YouTube Movie Clip Evidence Collector

What the tool does end-to-end

Given a list of YouTube channels:

Pull all videos (or newest N / since last run)

1. For each video:
    - Collected metadata (title, description, publish time, stats, duration, tags, etc.)
    - Computed a **risk score** (heuristics or ML)
    - Preserved evidence artifacts into a folder:
        i. video.json (API response)
        ii. watch.html (page snapshot)
        iii. thumbnail.jpg
        iv. captions.vtt (if present)
        v. manifest.json (hashes + timestamps)
2. Write a formatted **Excel report** summarizing results + file paths
3. Keep state in SQLite so subsequent runs are fully automated for minimal human interaction

Pipeline stages

- **Ingest**: channel discovery + video listing (YouTube Data API v3)
- **Enrich**: fetch details + stats + contentDetails
- **Detect**: score likely movie clip infringement
- **Preserve**: snapshot + download allowed artifacts + hash them
- **Report**: Excel output
- **State**: SQLite to dedupe + incremental runs

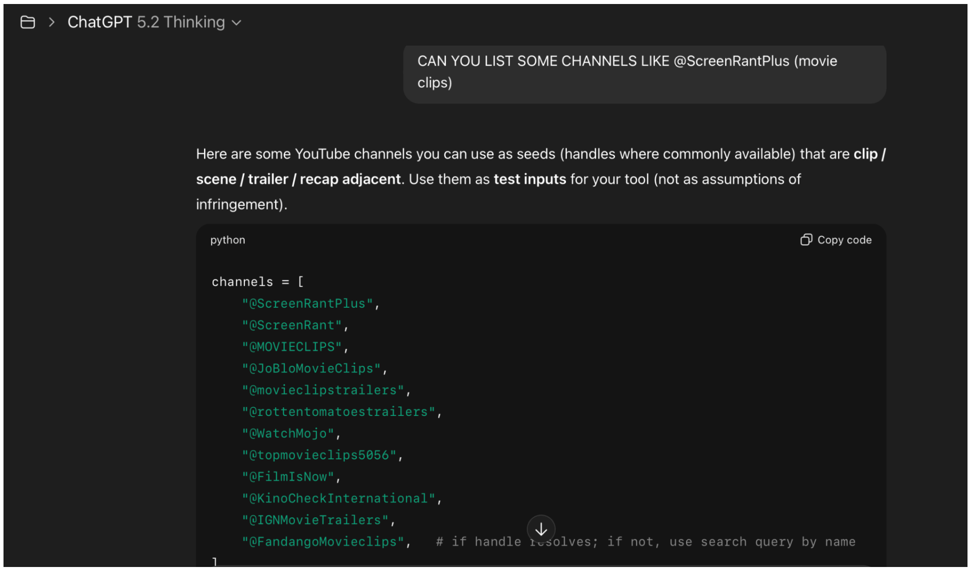
Note: I stored evidence of the video (metadata + snapshots + thumbnail + hashes), not the video media file itself.

Reason: Downloading the video stream can conflict with platform Terms unless you have explicit permission or a platform-approved approach.

Free components used:

- YouTube Data API v3: Free quota (daily units).
- Python open-source libraries: free.
- Local file storage: free.

LLM Prompts used throughout:

