Keynote

LLM types

- based LLM
- instruction tuned LLM

LLM types ≠ model types

- model: can be large, small
- LLM is part of generic model

Based LLM

- predict things, i.e. AI writing

Instruction tuned LLM

- respond according to instruction, i.e. Q&A

Prompting benefit

- no need to label data, build model, train model, test model
- can ask LLM to output based on prompt
- time efficient

Prompting by API

- similar to input in ai application with ui
- prompt input and print output by ourselves without ui
- API call > access model > call LLM to output/ respond/ execute based on prompt and given content

Prompting categories

- summarizing
- extracting
- inferring
- transforming
- expanding

Single prompt methodology

Opt 1: request to respond a task based on one content, i.e. extract username from a review
Opt 2: request to respond multiple tasks based on one content, i.e. extract username, product name from a review
Opt 3: request to respond a task based on multiple contents, i.e. extract username from multiple reviews
Opt 4: request to respond multiple tasks based on multiple contents, i.e. extract username, product name from multiple reviews

Prompting two principles

- principle 1: write clear and specific instructions
- principle 2: give the model time to 'think'

Install the OpenAI python library in vscode

- pip install openai
- import openai
- openai.api_key = "sk-..."

*API key = API token, need to be generated in openai website
*HK is not allowed to access openai

Prompt template in OpenAPI

Define prompt
- prompt = f"""Generate an answer based on my question"""

Ask LLM to respond
- response = get_completion (prompt)
- response = get_completion(prompt, temperature=0.7)

Print response
- print(response) OR
- print('Answer in this way: ' + response)

*in prompt, add \: make contents readable and manageable
*in prompt, add ```{text}```: ask LLM to focus on responding the original text as a whole
*in prompt, add <required output format>: ask LLM to output in required format
*only prompt use f-string, other contents do not
*temperature: degree of output randomness, change depend on prompt requirement

Principle 1 - Write clear and specific instructions

Clear: clarity with enough contexts for prompt execution

1. use delimiters to quote the task and avoid prompt injection, ensure execute content as a whole, i.e. ", "", ``, <>
2. ask for structured output formats to respond, i.e. JSON, HTML
3. check if the conditions are satisfied, i.e. check if required assumptions are enough for response/
ask to output in two different opts based on different conditions
4. few-shot prompting, i.e. show human demonstrations in texts before LLM respond

*make sure prompt also instruct how you want LLM output the answer, i.e. add comma to seperate answer

Principle 2 - Give the model time to 'think'

Think: spend enough time to do computation before responding

1. specify steps to complete a task, i.e. list out steps in required output format, format is 1st principle tactic
2. instruct model to think before conclude and complete a task, i.e. list out human thinking approach as a baseline

Prompt injection

- a sub-task that is included inside a task
- models do not know which task should be performed > respond randomly
- define the prompt with delimiters to identify the required task clearly

Few-shot prompting = human demonstrations

- generative AI/ robotics: both need human feedback (labeled), human demos for response reference

Prompt development lifecycle (iterative)

- idea/ problem
- implementation, i.e. refine prompt with specific requirements, e.g. word limit, focus output areas
- experimental result
- error analysis

Summarizing

- summarize the contents based on prompt
- can be single/ multiple tasks
- if multiple, define a List of all required contents > define for-loop of List > print(i, response, "\n")

*"\n": enter a newline to print the response of each content one by one, optional

If task is summarization/ generation > every run > every output is different

*except the summarized content is a fact that cannot be changed

Extracting

- extracting is part of natural language processing (NLP)
- use case: provide a content > ask to summarize > then ask to extract certain parts
- can be mixed with use of summarizing and inferring

Inferring

- serve high level purposes, i.e. ask to provide basic understanding of a content
- can be single/ multiple tasks
- classify sentiment of a content, e.g. customer reviews on a product, classify happy or not, yes/ no
- identify specific sentiments, i.e. happy, angry, etc
- answers can be >= 1, depend on content and prompt requirement
- extract keywords as an understanding of a content

If output has too many items, the most effective way is to output by JSON format

Transforming

- translate from language A to language B
- identify language
- translate to formal and informal tone
- tranform from format A to format B, i.e. HTML, JSON format
- transform from text A to text B, i.e. re-writing, shortening
- check gramma and spelling
- can be single/ multiple tasks, i.e. translate to multiple languages and show in formal tone

*transform format prompt needs to indicate old and new format
*if content is an article > respond as an article
*if content is a List of sentences > respond each sentence one by one
*content presentation can affect the output way

Universal translation

- provide required contents in different languages > define for-loop to output translation of different languages into certain one
- single prompt: request to respond a task based on multiple contents

Expanding

- create a response including all the other functions, i.e. extracting, based on given content
- e.g. write a customer service automated email that may use extracting, summarizing, transforming all at the same time

Temperature

- degree of randomness of model's output
- temperature = 0 > no randomness > absolute correct outputs only > increase reliability > for prediction model
- temperature > 0 > have randomness > creative and various outputs > increase variety and creativity > for conversational model

*temperature: range from 0.1 - 0.9
*define it in response = get_completion (prompt, temperature=0.7), depend on the nature of prompting you want

Chatbot

- user request as a prompt
- bot respond to prompt
- system define behavior of assistant, i.e. assistant role, assistant task/ duty, etc
- user > system > assistant
- user = end user
- bot = assistant
- system = manager of assistant
- system messages > prompt > define with clear and enough infomation/ all the info that users might ask about