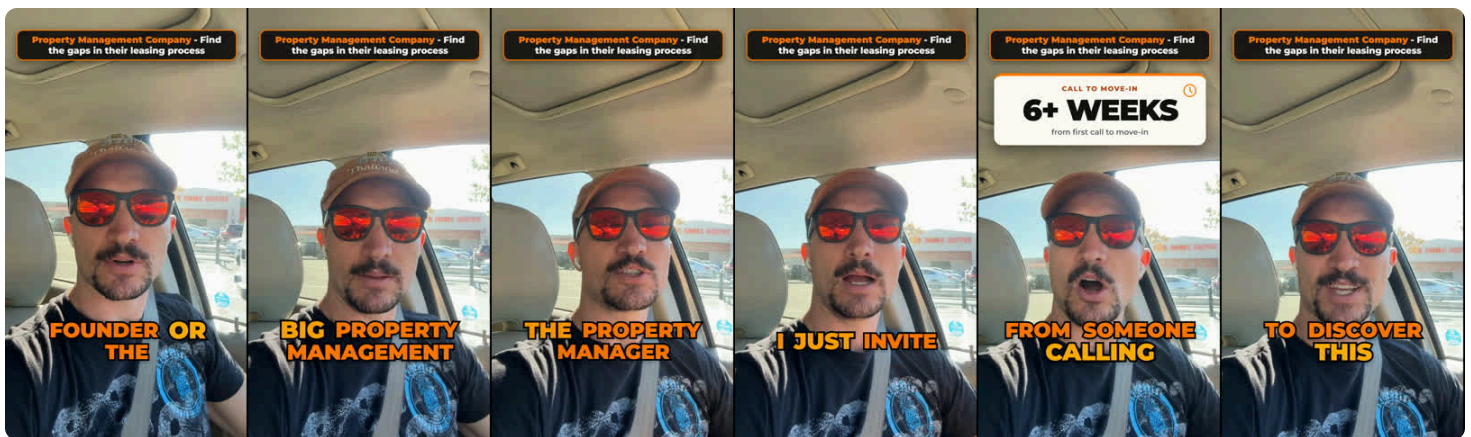


Edit your talking-head videos the Woofid way, with Claude Code

This PDF is written for your Claude Code. Give it this file and a phone video of you talking to the camera, and it will produce the same edit Woofid uses on its own videos: bright orange word-by-word captions, a headline box across the top, and animated graphics that appear when you say the matching thing.



For the creator: how to use this

1. Film yourself on your phone, held **vertical**, talking to the camera (1 to 3 minutes works best).
2. Make an empty folder on your computer, for example `woofid-edits`, and open Claude Code in it.
3. Drag this PDF and your video into the Claude Code window, and type:
`Follow the Woofid Edit Kit PDF to edit my video.`
4. Answer any question it asks (usually just to confirm the headline text), then wait. The first video takes longer because it installs the tools. Your finished video and its thumbnail land in the `out` folder.

What you need: a Windows or Mac computer with Claude Code. Claude Code installs the rest (Python 3.10+, ffmpeg, Node.js 20+). An NVIDIA graphics card makes the transcription faster, but it is not required.

For Claude Code: the procedure

You are editing a creator's vertical talking-head video so that it matches the Woofid house style exactly. The look is fixed by the script in Appendix A: **do not change its CSS, colors, fonts, sizes, positions or animation timings**. Consistency across creators is the point. Your job is the editorial part: where the video starts and ends, the headline text, transcript fixes, and which graphics appear when. Work through these steps in order.

Step 1 • Set up the kit (once)

1. In the working folder, create `woofid_edit.py` with the exact contents of **Appendix A**. Copy it verbatim: every line, the same indentation (4 spaces), the same strings. The PDF prints it in a small monospace font with no wrapped lines. Then run `python -m py_compile woofid_edit.py`. If it reports an indentation or syntax error, the copy is wrong: fix the copy, not the design.
2. Make the folders `raw`, `edits`. The script creates `work` and `out`.
3. Install what is missing: `pip install faster-whisper pillow opencv-python-headless ; ffmpeg and ffprobe` on the PATH (Windows: `winget install Gyan.FFmpeg`; Mac: `brew install ffmpeg`); Node.js 20+ (`winget install OpenJS.NodeJS.LTS` or `brew install node`). The renderer (HyperFrames, pinned to version 0.8.116) and a headless Chrome are fetched by `npm` on the first render.
4. Optional, Windows with an NVIDIA GPU: `pip install nvidia-cublas-cu12 "nvidia-cudnn-cu12==9.*"`. Without it, transcription runs on the CPU automatically.

Step 2 • Transcribe

Copy the creator's video into `raw/` with a simple name (letters, digits, dashes). Run `python woofid_edit.py transcribe raw/NAME.mov`. It prints the full transcript and writes word timings to `work/NAME/words.json`. Read the whole transcript before planning anything.

