An open source PyMedPhys harness for interfacing MOSAIQ® with Claude®

Stuart J Swerdloff, PhD[1] and Simon Biggs[2]

[1]ELEKTA Pty, Wanaka, New Zealand
[2]Anthropic PBC, Wagga-Wagga, Australia

Abstract We introduce tools added to the PyMedPhys project for connecting an anonymized Oncology Information System database, MOSAIQ® , to a Large Language Model, Claude2® for natural language querying. The approach to anonymizing utilizes a custom strategy file provided to the 'pynonymizer' package, and the interface to Claude2® leverages existing capabilities in PyMedPhys and adds a Streamlit based GUI and application logic for querying via Claude2 using Retrieval Augmented Generation of the schema and data dictionary and prompting.

1 Introduction

Large Language Models have achieved significant capabilities and notoriety in the last year. Maintaining the privacy of health information is paramount, and leakage of Private Health Information to a system not within the complete control of the responsible institution must be avoided.
Oncology Information Systems are repositories for both structured and unstructured data that represent many patients' diagnosis, staging, prescription, treatment, and outcomes. Many of the clinical professionals involved in Cancer Care are pressed for the time to provide all the aspects of patient care. Use of Large Language Models to assist in extracting relevant information, correlating data, and generating first draft reports is expected to be of value to the clinicians.
Claude2® is a Large Language Model based on 'Constitutional AI', with a focus on safety, developed by Anthropic.
MOSAIQ® is an Oncology Information System that is used globally at thousands of facilities, developed by ELEKTA.
The current effort aims to place the tools for exploring the possibilities of using LLM as an AI Assistant in Oncology in the hands of cancer clinicians and clinical researchers and leave as much control over the data and prompting in their hands.

2 Materials and Methods

The software and configuration files are available in PyMedPhys [1].
The preliminary step is to anonymise the database that will later be interrogated. Anonymisation is performed using `pynonymizer` (https://pypi.org/project/pynonymizer/ or https://github.com/rwnx/pynonymizer), driven by a strategy file specific to the MOSAIQ® database. The strategy file lists the columns in tables that contain Private Health Information and the strategy for replacing the values (typically using a Faker update). There are approximately 50 columns that contain identifying information.
The database management system is Microsoft SQL Server (2016 or later).
An extract of the strategy file (in YAML format):

```
tables:
  Patient:
    columns:
      Last_Name:
        type: fake_update
        fake_type: last_name
      First_Name:
        type: fake_update
        fake_type: first_name
```

Steps taken to perform the anonymisation of the database:
The starting point was a sample database typically used for engineering development and testing that doesn't contain PHI (all patient and staff identifiers are fictitious), referred to here as "EDB". The "EDB" was backed up to file and named `pynom_msq.bak`, and then restored as `pynom_msq` in to a Microsoft SQL Server instance.
Pynominizer was run, restricted to the `ANONYMIZE_DB` step (given that the editable copy of the database was already present in SQL Server).
The simplest version of this MOSAIQ AI agent has three key components:

- Chat agent for user interaction
- SQL query creation agent which returns queries based on a schema and the conversation history
- A standard database read tool which accepts the queries provided by the SQL query creation agent executes them in read only mode, and then sends the results to the chat agent in order to converse with the user

A version of the above that makes significant improvements upon this simple version is provided within the discussion.
Given the chat agent and SQL query execution components are relatively standard, we will focus on the SQL query generation method here.
In order to direct Claude in MOSAIQ SQL query generation we provide it with both the conversation transcript and the full MOSAIQ SQL schema. In order to provide the schema to Claude in a format that it would expect we asked Claude

how it would design such a schema format. The format we used was:

```
<?xml version="1.0" encoding="UTF-8"?>
<database>
  <table name="ExampleTable">
    <column name="AnID" type="integer" />
    <column name="AnotherColumn"
        type="varchar(50)" />
  </table>
</database>
```

We then used the following SQL query to extract the desired database schema directly from the database itself, and formatted it according to Claude's preference for inclusion within the prompt:

```
SELECT TABLE_NAME, COLUMN_NAME, DATA_TYPE
FROM INFORMATION_SCHEMA.COLUMNS
```

This schema was about 5% larger than the 200k context window for Claude, so a few tables were excluded from the schema for implementation purposes.
In order to direct the SQL agent to produce appropriate queries the following key components of the prompt were utilised:

- System prompt
- Database schema
- Conversation transcript
- Reminder of key requirements
- Forced initial completion

Although there was more to it, the key parts of the system prompt were the following (shortened for brevity):

> You are an MSSQL AI agent. You respond only with valid Microsoft SQL Queries encompassed within <query> tags. You always provide 10 unique and diverse queries.
>
> Some queries that you request may not return a result, and some tables within the schema may not be relevant to your needs. So make sure that each of your queries targets different tables within the database so that you get good coverage over possible solutions and are able to return something meaningful.
>
> ...
>
> All queries assume the following database schema: schema

After the schema has been provided, the conversation transcript is then also included. After all of this a reiteration of the task is given as so:

> You respond only with valid Microsoft SQL Queries encompassed within <query> tags. All queries assume that the database is designed according to the provided schema within the <database> tags that was provided within your system prompt. You are to provide exactly 10 unique and diverse queries. Make sure that each of your queries targets different tables within the database.

