6-13-2019

A Questioning Agent for Literary Discussion

Robbie Culkin

Tim Shur

SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER ~~SCIENCE AND~~ ENGINEERING

Date: June 12, 2019

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Robbie Culkin
Tim Shur

ENTITLED

A Questioning Agent for Literary Discussion

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

[illegible]
Thesis Advisor

[illegible]
Department Chair

A Questioning Agent for Literary Discussion

by

Robbie Culkin
Tim Shur

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 13, 2019

A Questioning Agent for Literary Discussion

Robbie Culkin
Tim Shur

Department of Computer Science and Engineering
Santa Clara University
June 13, 2019

ABSTRACT

Developing a compelling and cohesive thesis for analytical writing can be a daunting task, even for those who have produced many written works, and finding others to engage with in literary discussion can be equally challenging. In this paper, we describe our solution: Questioner, a discussion tool that engages users in conversation about an academic topic of their choosing for the purpose of collecting thoughts on a subject and constructing an argument. This system will ask informed questions that prompt further discussion about the topic and provide a discussion report after the conversation has ended. We found that our system is effective in providing users with unique questions and excerpts that are relevant, significant, and engaging. Such a discussion tool can be used by writers building theses, students looking for study tools, and instructors who want to create individualized in-class discussions. Once more data is gathered, efficient and accurate machine learning models can be used to further improve the quality of question and excerpt recommendations. Co-creative discussion tools like Questioner are useful in assisting users in developing critical analyses of written works, helping to maximize human creativity.

Acknowledgements

We would like to thank our advisor, Dr. Yi Fang (Department of Computer Science and Engineering), for helping us with the technical details of our project from brainstorming and research to our final implementation. We would like to acknowledge our reader, Dr. Don Riccomini (Department of English), who has been very influential and engaged in our project, providing constructive feedback on our paper and presentation slides. Finally, Dr. Jackie Hendricks (Department of English), has given us much-needed perspective and information regarding Shakespeare, literary discussion, and analytical writing. Finally, we would like to thank the Santa Clara University School of Engineering and the Santa Clara University Honors Program for supporting this project and providing opportunities to showcase our work.

Table of Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Motivation

Developing a compelling and cohesive thesis for analytical writing can be a daunting task, even for those who have produced many written works. While a writer may have initial opinions about a topic, those opinions take time and focused thought to develop. Additionally, it is often difficult to find engaging study tools for literary works as most online resources only provide summaries and descriptions of events, characters, or settings. Similarly, instructors may want engaging assignments for their students, such as interactive classroom discussions or unique individual writing assignments.

Having an assistant that can elicit and capture a user's informed opinions of a literary source can boost productivity, save time, and keep the writer focused on generating compelling arguments. Such an assistant can also help students learn and instructors teach about a given work of fiction.

While current tools are able to suggest similar terms or topics to a provided term, they are unable to ask pointed questions that prompt the user to supply additional information. Instead, they leave it to the writer to ask themselves relevant questions. Additionally, although some writers prefer to engage in a Socratic seminar or prolonged conversation with a colleague or professor to flesh out opinions, in these conversations one participant's position on a topic may dominate and disrupt the free flow of ideas. Such conversations also lack a physical record, unless participants take time to note results.

For instructors, it is difficult to create individualized classroom assignments for large classrooms. With the variety of responses to literary discussion questions, multiple-choice assignments are too limiting; rather, instructors may want to engage each student in unique discussion experiences to check their understanding.

1.2 Solution

We propose a system that engages users in conversation about an academic topic of their choosing for the purposes of constructing arguments, studying, and evaluating understanding. To keep scope manageable for this project, we have chosen to focus on all thirty-seven of Shakespeare's major plays. This system utilizes a web interface to ask informed questions that prompt further discussion about the topic. The system records the conversation to allow the user to focus on producing creative ideas rather than documenting them. Once the conversation is concluded, the system provides users with a transcript of the conversation.

Our system fills the gap in current solutions by guiding users through the discussion process with a co-creative tool. Our proposed solution enhances the ideation process and engages users in unique literary discussions.

Chapter 2

Requirements

This section outlines the requirements for the system. We successfully implemented the critical requirements and have fulfilled our non-functional requirements. We also listed two recommended requirements which we did not satisfy but will examine for future research.

2.1 Functional Requirements

2.1.1 Critical Requirements

The system:

- Engages users in discussion about Shakespeare by asking users relevant questions.
- Makes a transcript of the conversation available to the user.
- Summarizes the user's responses for quick viewing.

2.1.2 Recommended Requirements

The system will:

- Organize ideas in a logical, hierarchical order.
- Interface with the user via voice.

2.2 Non-Functional Requirements

The system is:

- Fast: New questions are asked within 300 milliseconds of the conclusion of the user's response.
- Secure: User data and transcripts remain private.

Chapter 3

Use Cases

This section describes the different actions that a user can perform.

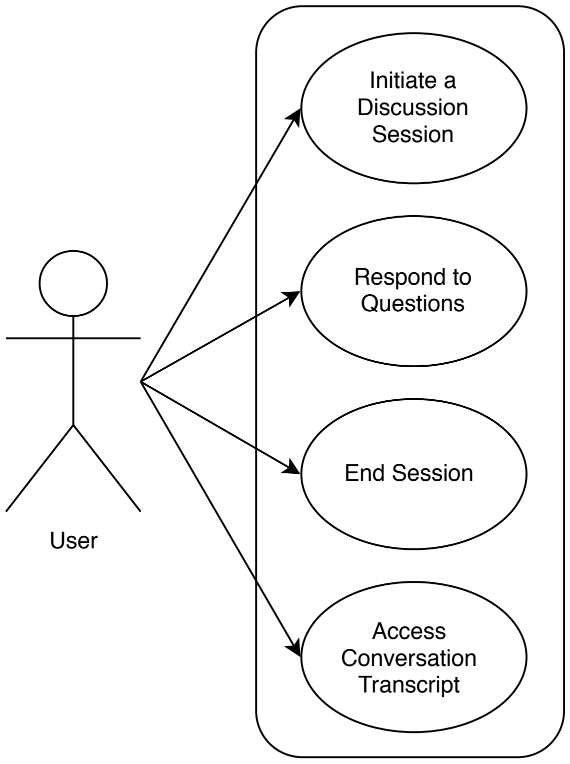


Figure 3.1: Use Case Diagram

Figure 3.1 shows the four actions that a user can take. First, the user can open the application and initiate a discussion session. This involves the user choosing a topic or prompt to discuss with the system. During the discussion, the user responds to questions generated by the system to elicit ideas. The user can elect to end the conversation at any time. After the conversation, the user is able to access and download a transcript of the conversation.

3.1 Study Tool

Students can use this system to check their understanding of a chosen work of Shakespeare. While practicing for our conference presentation demo (see Appendix A, Slides), Shur realized he had quickly relearned many of the important details about Macbeth, which he had last studied in 12th grade.

