ChatGPT

System Architecture Overview

We propose a modular system composed of distinct components: a **Prompt Interpreter** (natural-language → structured query), an **Ontology Mapper** (term normalization), a **Query Builder** (constructs E-Utilities queries), a **Retriever** (executes queries via NCBI tools), a **Metadata Aggregator** (parses and joins results), a **Summarizer** (generates human-friendly report), and an **Output Formatter** (JSON/DF). Each component can run as a separate service or agent. In broad outline:

- **Prompt Interpreter:** Ingests the user's NL query and identifies key concepts (assay, tissue, disease, species).
- **Ontology/Dictionaries:** Maps synonyms to controlled terms (e.g. "WGBS" → "Whole Genome Bisulfite Sequencing"; "brain" → anatomical ontology ID; "cancer" → disease ontology ID).
- **Query Builder:** Uses the mapped terms to formulate a structured search (e.g., Entrez ESearch query filters).
- **Retriever:** Calls NCBI APIs (via Entrezpy or Biopython's `Bio.Entrez`) and other tools (GEOparse, GEOfetch, pysradb) to fetch relevant series (GSE) or sample (GSM/SRP) metadata.
- **Metadata Aggregator:** Parses returned data (SOFT files, run tables) to extract fields like sample counts, tissue/disease annotations, assay types, etc.
- **Summarizer:** (Optional) Feeds aggregated findings into an LLM to produce a natural-language summary.
- **Output Formatter:** Packages results into machine-readable formats (JSON, Pandas DataFrame).

This pipeline can be visualized as a flow: **Natural Language Prompt → (Interpreter) → Structured Query → (Entrez/SRA Tools) → Raw GEO/SRA Metadata → (Aggregator) → Summary Statistics → (Formatter) → Output**.

Module Descriptions

- **Prompt Interpreter & Ontology Mapper:** This module uses NLP techniques (rule-based parsing or LLMs) to extract entities from the prompt. For example, "dna-methylation (WGBS)", "human", "brain", "cancer" should be recognized as assay="WGBS", organism="Homo sapiens", tissue="brain", disease="cancer". An NLP model (e.g. spaCy or a fine-tuned BERT) or a prompt-engineered LLM (GPT) can tag these terms. A mapping layer consults ontologies (e.g. MeSH, Disease Ontology, Uberon for tissues) to normalize synonyms and ensure consistent query terms. For instance, "dna methylation" might map to the assay "WGBS" and corresponding MeSH terms. Domain-specific libraries like BioPortal or the Disease Ontology API can assist mapping.

- **Query Builder:** Converts the structured criteria into a query string for GEO/SRA. For example, it might build an Entrez query such as `Homo sapiens[Organism] AND brain[Tissue] AND bisulfite[Title] AND (cancer[All Fields] OR neoplasm[All Fields])`. This builder uses templates and logical operators to include fields (organism, tissue, disease, assay) in one search. It can leverage Entrezpy or Biopython's `Bio.Entrez` to format ESearch parameters (database=`gds` for GEO DataSets, or `sra`/`sraproject`). These libraries support **E-Utilities**:

Entrezpy's `Conduit` class simplifies pipelining ESearch and EFetch calls [1]. For example:

```
import entrezpy.conduit
c = entrezpy.conduit.Conduit("user@example.com")
pipeline = c.new_pipeline()
term = 'Homo sapiens[Organism] AND brain[Tissue] AND WGBS[Title] AND
cancer[All]'
sid = pipeline.add_search({'db': 'gds', 'term': term, 'retmax': 100})
fid = pipeline.add_fetch({'retmode': 'xml'}, dependency=sid)
c.run(pipeline)
```

This would retrieve GEO Series (GSE) matching the criteria. Biopython's `Bio.Entrez` offers similar capabilities [2] [1]. The Query Builder can also incorporate advanced filters (dates, journal, etc.) and support multiple databases (GEO vs. SRA).

