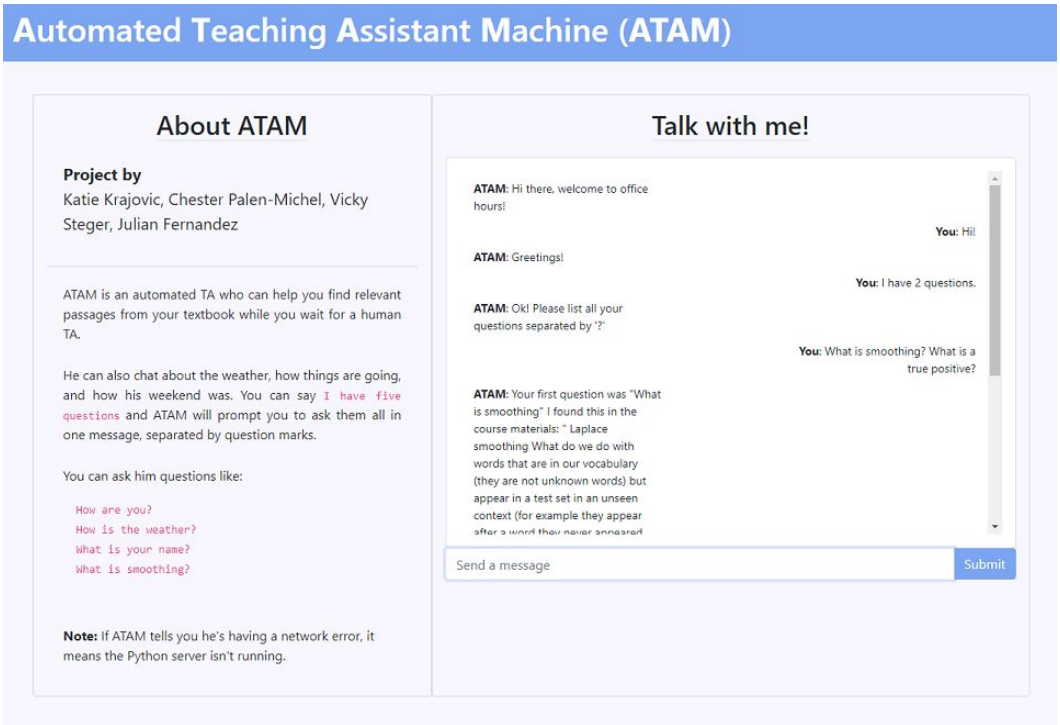
Group Final Project
COSI 233

Authors:
Julian Fernandez
Katie Krajovic
Chester Palen-Michel
Vicky Steger

1. Overview

ATAM (Automated Teaching Assistant Machine) is a chatbot that functions as a virtual TA. Students can ask him questions and receive answers taken from a textbook. He is accessible through a web interface that runs on a user's local machine.

It is a common format of virtual office hours in the time of Covid that students spend substantial time in a TA's Zoom waiting room, hoping for an opportunity to talk to the TA who is attempting to meet with students individually. The idea of ATAM is that he could be deployed into these waiting rooms to interact with students, answer common questions, review course content, and possibly even present and review activities.



Example of ATAM in Use

ATAM is in the early stages of development, and not all of this functionality is supported yet.

In ATAM's current state, he is able to handle basic greetings and small talk from students, search for answers to students' course-related questions in a textbook, address multiple questions asked at once or questions with pronouns that lack referents, and recognize questions on grades or assignments, as well as requests for the TA. When ATAM recognizes that a user has asked to end the session, he provides a small overview of the conversation for the human TA to review to ensure that any unanswered questions can be addressed.

In this paper we focus on ATAM's dialog system, and include a section on future work where we discuss current shortcomings and hopes for future development.

2. Implementation

In this section, we describe the details of our implementation. We start with an overview of the web services and overall structure of ATAM. We then describe the overall dialogue flow. Next we explain ATAM's question answering system. Finally, we cover the details of ATAM's question under discussion tracking.

