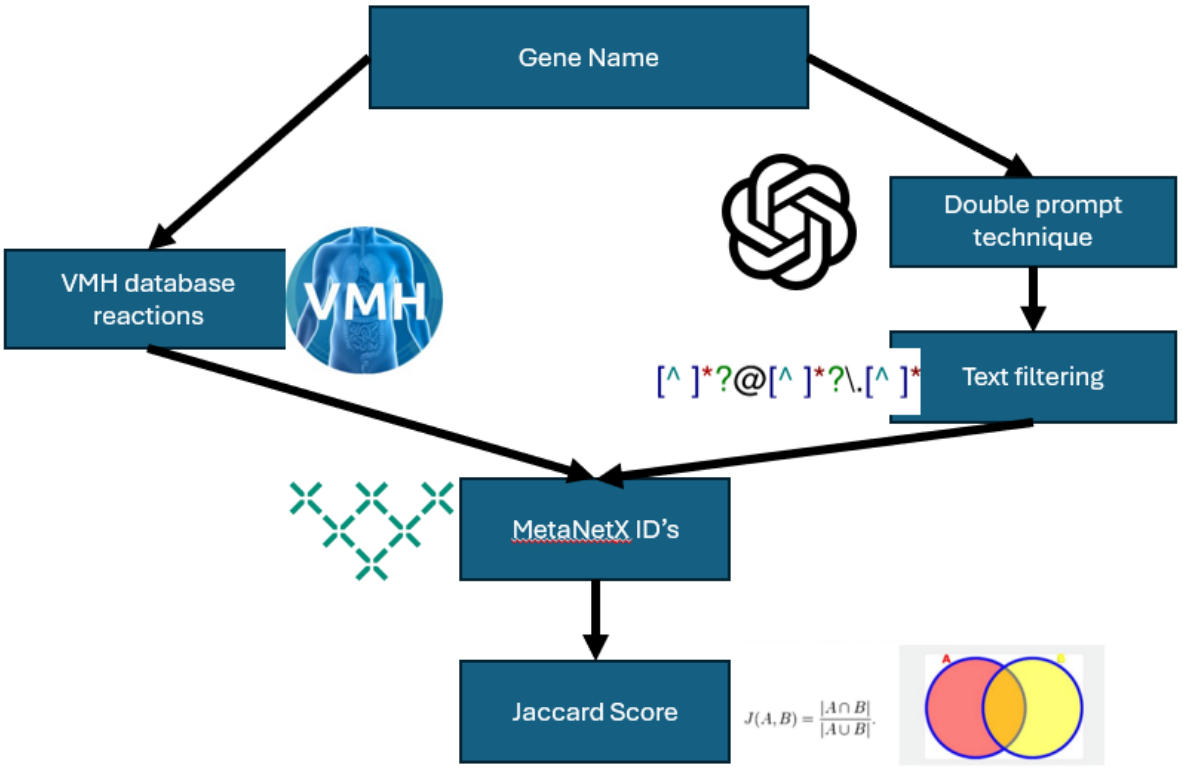
NLP Project – *Investigating the effectiveness of LLMs in aiding the manual curation of metabolic modelling*

Aaron Brennan (19420624)

Overview

Metabolic modelling is a detailed mathematical map of metabolism. They are used to identify metabolic alterations in diseases, predict how changes at the genetic level affect metabolic pathways and improve the yield of desired compounds in biotechnology applications among many other things. The manual curation of these models is both time consuming and expensive, requiring high levels of expertise within a laborious workflow of reading papers and searching databases. The aim of this project is to show that this curation can be aiding by Large Language Models, not as a replacement for human curators but as a helping hand to know where to look and narrow the search process. A subset of this curation is finding the *"metabolic reactions associated with each gene"*, I will be prompting GPT4 on this problem and after processing the output, I will evaluate its performance in relation to the reactions given by a metabolic database called VMH. The project's GitHub page can be found here

Methodology



The methodology can be broken down into a few different sections, such as getting the VMH reactions, prompting ChatGPT, parsing the ChatGPT outputs, combining both the VMH and ChatGPT outputs into MetaNetX ID's for evaluation using the Jaccard score