3.2 Analytical Writing Tool

Writers can use this system to provoke inspiration during the ideation phase of writing. Questions concerning relationships, themes, and more elicit responses that can be easily crafted into theses. Once the discussion session is concluded, writers may view their responses for later reference.

3.3 Class Discussion Tool

Instructors can use this system to lead in-class discussions by drawing from generated questions for conversation starters. Instructors may also assign students to engage in conversation with the system, then have students submit the discussion report to the instructor for evaluation.

Chapter 4

Activity Diagram

In this section we outline a user's typical action flow as they interact with the system.

As shown in Figure 4.1, the user initiates a discussion session with the system and receives a request to provide their topic of discussion or discussion prompt. Using this topic, the system asks a relevant question. The user responds with their answer to the question. The system asks another question based upon the user's response, unless the user elects to end the discussion. One the discussion is finished, the user may view the results of the discussion.

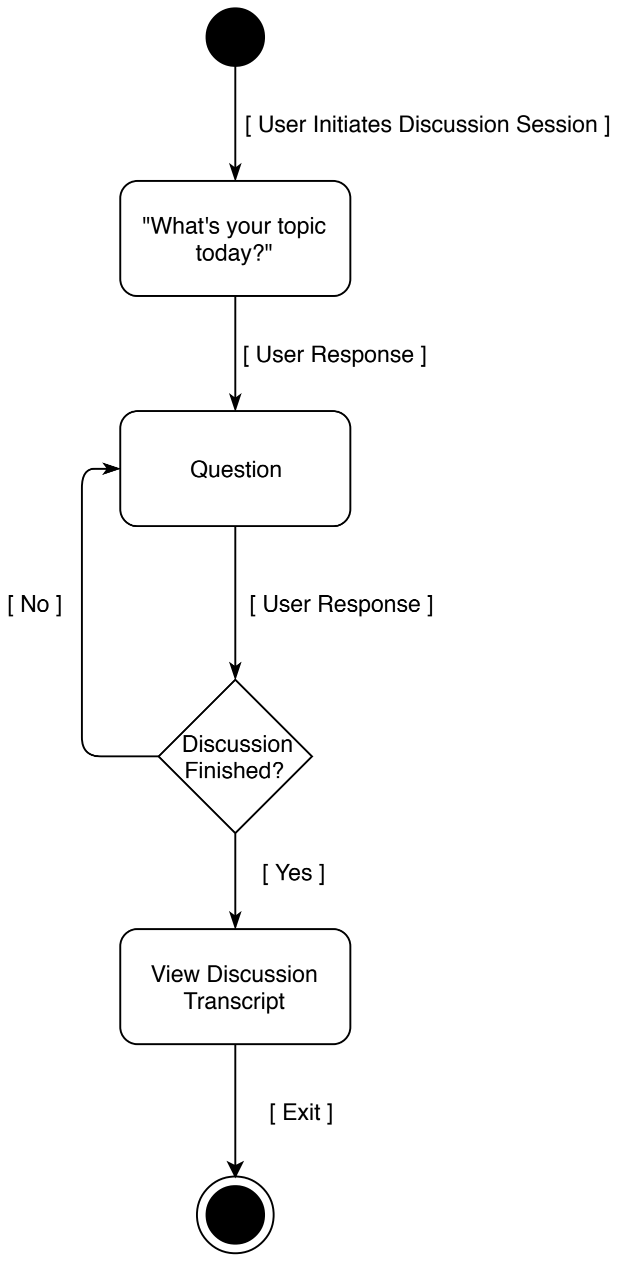

[User Initiates Discussion Session]
"What's your topic today?"
[User Response]
Question
[No]
[User Response]
Discussion Finished?
[Yes]
View Discussion Transcript
[Exit]

Figure 4.1: User Activity Diagram

Chapter 5

Architectural Diagram

In this section, we detail our system architecture.

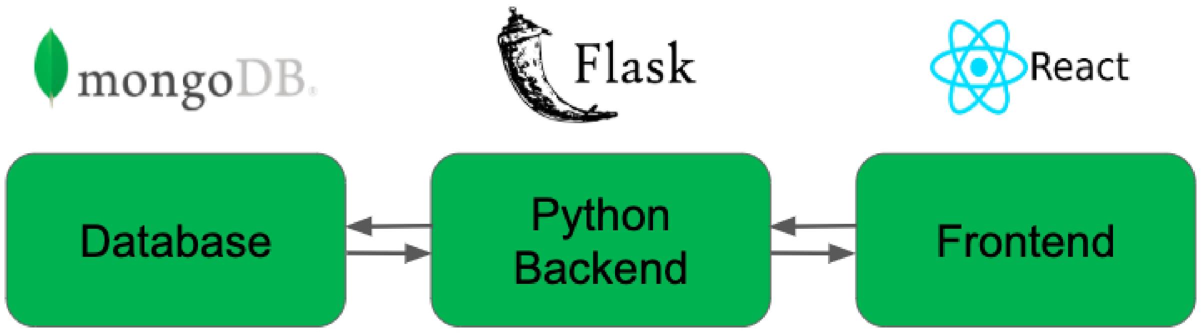


Figure 5.1: Architectural Diagram

Our system uses a client-server architecture, shown in Figure 5.1. Our system is separated into three modules: the MongoDB database, the Flask backend, and the ReactJS frontend. The frontend application runs on the client's machine, the rest of the computation occurs on the backend server, and data is stored in the database. Modules are inter-connected via RESTful APIs. We chose to separate these three modules such that there is loose coupling between modules and high cohesion within each module. This made building and testing each module much simpler as we could test each module independently without worrying about dependencies between modules. Then, after ensuring that our message-passing methods for communicating between modules worked properly, connecting the entire system was simple and relatively bug-free.

The frontend is a single-page application built with ReactJS and handles only the rendering and interactions with the website. For each session, the client creates a unique `sessionId` which is used to identify the session. Once a discussion is in progress, the frontend keeps a local state of the conversation history, saving each question and corresponding user response. The frontend components were designed such that the client can properly render a discussion given any discussion state. Each time the user changes the state by selecting a play or entering a new response, a `POST` request is sent to the Flask backend to save the new session information in the database. Then, a `GET` request is sent to receive the next question to ask the user.

The backend is a Flask server written in Python which runs on the server side and handles all computations and data processing necessary to generate a new question for the user given a discussion state. The server waits to react to requests sent by the client. When the client sends an discussion update to the server, the server connects to the database and updates the session data. When the client requests a new question, the server examines the discussion, generates a new question, and sends it to the client.

Chapter 6

Technologies Used

This section details the technologies we use in our system. The design rationale for each technology is detailed in Chapter 7.

