

Grounded Video Analysis for Public-Speaking Feedback

Raghav Rajesh, Technical Researcher, PitchMe
June 2026

SUMMARY

PitchMe gives speakers written feedback on how they deliver a talk. Producing that feedback by sending the whole video to a vision-language model is expensive, and an earlier attempt to cut the cost by passing raw pose coordinates to a text model produced vague and partly invented comments. This report describes a pipeline that separates measurement from language. Code detects movement events in MediaPipe Pose landmarks and a text model is allowed only to describe those events, while a single vision call on at most eight keyframes covers facial expression and gaze. The pipeline was implemented in full, with 58 passing unit and integration tests, and produced a complete report for a 20-second test clip for \$0.00092, more than ten times below the \$0.01 target. Its accuracy was low: on a deliberately difficult clip it recovered 2 of 11 annotated events (recall 0.18, precision 0.14). Error analysis traces most of the misses to the choice of sensor rather than to tuning, because MediaPipe Pose does not track fingers and the detector does not register brief head turns. I therefore recommend closing this line of work, and the report sets out which parts are worth reusing and how a successor should scope the problem before choosing a model.

1 Introduction

PitchMe is a public-speaking coaching product. A user uploads a recording of themselves speaking and receives a structured report on their delivery, covering posture, hand gestures, head movement, facial expression and gaze. The report needs to point to specific moments in the video, because a comment such as “you turned away from the audience at 0:12” helps a speaker in a way that general advice does not.

The two obvious ways of producing this report both fall short. Sending the full video to a vision-language model (VLM) covers every category, but at an estimated \$0.10–0.50 per video it is too expensive to offer at scale. A cheaper prototype, built before this work, extracted body landmarks with MediaPipe Pose and passed the per-frame coordinates to a text model. The text model could not follow numerical change across hundreds of frames. Its output was generic (“maintains steady posture” whatever the video showed), its time ranges were ten seconds or longer, and some categories contained events that had not happened.

The work in this report starts from a division of labour: code should make the measurements, and the language model should only put them into words. A detector finds candidate movement events in the landmark signals, each with exact start and end times, and the text model narrates those candidates and nothing else. Facial expression and gaze, which pose landmarks cannot measure, are handled by one small vision call on a few selected frames. Three questions guided the work:

- RQ1** Grounding. Does replacing “raw coordinates in, model guesses out” with “detected events in, model narration out” give more accurate and better-grounded feedback?
- RQ2** Keyframe vision. Can about eight keyframes per video, sent to a vision model, cover facial expression and gaze at around one-hundredth of the cost of a full-video VLM pass?
- RQ3** Cost. Can the combined pipeline produce a complete coaching report for under \$0.01 per video using gpt-4.1-nano?

The answers are mixed. The cost target was met by a wide margin and the vision call stayed within its budget. Grounding did what it was designed to do, in that the text model could no longer add events or change their timing, but it also limited the report to what the detector could see, and on a stress-test clip the detector saw little. Section 2 describes the pipeline, Section 3 its implementation and Section 4 the evaluation. Section 5 gives the recommendation and Section 6 the guidance for anyone who restarts this work. Section 7 documents a separate tool from the same internship, which turns recorded talks into training data for a delivery-scoring model. Appendix A lists where the code and data are kept and how to run both projects.

2 System design

The pipeline takes an MP4 recording of one speaker and returns a JSON report with a fixed schema. Two stages run locally at no cost: landmark extraction and event detection. Two narrow API calls then run in parallel, one to narrate the movement events and one to describe the face and gaze in a few keyframes. Their outputs are checked against what was actually measured or sent, merged, and written out with a cost breakdown (Figure 1). The subsections below follow the stages in order.