- **Data Retriever:** This module executes queries and collects raw metadata. It uses multiple tools:

- **Entrezpy / Bio.Entrez:** For ESearch/EFetch. For example, using Entrezpy we can search the GEO DataSets (`db='gds'`) or Samples (`db='gds'` or `db='gds'` if searching series), and then EFetch or ESummary to get details. Bio.Entrez can also download SOFT files (though GEO's E-utilities don't natively output SOFT, one may fetch via FTP [3]).
- **GEOparse:** A Python library to download and parse GEO SOFT files into Python objects [4]. For each GSE ID, `GEOparse.get_GEO(geo="GSEXXXXX")` downloads the data and returns a `GSE` object containing metadata, sample tables, etc [4] [5]. This makes it easy to inspect `gse.metadata`, `gse.gsms`, and `gse.gpls`. For example:

```
import GEOparse
gse = GEOparse.get_GEO(geo="GSE1563", destdir="./data")
print(len(gse.gsms), "samples; fields:", list(gse.metadata.keys()))
```

- **GEOfetch:** A command-line/Python tool that automates downloading and organizing GEO/SRA data [6]. GEOfetch can accept a list of GSE/SRP accessions and fetch raw/processed files and metadata, outputting a standardized *PEP sample table* (CSV) of metadata [6]. For instance, in Python:

```
from geofetch import Geofetcher
geof = Geofetcher(processed=False, just_metadata=True)
geof.get_projects("GSE160204")
```

This yields a project directory with a `samples.csv` describing each sample's attributes. GEOfetch supports filters and can combine projects; it "works with GEO and SRA metadata" and "standardizes output metadata" [7].
- **pysradb:** For sequencing experiments linked to SRA. Using `SRAWeb.sra_metadata`, one can fetch the SRA run table for an SRP (project) or GSE (since GEO Samples often have GSM↔SRR links) [8]. This returns a Pandas DataFrame of metadata, e.g. sample_accession, experiment_title,

library_strategy, etc [9]. Example:

```
from pysradb import SRAWeb
db = SRAWeb()
df = db.sra_metadata('SRP123456')  # returns a pandas DataFrame of the run
table
```

The ability to filter this DataFrame (e.g. only RNA-Seq rows) facilitates precise retrieval [10] [9].

The Retriever may also maintain a local cache or database (e.g. SQLite) to avoid repeated fetches. A local **GEOmetadb** database is an option: it is a pre-parsed SQLite database of all GEO metadata [11]. Queries that would require multiple E-utilities calls can often be done with one SQL query if GEOmetadb is used. The system could download the GEOmetadb SQLite (via its R package or direct link) and use `sqlite3` to run queries for series/sample metadata [11]. This improves speed and avoids API rate limits.

- **Metadata Aggregator:** Once raw data (SOFT files or tables) is retrieved, this module parses and condenses the information. It might load the GEOparse `GSE` objects or PEP CSVs and extract key attributes: sample count, experiment design, organism, tissue, disease, platform (GPL), assay type, publication, etc. It then collates across all matching GSEs. For example, it can count how many samples labeled "brain", how many series mention "cancer", or the distribution of assays (WGBS vs RRBS, etc). This may involve simple Python/pandas operations (group-by on metadata fields). The result is summary statistics and a structured record (e.g. each GSE with its metadata, or aggregated counts by category). Standard tools like pandas make this straightforward once data is in DataFrames or dicts.

- **Summarizer:** Optionally, an LLM can be used to translate the structured output into a coherent paragraph. For example, given the aggregated JSON or table of metadata, one could prompt a model: "Summarize the following GEO search results for DNA methylation in human brain cancer: {list of experiments and sample counts}." The LLM might then say: *"We identified X GEO Series of whole-genome bisulfite sequencing in human brain cancer, comprising Y total samples. These studies primarily focus on [glioblastoma etc.] with [number] cases and [number] controls, using [Illumina] platforms… "* (etc.). Such a summary makes results accessible to a non-technical user.

