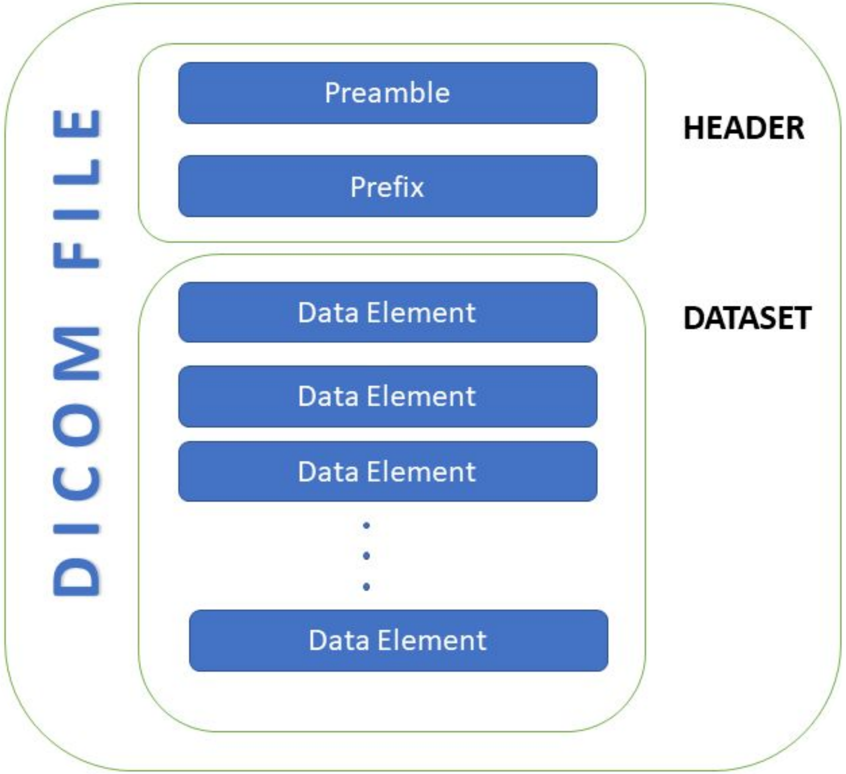
# LoxLM

## DICOM to BIDS conversion is hard

- ❖ There exist no fully automatic tools to convert between the two standards
- ❖ Conversion requires a technician with knowledge of the dataset and the conversion tool

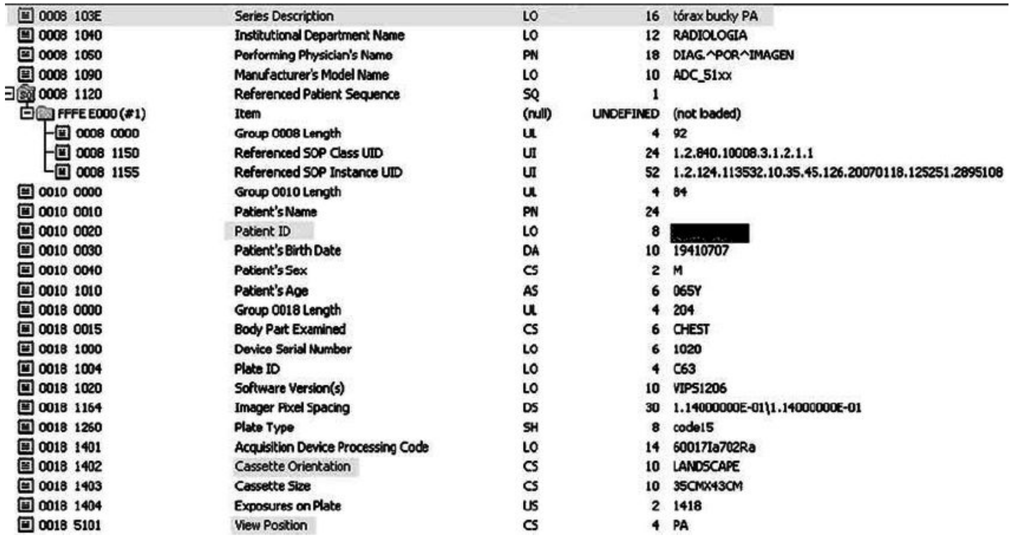
## DICOM format

Header information describes the raw data elements.



