

# AILI

## Your Friend at Uni

---



# UNSW

THE UNIVERSITY OF NEW SOUTH WALES

Team: *Mind Reader*

Members:

Ruixue Gu, z5107143, Project Master  
Mou Feng Zhang, z5187906, Developer  
Theotius Hall, z5059325, Developer  
Tianyu Han, z5102893, Developer

## 1. Introduction

### Existing System:

Currently, in order to find information concerning UNSW, students must either attempt to navigate through the confusing and opaque university websites or make a visit to student services. The different student services are a great way for students to find the help they need, especially with some of the complicated tasks facing those new to uni. However, there are many simple questions that only serve to take up the time of the everyone involved.

### Comparison with AILI:

AILI is what we call the chatbot we implemented, it is a Strong Decision Rule-based bot with several optimizations related to the domain of UNSW.

Existing System	AILI
Can help guide students through complicated, multi step processes and provide them with information that they weren't aware they needed.	Can help with simple to medium queries about the uni, courses, subjects, etc.
Only available during office hours	Available 24/7 unless there is a server issue
Must pay wages to people working in the office	Aside from dev and some minor maintenance it's free
Can only serve a few people at a time	Can serve as many people as the servers allow
Limited to phone, email and office desk	Can be instantly accessed on the web at any time and place
Takes time to reply to emails, walk to office or call up	Very quickly replies as soon as it receives a new message

As shown here there is still a definite use for the current existing system however AILI could be deployed alongside to complement the student services.

## 2. Background

### Usage Scenario

Chatbots are being utilised all over the place to help people find information, book appointments and perform other simple tasks that don't require the attention of a

support employee. With their usefulness apparent they are also a focus of research and are becoming smarter and more convincing all the time.

Many students have experienced large wait times when trying to contact help services at UNSW. A lot of these students only needed a small bit of information or help with a task that could have been solved by an automated chatbot. Employing such a system would free up a significant amount of time for the help staff and students alike.

The goal is to lower the burden on the support faculty and reduce the wait time for students who need help. This project aims to use a chatbot to help within the domain of student services here at UNSW, from important dates to information about courses.

There are essentially two different categories of chatbot:

<b><i>Rule-based bots</i></b>	<b><i>AI bots</i></b>
<i>In a rule-based approach, a bot answers questions based on some rules on which it is trained on. The rules can range from very simple to very complex.</i>	<i>AI allows the bot to learn from the interactions it has with its users. Involved in this learning are analytics platforms, integrations with APIs and other elements that provide the AI with the resources it needs to communicate effectively with the user.</i>
<b><i>Merits</i></b> <i>The creation of these bots can be relatively straightforward when using some rule-based approaches but it can be difficult to model complex functionality.</i>	<b><i>Merits</i></b> <i>AI bots can learn from dialog with end users. The more bots interact with users, the more intelligently the bots reply. This makes the learning process much longer but allows for some extremely complex behaviour.</i>
<b><i>Demerits</i></b> <i>The bot is not efficient in answering questions, whose pattern does not match with the rules on which the bot is trained</i>	<b><i>Demerits</i></b> <i>It is fairly complicated and time-consuming to implement an AI bot and the learning process is potentially life-long or will require a huge amount of conversational data.</i>
<b><i>Existing Systems</i></b> <i>Taobao Customer Service Chatbot,  Ctrip Customer Service Chatbot,  Qunar Customer Service Chatbot, Alime</i>	<b><i>Existing Systems</i></b> <i>Siri, Xiaoice, Alexa, Google Assistant</i>

