Olympus Engine

The Video Review Kit

Give Claude real sight and sound on video. Hand the whole file to Gemini, bring its read back, and check the second it flags.

About this document. This is reference material. It describes a small tool that hands a video to Google's Gemini and brings Gemini's read back to Claude. Check each command against the docs for the program it runs (Node, npm, Claude Code) and decide for yourself. It carries no keys, accounts or files of ours. When you use the method, your files go to Google, so read the "Honest limits" section before you start.

01 What Claude misses on video

Claude is very good with one still picture: a screenshot, a photo, a single ad. Video is different. Claude takes in pictures, not sound. Hand it a video and the best it can do is look at a few still frames pulled out of it.

So it misses the things that make a video work or fail:

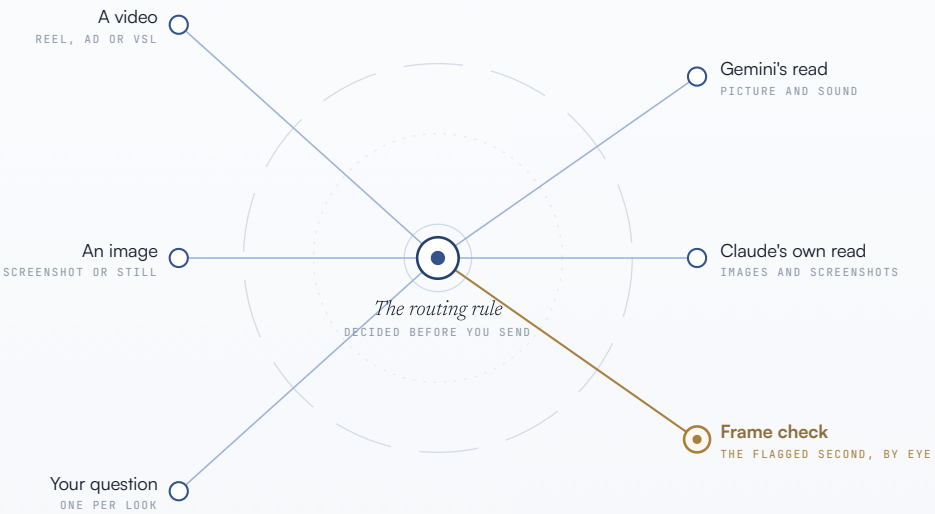
- the spoken hook in the first seconds
- the music and the sound effects
- the pacing, and where the cuts land
- anything that only happens in motion

If you ask Claude to review a reel that way, it is guessing from a handful of screenshots.

OLYMPUS ENGINE :: HOW IT ROUTES

Stills stay home, *video goes out*

You and Claude follow one rule. Single images stay with Claude. Video goes to Gemini through a small tool, and Gemini watches it with the sound on. A big image batch goes out only on a paid key.



BLUE CARRIES THE ROUTING. GOLD MARKS THE CHECK.

01

Stills stay home

Claude already sees one picture well. Keep single images and screenshots with it.

02

Video goes out

Gemini watches the whole file with the sound on.

03

One question per look

Dense, specific questions get better reads than vague ones.

04

Check the flagged second

Look at it yourself before you change anything.

© 2026 Atlas Saint Enterprises · Powered by Olympus

02 The fix in one picture

Gemini can watch a whole video, picture and sound together. So the fix is a handoff:

1. You ask Claude a question about a video.
2. A small tool sends the whole file to Gemini with your question.
3. Gemini watches it, with sound, and writes its read.
4. The read comes back to Claude, and Claude works from it.

Claude stays in charge of the conversation. Gemini is the one that watches.

03 The routing rule

Use the right eyes for the job.

- **One image, a screenshot or a PDF: keep it with Claude.** Claude already sees a still picture well. Sending it out uses up a request (each call to Gemini counts toward your daily limit) and gains nothing.
- **Any video, short or long: send it to Gemini.** This is the job the tool exists for.
- **A big batch of images:** send it to Gemini only if your key is on a paid plan. On the free plan, put the images together into a few contact sheets (a grid of small pictures in one image) and let Claude read those.

If an image goes to the tool by mistake, it still works. It is just the wrong tool for it.

04 Set it up

This takes about fifteen minutes if Node and Claude Code are already on your computer.

What you need

- Claude Code (or another Claude app that can use MCP tools; MCP is the way Claude plugs in outside tools).
- Node, version 20 or newer.
- A Gemini API key (a secret code, like a password, that lets the tool use Gemini on your account). Make one in Google AI Studio (aistudio.google.com). It is free to start.

Step 1. Build the tool. The tool is three small scripts plus a `package.json` and a `.gitignore`. The section "Build spec" at the end of this document says exactly what they do. You can write them yourself, or open an empty folder in Claude Code, give it this document, and ask it to build the tool the build spec describes. Read what it writes before you run it.

© 2026 Atlas Saint Enterprises · Powered by Olympus

Step 2. Install its two packages. In the tool's folder, run `npm install`.

