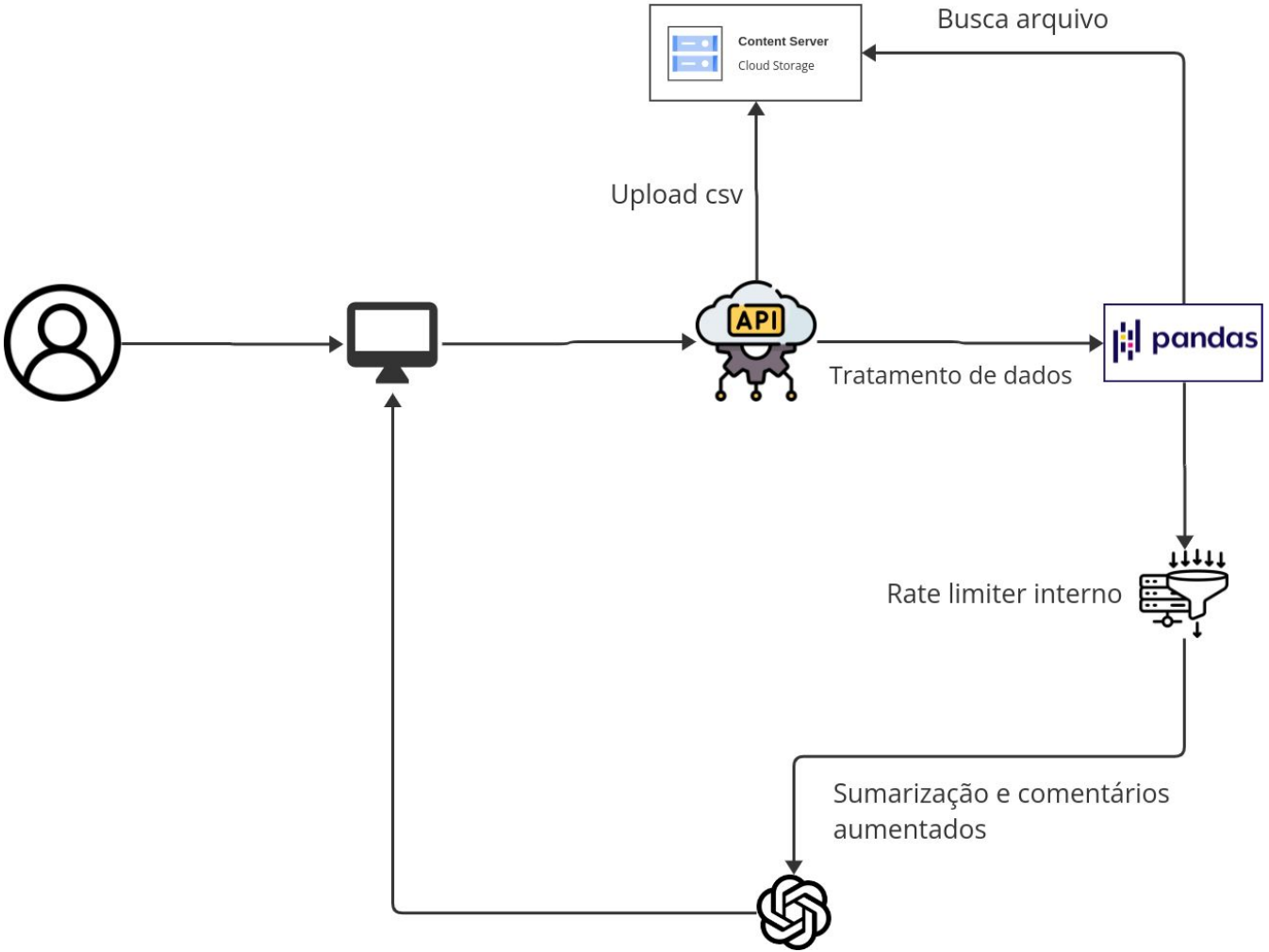
Análise de textos com Langchain e Gemini

Gemini

1011 011 01 1011001 10 11011 011 01 110110 110111 1101

Lidando com LLMs no mundo real

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1


Content Server
Cloud Storage
Busca arquivo
Upload csv
API
pandas
Tratamento de dados
Rate limiter interno
Sumarização e comentários
aumentados

> Problemas enfrentados

- Tempo gasto no tratamento, extração dos dados, aplicação das regras e criação dos prompts;
- Grande números de requests;
- Limite de requests para a LLM;
- Alucinação;
- User Experience.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

```python
# ======================== routes.py ========================
@router.get("/{id}/analyze")
async def get_analysis(
    id: str,
    df: Annotated[pd.DataFrame, Depends(get_df_from_stored_file)],
    storage: GCPStorage = Depends(get_file_storage),
) -> DataResponse:
    data = await process_file(df)
    return DataResponse(**data)

# ======================== processing.py ========================
async def process_file(df: pd.DataFrame) -> dict:
    ...
    return await mount_data(df)


async def mount_data(df: pd.DataFrame) -> dict:
    filtered_data = filter_data(df)
    return await group_data_by_feature(filtered_data)


async def group_data_by_feature(items: list) -> dict:
    ...
    response = await asyncio.gather(*[mount_item(item) for item in items])
    ...
    return grouped_items


async def mount_item(item: pd.Series) -> Item:
    ...
    new_item = Item(...)
    prompt = build_prompt(new_item)
    response = await get_answer_from_llm(prompt)
    ...
    return new_item

```

```python
# ======================== llm.py ========================
_SEMAPHORE = asyncio.Semaphore(10)

async def get_answer_from_llm(prompt: list) -> str:
    chat = OpenAIChatRequest(model='gpt-4o', temperature=0.0, messages=prompt)
    async with _SEMAPHORE:
        async with httpx.AsyncClient(timeout=600) as client:
              response = await client.post(
                  urljoin(self.url, '/openai/v1/chat/completions'),
                  headers=self.headers,
                  json=chat.model_dump(),
              )
    ...
    return response_data['choices'][0]['message']['content']
```

E as requisições em batch?

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

```python
from openai import OpenAI

client = OpenAI()

batch_input_file = client.files.create(
  file=open(FILE_NAME, "rb"),
  purpose="batch"
)

batch_job = client.batches.create(
    input_file_id=batch_input_file.id,
    endpoint="/v1/chat/completions",
    completion_window="24h"
)
```

> Vantagens e desvantagens

- Diminui o número de requests para a LLM;
- Diminui os custos;
- Entretanto, na OpenAI, o processamento pode levar até 24h;
- Langchain ainda não suporta.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

> Quem sou eu?

YURI SALES

Desenvolvedor Back-End, pós-graduado em Engenharia de Software

linkedin: /yuriscosta
github: @yurisalesc

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1