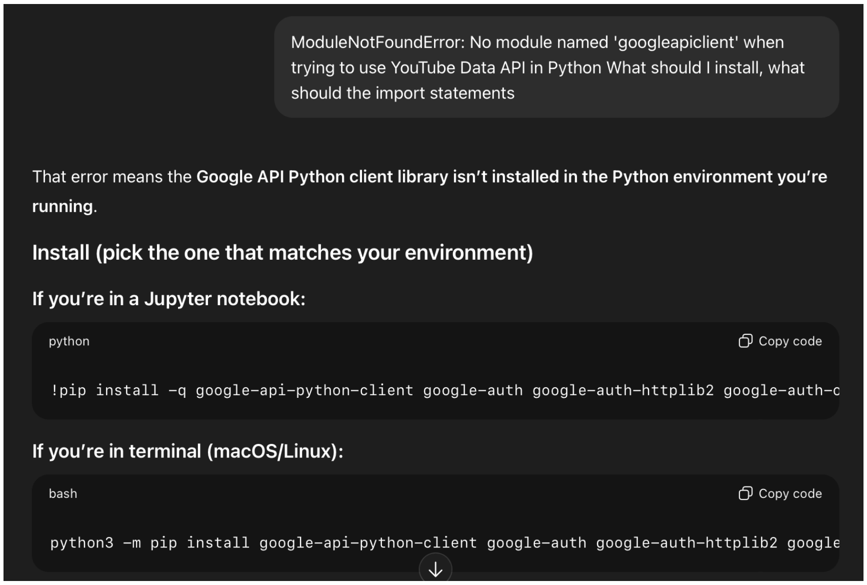


CAN YOU LIST SOME CHANNELS LIKE @ScreenRantPlus (movie clips)

Here are some YouTube channels you can use as seeds (handles where commonly available) that are **clip / scene / trailer / recap adjacent**. Use them as **test inputs** for your tool (not as assumptions of infringement).

```python
channels = [
    "@ScreenRantPlus",
    "@ScreenRant",
    "@MOVIECLIPS",
    "@JoBloMovieClips",
    "@movieclipstrailers",
    "@rottentomatoestrailers",
    "@WatchMojo",
    "@topmovieclips5056",
    "@FilmIsNow",
    "@KinoCheckInternational",
    "@IGNMovieTrailers",
    "@FandangoMovieclips",   # if handle resolves; if not, use search query by name
]
```

I needed a list of realistic YouTube channels that frequently post movie clips or clip-like content so I could test my end-to-end pipeline across multiple accounts, validate that channel resolution works, and generate a richer Excel report.



ModuleNotFoundError: No module named 'googleapiclient' when trying to use YouTube Data API in Python What should I install, what should the import statements

That error means the **Google API Python client library isn't installed in the Python environment you're running**.

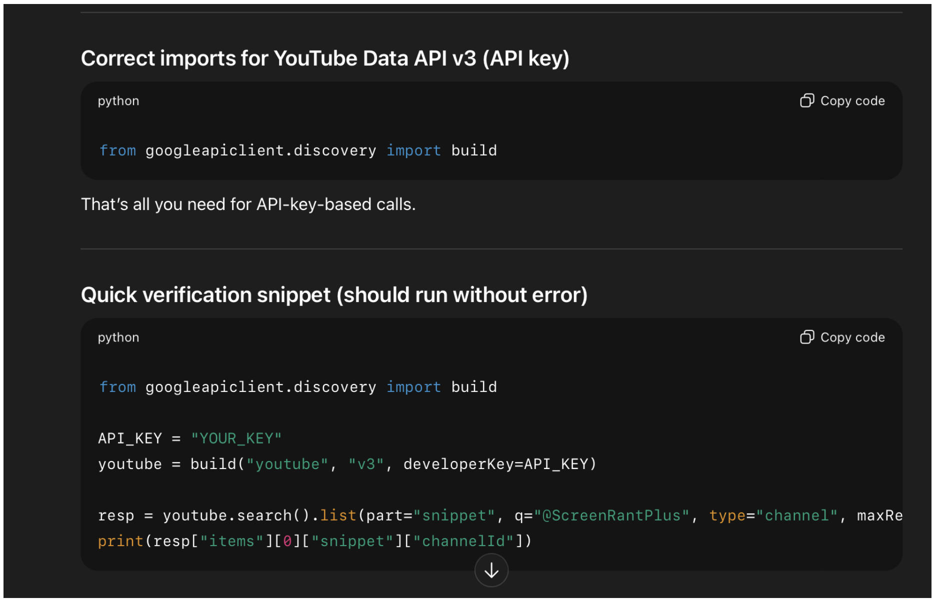
Install (pick the one that matches your environment)

If you're in a Jupyter notebook:

```python
!pip install -q google-api-python-client google-auth google-auth-httplib2 google-auth-c
```

If you're in terminal (macOS/Linux):

```bash
python3 -m pip install google-api-python-client google-auth google-auth-httplib2 google
```



Correct imports for YouTube Data API v3 (API key)