- **Output Formatter:** Finally, the system formats the findings. A machine-readable **JSON** could include fields like `{ "query": {...}, "series": [{ "GSE": "GSE12345", "num_samples": 10, "tissue": "brain", "disease": "glioblastoma", ...}, ...] }`. Or it could return a pandas DataFrame (e.g. rows=GSEs or summary rows). The Formatter should also return metadata about the query (time, tools used, etc.) for reproducibility. JSON schemas or PEP tables ensure consistency.

Technology and Libraries

- **Natural Language Processing:** Could use spaCy with biomedical NER models (SciSpaCy) or custom keyword matching. Alternatively, use an LLM (e.g. GPT) with an instruction to extract and map query terms. No specific citation, but LLMs and NLP are standard.

- **Ontology/Dictionaries:** UMLS/NCBI Taxonomy for species, Disease Ontology/MeSH for diseases, Uberon or Tissue Ontology for tissues. Libraries: `owlready2`, `bioversions`, or direct API calls to Ontology Lookup Service.

- **Entrezpy vs Biopython:** Entrezpy (a purpose-built Python wrapper for NCBI E-Utilities) simplifies pipeline execution [1]. Biopython's `Bio.Entrez` provides similar access (requires respecting the 3-second delay rule [2]). Either can search GEO (`db="gds"` for DataSets) and SRA. Cite Entrezpy: "Entrezpy automates querying NCBI Entrez databases via E-Utilities [1]."

- **GEOparse:** A Python library to fetch and parse GEO SOFT files [4]. It can load GSE, GSM, GPL as Python objects. Installation via `pip install GEOparse`. Cited above examples show usage [4].

- **GEOfetch:** A tool/PEPkit for GEO/SRA data. It can be invoked as CLI or via `geofetch` Python API [6]. We would use it to fetch metadata-only or include files. Its PEP output is CSV-formatted sample metadata.

- **pysradb:** A Python tool to access the SRA metadata (leveraging the SRAmetadb SQLite) and download runs [8]. Use `pip install pysradb`. It returns pandas DataFrames [9].

- **NCBI E-Utilities:** The low-level API (HTTP queries) is accessible via Biopython or direct REST calls. Useful for ad-hoc queries or data not covered by above tools.

- **Local Database (Optional):** GEOmetadb provides an up-to-date SQLite of GEO metadata [11]. Using it (via Python's `sqlite3` or `pandas.read_sql_query`) can speed searches. Similarly, the SRAmetadb (for pysradb) can be downloaded for offline SRA metadata.

- **LLM (Optional):** Any GPT-3.5/4 or other transformer for text summarization. Use via OpenAI API or local LLM library (e.g. Hugging Face Transformers). No specific citation needed.

Pipeline Flow

1. **User Prompt**: e.g. "Metadata/summary of WGBS data in human brain with cancer."
2. **Interpretation**: NLP/LLM extracts keywords: `assay="WGBS"`, `tissue="brain"`, `disease="cancer"`, `organism="Homo sapiens"`.
3. **Query Construction**: Terms are mapped to controlled vocab. The Query Builder formulates an Entrez search string such as:

```
Homo sapiens[Organism] AND brain[Tissue] AND (cancer OR neoplasm)[All
Fields] AND (WGBS OR bisulfite)
```

4. **Entrez Search**: Submit the query via Entrezpy (e.g. `db='gds'`). This returns matching GEO series IDs (GSEs) and counts [1].
5. **Result Validation**: Ensure results are indeed functional genomics (filter out irrelevant hits). Possibly refine query or check keywords in descriptions.