## DICOM Headers

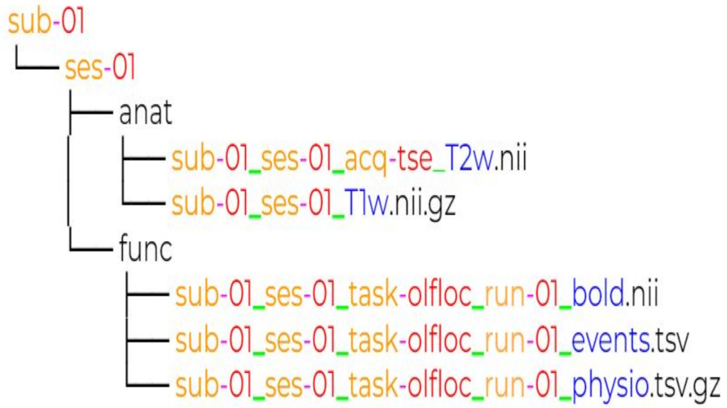
- Many DICOM tags exist
- The values for these tags are frequently entered manually by scanning technician. (INCONSISTENCY)



| | | | | |
|---|---|---|---|---|
| 0008 103E | Series Description | LO | 16 | tórax bucky PA |
| 0008 1040 | Institutional Department Name | LO | 12 | RADIOLOGIA |
| 0008 1050 | Performing Physician's Name | PN | 18 | DIAG.^POR^IMAGEN |
| 0008 1090 | Manufacturer's Model Name | LO | 10 | ADC_51xx |
| 0008 1120 | Referenced Patient Sequence | SQ | 1 | |
| FFFE E000 (#1) | Item | (null) | UNDEFINED | (not loaded) |
| 0008 0000 | Group 0008 Length | UL | 4 | 92 |
| 0008 1150 | Referenced SOP Class UID | UI | 24 | 1.2.840.10008.3.1.2.1.1 |
| 0008 1155 | Referenced SOP Instance UID | UI | 52 | 1.2.124.113532.10.35.45.126.20070118.125251.2895108 |
| 0010 0000 | Group 0010 Length | UL | 4 | 84 |
| 0010 0010 | Patient's Name | PN | 24 | |
| 0010 0020 | Patient ID | LO | 8 | [redacted] |
| 0010 0030 | Patient's Birth Date | DA | 10 | 19410707 |
| 0010 0040 | Patient's Sex | CS | 2 | M |
| 0010 1010 | Patient's Age | AS | 6 | 065Y |
| 0018 0000 | Group 0018 Length | UL | 4 | 204 |
| 0018 0015 | Body Part Examined | CS | 6 | CHEST |
| 0018 1000 | Device Serial Number | LO | 6 | 1020 |
| 0018 1004 | Plate ID | LO | 4 | C63 |
| 0018 1020 | Software Version(s) | LO | 10 | VIPS1206 |
| 0018 1164 | Imager Pixel Spacing | DS | 30 | 1.14000000E-01\1.14000000E-01 |
| 0018 1260 | Plate Type | SH | 8 | code15 |
| 0018 1401 | Acquisition Device Processing Code | LO | 14 | 60017Ia702Ra |
| 0018 1402 | Cassette Orientation | CS | 10 | LANDSCAPE |
| 0018 1403 | Cassette Size | CS | 10 | 35CMX43CM |
| 0018 1404 | Exposures on Plate | US | 2 | 1418 |
| 0018 5101 | View Position | CS | 4 | PA |

## BIDS format

- ❖ Widely adopted neuroimaging data format
- ❖ Used for most processing pipelines
- ❖ Standardized file storage structure and naming scheme
- ❖ Mapping from DICOM header information to bids naming scheme is non trivial



```
sub-01
└─ses-01
   ├─anat
   │  ├─sub-01_ses-01_acq-tse_T2w.nii
   │  └─sub-01_ses-01_T1w.nii.gz
   └─func
      ├─sub-01_ses-01_task-olfloc_run-01_bold.nii
      ├─sub-01_ses-01_task-olfloc_run-01_events.tsv
      └─sub-01_ses-01_task-olfloc_run-01_physio.tsv.gz
```