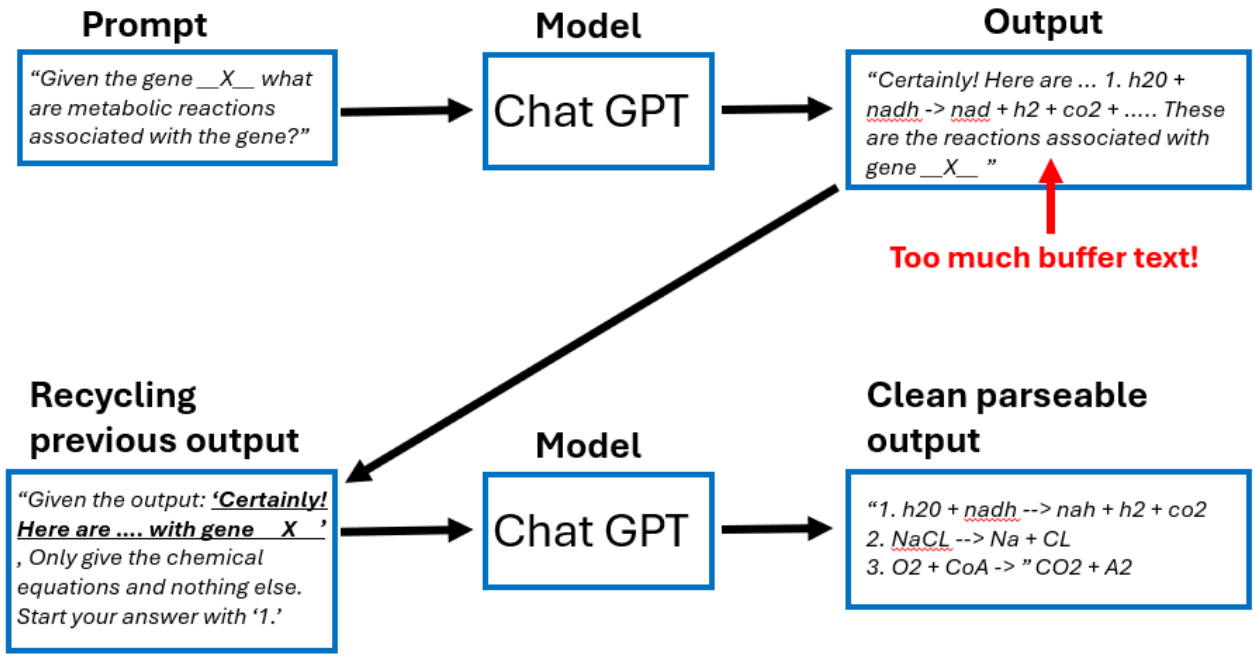
.

VMH Reactions

For each gene, the metabolic reactions associated with it can be found on the VMH database. VMH has an easy-to-use API which makes it simple to collect the reactions associated with the gene. After calling the API, I get a list of reactions, where each element in the list, is a list of metabolites associated with the specific reaction.

```
def get_gene_reactions(ncbi_id):

    reactions = []
    # Initialize a client & load the schema document
    client = coreapi.Client()
    schema = client.get("https://www.vmh.life/_api/genereactions/" + ncbi_id)

    for item in schema['results']:
        reactions.append(item['formula'])

    return reactions
```

Prompting GPT4, using a double prompt technique for more parse-able outputs:



On the other hand, I get the GPT4 predicted reactions by prompting the model in two stages. In the first stage I ask it to give me the reactions associated with a gene. This initial output gives me a lot of buffer text before and after the main piece of information I need. To remove the buffer text, I pass its original answer to itself and ask to only give the reactions and no other piece of text.

The reason I don't just ask for a clean parse-able output in one prompt is that either the model still outputs buffer text or the reactions outputted are worse or wrong usually. This is only from what I have noticed and hasn't been tested rigorously.

When prompting, the temperature is always set to zero to ensure reproducible and reliable outputs. The number of tokens to be outputted by the model will be small (150 tokens) since it is just outputting chemical reactions, this is for efficiency but also to save costs. Here is a sample piece of code for prompting GPT using the OpenAI API:

```
def askGPT4(geneName):
    # what is the protein associated with gene X
    # metabolic reactions catalyzed by the protein Y
    question = "what are all the metabolic reaction(s) catalyzed by the gene: " + geneName + "? only give the chemical reactions and no
other information. Note there may be more than 1 reaction."
    message=[{"role": "user", "content": question}]

    response = openai.ChatCompletion.create(
    model="gpt-4",
    max_tokens=150,
    temperature=0,
    messages = message)

    answer = response['choices'][0]['message']['content']

    return help_format_answer_with_gpt(answer)
```

Parsing GPT4 outputs:

Now that I have the text output from GPT4 I use regex to separate the output into a list of reactions. Each reaction starts with a number then a full stop which can be easily recognised. Also, each new line is a reaction which can be found be searching for "\n".

Then to break up a reaction into a list of metabolites we can separate metabolites from the "+" or "→" symbol in a chemical reaction.