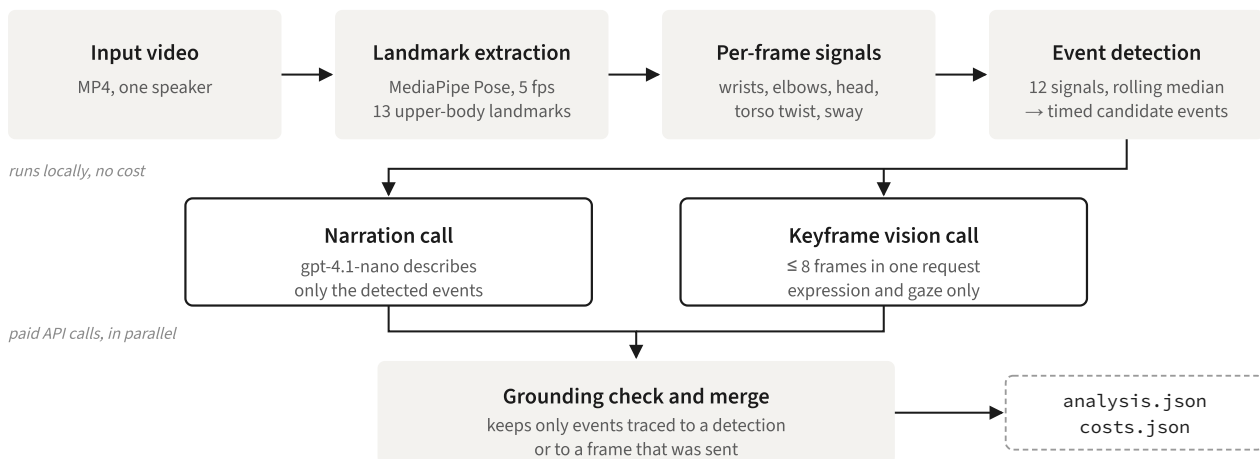


Figure 1. Pipeline overview. Shaded boxes run locally at no cost; the two outlined boxes are the only paid API calls. Keyframes are chosen from the peaks of detected events, so the vision call depends on the detector as well as on the video. The grounding check discards any output that cannot be traced to a detected event or to a frame that was sent.

2.1 Landmarks and per-frame signals

The detector needs signals that describe body movement independently of where the speaker stands in the frame. The video is sampled at 5 frames per second and each frame passes through the MediaPipe Pose Tasks API. Of the 33 landmarks the model returns, the pipeline keeps the 13

above the waist. Each frame is stored twice: once relative to the shoulders, so that arm and head geometry is unaffected by the speaker’s position, and once in image coordinates (the “global” signal), so that swaying, bobbing and stepping towards the camera remain visible.

From these landmarks, `features.py` computes the per-frame signals used by the detector: wrist position zones and wrist speeds, elbow angles, the distance between the wrists, head yaw, pitch and roll, shoulder roll, torso twist, and sway, bob and approach from the global signal. Head angles are computed only when all five face landmarks have a visibility score above 0.5, which stops the pipeline from reporting head movement when the face is partly out of frame or hidden. Speeds use the actual interval between frames, so they remain correct if the sampling rate changes.

2.2 Event detection

A fixed threshold on a raw signal treats an animated speaker and a still one in the same way, so the detector instead compares each signal with the speaker’s own recent behaviour. `detect.py` holds a table of 12 signals, each with its own threshold and minimum duration. For every signal it computes a rolling median over the previous 9 to 15 frames and flags frames that deviate from it by more than the threshold. Hysteresis prevents one movement from being split into many short events, and spans shorter than the minimum duration (0.4 s for head movement) are discarded. A separate pass records changes of wrist zone, such as a hand rising from waist height to above the shoulder. The output is a list of candidate events, each with start, end and peak times and the measured values behind it. Because “notable” is defined relative to the speaker, and because every candidate carries its measurements, each statement in the final report can cite the number that produced it.

2.3 Grounded narration

The earlier prototype showed that a text model asked to find events in coordinate data will invent them, and a prompt rule that required one event per category in every time segment made this worse by forcing it to produce filler. In the new design the candidates are serialised as compact JSON and sent to `gpt-4.1-nano` with a system prompt that asks it to describe each detected event, explain what it means for the speaker, and cite the measured values. `llm_analyze.py` then filters the reply: an event is kept only if it matches a detected candidate in both category and timing. The model therefore cannot add events, move them in time or merge two of them, and a category with no candidates stays empty rather than being filled. As a result, any error in the movement section of the report can be traced to the detector, where it can be measured and corrected.

