Autism Treatment Database Website

Cole Oliva, Richard Jefferson, Logan Kalloway, Alex Lo

Contents

1. Project Overview and Features

1.1 Background

Medical professionals working with families to help improve the mental health of their autistic children, often have limited resources for easy-to-access, detailed information for potential treatments. Studies and trials are often hard to find, and those that are easily accessible often contain undesirable side effects. Our website mainly aims to provide for medical professionals a centralized database where information on potential treatments, backed by clinical trials and studies, can be easily found, providing more information for the medical professional. A secondary goal is to also easily provide this information to parents, who can verify this information themselves with our simple to use database. The database will be regularly updated for up-to-date data on the most recent studies.

1.2 Major Features

1. Website – A website will be available to look at treatment options.
2. Search – Searching (with filters included) allows users to determine what treatments exist for their symptoms
3. LLMs Summarization – Our website increases the amount of data articles provide by using a refined LLM to summarize articles about treatment options.
4. Database – A backend database holding article information to be retrieved by the front end (inaccessible by end users)
5. API Puller – uses a built query based on keywords from maintainers (developers and customers) to populate database with pulled articles from Pubmed

2. Project Architecture and Design

2.1 Architecture

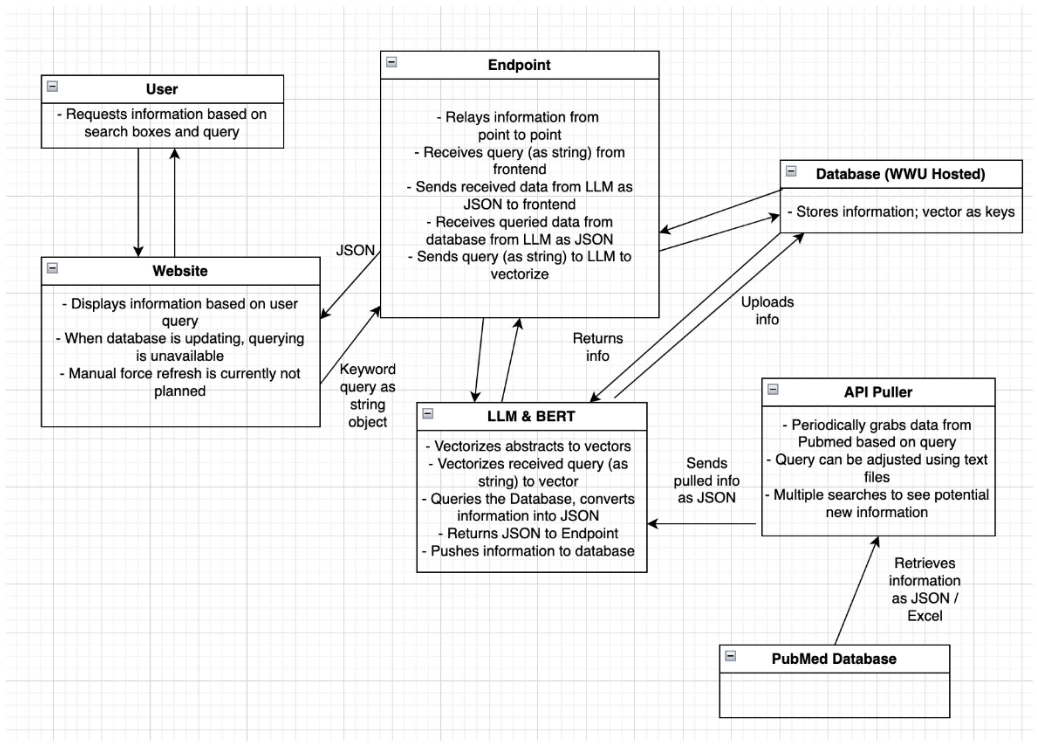
The website will be deployed to the world wide web. This website will have resources to learn more about Autism Spectrum Disorder (henceforth referred to as ASD), and a place to send a message to the website developers and client. The connection from this frontend to the backend database is through a search function, monitored by an LLM. When a user searches, the query is vectorized by a MedBERT encoder. The resulting vector is used to perform a nearest neighbor search in the database (each row in the database corresponds to an article, which corresponds to a vector). The LLM then categorizes the nearest neighbors into relevant categories, and returns a JSON to the frontend, which then displays the search results.

