RAG for LLM

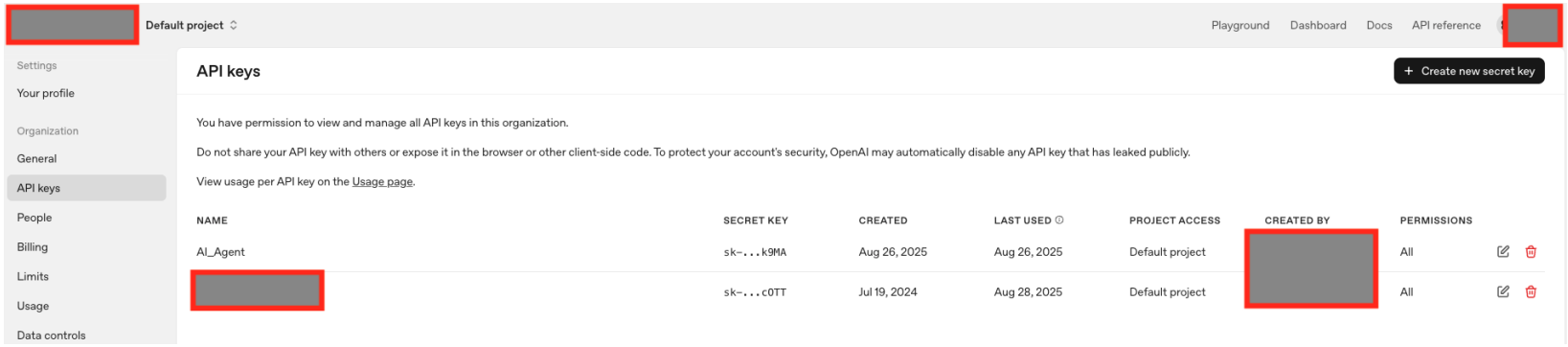
北岛老师

Outline

- How to use LLM through API?
- RAG Modes
  - Query
  - TF-IDF or BM2.5
  - Embedding
- Project
  - Data
  - Task

How to use LLM through API?

- Go to https://platform.openai.com/settings/organization/api-keys



How to use LLM through API?

- Three ways to store it
  - In notebook

```
import os
os.environ["OPENAI_API_KEY"] = "sk-..."
```

  - In .env file (safer)
    - Create a .env file containing

```
OPENAI_API_KEY=sk-xxxxxxx
OPENAI_MODEL=gpt-4o-mini
```

    - In a notebook

```
from dotenv import load_dotenv
import os

# load from current folder
load_dotenv(dotenv_path=".env")

# or if your .env is in parent directory:
# load_dotenv(dotenv_path="../.env")

# sanity check
print("OPENAI_API_KEY loaded?", bool(os.getenv("OPENAI_API_KEY")))
```

How to use LLM through API?

- Three ways to store it
  - In a corporate setting, store it in secret

```
def get_openai_secret_keys():

    """
    Get the secret open ai api keys for the current user.
    """

    try:
        current_user = dbutils.notebook.entry_point.getDbutils().notebook().getContext().userName().get().split("@")[0]
        scope_name = f'openai-{current_user}'
        api_user_key = dbutils.secrets.get(scope = scope_name, key = "token")
    except Exception as e:
        raise Exception(str(e))

    try:
        dbutils.secrets.list(scope = "openai")
        org_key = dbutils.secrets.get(scope = "openai", key = "token_org")
        return (current_user, api_user_key, org_key)
    except Exception as e:
        raise Exception(str(e))


CURRENT_USER, PROJECT_API_KEY, ORGANIZATION_API_KEY = get_openai_secret_keys()

openai_client = OpenAI(
    api_key=PROJECT_API_KEY,
    organization=ORGANIZATION_API_KEY
)
```

How to use LLM through API?


```python
from openai import OpenAI
client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))
system_prompt = "Write clear travel recommendations."
user_prompt = "I need a 5 day travel plan from San Francisco to Los Angeles."
messages = [
    {"role": "system", "content": system_prompt},
    {"role": "user", "content": user_prompt}
]
response = client.chat.completions.create(
    model="gpt-4o",
    messages=messages,
    temperature=0.5
)
plan = response.choices[0].message.content.strip()
print(plan)
```

How to use LLM through API?

- Parameters of client.chat.completions.create
  - model: name of the model, e.g. "gpt-4o-mini", "gpt-4o", "gpt-3.5-turbo".
  - temperature (default 1.0): Controls randomness.
    - Lower → more deterministic (e.g. 0.2)
    - Higher → more creative (e.g. 0.8–1.2)
  - top_p (default 1.0): Controls diversity via nucleus sampling.
    - top_p=1.0 → no restriction
    - top_p=0.9 → only sample from the top 90% probability mass
  - n (default 1): How many completions to generate for each prompt.
  - max_tokens: Maximum number of tokens to generate in the reply.

How to use LLM through API?

- Parameters of client.chat.completions.create
  - messages: a list of dictionary. Each dict is like {"role": "system"|"user"|"assistant"|"tool", "content": "text"}
    - system: instructions for the model

      {"role": "system", "content": "You are a travel planner who speaks concisely."}

    - user: human input (your prompt)

      {"role": "user", "content": "Plan me a 3-day trip to New Orleans focused on music and food."}

    - assistant (optional): model responses in earlier runs

      {"role": "assistant", "content": "Sure, here's a draft itinerary..."}

    - tool: used when model calls a tool/function

RAG Modes

- RAG = Retrieval-Augmented Generation.

- It’s an architecture where a language model (LLM) is combined with an external knowledge source (e.g., a database, vector store, or documents) to improve its answers.