6. **Metadata Retrieval**: For each GSE, use GEOparse to download the SOFT file (e.g. `GEOparse.get_GEO(geo="GSExxxxx")`) and extract metadata (platform, sample attributes) [4]. Alternatively, use GEOfetch with `--just-metadata` to bulk-download metadata [12]. If raw SRA IDs are present (SRX/SRR), pysradb can fetch the run table (e.g. `db.sra_metadata('SRPxxxxx')`) and sample attributes [9].
7. **Aggregate Data**: Combine metadata from all GSEs. Summarize: total number of series found, total samples, breakdown by tissue or disease if varied, list of assay types (some GSE might contain multiple assays). Compute any desired summary stat (e.g. average read depth if obtainable from runs).
8. **Output Generation**: Package the structured data into JSON or DataFrame. Include raw metadata (or links to files) and the summarized stats. For example:

```
{
  "query_terms":
{"assay":"WGBS","tissue":"brain","disease":"cancer","organism":"Homo
sapiens"},
  "total_series_found": 5,
  "total_samples": 120,
  "series": [
    {"GSE":"GSE12345","title":"...", "platform":"GPLXXX","samples":
20,"disease":"Glioblastoma",...},
    ...
  ]
}
```

9. **Human-Readable Summary**: Optionally, feed the JSON or a distilled version into an LLM prompt to get a brief narrative: *"We found 5 GEO series of WGBS experiments in human brain cancer. In total, 120 samples are included (e.g. 60 tumors, 60 controls) across these studies. The experiments were run on [platforms] and include [any notable finding]."*.

MVP Roadmap (Single-Agent)

Phase 1 – Core Search and Retrieval:
- **1.1 Term Extraction:** Hard-code simple keyword matching (e.g. "WGBS", "brain", "cancer") and mapping to query terms.
- **1.2 Query Builder & Search:** Use Entrezpy to perform a basic ESearch on GEO (db=gds). Retrieve GSE IDs.
- **1.3 Metadata Fetch:** For each GSE, use GEOparse to download the SOFT file and extract sample metadata (subject, tissue, assay).
- **1.4 Basic Aggregation:** Summarize count of series and samples, list assay types and tissues. Output as JSON/CSV.
- **1.5 CLI/Function Interface:** Expose as a simple function or command-line script taking a prompt and returning JSON.
- **1.6 Validation:** Test with queries (e.g. "WGBS brain cancer Homo sapiens") and compare output to manual GEO searches.

Phase 2 – Enrichment and Robustness:
- **2.1 NLP Integration:** Replace hard-coded parsing with an NLP model or configurable dictionaries to handle varied phrasing.
- **2.2 Additional Filters:** Support date ranges, exclude deprecated series, etc.
- **2.3 Use GEOfetch:** Integrate GEOfetch for faster batch metadata retrieval (especially for large results).
- **2.4 pysradb Integration:** When available, use pysradb to verify assay types (e.g. confirm "Bisulfite-Seq" vs generic RNA-Seq entries).
- **2.5 Output Formatting:** Return results as Pandas DataFrames (for easy analysis) and JSON with schemas.

Phase 3 – LLM Summarization & UI:
- **3.1 LLM Summarizer:** Add a step to call an LLM with the aggregated metadata to produce a textual summary.
- **3.2 Logging & Error Handling:** Implement robust logging, retry for API timeouts, and clear error messages if no data is found.
- **3.3 Packaging:** Build the system as an installable library or API service (e.g. Flask/ FastAPI).
- **3.4 Documentation & Examples:** Provide examples of prompts and outputs.

Single-Agent vs Multi-Agent

The MVP can be a single Python application (one agent) that sequentially performs these steps. However, even in MVP, it's helpful to modularize (e.g. separate classes or functions for "search" vs "fetch").