Step 3. Give it your key. Two safe ways. Pick one.

- **Easiest:** a file named `.env` in the tool's folder, holding one line that starts with `GEMINI_API_KEY=` and ends with your key. Make sure your `.gitignore` lists `.env` so the key never gets committed.
- **Or:** set `GEMINI_API_KEY` as an environment variable on your computer (a setting your computer hands to every program it starts).

Never put the key in Claude's MCP settings, and never paste it into a chat.

Step 4. Pick your model. Open Google's models page (ai.google.dev/gemini-api/docs/models), pick a current Flash model, and add a second line to your `.env` that starts with `GEMINI_MODEL=` and ends with that model's name. If you set the key as an environment variable, set `GEMINI_MODEL` the same way. Do this even if the tool already has a default. Claude often suggests a model name from memory, and Google retires models: an old name fails with a "404" that says the model is no longer available to new users.

Step 5. Try it once from the terminal. In the tool's folder, run it on a short video with someone talking, with the real path to your video (its full address on your computer):

```
node watch.js -q "What is said in this video?" "/full/path/to/short-video.mp4"
```

If the answer quotes the words, sight and sound are working.

Step 6. Connect it to Claude Code. Register the MCP server (the small program Claude Code starts so it can reach the tool) once, for your user, with the full path to the server file:

```
claude mcp add --scope user video-review -- node "/full/path/to/watch-mcp-server.js"
```

Do not add your key to this command. The server reads it from your `.env` file or your environment.

Step 7. Restart Claude Code all the way. Quit it and open it again. A running Claude Code does not pick up a new tool or a new environment variable until it starts fresh.

05 Your first review

`watch_media` is the name of the one tool the kit adds to Claude. Pick one short reel and paste this to Claude, with the real file path:

```
Use the watch_media tool on /path/to/reel.mp4. Ask it: "Watch this video with the sound on.
Give me a timestamped breakdown: the hook in the first three seconds (what is said and what
is shown), every cut with its time, every piece of on-screen text, what the voiceover says,
what the music does, and where the pacing speeds up or slows down." Then give me your notes
on what to keep and what to change.
```

© 2026 Atlas Saint Enterprises · Powered by Olympus

Then check the flagged second. Gemini will point at moments: a line that is hard to read, a cut that comes too late, small print that seems to be missing. Before you change anything because of a flag, look at that exact second yourself. Pause the video there and take a screenshot, then give the screenshot to Claude and ask what it sees. One quick look settles it.

For example, say the read tells you the small print at the end is missing. You pause at that second and it is right there, plain to read. Without the check, you would have fixed something that was never broken.

06 Five questions that work

Dense, specific questions get better reads than vague ones. Ask one question per look.

1. **Timestamped breakdown.** "Give me one line per beat: the time, what we see, what we hear, and any on-screen text."
2. **Score a batch.** "Score each video from 1 to 10 on how well the first three seconds stop the scroll, with one line on why." Use the tool's `each` mode so every file gets its own answer. For example, say you have five launch videos and want to know which one to post first: one question, five scores, and a reason for each.
3. **Compare two openings.** "Compare these two videos on the first two seconds only. Which opening pulls harder, and why? Quote what is said and shown in each."
4. **Pull every text overlay.** "List every on-screen text overlay with its start time, end time and exact words, as JSON." Turn on the tool's `json` option.
5. **Find where people leave.** "Flag every second where a viewer would likely swipe away, and say what is happening in the picture and in the sound at that moment."

07 Honest limits

- **It is one model's read, and it can be wrong.** Treat a flag as a place to look, not as a verdict. Small print and small numbers are the easiest things for it to misread.
- **It looks at about one frame each second** (Google's video page says the model samples one frame per second). A flash cut (a shot that is on screen for less than a second) can slip past it, which is one more reason to check the flagged second.
- **Very long videos can hit Gemini's limits.** Google's video understanding page lists the current ones.
- **The free plan has a daily cap for each model.** Big batches use it up fast. Check your own limits in Google AI Studio, because they change by model and by plan.
- **What you send leaves your computer and goes to Google.** On the free plan, Google's terms say it may use what you send to improve its products, and that people may read it. On a paid plan, the terms say it does not use your prompts or files to improve its products. Rules can differ by region. Read Google's current Gemini API terms for your plan, and never send media you are not allowed to share, such as unreleased work or someone else's private files.

© 2026 Atlas Saint Enterprises · Powered by Olympus

- **Google keeps uploaded files for a short time.** Its Files API page (the Files API is the part of Google's service that holds uploaded videos) says uploads are stored for 48 hours and then deleted.

