Mirco Gijsens 0591381

Report Challenge 1: LLM-to-AutoML query translator.

1. Introduction

The aim of this report is to provide a means to parse generic English language query's to AutoML tokens that can be read by the AutoML system.
To facilitate this process we use LLM's, as requested by the challenge.

Large language models or LLM for short is an AI program that can recognize and generate text. They are trained on a huge amount of data and uses deep learning in order to understand how sentences, words, etc function. (Cloudflare, sd)

The challenge gave the following approach to consider: First use LLM's to rewrite the queries into easily parsable English only using specific terms. And then use this intermediate step to parse it into AutoML tokens.

At face value this approach seems valid (which it is), but after doing some research and trying things with LLM's there seems to be a simpler way than using intermediate steps to parse the queries.

2. Approach

Instead of using Intermediate steps to rewrite and then parse the queries, I make use of prompt engineering to pass a specific prompt to an LLM to get the desired outcome.

LLM's can do a lot, even write SQL queries as demonstrated here:
https://python.langchain.com/docs/use_cases/sql/prompting/.
So, it is not a stretch that it can transform queries into tokens directly.

To make this possible I make use of Ollama[1] which makes it possible to run LLM's locally on your computer.

3. Implementation

The basis of the project is to have a good enough prompt that makes the LLM's transform the queries. Luckily Ollama provides a means to do this more easily and consistently.

Ollama makes it possible to create your own models based on other models where you can specify the behavior of your created model.

To do this we create a so called Modelfile[2]. The Modelfile is the blueprint of your model. Here you can specify some things like parameters, messages, etc. that your model will use.

Choosing a base model

---

[1] https://github.com/ollama/ollama

[2] https://github.com/ollama/ollama/blob/main/docs/modelfile.md

To start off we need to choose a model that we will base our own model on. This can be a plethora of models but for this project I used mistral. Here you can find some valid reasons to use Mistral: https://www.promptingguide.ai/models/mistral-7b. It is not necessary to use mistral yourself, you could use another base model but do note that my solution was specifically build on mistral and my not have the same results for other models.

```
FROM mistral
```

Adding parameters

Parameters are a way to tweak a few behaviors of the model. For example, how creative it will be, etc.

For my implementation I set three parameters (Ollama, sd):

- temperature: The temperature of the model. Increasing the temperature will make the model answer more creatively.
- mirostat_eta: Influences how quickly the algorithm responds to feedback from the generated text. A lower learning rate will result in slower adjustments, while a higher learning rate will make the algorithm more responsive. (Default: 0.1)
- Seed: Sets the random number seed to use for generation. Setting this to a specific number will make the model generate the same text for the same prompt. (Default: 0)

I used these parameters to make sure that my model would be flexible and still would generate great responses. You could play with these parameters to create different behaviors for your model.

```
PARAMETER temperature 0
PARAMETER mirostat_eta 0.5
PARAMETER seed 42
```

I set the temperature to 0.3 because I would like consistency in the responses given and the model will not give wildly different responses. When increasing this there is a greater variation of responses possible. So, when you want more diverse responses, you can increase the temperature. For now, 0.3 should be adequate.

I set mirostat to 0.5 so the model will learn faster from the feedback and give better responses.

Adding SYSTEM

This specifies the system instruction message to be used in the model template. Basically you use this to specify how the model should behave.

In here I specify what the model can do, which datasets there are and the list of the AutoML tokens that the model can use.

```
SYSTEM """You rewrite queries into sequential instructions using only the symbols described
below and the data of the given datasets and its columns. You can only use the symbols that
are necessary to rewrite the query. Do not use symbols or datasets end columns that are not
necessary. The instructions must be separated by a semicolon.

Datasets with name and columns:

- "objects": id (integer), surface in square meters (float), height in square meters
(float), mass in kilograms (float).
- "buildings": id (integer), age (integer), EPC (string), zipcode (string)

List of allowed symbols with name and description:

- "LoadData": Load a dataset.
- "SplitTrainTestData": split a dataset into a training and testing dataset.
- "CalculateVolume": calculate volume as a product of a height and surface.
- "CalculateDensity": calculate the density as the mass divided by volume.
- "CalculateAverage": Calculate average.
- "RemoveMissingRows": remove rows with missing data.
- "RemoveMissingColumns": remove columns with missing data.
- "ImputeValuesColumn": impute values for a column.
- "ClusterData": cluster data with a clustering algorithm (algorithm not important).
- "KnnRegression": learn a KNN-based regression model (model is just training data and a
similarity metric).
- "ClusterCount": count clusters out of a clustering result.
- "RegressionMSEError": calculate the mean squared error of a regression model given a test
dataset.

The output of an individual instruction should be:
symbol('symbol of the previous instruction' and / or 'data' separated by a comma as
arguments).

Note that before you can use a symbol as argument of another instruction it needs to be
initialized beforehand as a separate instruction. Every symbol needs to have its own
separate instruction. You can't use for example: "LoadData(objects);
CalculateDensity(CalculateVolume, mass in kilograms (float))"; but you need to use this:
"LoadData(objects); CalculateVolume(LoadData, surface in square meters (float), height in
square meters (float)); CalculateDensity(CalculateVolume, mass in kilograms (float))"
because volume was not yet calculated.

The start of the instructions should use #start# end the end #end#.

"""
```

