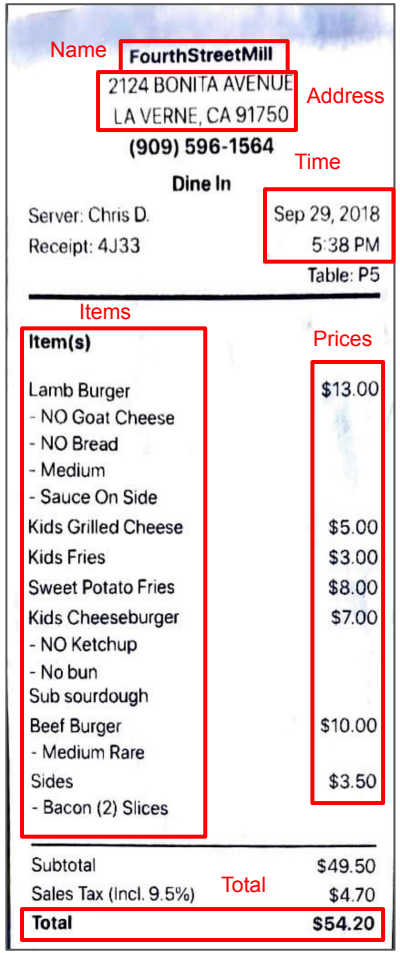
Badal:
Prompt Engineering for Generative models

Created and presented by: Ilya Ovsyannikov

Agenda

- Extracting data from Receipts, querying against LLMs
- Defining problem types
- Roles
- Queries
- LangChain Solutions
- Links



- Name
- Address
- Time
- Items
- Prices
- Total

str

BigQuery Table

Running queries against LLMs with the extracted information

LLM tasks:

- Time Parsing (+ Breakfast, Lunch or Dinner)
- Alcohol Checker
- Calories Estimates
- Image Generation
- Budget Policy Compliance
- Restaurants rating (similar restaurants nearby)

Defining the problem types:

Task	Problem type
Time Parsing (+ Breakfast, Lunch or Dinner)	Extraction, Classification
Alcohol Checker	Classification
Calories Estimates	New Content Generation,Lookup
Image Generation	Image Generation
Budget Policy Compliance	Extraction
Restaurants rating (similar restaurants nearby)	New Content Generation, Lookup

Roles: System, User, Assistant

SYSTEM

- Helps set the behaviour of the assistant:
- provide specific instructions about how it should behave throughout the conversation.
- Is optional and the model's behaviour without a system message is likely to be similar to using a generic message such as "You are a helpful assistant."

USER Provide requests or comments for the assistant to respond to

ASSISTANT Store previous assistant responses, but can also be written by you to give examples of desired behavior.

⊕ **Add message**

	Query --> Time parsing + classification	GPT 4	PaLM
Unstructured	From the text **date_time** extract: date, time average, day of the week, meal time. For meal time, classify the time into one of the 3 groups: breakfast, lunch or dinner.	Date: September 29, 2018 Time: 5:38 PM Day of the week: Saturday Meal time: Dinner	The date is Sep 29, 2018. The time is 5:38 PM. The day of the week is Sunday. The meal time is dinner.

Structured	**System:**
	- You are a helpful assistant.
	- Return JSON only, no descriptions.
	User:
	- **Extract** from the text in """ the following values:
	- time as '%H:%M:%S',
	- date as '%Y-%m-%d',
	- day of the week,
	- **Classify** day period: Breakfast, Lunch, Dinner.
	Text:
	'''date_time'''
	Assistant:
	- Timeframe for Breakfast: from earliest possible until 11:30 am
	- Timeframe for Lunch: from 11:30am to 5:00pm
	- Timeframe for Dinner: from 5:00pm till the latest possible time
	- if values not detected/empty/corrupted: use 'none' value.
	- if multiple times detected: calculate average time and use as result time
	- Return as json file with **mandatory keys:** time, date,
		{
		"time": "17:38:00",
		"date": "2018-09-29",
		"day_of_week": "Saturday",
		"day_period": "Dinner"
		}
			``json
			{
			"time": "17:38:00",
			"date": "2018-09-29",
			"day_of_week": "Friday",
			"day_period": "Dinner"
			}

	Query --> Alcohol checker	GPT 4	PaLM
Unstructured	I ordered these items: **items**. Does it include alcohol beverages?	Based on the list of items you provided, it does not include any alcoholic beverages. You have ordered various food items such as burgers, grilled cheese, fries, and sides, but no specific drinks were mentioned in your list.	No, the order does not include alcohol beverages.
Structured	**System:** - You are a helpful assistant. - Return a class only, nothing else **User:** - **Classify** the text in """ whether it includes alcohol drink(s) or not. Text: '''**items**''' **Assistant:** - There are only three classes: yes, no, unknown. - If no text is provided, the class is "unknown". - Return a class only, nothing else	no	. no

