# **Arctos Search Query Chatbot Documentation**

## **Overview**

This AI-powered Streamlit chatbot enables users to query the Arctos biodiversity database using natural language (e.g., “Find fish collected in Alaska in 2001”). It extracts structured fields from the input, generates a corresponding Arctos search URL, and optionally displays sample results from the Arctos API.

Built as part of a UC Davis MSBA practicum project in collaboration with the **Museum of Vertebrate Zoology (MVZ)** and the **Arctos Consortium**, the tool significantly simplifies search workflows for curators, researchers, and external users.

## **Project Structure**

| **File** | **Description** |
| --- | --- |
| **app.py** | Main Streamlit app – handles UI, API logic, field extraction, URL building |
| **.streamlit/secrets.toml** | Secure storage of API keys (OpenAI, Arctos, Google Sheets) |
| **requirements.txt** | Python dependencies |
| **\_portals.csv** | Mapping of institutions to GUID prefixes |
| **README.md** | Setup and usage instructions |

## **Key Features**

* Natural language search for Arctos specimen data
* Field extraction via OpenAI GPT-3.5
* Arctos-compatible search URL generation
* Support for guid\_prefix mapping from institutions
* Display of sample results from the Arctos API
* Google Sheets logging of query history with timestamp

## **Functional Flow**

**1. User Query Input** The chatbot interaction begins when a user enters a natural language search query through the Streamlit interface. For example, a user might type: *“Search for reptiles in Bridgewater State University Natural History Collection.”* This input is designed to mimic how a researcher might naturally describe their information needs, without requiring them to know any of Arctos’s technical parameters or search syntax.

**2. Field Extraction Using GPT-3.5** The entered query is sent to OpenAI’s GPT-3.5 model via a custom-designed prompt. This prompt is engineered to extract specific structured fields relevant to Arctos search functionality, such as taxon\_name, institution, state, country, verbatim\_date, and others. The model returns a structured JSON object like:

{"taxon\_name": "reptiles",

"institution":"Bridgewater State University Natural History Collection",

"state": "Massachusetts"}

This structured data ensures consistency in downstream logic and allows for modular handling of field-level parsing and validation.

**3. Field Mapping and Preprocessing**

Once extracted, the fields are mapped to Arctos-compatible API parameters using a predefined dictionary (FIELD\_TO\_ARCTOS\_PARAM). This mapping ensures that field names align with those expected by the Arctos API (e.g., state maps to state\_prov, taxon\_name maps to scientific\_name). In this step, taxon names are also normalized: if a plural is detected (e.g., “reptiles”), it is converted into its singular form (e.g., “reptile”) using the inflect library to improve matching accuracy.

### **4. Institution to GUID Prefix Resolution**

### If the query includes an institution (e.g., a museum or university collection), the chatbot cross-references the institution name against a lookup table (\_portals.csv) containing known Arctos GUID prefixes. These prefixes are identifiers used in Arctos to distinguish between different collections (e.g., BSUNH:Herp for Bridgewater State University Natural History Herpetology collection). If the taxon\_name is present and recognized (e.g., mapped to herp for reptiles/amphibians), it further filters the available prefixes to ensure the most relevant one is selected.

### Resulting URL: https://arctos.database.museum/search.cfm?guid\_prefix=BSUNH%3AHerp

**5. URL Construction and Result Retrieval**

Using all the mapped and resolved fields, a valid Arctos search URL is constructed using urllib.parse.urlencode. The chatbot also sends the same parameters to the Arctos API’s getCatalogData method to fetch sample records. These records, if available, are parsed and formatted into readable summaries (e.g., “*Anaxyrus boreas collected from Alameda County on 2001-05-21*”) and displayed in the UI. This dual-layer approach allows both quick hyperlink access and inline previews for the user.

### **6. Query Logging to Google Sheets**

### To support analysis and transparency, each query is logged to a Google Sheet using a Google Service Account. The log entry includes the timestamp, original user query, extracted fields (in JSON format), and the generated Arctos URL. This lightweight tracking infrastructure enables administrators to monitor usage, identify common search patterns, and analyze user behavior over time. It can also inform future improvements, like autocomplete suggestions or additional metadata filters.

## **Deployment (Streamlit Community Cloud)**

### **Prerequisites**

* A public GitHub repository with the app files
* .streamlit/secrets.toml added via Streamlit dashboard

### **Required Files**

#### **.streamlit/secrets.toml**

OPENAI\_API\_KEY = "sk-..."