For example, the code to parse a reaction can be seen here:

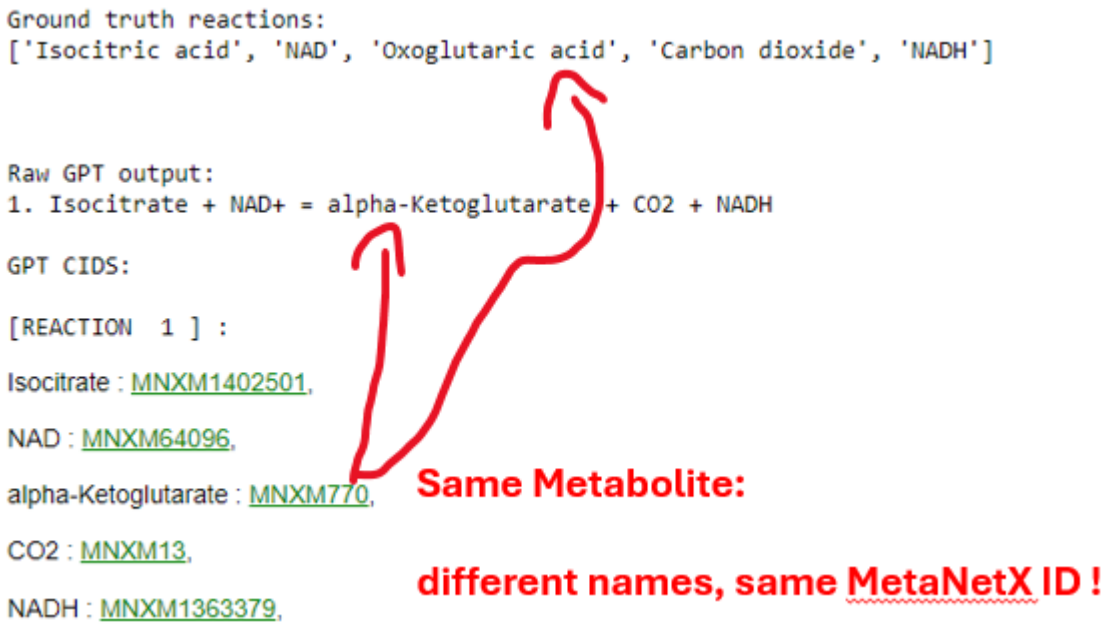
```
def extract_compounds(chemical_formula):
    # Split the formula by spaces and then filter out the '+' and '->' symbols
    elements = [x.strip() for x in chemical_formula.split() if x not in ['+', '->', '<=>']]

    # Extract compounds by removing the part inside brackets, if present
    # and removing any leading numbers
    compounds = [re.sub(r'^\.?\s*', '', elem.split('[')[0]) for elem in elements]

    # Filter out empty strings
    compounds = [comp for comp in compounds if comp]

    return compounds
```

Converting to MetaNetX IDs

Currently we have a list of the reactions from VMH and a list of reactions from GPT4. Different metabolites commonly have many synonyms associated with it e.g. Water and H20. If we just evaluated the names of the VMH metabolites against the names GPT predicted, we will get many false negative matches.

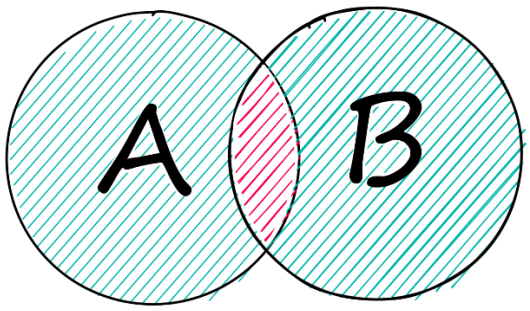
To solve this, we convert the VMH names and GPT names to the one consistent naming system. MetaNetX was chosen as it has a direct match to the VMH naming system and has a look up system which can match the natural language naming given by GPT4 to the correct MetaNetX metabolite.

Here is an example where Oxoglutaric acid has a few different synonyms. GPT4 correctly knew this metabolite was in the reaction but called it by its alternative name; alpha-Ketoglutarate. However, both names corresponded to the same MetaNetX ID resulting in a match.