Step 3 • Choose the start and end

Run `python woofid_edit.py sheet raw/NAME.mov` and look at `work/NAME/start.jpg` (the first 12 seconds, 5 frames per second: frame n , counted from 0 left to right and top to bottom, is at $n/5$ s) and `overview.jpg`.

- **Start:** cut away any dead setup at the front (pressing record, sitting down, looking away). If the creator opens with a move, such as spinning around in a chair toward the camera, start **mid-motion**, as they turn to face the lens: that is the Woofid opening. Never cut into the first word: the start must be at least 0.15 s before the first word's start time in `words.json`. If the video already opens cleanly, use 0.
- **End:** about 0.5 s after the last word ends, and before the creator reaches for the phone. Leave `end` as `null` if the video already ends cleanly.
- Times in the edit file are in seconds of the **original** video for `start / end`; the script shifts the captions for you.

Step 4 • Write the edit file

Create `edits/NAME.json`. Appendix B is a complete real example. The fields:

FIELD	WHAT TO PUT
video	NAME (the folder name under <code>work/</code>)
source	<code>raw/NAME.mov</code>
start , end	From step 3 (seconds, or <code>null</code> for end)
who	The headline box, shown for the whole video, in the form Who - What : the part before " - " shows in orange, the rest in white. For a client story: the client type, then what was done for them (<i>Financial Advisor - Manage their website and get more clients</i>). For other videos: the topic, then the promise (<i>Small Business Owners - Answer every lead in 5 minutes</i>). Keep it under about 65 characters. Describe only what the speaker actually says happened. Show it to the creator and use their wording if they give you one.
fixes	Misheard words in this video, as <code>["misheard lowercase words", "Correct Words"]</code> pairs. Common AI names (Claude, ChatGPT, Gemini, Claude Code...) are already fixed by the script. Check names of products, companies, people and places in the transcript and add a fix for each one whisper got wrong.
drop	Optional <code>[start, end]</code> ranges, in seconds of the edited video, whose captions to remove (a stumbled take the speaker repeats, or a transcription loop). Usually empty.
cues	The graphics, in spoken order (rules below).
thumb	The thumbnail (step 7): <code>{"at": seconds, "headline": "...", "label": "Client story"}</code> . Fill it in at step 7; <code>label</code> is optional (the small line after WOOFID, default <i>Client story</i>).

Graphics: the rules

- **How many:** about one every 15 to 20 seconds, spread across the whole video: 5 to 9 for a 2-minute video. Never two at once (the script delays a graphic that would overlap the previous one).
- **When:** `at` is a phrase of 1 to 4 words copied from `words.json`; the graphic appears when its first word is spoken. Use a distinctive phrase. If it occurs more than once, add `"after_t"` (seconds in the edited video) to skip earlier matches. Cues must be in the order they are spoken.
- **What:** each graphic restates the point being made *right then*, in fewer words than the speaker uses. Never invent numbers, names or results the speaker did not say.
- **Copy:** `kicker` 1 to 3 words (The client, The problem, Step 1, What I found, The result). Titles 6 words or fewer. List items and flow steps 4 words or fewer. Sentence case.
- **dur:** 4.5 to 7 seconds; longer for lists, comparisons and flows.

TYPE	USE IT FOR	FIELDS
card	Who the client is, the problem, the goal, the result, a key idea	<code>kicker, title, icon</code> ; optional <code>sub</code>
stat	One striking number the speaker says	<code>kicker, value, label, icon</code> ; optional <code>count: [from, to]</code> to count up (then <code>value</code> is the final text), <code>prefix, suffix</code>
list	2 or 3 things found or done, ticked off one by one	<code>kicker, items</code>
flow	A process of 3 or 4 steps; the last step is highlighted	<code>kicker, steps: [[icon, label], ...]</code>

vs	Before and after, old way and new way	kicker, before, after (1 or 2 items each); optional before_label (default Old way), after_label (default With AI)
tiers	Comparing 2 to 4 options on one scale	kicker, rows: [[name, percent, note], ...]

Icons: bank, deck, trophy, chart, brain, globe, calendar, receipt, building, shield, camera, search, file, people, rocket, money, clock, phone, mail, code, gavel, spark, plug, dash.

Step 5 • Build and render

```
python woofid_edit.py build edits/NAME.json      # trims the video, writes work/NAME/project/index.html
python woofid_edit.py render edits/NAME.json    # checks, then renders out/NAME-edit.mp4 (a few minutes)
```

If build stops with *anchor not found*, the `at` phrase is not in `words.json` exactly as written (after the fixes are applied): pick words that are there. Rendering a 2-minute video takes 3 to 6 minutes.