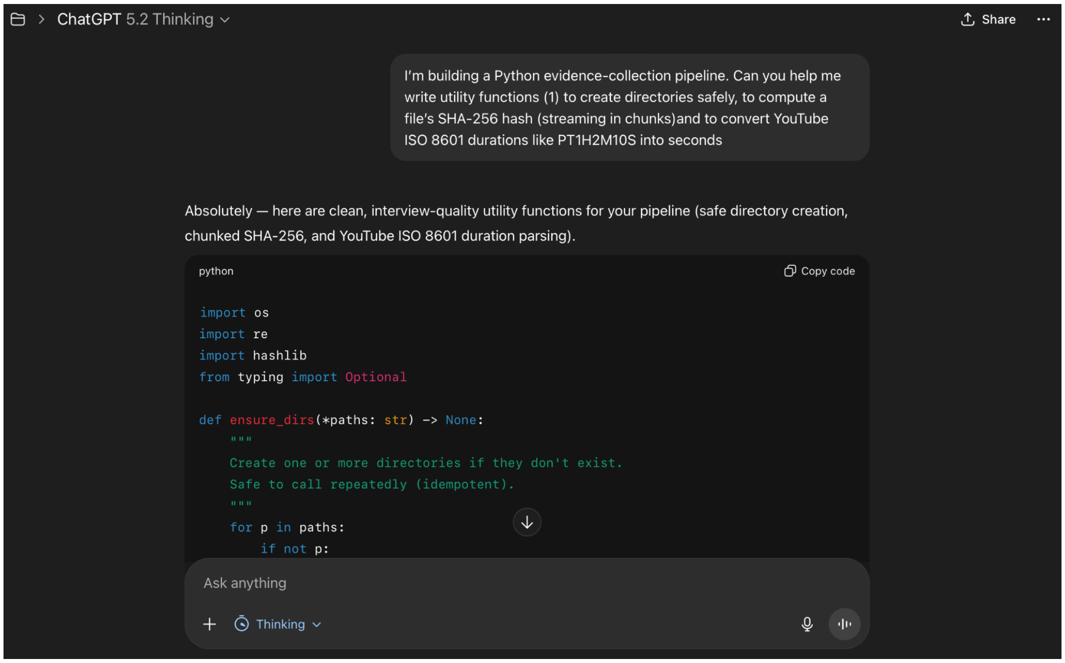
```python
from googleapiclient.discovery import build
```

That's all you need for API-key-based calls.

Quick verification snippet (should run without error)

```python
from googleapiclient.discovery import build

API_KEY = "YOUR_KEY"
youtube = build("youtube", "v3", developerKey=API_KEY)

resp = youtube.search().list(part="snippet", q="@ScreenRantPlus", type="channel", maxRe
print(resp["items"][0]["snippet"]["channelId"])
```

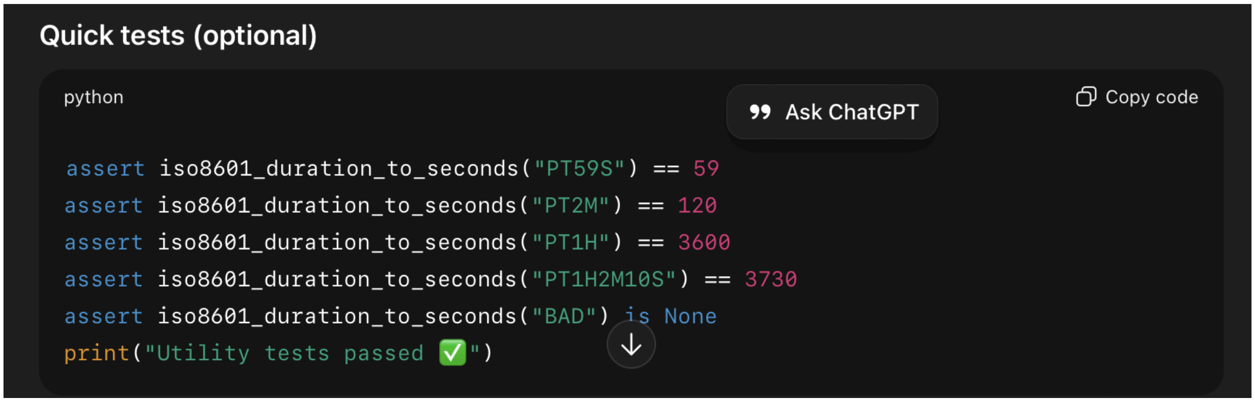
I hit an import error while integrating the YouTube Data API into my Python notebook. I needed to know the correct packages to install, the correct import statement for build(), and a minimal verification call to confirm the API client works before continuing development.