2.4 Keyframe vision call

Pose landmarks include only five points on the face (the nose, eyes and ears), which is too few to judge expression or gaze, and a full-video VLM pass would bring back the cost problem. `keyframes.py` therefore selects at most eight frames per video. Frames at the peaks of detected events, such as the largest head turn or the fastest wrist movement, are chosen first because they are the moments most worth checking visually; the remaining slots are spread evenly across the video, and no two frames are less than 5 s apart. The selection is deterministic. `vlm_analyze.py` reads each frame with OpenCV, downscales it and sends all of them in a single request. The

model must answer from a fixed vocabulary (expression: neutral, smile, frown, tense, surprised or other; gaze: camera, left, right, up or down), and any answer that uses another label or refers to a timestamp that was not sent is discarded. If the call still fails after one retry, the face and gaze sections are left empty and the movement report is delivered as normal, so a problem with the vision call never blocks the rest of the report.

2.5 Output schema and cost accounting

`pipeline.py` merges the two results into a report with eight fixed top-level keys. Three come from the movement path (`postureAndBodyChanges`, `handGestures`, `headDirectionAndMovements`) and two from the vision call (`facialExpression`, `eyeMovements`). The other three are placeholders: `audioEvents` and `transcription` stay empty because audio analysis was out of scope, and nothing currently maps to `otherEventsInVideo`. Each event records its start and end times (as MM:SS and in seconds), a label that pairs the observation with its coaching implication, and the measured evidence. Every API call is priced against a dated table of June 2026 rates, and the totals are written to `costs.json` alongside the report.

3 Implementation

The pipeline lives in `pitchme-research/Third Trial/` and was built from an 18-task implementation plan, all of which was completed. It runs on Python 3.13 with MediaPipe 0.10.35. Table 1 lists the modules.

Table 1. Modules in `src/`, in the order the pipeline uses them.

Module	Responsibility
<code>extract_landmarks.py</code>	Runs MediaPipe Pose at 5 fps; stores shoulder-relative and global coordinates for 13 landmarks
<code>features.py</code>	Geometry helpers and the per-frame signal timeline, including the face-visibility gate
<code>detect.py</code>	Threshold table, rolling-median detector with hysteresis, wrist-zone transitions, candidate assembly
<code>encoders.py</code>	Serialises candidate events as compact JSON for the prompt
<code>prompt.py</code>	Narration-only system prompt: describe the detected events, cite evidence, add nothing
<code>llm_analyze.py</code>	Narration call and the movement grounding filter; fills MM:SS times
<code>keyframes.py</code>	Keyframe selection: event peaks first, then even fill; budget 8, minimum gap 5 s
<code>vlm_analyze.py</code>	Frame capture, the single vision call (one retry, empty on failure), vocabulary and timestamp checks
<code>costs.py</code>	Dated price table and per-call cost calculation
<code>pipeline.py</code>	Single entry point: runs the stages, merges the results and totals the cost
<code>main.py</code>	Command-line interface; writes <code>analysis.json</code> and <code>costs.json</code>
<code>eval.py</code>	Scores a report against hand annotations: per-category matching, recall, precision, mean IoU

The test suite has 58 tests in 13 files and covers every module. The unit tests make no network calls, and one integration test runs from detected candidates through narration to a grounded report with the API mocked. Two defects found during the work were fixed:

- `gpt-4.1-nano` sometimes omits a category it has nothing to report instead of returning an empty list. The original code treated a missing key as fatal and aborted the job. Post-

processing now fills every schema key, and only a reply that is not a JSON object raises an error (commit a854c89).

- The minimum-duration check in `detect_events` compared floating-point spans directly, so a span such as 2.4–2.8 s, which evaluates to 0.3999..., was discarded as shorter than 0.4 s. A tolerance of 10^{-9} was added to the comparison (`detect.py`, line 79). This fix was made after the evaluation in Section 4, so the scores reported there do not include it.