The frontend consists of 4 main pages; Home, Search, FAQ, and About. There is emphasis on the Search page, as that is the primary function of our project. The LLM is trained on this specific task and operates as a third-party interpreter/messenger to receive/send queries/responses from the frontend to the backend, and vice versa. Our backend is similar to a normal SQL database but includes a column of vectors that are vectorized versions of each studies' abstract. This allows for our nearest-neighbor search. Each row of the database corresponds to a specific piece of information contained in the study ('treatment type', 'age range', etc…). On the other side of our backend, we use an API connected to PubMed to obtain relevant articles and update the database accordingly. Multiple different search strategies are used to increase relevance of articles retrieved.

2.2 Dependencies

- **React JS –** frontend
- **Nodemon, Python libraries –** endpoints, server connections to backend
    - also acts as an API between frontend/backend
- **PyTorch –** LLM
- PEFT (Parameter Efficient Tuning) – LLM
- **PostgreSQL –** Server database
- **NCBI E-Utilities API**

2.3 Architecture Graphic



User
- Requests information based on search boxes and query
Website
- Displays information based on user query
- When database is updating, querying is unavailable
- Manual force refresh is currently not planned
JSON
Keyword query as string object
Endpoint
- Relays information from point to point
- Receives query (as string) from frontend
- Sends received data from LLM as JSON to frontend
- Receives queried data from database from LLM as JSON
- Sends query (as string) to LLM to vectorize
Returns info
Database (WWU Hosted)
- Stores information; vector as keys
Uploads info
LLM & BERT
- Vectorizes abstracts to vectors
- Vectorizes received query (as string) to vector
- Queries the Database, converts information into JSON
- Returns JSON to Endpoint
- Pushes information to database
Sends pulled info as JSON
API Puller
- Periodically grabs data from Pubmed based on query
- Query can be adjusted using text files
- Multiple searches to see potential new information
Retrieves information as JSON / Excel
PubMed Database

3 Project Frontend

3.1 Home Page

The Home Page serves as the entry point, providing an overview of the website's purpose and guiding users to explore treatment options. It includes a brief introduction to the database, highlighting its role in aggregating and summarizing autism treatment studies. Navigation links to the Search, FAQ, and About pages are prominently displayed. The most prominent feature, the Search function, should be the first thing that catches a user's eye.

3.2 Search Page

The Search Page is the core feature, allowing users to find treatment studies based on specific criteria. End users can input keywords or phrases related to treatments, symptoms, or demographics. The search functionality leverages a MedBERT encoder to vectorize queries and retrieve the most relevant studies from the database using a nearest-neighbor search. Results are categorized and summarized using a Large Language Model (LLM) to enhance readability and comprehension.

