the course, and that can be reasonably trusted to be correct, given that it has been trained on questions that were answered by instructors of the course.

3. OVERVIEW OF APPROACH

3.1. Design

3.1.1. Design Goals

The following goals were kept in mind while designing and developing CGTA:

Target Audience: TAs and instructors in CG@Penn courses, as they will find the most use in a program that generates answers to Piazza/Ed questions, built on an archive of posts specific to their course(s).

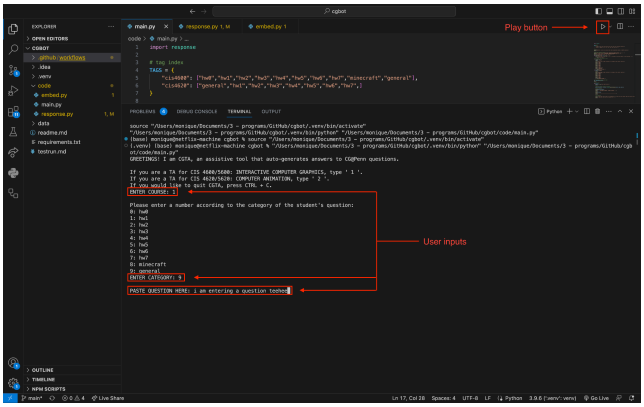
User Goals & Objectives: The user will be able to quickly generate an answer to a given question specific to their course, provided there is a post in that course's archive that is similar enough to supply enough information to craft a relevant and helpful answer. If the answer is insufficient, the user should be told so by the program, and then the user should be freely able to modify and rewrite the answer as needed. The user should not need to use anything more than the program (for generating answers) and Piazza/Ed (to retrieve and answer the post), and possibly the assignment specs on the course website if the program's answer is insufficient.

Features & Functionality: The user will be able to specify the class for which they are TAing, as well as the category of the post to which they are attending. The user will also be able to generate answers to multiple posts in a single session, provided all posts fall under the same class and category.

Input & Output: The user inputs the course for which they are TAing, and the category and question body of the post they are seeking to answer. The output they receive is a generated answer to the question, a confidence measure based on how similar the question was to previously archived posts, and a suggestion on whether to modify or rewrite the generated answer.

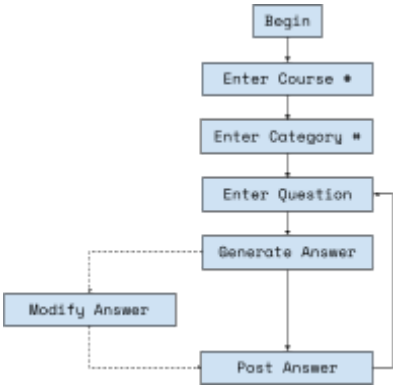
3.1.2 User Interface

CGTA runs locally on the user's computer in the VS Code Terminal. As such, there is not much by way of GUI. The program is run by pressing the Play button in the top-right corner of VS Code when main.py is open, and all inputs on the user's part are entered in the Terminal.



After running the program, the user must enter the number corresponding to the course for which they are TAing. Then, they must enter the category of the post they are answering. Finally, they must paste in the body of the question from the post and hit Enter in order to receive a generated answer. From there, they may modify or rewrite the answer as they see fit, and then upload it as a response to the Piazza/Ed post.

The workflow for the user is as follows:



3.2. Data Collection

In order to build the archive from which CGTA can draw its knowledge, two courses in particular were of interest – CIS 4600/5600 (Interactive Computer Graphics) and CIS 4620/5620 (Computer Animation) – due to the overlap of requirements for both undergraduate and masters students in the CG@Penn program, and thus high enrollment rate. This meant that these courses would need the most assistance, and additionally would have the most data to work with.