Without RAG

User: *“What’s the revenue of ACME Corp in 2023?”*

LLM: *“I don’t know, but I’ll guess...”* 🤷

With RAG

1. Query vector → retrieve passage: *“ACME Corp 2023 annual report: revenue = $4.2B”*
2. Prompt → *“Using the provided data, answer: What’s ACME’s revenue in 2023?”*
   LLM: *“ACME Corp reported $4.2B revenue in 2023.”* ✅

RAG Modes

- Two steps of RAG systems
  - Step 1. Retrieval
    - Given a user question, query data to provide relevant information.
    - The information can be passed in the format of text, rows of a table or structured data.
  - Step 2. Generation
    - The LLM gets the original user question + the retrieved context in its prompt.
    - It then generates an answer grounded in that context.
  - The whole process is hard-coded. The engineers need to know where to pull data, organize the query and embed the data in user messages.

RAG Modes

- There are 3 modes of RAG
  - Query: exact keyword match only
  - TF-IDF or BM25: smart ranking based on keyword match, consider document length
  - Embedding: catch semantic similarity
    - cheap food v.s. affordable street eats
    - models:
      - all-MiniLM-L6-v2: lightweight, English only
      - all-distilroberta-v1: medium
      - E5: state of art by Microsoft

RAG Modes


```python
docs = [
    "San Francisco Chinatown cheap food tour",
    "Golden Gate Bridge scenic photography spot",
    "SF food trucks: affordable street eats near SOMA",
    "Jazz club in New Orleans on Frenchmen Street",
    "Seattle Pike Place Market seafood tasting",
    "Miami South Beach nightlife and bars",
    "Budget-friendly museums in Chicago",
    "San Francisco farmers market local produce",
    "Austin BBQ and live music on Rainey Street",
    "Orlando theme parks for families"
]
query = "cheap food in San Francisco"
```

```
Query: 'cheap food in San Francisco'

Exact keyword match (top):
  score= 4.0 | San Francisco Chinatown cheap food tour
  score= 2.0 | San Francisco farmers market local produce
  score= 1.0 | SF food trucks: affordable street eats near SOMA
  score= 1.0 | Jazz club in New Orleans on Frenchmen Street
  score= 1.0 | Budget-friendly museums in Chicago

BM25 (top):
  score=  5.6768 | San Francisco Chinatown cheap food tour
  score=  2.5184 | San Francisco farmers market local produce
  score=  1.3574 | Budget-friendly museums in Chicago
  score=  1.1000 | Jazz club in New Orleans on Frenchmen Street
  score=  1.1000 | SF food trucks: affordable street eats near SOMA

Embeddings (top):
/Users/bd11/opt/anaconda3/envs/ai_project/lib/python3.12/site-packages/
  from .autonotebook import tqdm as notebook_tqdm
  score=  0.7879 | San Francisco Chinatown cheap food tour
  score=  0.6659 | San Francisco farmers market local produce
  score=  0.6300 | SF food trucks: affordable street eats near SOMA
  score=  0.4301 | Seattle Pike Place Market seafood tasting
  score=  0.3305 | Miami South Beach nightlife and bars
```

Project

- Data: 3 tables
  - activities (city, name, theme, duration_hours, cost_usd, opening_hours, notes)
  - hotels (city, name, neighborhood, nightly_price_usd, review_score, walk_score, notes)
  - flights (origin, destination, airline, price_usd, depart_time, arrive_time, on_time_rate)

Project

- Data: 3 tables

activities (225, 8)

	id	city	name	theme	duration_hours	cost_usd	opening_hours	notes
0	A0001	New York City	Central Park	sports	1.0	40	08:00-20:00	Accessible by transit
1	A0002	New York City	The Metropolitan Museum of Art	family	2.6	30	10:00-17:00	Great photo spots
2	A0003	New York City	Times Square	beach	3.0	40	08:00-19:00	Book ahead
3	A0004	New York City	Statue of Liberty	music	3.1	20	08:00-17:00	Local favorite
4	A0005	New York City	Brooklyn Bridge	sports	1.2	40	08:00-19:00	Local favorite

hotels (150, 8)

	id	city	name	neighborhood	nightly_price_usd	review_score	walk_score	notes
0	H0001	New York City	The Plaza	Historic District	263	4.0	86	Close to transit
1	H0002	New York City	The Standard, High Line	Waterfront	146	3.8	86	Great nightlife
2	H0003	New York City	The Langham, New York	Arts District	123	3.8	77	Good breakfast
3	H0004	New York City	The Pierre	Waterfront	285	4.1	69	Near museums
4	H0005	New York City	The Bowery Hotel	Near Transit	117	4.0	80	Good breakfast

flights (210, 8)

	id	origin	destination	airline	price_usd	depart_time	arrive_time	on_time_rate
0	F0001	San Francisco	Philadelphia	Alaska	223	10:45	10:43	0.84
1	F0002	Portland	San Francisco	Spirit	601	16:10	12:45	0.78
2	F0003	Austin	Denver	American	398	12:30	14:10	0.93
3	F0004	Boston	San Francisco	Alaska	162	12:30	12:45	0.85
4	F0005	Denver	Portland	Delta	397	18:25	08:55	0.80

Project

- Trip Planner with RAG
  - Input:
    - origin: str.
    - destination: str
    - start_dt: str. Travel start date
    - end_dt: str Travel end date
    - budget: int
    - themes: str | List[str]. The theme of the travel, e.g., nature, or [beach, nightlife]
  - Output:
    - The travel plan including the flight, lodging and activities that is within budget.
    - Optional but preferred: daily plan
  - Approach: use all three RAG modes