## System Architecture

### ChatBot System Architecture

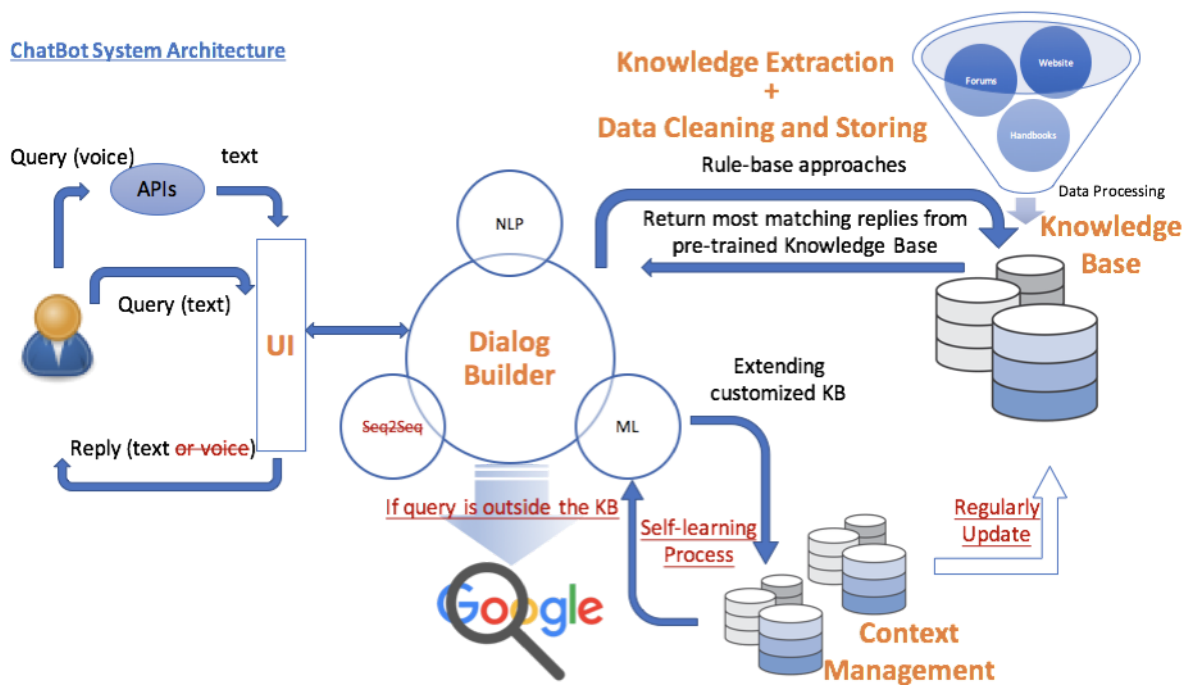


Figure 1: Overall System Architecture

Overall, there are four components in our system: Dialog Builder ([Section 3](#)), Data Extraction, Cleaning and Storing ([Section 4](#)), UI ([Section 5](#)) and Context Management ([Section 6](#)).

We successfully implemented most function modules mentioned in our proposal and reasonably extend the functionalities. (See [Section 8](#) Functionalities for details.)

### 3. Dialog Builder

Input

Output

User's query

Answer from chatbot

What's ur name

I am Aili, your friend at UNSW.

At first, by utilizing Stanford coreNLP and Random Forest Algorithm(RF), we implemented a relatively Weak Decision Rule, acting as a classifier to organise user inputs into "Statement", "Chat" or "Question". After this classification we send the

input to the relevant bot, which is setup to handle a specific kind of message, and return its answer to the user.

Main Steps: (See Fig. 2 below for details.)

- Parse input sentences to dependency graph utilizing Stanford coreNLP
- Extract features from dependency graph preparing for classification
- Prepare training data to train Random Forest Classifier

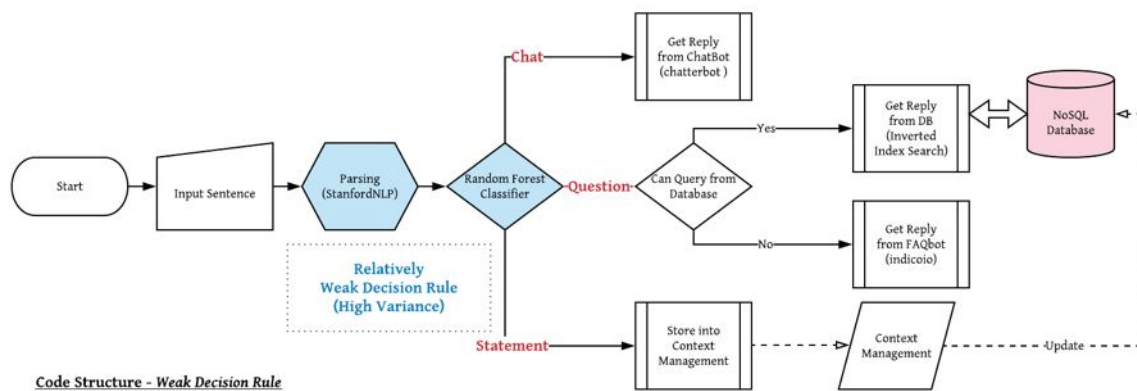


Figure 2: Code Structure of Discontinued Weak Decision Rule Bot

The RF achieved very high accuracy (91% approx.) for training data but did not perform well when testing random customized inputs, which suggests that the model has high variance.