2.1 Architecture Details

ATAM is implemented as two web services, referred to here as the Python service and React service according to the language each was written with. The React service provides a UI designed with Bootstrap that gives students a chat interface where they can send and receive messages from the Python service.

When a student sends a message through the UI, which runs on port 3000, their message becomes a POST request to port 5000, where the Python service runs. The Python service is implemented as a Flask app that processes the text contained in the user's message and in most cases, passes it along to ATAM's Wit.AI app for intention detection. A JSON file is used to return hard-coded messages from ATAM about all topics that aren't questions about the textbook.

2.2 Dialogue Flow

This section presents an overview of the dialog flow. This is also represented in a figure included at the end of this document (See Appendix A). The following discussion will elaborate on the information presented in the figure.

ATAM initiates dialog with a hard coded initial turn - “Hi there, welcome to office hours”. The system then waits for user input. When it receives input, it is sent to Wit.ai which classifies it into one of 12 intents, or “*no intent*”, and identifies relevant slots/entities.

4 of the intents are designed around small talk: *greeting, personal, time_related, and weather*. *Greetings* encompass things like “hi”, “hello”, “howdy”. ATAM responds with a random greeting selected from a set of hard coded potential responses. *Personal* covers utterances like “how are you?”, “how’s it going”, etc. Again ATAM provides a randomly selected response of the form “I’m well/tired/great. How are you?/How can I help?”. *Time_related* represents utterances that refer to a specific period, for example “how was your weekend?” or “did you enjoy winter break?”. A response will provide a description of a time period and enquire about a user’s time period (e.g. “it was good. How was your weekend?”). Finally, *weather* includes all utterances referencing the weather. ATAM responds with a generic comment about the weather, and asks how he can help the user.

There is also a hard coded response for when Wit does not classify the utterance into any of our specified intents. We treat this as a fallback intent, and respond with a helpful message about what kinds of questions ATAM is prepared to respond to.

Two additional intents are used to categorize utterances that relate to grades and assignments. *Grades* covers anything relating to a student’s grade on an assignment, exam, or overall. It can also relate to the grading of an assignment, like the rubric used or the points allocated. *Assignment* includes any utterance that relates to a course assignment. This could be asking for help, clarification, or a due date. When either of these intents are recognized, ATAM explains to the student that only the TA can discuss grades and assignments, and offers to answer questions about course content while the student waits for the TA to become available.

One of the most critical intents for ATAM’s current functionality is *question*. The *question* intent covers all utterances relating to course content (excluding grades and assignments for the reasons stated above). These utterances might take forms like “What does the viterbi algorithm do?”, “I’m confused about what an HMM is used for.”, or “Could you explain the average perceptron to me?”.

When ATAM recognizes a question, a query is constructed based on the input utterance and then compared against the course textbook (or any relevant course materials) to find a section of content that might help to answer the user’s question. ATAM then presents the result of the query to the user, and asks if the content was helpful. If a user

answers with yes, then ATAM moves on. If the user answers with no, then ATAM will present the second best result of the query. After three unsuccessful attempts at answering a question, ATAM gives up and asks for a new question.

As mentioned above, *yes* and *no* are possible user inputs, and therefore each has its own intent. Many variations of negative and affirmative utterances are handled (e.g. yep!, yeppers, kinda, not really, etc).

ATAM can also handle multiple questions. The *multi_question* intent recognizes utterances like “I have several questions” or “I have 17 questions today”. When a *multi_question* intent is recognized, ATAM prompts the user to input each of their questions separated by ‘?’. ATAM then stores these questions in a list of pending questions, and begins by answering the first question the user listed. Any time ATAM completes a question-answering routine (regardless of whether it resulted in success or failure) and the list of pending questions is not empty, he asks the user if they would like to discuss the next question on the list. The user can respond yes or no, and ATAM will proceed accordingly. If the user says yes, the next question is immediately addressed. If the user says no, ATAM waits for the user to bring up a new topic, and only after answering a new, different question will he prompt again whether the user wants to talk about the next question on the pending questions list.