I'm building a Python evidence-collection pipeline. Can you help me write utility functions (1) to create directories safely, to compute a file's SHA-256 hash (streaming in chunks)and to convert YouTube ISO 8601 durations like PT1H2M10S into seconds

Absolutely — here are clean, interview-quality utility functions for your pipeline (safe directory creation, chunked SHA-256, and YouTube ISO 8601 duration parsing).

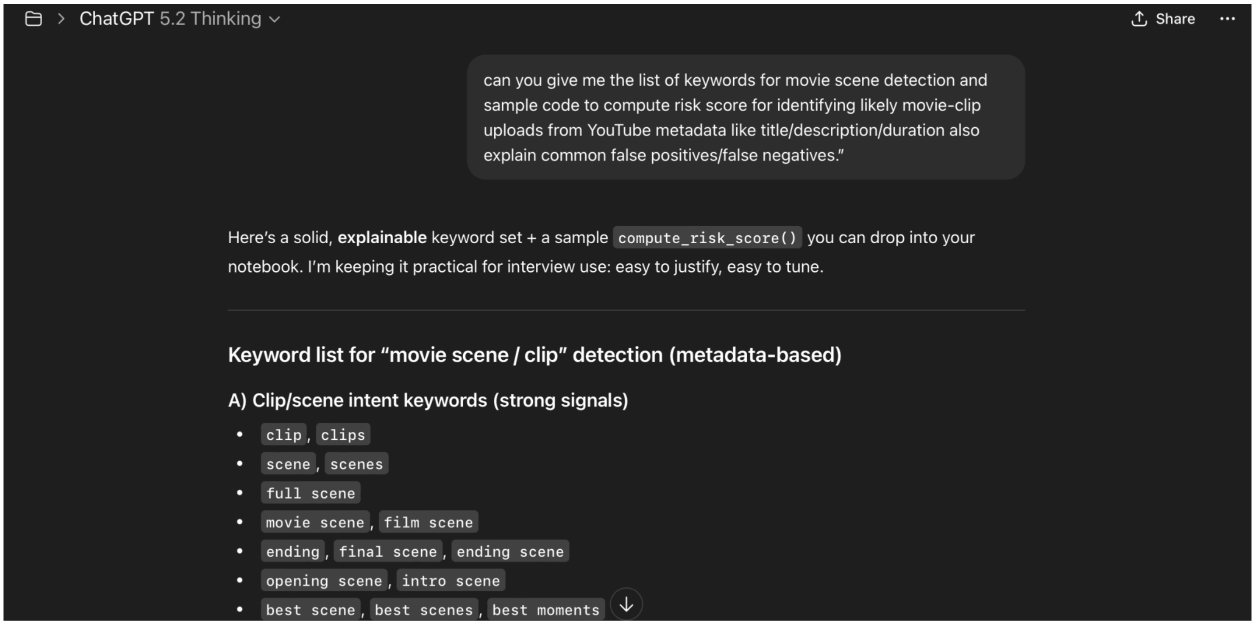
```python
import os
import re
import hashlib
from typing import Optional

def ensure_dirs(*paths: str) -> None:
    """
    Create one or more directories if they don't exist.
    Safe to call repeatedly (idempotent).
    """
    for p in paths:
        if not p:
```



Quick tests (optional)

```python
assert iso8601_duration_to_seconds("PT59S") == 59
assert iso8601_duration_to_seconds("PT2M") == 120
assert iso8601_duration_to_seconds("PT1H") == 3600
assert iso8601_duration_to_seconds("PT1H2M10S") == 3730
assert iso8601_duration_to_seconds("BAD") is None
print("Utility tests passed ✅")
```

I needed core helper functions to support evidence preservation and reporting reliably creating output folders, generating cryptographic hashes to verify integrity/chain-of-custody without loading large files into memory, and converting YouTube's ISO 8601 duration strings into seconds for scoring and Excel output. I asked to ensure best practices (chunked hashing, edge-case handling) and to reduce implementation errors, then verified correctness with small assertions.