## Existing Approaches

HeudiConv, dcm2bids, bidscoin

- ❖ There is no existing approach eliminates the need for the researcher to manually create mappings between DICOM and BIDS formats

## Our Goal

Create a tool that automatically creates mappings from DICOM data to BIDS file naming

These mappings can be inputted into existing converters to conduct the data conversion

## Our Approach

Use **R**etrieval **A**ugmented **G**enerators an extension  of **L**arge **L**anguage **M**odels

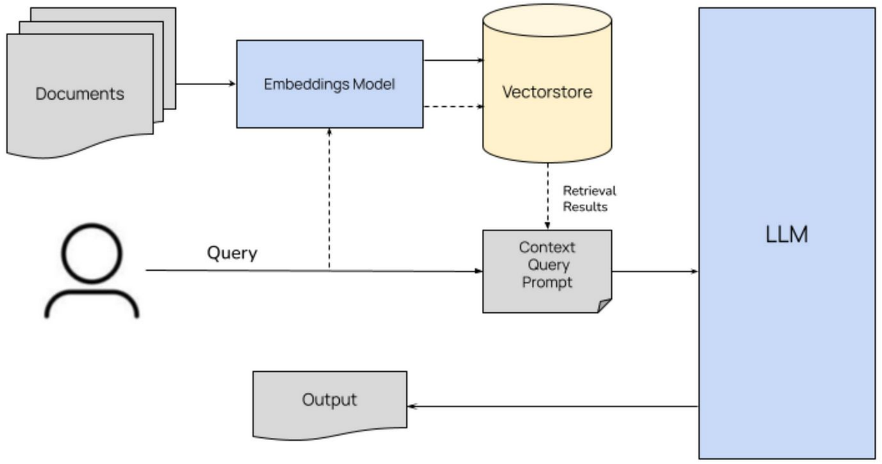
## Large Language Models (LLM)

- ❖ Extremely trendy, LLMs like ChatGPT, Gemma, and Llama are large neural networks based on the transformer architecture.
- ❖ They are trained on enormous amount of natural language data.
- ❖ They take in a “prompt” written and natural language and output a chain of natural language tokens to form a response.