ARCTOS\_API\_KEY = "45342545-..."

GSHEET\_SHEET\_NAME = "Arctos Chatbot Logs"

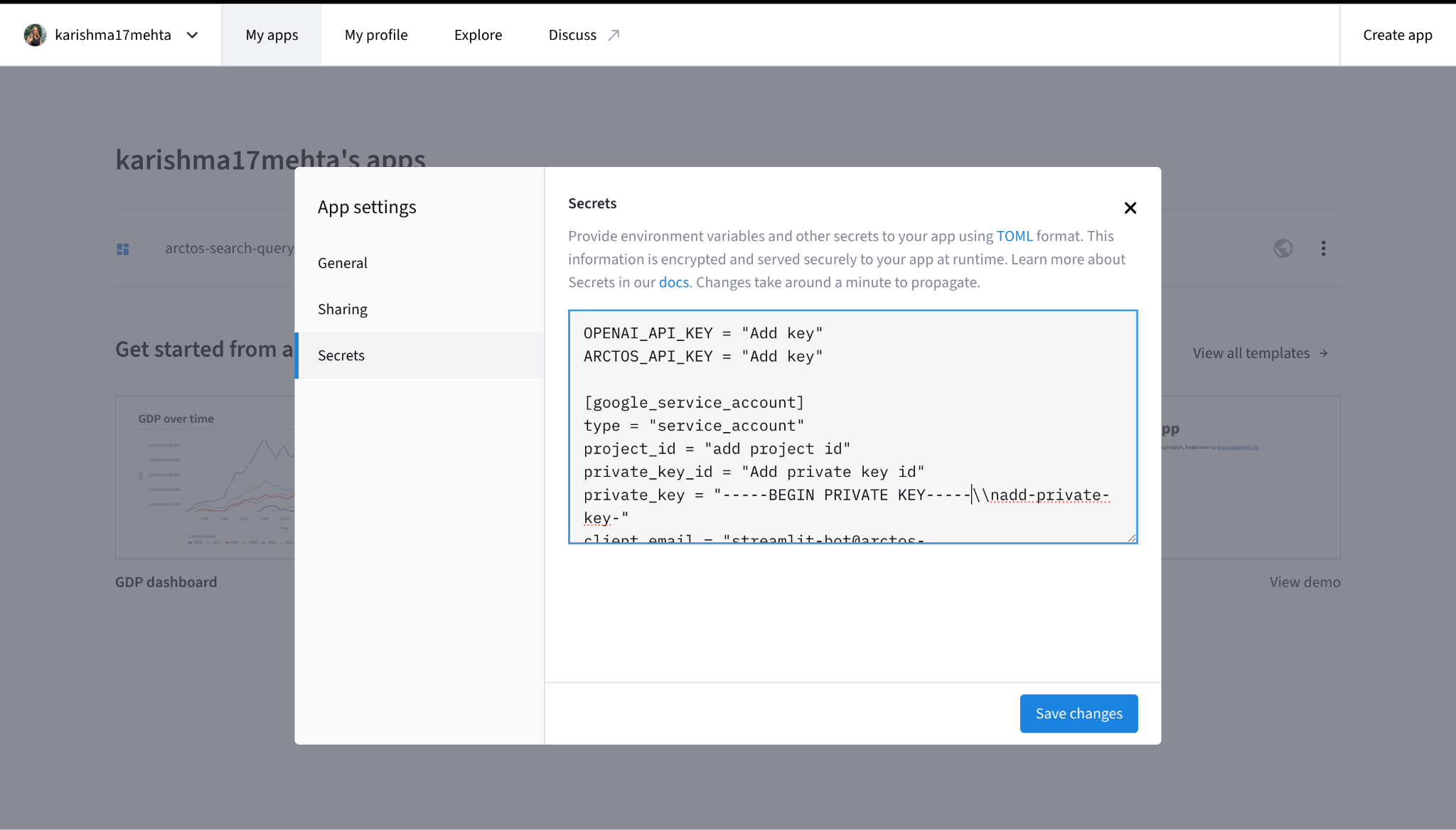
GSHEET\_URL = "https://docs.google.com/spreadsheets/d/your\_id\_here"

[google\_service\_account]{

"Type" = "service\_account",.....