Therefore, we changed our model to a Strong Decision Rule using definite decision rules instead of classification with the intention of lowering variance.

Main Steps: (See Fig. 3 below for details.)

- Process input sentence using primary NLP functions in NLTK
- If query concerns a real timetable, get a reply from corresponding websites; otherwise
- If query concerns a course or program (Handbook) related information, get a reply (Information Retrieval) from corresponding collections in NoSQL database; otherwise
- If analogous query has been answered in Frequent Ask Questions, get corresponding answers; otherwise
- Connected to external search engine (Google) to get reply.
- If a query goes to one of the decision rules (“FAQ Information?” or google) it will be stored in Context Management. This allows developers to find out what kinds of information people are requesting that the bot does not understand so they can update their database and add on new rules regularly.

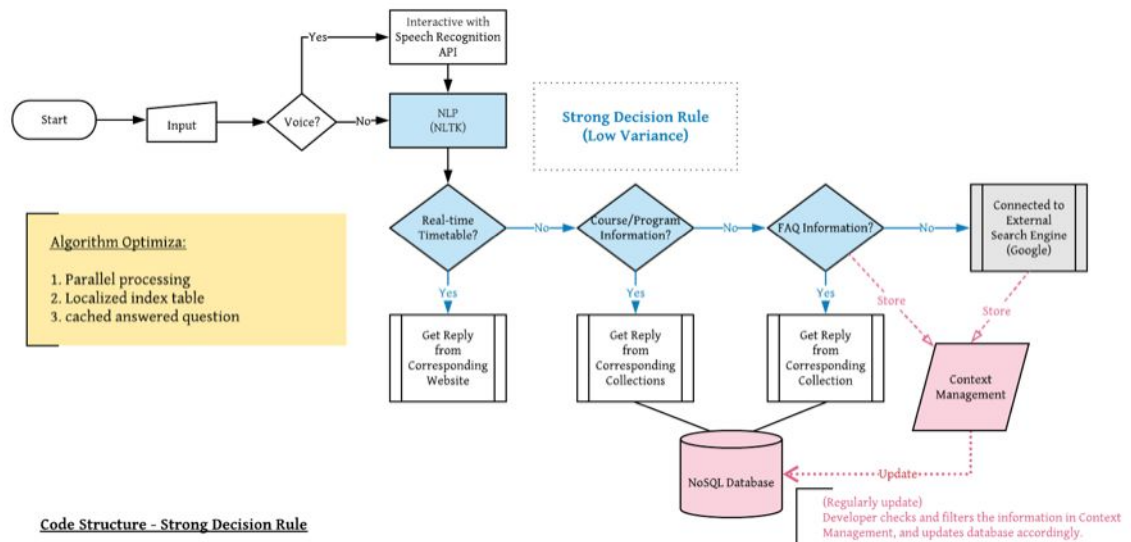


Figure 3: Code Structure of Strong Decision Rule

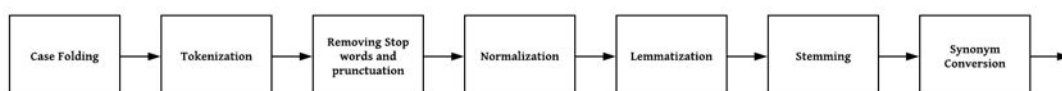
## Dialog Builder / Query Generation (upgraded)

Translate parsed sentences into valid queries, categorize the query and act accordingly with wise usage of NLP and Classification in Machine Learning.

Due to the issues of high variance in our first implementation, we upgraded our model to Strong Decision Rule. Now using definite decision rules instead of classification to lower the variance in our answers.

### Core Features: (finished with changes)

- Select a known statement that most closely matches the input statement.
- Return a known response to the selected match and a confidence value. based on the matching-
  - Modified: -using Information retrieval through tf-idf and decision rules in sklearn.
- Utilize Natural Language Processing, Sentence Classification and Natural Language Grammar processing.
- Updates to original plan:
  - Modified feature above.
  - We implemented relatively weak decision rules and strong decision rules and through comparison we chose the strong model for our final product.
  - The final version of Nature Language Processing flow is as below in NLTK.