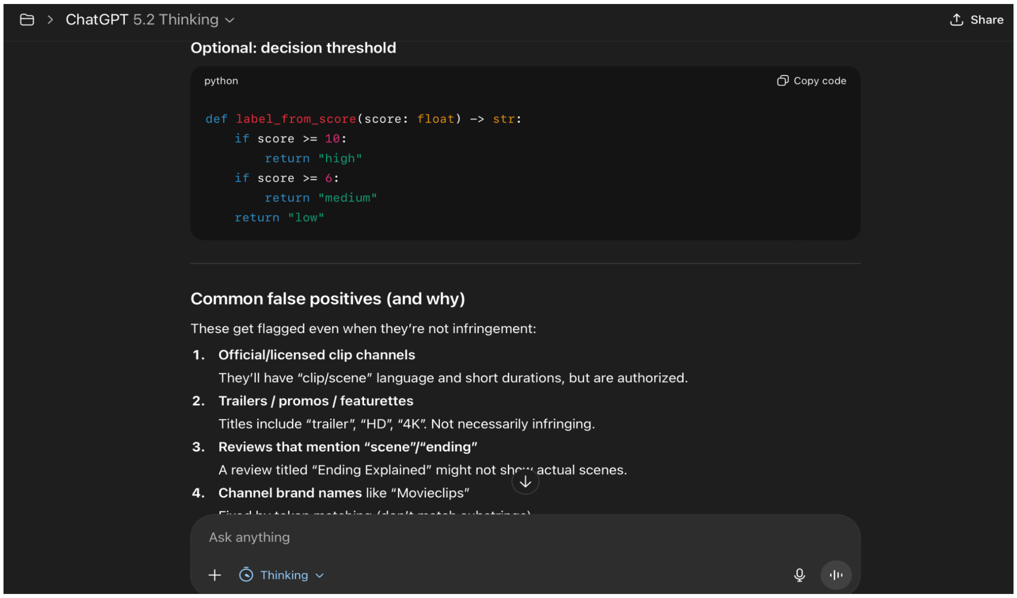
can you give me the list of keywords for movie scene detection and sample code to compute risk score for identifying likely movie-clip uploads from YouTube metadata like title/description/duration also explain common false positives/false negatives."

Here's a solid, **explainable** keyword set + a sample `compute_risk_score()` you can drop into your notebook. I'm keeping it practical for interview use: easy to justify, easy to tune.

Keyword list for "movie scene / clip" detection (metadata-based)

A) Clip/scene intent keywords (strong signals)

- `clip`, `clips`
- `scene`, `scenes`
- `full scene`
- `movie scene`, `film scene`
- `ending`, `final scene`, `ending scene`
- `opening scene`, `intro scene`
- `best scene`, `best scenes`, `best moments`



Optional: decision threshold

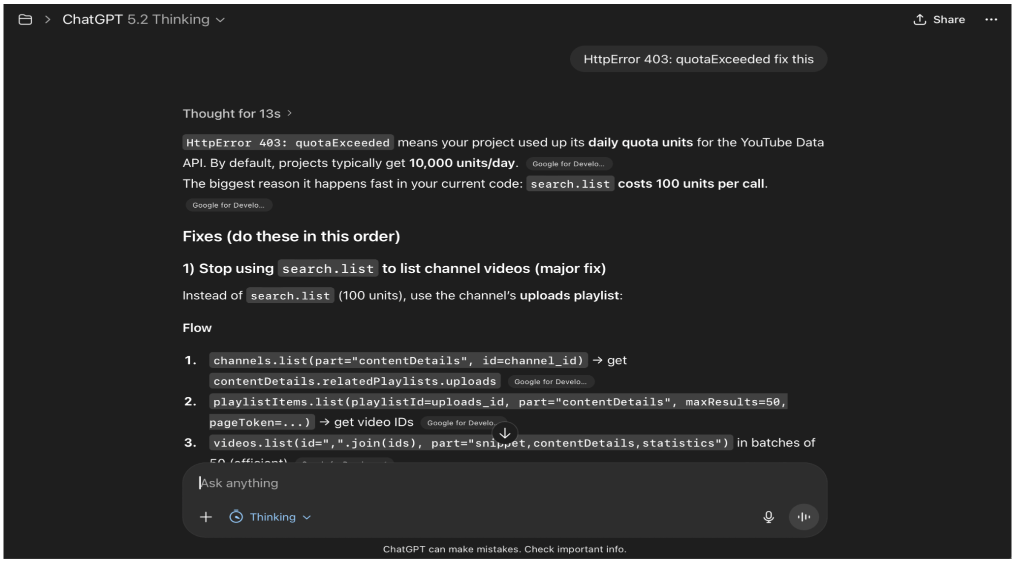
```python
def label_from_score(score: float) -> str:
    if score >= 10:
        return "high"
    if score >= 6:
        return "medium"
    return "low"
```