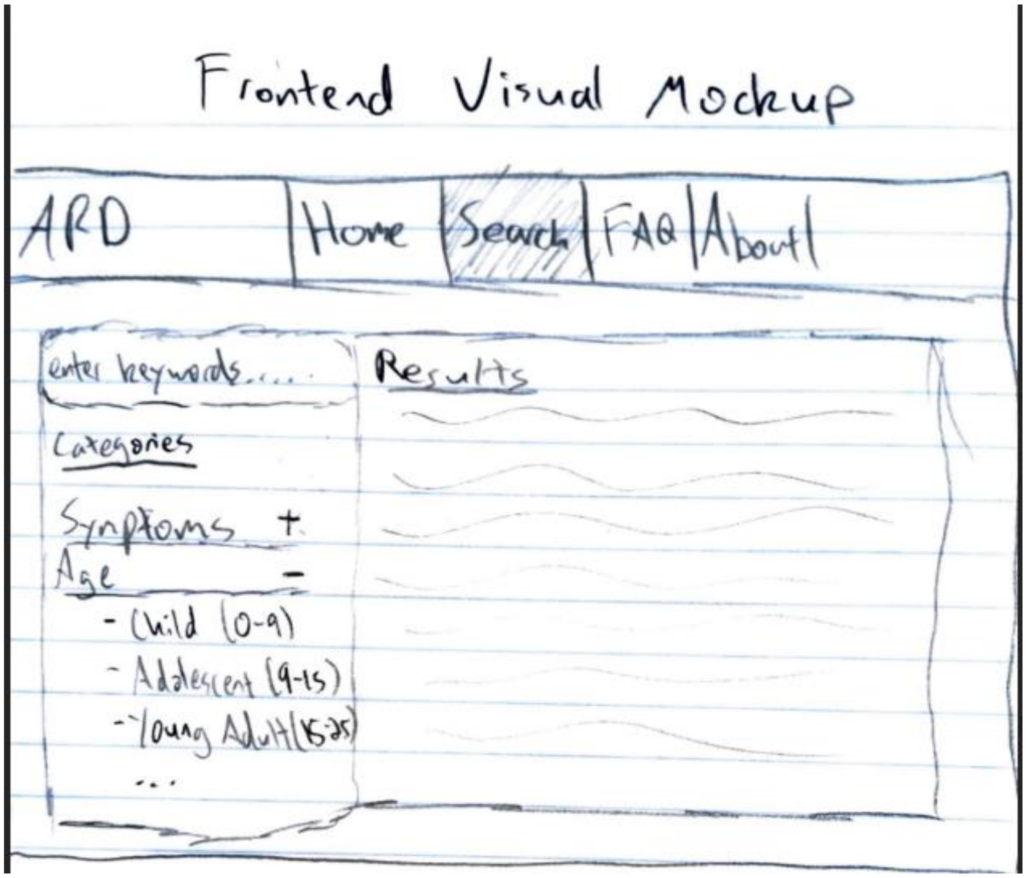
3.3 FAQ

The FAQ Page addresses common questions about the database, its sources, and how to interpret the information provided. It aims to assist both medical professionals and parents in understanding the scope and limitations of the data provided.

3.4 About

The About Page provides information about the team behind the project, the motivation for creating the database, and the methodologies employed in data collection and summarization. It also includes contact information for users who wish to provide feedback or inquire further.

3.5 Mockup for frontend



Pictured above is a quick mockup of how a basic search function would look. The side-by-side view is currently in contention, and while we do like the results section taking up most of the space, it might cause too much clutter. Subject to change.

4 Project Backend

4.1 Backend Services

The backend is responsible for handling search queries, managing the database of treatment studies, and interfacing with external APIs for data updates.

- **Search Service**: Processes user queries, vectorizes them using MedBERT, and retrieves relevant studies.
- **Summarization Service**: Utilizes an LLM to generate concise summaries of study abstracts and findings.
- **Data Ingestion Service**: Periodically fetches new studies from PubMed and other sources, updating the database accordingly.

4.2 Project Database

This project uses a PostgreSQL database hosted on Neon.tech, a cloud-based platform designed for scalability and performance. The database is organized into multiple relational tables to improve readability, ease of updates, and query efficiency. Each table serves a specific purpose, whether it's storing informationabout treatments, related research papers, or the journals in which those papers are published.