# Retrieval Augmented Generation

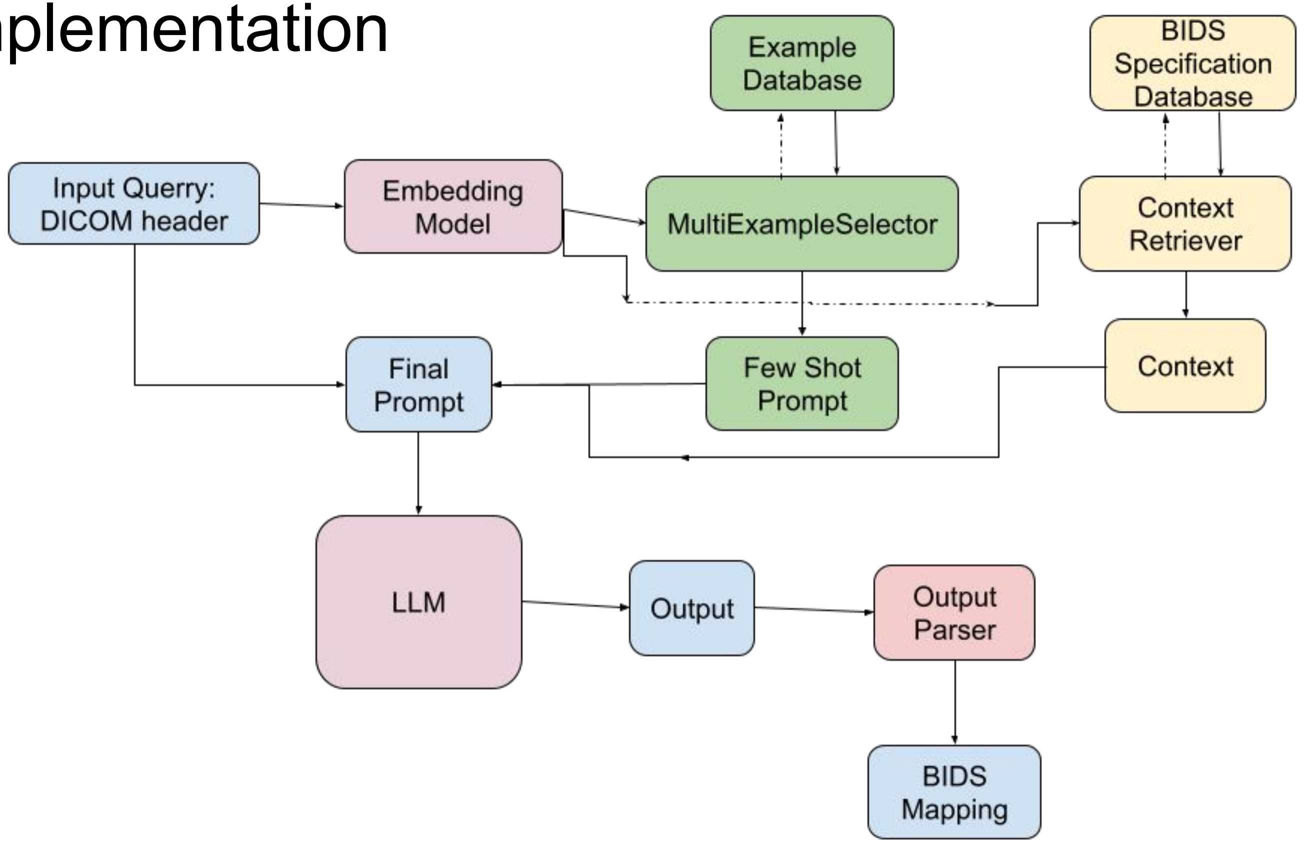
**Problem:** LLMs have a tendency to "hallucinate" i.e. output untrue gibberish

**Solution:** Provide "context" to the prompt which ideally provides all of the information the LLM needs to pull from.



https://www.google.com/url?sa=i&url=https%3A%2F%2Fblog.stackademic.com%2Fmastering-retrieval-augmented-generation-rag-architecture-unleash-the-power-of-large-language-a1d2be5f348c&psig=AOvVaw1AkZ1-bpGzEb5lTqaR5W0r&ust=1720800698407000&source=images&cd=vfe&opi=89978449&ved=0CBEQjRxqFwoTCJie0qywn4cDFQAAAAAdAAAAABAE

## Our Implementation



## Input Query

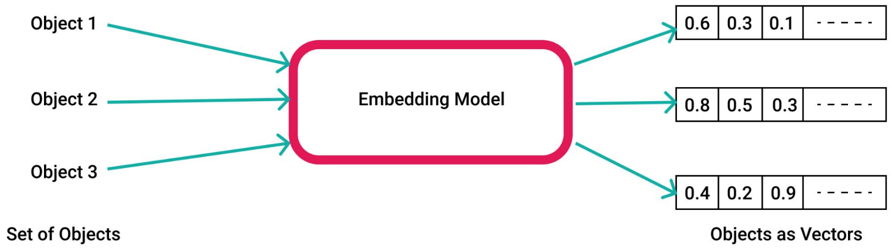
Current Support:

- ❖ Requires one of SeriesDescription and ProtocolName
- ❖ Other fields are optional