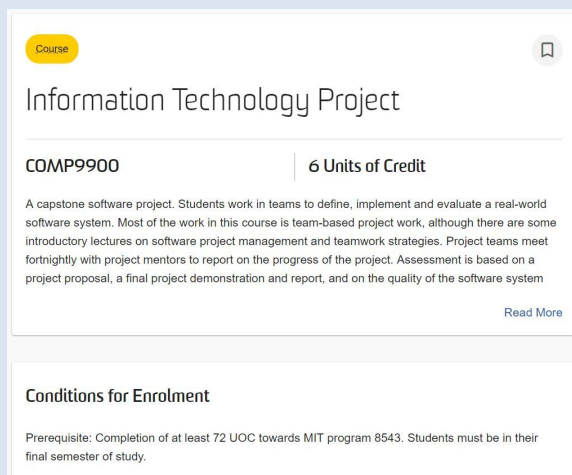
### Extra/Optional Feature: (finished with changes)

- Build an actual timetable, if users query for one, for some courses.
- Feed interactions with end users into knowledge base that enables life-long learning by chatbots.
- Updates to original plan:
  - Instead of building a timetable we can grab real-time timetable info and display it to the user.
  - Developer will finalize learning process periodically by updating database and adding on rules instead of using it for machine learning.
  - Relying on the definite decision rules to get a reply from corresponding resources or collections in our database instead of recognizing the type of queries and choosing the appropriate model automatically.

## 4. Data Extraction, Cleaning and Storing

### Input

#### UNSW Webpages



### Output

key-value pairs stored in collections in NoSQL database (MongoDB)

```
{"type": "Course",
```

```
"title": "Information Technology Project",
```

```
"key": "comp9900",
```

```
"detail": "A capstone software project. Students work in teams to define, implement and evaluate a real-world software system. ...",
```

```
"enrolment": "Prerequisite: Completion of at least 72 UOC towards MIT program 8543. Students must be in their final semester of study."
```

```
"credit": "6 Units of Credit",  
.....}
```

## **Data Extraction**

Data extracting is a very import module of this project. The more valuable the data used to train the chatbot, the more helpful an answer we can give to users. In this project, we will focus on information on the school website. We extracted the information we needed from the web pages based on keywords and specific HTML format using the BeautifulSoup library for static pages and selenium for dynamic ones.

### **Core Features: (finished)**

- For different types of data, we have different extractors.
- Structured data extractor: This type of data is most valuable. Data is well organized and described. We extracted the data, did data preprocessing and stored it in the database. (Large)
- Real-time data extractor: Some data must be real-time, such as available positions of courses. We did not store such data in the database, instead we extracted it from web upon a user's query. (Medium)

### **Extra/Optional Features: (finished)**

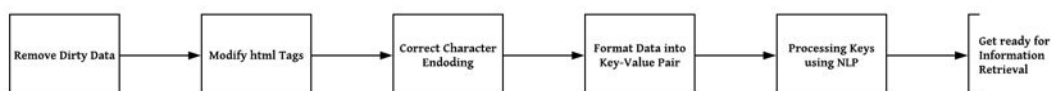
- Unstructured data extractor: This kind of data has not been described or is poorly described. We had to do much more work to extract useful information from it. But we still used this kind of data to supplement our feedback. (Large)
- Automatic updating of our structured and other stored data. (Small)

## **Data Cleaning and Storage**

Having structured data stored locally allows for a faster search and response process when answering a query. We need to process and clean the data in order to ensure it is more relevant and searchable while optimising storage space.

### **Core Features: (finished and added more data preprocessing)**

- The data we get from the websites is stored locally in a database (Small)
- By comparison (NoSQL, SQL, etc.), we decide the data structure/schema of knowledge base that guarantees higher efficiency and lower space-consuming.
- Updates to original plan:
  - Added more Data Preprocessing to increase accuracy of further queries.