The system is designed to support:

- Linking treatments to clinical literature via PubMed IDs (PMIDs)
- Storing metadata about journals and papers
- Providing semantic search capabilities for better paper discovery

Schema Explanation

- journals
  - id: Unique identifier for the journal.
  - journal: Name of the journal.
  - publisher: Organization or entity that publishes the journal.
  - impact_factor: Numeric value indicating the journal's influence.
  - issn: International Standard Serial Number.
  - url: Link to the journal's website.
- treatments
  - id: Unique identifier for the treatment.
  - treatment_name: Name of the treatment.
  - duration: Length of time the treatment is applied.
  - primary_outcome_area: Main area of impact (e.g., symptom reduction).
  - primary_outcome_measures: Metrics used to evaluate effectiveness.
  - results_primary_measure: Results based on the primary measure.
  - journal_id: Foreign key linking to journals.id.
- treatment_pubmed_link
  - treatment_id: Foreign key linking to treatments.id.
  - pmid: Foreign key linking to either pubmed_papers.pmid or semantic_paper_search_view.pmid.
- pubmed_papers
  - pmid: Unique PubMed identifier.
  - doi: Digital Object Identifier.
  - title: Title of the paper.
  - pub_date: Publication date.
  - abstract: Abstract text.
  - authors: List of authors.
  - journal: Journal where published.
  - keywords: Keywords describing content.
  - url: Link to full paper.

    - affiliations: Author affiliations.
    - embedding: Vector representation for semantic analysis.
- semantic_paper_search_view
    - This table helps power the search function on the website. When someone enters a keyword or phrase, the system looks through this table to find the most relevant research papers. It includes summaries, author information, keywords, and other details to make it easier to match what users are searching for.
    - pmid, title, pub_date, authors, journal, abstract, keywords, url, affiliations, embedding.

5 API Puller

5.1 Pubmed, Entrez, and E-Utilities

The Puller is built on top of NCBI's E-Utilities API, allowing for access into the NCBI Entrez System and its databases, including Pubmed, the database in which our articles are sourced from. We also use an API key (obtainable through a free sign-up to NCBI) in order to increase the max number of requests from 3 to 10 per second. If more requests per second are necessary, it is possible to request more through eutilities@ncbi.nlm.nih.gov.

We specifically use ESearch to retrieve the relevant IDs related to each query, and EFetch to retrieve information associated with each ID in a specific format. More information of each feature can be found here: www.ncbi.nlm.nih.gov/books/NBK25501/.

5.2 Relevant Query Documents

In order to use ESearch and Efetch, we need to build a query. We have an existing general query, built with references and input from our customer, but it does not contain the most relevant articles to our database. To refine our query, and in case of future information change, we need an easy way to quickly change our query and is intuitive for our maintainers in the future. Ideally, without a deep dive into our code so that laypeople can easily change it as well.

To this end, we have included 2 txt files, namely biologic_terms.txt and social_terms.txt. On each line of each txt file, you need to include the name of the treatment, and its relevant filter (e.g. "Antibodies, monoclonal"[MeSH]). This is a relatively simple and easy way to add and remove treatments, and to narrow down results for relevant queries. We have confirmed with our customer that this is a suitable method for altering their desired query.

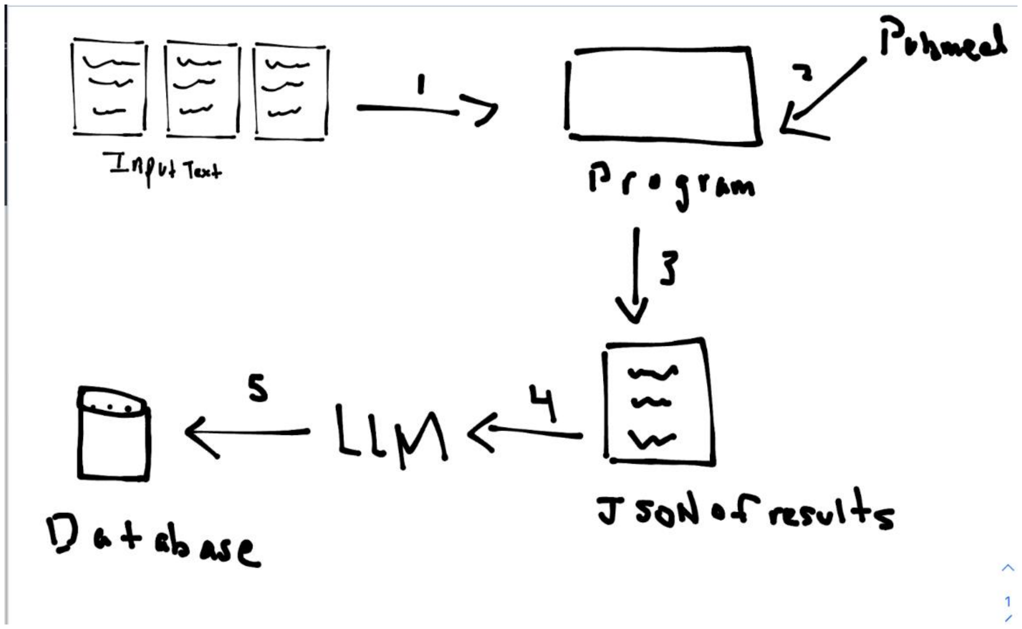
We also have an excel document, named Treatment_Names.xlsx, to do individual searches based on one treatment, that operates in a similar fashion. However, one just needs to provide the treatment name, and nothing else.

