GrocerBot:

Cameron O’Dell (16334423)
Daniel Sponsler (12559213)
Reed White (12367576)

Project Proposal

This project aims to develop a simple, portable, customizable chat system using retrieval augmented generation (RAG). This system will be kept as light as possible, to be able to run on a variety of user hardware with a variety of large-language models (LLM), both local and remote. Our goal is to keep the entire system, including the loading of the model, the creation of the vector database, the RAG retrieval, the model inference, and the end user interface contained in a single Python file.

Users

The resulting product will be aimed at individuals or smaller business entities that wish to utilize the power of RAG in smaller settings and with smaller corpuses. Our target users are put off by the existing RAG solutions because they require too much technical experience to work with, or are too demanding in terms of resources. Our project will address these needs by providing a simple, intuitively designed product which can run on less powerful hardware.

Scenarios

- Trinkets Incorporated sells trinkets on Amazon. It has a website running on a few servers in AWS. Its leadership decides it wants to create a small chatbot to help their customers decide what sort of trinket they should buy, but they don’t sell enough trinkets to warrant subscribing to any full-service AWS chatbot offerings. Trinket decides to install GrocerBot on one of their AWS servers, loads a list of their current offerings, and exposes the GrocerBot to their end users via their AWS gateway.
- Bob owns a small business selling doodads to his local community. Bob has a decent level of technical ability, and is currently hosting his website from a small server sitting in his closet. Bob wants to offer basic doodad support to his users with a chatbot. So, Bob installs the GrocerBot chatbot on this server, points it to a small set of support instructions he created, and puts a link to his new bot in his website.

Needs

This project believes that our target users have the following needs:

- Simplicity: current chatbot offerings are complicated. More users need access to simpler systems that are easier to implement.
- Lightness: current chatbot offerings require massive local LLMs and datastores (that many users cannot run on their existing hardware) or remote LLM API calls (that many users cannot afford). More users need chatbot systems that can run on smaller hardware.
- Customizability: while many current chatbot systems offer customizability, they often require deep technical knowledge and hardware to implement. More users need the ability to quickly and easily integrate their system data into a chatbot.

Requirements

Features

This project will focus on delivering the following features:

- RAG system: this system will use Retrieva to generate text content, in the form of a chatbot.
  - Priority: Required, 1
  - Feasibility: High
- RAG UI: this system will include a webapp user interface (UI) to interact with the RAG.
  - Priority: Required, 2
  - Feasibility: High
- Customizable corpus: this system will allow users to define and import their own corpus
  - Priority: Required, 3
  - Feasibility: High
- Customizable model: this system will allow users to define and utilize their own LLMs
  - Priority: Optional, 5
  - Feasibility: Medium
- Customizable data store: this system will allow users to utilize their own datastore
  - Priority: Optional, 6
  - Feasibility: Medium
- Lightweight local installs: this system will allows users to install and run the webapp on a variety of Windows or Linux servers with minimal effort
  - Priority: Required, 4
  - Feasibility: High
- Platform integration: this system will be integrable with popular website hosting services
  - Priority: Optional, 7
  - Feasibility: Low

Product Type

This product is intended to be an installable web application for small businesses or individuals. It should be installed into Windows or Linux servers and will provide a webapp UI that can be exposed to the customers of those small businesses or individuals, to provide a simple, lightweight chatbot system. A version may be developed that can be installed as a plugin to popular SaaS hosting services.

System Architecture

Upon installation into a servers, this product will have the following components, shown in the below diagram:

- End User: a customer/client of the small business or individual that installed this system. Interacts with the UI, usually in the form of a question expecting a response.
- UI: provides the webapp UI. Receives end user prompts, sends prompts and retrieves content from the RAG system, sends prompts/content to and receives content from the LLM, and displays results to the end user.
- RAG: provides the RAG system. Receives prompts from the UI, queries the vector store for similar results, and sends retrieved content back to the UI.
- Vector Store: provides the vector store. Stores the corpus, receives queries from the RAG system, and sends similar results in the corpus back to the RAG system.
- Generative Model: provides the LLM. Receives prompts and retrieved content from the UI, generates a reply based on those details, and sends the generated response back to the UI.