6.1 Frontend Development

- ReactJS
- JSX
- Sass
- JavaScript

6.2 Backend Development

- Python 3
- Flask

6.3 Testing

- Jest / Enzyme
- Postman

6.4 Source Control

- Git

For frontend technologies, all of the user-facing components are built as React components. The formatting of each page uses JSX to mix HTML-like tags with React components and JavaScript logic. For styling our website, we use Sass instead of CSS for the additional features it provides such as nesting, mixins, and variables. To send RESTful API requests to our Flask server, we use the `axios` library for its simple Promise-based style. The `react-to-print` library helps us convert our React components into printable pages. Finally, the `textarea-caret` library helps us replace the default cursor with a new blinking green cursor.

For backend technologies, all of the server-side code was built using Python 3.6. We use the `flask` and module to build a Flask app capable of handling RESTful API requests. To connect to the MongoDB database from Python, we use the `pymongo` module. `beautifulsoup4` was used to do early web scraping to gather the data necessary for our system. Finally, we use the `numpy`, `pandas`, `scikit-learn`, and `gensim` modules to perform our data processing and generate new questions for the user.

Chapter 7

Design Rationale

Our simple system design enabled us to devote most of our time towards developing a high quality questioning agent. By simplifying our architecture and using easy-to-configure technologies, we minimized development time on the system interface to maximize our efforts on the core component: the questioning agent. By doing so, we focused on developing novel approaches and applications in the field of natural language processing, as it pertains to question generation and selection.

To match our simple system design, we used a client-server architecture where the core algorithms are processed on the server and the user interface is rendered on the client side. Since there are no complex system interactions, this simple architecture design helps us effectively and efficiently fulfill our requirements.

In order to build this system, we used ReactJS to create a basic web application. This is a popular, modern technology that controls the view layer and enabled us to build a reactive single-page-application. On the server side, we used Python 3. Python 3 is known for its quick development time and large number of libraries, especially in the fields of natural language processing and machine learning. We leveraged these advantages to build our system quickly and effectively.

We also utilized a few testing libraries to streamline the testing of our application. For the frontend, we used Jest and Enzyme to test our React components. For the backend and internal API, we used Postman to simulate REST API requests and verify the returned data. More information on testing is located in our test plan in Chapter 10.

Finally, we used Git for source control to ensure that we were able to coordinate our work and avoid data loss.

Chapter 8

User Interface

This section outlines the user interface for our system. The system has a basic user flow and minimalist interface so the user will not be distracted. First, the user is greeted by a home page with a drop-down menu to choose a Shakespeare play they would like to discuss. Upon choosing a play, they are prompted to begin a discussion session, as shown in Figure 8.1.

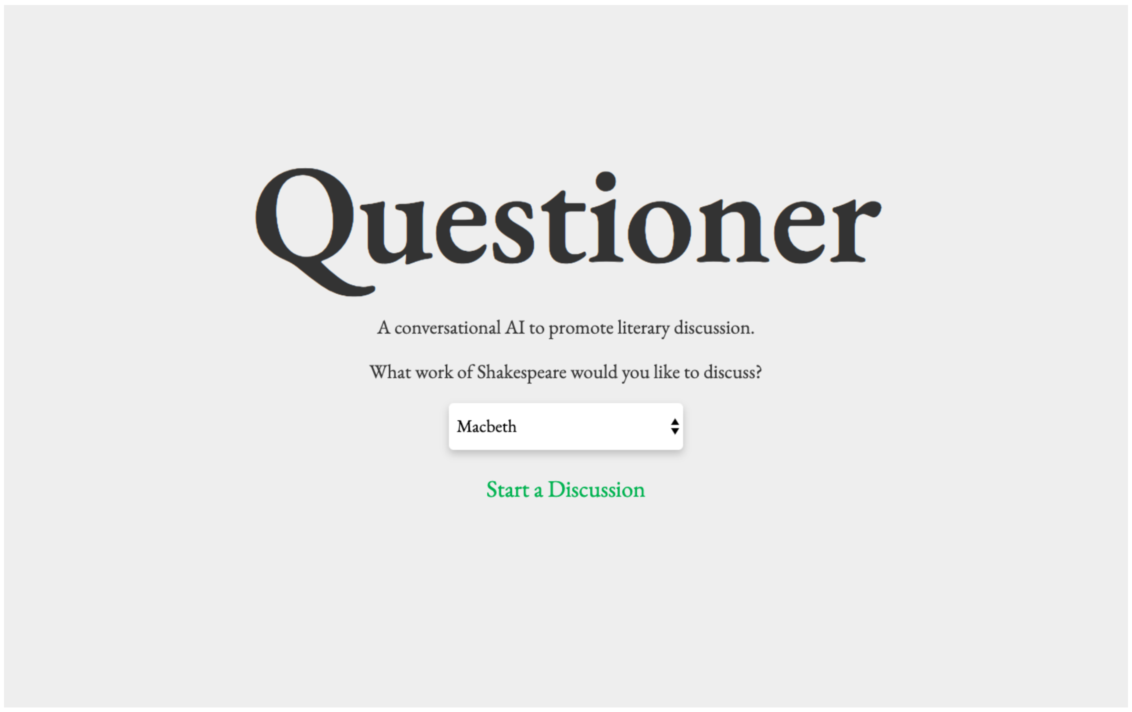
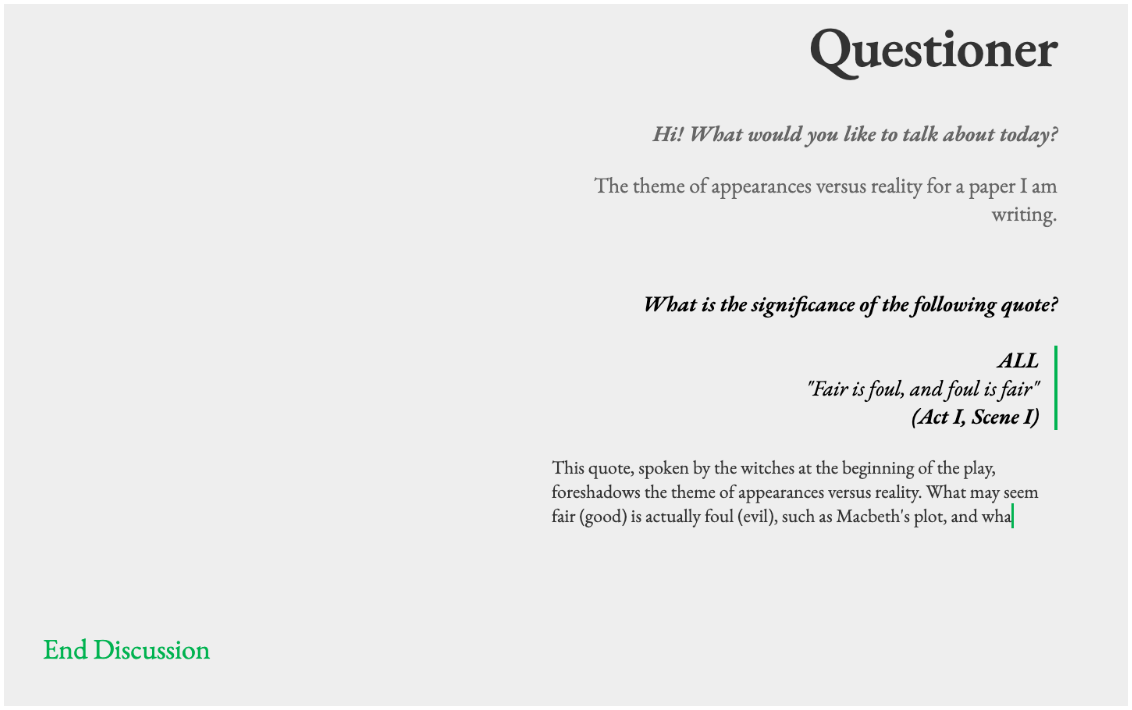