This is done to allow the user to ask follow up questions about the question that was just answered. For example if the user had asked two questions, like “what is Viterbi? What are some applications of Viterbi?”, ATAM would begin by answering “what is Viterbi”. The answer to the first question might lead to a follow up question from the user. In this case once the question had been answered ATAM would ask if the user wanted to discuss their next question, relating to the applications of viterbi. The user could say no, and instead ask a follow up question. After the follow up question has been answered, ATAM will prompt again, asking whether the user is ready to move on to the next topic. This process can continue until the user is finished asking follow up questions and decides they are ready to move on to the next question.

The final critical intent is *want_ta*. This encompasses utterances that signal that the user wants to speak with the TA and not ATAM. These range from things like “I would like to talk with the TA now”, in which the user explicitly requests to work with the TA, to “you are horrible at answering questions!” where the user expresses frustration suggesting that ATAM is not successfully answering the user’s questions and they are losing patience. In both of these cases, ATAM responds that the TA will be with the student as soon as possible, and offers to continue to attempt to answer questions in the meantime.

The final intent is *exit*, which signals that the user is finished interacting with ATAM. ATAM says goodbye to the user and produces a log of the conversation. Ultimately this log could be sent to the TA as a summary of what ATAM was able to handle for the student and what may still be unclear.

2.3 Question Answering

Course content questions are answered by ATAM by considering query-sentence similarity using an index of sentence embeddings. The textbook is converted from pdf to plain text and preprocessed to remove sentences that are just the list of references and single word sentences before indexing. Sentences are indexed using Annoy (Spotify's approximate nearest neighbor library). For the sentence embedding, we use HuggingFace's off the shelf sentence transformers. When querying a sentence, we let Wit.AI identify the search term in a question intent, and pass that to sentence transformers as if the query terms are their own sentence.

We found that the best matching sentences alone were typically not informative enough. Instead, we take a window of sentences around the best matching sentence for additional context. We had also attempted to use HuggingFace's off-the-shelf question answering system which was built for the SQuAD dataset; however, we found it did not provide good results for our purposes. The responses tended to be too short and not informative enough to be helpful.

One example from using SQuAD for question answering:

> User: What is a hidden markov model?
> ATAM: generative

While this is technically a correct answer to the question. It isn't actually that helpful to a student who wants to know a more complete definition of a hidden markov model. HuggingFace's transformers allow us to avoid such minimalist responses.

2.4 Question Management

As mentioned in an earlier section, ATAM can recognize when a user asks more than one question at a time. In addition to identifying the *multi_question* intent, ATAM also splits all user input on '?'. If a user inputs an utterance like "what is an HMM? What is Viterbi?", ATAM will recognize two questions, add them to the pending questions list and begin answering the first one.

Both question answering and multi-question management involve requesting input from the user, either to determine if the provided content was helpful or to determine if the user would like to move on to the next question. This is handled with a series of states. ATAM can be in a qa_follow_up state or a pending_follow_up state. These signify that he has requested specific input from a user, and *yes* and *no* intents are processed in specific ways depending on the presence or absence of these state flags.

ATAM also has states for *multi_question,* representing when he has prompted the user to input their list of questions, and *first_of_multi*, when he provides slightly different responses for the first question as opposed to the rest.

2.5 Question Under Discussion

ATAM tracks the question under discussion (QUD) by taking advantage of the entity tagging inherent in Wit.Ai responses, as well as the intents that were created to categorize user input. When a user submits an utterance, it is first sent to Wit.Ai for parsing. As long as its intent is recognized as something other than a yes or no response, it is considered a relevant update to the QUD. For most intents, the QUD will be updated to be the utterance currently being considered, with the topic being stored as the name of that intent. For example, the utterance “Hello” would be stored as the QUD, and “greeting” would be stored as the topic. For utterances classified as questions, the full question will again be stored as the QUD, but the piece of the question recognized by Wit.Ai as its main entity will be stored as the topic. For example, “What is the efficiency of bubble sort?” would have the main entity “efficiency of bubble sort”.