### **Extra/Optional Feature: (finished with optimization)**



- Deploy our database to the cloud for greater performance, accessibility and data continuity. (Small)
- Updates to original plan:
  - Localize up-to-date index table and use the it to do information retrieval. This reduces network transmission time by only retrieving the target value and enhances query results through a more efficient algorithm on the server.
  - Parallel processing to write data to reduce response time as a result of bidirectional data flow (answers can be returned to client and data can be written to database synchronously).

## 5. UI

The purpose of the chatbot is to allow students to find information that may not need a staff member's attention and to be available from anywhere and at any time. A modular UI that can be used in different ways and incorporated into websites would best address this.

We practically implemented two kinds of UI, pop-up dialog box and website chat window, and decided to move to the latter as we felt it better displayed our ChatBot's functionality.

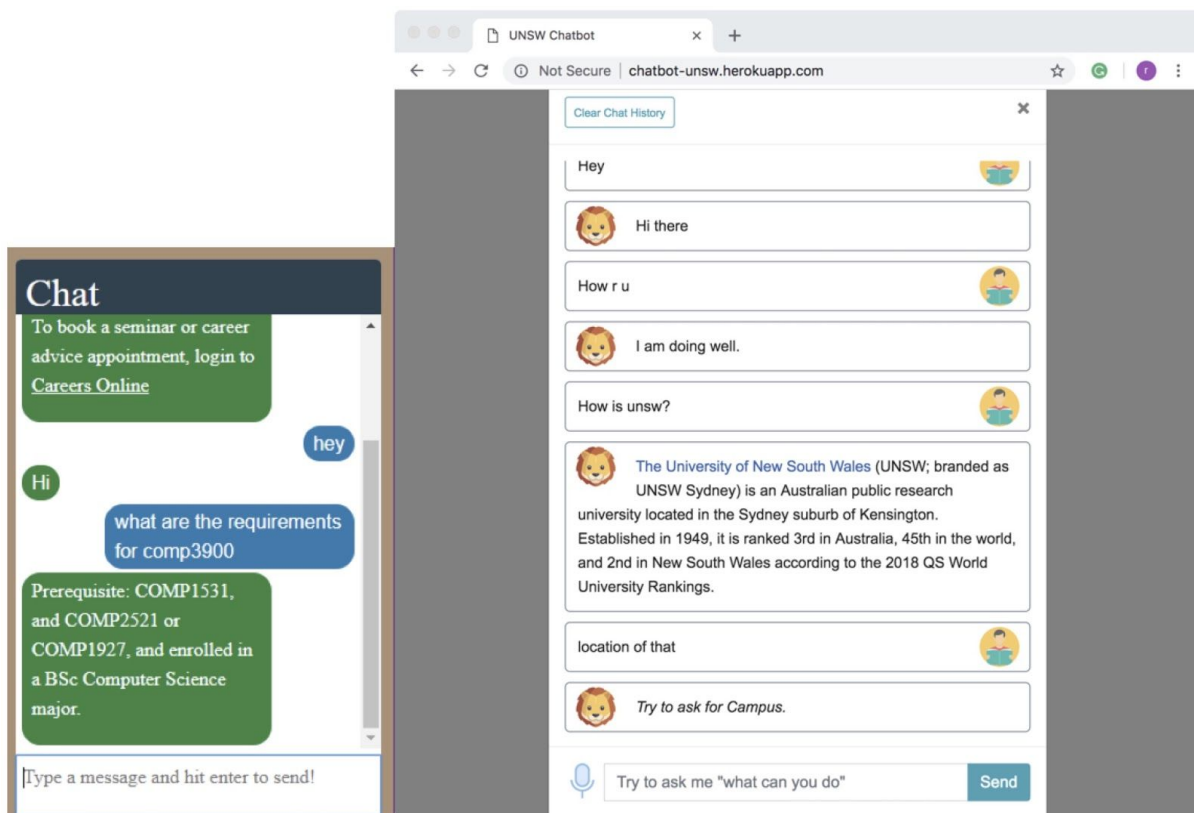


Figure 4: User Interfaces (pop-up dialog box & website chat window)

**Core Features: (finished)**

- A text chat in the same style as a messaging app or website.
- The user can type questions that are sent to the chatbot and it will respond in kind.
- “Typing” animation should be displayed to the user while the bot processes its reply.
- The chat can be inserted into a website or app in a modular fashion.