If you want you can expand the list of tokens or datasets that the model can use you can add them here.

The system message is basically the prompt that tells the LLM how it should rewrite the queries. You can play with this. Add some more descriptions, etc.

I am aware that this prompt could be written more compact or differently, but that is a question of experimentation to optimize it. For now, this prompt works adequate.

Giving examples:

The last thing we will do in the Modelfile is giving it some examples. This is handy because now the model has some examples it can base its responses on. This makes the model output more accurate. You need to be careful with giving examples. Wrong examples can make the LLM give wrong answers and to many examples will restrict the creative freedom of the LLM. To make the model rewrite the queries more accurately you can give an example for each token. Here I did not do that. I only give a few examples and it still gives adequate results.

```
MESSAGE user In the objects dataset, how many groups based on volume can be
found?
MESSAGE assistant #start# LoadData(objects); CalculateVolume(LoadData, surface
in square meters (float), height in square meters (float));
ClusterData(CalculateVolume); ClusterCount(ClusterData) #end#.
MESSAGE user How many types of objects from the objects dataset have the same
mass?
MESSAGE assistant #start# LoadData(objects); ClusterData(LoadData, mass in
kilograms (float)); ClusterCount(ClusterData) #end#.
MESSAGE user Calculate the density in the objects dataset.
MESSAGE assistant #start# LoadData(objects); CalculateVolume(LoadData, surface
in square meters (float), height in square meters (float));
CalculateDensity(CalculateVolume, mass in kilograms (float)) #end#.
```

4. Thought process to get to the solution

At the start of the project, I found 2 possible avenues to solve the problem.

- Named Entity Recognition
- LLM prompting

I first briefly looked at Named entity recognition[3] to basically assign labels to various objects inside a given text. My first Idea was to first create a set of labels that the NER would use to assign objects and then use those labels to parse it to correct AutML tokens.
I quickly abandoned the idea after I found an easier way to do the transformation using prompting after I read the following article:

---

[3] https://spacy.io/usage/linguistic-features#named-entities

https://python.langchain.com/docs/use_cases/sql/prompting/ which presented a way to create SQL statements using prompts to LLM's.

I abandoned the idea of NER (Named entity recognition) early on and went all in with prompting. This does not mean in my opinion that NER would not have been a viable solution but in this instance, I chose prompting.

5. Prompting

I first started with the original task by trying to create intermediate instructions for the following question: "Given the objects dataset, how many types of objects can be found, according to their volume?".

So, the first prompt I tried was something of the following:

"You are an expert in rewriting questions in short instructions to solve them. Given an input question, rewrite the question into a set of short sequential instructions separated by a comma. You can only use the following terms to construct the instructions: load, dataset, calculate, volume, cluster, regression, data, count, clusters, a, and, perform, calculate, using, id, surface, height, mass. You can also use the column names of the datasets below. Never use all given terms and columns you must only use the terms and columns that are needed to rewrite the given questions. And only use the given terms and columns nothing else.

Only use the following datasets:

Object dataset:
columns: age, weight, Height, width, volume.

Rewrite following question: 'Given the objects dataset, how many types of objects can be found, according to their volume?'".

Note that I did not fiddle with parameters or gave examples.

This gave me the following answer:

"Load Object dataset,
Calculate volume for each object using height, width, and length columns,
Count distinct volumes in the dataset,
Determine number of unique counts as number of object types."

A reasonable answer with correct steps. This result gave me the confidence that prompting would be a valid strategy.

Next, I tried the same strategy but using a less elaborate prompt:

"Rewrite the following question into short instructions separated by a comma.
Question: 'Given the objects dataset, how many types of objects can be found, according to their volume?'
You can only use the following terms: load, dataset, calculate, volume, cluster,

regression, data, count, clusters, a, and, perform, calculate, using, id, surface, height, mass. No other terms can be used. "

This gave the following answer:

"1. Load objects dataset,

2. Calculate volume for each data point,

3. Perform clustering or regression analysis, using id, surface, height, and mass,

4. Count unique clusters or regression outputs, a,

5. Count clusters or regression results is the number of object types,

6. Output result: count of clusters or regression results."

Which again showed promise. It also told me that LLM can find the meaning of queries rather easily and could give reasonable instructions.