Whenever a new utterance is submitted, if it is a question, its topic will be compared to the existing topic to look for overlap. If any words are found in common between the current topic and previous topic, this is taken to be the overall topic of conversation. For example, if the question following “What is the efficiency of bubble sort?” is “Why use bubble sort?”, the topics “efficiency of bubble sort” and “bubble sort” would be compared, and the overall topic would be judged to be “bubble sort”. If the new utterance is any other non-yes-or-no intent, its intent will become the new topic with no overlap comparison.

This topic tracking allows ATAM to perform very basic anaphora resolution. When a user’s utterance is judged to have a question intent or cannot be properly recognized by Wit.Ai, if it contains a non-gendered pronoun, ATAM will add the current topic to the end of the utterance and submit it to Wit.Ai for another judgment. This changes an

ambiguous sentence like “What is it” to something like “What is it Viterbi”, which testing showed had both higher chances of recognition in Wit.Ai, as well as higher chances of returning successful results from the question answering code. Only a question or unknown intent are considered for anaphora resolution to avoid utterances such as “Hello” followed by “How’s it going?” being altered to “How’s it going? Greeting” and harming the Wit.Ai analysis of the intent.

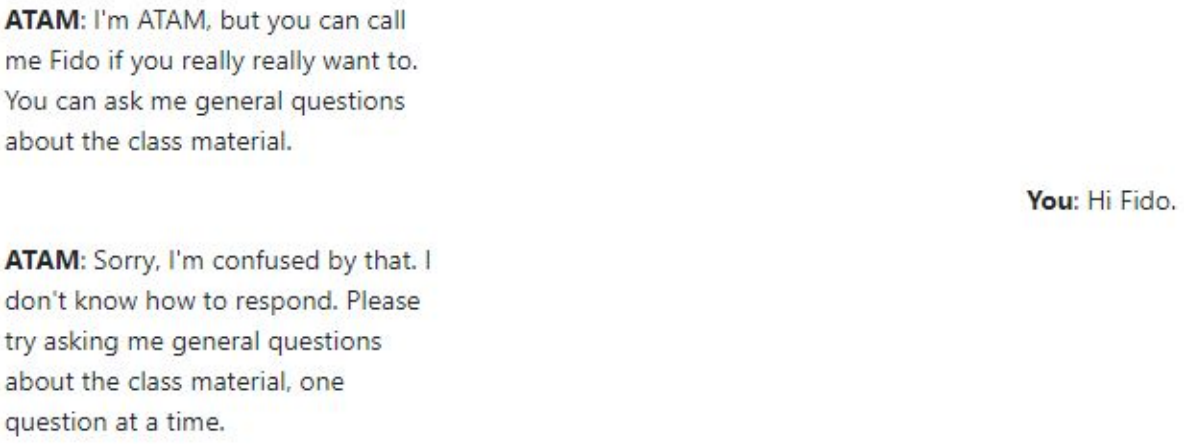
2.6 Logging

ATAM also has a rudimentary logging system to provide an overview of what was discussed with a user and which questions may still be unanswered. This log file is created when ATAM recognizes a user’s intent to exit the program. It is saved in the same location as the main qa_web_app.py file with the name “conversation_summary.txt”, and contains a list of all non-yes-or-no questions submitted by the user, grouped by intent. For question intents, the log also stores whether or not ATAM believes the question was successfully answered, based on the user’s confirmation or denial of an answer’s usefulness. If the user does not respond when asked if an answer was helpful, the question will be given no label, to allow the reviewer to know it may still require attention.