Common false positives (and why)

These get flagged even when they're not infringement:

1. **Official/licensed clip channels**
   They'll have "clip/scene" language and short durations, but are authorized.
2. **Trailers / promos / featurettes**
   Titles include "trailer", "HD", "4K". Not necessarily infringing.
3. **Reviews that mention "scene"/"ending"**
   A review titled "Ending Explained" might not show actual scenes.
4. **Channel brand names** like "Movieclips"

I needed an explainable heuristic detector that works using only YouTube metadata (title, description, duration) to prioritize likely movie-clip posts. I asked for a keyword set to seed scoring rules, a sample scoring function that avoids common substring mistakes, and a clear discussion of false positives/negatives so I can justify the approach and limitations.



HttpError 403: quotaExceeded fix this

Thought for 13s

`HttpError 403: quotaExceeded` means your project used up its **daily quota units** for the YouTube Data API. By default, projects typically get **10,000 units/day**. Google for Develo...
The biggest reason it happens fast in your current code: `search.list` **costs 100 units per call**. Google for Develo...

Fixes (do these in this order)

1) Stop using `search.list` to list channel videos (major fix)

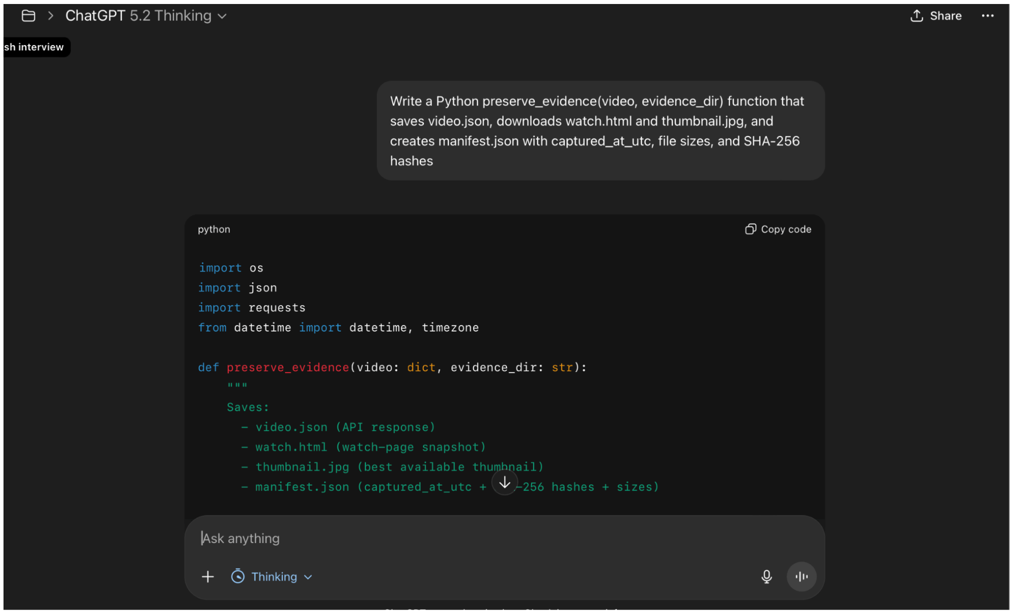
Instead of `search.list` (100 units), use the channel's **uploads playlist**:

Flow

1. `channels.list(part="contentDetails", id=channel_id)` → get `contentDetails.relatedPlaylists.uploads` Google for Develo...
2. `playlistItems.list(playlistId=uploads_id, part="contentDetails", maxResults=50, pageToken=...)` → get video IDs Google for Develo...
3. `videos.list(id=",".join(ids), part="snippet,contentDetails,statistics")` in batches of

ChatGPT can make mistakes. Check important info.