And finally we put "words in the assistant's mouth" by starting their response for them, ensuring adherence to our output schema that we subsequently parse to actually undergo the queries ourselves:

> Assistant: <query> SELECT DISTINCT

The assistant will then undergo its textual completion directly after the DISTINCT keyword in order to create its first query and then subsequent queries.
As an evaluation, for this agent, the following questions were asked and the result was then recorded:

- Please find the patient ID that has had the most radiotherapy treatments
- Please find the most common cancer diagnosis that patients have had
- What is the most common number of fractions used to treat patients? Do not give a range, instead, find exactly the most common fraction number.

The agent was only given one shot in each attempt and was not given the opportunity to refine or further verify its answer. For all testing we used the publicly available version of Claude, version 2.1 via the Anthropic Python PyPI package version 0.14.0. The Mosaiq DB utilised was a testing instance that corresponds to the same schema as that used by MOSAIQ version 3.1.3.

3 Results

See tables 1, 2, and 3 for a summary of the results provided by the agent for each of the provided questions. Of note, the only question for which the majority of the answers were the correct one was for the question "Please find the most common cancer diagnosis that patients have had". An example SQL query that the agent created to answer this question was:

```
SELECT TOP 10 tpg.Description AS CancerType,
    COUNT(m.MED_ID) AS NumDiagnosed
FROM Medical m JOIN Topog tpg ON tpg.TPG_ID =
    m.TPG_ID GROUP BY tpg.Description ORDER BY
    NumDiagnosed DESC
```

Case	Count
Agent said it couldn't find a result	18
31587	2
10003	3
31215	1

Table 1: Results for "Please find the patient ID that has had the most radiotherapy treatments"

For the last question regarding fractionation, in some example responses, it becomes clear that in those particular cases Claude misunderstood the question:

> Based on the queries and analysis, the most common number of fractions used to treat patients is 1 fraction per day. This was determined by:
>
> - Calculating the average number of fractions per day across all patient treatments, which came out to be 1 fraction per day on average.
> - Analyzing the number of treatments (fractions delivered) grouped by number of fractions per day. This showed 1 fraction per day was the most common configuration, used for 54 different sequences of treatment fractions.

Another which was more successful both understood the question and was able to explain its processes:

> Based on the queries I tried, 10 fractions is the most common number used to treat patients in the database. This was determined by:
>
> - Grouping the Fractions field from the Site table and counting the number of patients for each fractions value.
> - Ordering the result by the number of patients in descending order.
> - Taking the top result which showed 10 as the most common number of fractions with 109 patients.

4 Discussion

As can be seen from the results, the agent as built above is inconsistent and while it can find answers to some classes of questions, others it fails dismally.
A second, more complicated version of the above, which aims to improve both speed and quality of the SQL query creation agent splits the SQL creation agent up into the following steps:

- Table selection agent, by first only providing the table name schema and asking it to select just a few tables to work within, the context usage is significantly reduced

Case	Count
Agent said it couldn't find a result	21
Malignant neoplasm of prostate	26
Provided diagnosis codes	1
The "NULL" diagnosis (no diagnosis entered)	3
Prostate cancer	1
Bladder cancer	1
Malignant neoplasm	1
Answered wrong question	1

Table 2: Results for "Please find the most common cancer diagnosis that patients have had"

Case	Count
Agent said it couldn't find a result	20
0 fractions	1
1 fraction	4
10 fractions	3
14 fractions	1
23 fractions	1
152 fractions	1
Answered wrong question	1

Table 3: Results for "What is the most common number of fractions used to treat patients? Do not give a range, instead, find exactly the most common fraction number"

- Query creation agent that is only given the schema for the tables selected by the table selection agent

Then, due to the significantly reduced token context utilised we would be able to run this two step query creation agent about 10 times in parallel with a voter agent selecting the best and worst queries based on the results, and then feed back the previously successful queries into subsequent multiple reruns of the SQL query creation agent in a reflexion like approach [2]. The result of which being both a reduced total time to execute (due to not having the entire schema passed) and a hypothesised improvement on completion quality. This hypothesis could be further tested by repeating the above evaluations and iterating for an optimum configuration.

5 Conclusion

We present here an open source harness for interfacing MOSAIQ® with large language models. As well as providing a means to undergo anonymisation of the underlying database. We also provide an example approach for the evaluation of the quality of the underlying model and its interfacing agent. In this case although valid and relevant SQL queries were able to be produced, in its current implementation the agent set up was only able to answer some classes of questions, and in those cases it still wasn't able to do so reliably.
Nevertheless the open source approach, the significant room

for improvement in the agent infrastructure, as well as improvements in the underlying language model being predicted through the scaling laws [3, 4] shows potential for large language model interfaces to radiation oncology information systems.

References

[1] S. Biggs, M. Jennings, S. Swerdloff, et al. "PyMedPhys: A community effort to develop an open, Python-based standard library for medical physics applications". *Journal of Open Source Software* 7.78 (2022), p. 4555.

[2] N. Shinn, F. Cassano, A. Gopinath, et al. "Reflexion: Language agents with verbal reinforcement learning". *Thirty-seventh Conference on Neural Information Processing Systems*. 2023.

[3] J. Kaplan, S. McCandlish, T. Henighan, et al. "Scaling laws for neural language models". *arXiv preprint arXiv:2001.08361* (2020).

[4] J. Hoffmann, S. Borgeaud, A. Mensch, et al. "Training compute-optimal large language models". *arXiv preprint arXiv:2203.15556* (2022).