After this I decided to try to remove the need for intermediate instructions and just gave back the final solution.

To do this I used a Modelfile (see implementation) so I had more control about what the model would give back.

The first model as a proof of concept was the following:

```
FROM mistral

SYSTEM """You rewrite queries into sequential instructions using only the
symbols described below. You can only use the symbols that are necessary to
rewrite the query. The instructions must be separated by a comma.
List of allowed symbols with name and description:

- "LoadData": Load a dataset.
- "SplitTrainTestData": "split a dataset into a training and testing dataset"
- "CalculateVolume": calculate volume as a product of a height and surface
- "CalculateDensity": calculate the density as the mass divided by volume:
- "CalculateAverage": Calculate average of the given column or data.
- "RemoveMissingRows": remove rows with missing data
- "RemoveMissingColumns": remove columns with missing data
- "ImputeValuesColumn": impute values for a column
- "ClusterData": cluster data with a clustering algorithm (algorithm not
important)
- "KnnRegression": learn a KNN-based regression model (model is just training
data and a similarity metric)
- "ClusterCount": count clusters out of a clustering result
- "RegressionMSEError": calculate the mean squared error of a regressionmodel
given a test dataset

The ouput of the instruction should be:
```

```
symbol - "the name of columns or data or dataset"
"""
```

This model only had a message object and had a list of the AutoML tokens and their description.

This gave the following answer back:

"LoadData: dataset, SplitTrainTestData: dataset, CalculateVolume: dataset[symbol], ClusterData: Volume[symbol], ClusterCount: result"

This gave me the confidence that LLM could directly translate the queries to input tokens.

Of course, the output was not 100% the way I had Imagined (but still pretty good). To make sure that I got the correct output I used messages to give directions to the model.

```
FROM mistral

SYSTEM """You rewrite queries into sequential instructions using only the
symbols described below and the data of the given datasets and its columns.
You can only use the symbols that are necessary to rewrite the query. The
instructions must be separated by a comma.

Datasets with name and columns:

- "objects": id (integer), surface in square meters (float),height in square
meters (float), mass in kilograms
  (float).

List of allowed symbols with name and description:

- "LoadData": Load a dataset.
- "SplitTrainTestData": "split a dataset into a training and testing dataset"
- "CalculateVolume": calculate volume as a product of a height and surface
- "CalculateDensity": calculate the density as the mass divided by volume:
- "CalculateAverage": Calculate average of the given column or data.
- "RemoveMissingRows": remove rows with missing data
- "RemoveMissingColumns": remove columns with missing data
- "ImputeValuesColumn": impute values for a column
- "ClusterData": cluster data with a clustering algorithm (algorithm not
important)
- "KnnRegression": learn a KNN-based regression model (model is just training
data and a similarity metric)
- "ClusterCount": count clusters out of a clustering result
- "RegressionMSEError": calculate the mean squared error of a regressionmodel
given a test dataset
```

```
The output of an individual instruction should be:
symbol(input symbols and/ or data separated by a comma).

The start of the instructions should use this #start# end the end #end#.
"""
MESSAGE user In the objects dataset, how many groups based on volume can be found?
MESSAGE assistant #start# LoadData(objects); CalculateVolume(LoadData, surface in square meters (float), height in square meters (float)); ClusterData(CalculateVolume); ClusterCount(ClusterData) #end#.
MESSAGE user How many types of objects from the objects dataset have the same mass?
MESSAGE assistant #start# LoadData(objects); ClusterData(LoadData, mass in kilograms (float)); ClusterCount(ClusterData) #end#.
MESSAGE user Calculate the density in the objects dataset.
MESSAGE assistant #start# LoadData(objects); CalculateVolume(LoadData, surface in square meters (float), height in square meters (float)); CalculateDensity(CalculateVolume, mass in kilograms (float)) #end#.
```

This gave back the following answer:

"#start# LoadData(objects); CalculateVolume(LoadData, surface in square meters (float), height in square meters (float)); ClusterData(CalculateVolume); ClusterCount(ClusterData) #end#.

This gave back what I would like to see, namely the AutoML token and the input parameters it uses for the token to work. This shows the importance of using example messages, so that the model has a framework it can use to answer questions.

This model was then refined by testing different variations of the model until it became the Modelfile in the section Implementation.

Note that the Modelfile in the implementation is a model that works adequate, but it could always be improved upon.