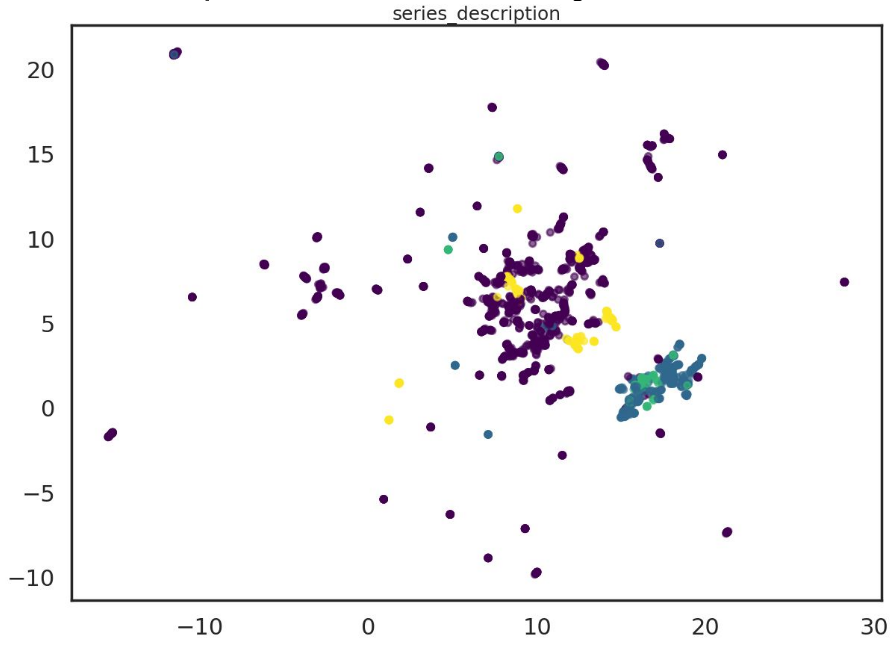
```
"SeriesDescription":"t1_mprage_iso0.6_angulated",
"ProtocolName":"t1_mprage_iso0.6_angulated",
"RepetitionTime":3.1,
"EchoTime":0.00252,
"InversionTime":1.5,
"PulseSequenceType":"NA",
"FlipAngle":5,
"Manufacturer":"Siemens",
"ManufacturersModelName":"Investigational_Device_7T"
```

## Embedding Model

- ❖ Projects strings into vector space
- ❖ Ideally conserves semantic relationships
- ❖ It an be difficult to get meaningful embeddings with some DICOM entries
  - ➢ Some embedding models perform better than others
  - ➢ Expanding Series Descriptions improved performance



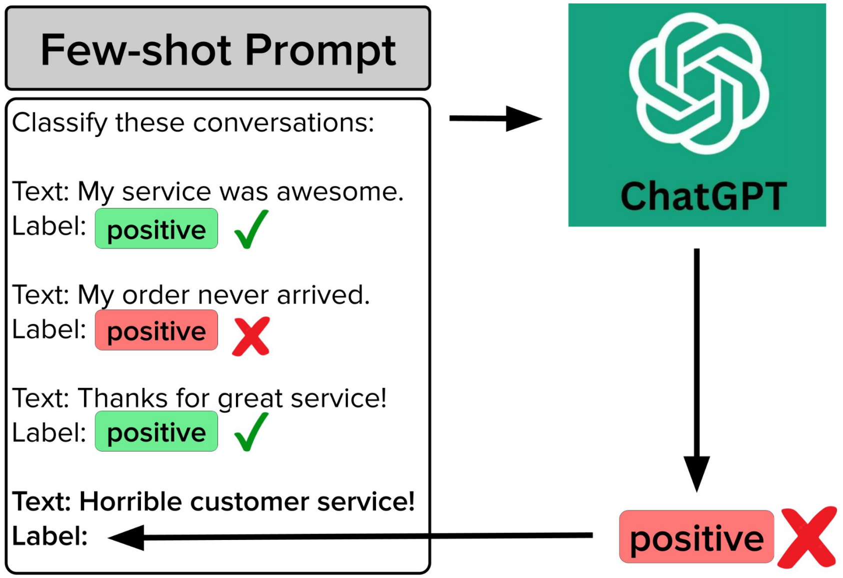
OpenAI : text-embedding-3-small



## ● Example Prompting: Few Shot Prompts

To assist the LLM, examples are added to the prompt using a FewShotPrompt.

Examples demonstrate to the model how it should respond by giving some correct question and answer responses relevant to the input query.


Few-shot Prompt
Classify these conversations:
Text: My service was awesome.
Label: positive ✓
Text: My order never arrived.
Label: positive ✗
Text: Thanks for great service!
Label: positive ✓
Text: Horrible customer service!
Label:
ChatGPT
positive ✗

https://www.google.com/url?sa=i&url=https%3A%2F%2Fcleanlab.ai%2Fblog%2Freliable-fewshot-prompts%2F&psig=AOvVaw3japMRCS7Oz3y10erUk20x&ust=1720803699373000&source=images&cd=vfe&opi=89978449&ved=0CBEQjRxqFwoTCKC_vry7n4cDFQAAAAAdAAAAABAE

## Examples

Examples are constructed in the same way as input queries.