4 Evaluation

4.1 Setup

All results come from one 20-second clip that was chosen to be difficult. It contains rapid head turns (0–7 s), finger counting (7–12 s), a circling “speed-bag” movement of the fists (12–16 s) and pulsing fists (16–19 s). I annotated 11 events by hand: 4 head movements (turns, a look upwards and a return to centre) and 7 hand gestures. `eval.py` matches predicted events to annotated ones greedily within each category, counting a pair as a match when their temporal intersection over union (IoU) is at least 0.3, and reports recall, precision and the mean IoU of matched pairs, both overall and per category. The pipeline was run on 16 June 2026 and scored against these annotations the same day.

The earlier coordinate-based prototype was run on the same clip before the redesign, but its output was not saved, so the comparison with it is qualitative. Because the evaluation uses a single clip annotated by the developer, its results should be read as a diagnosis of where the design fails rather than as a measure of typical accuracy.

4.2 Cost

Table 2. API cost of one full run on the 20 s test clip (gpt-4.1-nano for both calls, June 2026 prices).

Component	Cost (USD)
Narration call	0.00069
Vision call, 4 keyframes	0.00023
Total per clip	0.00092
Target	0.01000

A complete report cost \$0.00092, more than ten times below the target, which answers RQ3 for this clip (Table 2). The vision call accounted for a quarter of the total. For this 20 s clip the selector chose four of the eight possible keyframes, at 2.6, 8.8, 14.2 and 19.8 s.¹

4.3 Accuracy

Table 3. Detection results on the test clip, matched per category at IoU ≥ 0.3 . A dash means the value is undefined because the denominator is zero.

Category	Predicted	Annotated	Matched	Recall $\uparrow$	Precision $\uparrow$
Hand gestures	8	7	2	0.29	0.25
Head direction and movement	0	4	0	0.00	–
Posture and body changes	4	0	0	–	0.00
Facial expression	0	0	0	–	–
Eye movement (gaze)	2	0	0	–	0.00
Overall	14	11	2	0.18	0.14

The pipeline recovered 2 of the 11 annotated events, both of them hand gestures: the raised peace sign at 7–8 s and the speed-bag movement at 12–16 s (Table 3). Both were well placed in time, and the mean IoU of the matched pairs was 0.49. The pipeline missed every head movement and every finger-counting gesture, and 12 of its 14 reported events had no annotated counterpart. Facial expression was correctly left empty, since all four keyframes showed a neutral face.

Compared with the earlier prototype, the kind of error changed. The old output was vague and partly invented; the new output consists of specific, timed events, each backed by a measured value. None of the 12 unmatched events was introduced at the narration step: 10 came from the detector (6 hand gestures and 4 posture changes) and 2 from the vision call. This supports the grounding half of RQ1. It does not show that the feedback is more accurate overall, because the earlier output was never scored.

4.4 Error analysis

The misses fall into three distinct groups, and only the first can be addressed by tuning.

Head movements: a tuning miss. Most head movements in the clip were shorter than the 0.4 s minimum duration, and the 9-frame rolling median absorbed head positions that were held for a moment, so the deviation never lasted long enough to register. The movement itself was sensed: the posture false positives at 0.6–3.6 s and 4.0–4.8 s coincide with the head turns, where the torso-twist signal fired instead of a head signal. Lowering the head minimum duration from 0.4 s to 0.2 s, or lowering its threshold, is the obvious adjustment. It was not tried, and it is likely to add false positives from ordinary posture shifts.

Finger gestures: an architectural gap. MediaPipe Pose tracks wrists and elbows but not fingers, so finger configuration produces no signal in any of the 12 detectors. The vision call could in principle see the hands, but it was limited to face and gaze to control cost, and its prompt instructs it not to comment on hands. Keyframe density was a further limit, since only one of the four keyframes (8.8 s) fell inside the 7–12 s finger-counting window. Three remedies were considered and none was implemented: extending the vision call to hands with denser keyframes, which is cheap but gives up grounding for hand events; adding the MediaPipe Hands model, which keeps grounding but requires substantial engineering; or declaring finger-level gestures out of scope.