6. Notes about the output

As output I do not only show de AutoML symbols but also the columns or data sources that the Symbol need to use.

The output has this form: "symbol('symbol of the previous instruction' and / or 'data' separated by a comma as arguments)." The instructions are a chain that always takes the output of the previous instruction and other arguments. To get this get the right output, examples are very important, because the LLM will structure its response like the given examples to get a uniform output.

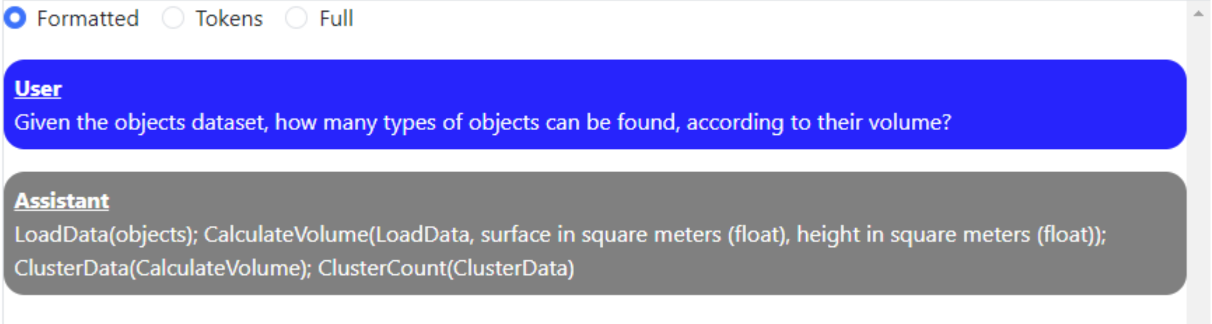
This is a lot of information and maybe a bit too much. But I wanted to show that LLM can do many things.

A thing to note is that the output tends to nest symbols even when the symbol has not been used before. For example, "CalculateDensity(CalculateVolume, mass)". Here CalculateVolume is treated as a nested symbol instead of a separate instruction. This fixes itself when AutoML gives back an error saying that CalculateVolume is not initialized yet and the LLM rewrites the output. More on this in the following section.

7. Examples and features.

In this section I will show some examples and some comments about the example.

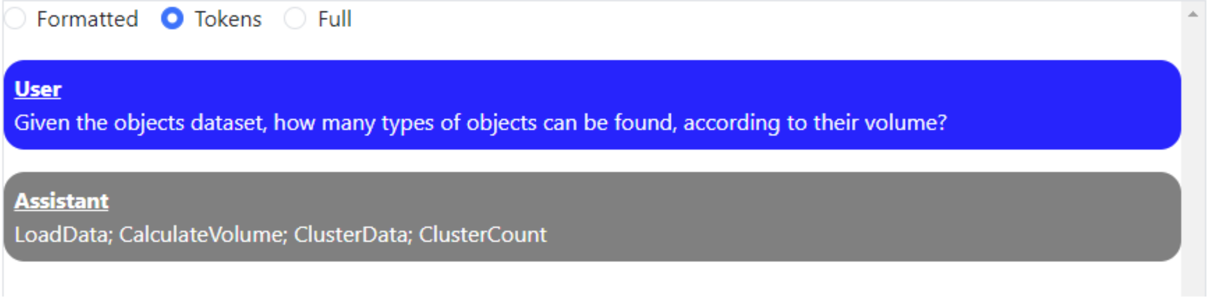
- Given the objects dataset, how many types of objects can be found, according to their volume?



● Formatted ○ Tokens ○ Full

User
Given the objects dataset, how many types of objects can be found, according to their volume?

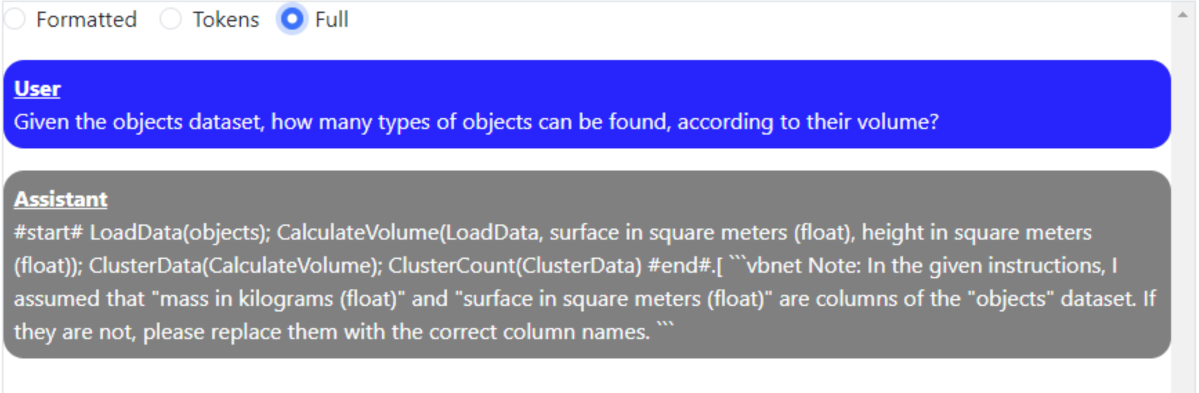
Assistant
LoadData(objects); CalculateVolume(LoadData, surface in square meters (float), height in square meters (float)); ClusterData(CalculateVolume); ClusterCount(ClusterData)



