Prompting Is Programming: A Query Language for Large Language Models

Critique by
Alex Mathai
Batool Taraif

Background

- Mask Decoding
  - A filtering approach, isolating specific tokens for decoding
  - Utilizes a binary mask – 1 to include, 0 to exclude, and applies it to softmax output

- Few-shot prompting
  - Uses broad text-sequence datasets to train language models, making them adaptable for various tasks without specific preparation.
  - Example: Translating 'cheese' to French by providing successful translation pairs, asking the model to complete the sequence, turning tasks into simple sequence completions.

- Multi-part prompting
  - Divides tasks into smaller, manageable steps
  - Enhances understanding and boosts confidence

Related Works

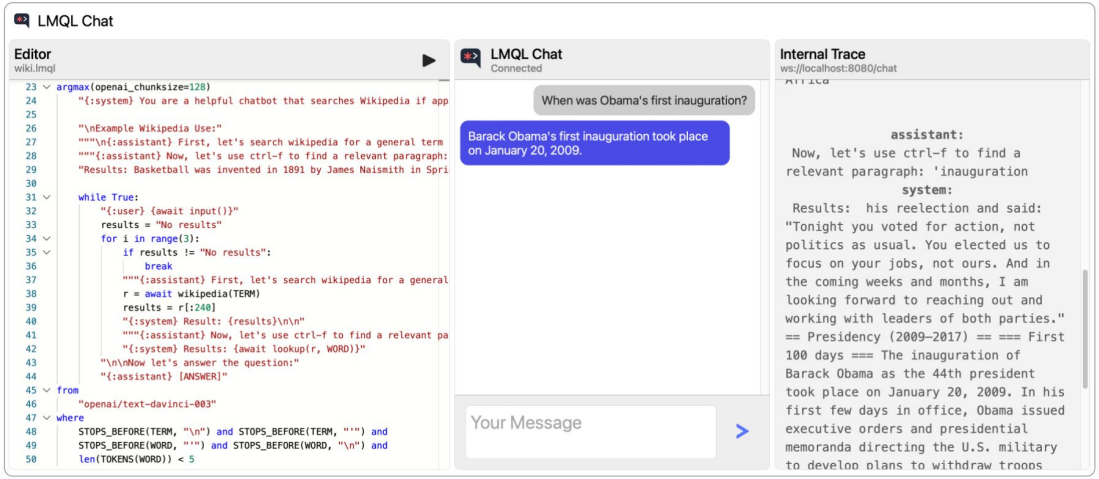
- Guidance Library
  - Another Domain Specific Language for constructing and prompting LMs
  - allows users to constrain generation (e.g. with regex and CFGs) as well as to interleave control (conditional, loops) and generation seamlessly
  - More efficient control over LMs in comparison to traditional prompting methods

```
import guidance

# Define a Guidance program for a chatbot
program = guidance('''
{{#system~}}
You are a helpful assistant.
{{~/system}}
{{#user~}}
I want a response to the following question:
{{query}}
{{~/user}}
{{#assistant~}}
{{gen 'response'}}
{{~/assistant}}
''')

# Execute the program
output = program(query='How can I be more productive?')
response = output['response']
```

```
Output: 'One way to be more
productive is to prioritize your
tasks and focus on one task at a
time. Also, taking regular breaks can
help maintain your focus and energy
levels.'
```

Impact

- [LMQL Chat]
  - Extends LMQL from static prompting to chat, enabling users to build interactive systems on top of LLMs
  - Makes it easy to create conversational agents with features like tool usage, internal reflection or safety constraints




```
argmax
   "Hey!"
   "2 + 2 = [ANSWER]"
from
   "jeffwan/vicuna-13b"
where
    INT(ANSWER)
```

Figure 1: A simple LMQL query.

```
argmax
   "{:system} You are a helpful chatbot."
   while True:
      "{:user} {await input()}"
      "{:assistant} Interal: [REFLECTION]"
      "{:assistant} [ANSWER]"
from
   "chatgpt"
where
   STOPS_AT(REFLECTION, ".")
```

Figure 2: A simple chatbot in LMQL Chat.

Strengths

- Allows for a very easy and intuitive framework that forces the LLM to adhere to constraints that are required for a downstream task
- Directly applicable to industry downstream tasks like Chatbot systems
- The framework is agnostic to the LLM being used and hence can be integrated with open source and closed source models
- Supports many kinds of constraints like length constraints and string matching. The framework also allows for loops !
- Allows for integration of tools into decoding - like a calculator !

Weaknesses

- The additional constraint checking on the output is compute intensive and hence introduces latency into the generation process
- It is important to consider whether the constraints significantly impact performance for your specific application
- It applies a constraint on string matching rather than the underlying logic

Questions

Thoughts on other kinds of constraints that are not as restrictive - like neural constraints ? Maybe something like the output should have positive/negative sentiment ?