Timing fragmentation. The fast sequence between 12 and 19 s was split or clipped at event boundaries, partly because of the floating-point duration bug described in Section 3. For example, the pulsing-fist event was reported at 19.2–20.2 s, just after the annotated end at 19 s, and so did not match.

The clip is close to a worst case for this design, since it is dominated by the two kinds of movement the detector handles worst. A typical talk, with slower and larger movements, would probably score higher, but no second clip was annotated, so that expectation remains untested.

¹ A later run on 30 June 2026, scored against the same annotations, reported 15 events with the same two matches (recall 0.18, precision 0.13) at a cost of \$0.00105.

5 Discussion and recommendation

5.1 Answers to the research questions

RQ1, grounding: partly. Grounding removed invented events and vague timing from the narration step, and every reported movement event can be traced to a measurement. The same rule means the report can contain only what the detector finds. A signal that is missing from the threshold table, or that never crosses its threshold, disappears from the report with no fallback, and on the test clip the detector found little.

RQ2, keyframe vision: yes, within limits. One call on four frames cost \$0.00023, returned answers within the fixed vocabulary and classified the neutral expression correctly. Its two gaze events were false positives, however, and with at most eight frames per video the call cannot follow anything that changes faster than the keyframe spacing.

RQ3, cost: yes. A complete report cost \$0.00092 for a 20 s clip, against a target of \$0.01.

5.2 Recommendation: do not develop this pipeline further

I recommend closing this line of work rather than tuning it, because its main limitation is the sensor rather than the parameters. MediaPipe Pose cannot see fingers, and adding them means either integrating the MediaPipe Hands model, which is a substantial piece of engineering, or giving hand events to the vision model, which breaks the grounding rule. Head movement could be improved by tuning, but brief glances and nods sit close to the detection floor, and lowering it is likely to produce false positives on normal changes in posture. A coaching product that misses every head turn and every finger gesture on a difficult clip, and cannot close either gap without rebuilding its sensor layer, does not meet its core use case. A good score on a second, more typical clip after tuning would strengthen the case for coaching gross body movement alone, but it would not change the conclusion for finger gestures.

Two parts of the codebase are worth keeping. The grounding pattern (detect in code, let the model narrate, filter its output against what was detected) can be applied to any sensor, and the per-call cost tracking made the cost question easy to answer. A future attempt should start from MediaPipe Holistic, which tracks the body, hands and a full face mesh in a single pass, or from a full-frame vision model, rather than from MediaPipe Pose.

5.3 Lessons

- *Do geometry in code.* Language models cannot reliably track numerical change across hundreds of frames, and when asked to, they fall back on safe, plausible phrasing.
- *Do not force coverage.* The rule requiring one event per category in every segment was the main structural cause of generic output in the earlier prototype. An empty category is a correct answer when nothing happened.
- *Check what a sensor can measure before building on it.* Pose landmarks are suitable for gross body movement only; they cannot support judgements about expression, gaze or fingers.
- *Expect model output to vary in shape.* gpt-4.1-nano sometimes leaves out empty keys. Normalising the schema is more robust than rejecting the reply.

6 Guidance for a successor

The priorities below are not tasks for continuing this pipeline. They are questions that any future attempt should answer before writing code, and they draw on advice from two computer-vision researchers, Prof. Dileepa and Prof. Isuru, whom I consulted in June 2026.

1. *Decide what the data is for before choosing a tool.* This project chose MediaPipe Pose first and discovered its limits later. In the experts' analogy, MediaPipe is like fitting ready-made wheels to a car: quick to get moving, but you are limited to the wheels it provides. Building on OpenCV is like building the car yourself, which takes longer but leaves every component under your control. The choice should follow from the problem definition.
2. *Fix the scope precisely.* "Public-speaking coaching" is too broad to design a sensor layer for. Decide which body events matter for coaching and why, how accurate their detection needs to be, and what the report should do when a signal is missing. These are product decisions and should come before the engineering ones.
3. *List the required signals and check model coverage before choosing a model.* Write down every signal the report needs, for example wrist speed, head yaw, finger configuration and eye contact, and confirm that the model's landmark set covers each one. Read the full landmark index and visibility definitions, not only the quick-start guide. MediaPipe Pose provides 33 landmarks, of which this pipeline used 13, and none of them is a finger joint.
4. *Choose between 2D and 3D landmarks deliberately.* This pipeline used 2D coordinates for most of its geometry and the depth coordinate only for torso twist, a mix that was never formally decided. 3D landmarks support depth signals such as leaning in or turning away from the camera, but they are noisier when estimated from a single camera.