With the addition of the BIDS suffix.

Internally they are stored as pydantic BaseModels

Examples are composed of DICOM header information.

Some fields are strings and some are floats

```
"index":"T1w",
"SeriesDescription":"GIfMI_T1_MPRAGE",
"ProtocolName":"NA",
"RepetitionTime":2.25,
"EchoTime":0.00418,
"InversionTime":0.9,
"PulseSequenceType":"NA",
"FlipAngle":9,
"Manufacturer":"SIEMENS",
"ManufacturersModelName":"NA"
```

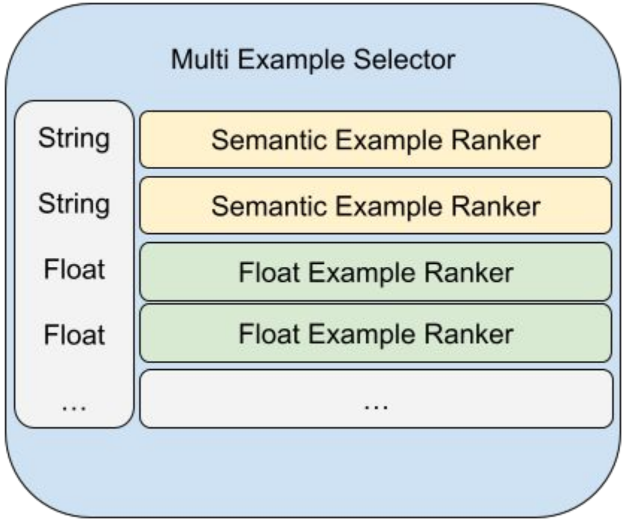
## Multi Example Selector

Problem: Find k most similar Examples in database to the input query.

- Individual fields in the example are treated separately
- Comprised of:
  - **SemanticExampleRanker**: for strings
  - **FloatExampleRanker**: for floats

Future Extension:

- Group some float fields together because they are not independent



## Semantic Example Ranker

- ❖ String fields are embedded into vector space
- ❖ Distance between embedded query string and examples in database is calculated using cosine similarity.
- ❖ Distances are normalized [0,1]
- ❖ Some examples in database may be missing a field. This distance in this case is returned np.nan

## Float  Example Ranker

- ❖ Distances between query float value and example database floats are calculated as the absolute value of the difference
- ❖ Distances are normalized between [0,1]
- ❖ Missing values return np.nan

## Putting it all together

The distance between each example in the database is calculated with the input query.

This is done for each field of the input example

The resulting distance matrix shape (# of examples, #of fields in query)

This is scaled by weight matrix W (1, # fields examples) with values $\in$ [0,1]

The scaled distance matrix is summed yielding vector of shape (# examples, 1)

This is divide by a vector that corresponds to the number of non nan values in each column of the distance matrix

The k examples with the lowest distance are returned.

## Context Retriever

- ❖ The context retriever selects documents parsed from the BIDS specification.
- ❖ These documents are stored in a vector database
- ❖ They are selected based off semantic similarity score between an embedded input query.

## Final Prompt: many parts

System Prompt:

- Instructs the LLM in what it is and how its response should be formatted. Format instructions are derived from pydantic BaseModel that Examples are stored as.

Few Shot Prompt:

- Example responses

Context:

- Retrieved BIDS context

Input Query:

- Finally the inputted DICOM query

## LLMs

- Using Ollama it is very easy to run models locally (if you have the compute necessary)
  - Popular models like Llama 3 and Gemma have distinct *personalities*
  - Output formats can be inconsistent

- OpenAI models
  - With API key very easy to execute
  - GPT3-turbo similar performance, with more consistent and coachable formatting
  - GPT4o is WAY better than all other models I tested