Posts were sorted and transcribed by hand into .txt files to be stored locally with CGTA; while this could have been done with

some web-scraping, it was easier to throw out unresolved questions (which would be useless for a question-answering tool) this way, and images/videos could be transcribed as they were encountered. These .txt files were then sorted into a file tree organized by class > category > semester. The formatting of each post within the files had to be very specific, so that they could be split into separate items during encoding. Here is an example of how a question would be formatted:

```
###question @413
Post title
Paste the body of the student's question here.
IMAGE START
If the student has an image or video, describe it here.
You don't have to be super descriptive, especially if the
student has already described what's going on in the picture.
IMAGE END
#posttag
Paste the instructor/other student's answer here.
If there is a follow-up question, paste it below the answer.
If there is an answer to the follow-up question, paste it here.
Etc...
```

Some other features that were kept in mind:

- The most crucial part of this formatting was making sure the post began with `###question`, because this is how the file reader splits posts into separate entries.
- If a Piazza/Ed post was unresolved, it was not logged. If its main question was resolved, but had an unresolved follow-up, the follow-up was not logged.
- If the post contained a hyperlink (e.g., if someone typed "Click here", and clicking the words led to a StackOverflow post), the link was copied and pasted underneath the paragraph in which it appeared.

This procedure also serves as a guideline for any TAs or instructors seeking to expand the archive, which was documented in the README under CGTA's repository.

3.3. Development

CGTA uses fairly simple functions in order to complete its tasks; most of the heavy lifting is done by the libraries and APIs. The main features are as follows:

Embeddings: Upon receiving a question and its corresponding course and category, CGTA reads all files under the path course/category/, then splits the files into a list with each post being a separate item. Each post is then further split into a two-item list of (a) main question and (b) answer/follow-ups, so that the encoder does not have to process so many words. The student's question and all other posts in the list are encoded using the SentenceTransformer library (https://www.sbert.net/docs/quickstart.html), which encodes by measuring distance in vector space through cosine similarity, and each post is assigned a score based on how similar it is to the student's question. The posts are sorted by similarity score, and the top five most similar posts (questions, answers, and follow-ups included) are sent along to the response generator.

Response generation: After retrieving the five most similar posts, CGTA uses the OpenAI API (https://platform.openai.com/docs/introduction) to generate an answer to the student's question. Extensive testing in GPT Playground allowed me to tweak the model's settings to be more appropriate for CGTA's user base. These settings, as well as a TA-specific system prompt, were pasted into the program and accessed by the API in order to generate a response.

Confidence score: The confidence score is simply the similarity score of the archived post that is the most similar to the student's question.

4. DOCUMENTATION/TUTORIAL

CGTA's GitHub repository is freely available (though the average non-CG@Penn person may not find much use for it) at https://github.com/falseaxiom/cgbot. In order to use CGTA, you will need Visual Studio Code and an OpenAI account with credits on it. Then, follow these steps:

1. Download code from repository and open in VS Code.
2. In VS Code, start a virtual environment.
    a. Go to the Command Palette and type ">Python: Create Environment...".
    b. Select "Venv".
    c. Select whatever Python installation you have on your computer (I'm using Python 3.9.6 64-bit, so this version or similarly recent will probably work best).
    d. When prompted for dependencies to install, select `requirements.txt`.
3. Open `response.py` and paste an OpenAI API key from your account as the value of `openai.api_key`. If you don't have one and don't know how to get one, do the following:
    a. Log into your OpenAI account in your browser.
    b. Click on your profile in the top-right corner and select "View API Key" from the drop-down menu that appears.
    c. Select "+ Create new secret key" and (optionally) give it a relevant name.
    d. Copy the secret key and save it somewhere.
    e. Paste it as the value of `openai.api_key`, as mentioned above.
4. Open `main.py` and run the Python script.
5. When prompted `ENTER COURSE:`, type the number corresponding to the course whose question(s) you would like to answer and press `Enter`.
6. When prompted `ENTER CATEGORY:`, type the number corresponding to the category/tag the post falls under in Piazza/Ed and press `Enter`.
7. In the terminal, when prompted `PASTE QUESTION HERE:`, paste the student's question and press `Enter`.
8. The generated answer will print directly to the terminal. Paste this into Piazza/Ed and edit/rewrite as you see fit.
9. To generate further answers for questions, repeat steps 7-8. To quit, simply press `Ctrl+C`.

5. USER TESTING

In order to test the efficacy of CGTA, I asked 3 colleagues to download and use the program, two of whom were TAs for CIS 5600: Interactive Computer Graphics, and the third being well-acquainted with embeddings and OpenAI in general. I provided each tester with general instructions on how to use CGTA, as well as the directive to test some assignment-related questions on it, preferably pulling questions from their own Piazza archives if they had access (both TAs had served as TAs much longer than me, and as such had posts that CGTA had never seen).