○ Formatted ● Tokens ○ Full

User
Given the objects dataset, how many types of objects can be found, according to their volume?

Assistant
LoadData; CalculateVolume; ClusterData; ClusterCount



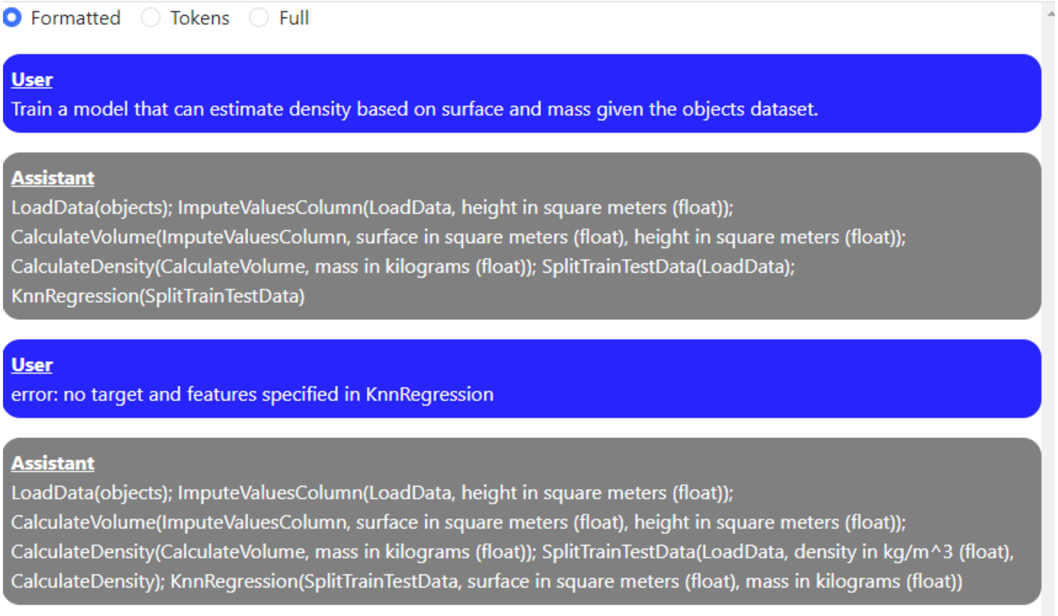
○ Formatted ○ Tokens ● Full

User
Given the objects dataset, how many types of objects can be found, according to their volume?

Assistant
#start# LoadData(objects); CalculateVolume(LoadData, surface in square meters (float), height in square meters (float)); ClusterData(CalculateVolume); ClusterCount(ClusterData) #end#.[```vbnet Note: In the given instructions, I assumed that "mass in kilograms (float)" and "surface in square meters (float)" are columns of the "objects" dataset. If they are not, please replace them with the correct column names. ```

As shown the LLM model correctly transforms the queries on the first try. A reason why this is correct from the first time is because a similar query was given as example in the Modelfile.

- Train a model that can estimate density based on surface and mass given the objects dataset.



