# CSC200: Chatbot Implementation Project Report

ADIAN GOLDFARB (AGOLDFA7), PAUL OUELLETTE (POUELLET), RAFAELLO SANNA (RSANNA), and YIYAO YU (YYU57)

## 1 INTRODUCTION

Conversational chatbots have been a subject of interest for many years. While it is not difficult to create a chatbot that is capable of generating phrasing and sentences. It is much harder to have the program initiate human-like conversation as well as provide accurate answers to questions. In order for the conversation to sound natural to a human communicating with the chatbot, it must be able to utilize a dialogue model trained on a large enough sample of human conversation so that it can extrapolate data when none is available. Meanwhile, for responses to be accurate, the chatbot would have to lookup questions and responses within some form of data store, which would provide accurate but generic-sounding answers for anything within, while also having difficulty in answering questions not included within the database. The effectiveness of these two different types of chatbots are on two ends of the same spectrum, as one prioritizes conversation realism at the cost of information accuracy, while the other guarantees accuracy at the cost of conversation uniqueness. For our CSC200/200H chatbot assignment, we've decided to implement a Computer Science advising chatbot using both methods, where one uses a Question and Answer model with database query and the other uses a fine-tuned Conversation model, both provided by the "rust-bert" crate in Rust (`https://github.com/guillaume-be/rust-bert`). We compare the implementation difficulty as well as maintenance difficulty between the two different chatbots to see what makes each approach stand-out.

## 2 CONTRIBUTIONS

This chatbot project is implemented by the following team members, listed in alphabetical order, with specific details on contributions as follows:

- **Adian Goldfarb (agoldfa7):** Trained two NLP models based on Microsoft's GPT2 NL model. Attempted to finetune DialoGPT model.
- **Paul Ouellette (pouellet):** Worked on parsing CDCS query results and custom model loading, created default context file.
- **Rafaello Sanna (rsanna):** Implemented a keyword-based fuzzy parser for course names to gather context, work on database back-end
- **Yiyao Yu (yyu57):** Performed initial CDCS database parsing and manual data filtering, implemented main chatbot framework and Q&A chatbot database initial load/store and lookup

## 3 PROJECT SPECIFICATIONS AND ASSUMPTIONS

The overarching idea behind the assignment is very open-ended. The only thing that is required is to implement a chatbot that is capable of fulfilling the role of advising students in the Computer Science department through a command-line interface. Because of this, we identified two problems that we would like to analyze and attempt to solve for this assignment:

(1) to what extent could the chatbot recognize and respond to natural language input?
(2) how factual are the statements outputted by the chatbot?

Problem (1) is a reasonable question to ask: since what we are implementing is a chatbot, there should be some amount of focus on the "chat" aspect of the bot. It needs to be able to receive and

respond to the user's questions as in a natural language such as English. Having the user converse with the bot using a limited set of grammatical constructs or a "fill in the blanks" input is too limiting for conversation. At minimum, the chatbot needs to be able to identify natural language queries and respond accordingly, regardless of whether the response is mechanical or not.

For (2), the response of the chatbot needs to be as factual as possible. Since this is an advising chatbot, it needs to provide information that is accurate enough for the user to act upon. We recognize that this is difficult for an actual human advisor to achieve, as even we may make mistakes in conveying information, therefore, we are only requiring that the chatbot provides information at roughly the same accuracy as a new human advisor who is advising his first student. This means that the chatbot may provide information that is incorrect, but for the majority of the time the general frame of the response should be accurate. Note that this does not deal with unrelated responses to conversation, issues stemming from the chatbot not being able to recognize the user's intent should fall under problem (1) as it is a problem of language processing.