3. Future Work

ATAM is not yet ready for deployment. When the ATAM Python service runs locally, there is no notion of a session or of the possibility of two different simultaneous users. ATAM operates as if all messages are from a single student. A database would need to be introduced along with user authentication before ATAM could be functionally hosted on a site and handle concurrent users.

3.1 Responses to Default Responses



ATAM isn't perfect. He can't handle user references to his own diction. He says we may call him Fido, but that doesn't mean he'll understand it when we do. While it's possible to change all default dialogue to avoid prompting responses that ATAM can't handle, the ideal solution would be to teach ATAM to work with these responses, perhaps through the addition of another intent.

3.2 Assignments Question Answering

We would have liked to set up more complete functionality for answering student requests about grades and questions about assignments. For assignment specific questions, in easy cases we could potentially take a similar approach to textbook content questions with indexing assignment instructions and trying to point students to a relevant excerpt. However this alone is likely to still be somewhat poor performing. What would be ideal is to train a model specifically for this type of domain specific question answering, or potentially do some sort of domain adaptation or transfer.

3.3 Grades

We would like for ATAM to be able to handle student queries about grades. To do this we'd likely need to connect to a database which stores those grades. We may also need to handle the issue of confidentiality for ATAM to handle grades. While students can often just view their grades on an online platform like LATTE, it could be advantageous to discuss with a chatbot that can actually do reasoning regarding grades. For example, "If I get a 80% on PS 4, what do I need to get on the final to pass the course." This would require additional dialogue state tracking so ATAM prompts users for guesses about assignments that haven't been completed yet in order to come up with the hypothetical grades.

3.4 Common answer lookup

In the future we would like to provide the option for the TA to specify a list of anticipated common questions and answers that ATAM could store and reference when answering student questions. There are frequently questions asked over and over in office hours, maybe about a particularly tricky piece of an assignment, and TAs can often anticipate what they might be even before their office hours start. Giving ATAM the ability to answer these quick and common questions in the waiting room, before the student even makes it to the TA, could drastically reduce the wait-time for students with simple questions, and also maximize the efficiency of office hours by filling the TA's time with the most challenging questions.