08 When it breaks

- **The tool cannot find your key, or does not show up in Claude.** A new environment variable or a new tool is invisible until Claude Code fully restarts. The `.env` file avoids the restart for the key. Run `claude mcp get video-review` to see whether the server connected, and check the path in your `claude mcp add` command.
- **A big batch stops or hangs.** You probably hit the free plan's daily cap. Google's rate limits page says limits count per project (the account folder your key was made in), not per key, so a second key in the same project does not help. The same page says the daily cap resets at midnight Pacific time. Wait, add billing, or switch the images to contact sheets.
- **A "404" that says the model is no longer available to new users.** The model in your setting has been retired or closed to new keys. It can still show up in Google's model list and fail when you call it. Pick a current model from Google's models page, set it with `GEMINI_MODEL`, and test it with one short call.
- **"Timeout processing."** The file sat in Google's processing step too long. Try a shorter or smaller file, or try again later.
- **"Failed to process."** Usually an odd or damaged file. Export it again as a standard MP4 and retry.
- **An error about the key or the request (400 or 403).** Retrying will not help. Check that the key is right and the file path exists.

09 Build spec

This section describes the tool as plain behavior, so you or your own Claude can write it and check it. It is a description, not code to trust. Read it, change what you like, and review whatever gets written.

The pieces. One shared core and two ways in:

- `watch-core.js`: all the logic.
- `watch.js`: a command line tool that calls the core.
- `watch-mcp-server.js`: an MCP server with one tool, `watch_media`, that calls the core. Server name `video-review`.

Node 20 or newer, with two packages pinned to exact versions in `package.json`: Google's official Gemini SDK for JavaScript (`@google/genai`) and the MCP SDK (`@modelcontextprotocol/sdk`). Nothing else.

Google's Interactions API page recommends that newer API for new projects and says the older `generateContent` call is still fully supported. Either one works for this tool; Google's current SDK docs show both.

The tool's inputs

INPUT	MEANING
`question`	Required. What to ask about the files.
`paths`	Required. Files or folders. A folder expands to the images and videos directly inside it.
`mode`	`together` (default): all files in one request, one answer, good for comparing. `each`: one request and one answer per file, good for scoring a batch.
`json`	Optional. When true, add "Respond only with valid JSON." to the question.
`model`	Optional. Overrides the model for this call.

The command line tool takes the same things. The question goes after `-q` and the file or folder paths follow it. Its three options are `--each`, `--json` and `--model`.

How each file goes to Gemini.

- An image under 18MB goes inside the request itself. Google's docs set a size limit on files sent inside a request, and keeping inline images under 18MB stays well inside it.
- A video of any size, or any file of 18MB or more, goes through Google's Files API: upload it, then check its state every 3 seconds until it is ready. Stop after 8 minutes with the error "timeout processing" and the file name. If Google reports the file as failed, stop with "Gemini failed to process" and the file name.
- In `each` mode, a failure on one file is reported for that file and the rest still run.

Errors and retries.

- Retry only errors that say to try again: 429 (too many requests) and 503 (busy), plus other 5xx server errors. Try up to four more times and wait longer before each try. Google's JavaScript SDK includes a retry option that does this; the tool uses that option or its own loop, never both.
- Never retry a 400 or 403. Those mean a bad key or a bad request.
- Never switch to a different model on its own. If the model fails, say so and name the model.

The model. From the `model` input, else the `GEMINI_MODEL` environment variable or the `.env` line, else a default. The default is not picked from memory, because model names go stale: it comes from Google's current models page, or is left blank for you to set, and it is tested with one call before anyone relies on it.

Exiting. On an error, the tool prints it, sets the exit code and lets the process end by itself. Forcing an exit right after a failed network call can print a harmless but alarming crash line on Windows.

The key. From the `GEMINI_API_KEY` environment variable, else from a `.env` file in the tool's own folder (read only the `GEMINI_API_KEY` and `GEMINI_MODEL` lines). Find the `.env` file by the script's location, not the folder you run it from, so the MCP server finds it no matter where Claude starts it. Never read the key from anywhere else, never print it, never write it to a log.

© 2026 Atlas Saint Enterprises · Powered by Olympus

What it talks to. Only the Gemini API. No analytics, no other network calls. The MCP server talks to Claude over stdio only, so it writes any log lines to stderr, never to stdout.

Files in the folder. The three scripts, `package.json`, a `.gitignore` that lists `.env` and `node_modules`, and your own `.env`.

A quick test. Run the command line tool on a short video where someone says a clear sentence, and ask what was said. The answer should quote the sentence. Then run it on one image to see that images work too, even though they belong with Claude.

---

10 Why we made this

A lot of people online sell AI advice and have never built anything with it. We build and run AI inside multiple seven-figure operations every day, and what is in this guide comes from what broke there.

Follow @olympus.engine on Instagram for more tools like this one.

Behind every lasting thing
is an unseen order.



The same guide is in the companion file video-review-kit.md. Video-Review-Kit-Blueprint.pdf shows the wiring on one sheet, and Video-Review-Kit-Rite.pdf says why the method exists. Google changes its models and limits often, so check its current docs before you rely on any limit named here.

olympusengine.io

© 2026 Atlas Saint Enterprises · Powered by Olympus