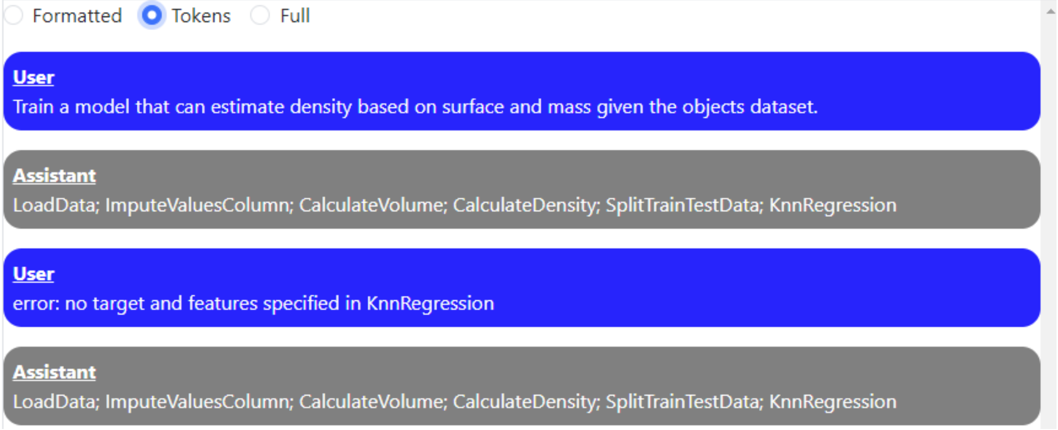
● Formatted ○ Tokens ○ Full

User
Train a model that can estimate density based on surface and mass given the objects dataset.

Assistant
LoadData(objects); ImputeValuesColumn(LoadData, height in square meters (float)); CalculateVolume(ImputeValuesColumn, surface in square meters (float), height in square meters (float)); CalculateDensity(CalculateVolume, mass in kilograms (float)); SplitTrainTestData(LoadData); KnnRegression(SplitTrainTestData)

User
error: no target and features specified in KnnRegression

Assistant
LoadData(objects); ImputeValuesColumn(LoadData, height in square meters (float)); CalculateVolume(ImputeValuesColumn, surface in square meters (float), height in square meters (float)); CalculateDensity(CalculateVolume, mass in kilograms (float)); SplitTrainTestData(LoadData, density in kg/m^3 (float), CalculateDensity); KnnRegression(SplitTrainTestData, surface in square meters (float), mass in kilograms (float))



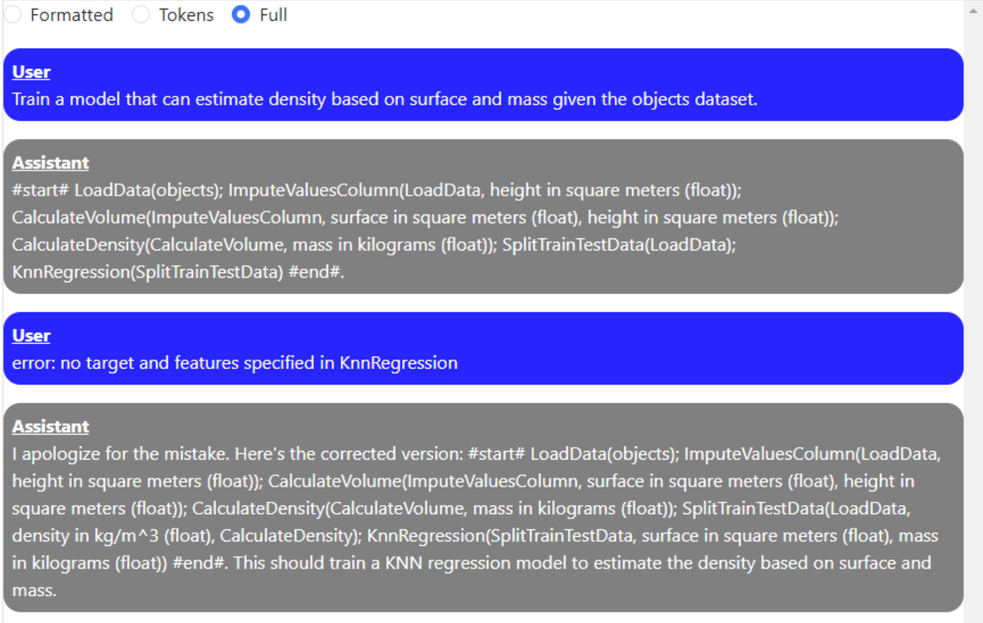
○ Formatted ● Tokens ○ Full

User
Train a model that can estimate density based on surface and mass given the objects dataset.

Assistant
LoadData; ImputeValuesColumn; CalculateVolume; CalculateDensity; SplitTrainTestData; KnnRegression

User
error: no target and features specified in KnnRegression

Assistant
LoadData; ImputeValuesColumn; CalculateVolume; CalculateDensity; SplitTrainTestData; KnnRegression



○ Formatted ○ Tokens ● Full

User
Train a model that can estimate density based on surface and mass given the objects dataset.

