# Cheadle Center for Biodiversity and Ecological Restoration Interim Report

**Team Members:** Kasturi Sharma, Daniel Yan, Keon Dibley, Bennett Bishop, Sean Reagan, Casey Linden

**Sponsors:** Katja Seltmann and Maddie Ostwald

**Mentor:** Professor Baracaldo

**Date:** April 25, 2025

**Introduction:**

For our Capstone project, we were tasked with creating a chatbot for the UCSB Cheadle Center for Biodiversity and Ecological Restoration (CCBER). They have a large corpus of data that provides key information on bee species, such as their taxonomies, features, and characteristics. The purpose of our project is to seamlessly include this information in a chatbot so that researchers and other interested parties can quickly access niche bee information without wasting time combing through textbooks and datasets. This tool could save many hours for researchers in finding relevant information and could even be expanded in the future with the inclusion of more data.

To create our chatbot, the main technique we used was **Retrieval Augmented Generation (RAG)**. RAG is a technique that enhances a Large Language Model (ChatGPT, etc.) with an outside source of relevant information to create a model able to generate information typically inaccessible to other LLMs. In short, a RAG model takes a user's query and searches the outside data sources for relevant context, including this context in the generated response to the user's question. In our case, we are using this method to include key bee information in the chatbot's responses to provide accurate, specific, and accessible answers to people's bee questions. Next, we will go over our progress and challenges so far in depth and provide a look at what we hope to accomplish before the end of our project.

**Problem of Interest:**

The problem we are aiming to address is how to manageably go through decades worth of data. The Big Bee lab within CCBER has vast amounts of extensive and rigorous data that they often need to be able to access quickly. However, the way the data is currently organized

makes it difficult for easy access. It can take researchers in the lab hours and up to weeks to find information that they are looking for. In order to make this process simpler for them, we are making a chatbot that cuts down on this time. Our main goal with this chatbot is to help researchers within the lab focus their time and effort on novel ideas rather than be bogged down searching for answers from the past. We want our chatbot to be as easy to use and accessible as possible. We also want this chatbot to serve as an educational tool for anyone else who is interested in learning more about bees. That is why we hope to host our chatbot on a public domain that can be accessed by anyone.

**Methods and Materials:**

For the project, we encountered a wide variety of different datasets. In total, we had to deal with three different data types. These included PDFs, CSVs, and TXT. For the PDFs, we had a total of three books that included pertinent information to train our model such as bees' structure, identification, and taxonomy. These books also gave us numerous data types to deal with such as text data and image data, as expected, but also included an interesting data type of decision tree-like key data. Our main goal with this data was getting it into a form (natural language) that could be fed into our model. Another data type that we dealt with was the CSV data. This data included 4 different CSV files of vastly different sizes. Two of them were much smaller, but the biggest of the bunch went all the way to a few gigabytes of data. This was a challenge to deal with because we needed a system that was large enough to handle the huge amount of conversions that we needed to perform to handle the data. The main goal with the CSV files was to convert them to natural language, a process which will be expanded upon later. Finally, we had a TXT file that we were able to put that to use right away for use in our model.

This TXT file included numerous definitions of the anatomy of certain bee species and related species.

We used various different methods to make these different formats usable within our model to produce intelligent responses. Starting with the PDFs, a key challenge was making sure the images were properly interpreted by the model. We wanted to extract the images from the provided PDFs. Then, our target was to store them in a cloud warehouse. Finally, we aimed to embed them so we can semantically search and pull them up for our responses. Our initial approach was to write python code, but it did not work as intended, giving us images like the one shown in **Fig 1** along with the images that we wanted.