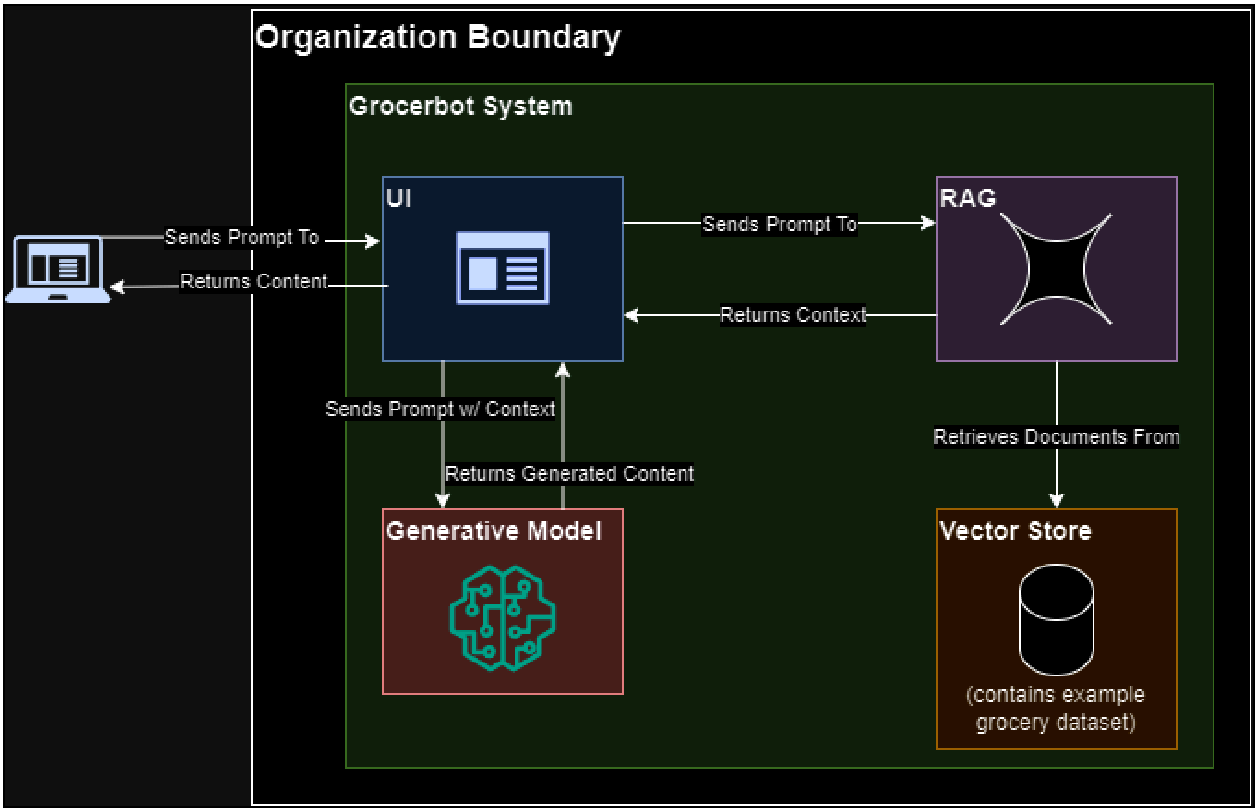

FIGURE I. Expected High-Level Architecture

Comparative Analysis

The following chatbot systems were investigated and their features compared with the proposed system:

System	Model	Language	Opensource	Gui	Local/Remote	Multimodal
GrocerBot	Any	Python	Yes	Yes	Both	No
Dialogflow	Gemini	Multiple	No	Yes	Remote	Yes
Lex	Claude	N/A - UI	No	Yes	Remote	Yes
Botpress	Any	JavaScript	Yes	Yes	Remote	No
Botkit	Any	JavaScript	Yes	No	Local	No

Dialogflow, Lex, and Botpress are all extremely impressive systems, but they require subscriptions and cannot be run on local hardware. Botkit is the most directly comparable system, but it is complex and does not include an integrated GUI.