However, even with a focus on two specific problems for this assignment, the overall scope of this project is still too large. Designing realistic chatbots that are capable of providing accurate information part of ongoing research at large corporations such as Apple or Microsoft. We are unable to satisfy all types of conversation within our timeline of one week of design and analysis. Therefore, the following assumptions have been made in order to limit the scope of the chatbot project:

(1) The user is rational and always provides input relating to advising
(2) All courses mentioned should be in the format CSCXXX, with no spaces between the number and the department, and the department is stylized as capitalized letters.
(3) There is a maximum of one course information mentioned per user input All courses mentioned are in the Computer Science department.
(4) (For Q&A model only) All user inputs are questions. And these questions should not result in a boolean (yes or no) answer

Assumption (1) is made out of rationality. In an actual serious setting, a user would probably not ask a advising chatbot about its favorite color just like how a student wouldn't ask a professor his or her favorite ramen flavor during an advising session. Asking such questions would throw-off most humans, which are already better at handling random questions than a bot. Measuring correctness of response from random input giberish is meaningless when we already know that the chatbot does not have a correct answer. While it may be possible to filter out such user input in some way that preserves "naturalness" of the conversation. Our group decided to focus on the two problems that are listed above.

Assumption (2) is made out of convenience. We already know that how information is presented to a transformer-based model may affect whether it recognizes specific tokens or not. And since we provided all the data in the form described above, this assumption allows us to not have to do repeated work for training models or providing contexts for "CSC171", "csc171", "CSC 171", or even "cSc 171", since semantics of the course number is the only thing that matters here. If user representation really matters, this assumption can be safely removed by performing an extra search and replace on the user input string, which should allow the correct tokens to be passed to the chatbot model. Our current implementation recognizes all variants of course numbers as courses, but the models have not been trained on these specific course numbers.

Assumption (3) is made due to our resource constraints. While we could theoretically train/fine-tune a model to recognize relations between two different courses, our group does not have the resources or time to train such a model within the given one-week time frame (for $n$ courses, there

would be $O(n^2)$ amount of work instead of $O(n)$). Meanwhile, using only training data from one course at a time allows us to better fine-tune our models, making the output results more accurate.

Assumption (4) is made out of necessity, as the Q&A model that we used only supports answering questions based on a given context. While we could have merged the Q&A model with another model to provide responses regardless of whether the input was an open-ended question, such an approach sacrifices the accuracy of answers provided by the Q&A model.

## 4 IMPLEMENTATION DETAILS

Based on the problems that we've identified and would like to solve as well as the assumptions we've made for this project, we came up with the following implementations for our chatbot, with one using a focusing on answering questions accurately and the other focused on making sure generated dialogue is as natural as possible. The two different implementations are built on the same common-base for easy comparison and development.

### 4.1 Course and Advising Data Collection

We decided to use our own course information parsing script instead of the one provided for this assignment. This is due to two reasons: first off, we believe that difficulty in generating a new course database from CDCS information should be considered when thinking about maintainability of the chatbot. As the information provided by Justin in SQLite3 format is unable to be easily updated without additional work put in when the course list updates. Second of all, the CDCS information needs to be manually filtered due to its use of abbreviations and outdated prerequisite names. A lot of course titles use ampersand for "and", which is unlikely to be used by the user supplying input. Some courses refer to the old Arts and Sciences naming scheme for courses, which is difficult to lookup. Some of the courses listed in CDCS aren't courses at all, but PhD research reservations. Because of this, we decided that using our own scraped then manually sanitized data is the best course of action, for both the scope of this project as well as future maintainability. While we understanding that manually sanitizing the information may be time consuming for future database updates, only very few courses get updated or added each semester, and we believe our handling of this problem shouldn't be a major overhead in terms of adding new course information into the system. We expect a maximum of 1 hour needed to add all new courses for a new semester to the database if needed.

Data collection scripts and course information are located in `scripts/` and `data/` respectively. Some scripts may require additional dependencies in order to work, such as `BeautifulSoup` (`https://pypi.org/project/beautifulsoup4/`).