	Query --> Calories estimates	GPT 4	PaLM
Unstructured	"I ordered these items: **items** Give me estimates of calories of the entire meal.	Here is an estimated calorie count for each item in your meal: 1. Lamb Burger (No Goat Cheese, No Bread, Medium, Sauce on Side) - 390 calories 2. Kids Grilled Cheese - 300 calories 3. Kids Fries - 200 calories 4. Sweet Potato Fries - 350 calories 5. Kids Cheeseburger (No Ketchup, No Bun, Sub Sourdough) - 350 calories 6. Beef Burger (Medium Rare) - 550 calories 7. Bacon (2 Slices) - 80 calories These estimates are approximate and may vary based on portion sizes and specific ingredients used by the restaurant. Adding up the above calorie counts, the entire meal is estimated to be around 2,220 calories.	The total number of calories in your meal is approximately 1,600. This is based on the following estimates: * Lamb Burger: 300 calories * Kids Grilled Cheese: 200 calories * Kids Fries: 150 calories * Sweet Potato Fries: 200 calories * Kids Cheeseburger: 250 calories * Sub sourdough: 300 calories * Beef Burger: 400 calories * Bacon (2 slices): 100 calories Please note that these are just estimates, and the actual number of calories in your meal may vary depending on the specific ingredients used.

Structured

System:

- You are a helpful assistant.
- Return JSON only, no descriptions.

User:

- The text it """ represents the text from a restaurant receipt.
- **Return** JSON file:
- key=item in the receipt
- value=estimated calories in the item
- Add the last key "total" and its value = sum of all item's calories

Text:

'''items'''

Assistant:

- Each key represent a single item from the menu.
- Include customization into the item. Ex. Stake, (medium rare, no ketchup, no bun).
- Add the last key "total" and its value = sum of item's calories.
- If no/empty/corrupted list items provided, return empty JSON.
- If ambiguous items, assign value = "unknown"
- Return JSON only, no descriptions.

{
"Lamb Burger - NO Goat Cheese - NO Bread - Medium - Sauce On Side": 600,
"Kids Grilled Cheese": 300,
"Kids Fries": 250,
"Sweet Potato Fries": 350,
"Kids Cheeseburger - NO Ketchup - No bun - Sub sourdough": 400,
"Beef Burger - Medium Rare": 500,
"Bacon (2) Slices": 100,
"total": 2500
}

{
"Lamb Burger": 420,
"Kids Grilled Cheese": 240,
"Kids Fries": 150,
"Sweet Potato Fries": 200,
"Kids Cheeseburger": 300,
"Sub sourdough": 300,
"Beef Burger": 400,
"Bacon (2) Slices": 100,
"total": 2210
}

	Query --> YELP Rating	GPT 4	PaLM
Unstructured	Give me a list of 5 restaurants similar to name near address. List items based on YELP ratings. Include the YELP rating scores.	1. Cafe X2O La Verne - 4.5 stars (https://www.yelp.com/biz/cafe-x2o-la-verne-la-verne) 2. La Verne Bistro - 4 stars (https://www.yelp.com/biz/la-verne-bistro-la-verne) ETC.	1. The Counter (4.5 stars) 2. The Habit Burger Grill (4.5 stars) 3. Rubio's Coastal Grill (4.5 stars) 4. The Cheesecake Factory (4.5 stars) 5. BJ's Restaurant & Brewhouse (4.5 stars)

Structured	**System:**
	- You are a helpful assistant.
	- return only JSON, no extra descriptions
	User:
	Generate me a list of top 5 restaurants similar to the restaurant provided in the "text" below:
	Text:
	"name:
	address:"
	Assistant:
	Conditions:
	- rating is based on YELP
	- restaurants are within 3km from the one mentioned in the "user" message
	Output format:
	- JSON format
	- **Mandatory Keys:** name of the place, address, rating, cuisine type, distance from the restaurant mentioned in the prompt
		{
		"1": {
		"name": "Café Wang",
		"address": "3648, E Foothill Blvd, Pasadena, CA 91107",
		"rating": 4.5,
		"cuisine_type": "Taiwanese, Chinese",
		"distance": 2.3
		},
		"2": {
		"name": "Brick Market & Deli",
		"address": "1055 W Foothill Blvd, Claremont, CA 91711",
		"rating": 4.0,
		"cuisine_type": "Sandwiches, Delis",
		"distance": 1.8
		},...
		}
			``
			[
			{
			"name": "The Counter",
			"address": "2030 Bonita Avenue, La Verne, CA 91750",
			"rating": 4.5,
			"cuisine": "Fast food",
			"distance": 0.12
			},
			{
			"name": "The Habit Burger Grill",
			"address": "2110 Bonita Avenue, La Verne, CA 91750",
			"rating": 4.5,
			"cuisine": "Fast food",
			"distance": 0.15
			},...
			]

