# Homework 8: Few-shot Learning and Prompt Engineering

Due April 15th at 10pm

This week, you will explore *few-shot learning* in large language models (LLMs). Instead of training the model to do a specific task, we represent it as a next-token prediction task. To do this, we will engage in *prompt engineering*: exploring different ways of presenting the task as a prompt. This helps LLMs solve tasks that at first glance, may not seem like text generation problems.

## 1 Our LLM

You will work with a popular LLM called LLaMA. I have given you a library that sends queries to a LLaMA model running on my research server. To use it, **you must be on the Wellesley Secure network or the Wellesley VPN.**

**Test out the LLaMA model**. Import the **query_llama.py** library into a Python shell or Python program, then call its **completion_query** function:

```
python
>>> import query_llama
>>> query_llama.completion_query("My favorite flavor of ice cream is")
" chocolate, I'm not sure why"
```

You should see a completion to the string inside of the query function. If you run into a connectivity issue, let me know right away! Note that the model output will be different each time you run it.

## 2 PuzzleQA

We will work with a dataset of puzzles from a game show that airs on National Public Radio. Wellesley alum Michelle Zhao built this dataset and used it to benchmark large language models for her thesis. Her work was published at the International Conference on Computational Creativity.

On the original quiz program, the questions are free-response. However, to make it easier to score, Michelle created a multiple-choice version. The **puzzleqa.csv** file contains a subset for you to use.

### 2.1 PuzzleQA items

Each puzzle in the dataset consists of several parts:

- Category: type of puzzle
- Explanation: description of the game

- Question: the question to answer
- Opt0-Opt3: multiple choice answer options
- Gold: the correct answer

The model's goal is to guess the correct answer (Gold) given the explanation of the game, the question, and the set of answer options.

## 3 Prompt Engineering

I have given you some starter code to show you how to evaluate LLaMA on a multiple-choice version of the PuzzleQA dataset. Your task is to improve on this basic prompt.

### 3.1 Starter code

Read the starter code to make sure you understand it. The prompt is formatted using the **basic_mc_prompt** function. This function name is passed to the **runMCtask** function as an argument (to make it easy to swap in different prompt formatting functions later in the assignment).

For instance, if you run the **basic_mc_prompt** function on the first item in the dataset, the prompt it produces is as shown below:

```
Explanation: Every answer is a two-word phrase in which the first word
starts with O. Drop the O, and you'll get a new word that ends the phrase.
Question: area high in the atmosphere where the air is thin
Choice 0: Oman man
Choice 1: omission mission
Choice 2: ozone zone
Choice 3: orange range
Answer: Choice
```

The function will also return the correct answer choice for this item (2 in this case).

If you query the model with this prompt, you might get a response like the one below:

```
 2. If you drop the O, you
```

This is a correct response, because the first token produced by the model is the correct answer number, 2.

**Run the basic multiple choice task on the dataset and record how well LLaMA scores.**

### 3.2 Prompt Formatting

Make a copy of the **basic_mc_prompt** function called **custon_mc_prompt**. Play around with how the prompt is formatted and make some observations about how this affects the model performance.

**Try out some changes to the following components:**

- Formatting of the question component

- Formatting of the answer options
- Formatting of the correct answer

**Describe in text what you tried and whether it seemed to improve performance or not.**

## 3.3 Few-shot Prompting

In *few-shot prompting*, we give the model some examples of the task we would like it to solve as part of the prompt.

**Implement the fewshot_mc_prompt function**. Your function should create a prompt that has 3 example problems before the problem that the model is supposed to solve.

You may take problems from the **more_puzzleQA.tsv** file to use as examples. You are also free to construct your own examples if you would like.

**Hint:** You may use **basic_mc_prompt** to help format the example questions.

**Run the multiple choice task using your few-short prompting function and record how well LLaMA scores.**

## 3.4 Chain-of-Thought Reasoning

In *chain-of-thought prompting* (Wei et al. 2022), we prompt the model to explain its reasoning before generating an answer. This is supposed to help the model produce logically consistent answers.

### 3.4.1 Reasoning examples

To implement chain-of-thought (CoT) prompting, you will need to construct some examples of step-by-step reasoning for PuzzleQA problems to show the model. Tables 9 and 10 in the appendix of this paper have example chain-of-thought prompts.

### 3.4.2 Prompting function

**Implement the cot_mc_prompt function.** Your function should create a prompt that has 3 examples of reasoning through a PuzzleQA problem. It should also prompt the model to produce step-by-step reasoning before the answer to the question.

### 3.4.3 Evaluation

Because the model is now generating reasoning as well as an answer, we will need to increase the maximum number of tokens that we let the model generate. This is controlled by an optional "max_tokens" argument in the **completion_query** function.

Try printing out the model answers and see if they are getting cutting off. **Raise the token limit until you see that the model is able to complete its response.** What token limit did you use?

**Run the multiple choice task using your chain-of-thought prompting function and record how well LLaMA scores.**

## 4 Game Generation

In her thesis, Michelle also experimented with having models invent new puzzle games.

### 4.1 Generation prompting

**Complete the generation_prompt function.** You should show the model some example puzzles and have it generate a new puzzle.

### 4.2 Generating games

Next, **complete the runGenerationTask function.** It should take a number of generation attempts to make. For each attempt, it should call your **generation_prompt** function to construct a prompt and then pass it to **completion_query** to prompt the model. You will likely need to increase the token limit even higher.

To make it easier for you to read the puzzles, I have given you an **export** function that takes a list of generated games and writes them out to a text file. Your **runGenerationTask** function should return a list of games, which you can pass to this function in **main**.

### 4.3 Evaluation

How well does the model do at generating puzzles? **Write a paragraph about the generated games, and give some examples of games that you generated.** Do they make sense? Would they be fun to play? If not, what issues do you see?

## 5 Reflection Questions

### Question 1

Game generation is more difficult to evaluate than question-answering, since there is no one right answer. Can you think of some way to automatically evaluate the generated games?

### Question 2

Do you think the ability of AI to play games affects how much fun they are? Why or why not?

### Question3

In 2011, IBM's Watson AI system competed against two human opponents on the popular trivia show Jeopardy! Read this article about the subsequent failure of the Watson project. What do you think this story tells us about the using puzzles and games to measure progress in AI?