Table 4. Options that were considered and should not be pursued.

Option	Reason
Tuning and redeploying this pipeline	The main gaps are architectural (Section 5.2). Restart from a better-scoped sensor layer instead.
Raw-coordinate encoder and A/B comparison mode	Dropped in the 11 June 2026 design and never built.
MediaPipe Face Landmarker blendshapes in place of the vision call	Superseded by the closure of this work. Evaluate a holistic model instead.
OpenAI Batch API	At about \$0.001 per video, batching would add asynchronous complexity for no meaningful saving at research scale.
Audio transcription (Deepgram)	Deferred until after the MVP. The schema fields exist but are always empty.
Multi-person video	MediaPipe Pose tracks only the first person it detects, and this is not an MVP use case.

7 Companion project: a dataset builder for delivery scoring

Alongside the pipeline above, I built a separate tool that turns recordings of public talks into training data for a small model that scores spoken delivery. Human coaches are expensive and slow, and the aim is a model cheap enough to score any clip. The tool relies on knowledge distillation: a large language model scores a few hundred clips once, those scores become training labels, and a small model trained on them can then score new clips at almost no cost. Each clip is scored from 1 (poor) to 5 (excellent) on the four dimensions in Table 5.

Table 5. Scoring rubric. Each dimension is scored from 1 to 5.

Dimension	What it measures
Pacing	Speaking rate and whether the rhythm feels natural; roughly 130–165 words per minute is ideal.
Vocal delivery	How expressive and varied the voice is. A flat, monotone delivery scores low even when the words are good.
Clarity	How easy the speaker is to understand: clean articulation and complete, coherent sentences.
Overall	A holistic judgement of how compelling the delivery is. It is not the average of the other three scores.

A single script, `src/build_dataset.py`, runs the seven stages in Table 6. Stages can be run one at a time or together with `--stage all`, and each stage skips work it has already completed, so the pipeline can be stopped and restarted safely. The scoring stage saves after every clip, so an interruption never loses progress.

Table 6. Stages of the dataset builder.

#	Stage	What it does	Tool
1	download	Downloads the audio of each YouTube video listed in <code>sources.csv</code> as MP3	yt-dlp
2	transcribe	Transcribes each recording with start and end times for every word	Whisper (local)
3	segment	Cuts each talk into 30–60 s clips at natural pauses between sentences	pydub
4	features	Measures speaking rate, pauses, pitch variation, loudness and filler phrases	librosa
5	score	Sends each clip’s features and transcript to Claude for the four scores	Claude API
6	assemble	Merges features and scores and splits them into training, validation and test sets	pandas
7	export	Writes the training files, one clip per line	

The choice of source videos matters as much as the code. Each row of `sources.csv` records a URL, the speaker, a category, a licence and a quality tier. The dataset needs both strong speakers (TED talks, Toastmasters champions, professional keynotes) and amateur ones (student speeches, meetup talks, Three Minute Thesis entries), because a model that only ever sees good speakers never learns what weak delivery looks like. `src/collect_sources.py` adds a whole YouTube channel at once.

Before a clip is scored, the pipeline measures it, and these measurements are given to the scoring model together with the transcript (Table 7). The output in `data/out/` consists of `sft_train.jsonl`, `sft_val.jsonl` and `sft_test.jsonl`, in which each line is one clip written as a short conversation (the measurements and transcript in, the four scores out), and `metadata.csv`, which holds the full dataset. Training the small model means teaching it to reproduce the scores the large model assigned.

Table 7. Features measured for each clip.

