Case Study 1: AI Instagram Content Generator – Multi-Agent System

1. Executive summary

The project is a multi-agent pipeline that turns gameplay videos and screenshots (from a Google Drive folder) into Instagram-ready posts. It automatically ingests media, extracts frames and transcripts, understands visual & audio content, finds relevant trends, generates multiple caption + hashtag variants using an LLM, runs quality control, and packages the best result for download via a simple FastAPI UI.

Key outcomes:

- Automated end-to-end content generation for short-form social media posts.
- Multi-agent modular design for clear responsibilities and easy extension.

Simple web UI for users to submit a Drive folder and retrieve packaged outputs.

2. High-level features

FastAPI UI at /ui — paste a Google Drive folder link and run the pipeline.

- Content Understanding Agent: video scene/shot detection, keyframe extraction (ffmpeg), optional ASR (Whisper), image captioning (BLIP).
- Trend Analysis Agent: uses ASO seeds and Google Trends (pytrends) to produce trending keywords and a TrendFit alignment score.
- Generation Agent (LLM): uses Gemini (configurable; default gemini-2.5-flash) to produce multiple caption/hashtag variants.
- Quality Control Agent: checks text and media quality (length, hashtags, banned terms, resolution/aspect/duration), evaluates trend alignment.
- Finalize Agent: selects best variant, creates bundle.zip with assets and JSON summaries.

3. Pipeline / orchestration

Orchestration style: lightweight, LangGraph-style state graph. Each agent is a node:

ingest → content_understanding → trend_analysis → generation → qc → finalize.

Supports conditional branching (e.g., QC pass/fail), retries, timeouts, and job isolation (storage/<job_id>/).

Orchestrator runs the graph and records state for each step.

4. Project structure (file map)

```
app/
├── agents/
│   ├── content_understanding_agent.py
│   ├── trend_agent.py
│   ├── generation_agent_llm.py
│   ├── qc_agent.py
│   └── finalize_agent.py
├── graph/
│   └── flow.py
├── llm/
│   └── gemini_llm.py
├── services/
│   ├── video.py
│   ├── asr.py
│   └── drive.py
├── templates/
│   ├── form.html
│   └── results.html
├── main.py
├── orchestrator.py
├── storage/
├── .env
└── requirements.txt
```

5. Agent responsibilities

- **Content Understanding Agent**
    - ffmpeg for video scene & keyframe extraction
    - **Whisper** (optional) for transcripts / SRT
    - **BLIP** (Salesforce/blip-image-captioning-base) for frame captions & lightweight tag extraction
- **Trend Analysis Agent**
    - Seeds from ASO keywords + description
    - **Google Trends / pytrends**
    - **TrendFit score**: Sentence-Transformers (all-MiniLM-L6-v2) for caption ↔ trend alignment
- **Generation Agent (LLM)**
    - **Gemini** (default: gemini-2.5-flash) generates multiple caption/hashtag variants
- **Quality Control Agent**
    - Text: length bands, hashtag count, repetition/spam, banned terms
    - Media: resolution, aspect ratio, duration, bitrate (ffprobe)
    - Trend alignment via TrendFit score
- **Finalize Agent**
    - Selects the best variant
    - Produces hashtag list and summary
    - Packages results into bundle.zip

6. Key technologies & models

FFmpeg / ffprobe — video processing and keyframe extraction.

Whisper (optional) — automatic speech recognition and SRT generation.

BLIP (Salesforce/blip-image-captioning-base) — image captioning and tag extraction.

Sentence-Transformers (all-MiniLM-L6-v2) — TrendFit embedding comparisons.

Gemini (LLM) — caption and hashtag generation pytrends — Google Trends queries for trend discovery.

FastAPI — web UI endpoints and orchestration entrypoints.

Zip packaging — produce downloadable bundles.

7. Inputs / expected Drive folder contents

The Google Drive folder should include:

gameplay.mp4 (or other video file)

Multiple screenshots (*.jpg, *.png)

aso_keywords.txt (seed keywords)

description.txt (app/game description)

The UI expects a Drive folder link and then runs the pipeline automatically.

8. Outputs

Files produced under storage/<job_id>/results/:

results/captions.json — all generated caption variants.

results/scores.json — QC and TrendFit scores for each variant.

results/trends.json — fetched trending terms and seed alignment.

results/summary.json — chosen caption/hashtag and metadata.

results/bundle.zip — packaged assets: selected images, captions, hashtags, SRT (if any), and JSON summaries.

QUESTIONS:

- Which visual/video processing models did you use, and why?

I use ffmpeg/ffprobe for scene detection and keyframe extraction because it's fast and reliable, BLIP for frame captioning as it offers a good quality/latency trade-off, and Whisper optionally for ASR. I considered BLIP-2/Florence-2 (better quality but heavier) and Faster-Whisper/Vosk (lower-latency/offline), but chose the current stack for speed and practical cost.

- How do you ensure the trend data is up to date?

I query Google Trends through pytrends with timeframe="now 7-d" and a configurable geo, aggregate related_queries across seeds (ASO + description), and fall back to the normalized seeds if the API fails. If available, I also re-rank results using Instagram/TikTok hashtag stats provided in the assets.

- How do you measure whether the content is aligned with current trends?

I compute a TrendFit score: embed the caption and trend terms with a sentence-transformer, take cosine similarities, and average the top-5 values to produce a 0–100 score. Content below a threshold (e.g. <60) is flagged for automatic revision or review.

- What does the quality control agent evaluate? Provide two problematic examples.

My QC checks text (length, banned words, repetition, hashtag rules), media (resolution/aspect 9:16, duration, fps, audio via ffprobe), and trend alignment (TrendFit). Examples:

• "FREE HACK — unlimited coins!!! " — banned words + spam.

• Video 640×640 (2s) + 12-char caption — fails media and caption rules.

- What techniques did you use to understand and analyze the video content?

I use ffmpeg/ffprobe for scene/shot detection and keyframe extraction, BLIP for per-frame captions/tags, and Whisper (optional) for transcripts. Seeds are built from ASO + description + BLIP tags + transcript, and frame selection is dynamically budgeted with CU_MAX_FRAMES / PROC_MAX_FRAMES.