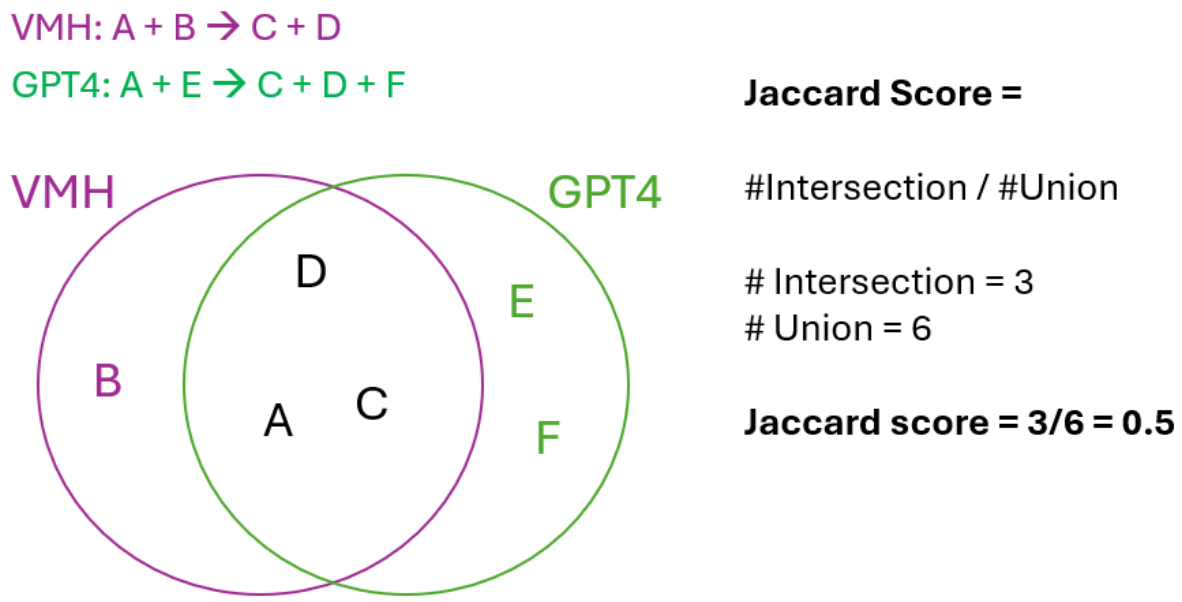


Evaluation using the Jaccard Score

$$Jaccard = \frac{intersection(A, B)}{union(A, B)}$$



To evaluate how similar two reactions are I used the Jaccard Score, this measures the similarities between two sets. In this case the two sets are the metabolites in the reactions I want to compare. Here is an example of what a Jaccard score evaluation would look like for a single reaction:

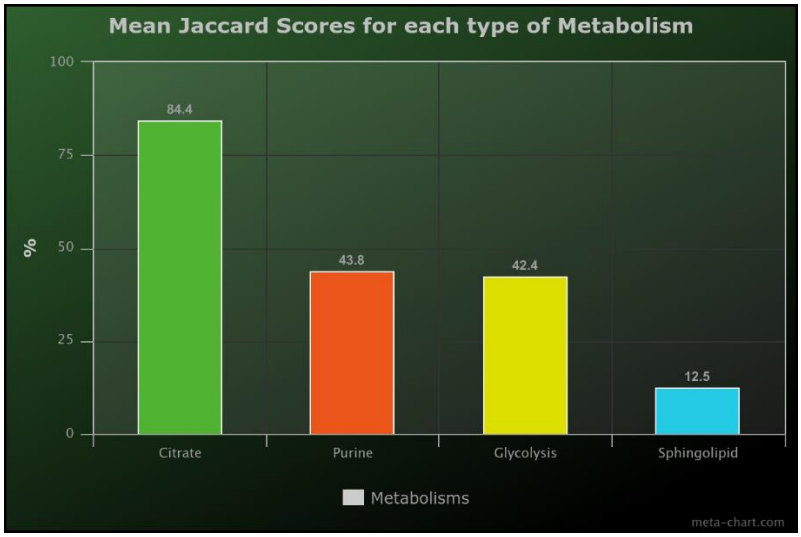


For a given gene the Jaccard scores are averaged across all the reactions associated with that gene. Then for the type of metabolism e.g. Glycolysis, the gene scores are averaged, for all the genes associated with that type of metabolism.

Results

The mean Jaccard score was evaluated across 4 different types of metabolism. Here the results for each type:

- Citrate: 84.6%
- Purine: 43.8%
- Glycolysis: 42.4%
- Sphingolipid: 12.5%



Conclusions and future work.

From the results it is clear to see that the more common metabolisms (e.g. Citrate) discussed in academic literature performed much better than the more obscure metabolisms such as Sphingolipid.

A score of 84% for Citrate metabolism is remarkable considering we are only prompting an LLM. Any curators looking to create a citrate metabolic network will have most of their job done for them or at least verified by the model.

In terms of improvements there is a long way to go, for starters finetuning the model on a large corpus of biochemistry literature will certainly improve results. The use of “tools” is also a promising step forward. Giving the model access to internet search, or a database of biochemistry literature would help the model make accurate, grounded predictions instead of hallucinating it’s predictions as we sometimes see.