I encountered a hard failure where the YouTube API stopped responding due to exceeded daily quota units. I needed a concrete mitigation plan that keeps the pipeline end-to-end, reduces unit consumption, and adds robustness (backoff/caching) so the tool functions reliably in real usage.

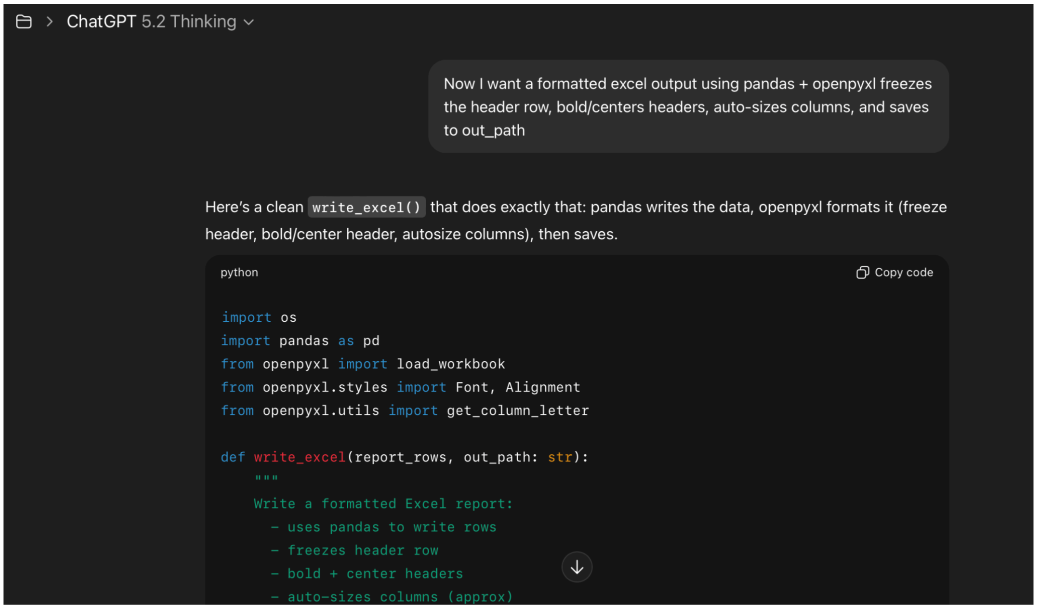


Write a Python preserve_evidence(video, evidence_dir) function that saves video.json, downloads watch.html and thumbnail.jpg, and creates manifest.json with captured_at_utc, file sizes, and SHA-256 hashes

python

```python
import os
import json
import requests
from datetime import datetime, timezone

def preserve_evidence(video: dict, evidence_dir: str):
    """
    Saves:
      - video.json (API response)
      - watch.html (watch-page snapshot)
      - thumbnail.jpg (best available thumbnail)
      - manifest.json (captured_at_utc + -256 hashes + sizes)
```

I needed the core evidence preservation function for my pipeline to store per-video artifacts (API JSON, page snapshot, thumbnail) and generate a manifest with timestamps and SHA-256 hashes so I can verify integrity and reference the stored evidence paths in the Excel output with minimal manual work.



Now I want a formatted excel output using pandas + openpyxl freezes the header row, bold/centers headers, auto-sizes columns, and saves to out_path

Here's a clean `write_excel()` that does exactly that: pandas writes the data, openpyxl formats it (freeze header, bold/center header, autosize columns), then saves.

python

```python
import os
import pandas as pd
from openpyxl import load_workbook
from openpyxl.styles import Font, Alignment
from openpyxl.utils import get_column_letter

def write_excel(report_rows, out_path: str):
    """
    Write a formatted Excel report:
      - uses pandas to write rows
      - freezes header row
      - bold + center headers
      - auto-sizes columns (approx)
```

I needed the pipeline to automatically generate a clean, reviewer-friendly Excel deliverable (not raw data). Freezing the header improves navigation, bold/centered headers improve readability, and auto-sizing columns prevents truncated evidence paths reducing manual work and ensuring consistent output.