Cost Analysis

Costs are expected to be quite minimal (near zero) for the scope of the project in this course. However, if the project scope were to be expanded to a fully functional application used across multiple businesses here is an example of the cost of scaling.

Labor (document/development time) $1900 per individual, considering an average data scientist salary of ~$120,000 per year (~$58 per hour) and assuming roughly four hours of development time per week for eight weeks. This results in roughly $5500 for three individuals.

GPUS (Nvidia A100) $8,000+ per unit, however expansion could be done with lower compute and cost GPUs. GPUs could also be rented for less up front cost per unit of usage.

CPUS (Intel core i9-13900k) $400+ per unit, multithreading capabilities would be used for non-AI processes

Storage (SSD) $100-200 per unit for 1TB+ however storage could be rented at a cheaper (monthly) rate through storage services. A combination of physical and cloud storage space would likely be used in the expansion process of scaling up.

Electricity costs – running the operations locally and not in a cloud environment will consume electricity. Researchers at HuggingFace did a study across 88 models (Power Hungry Processing: ⚡ Watts ⚡ Driving the Cost of AI Deployment? (arxiv.org)) “Most tasks they tested use a small amount of energy, like 0.002 kWh to classify written samples and 0.047 kWh to generate text. If we use our hour of Netflix streaming as a comparison, these are equivalent to the energy consumed watching nine seconds or 3.5 minutes, respectively. (Remember: that’s the cost to perform each task 1,000 times.) The figures were notably larger for image-generation models, which used on average 2.907 kWh per 1,000 inferences.”

The average price of electricity in the US is 15.45 cents per kWh. So, while the electricity costs associated would be low, it would be something to pay attention to when factoring in scaling up the business model.

Development Timeline

Task Name	**Week 7 (Sep 30 - Oct 6)**	**Week 8 (Oct 7 - Oct 13)**	**Week 9 (Oct 14 - Oct 20)**	**Week 10 (Oct 21 - Oct 27)**	**Week 11 (Oct 28 - Nov 3)**	**Week 12 (Nov 4 - Nov 10)**	**Week 13 (Nov 11 - Nov 17)**	**Week 14 (Nov 18 - Nov 24)**	**Week 15 (Nov 25 - Dec 2)**	**Deliverables**
Project Planning	🟥	🟥								Project Proposal/Midterm Report
RAG Development		🟦	🟦							RAG Code
GUI Development			🟩							GUI Code
Alpha Testing				🟪						Alpha Test Results
Product Refactoring					🟦					Minival Viable Product (0.1)
Beta Testing						🟪				Beta Test Results/User Feedback
Product Feedback Integration						🟥	🟥			Updated Product (0.5)
Documentation							🟧	🟧		Product Documentation
Final Refactoring								🟥	🟥	Final Release (1.0)

https://docs.google.com/spreadsheets/d/1Uqt0qnTNt5lj2g_YLy7X2Q4pg_zlFO0Z/edit?usp=sharing&ouid=103921787132313457695&rtpof=true&sd=true

GitHub Repository

This project will be kept in the following repository:

https://github.com/mist861/5542-Big-Data-Analytics

Versioning will be controlled with branches and pull requests. Development will be performed in Google Colab or local Jupyter notebooks. No code commits will be performed directly to the main branch.

Challenges and Risks

Challenges:

- It may be difficult to integrate the RAG with the UI.
- It may be difficult to ensure the product is easily installable on a broad range of machines that potential customers may be using.
- We may encounter difficulty ensuring that our product is more simple and intuitive to use than existing alternatives.
- We may have difficulty providing the desired customizability without undermining the simplicity of our design.
- We may have problems identifying satisfactory models for the LLM and vector store.

Mitigation Strategies:

- We can make use of modularity to allow fixes to be made to either the RAG or UI without impacting the other.
- We can provide detailed documentation to make the installation process simple, and potentially make use of containerization software.
- During testing, we can identify issues which overcomplicate the design and redesign to address these issues.
- We can layer our design to allow for customizability options while providing beginner users with an intuitive and uncomplicated interface.
- We can test several models and combinations of models, noting each’s performance and efficiency, to find the most lightweight models which give satisfactory results.