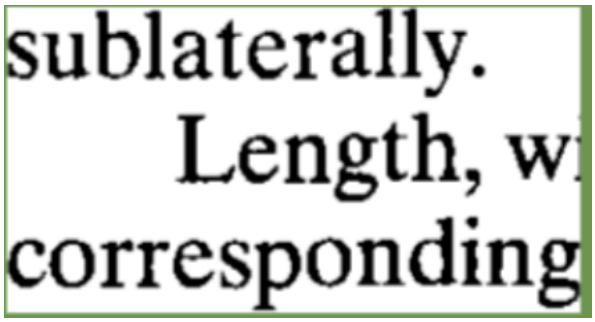


**Fig 1:** Example of how images looked initially with python code

Instead we are looking for images closely resembling something like the one shown in **Fig 2.**

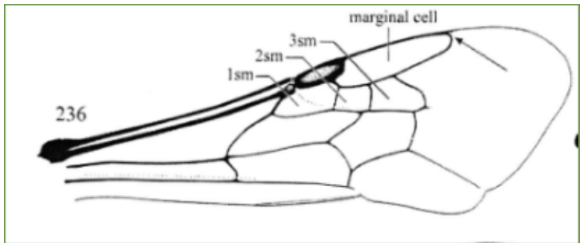


**Fig 2:** Example of what we wanted the images to look like

Our solution was not as glamorous as some fancy python code. For this, we split up the work and manually screenshotted all of the images in one of the books. This proved to be the most

effective method for what we did next. For the images to be properly assigned they needed some descriptive captions. Luckily for us, the book includes figure captions within the text. To use these, we wrote a script to scan through the PDF and extract all of the references to a particular figure. We were then able to feed these into an LLM to generate the descriptions of the figures. At the end of this process, we are now able to store the images and captions in our vector store that houses our data so that we can pull up both text and any applicable images when a user inserts a query.

Another interesting method was the conversion of the CSV files into natural language. Our main goal was to present the chatbot with TXT files that included information about as many bee species and taxonomic hierarchies as possible. For this we had to write python scripts that would take each of the columns and tidy them into a readable sentence that the chatbot could use to inform the responses that it contributes.

| flags | taxonomic_status | source | accid | id | kingdom | phylum | class | order | family | subfamily | tribe | subtribe |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | accepted | DiscoverLife | 0 | 4 | Animalia | Arthropoda | Insecta | Hymenoptera | Andrenidae | Panurginae | Calliopsini | |
| | synonym | DiscoverLife | 4 | 5 | Animalia | Arthropoda | Insecta | Hymenoptera | Andrenidae | Panurginae | Panurgini | Perditina |
| | accepted | DiscoverLife | 0 | 6 | Animalia | Arthropoda | Insecta | Hymenoptera | Andrenidae | Panurginae | Calliopsini | |
| | accepted | DiscoverLife | 0 | 7 | Animalia | Arthropoda | Insecta | Hymenoptera | Andrenidae | Panurginae | Calliopsini | |
| | synonym | DiscoverLife | 7 | 8 | Animalia | Arthropoda | Insecta | Hymenoptera | Andrenidae | Panurginae | Calliopsini | |
| | accepted | DiscoverLife | 0 | 9 | Animalia | Arthropoda | Insecta | Hymenoptera | Andrenidae | Panurginae | Calliopsini | |
| | accepted | DiscoverLife | 0 | 10 | Animalia | Arthropoda | Insecta | Hymenoptera | Andrenidae | Panurginae | Calliopsini | |
| | synonym | DiscoverLife | 10 | 11 | Animalia | Arthropoda | Insecta | Hymenoptera | Andrenidae | Panurginae | Panurgini | Camptopoeina |
| | accepted | DiscoverLife | 0 | 12 | Animalia | Arthropoda | Insecta | Hymenoptera | Andrenidae | Panurginae | Calliopsini | |
| | accepted | DiscoverLife | 0 | 13 | Animalia | Arthropoda | Insecta | Hymenoptera | Andrenidae | Panurginae | Calliopsini | |
| | synonym | DiscoverLife | 13 | 14 | Animalia | Arthropoda | Insecta | Hymenoptera | Andrenidae | Panurginae | Panurgini | Camptopoeina |
| | accepted | DiscoverLife | 0 | 15 | Animalia | Arthropoda | Insecta | Hymenoptera | Andrenidae | Panurginae | Calliopsini | |
| | synonym | DiscoverLife | 15 | 16 | Animalia | Arthropoda | Insecta | Hymenoptera | Andrenidae | Panurginae | Panurgini | Camptopoeina |
| | synonym | DiscoverLife | 15 | 17 | Animalia | Arthropoda | Insecta | Hymenoptera | Andrenidae | Panurginae | Panurgini | Camptopoeina |
| | accepted | DiscoverLife | 0 | 19 | Animalia | Arthropoda | Insecta | Hymenoptera | Andrenidae | Panurginae | Calliopsini | |
| | accepted | DiscoverLife | 0 | 21 | Animalia | Arthropoda | Insecta | Hymenoptera | Andrenidae | Panurginae | Calliopsini | |
| | accepted | DiscoverLife | 0 | 22 | Animalia | Arthropoda | Insecta | Hymenoptera | Andrenidae | Panurginae | Calliopsini | |
| | synonym | DiscoverLife | 22 | 23 | Animalia | Arthropoda | Insecta | Hymenoptera | Andrenidae | Panurginae | Panurgini | Camptopoeina |
| | accepted | DiscoverLife | 0 | 24 | Animalia | Arthropoda | Insecta | Hymenoptera | Andrenidae | Osseinae | | |