}

This is a screenshot of the Secrets.toml in Streamlit Community Cloud.



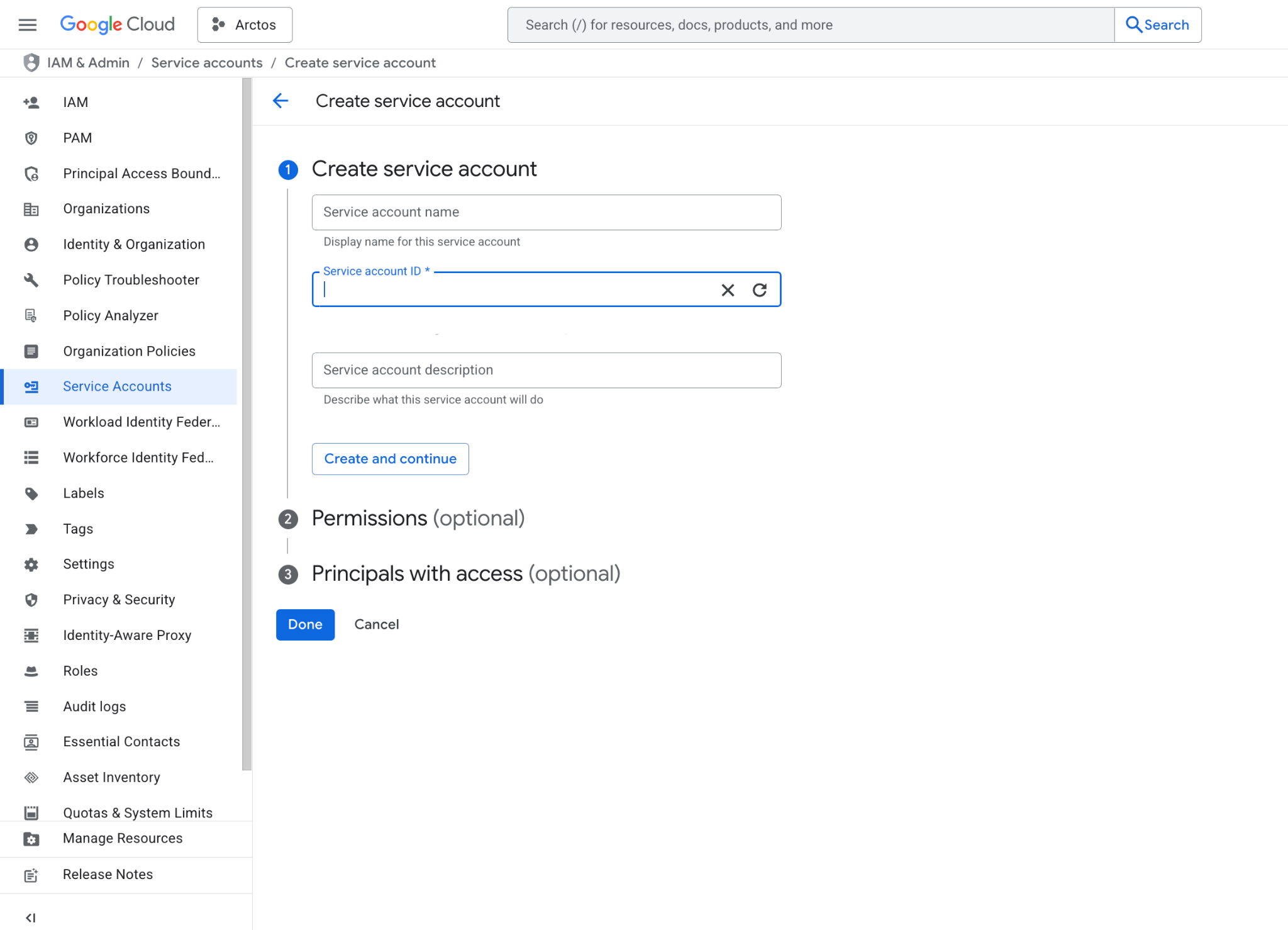
#### **requirements.txt**

* streamlit
* openai
* requests
* pandas
* inflect
* python-dotenv
* gspread
* oauth2client

### **Google Sheets Setup**

1. Create a Google Cloud project.
2. Enable Google Sheets and Google Drive API.
3. Create a **Service Account**.
4. Share your Google Sheet with the service account email.
5. Paste the credentials JSON into your GSHEET\_CREDENTIALS in Streamlit secrets.