Questioner
A conversational AI to promote literary discussion.
What work of Shakespeare would you like to discuss?
Macbeth
Start a Discussion

Figure 8.1: Home Page

Once the user begins a discussion session, they can enter their prompt or topic and begin their discussion. The system displays a question, a cursor that indicates a response should be entered, and a button to end the conversation. The question is generated by our system with the intention to provoke ideas and encourage deep discussion on the chosen topic. Our system can also present an excerpt to the user and ask them to describe its significance. After each user response, the system will give the user another question or excerpt. An example in-progress discussion is shown in 8.2.


Questioner

Hi! What would you like to talk about today?

The theme of appearances versus reality for a paper I am writing.

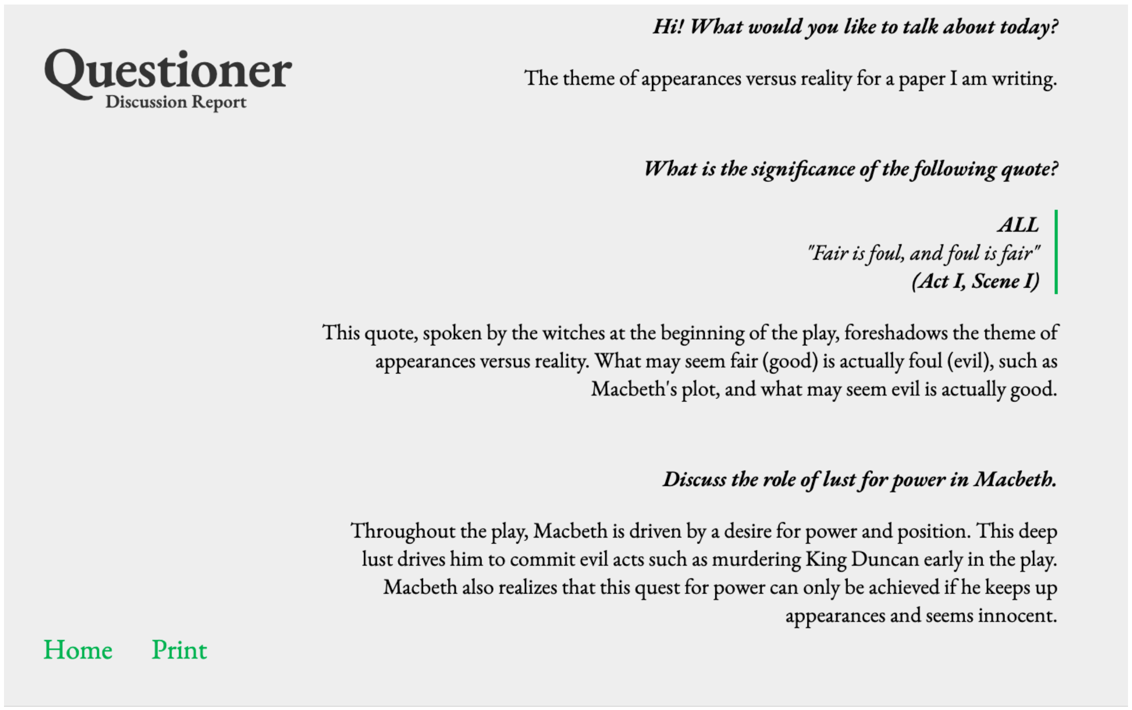
What is the significance of the following quote?

> ***ALL***
> *"Fair is foul, and foul is fair"*
> ***(Act I, Scene I)***

This quote, spoken by the witches at the beginning of the play, foreshadows the theme of appearances versus reality. What may seem fair (good) is actually foul (evil), such as Macbeth's plot, and wha

Figure 8.2: Example Discussion

After the discussion, the user can examine a discussion report where they can view past questions and their responses. An example discussion report is shown in 8.3.



Hi! What would you like to talk about today?

The theme of appearances versus reality for a paper I am writing.

What is the significance of the following quote?

> ***ALL***
> *"Fair is foul, and foul is fair"*
> ***(Act I, Scene I)***

This quote, spoken by the witches at the beginning of the play, foreshadows the theme of appearances versus reality. What may seem fair (good) is actually foul (evil), such as Macbeth's plot, and what may seem evil is actually good.

Discuss the role of lust for power in Macbeth.

Throughout the play, Macbeth is driven by a desire for power and position. This deep lust drives him to commit evil acts such as murdering King Duncan early in the play. Macbeth also realizes that this quest for power can only be achieved if he keeps up appearances and seems innocent.

Figure 8.3: Example Discussion Report

Chapter 9

Backend

In this section we detail the Python backend mechanism that provides questions for the frontend to display.

9.1 Excerpt Recommendation Model

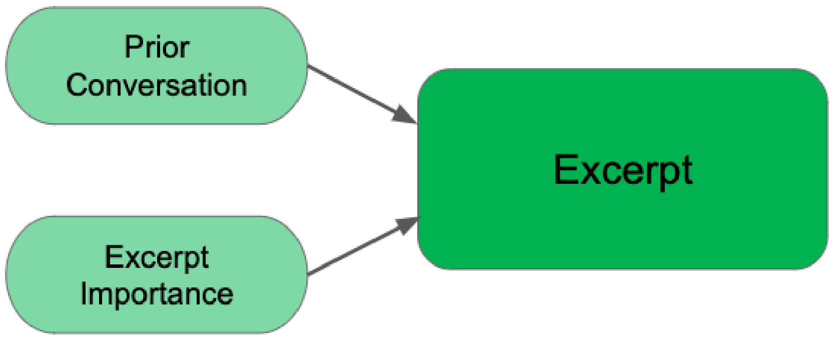


Figure 9.1: Excerpt Recommendation Model Flowchart

Shown in Figure 9.1, the excerpt recommendation model takes two factors into account when selecting from all possible excerpts from the user's chosen work. Excerpt importance is calculated based on the excerpt's occurrence in search results. We use this method of estimating importance because more significant excerpts are more likely to be cited in academic papers and online discussion. A subsequent iteration of the system will compare text embeddings of excerpts with the embeddings of prior user conversation to choose the most relevant and significant excerpts.