**Fig 3:** Process of using Python to convert tabular data into Natural Language

**Preliminary Findings:**

As we continued through this project, we discovered the intricacies of building a chatbot. Our initial task was to go through the data we were given and make them usable. Handling text data was relatively straightforward. We then converted tabulated data into natural language. Our

biggest struggle with the data came from the images. We ultimately found that our best method to move forward with this was to generate text captions using a large-language model (LLM) for all our images.

Once we were able to figure out how to deal with the data provided, our main goal was to started building our model. We decided to build a chatbot using retrieval augmented generation (RAG) that takes in the data we provide along with an LLM. In order to do so we had to figure out which LLM would fit our monetary and technical needs best. We eventually landed on the GPT-4 model from OpenAI. Now, we had to focus on ensuring we had proper cloud storage and a vector database. After some deliberation, we decided to use Amazon Web Services (AWS) for cloud storage and QDrant for our vector database.

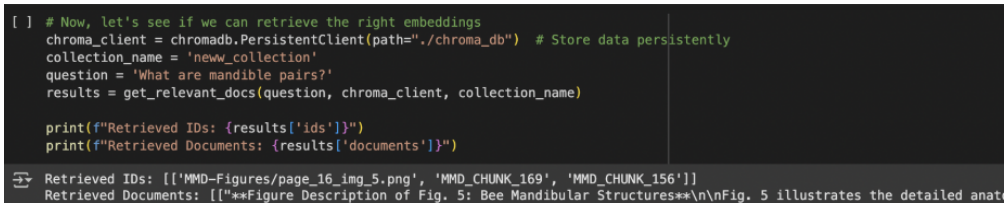


**Fig 4:** Chatbot taking in a prompt


**Fig 5:** Chatbot providing a response

Another one of our key findings is comparing how our chatbot performs relative to other models out there. In order to test this, we asked our sponsors to come up with a list of questions

that we can use to train our model and run validation tests on. Realistically, we would need hundreds if not thousands of questions. However, using the questions we were given, we tested our model against ChatGPT and asked our sponsors to rank both answers. Fortunately, our chatbot already performs better than ChatGPT as it has more specific and niche data available.

Lastly, we had to figure out what domain is best for us to use to host our chatbot. After some deliberation, we realized we want to use Streamlit as it allows us to directly create a website using python code. We are currently working to deploy it for public use. **Fig 6** shows a screenshot of what our current website looks like.

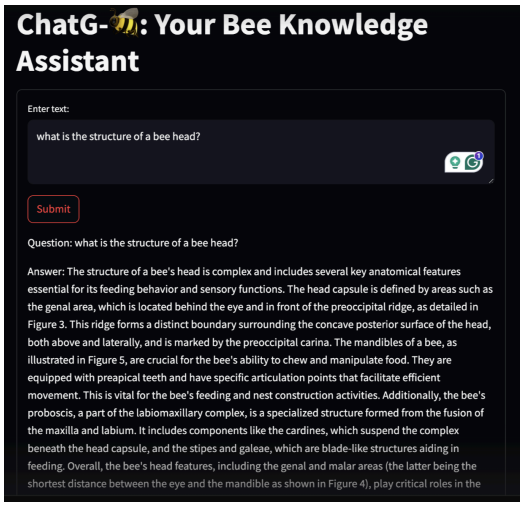


**Fig 6:** Screenshot of current website

## Discussion

Throughout the course of the project, there have been several hurdles that have needed to be addressed. Firstly, as a good proportion of our total data to work with has been in the form of PDFs, text and image extraction has posed a significant proportion of the preprocessing workload. Extracting text from PDFs using Python presented several challenges. Unlike plain text or HTML documents, PDFs are designed primarily for layout and visual presentation rather