5.3 API Puller Program

The API Puller Program is planned to run every week to update with new information, though there should be no problem running it on demand. It reads information from the relevant files, builds the query based on the information. It then uses ESearch and EFetch from E-Utilities to retrive the relevant information of the articles returned from the query in an Excel format.

We then convert the Excel sheet to a JSON, which we pass to the LLM to summarize the information and store it in the database.

5.4 Diagram of Puller and Docs

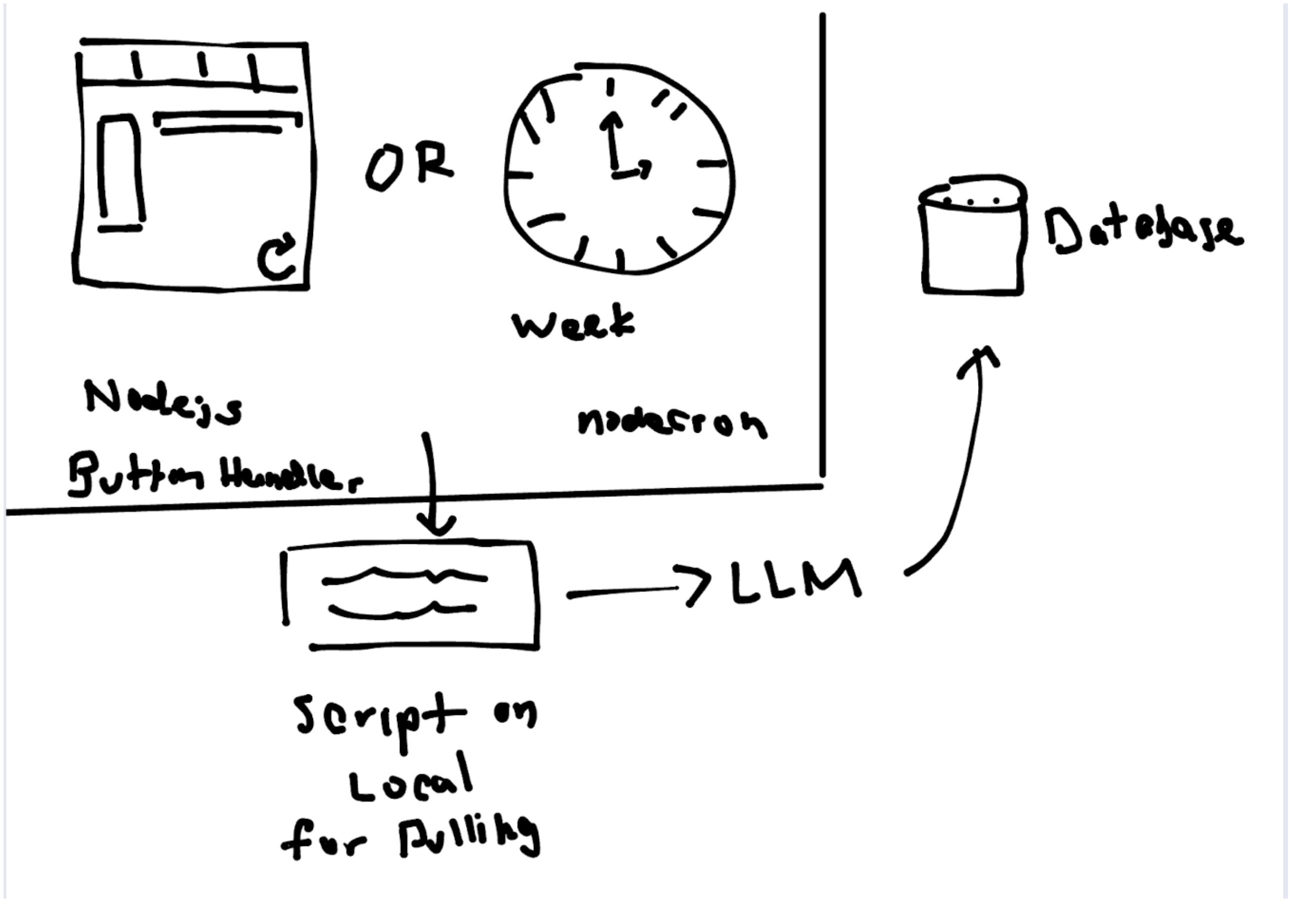


Subject to Change; Program takes input text, pulls from Pubmed, convert results into JSON, pass to LLM, which summarizes information and passes entirety to Database

5.5 API Puller Program Scheduling

Our framework is based on Node.js, and as such, we will be using the node-cron and nodemon library and tool to cause our script to run whenever a condition is met. Nodemon allows us to restart our server automatically whenever the database is updated, and if we are attaching to a local computer, node-cron can automatically schedule jobs to run. This allows the database to automatically update on a tentative schedule of every week, at common downtimes, such as midnight on Sunday potentially. As PostGresSQL is also ACID compliant, we can safely read from the database even on a manual refresh, though we cannot guarantee the updating results will show up.

5.6 Diagram of Update Method



Subject to Change; Script runs from either manual refresh (Node.js handles the button press), or time (node-cron). The input is passed to the LLM and the database afterwards. The website is still accessible during the update.

6 LLM Summarization and MedBERT Vectorization

6.1 LLM Summarization