Feature	Meaning
<code>wpm</code>	Speaking rate in words per minute
<code>pause_count</code> , <code>longest_pause_s</code>	Number of pauses mid-speech and the longest silence in the clip
<code>pitch_mean</code> , <code>pitch_std</code> , <code>pitch_cv</code>	Pitch and its variation; <code>pitch_cv</code> is normalised so that deep and high voices can be compared fairly
<code>energy_mean</code> , <code>energy_std</code>	Loudness and its variation
<code>discourse_count</code> , <code>discourse_rate</code>	Use of filler phrases such as “you know”, “I mean”, “sort of”, “like” and “basically”
<code>auto_pacing</code>	A rule-based pacing score derived from <code>wpm</code> , used as a sanity check and not as a training label

Three design choices protect the quality of the data. Clips from the same speaker always stay in the same split, which stops a speaker’s voice from appearing in both training and test data and inflating the results. The split ratios default to 80/10/10. Heavy libraries are loaded only by the stage that needs them, so a single stage can run without every dependency installed. Settings that are commonly adjusted, such as the Whisper model size, clip length, the maximum number of clips per source video and the scoring model, live in `config.yaml`. This section describes the tool’s design; it does not report results from a scored dataset or a trained model.

Appendix A. Repository map and how to run

A.1 Video analysis pipeline

Table A1. Location of code, data and records, relative to `pitchme-research/Third Trial/` unless stated otherwise.

Resource	Location
Source code	<code>src/</code>
Tests (58, in 13 files)	<code>tests/</code>
Ground-truth annotations (11 events)	<code>tests/fixtures/ground_truth_WhatsApp.json</code>
Blank annotation template, with instructions	<code>tests/fixtures/ground_truth_template.json</code>
Design specifications	<code>docs/superpowers/specs/</code>
Implementation plans	<code>docs/superpowers/plans/</code>
Product requirements	<code>PRD.md</code>
Decision log	<code>MEMORY.md</code>
Run outputs (local, not committed)	<code>outputs/analysis.json</code> , <code>outputs/costs.json</code>
Test clip (local, not committed)	<code>WhatsApp Video 2026-06-04 at 9.35.55 PM.mp4</code>

```
cd "Third Trial"
.venv\Scripts\Activate.ps1          # Windows PowerShell, Python 3.13.7
pip install -r requirements.txt
# copy .env.example to .env and set OPENAI_API_KEY

python -m src.main <video.mp4> --debug      # full pipeline: movement and vision call
python -m src.main <video.mp4> --skip-llm    # landmark extraction only, no API cost
python -m src.main <video.mp4> --no-vlm     # movement only, no vision call

python -m src.eval <video.mp4> tests/fixtures/ground_truth_WhatsApp.json
                                           # re-runs the pipeline (paid), then scores it
.venv\Scripts\python.exe -m pytest tests/ -q
```

Further options are `--fps`, `--keyframes`, `--vlm-model` and `--outdir`. To score a saved report without paying for another run, call `src.eval.score()` directly on `outputs/analysis.json`. Both models can be overridden with the environment variables `LLM_MODEL` and `VLM_MODEL`.

A.2 Dataset builder

Requirements: Python 3.9 or later and `ffmpeg`. The Anthropic API key is needed only for the scoring stage. Before scoring, check that `config.yaml` names a current model and that the rubric file it reads, `schema.md`, is in the project folder. Transcription and feature extraction use a lot of CPU, which is expected.

```

winget install Gyan.FFmpeg          # Windows
pip install -r requirements.txt
# add ANTHROPIC_API_KEY=<your key> to a file named .env
python -m unittest discover -s src -t src # check the setup

python src/build_dataset.py --stage all
python src/build_dataset.py --stage features # or one stage at a time

python src/collect_sources.py "https://www.youtube.com/@TED/videos" \
    --tier good --speaker-from-title --limit 40

```

Table A2. Project layout of the dataset builder.

File or folder	Contents
src/build_dataset.py	The seven-stage pipeline
src/collect_sources.py	Adds whole YouTube channels to sources.csv
src/test_logic.py	Unit tests for the core logic
config.yaml	Whisper model size, clip length, clips per source, scoring model, split ratios
sources.csv	The list of videos to process
requirements.txt	Python dependencies
data/out/	Output files (not committed)