USDA LLM Model Pipeline

1. What the pipeline does (high level)

Stage	Script	What it Produces
Harvest & clean	API_Document_CommentsDownloader.py	One or more JSON files, each containing 'comment_id', full text, and a list of local PDF/Image attachments. All attachments are saved to *./Downloads/USDA_JSON/attachments* by default.
Summarize with GPT4o	LLM_Model_FINAL.py	Processed_comments.csv with columns: who_type, who_name, what, why, issues, scientific_legal_support, *etc*
Bucket issues	Comment_clustering.py	processed_with_categories.csv (each comment now has **high-level issue categories**) sorted_by_issue.csv (one row per comment × category pair, ready for pivoting) Two log files capturing every GPT response (for audit)

NOTE: The output of each stage is the required input of the next, so you **run the scripts in the above order**.

2. Environment & One-Time Setup

All three scripts are pure Python 3.10+. You can keep them in the same project folder.

Install system tools

- Windows:
  - Tesseract-OCR → default path C:\Program Files\Tesseract-OCR (used for OCR).
  - Poppler – place poppler/bin on your PATH (used by pdf2image).

Install below packages using -

```
pip install requests pandas pdfplumber pdf2image pytesseract pillow \
        python-dotenv openai
```

Create a .env in the project root –

```
OPENAI_API_KEY=sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

(The loader is called in the LLM and clustering scripts)

3. Stage-by-stage walkthrough

3.1 API_Document_CommentsDownloader.py

1. **User supplies a Regulations.gov URL** (document or docket).
   The helper extract_docket_id_or_document_id() parses the URL and classifies it .
2. **Find the internal objectId** for that document via the Regulations.gov REST API (/v4/documents/{id}) .
3. **Iterate through comments (250 per page)**
   - For each comment the script hits the comments endpoint, then follows the comment's **self link** to get the full body text .
   - Attachments are listed via /comments/{comment_id}/attachments.
     Every file is streamed to disk in ./Downloads/USDA_JSON/attachments, preserving its original MIME type.
4. **Light cleaning & incremental saving**
   Only three keys are stored: comment_id, text, attachments.
   Each page is written to its own file (e.g. comments_FSIS-2023-0001_page_3.json) so partial runs can be resumed easily.
   ➝ *Output ready for Stage 2.*

Change API key or rate: edit the API_KEY constant and/or sleep(0.1) delay between comment downloads.

3.2 LLM_Model_FINAL.py – comment summarization

1. **Configuration block (top of the file)**
   - JSON_FOLDER, PDF_FOLDER, OUTPUT_FILE – point to Stage 1 outputs.
   - USE_API=False lets you dry-run without burning tokens .
   - MAX_TOKENS is a hard truncate safeguard before sending text to GPT-4-turbo.
2. **Reading comments & attachments**

- For every JSON page file, each comment’s body text is taken.
- PDFs are opened locally; text is extracted by **pdfplumber** first, and (if empty) by **Tesseract OCR** on page images .

3. **Prompt-driven extraction**
   The classify_comment_by_issue() function contains the full system & user prompt (≈ 80 lines). It asks GPT-4-turbo to return valid JSON with keys:
   who_type, who_name, what, why, issues[], scientific_legal_support .
4. **Error handling**
   • Regex extract_json_block() strips Markdown fences and pulls out the first JSON object .
   • If parsing fails, a safe fallback row is inserted and the comment is flagged for manual review.
5. **Export**
   A single processed_comments.csv is written with one row per comment.
   ➜ *Output ready for Stage 3.*

How to edit the prompt
*Modify the big triple-quoted prompt variable inside classify_comment_by_issue().
*Change the model (model="gpt-4-turbo") or temperature (defaults to 0).

3.3 comment_clustering.py – thematic bucketing

1. **Load the CSV from Stage 2** (INPUT_FILE constant).
   The issues column is split into Python lists.
2. **Batch set of unique issues** (Counter > all_issues).
   Large dockets may have >1 000 unique phrases, so issues are chunked into **≤ 500** per GPT call .
3. **First GPT task – “Group issues into 8–15 categories”**
   - Prompt builder build_prompt() shows *every* issue as a bullet list; GPT must return an array of {category, related_issues[]} objects.
   - Each raw response is saved to gpt_issue_grouping_raw.txt for transparency.
4. **Second GPT task – “Merge near-duplicate category names”**
   This guard-rail collapses e.g. “Worker Safety and Health” & “Worker Conditions” into a single canonical label using another JSON-returning prompt .
5. **Map every comment to one or more high-level categories** (high_level_issues list).

- processed_with_categories.csv keeps the original wide format.
- sorted_by_issue.csv explodes the list so each comment-category pair is a row (handy for Power BI filters or pivot tables).

To change how clustering works
Tweak build_prompt() (lines near the top of the file) or the consolidation prompt in consolidate_categories().
Swap the model to gpt-4o, gpt-4o-mini, etc.
Adjust BATCH_SIZE to trade off speed vs. token load.

CUSTOMIZATION CHEAT SHEET:

Change you want	Where to edit
Use a different Regulations.gov API key	Top of API_Document_CommentsDownloader.py (API_KEY)
Switch from per-document to per-docket harvesting	Accept docket IDs in extract_docket_id_or_document_id() and pass them to get_comments_by_object_id() (API supports both)
Store downloads somewhere else	Paths under download_file() and JSON_FOLDER/PDF_FOLDER constants
Try GPT-4o or reduce temperature	The model and temperature parameters in each client.chat.completions.create() call
Add / remove output columns	In Stage 2, edit the data dict before it is appended to results
Change the number of high-level categories	Edit the line “Return only 8–15 categories TOTAL” in build_prompt()