	Query --> Budget Policy	GPT 4	PaLM

Structured	**System:**
	- You are a helpful assistant.
	- Return JSON only, no descriptions.
	
	User:
	- **Extract** budget values from it based on seniority:
	- **"'txt_content'"**
	
	Assistant:
	- The text in """ is a budget policy document.
	- **Extract values** and return them in JSON format with the keys: breakfast_budget, lunch_budget, dinner_budget.
	- **Mandatory keys** for seniority level: junior, senior, executive
	- if no/irrelevant/corrupted values are detected, fill the key values with 'no_values_detected' phrase.
	- do not include dollar sign or any other currency symbols.
		{
		"junior": {
		"breakfast_budget": 30,
		"lunch_budget": 35,
		"dinner_budget": 40
		},
		"senior": {
		"breakfast_budget": 40,
		"lunch_budget": 45,
		"dinner_budget": 50
		},
		"executive": {
		"breakfast_budget": 50,
		"lunch_budget": 55,
		"dinner_budget": 60
		}
		}
			``
			{
			"junior": {
			"breakfast_budget": "30",
			"lunch_budget": "35",
			"dinner_budget": "40"
			},
			"senior": {
			"breakfast_budget": "40",
			"lunch_budget": "45",
			"dinner_budget": "50"
			},
			"executive": {
			"breakfast_budget": "50",
			"lunch_budget": "55",
			"dinner_budget": "60"
			}
			}
			``

Solution by LangChain

Prompt Templates = reproducible way to generate a prompt.

A prompt template can contain:

- instructions to the language model,
- a set of few shot examples to help the language model generate a better response,
- a question to the language model.

```python
from langchain import PromptTemplate

# An example prompt with no input variables
no_input_prompt = PromptTemplate(input_variables=[],
                                 template="Tell me a joke.")
no_input_prompt.format()
# -> "Tell me a joke."

# An example prompt with one input variable
one_input_prompt = PromptTemplate(input_variables=["adjective"],
                                  template="Tell me a {adjective} joke.")
one_input_prompt.format(adjective="funny")
# -> "Tell me a funny joke."
```

Solution by LangChain

Output Parsers = help structure language model responses.

2 main methods:

- "Get format instructions" – returns a string containing instructions for how the output should be formatted.
- "Parse" – parses a response string into some structure.

Other parsers:

- Json
- DateTime
- List
- Enum
- Structured output
- etc.

```python
from langchain.prompts import PromptTemplate
from langchain.llms import OpenAI
from langchain.output_parsers import PydanticOutputParser
from pydantic import BaseModel, Field

model_name = 'text-davinci-003'
temperature = 0.0
model = OpenAI(model_name=model_name, temperature=temperature)

# Define your desired data structure.
class Joke(BaseModel):
    setup: str = Field(description="question to set up a joke")
    punchline: str = Field(description="answer to resolve the joke")

# Set up a parser + inject instructions into the prompt template.
parser = PydanticOutputParser(pydantic_object=Joke)

prompt = PromptTemplate(
    template="Answer the user query.\n{format_instructions}\n{query}\n",
    input_variables=["query"],
    partial_variables={"format_instructions": parser.get_format_instructions()}
)

# And a query intented to prompt a language model to populate the data structure.
joke_query = "Tell me a joke."
_input = prompt.format_prompt(query=joke_query)
output = model(_input.to_string())
parser.parse(output)
```

Links:

- https://www.youtube.com/watch?v=zNACfPuaqaI
- https://docs.google.com/spreadsheets/d/1gn1YWVqSidbT5S723LJdF8q3oDeexFum8UuRu0bhJyw/edit#gid=575350139
- https://platform.openai.com/docs/guides/gpt-best-practices
- https://cloud.google.com/vertex-ai/docs/generative-ai/text/text-overview
- https://www.linkedin.com/pulse/how-prompt-chat-gpt-get-better-results-matthew-k-roberts/
- https://docs.langchain.com/docs/components/prompts/output-parser
- https://docs.langchain.com/docs/components/prompts/
-