Multi-Agent Orchestration

For scaling and separation of concerns, we can split into agents or microservices:

- **Agent 1: Prompt Interpreter/Query Planning.** This agent receives the user's prompt, uses an LLM or rules to identify query parameters, consults ontologies, and produces a structured query plan. It might run as an LLM agent (e.g. a custom GPT-based agent) that outputs JSON like `{"assay": "WGBS", "tissue": "brain", ...}`.
- **Agent 2: Retriever.** This agent takes a structured query, issues Entrezpy/Biopython calls to search GEO/SRA, and collects raw results. It could run asynchronously or in parallel (submitting multiple queries). For very large queries, it could chunk the query space.
- **Agent 3: Data Parser/Aggregator.** Upon receiving raw data (SOFT files, run tables), this agent extracts the relevant fields and aggregates statistics. It can utilize GEOparse, pysradb, etc. It may store intermediate results in a database or cache.
- **Agent 4: Summarizer.** When given the aggregated data, this agent crafts a human-readable summary (via an LLM prompt).
- **Agent 5: Output Manager.** Formats and returns the final result (JSON, DataFrame) to the user.

These agents can communicate via messages or a broker (e.g. RabbitMQ) or simply call each other in sequence. Tools like LangChain or custom orchestrators can manage these steps. The advantage is that each agent can be scaled independently or replaced (e.g. swapping in a better NLP agent for prompt interpretation).

Extensibility

- **Additional Repositories:** The system could be extended to query EBI's ArrayExpress (using their REST API or `bioservices`), or other genomic databases (SRA standalone, EGA for controlled data, etc.). For example, the Bioservices library's ArrayExpress client can fetch metadata from ArrayExpress experiments.
- **Full-Data Download:** For identified experiments, integrate data download (via Aspera, `prefetch`, or AWS S3) to obtain raw FASTQs or processed files. The pipeline could then optionally trigger downstream analysis.
- **Advanced Summaries:** Use machine learning to detect trends or cluster results (e.g. group series by similarity of metadata). Integrate knowledge graphs linking samples to known annotations.
- **Interactive UI/Notebook:** Provide a web interface or Jupyter widget where a user enters a prompt and sees interactive tables/plots of results.
- **Caching and Databases:** As usage grows, add a metadata cache (e.g. ElasticSearch or a relational DB) to store previous query results for instant retrieval.
- **Cloud/Batch Execution:** Containerize agents (Docker) and deploy on Kubernetes or serverless (AWS Lambda) for scalability. Use workflow managers (Airflow, Nextflow) for scheduling.
- **Community Contributions:** Allow adding new filters or ontologies via config files.

Throughout development, each module's behavior should be documented and logged. Unit tests for query parsing and data retrieval will ensure reliability. With this blueprint, a team can implement the MVP steps and iteratively build out additional agents and features.

References: We leverage existing tools and libraries. For example, Entrezpy provides a Python interface to NCBI E-Utilities [1]. GEOparse allows downloading and parsing GEO Series in Python [4]. GEOfetch can fetch GEO/SRA metadata and output standardized PEP tables [6]. Pysradb's SRAWeb returns SRA run metadata as pandas DataFrames [9]. The GEOmetadb package compiles all GEO metadata into a local SQLite database for powerful queries [11]. These and other references guided our design.

[1] Entrezpy: NCBI Entrez databases at your fingertips — entrezpy .dev documentation
https://entrezpy.readthedocs.io/en/master/

[2] [3] Accessing NCBI's Entrez databases — test test documentation
http://biopython-tutorial.readthedocs.io/en/latest/notebooks/09%20-%20Accessing%20NCBIs%20Entrez%20databases.html

[4] [5] Usage — GEOparse 1.2.0 documentation
https://geoparse.readthedocs.io/en/latest/usage.html

[6] [7] [12] GEOfetch - PEPkit: the bio data management toolkit
https://pep.databio.org/geofetch/

[8] [9] [10] Python API — pysradb 2.1.0 documentation
https://saket-choudhary.me/pysradb/python-api-usage.html

[11] Bioconductor - GEOmetadb
https://www.bioconductor.org/packages/release/bioc/html/GEOmetadb.html