Creating google cloud service account for the Google Cloud Project and downloading the credentials JSON format.



## **Known Limitations**

1. **Inconsistent Field Extraction from GPT**

Despite the use of carefully crafted prompts, GPT-3.5 occasionally misclassifies fields or omits important details. For instance, it may place institution names under location, or fail to distinguish between collector names and institutions. These inconsistencies are especially problematic when queries include ambiguous phrasing or entities that could plausibly belong to more than one category. While most outputs are usable, some require manual correction or validation.

1. **Incomplete or Incorrect GUID Prefix Resolution**

When taxon and institution data are both present, the app attempts to resolve the correct guid\_prefix by matching taxon keywords to prefixes from \_portals.csv. However, this process depends heavily on both correct extraction and consistent formatting. If the taxon name is not standardized or not present in the predefined TAXON\_CATEGORY\_MAP, the filtering may either fail or produce an overly broad result. Institutions with multiple collections may also lead to partial or incorrect matching.

1. **Lack of Robust Taxonomy Handling**

The app currently singularizes taxon names using basic inflection logic but does not yet validate them against a controlled vocabulary or taxonomy service. This can lead to mismatches when users input vernacular names, misspellings, or plural forms not included in the TAXON\_CATEGORY\_MAP. It also doesn’t handle hierarchical taxonomic reasoning (e.g., identifying that “wolves” fall under *Canis* genus).

1. **Limited Error Feedback and Correction Mechanisms**

If GPT fails to extract key fields or if Arctos returns no results, users currently get generic fallback messages. The app doesn’t yet support editable field previews, which could allow users to correct extraction errors before submitting. Similarly, there is no fallback strategy (e.g., retry with fewer constraints) when Arctos returns no matches.

1. **Variable Results Across Identical Queries**

Due to the probabilistic nature of GPT, identical queries can occasionally yield different field extractions depending on slight variations in formatting or previous inputs. While this improves diversity in language understanding, it can reduce predictability and complicate debugging.

## **Future Enhancements**

1. **Advanced Google Sheets Logging & Analytics Dashboard**

Expand the current logging functionality to collect metadata like user session time, institution frequency, search success/failure status, and field extraction errors. This data can feed into a separate analytics dashboard (e.g., using Google Data Studio or Streamlit) for monitoring usage trends, identifying common query patterns, and guiding future improvements based on real user behavior.

1. **Editable Field Review Interface Before Query Execution**

Introduce an optional “Review & Edit Fields” section before executing the Arctos query. Users can confirm or revise GPT-extracted fields directly in the interface, improving trust, transparency, and reducing the likelihood of misinterpreted queries. This also helps collect feedback on GPT performance.

1. **Natural Language Correction and Query Clarification Prompting**

For ambiguous or incomplete queries, add a follow-up step powered by GPT to ask clarifying questions (e.g., “Did you mean the institution or the location?” or “Can you specify a year range?”). This conversational fallback could help improve result quality and user experience, especially for vague or compound queries.

1. **User Feedback Loop for Continuous Learning**

Include an optional feedback form (or thumbs-up/thumbs-down mechanism) on query results to capture whether the returned data matched user intent. These signals can be used to evaluate GPT performance, refine prompts, or inform future fine-tuning.

1. **RAG Pipeline or Fine-Tuned LLM Model for Improved Accuracy**

Move beyond prompt engineering to deploy a Retrieval-Augmented Generation (RAG) system or a fine-tuned version of GPT on labeled Arctos queries. This would improve field precision, reduce hallucinations, and enhance institution/taxon disambiguation.

1. **Browser Plugin or Embedded Widget for Direct Integration**

Package the chatbot as a browser extension or an embeddable iframe widget so it can be deployed on the Arctos homepage, institutional websites, or research portals for real-time query assistance.

1. **Offline Mode for Educational and Museum Exhibits**

Develop an offline, limited-functionality version that can be used in museum settings or classrooms to educate users about taxonomy, biodiversity collections, and specimen metadata—without requiring API keys or internet access.

## **Authors**

Developed by UC Davis MSBA Students:  
**Karishma Mehta, Shivani Tayade, Ji Yeon Woo, Somdeb Mukhopadhyay**