Home test unleash.so

Strategy

We design a system with multiple levels that have overlap and redundancy. A main key of the system is to measure customer satisfaction. The goal is to optimize the configuration, similarly to searching hyper-parameters in classic data science projects, while balancing cost and satisfaction. The system's satisfaction percentage will be determined deterministically, and we will strive to keep it above this threshold. If we are above the satisfaction percentage, we will try to reduce steps or switch to open source tools to reduce costs. If we fall below the satisfaction percentage, we will optimize the steps themselves by engineering prompts or by adding steps from new information learned from the error analysis.

The absence of a ground truth

We don't have any ground truth to evaluate the model. In the future, we will be able to measure models using data that has been collected by using metrics such as BLEU or ROUGE score between prediction and old data that get positive rating from the users. So at the start, our only option is to ask users to provide feedback to the model. This forces us to do live experiments on our users, and therefore we need to be very careful. At this stage, we prefer multiple steps with high cost in order to ensure high quality.

The rating system

The chatbot will have a rating system with thumbs up or down for each response. When the user clicks on the thumbs up, it will be recorded. If the user clicks on the thumbs down, a checklist will pop up with options such as "the response is wrong," "I prefer that the chatbot not answer this type of question," "the system doesn't include important resources," and free text for input. The thumbs up will be used for A/B testing, while the thumbs down will be used for error analysis and secondary metrics for the A/B system. The rating system can be subject to A/B testing to determine the best system at a later time (stars against thumb).

There is two option to perform A\B testing:

1. To use different models to different companies.
2. Randomly generated answers from a different configuration across all the companies.

We think the second option is better because there may be a variance between companies.

The pipeline

1. A question is given to the system.
2. The question is passed to the decoder that inflates the question. Prompt: "Rewrite the question. Keep only relevant information in your output. Organize the text and break it into separate tasks: [*original question*]"

3. Append the original question to the output of the previous step. Text: “[*original question*]. Another way to ask this is: [inflate question]”.
4. Send to decoder prompt: “ At the JSON, I provide a description of the database [*JSON*]. My question is: [concatenate question]. On a scale of 1 to 5, with 5 being very sure and 1 being not sure at all, how confident are you that you can answer the question”
    - Our approach heavily relied on the ability of decoders as zero\few shot learning. In the future, this step could be replaced by fine-tuning an encoder for classification.
    - This step can be used to determine how to handle the task. If the model is more confident, we can send the task to a cheaper or open-source model. If the model is less confident, we can bring in ‘the heavy guns’ to handle the task.
5. Optional step: Send to decoder prompt: “In the JSON I describe the database [JSON that describes the data]. My question is: [concatenate question]. Which table and column should be kept to answer the question? Return the database and columns that are relevant for the data in JSON format”
6. Automatically filter the database according to the previous answer using code.
7. Send to decoder prompt: “ The JSON describes a database [*filter JSON*]. Write code for this question:[*concatenate question*] ”
8. Run the code from the previous section. If an error message is received during the run, the system will send the prompt from the previous section, along with the code and the error message, and ask the model to fix the code. The system will do this up to 5 times, or until the code runs successfully.
9. Optional step: Send to decoder prompt: “[*original question*]. I think the answer is [*answer from previous step*]. Does it look reasonable to you? If it doesn't return only the word no”. This step can be remove at later time

As I described in the beginning, this pipeline contains a lot of redundancy. This pipeline is an “opening proposal” and it prioritizes satisfaction over cost. The next step is to optimize the prompts to better result. After we have satisfaction from the result, we can try to replace\delete steps to make the process leaner and cheaper.
Few shots is a critical part of the process. I did not detail it because it makes the document less readable and in any case it is a process that needs to be optimized through A\B testing.

Notes:

1. The proposal includes a reflection of the model on itself. This is part of the emerging capabilities of models like ChatGPT and will not work with less deep models. The inspiration for the proposal was drawn from agent models
2. A hidden assumption is that the models themselves do not change. This assumption is not correct when using an external API. For example, we know that OpenAI plays with their models all the time, and the quality of the models has deteriorated over the last period, probably to improve run time.
3. On Tuesday, lamma2 was released by Meta. According to some metrics, it is better than GPT3.5. It is an interesting option to test, because besides the savings in costs, it allows to guarantee customers that no information of theirs is sent to a third party.