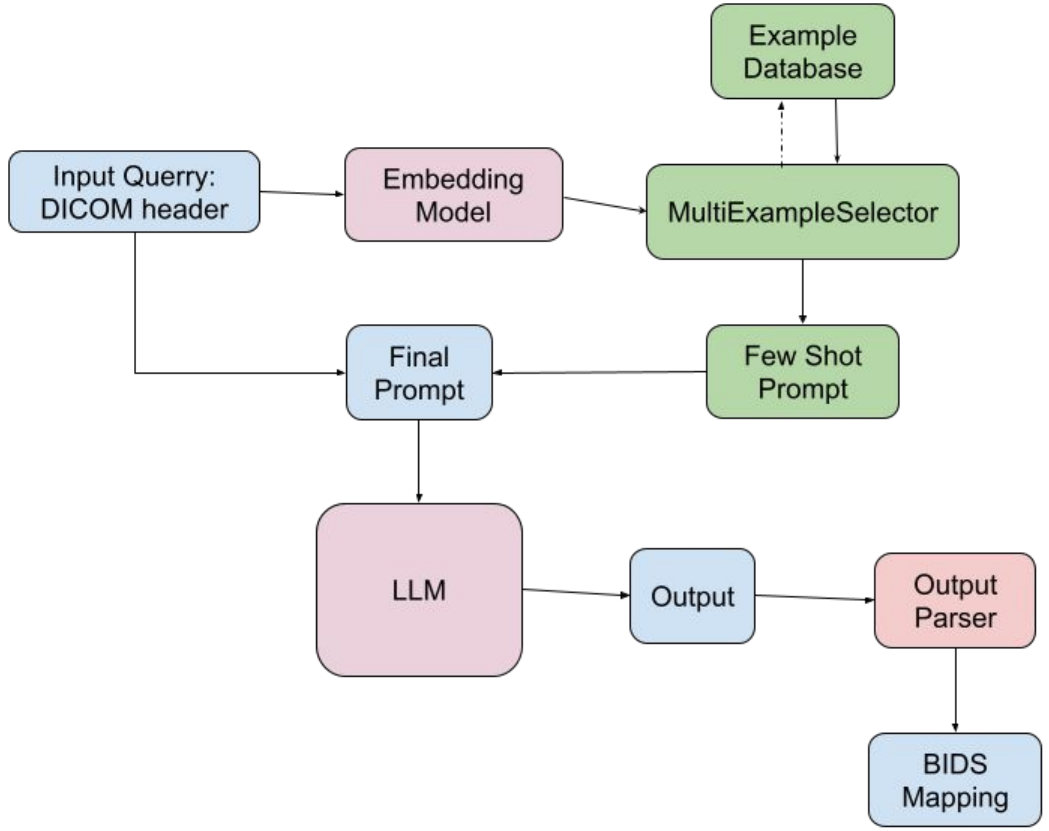
## Output Parser

- LLM outputs can be variable and frustrating to deal with
- Langchain provides some output parsers that expect JSON or pydantic output
- These are often unreliable

## Results and Difficulties

- Initial results were very promising.
- It is hard to wrangle LLMs to be consistent.
- Its like trying to get a child to complete a task. Often it works well, but sometimes they misbehave very unpredictably
- **GPT4o** saved this project.
  - Achieves ~97% correct classification on test set of ~1450 examples
  - It is possible this would be higher in practice, because some examples are very poor

## Final Architecture



## Areas for improvement

- Expand testing
  - So far I have only tested the performance on datasets extracted from openneuro
  - It would be good to test on large datasets like ppmi, etc.
  - It would be difficult to test accuracy on these datasets that have not been converted to bids already
- Develop integration with conversion tools
  - Currently LoxLM can ingest dicom files and output json files associating the corresponding bids suffix with the file
  - This tool would be more useful if it is integrated in the pipeline of other tools such as nipoppy

## Thoughts about LLM development tools

- Ollama
  - Convenient for rapid prototyping, free, and containerizable
  - Open source models are not as performanent as proprietary models
- LangChain
  - Easy to get started, many integrations with other software e.g. vector stores, etc.
  - Many of their implementations seem underdeveloped, documentation is a little all over the place
  - It is possible to write custom classes for most things
- Milvus
  - Pretty straightforward vector database, runs in docker, integrated with LangChain
  - Some periodic issues where I have to completely remove and recreate the database