In the process of testing CGTA, users found that its answers tend to be vague, often to the point of not sufficiently answering their questions. For instance, when asked the question, "what steps do i take to ensure my puppet head pivots at the neck?" for Homework 2, CGTA gave a technically correct, but more conceptual, answer that does not address what the student is actually looking for, which the tester noted is probably "the sequence of transforms required here. … Of course, it's on the student to be very detailed, but sometimes it's also on the TA to decipher their question."

In scenarios where CGTA was completely correct, testers noted that the confidence score was unexpectedly low; this is likely due to the fact that although there was an archived post that was very similar in concept to the question asked, it contained so much other information (and thus, many more words) that it resulted in a misleadingly low similarity score.

CGTA notably struggled with answering questions specific to the structure of an assignment, which is understandable, since CGTA does not have access to the solution code, and thus cannot extrapolate and provide information about an assignment that was not already archived from posts in previous semesters. However, CGTA did well with C++ questions, since these are less specific and have a broader range of resources available in the public Internet, which OpenAI can access in order to provide more detailed and accurate answers.

In general, testers were pleased with CGTA's ability to pull archival information and use it to provide coherent and seemingly-relevant answers to their questions. They noticed, however, that asking CGTA vague questions caused it to provide vague answers. Since students cannot be expected to optimize the wording of their question to benefit CGTA's algorithm, this remains a problem, although understandably so.

6. DISCUSSION

6.1. Accomplishments

Throughout the process of working on CGTA, much of the original intended functionality of this virtual assistant has been achieved. CGTA is able to:

- Receive, and access the archive of, what specific course the user is working for.
- Receive, and access the archive of, the category of the post the user is seeking to answer.
- Retrieve archived posts that are the most similar to the question from this post.
- Use this retrieved archival information in order to generate a coherent answer that is within the scope of the course.
- Alert the user to the accuracy of the generated answer via confidence scoring.
- Provide suggestions to the user based on this score, indicating whether the generated answer should be posted as-is, modified, or rewritten.
- Condense all of the above functionality into a single program, so as to streamline the workflow for the user.

6.2. Limitations

While CGTA achieves much of what it set out to do, it is not without fault. Here are some limitations that, due to the short time frame of this project, could not be sufficiently remedied:

Unintuitive confidence scoring: The confidence score is just the similarity score of the archive post that is most similar to the student's question. Due to the nature of embeddings, the similarity score is calculated via similarity in wording, so a question with very similar concept, but worded differently, could potentially have a low similarity score, resulting in a low confidence score, even if the generated answer fits the question very well. This is especially problematic when it comes to CIS 4620 (Computer Animation) posts, as many archived posts included complex mathematical equations that could be written out differently (for instance, with curly braces instead of square brackets) but still have the same end result.

Convoluted setup: The fact that it can only be run in VS Code with a virtual environment, and that input/output takes place entirely in the terminal, is less than ideal. The original conceit was for CGTA to be a web-app or, even better, integrated directly into Piazza/Ed, so that it could answer students' questions immediately as they come in, but a lack of understanding of how to integrate OpenAI API into a website (much less a Piazza/Ed extension...!) designated this luxury a time-sink.

No follow-ups: Unfortunately, the most recent version of GPT is incapable of handling follow-up questions in custom models--or at least, not reliably, and not without a very complex setup. Due to the nature of how CGTA can be used in its current state--namely, human TAs manually entering questions, vetting/editing answers, and posting them to Piazza as they are available--a situation in which CGTA would need to immediately answer a follow-up question in a single session is highly unlikely, as there is an extremely low chance that the TA will see the first question within seconds/minutes of it being posted, answer it, have the student see the answer, and then have the student add to the thread within a few more seconds/minutes. Therefore, the feature is unnecessary for now.