Step 6 • Check your work before you hand it over

Run `python woofid_edit.py stills out/NAME-edit.mp4` and look at `out/NAME-edit-check.jpg` (6 frames). Also look at the frame at each graphic's start time if anything seems off (`ffmpeg -ss SECONDS -i out/NAME-edit.mp4 -frames:v 1 frame.png`). Confirm:

- The first frame is not dead setup, and the first word is not clipped.
- Captions are spelled right, including names and brands. Fix with `fixes`, then build and render again.
- The headline box reads correctly and fits on at most two lines.
- Each graphic matches what is being said when it appears.

Step 7 • Make the thumbnail

The thumbnail is what people see before they press play: 1280 × 720, the speaker on the right, framed on their face, and a big headline on the left with the client type from `who` in an orange tag above it.



1. Write the headline. 3 to 6 words, punchier than the headline box, uppercase is applied for you. Put the payoff in `*stars*` so it shows in orange: *Find the *leasing gaps**, *His AI deck *won the bank**, *Need a permit? *Ask AI**, *Edit *100 photos* at once*. Use only what the speaker says in the video: no invented numbers or results. Show it to the creator with the headline box text.

2. **Pick the frame.** Make a contact sheet of the clean cut (no captions or graphics):

```
ffmpeg -i work/NAME/project/assets/talk.mp4 -vf "fps=1/4,scale=216:-2,tile=8x4" -frames:v 1  
work/NAME/thumb-sheet.jpg
```

Tile n (counting from 0, left to right, top to bottom) is about $n \times 4$ seconds. Choose a frame where the speaker faces the camera, eyes open, mouth closed or mid-word with an open expressive gesture; avoid blinks, looking away and hands over the face. Put its time in `thumb.at` (seconds of the edited video).

3. **Render it:** `python woofid_edit.py thumb edits/NAME.json` → `out/NAME-thumb.jpg`. The script finds the face and centres the crop on it.

4. **Look at it.** The face must be fully in the right-hand panel and not cut by the edge; the headline must not run into the face. If it does, pick another frame or shorten the headline, and render again.

Then tell the creator where the video and thumbnail are, what the headline says, and list the graphics with their times, so they can ask for changes. Do not claim you watched the whole video unless you did.

The house style, for reference (already built into the script)

ELEMENT	SPEC
---------	------

Frame	1080 × 1920, 30 fps; slow push-in from 100% to 106% over the whole video; audio high-passed at 70 Hz and normalized to -16 LUFS
Captions	Montserrat 900, 92 px, uppercase, orange #FF7A00 with a 14 px black outline, centered around $y = 1480$ (chest height). 1 to 3 words at a time, breaking at punctuation and pauses; the word being spoken pops to 114% and #FFB000. Filler words (um, uh) are dropped.
Headline box	Top, $y = 96$ px, for the whole video: dark rounded box, 5 px orange border, 40 px text, the part before " - " in orange
Graphics	White cards 940 px wide with a 12 px orange top edge, in the zone from $y = 290$ to 910 px, dropping in with a slight bounce and lifting out at the end

Appendix B • A complete edit file (real example)

The edit behind the frames on page 1. The speaker tells a story about a property management client.

```
{
  "video": "IMG_6161",
  "source": "raw/IMG_6161.mov",
  "start": 0.0,
  "end": 151.6,
  "who": "Property Management Company - Find the gaps in their leasing process",
  "fixes": [],
  "drop": [],
  "cues": [
    {
      "at": "discovery call",
      "type": "card",
      "icon": "people",
      "kicker": "Step 1",
      "title": "Discovery call with leadership",
      "sub": "30 to 60 minutes of listening",
      "dur": 5
    },
    {
      "at": "more of a diagnostics",
      "type": "vs",
      "kicker": "How it works",
      "before_label": "Not a",
      "before": [
        "Sales consultation"
      ],
      "after_label": "It's a",
      "after": [
        "Diagnostic",
        "Listen, then advise"
      ],
      "dur": 5.5
    },
    {
      "at": "several hundred million",
      "type": "card",
      "icon": "building",
      "kicker": "The client",
      "title": "Big property management company",
      "sub": "Several hundred million in AUM",
      "dur": 5
    },
    {
      "at": "broken leasing process",
      "type": "card",
      "icon": "shield",
      "kicker": "The problem",
      "title": "A broken leasing process",
      "dur": 4.5
    },
    {
      "at": "it's paper",
      "type": "list",
      "kicker": "What I found",
      "items": [
        "Paper applications",
```

```

    "No CRM, no lead tracking",
    "Info stuck in a file"
  ],
  "dur": 7
},
{
  "at": "granular details",
  "type": "card",
  "icon": "search",
  "kicker": "Where the gaps hide",
  "title": "Talk to the boots on the ground",
  "dur": 5
},
{
  "at": "over six weeks",
  "type": "stat",
  "icon": "clock",
  "kicker": "Call to move-in",
  "value": "6+ WEEKS",
  "label": "from first call to move-in",
  "dur": 5
},
{
  "at": "no way to do",
  "type": "vs",
  "kicker": "Lead follow-up",
  "before_label": "Today",
  "before": [
    "No lead tracking",
    "No follow-up"
  ],
  "after": [
    "Every lead captured",
    "Drip campaigns"
  ],
  "dur": 6.5
},
{
  "at": "start with senior",
  "type": "flow",
  "kicker": "My process",
  "steps": [
    [
      "people",
      "Senior leadership"
    ],
    [
      "search",
      "Find the gaps"
    ],
    [
      "spark",
      "Build AI systems"
    ]
  ],
  "dur": 7
},
"thumb": {
  "at": 72.64,
  "headline": "Find the *leasing gaps*"
}
}

