FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Design of LLM prompts for iterative data exploration

Student: Mikoláš Fromm
Supervisor: Mgr. Tomáš Petříček, Ph.D.
Department of Distributed and Dependable Systems

Introduction

Iterative data exploration simplifies working with complex datasets such as databases and data banks. Instead of entering a complete query at once, users make decisions and enter partial steps gradually, which increases the likelihood of success. This approach can be generalized as a sequence of decision problems over a tree of possible paths, requiring semantic understanding at each point. Given the current capabilities of large language models (LLMs) in text semantics, we aim to integrate LLMs into this decision process. Additionally, we conduct experiments to enhance the success of LLMs responses.

Goals

1. Implement a C# web-based, LLM boosted, iterative assistant for tabular data exploration.
2. Propose prompting strategies to communicate effectively with LLMs.
3. Create a simulation framework to systematically test the scoring of the prompts.
4. Test and evaluate the prompting methods.

C# Implementation

The application allows the user, along with their objective, to upload their own CSV tabular file. Using the pre-implemented operations, the user can iteratively transform the inserted data into a new view, which is displayed to the user during query building. At each step, the LLM agent is prompted in the background to select the best next step based on the specified objective.

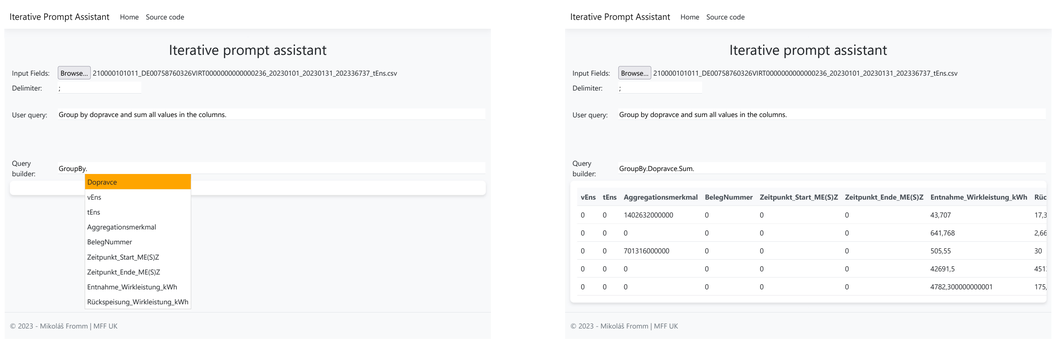

Figure 1: Query building with an LLM-whispered move (orange) on left, the transformed data preview on right.

Prompts proposal

We found out that the way an LLM is prompted strongly affects the quality of the responses. Therefore, through experiments, we verified whether its success can be improved using the following types of prompts:

1. **Step-by-step**: Numbered names of all subsequent actions, each on a new line. Serves as a baseline solution.
2. **Look-ahead-x**: Numbered names of all subsequent actions, recursively supplemented (up to depth $X - 1$) with the following actions, placed on new lines, but without numbering.
3. **Look-ahead-x (inline)**: Future actions are inserted in brackets (...) on the same line. Tests if reduced tokens affect the score.
4. **Keyword gen. and match**: LLM generates N relevant keywords for searching. The tree is filtered to include nodes with similar names, potentially reducing token usage.

Testing framework

To evaluate prompting strategies, we model available options as a tree and use LLM to find a path through the tree. We build two testing trees (Eurostat database, user's filesystem) and use various prompting strategies to choose the next best subtree. We compare recommendations with manually created test cases.

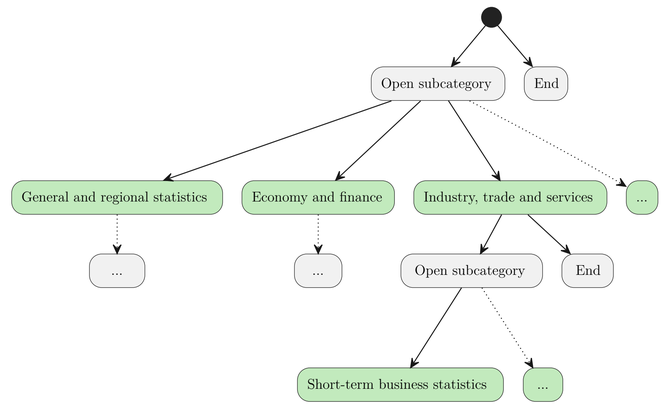

Figure 2: Tree representation of the iterative approach in data selection environment.

Prompt examples

The following examples illustrate the *step-by-step* prompt form:

```
"User initial input is: Group all people by their
hometowns and show the sums for each city.

The query built so far: GroupBy

---> Choose one of the following headers
by which you want to group the dataset:
> [0] FirstName
> [1] LastName
> [2] Age
> [3] Department
> [4] HomeTown

Answer the appropriate number!"
```

Figure 3: Prompt for the LLM can be created dynamically (from the header of the file in this case).

```
"User initial input is: Group all people by their
hometowns and show the sums for each city.

The query built so far: GroupBy.HomeTown

---> Choose one of the following Aggregations
you want to apply to the grouped dataset:
> [0] Sum
> [1] Avg
> [2] Concat
> [3] CountDistinct
> [4] CountAll

Answer the appropriate number!"
```

Figure 4: Prompt is usually created from the predefined set of implemented operations / options.

Results

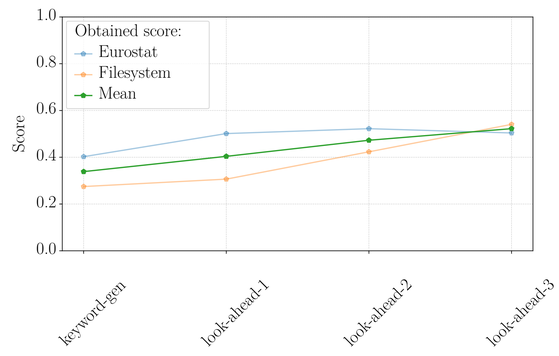

Figure 5: The impact of context size on the final score.

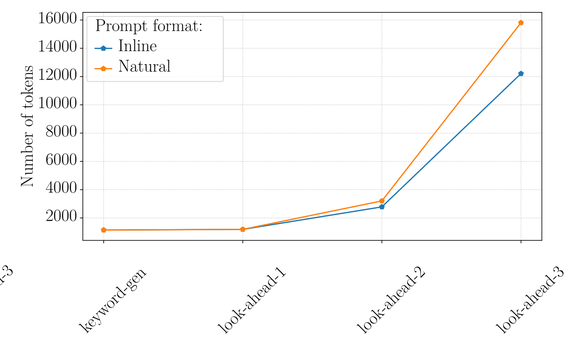

Figure 6: The impact of context size on the total token usage.

Discussion

We have verified that the proposed methods *look-ahead-x* significantly increase the accuracy of LLM responses, but they come at the cost of using a much larger number of tokens. On the other hand, the *keyword gen. and match* method used the least tokens but was also the least successful.