9.2 Character Template Recommendation Model

Shown in Figure 9.2, the character recommendation model works in two phases. First, the anchor character is chosen utilizing character importance, which is computed by analyzing how many lines that character has spoken in the chosen work. Prior conversation embeddings will also be used for this choice. For the second phase, the supporting character is chosen not only with character importance, but also with the number of scenes shared with the anchor character. These

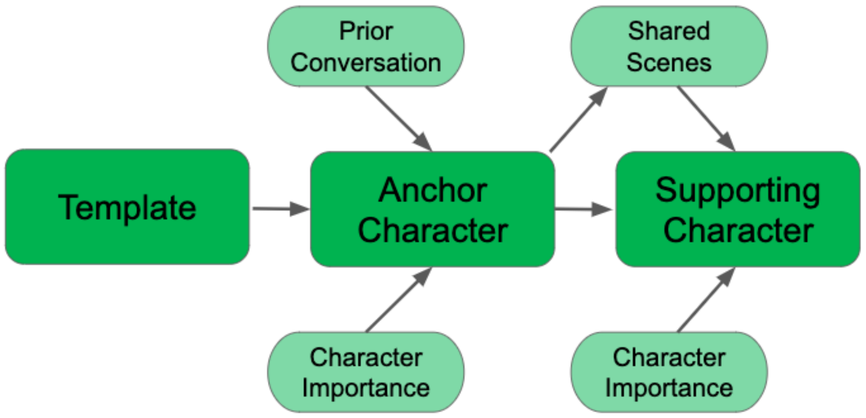


Figure 9.2: Character Recommendation Model Flowchart

shared scenes are drawn from a character co-occurrence matrix that was constructed from the original Shakespeare corpus.

9.3 Miscellaneous Template Recommendation Model

We also include a template model that asks questions regarding settings, themes, and motives that draws from numerous data sources. This model works similarly to the Character Template Recommendation Model described in Section 9.2, except that it operates on settings, themes, and motives rather than characters. This model uses similar prior conversation and importance metrics to choose the setting, theme, or motive.

Chapter 10

Test Plan

In this section, we detail our test plan for ensuring that our system is bug-free and meets our specified requirements. To test our system, we used the following testing strategies:

- Individually test functions and modules
- Test integration between frontend and backend modules
- Test system end-to-end to ensure smooth operation
- Test end-user operation

10.1 Frontend

For the frontend, we tested our application using Jest, a popular testing module in ReactJS. Jest enabled us to not only unit-test individual functions and components, but also perform snapshot tests that ensure that the system renders components correctly in our test cases. The Enzyme library helped make our tests more efficient and modular by only loading a lightweight version of each component which we could test using Jest.

10.2 Backend

For the backend, we tested our Python code using Postman. Using Postman, we were able to test our internal REST APIs in our Flask application by sending test API requests and verifying that we got the correct response. This enabled us to test individual functions and modules and ensure that our code was bug-free.

Chapter 11

Results

As mentioned in Chapter 10, Test Plan, we asked our peers to engage with our system. We used initial reactions to guide changes to the frontend UI and question generation process. Subsequent user testing yielded very positive reactions.

Our system received a very positive response from the audience at the Santa Clara University Senior Design Conference in May 2019. We also presented our project at the ”Analog/Digital: Premodern Technology Meets Silicon Valley” event in May 2019 at Santa Clara University. This event attracts many scholars of literature, including professors and researchers working with Shakespeare. At this event, our project received positive feedback, including requests to use our application in university classrooms for literary discussion.

Overall, the combination of relevant generated questions, an intuitive and minimalist user interface, and low system latency resulted in a product that satisfied our requirements and received positive reviews from test users.

Chapter 12

Risk Analysis

In this section, we detail several potential risks, their consequences, and the impact of each. For each risk, we provided several mitigation strategies to ensure that our project met the requirements and was completed on time.

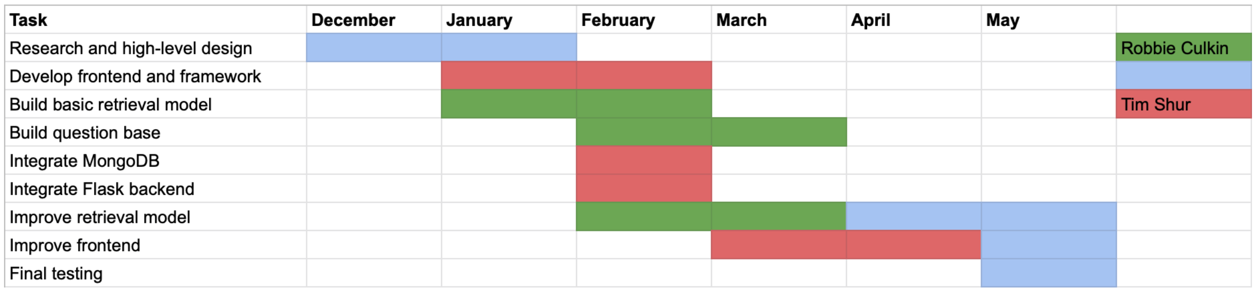
Table 12.1: Risk Analysis

Risk	Consequences	Probability	Severity	Impact	Mitigation
Time	System not fully completed on time	0.5	8	4.0	Prioritize tasks and develop new features incrementally Set deadlines
Bugs	System won't work, poor UX, takes longer to complete	0.9	4	3.6	Test functions and modules Write clean and modular code
Data Shortage	Algorithms are inaccurate, must explore other methods	0.6	6	3.6	Search for data sets early Explore low-data solutions
Insufficient Technologies	Cannot satisfy requirements, need to change technologies	0.3	6	1.8	Plan and justify design Research alternatives
Group member absent	Loss of productive time, takes longer to complete	0.35	5	1.75	Eat healthy, sleep well, be proactive and communicative Work from home if necessary

Chapter 13

Development Timeline

The Gantt chart in Figure 13.1 outlines the development timeline of our implemented solution. Each section is broken up into separate sub-tasks. The progress of each task is tracked by a date range, a percentage of completion bar, and a time-line to visualize each task over time.



Task	December	January	February	March	April	May	
Research and high-level design	🟦	🟦					🟩 Robbie Culkin
Develop frontend and framework		🟥	🟥				🟦
Build basic retrieval model		🟩	🟩				🟥 Tim Shur
Build question base			🟩	🟩			
Integrate MongoDB			🟥				
Integrate Flask backend			🟥				
Improve retrieval model			🟩	🟩	🟦	🟦	
Improve frontend				🟥	🟥	🟦	
Final testing						🟦	

Figure 13.1: Gantt Chart Development Timeline

Chapter 14

Societal Issues

In this chapter, we examine how this project raises several societal issues as any project or engineering application affects and is affected by society.