### 4.2 Chatbot Framework

The overall chatbot framework is fairly straightforward to use. Starting up the program starts a chatbot read-print-eval loop that initializes the model used and corresponding data structures then starts reading user input. User input is concatenated until a specified end of input indicator is inserted on a new line (`(over)`, `(o)`, or `.`), then the framework feeds the concatenated input to the model, retrieves a response as a string, and prints the respond using a formatter that prints out strings in the required output format. If a end session indicator is specified (`(over and out)`, `(oo)`, or `bye`), the framework checks if there is any remaining input that the model should handle. If so, it tries to print the generated response before it exits. Otherwise, a simple "bye" is printed before the exit. Details on the specifics of each model and their surrounding implementation will be described in the following two sections.

### 4.3 Q&A Model Implementation

The Q&A model implementation primarily uses the Question and Answering pipeline provided by `rust-bert`, which is a pre-trained model used to find answers to a question provided a given context string. Thus, the user input and a context is provided to the model to generate a response each time the user asks a question, which then produces an output substring from the context string.

One issue we realized early on with this pre-trained model is that the longer the context string provided, the less accurate the answer is. This is due to multiple factors: not only does the context have to contain course descriptions that refer to a subject using "it" or "this" which confuses the model, the model also hones in on certain keywords and ignore other parts of a sentence when searching for answers, which may happen to exclude important information such as the course number. Therefore, questions such as "What is the name of CSC171?" may end up only looking for sentences with "name" in it instead, making a statement in the context such as "the name of CSC172 is Data Structures and Algorithms" confusing to the chatbot as it seem to imply a name despite it being for the wrong course. This is worsened by the fact that the context scales linearly on average with the number of courses in the database. So by the time the entire course selection is added, the accuracy of the output statements may be questionable, as it is highly likely it is the answer for another course.

As a result, we decided to partition the context into several parts. Each course is given it's own unique context string scraped from CDCS information including the course title, instructor name, number of credits, terms offered and course description. General advising questions with no specific course given a "default" context with information from the department website. This partitioning of contexts limits the size of the context for each question. When the user asks a question, the input string is pattern matched against the course number to select a context string. If no course number can be matched, the default context is used. These contexts are stored in a file system backed plaintext database specified at runtime. (Our example database is `data/db`)

While regular expression matching of course numbers is trivial, it is much harder to determine the correct context to use in the case where a user mentions a course title instead of the course number. "Introduction to Artificial Intelligence" is the same as "Intro to AI" to humans, but are two completely different titles to a program. While the course database used has been manually edited to use full course titles instead of shortened ones sometimes listed on CDCS, users may still chose to use the short title, confusing the context selection logic. Our solution to this problem is a fuzzy keyword search. We first parse the keywords of various courses by removing punctuation and expanding various acronyms from CS jargon. We then compare the keywords to the words in the prompt using Levenshtein edit-distance to account for spelling mistakes. We give each course a score given by the proportion of the keywords found in the prompt to the number of keywords in the title. If the highest score is above a certain threshold, we assume that the input is referring to that course, and use that course's context. However, if no score reaches the threshold, we use the default context as defined before, assuming that the user is asking a generic question about advising.

### 4.4 Conversation Model Implementation

The conversation model is our attempt at making the response of the chatbot as natural as possible. In order to do this, we use the `rust-bert` conversation pipeline with our own custom model to generate responses. The model itself is pretty simple, it takes an input then generates an output response, while keeping track of all previous iterations of conversation to use as context. The main thing that we had to do was bake the CDCS information into the model.

As the course information is not enough information to fully train a model. We've decided to use pre-trained models as a base then fine-tune the models with our own set of questions and answers parsed from the course information we've collected. The models used are listed below. However, due to computational resource limitations, we have little success in getting this model fully running at a reasonable efficiency. The size of the models (around 7GB) also forces us to upload them to an external Google drive instead of including them in the blackboard submission.

