Summary of Recommended Steps for Advanced AI-Driven Parsing and Analysis Pipeline

February 2, 2025

Contents

1 Overview

This document outlines the recommended steps for creating a robust pipeline for advanced AI-driven parsing and analysis. The main areas addressed include:

- Organizing code for each week in separate directories.
- Integrating caching to address cost issues and avoid redundant LLM calls.
- Adding rate-limiting to handle API usage constraints.

- Using configuration files (`.env`, JSON, YAML) to define model keys, usage parameters, and an analysis depth variable.
- Implementing a basic validation layer that cross-checks or enforces expected schema/format.
- Wrapping everything in an orchestration script that sequentially calls each step and logs any errors.

Recommended next steps include:

- Updating environment configuration.
- Installing additional dependencies.
- Integrating with existing parsing scripts.
- Setting up proper logging.
- Creating configuration files for different analysis scenarios.

2 Update Environment Configuration

2.1 Goals

Centralize all API keys, rate-limit settings, and usage parameters so they are easy to manage and adjust.

2.2 Action Items

1. Create or update your `.env` file with the relevant keys and settings:

```
OPENAI_API_KEY=your_openai_key_here
ANTHROPIC_API_KEY=your_anthropic_key_here
GOOGLE_API_KEY=your_google_key_here
RATE_LIMIT_DELAY=1
ANALYSIS_DEPTH=shallow  % or moderate, deep
```

2. Use a library such as `python-dotenv` to load these variables:

```
from dotenv import load_dotenv
load_dotenv()
import os

OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
ANALYSIS_DEPTH = os.getenv("ANALYSIS_DEPTH", "shallow")
```

3. Keep `.env` out of version control by providing a `.env.example` file.

3 Install Additional Dependencies

3.1 Goals

Ensure your environment supports advanced AI parsing, rate-limiting, caching, and logging.

3.2 Action Items

- Install AI provider libraries:

```
pip install openai
pip install anthropic
```

- Set up caching (using a lightweight approach with DiskCache or similar):

```
pip install diskcache
```

- Install a robust Markdown parser:

```
pip install mistune
```

- Choose a logging library (using the built-in `logging` module or a package like `loguru`):

```
pip install loguru
```

Update your `requirements.txt` or `pyproject.toml` accordingly.

4 Integrate with Existing Parsing Scripts

4.1 Goals

Enhance your current parsing scripts to include caching, configuration-based settings, rate-limiting, and AI calls.

4.2 Action Items

1. Read environment variables in each script to set parameters (e.g., $\texttt{ANALYSIS}_DEPTH, \texttt{RATE}_LIMIT_DELAY$).$Addacachin$
2. cache = Cache('./cache$_dir'$)

   def cached$_a i_c all(prompt, model) : key = hashlib.sha256((prompt+model).encode()).hexdigest() if key in cache :$
   $return cache[key]$

Rate limiting time.sleep(float(os.getenv("RATE$_LIMIT_DELAY$","1")))

Make the API call here (pseudo-code) response = call$_a i_a pi(prompt, model) cache[key] = response return response$

Modify your existing parse functions to call `cached_ai_call` instead of making direct API calls.

5 Set Up Proper Logging

5.1 Goals

Collect logs from each step of the pipeline for easier debugging and error tracking.

5.2 Action Items

1. Initialize a logger at the entry point of your application:

```
import logging
from logging import handlers

logger = logging.getLogger(__name__)
logger.setLevel(logging.INFO)
```

```
handler = handlers.RotatingFileHandler('pipeline.log', maxBytes=5000000,
    backupCount=3)
formatter = logging.Formatter('%(asctime)s [%(levelname)s] %(message)s')
handler.setFormatter(formatter)
logger.addHandler(handler)

logger.info("Logging initialized for pipeline.")
```

2. Log key steps in your scripts:

```
logger.info("Starting markdown parse for file: %s", filepath)
logger.error("Failed to parse table in file: %s", filepath)
```

6 Create Configuration Files for Different Analysis Scenarios

6.1 Goals

Enable quick switching between different analysis modes (e.g., shallow vs. deep analysis).

6.2 Action Items

1. Create a `config/` directory containing configuration files:
   - `config/shallow.env`
   - `config/deep.env`
   - `config/medium.env`
2. Update your pipeline scripts to accept a config file path. For example, in a shell script:

```
# usage: ./build_pipeline.sh shallow
CONFIG_FILE=config/$1.env
if [ -f "$CONFIG_FILE" ]; then
    export $(cat "$CONFIG_FILE" | xargs)
else
    echo "Config file $CONFIG_FILE not found."
    exit 1
fi
```

3. Include additional keys for advanced scenarios:

```
PROVIDER_PRIORITY=OpenAI,Anthropic,Google
MAX_TOKENS=3000
TEMPERATURE=0.7
```

7 Orchestrate the Pipeline

7.1 Goals

Wrap all steps in an orchestration script that calls each process sequentially and logs any errors.

7.2 Example: build_pipeline.sh

```
#!/usr/bin/env bash

# Step 1: AI-enhanced markdown parsing
echo "Parsing homework markdown with AI..."
python ai_enhanced_parse_markdown.py ../resources/week2/
    homework_week_two_assignment_one.md ../output/week2/spreadsheets/
    homework_parsed.json
python ai_enhanced_parse_markdown.py ../resources/week2/week_two_overview.md ../
    output/week2/spreadsheets/overview_parsed.json

# Step 2: Merge & discover relationships
echo "Merging & discovering relationships..."
python merge_and_discover.py ../output/week2/spreadsheets/homework_parsed.json ../
    output/week2/spreadsheets/overview_parsed.json ../output/week2/spreadsheets/
    week2_concepts.json

# Step 3: Generate enhanced concept map (Graphviz)
echo "Generating concept map..."
python ai_create_concept_map.py ../output/week2/spreadsheets/week2_concepts.json
    ../output/week2/diagrams/week2_map.dot
dot -Tpdf ../output/week2/diagrams/week2_map.dot -o ../output/week2/diagrams/
    week2_map.pdf

# Step 4: Copy references
echo "Copying references..."
mkdir -p ../output/week2/references
cp ../resources/week2/assets/* ../output/week2/references/

echo "Pipeline complete!"
```

8 Final Directory Layout

Below is an example structure of your project:

```
project/
       build_pipeline.sh
       requirements.txt
       .env.example
       cache_dir/          # DiskCache or similar caching store
       config/
             shallow.env
             deep.env
             medium.env
       scripts/
             ai_enhanced_parse_markdown.py
             merge_and_discover.py
             ... (other scripts)
       resources/
             week2/
                   homework_week_two_assignment_one.md
                   ... (other files)
             week3/
       output/
             week2/
                   spreadsheets/
```

```
                    diagrams/
                    references/
            week3/
        pipeline.log
```

9 Summary

By updating your environment configuration, installing the necessary dependencies, integrating with your existing parsing scripts, setting up proper logging, and creating configuration files for different analysis scenarios, your pipeline will:

- Remain organized (with weekly directories and a consistent structure).
- Control costs (via caching and rate-limiting).
- Operate reliably (with logging and validation).
- Adapt easily to new analysis scenarios (configuration-driven).