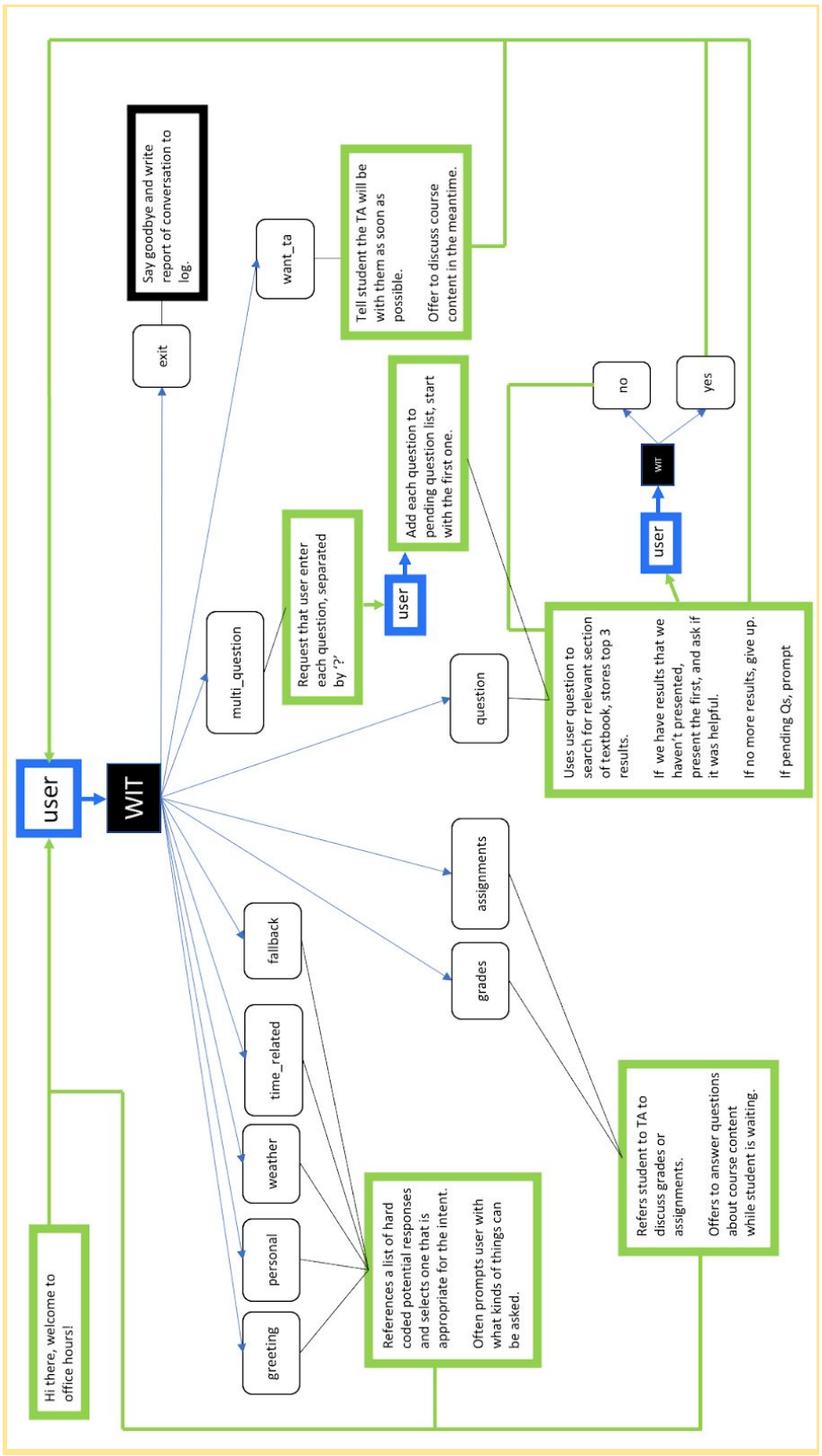
3.5 Activities while waiting

No matter how successful ATAM ultimately becomes, it is likely that students will still eventually exhaust their questions and be left sitting in the waiting room, waiting for their turn to talk to the TA. During this time we would like for ATAM to be able to present an activity relevant to what the students have been working on in class. The student could work on the activity and notify ATAM when they had finished an attempt at a solution. ATAM could then walk the student through an example solution, ideally including some visualization of the problem. ATAM could perhaps also analyze the student's solution to provide feedback, but substantially more work would be necessary to make this happen.

4. Conclusion

Overall ATAM is a reasonable proof of concept that demonstrates that a TA chatbot can be created in a fairly short amount of time. We show that using a combination of Wit.AI, Annoy, Flask, React and some hand crafted dialogue state and question-under-discussion tracking, a fairly strong proto-type can quickly be developed. There remain a number of interesting and exciting directions that ATAM could be extended and refined in the future. It is our hope that ATAM and future iterations of ATAM could greatly help to reduce wait times for students in office hours and the workload of teaching assistants, ultimately providing a better experience for everyone involved in a course.

Appendix A: Dialogue flow of ATAM


user
WIT
Hi there, welcome to office hours!
greeting
personal
weather
time_related
fallback
References a list of hard coded potential responses and selects one that is appropriate for the intent.
Often prompts user with what kinds of things can be asked.
grades
assignments
Refers student to TA to discuss grades or assignments.
Offers to answer questions about course content while student is waiting.
question
Uses user question to search for relevant section of textbook, stores top 3 results.
If we have results that we haven't presented, present the first, and ask if it was helpful.
If no more results, give up.
If pending Qs, prompt
user
WIT
no
yes
multi_question
Request that user enter each question, separated by '?'
user
Add each question to pending question list, start with the first one.
want_ta
Tell student the TA will be with them as soon as possible.
Offer to discuss course content in the meantime.
exit
Say goodbye and write report of conversation to log.