*4.4.1 GPT2.* The first NLP model was implemented on our Chatbot using OpenAI's gpt-2 (`https://github.com/openai/gpt-2`) language model. Training was done using text files parsed from Rochester's CDCS website. Implementation details can be found at (`https://drive.google.com/file/d/1Bgce9hI0OEpLC52zh1IIch_DG5VjkwEN/view?usp=sharing`). However the process was the following:

- Acquire tensor transformers from HuggingFace (`https://github.com/huggingface/transformers`)
- Run provided finetuning script (`run_langauge_modeling.py`) with parameters:
  - model_name: gpt2−medium
  - logging_steps: 500
  - train_ data_file: **courses_train.txt**
  - eval_data_file: **courses_verify.txt**
  - block_size: 128

Once training was complete, the .bin file was converted to a .ot file using **rust−bert**'s (`utils/convert_model.py`). This file can then be fed into a **rust_bert::pipelines::conversation::ConversationConfig** structure as a LocalResource. The performance of this model as it applies to our Chatbot is quite poor. The size of the gpt-2 data set (1.5 GB) is far larger than the size of our fine tuning data (48 KB). The script used did not bias the fine tune data nearly heavily enough, and the responses to basic queries regarding U of R courses is met with semantic nonsense.

*4.4.2 DialoGPT.* The next attempt was to select a Q and A response based model and finetune that. DialoGPT is a tuned version of gpt-2 trained on Reddit responses. Using a .csv file parsed from CDCS. Implementation details can be found at: (`https://colab.research.google.com/drive/1GM0P4MVYlCNheWnHeUru9qOgsCyENiNf?usp=sharing`) However the process was the following:

- Acquire tensor transformers from pip
- Prepare a masked langauge model (MLM) using
  - AdamW optimizer
  - transformers:
    - AutoConfig
    - AutoModelWithLMHead
    - AutoTokenizer
    - PreTrainedModel
    - PreTrainedTokenizer
    - get_linear_schedule_with_warmup
- Custom parameters (see class Args in notebook)

Data was constructed from CDCS parsed .csv file. Each course description was used as data rows, with the previous 7 rows used as context. Cell [13] contains the head of the data file. The model configurations was sourced from

(https://towardsdatascience.com/make-your-own-rick-sanchez-bot-with-transformers-and-dialogpt-fine-tuning-f85e6d1f4e30).

Unfortunately time restrictions prevented us from completing an appropriate model. The previous working configuration required too much GPU VRAM (Almost 20 GB). Even using gpt-2-small did not help this. Modifying batch sizes and various other parameters yielded confusing results, including but not limited to

- Asymmetric tensors
- Internal CUDA errors
- Missing header files
- Inconsistent results between runs

Our only thought for the last bullet point is that a shortage of VRAM is at fault. Tensors of varying sizes were being created with the same data and same parameters. Created models would almost always fail verification. 6 models were completed without memory errors, titled pm1-6. However all 6 models failed to be loaded into the **rust_bert::pipelines::conversation::ConversationConfig** structure as a LocalResource. Missing header files and Asymmetric tensors were often at fault. We thought verification in the colab would ensure same-sized tensors, but we were apparently mistaken. Given more time, We are confident we could get to the bottom of this issue. Unfortunately the complexity of the issues is too deep for us given the time frame.

All data can be found at:
(https://drive.google.com/drive/folders/1v276bQhSXx8IUsXWJZrX9yXVgj41z3v5?usp=sharing)

- data
  - Contains parsed data files, .csv's, and cache outpus from model
- output
  - All notebooks write to this directory
- model
  - dialo_trainer.ipynb
    - DialoGPT trainer
  - trainer.ipynb
    - gpt-2 trainer

## 5 TESTING AND RESULTS

Due to time constraints, we did not have time to setup a quantifiable number of tests to measure the naturalness of the conversation and accuracy of the response. However, we have determined a general method of testing provided that there is a large enough sample of testers.