To process the massive number of papers available, we will use a large language model to read through the papers and fill out the fields required. To save on costs connecting to an AI API we will use a local model to fill out the fields. For extra accuracy and to make use of limited sample data we will use several different finetunings of one LLM specifically trained in each field. We will then use a causal LLM to evaluate that treatment for a given outcome area, giving it a score out of 10 so we can display that to our users. For transparency, we will track which paper summarizations have been checked by a human and which have only been viewed by an AI and let the user choose if they want to see AI summarized papers in their results.

6.2 LLM Refining

To make sure this data given by the LLM is as accurate as possible, we will refine the model using perameter efficient training specifically low rank adaptation on a data set given by the Sendan Center. This will allow us to use smaller and easier to run models to summarize the papers well and to train a model efficiently without a lot of data to feed it. We will use several different finetunings to start to make our tuning as accurate as possible. We can then train one main model off the data we created and use that for processing new papers to save on server size if needed.

6.3 MedBERT Vectorization

To search for data, we are using a vectorized database with nearest neighbor search. A MedBERT will be used to vectorize the papers going into the database and to vectorize searches the users input to find treatments for what their search. The vector will encode the words and their context to a searchable set of numbers. To find treatments we will use the nearest neighbor searching functionality of PostgreSQL. Using a MedBERT will mean they are vectorized within a medical context making them more accurate when catagorizing and searching them. Vector searching allows us to find close matches to the search terms without needing the exact correct words that are used like in a keyword system but with the hybrid approach that PostgreSQL uses we can still use keyword search functions like removing studies outside of a specific age range or remove AI summarized results from the search.