14.1 Ethical

An ethical concern related to our project is the possibility of the automation of the instructive process. We feel, however, that this tool can augment the abilities of educators rather than replace them. Our tool reduces the difficulty of collecting and organizing data, instead allowing the user to focus on core analysis in their responses.

14.2 Social

Regarding social issues, our project may have a negative impact. By using our discussion tool, users are interacting with a website rather than discussing literary questions with other people. This may lead to an unintended consequence of users becoming accustomed to engaging an AI in discussion, impacting the social skills necessary to engage in discussion with their peers. However, our tool can also be used to better prepare students for engaging in-class discussions with pre-class discussion assignments.

14.3 Political

At a glance, our project has little political impact because it does not directly engage with the public realm. However, our project helps educate our users and encourage them to become more critical thinkers, thereby enhancing their ability to become critical thinkers in the political sphere.

14.4 Economic

We did not borrow any money for the initial stages of our project because we provided free labor and did not host our website on any costly platforms. However, the next stage of our project might require hiring paid engineers and investing in a cloud hosting solution for our website which will generate cost. As this project develops, we may be able to recuperate the costs of production with generated revenue from our service.

14.5 Health and Safety

Our system provides no major health or safety risks besides the inherent risks of using a computer.

14.6 Manufacturability

Since our project has no physical or manufacturable pieces, our project does not directly raise any issues of manufacturability. On the topic of development, the main problems may arise in the difficulty of hosting a web application with a heavy-weight machine learning backend.

14.7 Sustainability

This project has little engagement with the environment or the world's resources, and therefore, has little impact on broader issues of sustainability. In a more narrow sense, the product can continue to be viable and useful for a long time, as literary analysis and critical discussion will continue to be important for students, instructors, writers, and other people.

14.8 Environmental Impact

Our website has negligible environmental impact and does not raise any significant environmental issues beyond the energy consumption this website requires. With a machine learning model with significant more computation than simple, static webpages, our website will use more energy than most other websites.

14.9 Usability

Our product, with its intuitive design and simple, minimalist style, provides users with an interface that allows them to focus on sharing their thoughts and ideas.

14.10 Lifelong learning

One of this project's greatest impacts is its contribution to the societal issue of lifelong learning. Throughout the development of this project, we constantly had to learn new tools and techniques such as natural language processing, ReactJS, MongoDB, and how to write an internal API to communicate between different parts of our system. Moreover, our solution itself encourages people to keep learning, analyzing, and discovering new ideas. Questioner seeks to constantly engage users in literary discussion and challenge them to think about new questions and concepts. Thus, our project not only helped us continue to learn, but it will help our users keep learning and improving their literary skills.

14.11 Compassion

While our project does not deal with direct human suffering, this discussion tool seeks to ease the difficulty of literary discussion and analytical writing. It can be difficult for people to find ways to uncover their ideas and find quotes to support their thoughts. Our tool provides a solution to this need.

Chapter 15

Conclusion

We built Questioner, an online discussion tool that is useful for writers, students, and instructors alike. We engage users in literary discussion, seeking to assist people in developing critical analyses of written works. More broadly, this tool helps maximize human creativity and expand research in natural language understanding.

15.1 Experiences, Lessons Learned, Challenges

The biggest lesson learned when building this project was how to design and implement a year-long software application. We started with a concept, refined and scoped our idea, designed a system architecture, implemented our design, tested our implementation, and then redesigned and refined our system. Among our most valuable lessons were how to properly scope a project to fit with time constraints, how to choose a system architecture that effectively accomplishes our goals, and how to work with user feedback to refine our product. Moreover, we needed to learn some of the technical components of our project such as ReactJS, Flask, and MongoDB. We ran into several challenges which complicated our ability to achieve our recommended requirements:

- Because the machine learning model had to take prior conversation into account, it required high space and time complexity, making it difficult to provide users significant questions in a short amount of time.
- We had trouble finding good data sets with literary questions and responses to create accurate question-generation machine learning models. Therefore, we decided to create an initial system without this feature, and then implement it later once we have gathered enough user data.
- When attempting to implement a voice interface, we realized that speech-to-text programs are not as accurate as we would like, converting speech in a very unrefined state. Instead, we opted for a standard written entry and response model.

Moreover, we initially built this project hoping to solve some sort of problem with machine learning. However, we later learned that machine learning was not the most effective solution to our problem and did not provide the best questions and excerpts without sufficient data. Therefore, we came to the realization that we should not start with the technology and then look for a problem it can solve; rather, we must always start with the problem and find which technology solves it best.

15.2 Suggested Changes

We plan on continuing to improve this project in the following ways:

- Host the website on a cloud service to gather user data
- Leverage prior discussion with text embeddings
- Organize discussion report in an outline format
- Allow the user to interface via voice

Currently, our website is only available when locally hosted on our personal computers. If we host our website live on the cloud, we can find writers, students, and instructors to experiment with the system and give us more feedback. Furthermore, from these real users we will be able to gather more data to improve our system.

Secondly, our system does not yet take prior user responses into account when generating the next question. If we embed the user responses and questions with text embeddings (using the `gensim` module), we can compare generated questions with prior discussion to determine which question aligns best with the user's train of thought.

For simplicity, our discussion reports are raw transcripts of the discussion. However, future research can be done to add value to the discussion reports by organizing ideas into an outline format. This can be done by summarizing user responses and grouping similar ideas together. Moreover, broad ideas can be separated from supporting details and a hierarchical organization can be constructed.

Finally, more research can be done to integrate a voice interface into our system. Then, users would be able to respond to questions naturally via speech and actually start a conversation with our website. We did not pursue this feature because we foresaw issues with speech being too unrefined and speech-to-text programs being too inaccurate.

15.3 Broader Implications

Our project has broader implications outside the works of Shakespeare. This project can be generalized to other works of fiction as well, using the same information retrieval (IR) methods leveraged for the Shakespeare corpus. This project also serves as an excellent testbench for the application of natural language understanding not only to literature, but to user input. Further exploration of fictional literary works by the IR community will yield benefits for IR and literary circles alike. Lastly, we believe that tools like ours can be used to augment human creativity for important analyses.

Appendix A

Presentation Slides

In this section, we include the presentation slides we used in the Santa Clara University Senior Design Conference in May 2019.