The testers provide the same input to both implementations of the chatbot, with the input being questions of their choice. These questions and generated responses will be recorded so that we can determine factualness of the answers. Meanwhile, the testers will be prompted to rate the conversation of the two chatbot implementations on a scale from 1 to 5, with 1 being very robotic and predictable and 5 being basically human. The recorded percentage correctness allows us to determine accuracy of output, while the comparative conversation rating would allow us to at least determine which chatbot is better at initiating conversation.

While our two chatbot implementations have not tested in a quantifiable manner, we have run some example queries on both implementations with a mix of correct and incorrect responses. Some of the example queries are as follows:

- What undergraduate degree programs are available?
- What are the goals of the CS program?
- Who are the advisors?

- Who can I talk to about graduate school?
- Are there any research opportunities in the department?
- Who teaches CSC254?
- Who teaches Introduction to Artificial Intelligence?
- Who teaches Intro to AI?
- When is CSC282 offered?
- How many credits is CSC200H?
- How can I declare my major?
- What are the prerequisites for CSC242?
- Is the CSC172 lab required?
- Can I double major in computer science and data science?
- Can I use CSC291 for writing credit?

Answers to the example queries for the Q&A model are provided in Appendix A. The following subsections will discuss the results of running common queries on the two chatbot implementations.

### 5.1 Results on the Q&A Model

The Q&A model implementation is capable of handling most questions within its context at ease, providing relevant parts of the context as answers. Most of the responses are very accurate, even when the question is not phrased in the same way listed in the context. Realistically, this type of chatbot may be useful replacement as a search function for a Frequently Asked Questions section for a course description website such as CDCS, with links to the origin of the answer to allow the user to validate the responses it gives (and possibly a disclaimer of its accuracy to go with it).

One issue we've discovered with this model is that is may sometimes respond to the user with a phrase taken out of context. We've discovered that this may happen sometimes with negated sentences. For example, several course descriptions contain the following sentence: "If the course is closed, do NOT email the instructor, sign up for the waiting list here: <url>.". However, when a user asks what to do when a course is closed, the model may see "email the professor" as a reasonable substring response to return, which is the exact opposite of what the student should do, while this can sometimes be easily fixed by allowing the model to return longer responses, it does not solve the root problem of the Q&A model not understanding the idea of negating a predicate, as statements of longer length may still run into the same problem. Overall, the model would either need to be fine-tuned to understand negation and not return it (or on a easier but less useful level, return the predicate with negation), or negated sentences need to be rephrased in the course description.

While the Q&A model is capable of answering questions in its context correctly the majority of the time, the model does not handle questions not in its context well, as it is incapable of generating its own responses to questions. In fact, when a question is not within its context, the model will try to always answer something from its context, regardless of how useless the information may be. One way we could avoid this is to set a threshold for the confidence score of the answer and only display our answer when the confidence is high enough, but we realized that even with correct answers, the confidence score tends to be all over the place, making it possible for us to filter out a lot of correct responses with a low confidence such as 30%. Luckily, including these "bad responses" does not affect our overall accuracy, as all information included in the context database are factual, the response is most likely to be factual (other than in the case described above with negated sentences). It is simply not relevant to the conversation. However, this does mean that the conversation is a bit less natural, as the chatbot may respond to a completely different question that the user is asking, without prompting the user that it does not have a good enough answer.

The Q&A model also has trouble generating full length responses, due to it taking phrases and sentences from the original context string. While it can answer "What is a prerequisite for CSC172" with "CSC171", it is unable to form a full sentence with the response due to the lack of sentence generation. This means that the responses generated are uncharacteristically short, which may lack the usual polite tone of an advisor during an advising session. This itself is not too much of an issue, as it meets our minimum requirement of understanding natural language user input. However, we would like our chatbot to be able to respond naturally in this situation as much as possible, and the short responses feel a bit too impolite for this context. This naturalness problem stems societal norms in English culture, and the chatbot is unable to recognize tone differences or even generate sentences.