**Extra/Optional Features: (finished with changes)**

- Utilise a speech processing API, for example google assistant to allow users to interact with the chatbot. The user would propose a verbal question that would be converted to text by the google assistant and sent to the bot. The bot’s response could be provided in a text-to-speech form or simply as text, possibly both.
- This could be bundled together with the regular text chat, for example have a microphone button that would activate the speech processing element.
- Create a website that works on desktop and mobile to display our chatbot.

## 6. Context Management

The ChatBot utilises context management systems and integrates with APIs to gather information that helps it answer future questions more accurately and intelligently.

**Core Features: (finished)**

- Maintain a profile for each student and store all relevant information ready for further usage.

**Extra/Optional Feature:**

- After extraction and cleaning, utilise some data structure (ex. key-value pair) to feed into backend knowledge base.
- Updates to original plan:
  - Any query that goes to decision rules (“FAQ Information?” or Google) will be stored in a Context Management system. By checking and filtering that system a developer can update the NoSQL database and add on rules if required..
- Not Implemented:
  - Interact with log in system (if possible) to fetch relevant information.
  - Let the user rate the helpfulness and legibility of the chatbot.

## 7. Conclusion

### Implementation Challenges

#### 1. High variance of Random Forest Classifier

As we discussed in our overview, we attempted to implement a Random Forest Classifier based on the features extracted from a dependency graph parsed through Stanford Core NLP. At first, we fetched existing data with three classes (“Statement”, “Chat” and “Question”). After splitting this into training and test data, we achieved very high accuracy (91% approx.) both for training and test data. However, when testing random customized inputs, the classifier proved unstable, for example, it misclassified unusual greetings into “Statement” and could not always distinguish questions without a question mark.

We then prepared training data by collecting actual sentences and queries relevant to our ChatBot’s function as the tutor suggested that the data we were using may be too different from real inputs. Unfortunately, variance still remained high as our old model put the classifier right at the top and all follow-up processes relied on it’s accuracy. After discussion within the team, we decided to move to strong decision rule ensuring definite control of query flow, thus lowering the variance of the model.

#### 2. Specific data source causing inverted-index deviating

We are using UNSW websites as the data source, when utilizing inverted-index in Information Retrieval some non-key words will be entitled to high weights because of the restriction of the data source. For example, on examination of our database we noticed that “tell” is apparently not a keyword. This being because it appeared in the data source less frequently than larger and broader sources, it would normally have a relatively high weight when applying tf-idf operations.

First, we attempted to combine the existing data source with a broader data source to balance the overall assignment of weights. However, it more than doubled the current database and reduced the performance of database queries dramatically. Therefore, we decided to manipulate some of the features of the database manually.

#### 3. Time-consuming computation when utilizing indico to create FAQbot

Based on the data collected from FAQ from a UNSW website, we extracted text features and build up similarity matrix using indico’s Text Features API. Afterwards, the cosine similarities between client inputs and all tuples in the current matrix could be computed and top k confidence matching returned. However, this process proved

fairly slow, around a minute when run locally, which is not acceptable in practice. Therefore, we decided to move further towards information retrieval from our database with the help of NLP.

#### 4. More manipulation demanded to make Aili more smart

We released the ChatBot to public through WeChat, Facebook and LinkedIn waiting for more manipulation. The more clients talk with Aili, the more comprehensive the knowledge base will be and the more humanlike s/he will become. However, we could not gather as much information as we would have liked due to time and resource constraints.

#### Further Improvements

1. Our current system could not be regarded as “understanding” users’ queries. Instead of using a rule-based decision model, cutting-edge technologies, such as sequence-2-sequence model or RNN model could be implemented in the our project. This would allow the bot to better answer vague queries as well as search for more detailed and relevant information.
2. Chatbots are often expected to have personalities, or at least have some conversational knowhow. Adding this would make the experience of interacting with it smoother and more pleasant for users.

## 8. Description of Functionalities

Following our original plan we created a NoSQL database using MongoDB, we populated it with information gathered through web scraping and data processing. Next, we finalized the dialog builder with the tradeoff between bias and variance after attempting different NLPs (Stanford Core NLP or NLTK) and Decision Rules (RF or Strong Decision Rules). Meanwhile, we designed the most prevailing User Interface with our project logo on it, recognizing inputs not only in text but also in voice by means of speech recognition API.