A Questioning Agent for Literary Discussion

Robbie Culkin & Tim Shur

Advisor: Dr. Yi Fang

Problem

Motivation, Solution, Requirements

Motivation

- Literary study and analytical writing can be challenging
- A limited number of discussion questions are available online
- Instructors want to provide their students with engaging learning opportunities

Solution

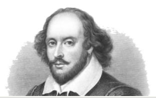
- Web-based discussion tool
- Questions and excerpts to:
  - Facilitate discussion
  - Check understanding
  - Provide inspiration for analytical writing
- Intuitive discussion report

Target Audience

- Writers
  - Develop arguments about a work
- Students
  - Check and develop understanding of a work
- Instructors
  - Evaluate student understanding with unique and engaging assignments
  - Lead in-class discussion

Focus on Shakespeare

- Relevant to many users
  - Required reading for many US high school students
- Well-structured
  - Act, Scene, Line, Speaker
- Public domain
- Lots of scholarly discussion



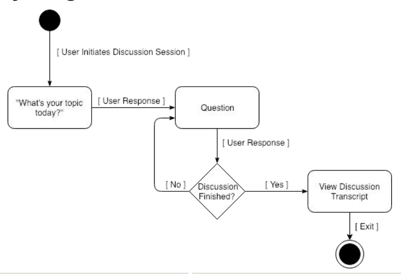
Requirements

- Functional requirements
  - Web application
  - Pose thought-provoking questions and excerpts
  - Generate a discussion report
- Nonfunctional requirements
  - Intuitive
  - Fast
  - Secure

System Design

Activity Diagram, Architectural Diagram, Technologies Used

Activity Diagram


[User Initiates Discussion Session]
"What's your topic today?"
[User Response]
Question
[User Response]
[No]
Discussion Finished?
[Yes]
View Discussion Transcript
[Exit]

Architectural Diagram

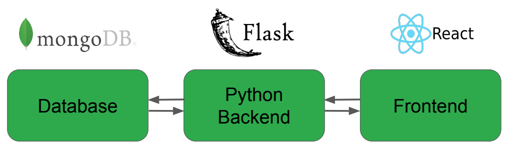


Figure 2: Architectural diagram

Technologies Used

- Frontend tools
  - ReactJS
- Backend tools
  - Flask
  - Python 3
- Database
  - MongoDB
- Source control
  - Git

Testing Methodology

- Unit test individual modules
  - Frontend
  - Questioning agent
  - Report generation
- Test integration between frontend and backend
- Test system with real users
  - Ensure intuitive UI and conversation flow

Implementation Details

Character Template Recommendation Model, Excerpt Recommendation Model

Template Questions

- Format
  - Discuss [character 1]'s relationship to [character 2].
  - What role does [theatrical element] serve in [play]?
- Blanks filled using retrieval/ranking model

Compare and contrast the characters of Hamlet and King Claudius. How alike or dislike are they and why?

Figure 3: Example template question

Character Template Recommendation Model

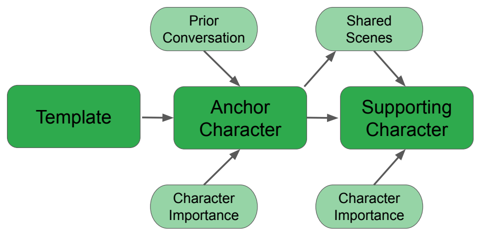


Figure 4: Template recommendation model

Anchor Character

- Given a list of characters and a play, choose with probability distribution

$$P(\text{character}|\text{play}) = \frac{|L_c|}{|L|};$$

$L = $ set of lines in play

$L_c = $ set of lines spoken by character

- Next step: Add metric for similarity to user input with text embeddings

Supporting Character

- Given a list of characters, a play, and an "anchor" character, choose with probability distribution

$$P(\text{character}|\text{play, anchor character}) = (1-\alpha)\frac{|L_c|}{|L|} + \alpha\frac{|S_c \bigcap S_a|}{|S_a|}$$

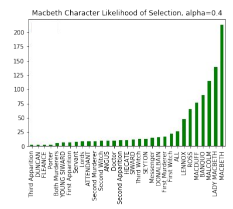
$L = $ set of lines in play

$L_c = $ set of lines spoken by character

$S_c = $ set of scenes where character speaks

$S_a = $ set of scenes where anchor character speaks

Supporting Character: Alpha Selection



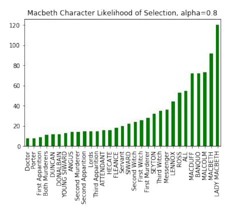
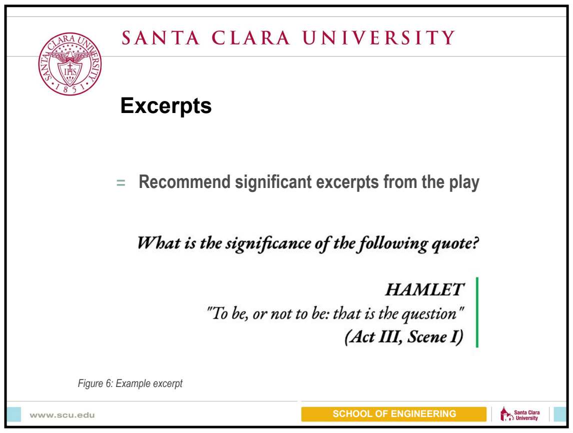


Figure 5: Varying alpha for character selection



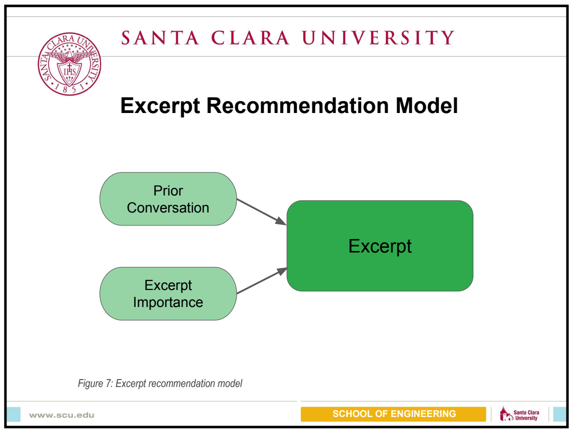
Excerpts

= Recommend significant excerpts from the play

What is the significance of the following quote?

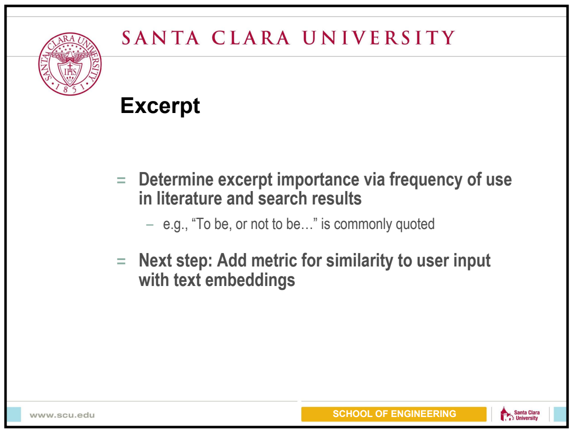
HAMLET
"To be, or not to be: that is the question"
(Act III, Scene I)