### 5.2 Results on the Conversation Model

The current iterations of the gpt-2 and DialoGPT model are subpar. The initial gpt-2 model suffers from a lack of fine-tuning data. This then led us to DialoGPT, which is gpt-2 finetuned on several million reddit comments. Finetuning this with a smaller amount of data showed great promise, specifically the Rick and Morty bot that was referenced in the methods. However, time constrains left us unable to prepare a functional model. All models that made it out of the training stage are in the (`models/`) directory. Raw pytorch output under (`models/bin/`) and transformed rust-bert models under (`models/ot/`).

Models can be found in (`models/`) in
(`https://drive.google.com/drive/folders/1jwvZh8oya68Tod3YOtpTBP_hgM7AmdeQ?usp=sharing`)

However, integrating generic models fine-tuned for other purposes led us to realize that while dialogue generated from the conversation model are realistic and natural, much of the answers generated are probabilistic, which would not meet our need for accuracy. Assuming that we have enough time to fine-tune the DialoGPT model, our answers for a specific course section would still be mixed up with information from another course, which is not what we want. Therefore we can conclude that the accuracy of such a conversation model would not meet the minimum accuracy requirements of our chatbot.

## 6 MAINTAINABILITY

While this assignment itself is a one-off project. We decided to also take a look at maintainability of the chatbots, as it's important for any real-life system to be able to update its information easily. For this section, we will assume that there will be 1 new course added to the CDCS database, thus the chatbot's internal course database would need to also be updated. We will look at the overall time consumption and resource requirement for each of these implementations and attempt to identify the where the difficulty lies.

### 6.1 Maintainability of the Q&A Model

As the Q&A database is in plaintext and partitioned into separate files for each course, adding a new course to the database is as easy as adding a new file to the "`data/db/`" directory. This makes it take a constant amount of time for each new course. However, each course should be manually sanitized before added into the database. Furthermore, since the Q&A is incapable of generating answers of questions that it does not have the answer to, the maintainer needs to be able to think of any related questions to the course beforehand and add in appropriate responses as predicate statements in order to maximize the chatbot's question interpretation ability.

For resource consumption, any amount of computational power could do, as the actual question and answer model is pre-trained and does not need to be modified for additional database entries. In fact, low computational requirement for adding a database entry makes it possible to create a

web-based submission form where a user with no computational background can add new entries to the database by simplifying filling out a form. This makes the Q&A model chatbot implementation extremely useful and flexible, despite it's limited responsiveness to things that are not open-ended questions.

### 6.2 Maintainability of the Conversation Model

As the conversation model chatbot implementation has the contents of the course database baked into the pytorch model via fine-tuning. Which makes it significantly more complicated to add a new course. As the conversation model can generate reasonable responses to questions, we would only need to provide the essential details of the course (number, title, credits, terms offered, instructor, description, etc) in question and answer form to fine-tune the model, which is overall less information. However, the time needed to train the model is significantly longer than the amount of time saved by preparing less information. Therefore, the time cost of adding a new course to the system is quite high.

In terms of computational resources, the fine-tuning of the model would require a significant number of GPUs to be feasible. The amount of computation also increases with each course added, as we would have to fine-tune a new model with all the data in our course database in order to not skew the model towards questions from the new course. This restriction is on top of the fact that we already have issues getting enough resources to train the model in the first place, making such a chatbot even more expensive to maintain.

## 7 FURTHER IMPROVEMENTS

Some ideas are not explored in our current implementation(s) of the chatbot due to time and resource constraints. They are listed below. We propose the following improvements to further enhance the capabilities of our advising chatbot if given the opportunity to work on it further:

### 7.1 Merged Models

As discussed above, the Q&A model is capable of providing reasonably accurate responses when responding to questions, while the conversation model is better at handling input that is not questions and generating small-talk in a natural way. It may be possible to have a classification model that runs on the input to determine whether the user input is a question or a statement, then run on different models based on their speciality. This would create a merged chatbot capable of handling different kinds of input, which would have the same accuracy as the Q&A model when handling questions, but also feel more human-like in response.