6.3. Future Work

There are endless future avenues for improvement upon CGTA, including:

- Improved similarity/confidence scoring - possibly the most important next step, since it is crucial for TAs to be confident in the correctness of the answer they are posting for their students.
    - Since CGTA relies mostly on syntactic similarity, one possible route would be to take advantage of ChatGPT's ability to summarize texts, condensing them into their base concepts. If one uses this feature to summarize both

the student's question and the questions in the archives, perhaps it will result in summaries that are very similar grammatically, thus giving a higher similarity score with the same scoring system.
    - Another route could be to find ways to score questions based on semantic similarity (aka similarity in meaning rather than wording), which would be more proper for CGTA's purposes.
- A more efficient way to support images in posts/questions/answers. With GPT-4, it is now possible to input and interpret images, so incorporating this feature in the future (when it is no longer so new and expensive) could be beneficial to improving question entry/scoring.
- The ability to ask follow-up questions, building on context provided from previous questions within the same session. As mentioned in the previous section, OpenAI has limited ability to do so at the time of writing, so progressing on this point will likely require waiting for an update to their system.
- A more user-friendly GUI, rather than typing everything into the VS Code Terminal.
- A web-app version, so that the program is easily accessible online, archives can be updated automatically across all users, and the user can remain in their browser for the whole workflow (copy question from Piazza/Ed post → switch tabs → paste into CGTA → copy generated answer → switch tabs → paste as answer to Piazza/Ed post).
    - Further, integration with Piazza/Ed would make this process even easier.
- Full or partial automation of CGTA's functionality, so that it can answer students' questions in real-time, or at least so that a human TA logging in can have a backlog of answers waiting for them to review and post.
    - An email account has already been set up, so that CGTA can potentially set up a Piazza/Ed account for the class, receive email alerts, scrape the question from there, and post the generated answer on its own.
    - Expanding this automation into having a self-maintaining archive database, where answering a question on Piazza automatically adds the post to the archive (or at least having a more high-level method for TAs to enter posts rather than editing the .txt file directly), would also be ideal.

7. CONCLUSION

Through the power of embeddings and OpenAI, CGTA is able to retrieve relevant information from a vast archive of CG@Penn Piazza/Ed posts and use that information to generate answers to questions posed by future semesters of students in CG@Penn courses. The quality of the answers depends largely on the specificity of the question, whether similar questions have been asked in the past, and how much it relies on intimate knowledge of the assignment specs. CGTA performs poorly with vague and unheard-of questions that inquire about assignment-specific things, but performs markedly better in response to carefully-worded, and particularly C++-related, questions. Future work in improving performance with assignment-related questions, ranking the confidence of its answers in general, and interpreting questions to uncover what students are truly seeking help with, is necessary to build a better-functioning CGTA. With these features improved, the hope is that CGTA can someday be partially- or fully-automated, answering Piazza posts in real-time so that students can receive help during any hour of the day.

8. ACKNOWLEDGMENTS

Thank you to my advisor and program director, Dr. Stephen H. Lane, for his guidance and feedback, as well as to my fellow CG@Penn classmates, who were gracious enough to lend their time and expertise to testing CGTA (and to Lennart Meincke in particular, who gave me the idea to utilize embeddings and provided me with many resources on how to do so).

REFERENCES

[AK20] Abdelhamid, S., & Katz, A. (2020, July). Using Chatbots as Smart Teaching Assistants for First-Year Engineering Students. In 2020 First-Year Engineering Experience.

[Bre23] bren0xa (2023). ChatGPT spotted on EECS 280 Piazza. Reddit.

[CGS16] Chopra, S., Gianforte, R., & Sholar, J. (2016). Meet percy: The CS 221 teaching assistant chatbot. ACM Transactions on Graphics, 1(1), 1-8.

[Che22] Chee, Z. X. (2022). Chatbot as a teaching assistant.

[CYC21] Chuaphan, A., Yoon, H. J., & Chung, S. (2021). A TA-Like Chatbot Application: ATOB. In Proceedings of the EDSIG Conference ISSN (Vol. 2473, p. 4901).

[EVT22] Essel, H.B., Vlachopoulos, D., Tachie-Menson, A. et al (2022). The impact of a virtual teaching assistant (chatbot) on students' learning in Ghanaian higher education.

[Mad16] Maderer, J. (2016). Artificial intelligence course creates AI teaching assistant. Georgia Tech News Center, 9.

[Man23] Mandelaro, J. (2023). How will AI chatbots like ChatGPT affect higher education?.

[Rav23] Ravaglia, R. (2023). ChatGPT Will Not Replace Teachers But Could Act As A Teaching Assistant.