than for easy data extraction which, like in our case, resulted in the text content within the PDFs stored in fragmented or non-linear ways, with inconsistent encoding and spacing. This made it difficult to preserve the original structure, such as paragraphs, tables, or columns, when extracting text as well as getting a clean extraction with readable results. We attempted to use Optical Character Recognition (OCR) tools which introduced its own set of challenges such as character recognition accuracy. In the end, there was still a need for manual pre-processing and cleaning to yield usable output.

In the case of one of our PDFs, text extraction from PDFs becomes significantly more problematic when the document lacks embedded text, like in our case where the text was not properly embedded. Here, the PDF did not contain reliable selectable or machine-readable text, rendering standard text extraction tools ineffective. As mentioned above, we attempted to use OCR once again to process the text, but it still resulted in inconsistent results that would require manual review of each individual page. At this current point, we decided that the most efficient way to deal with this issue was to contact the publisher to inquire about receiving the PDF in a more standard format but are still awaiting replies.

However, in the construction of our model itself, the main difficulty was enabling our RAG model to be multimodal. That is, we needed it to retrieve the most relevant data, whether it is image or text data, no matter the format. On our first attempt, we used a Contrastive Language-Image Pre-training (CLIP) embedding model. The CLIP model is trained to embed images and text into the same vector space. So, for example, an image of a horse would be embedded to roughly match the sentence ‘this is a horse.’ While testing, we found that this model was very biased towards text data, meaning for any query we sent, even for one which we knew we had a matching image, it would only retrieve text data. We realized that this is because

our image data consists mostly of minimal black and white diagrams and since the CLIP model was not trained on such niche data, it ultimately was unable to embed our images properly.

As a result, we moved away from this model and instead decided to generate text descriptions of all of our images and embed these descriptions along with other text chunks. By putting the image url into the metadata of these descriptions, when our RAG chatbot retrieves those embeddings, we are able to render the image.

**Future Work:**

The majority of our future work consists of evaluating our RAG model. There are two aspects to RAG evaluation, the retrieval and generation components. The retrieval component assesses how well the RAG model selects relevant information from the knowledge base. The generation component looks at how well the model generates a response based off of the retrieved context. There are many different ways to evaluate these components but the majority of these methods do not work for our project. For example, there is a metric for retrieval quality called Recall@k that measures the share of relevant chunks of text that are retrieved within the top k results. The issue with this metric is that we can never truly know how many relevant chunks there are in the text. A chunk is made up of a couple hundred words and we have ingested millions of words into our model. It would take way too much time to search through all of our documents and try to count the number of relevant chunks. For the generation component, one of the most popular metrics is Bilingual Evaluation Understudy (BLEU). This measures the overlap between the generated response and a set of reference answers based on n-gram matching. The problem with using this approach is that we would need our sponsors to come up with reference answers for every question in our dataset which would take time. The other issue is that if our model came up with a better answer than the reference answer, it would actually

have a worse score. Also, if our model's answer was a different length than the reference answer, this would also yield a worse score, even if the answer was more detailed. This could possibly lead us to fine tune our model to produce worse answers.

Our solution to this is to come up with around 100 relevant questions that could possibly be asked by a user. We will run these questions through our model to produce 100 answers. Our sponsors will then grade the answers based on three criteria: clarity, completeness, and correctness. Each one will be evaluated on a scale of 1 through 5, then all the scores will be added up and we will get a score. We will then adjust our model's weights, create 100 new questions, and repeat the process to fine tune our model.

Our final step is to finish up our website where we will be hosting our website for anyone to use. Right now we are utilizing a framework called Streamlit which allows us to use purely Python to create a web application. We are able to seamlessly input our RAG model using this framework. Our next steps include creating a more user friendly interface so anyone is able to use our application. We also have to find a way to save conversations with the chatbot so that users can come back days later to find the same information they were working with before. We are all very excited to finish up this project with our end goal very close in sight.