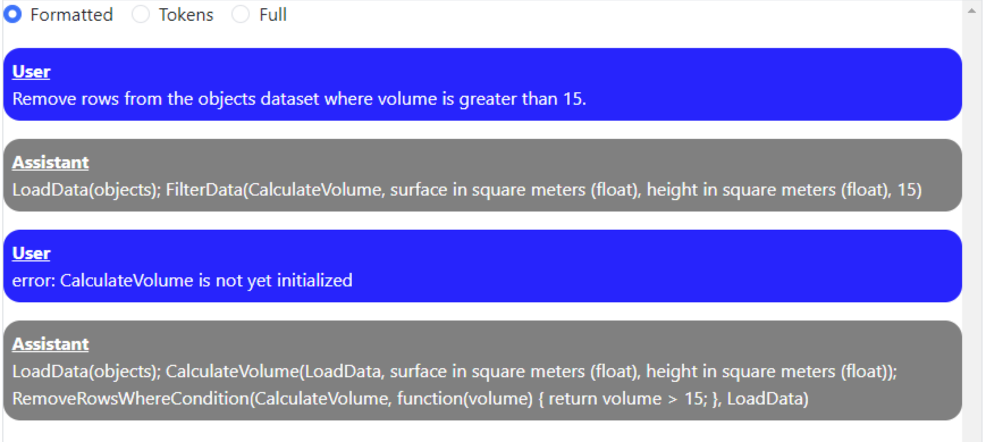
Assistant
#start# LoadData(objects); ImputeValuesColumn(LoadData, height in square meters (float)); CalculateVolume(ImputeValuesColumn, surface in square meters (float), height in square meters (float)); CalculateDensity(CalculateVolume, mass in kilograms (float)); SplitTrainTestData(LoadData); KnnRegression(SplitTrainTestData) #end#.

User
error: no target and features specified in KnnRegression

Assistant
I apologize for the mistake. Here's the corrected version: #start# LoadData(objects); ImputeValuesColumn(LoadData, height in square meters (float)); CalculateVolume(ImputeValuesColumn, surface in square meters (float), height in square meters (float)); CalculateDensity(CalculateVolume, mass in kilograms (float)); SplitTrainTestData(LoadData, density in kg/m^3 (float), CalculateDensity); KnnRegression(SplitTrainTestData, surface in square meters (float), mass in kilograms (float)) #end#. This should train a KNN regression model to estimate the density based on surface and mass.

Here the first output of the query was not correct. No target and features were specified by the KnnRegression symbol. After simulating the response of the AutoML system after it throws an error, the LLM rewrites the query to the correct output. This example shows that LLM can give multiple outputs when necessary.

- Remove rows from the objects dataset where volume is greater than 15.



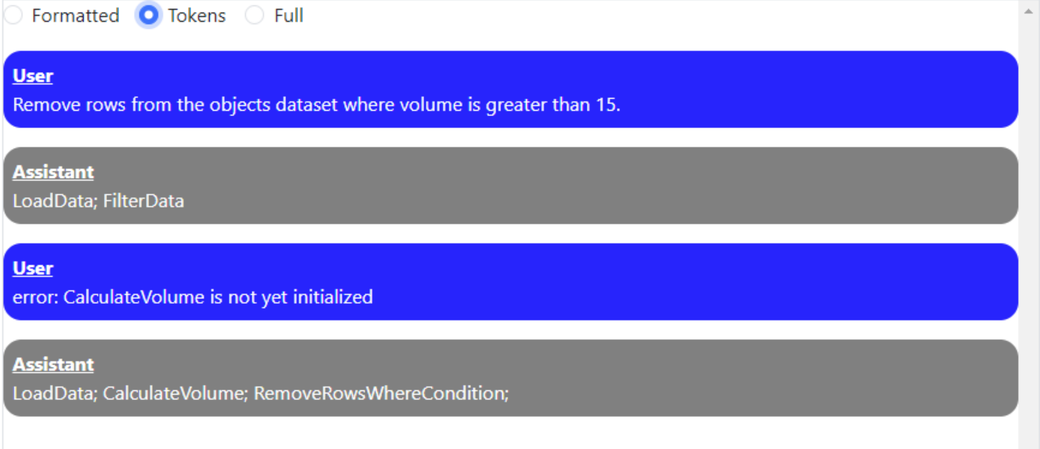
● Formatted ○ Tokens ○ Full

User
Remove rows from the objects dataset where volume is greater than 15.