Figure 6: Example excerpt



Excerpt Recommendation Model

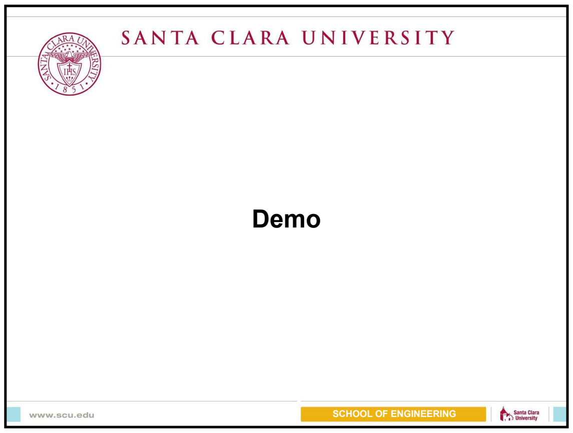
Figure 7: Excerpt recommendation model



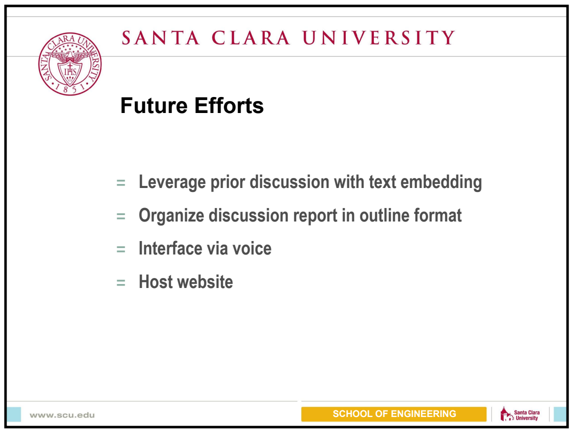
Excerpt

= Determine excerpt importance via frequency of use in literature and search results
  – e.g., “To be, or not to be…” is commonly quoted

= Next step: Add metric for similarity to user input with text embeddings



Demo



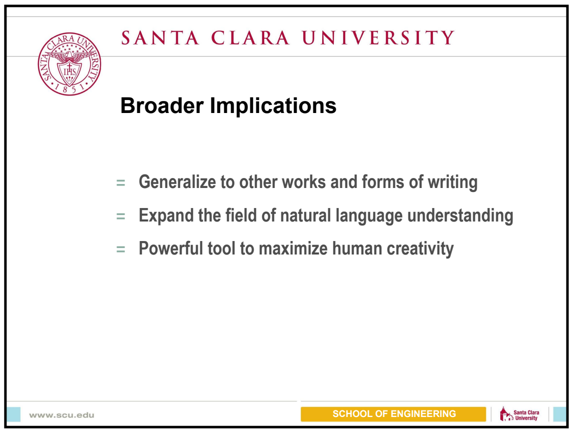
Future Efforts

= Leverage prior discussion with text embedding

= Organize discussion report in outline format

= Interface via voice

= Host website



Broader Implications

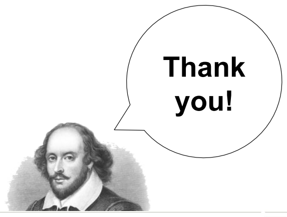
= Generalize to other works and forms of writing

= Expand the field of natural language understanding

= Powerful tool to maximize human creativity

Acknowledgements

- Dr. Yi Fang
- Dr. Don Riccomini
- Dr. Jackie Hendricks



Recommendations Based on Prior Conversation

- User Text Embedding
  - Transform text to vector
  - Text: "The priest played a pivotal role in Romeo & Juliet's demise"
  - Vector: [4.56, -6.03, …, 0.72]
  - Compare previous conversation with questions/excerpts to look for similarity

Data Sources

- Corpus of original Shakespeare plays
- Modern English translations of Shakespeare plays
- Online Discussion Questions
- Online Study Resources
  - Genre, setting, themes

Challenges

- Heavyweight document vector model
- Hosting a website with machine learning
- Shortage of user data

Use Case Diagram

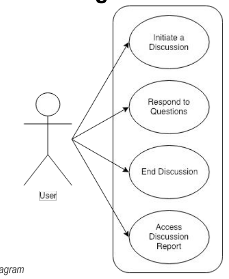


Figure 8: Use case diagram

Appendix B

User Manual

This section describes instructions for installing the application and details for the internal API.

B.1 Setup and Installation

This repository uses `mongodb` for the database, `flask` for running Python (3.6) on the backend, and `nodejs` with `reactjs` for the frontend.

B.1.1 MongoDB

To run a database locally, you will need to install `mongodb` on your system. Make sure you have a data directory to store database documents. To create a folder in the default location, you can run the following:

Listing B.1: Creating a MongoDB Data Directory

```
mkdir -p /data/db/
chmod 777 /data/db/
```

Now you should be able to start a `mongodb` instance by running the `mongod` command. By default, this starts the `mongodb` instance with the URI at `mongodb://localhost:27017`.

B.1.2 Flask

Next, we need to spin up a Flask instance. Create a `virtualenv` at `backend/env` and install the `requirements.txt` with the following:

Listing B.2: Installing Python3 Libraries

```
cd backend
pip3 install virtualenv
python3 -m virtualenv env
source env/bin/activate
pip3 install -r requirements.txt
deactivate
```

Finally, run the script to start Flask: `./bin/run_flask.sh`. The flask server can be reached by default with API routes at <`http://localhost:5000/api/v0/`>

B.1.3 React

To get React started, you will need to have `nodejs` installed. Then, run the following to install the required packages and run the frontend in development mode:

Listing B.3: Starting a React Server

```
npm install
npm start
```

This will open the web application in a new tab at <`http://localhost:3000/`>.

B.2 Launching the Application

After performing all steps detailed in Section B.1, you can launch all parts of the applications at once with the script `./bin/run_all.sh`. This script will run the MongoDB instance, the React server, and the Flask app all at once with debug logging in the `log/` directory.

B.3 Flask API

The Flask application has the following API routes:

`GET - /api/v0/questions`

This route requires a valid `sessionId` and will return the next question for the session based on the discussion context such as:

```
{
    "question": "What role does humor play in Hamlet?"
}
```

`GET - /api/v0/report`

This route requires a valid `sessionId` and will return a report of the discussion in the form:

```
{
    "session": {
        "sessionId": "16c209bc-42ca-4c22-9e80-172c1cf1cd51",
        "discussion": [
            {
                "msgId": "fc9e81ac-100b-45e2-aae0-393c65f500d8",
                "fromUser": false,
                "text": "How are you?"
            },
            {
                "msgId": "c22b3534-4c62-4aac-837b-45b88807ad7e",
                "fromUser": true,
                "text": "I'm doing well!"
            }
        ]
    }
}
```

`POST - /api/v0/response`

This route sends a session object of the form above to be inserted into the database. If the `sessionId` provided does not exist, a new document will be made in the given format. Otherwise, the `discussion` field is extended by the messages that are given in the body of the POST request.

Appendix C

Source Code and Data

Our entire project and all of the data that we used has been open sourced for transparency and visibility. The GitHub repository is called `questioner`, located under the username `robbieculkin`, and can be found at the following URL: `https://github.com/robbieculkin/questioner`.