### 7.2 Smart Context Merging for Q&A Model

Currently, the chatbots operates on the assumption that we do not talk about two courses at the same time. For the Q&A model, we could have the context key scanner look at all course titles and numbers, then concatenate multiple contexts to create a new context when dealing with questions mentioning multiple courses. Of course, this would lower the confidence of the answers due to the model's limitation. However, this would allow us to handle questions involving multiple courses we were unable to handle before.

### 7.3 Fully Custom Conversation Model

For our conversation model, we had to base much of our conclusions off of generic models due to resource constraints on training the model. If given enough resources, we may be able to train a conversation model from the ground up, using actual advising conversations as a basis to create a

better model for the chatbot. While accuracy would still suffer due to the probabalistic nature of the responses. This would help levitate some of the randomness of the responses.

## 8 CONCLUSION

Different transformer-based models provide different benefits to creating realistic advice providing chatbots. While the Q&A model is capable of generating accurate responses, it is limited by the context given and the lack of sentence generation, making it less natural. Meanwhile, conversation models are capable of generating sentences on the fly and provide natural conversation feedback, at the cost of information accuracy and steep modification cost. While both aspects are important to chatbots, the decision on which to use fall to the developer and the importance in terms of the specific project. For projects that value accuracy over naturalness, we recommend the context-based question and answering model. For projects that provide static information and require natural sentence generation, the conversation model is better. It may be useful to merge these two models together with a classifier to make use of the best of both worlds when creating a chatbot. The GPT based approach was not a success. This was mainly due to the complexity of fine-tuning giant models in such a short period of time. Given more time and further research into the proper methods that Microsoft used, we believe a neural network (NN) approach is viable, if not even desirable due to its adaptability. Previous examples of fine-tuned GPT models showed great promise, which leads us to believe that NN methods are viable for a context-specific chatbot.

## 9 APPENDIX A: EXAMPLE INTERACTION WITH Q&A MODEL

The number in parentheses is the confidence score.

```
> Hi, I'm Ted. How may I help you today?
> (over)
What undergraduate degree programs are available?
.
> BS in computer science, a BA in computer science, and a minor (0.787)
> (over)
What are the goals of the CS program?
.
> To give students a solid and rigorous background in computer science principles (0.342)
> (over)
Who are the advisors?
.
> full-time Computer Science faculty members (0.574)
> (over)
Who can I talk to about graduate school?
.
> Faculty Advisors (0.611)
> (over)
Are there any research opportunities in the department?
.
> Faculty Advisors are your go-to for questions about research within the Department (0.050)
> (over)
Who teaches CSC254?
.
> Michael Scott (0.987)
```

```
> (over)
Who teaches Intro to AI?
.
> George Ferguson (0.983)
> (over)
Who teaches Introduction to Artificial Intelligence?
.
> George Ferguson (0.990)
> (over)
When is CSC282 offered?
.
> every Fall and Spring (0.608)
> (over)
How many credits is CSC200H?
.
> 4.0 (0.889)
> (over)
How can I declare my major?
.
> please use the Hajim Intended Major Change form (0.153)
> (over)
What are the prerequisites for csc242?
.
> CSC172 and MATH150 (0.972)
> (over)
Is the CSC 172 lab required?
.
> Lab and workshop required (0.549)
> (over)
Can I double major in computer science and data science?
.
> not permitted (0.449)
> (over)
Can I use csc291 for writing credit?
.
> optionally available for writing credit (0.403)
> (over)
What will I learn in Intro to CS?
.
> algorithmic thinking and computational problem solving (0.465)
> (over)

(over and out)
> Bye!
> (over)
```