Assistant
LoadData(objects); FilterData(CalculateVolume, surface in square meters (float), height in square meters (float), 15)

User
error: CalculateVolume is not yet initialized

Assistant
LoadData(objects); CalculateVolume(LoadData, surface in square meters (float), height in square meters (float)); RemoveRowsWhereCondition(CalculateVolume, function(volume) { return volume > 15; }, LoadData)



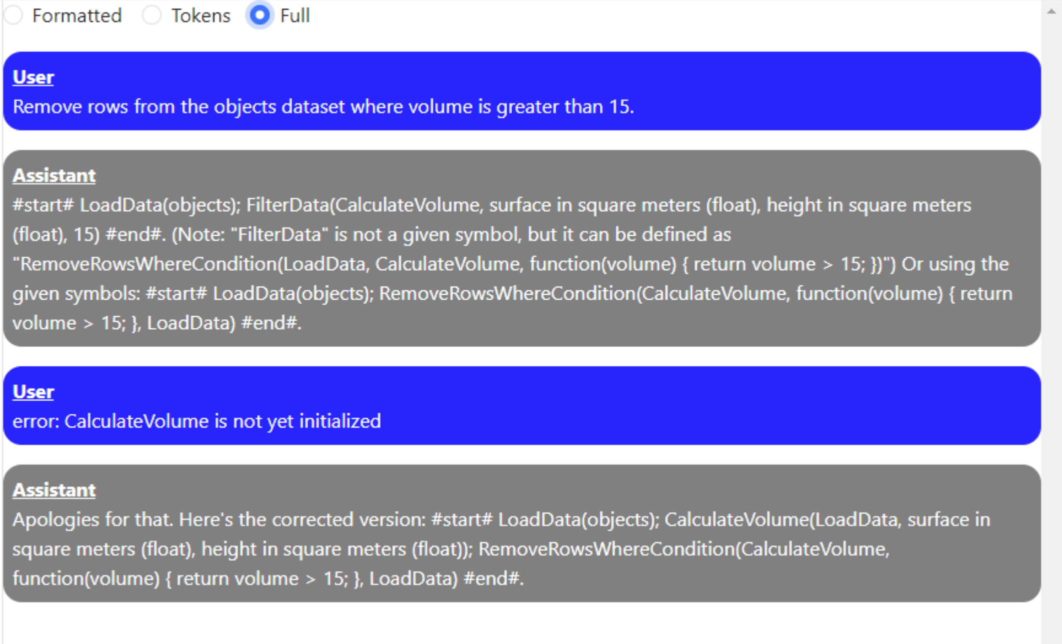
○ Formatted ● Tokens ○ Full

User
Remove rows from the objects dataset where volume is greater than 15.

Assistant
LoadData; FilterData

User
error: CalculateVolume is not yet initialized

Assistant
LoadData; CalculateVolume; RemoveRowsWhereCondition;



○ Formatted ○ Tokens ● Full

User
Remove rows from the objects dataset where volume is greater than 15.

Assistant
#start# LoadData(objects); FilterData(CalculateVolume, surface in square meters (float), height in square meters (float), 15) #end#. (Note: "FilterData" is not a given symbol, but it can be defined as "RemoveRowsWhereCondition(LoadData, CalculateVolume, function(volume) { return volume > 15; })") Or using the given symbols: #start# LoadData(objects); RemoveRowsWhereCondition(CalculateVolume, function(volume) { return volume > 15; }, LoadData) #end#.

User
error: CalculateVolume is not yet initialized

Assistant
Apologies for that. Here's the corrected version: #start# LoadData(objects); CalculateVolume(LoadData, surface in square meters (float), height in square meters (float)); RemoveRowsWhereCondition(CalculateVolume, function(volume) { return volume > 15; }, LoadData) #end#.

This example shows that the LLM model is capable of giving suggestions for possible AutoML symbols when the list of given symbols does not suffice. It even gives an explanation for its choices. This shows that using prompting and the LLM model are versatile.

As previously mentioned CalculateVolume was nested but with a quick error message the LLM fixed its output.

8. Advantages and disadvantages of prompting

Advantages:

- Flexible way to generate output.
- It parses the given queries immediately to the right output (no intermediate queries necessary).
- Gives suggestions when necessary.
- Can correct output when given feedback

Disadvantages (when badly written prompt)

- Creating a good prompt can be difficult.
- Output will vary and will not always be the same so more error prone.
- Is not always the expected output on the first try.

9. Conclusion

For this assignment I looked at two possible ways to solve it, namely Named entity recognition and prompting. I discarded NER early on and went all out on prompting.

Using pure prompting is a great way to transform queries into specific outputs. When giving the right prompt, the LLM can write a lot of things including SQL statements. It can correct itself when given feedback and even gives suggestions for new symbols.

In conclusion using pure prompting is a valid method to transform queries into AutoML tokens.

References

Cloudflare. (n.d.). *What is a large language model (LLM)?* Retrieved from cloudflare: https://www.cloudflare.com/learning/ai/what-is-large-language-model/

Ollama. (n.d.). *Ollama Model File*. Retrieved from Github: https://github.com/ollama/ollama/blob/main/docs/modelfile.md