After ensuring the ChatBot replied with stable and reasonable accuracy, we assigned additional rules to make the ChatBot seem more human-like and better answer the user’s questions. We then deployed the ChatBot to a public domain with the aim of getting more data on how users interact with it in order to optimise it further.

#### **Functionalities that are missing from the project proposal:**

- Text-to-speech replies
  - We tested some mainstream speech APIs in an attempt to implement text-to-speech but after some discussion we decided that the results were unsatisfactory. As this was not necessary for the project and only a minor optional feature we decided to cancel it.

- Sequence-to-Sequence Model
  - The FAQ data we collected was about 400 question-answer pairs. We found that not only was this was not enough to train our model but long sentences and multi-sentences are not applicable to seq2seq model anyway, therefore we did not utilize it.

**Functionalities that are different from the project proposal:**

- Regularly update database based on additional information collected
  - We originally intended to create an automatic process to update our NoSQL database in real-time. However, through considering that client inputs may vary and could be completely unrelated to the purpose of the ChatBot, to avoid unnecessary updates and to ensure that our database remained reliable we decided to keep it to manual updates
- The self-learning process is not real time as we expected
  - We originally planned for our ChatBot to train and improve itself through its interactions. However, we found that the conversations it had with users were too short to really use as training data outside of particular scenarios (e.g. reference words/pronouns). Consequently, we added on extra rules to make the bot more intelligent. We decided that it would be more desirable to have a static and reliable ChatBot and any improvements would be better done manually.
- Refer to an external search engine when we cannot answer a query
  - Following a suggestion from our tutor, we added a Google search as the last step of our decision rules if all else fails. We also make sure to record any query and reply in Context Management whenever this happens so that the ChatBot can potentially be improved or expanded based on how people use it.

Hey, Alumni. Your friend at **UNSW** 'Aili' is ready for chatting!  
**#Aili\_atUNSW #chatbot #UNSW #conversationalAgent**  
<https://lnkd.in/gA4jeVQ>

Many students have experienced large wait times when trying to contact help services at UNSW. A lot of these students only needed a small bit of information or help with a task that could have been solved by an automated chatbot. Employing such a system would free up a significant amount of time for the helpful staff and students alike.

Here is the chatbot 'Aili' we implemented for comp9900 with the goal to lower the burden on the support faculty and reduce the wait time for students who need help. This project aims to use a chatbot to help within the domain of student services here at UNSW, from important dates to information about courses.

Please feel free to manipulate with 'Aili'. The more you talk with him/her, the more comprehensive the knowledge base will be and the more humanlike he/she will become.

Our team members: **Daniel Zhang** Tianyu Han/**韩天宇** **Theotius Hall**

**UNSW Chatbot**  
chatbot-unsw.herokuapp.com



Figure 5: release of Aili

## 9. User Documentation/Manual

The project is now online: <https://chatbot-unsw.herokuapp.com/>  
(It may take several seconds to enter the webpage since the free web dyno will sleep after 30 minutes of inactivity.)

You can also install and run it in your computer locally, by steps:

- Install Python3
  - <https://www.python.org/downloads/>
- Install pipenv
  - pip install pipenv
- Download or clone project
  - git clone <https://github.com/Theotius/AILI.git>
- navigate to folder \chatbot-unsw:
  - cd AILI\chatbot-unsw
- Install all dependencies for the project
  - pipenv install
- Run nltk\_install script:
  - python nltk\_install.py
- Activate virtualenv of the project
  - pipenv shell
- Run the application
  - flask run

- view the webpage using link showed on the Terminal
  - <http://127.0.0.1:5000/>

## References

(For final version of the ChatBot)

### Data

- <https://www.handbook.unsw.edu.au/>
- <https://unswinsight.microsofttermportals.com/>
- <http://timetable.unsw.edu.au/2019/subjectSearch.html>
- <http://legacy.handbook.unsw.edu.au/2018/index.html>
- etc.

### Clouds/Services

- MongoDB in cloud
- Herokuapp
  - <https://www.heroku.com/>
  - Server hosting

### APIs

- JavaScript Web Speech API
- Google Custom Search Engine

### Libraries

- selenium: dynamic web page extractor
- BeautifulSoup: static web page extractor
- requests: HTTP library
- flask\_pymongo: MongoDB api
- re
- threading
- nltk
- sklearn
- scipy