```

Appendix A • woofid_edit.py (copy verbatim)

"""Woofid talking-head edit kit: orange word-pop captions, a top headline box, and topic graphics.

One script, one JSON edit file per video. Commands (run from the kit folder):

```
python woofid_edit.py transcribe raw/clip.mov      -> work/clip/words.json (word timings, source time)
python woofid_edit.py sheet raw/clip.mov          -> work/clip/start.jpg + overview.jpg (pick the start)
python woofid_edit.py build edits/clip.json       -> trims the video, writes work/clip/project/index.html
python woofid_edit.py render edits/clip.json      -> out/clip-edit.mp4
python woofid_edit.py stills out/clip-edit.mp4    -> out/clip-edit-check.jpg (6 frames to review)
python woofid_edit.py thumb edits/clip.json       -> out/clip-thumb.jpg (1280x720 thumbnail)
```

Needs: Python 3.10+, ffmpeg + ffprobe on PATH, Node 20+ (for npy),
pip install faster-whisper pillow opencv-python-headless

```
"""
import html
import json
import os
import re
import shutil
import subprocess
import sys
from pathlib import Path
```

```
HF_VERSION = "0.8.116" # pinned so every creator's render looks the same
KIT = Path(__file__).resolve().parent
WORK, OUT = KIT / "work", KIT / "out"
```

```
# Spelling fixes applied to every video: (misheard lowercase words, correct words).
FIXES = [{"chad", "ChatGPT"}, ("chat gpt", "ChatGPT"), ("claude code", "Claude Code"), ("cloud code", "Claude Code"),
        ("ai", "AI"), ("i", "I"), ("i'm", "I'm"), ("i'll", "I'll"), ("i've", "I've"), ("i'd", "I'd"),
        ("claude", "Claude"), ("gemini", "Gemini"), ("openai", "OpenAI"), ("powerpoint", "PowerPoint"),
        ("powerpoints", "PowerPoints"), ("excel", "Excel"), ("salesforce", "Salesforce"),
        ("sales force", "Salesforce"), ("quickbooks", "QuickBooks"), ("crm", "CRM"), ("aum", "AUM")]
FILLERS = {"um", "uh", "umm", "uhh", "hmm"}
```

```
def run(cmd, **kw):
    return subprocess.run(cmd, check=True, **kw)
```

```
def duration(p):
    out = run(["ffprobe", "-v", "error", "-show_entries", "format=duration", "-of", "csv=p=0", str(p)],
              capture_output=True, text=True).stdout
    return float(out.strip())
```

```
def esc(s):
    return html.escape(str(s), quote=True)
```

```
def bare(t):
    return re.sub(r"^\w'", "", t).lower()
```

```
# ----- transcribe
```

```
def transcribe(src):
    src = Path(src)
    d = WORK / src.stem
    d.mkdir(parents=True, exist_ok=True)
    if os.name == "nt": # Windows: let CUDA find the pip-installed cuBLAS/cuDNN DLLs, if present
        import importlib
        for lib in ("cublas", "cudnn"):
            try:
                b = os.path.join(list(importlib.import_module("nvidia." + lib).__path__)[0], "bin")
                os.add_dll_directory(b)
                os.environ["PATH"] = b + os.pathsep + os.environ["PATH"]
            except Exception:
                pass
    from faster_whisper import WhisperModel
    try:
        model = WhisperModel("large-v3", device="cuda", compute_type="float16")
    except Exception:
        print("No usable NVIDIA GPU: transcribing on the CPU (slower, same result).")
        model = WhisperModel("large-v3", device="cpu", compute_type="int8")
    # vad_filter + condition_on_previous_text=False stop whisper from looping on repeated phrases
    segs, _ = model.transcribe(str(src), word_timestamps=True, language="en", vad_filter=True,
                               condition_on_previous_text=False, beam_size=5)

    words, text = [], []
    for s in segs:
        text.append(s.text.strip())
        for w in s.words:
            words.append({"w": w.word.strip(), "s": round(w.start, 3), "e": round(w.end, 3)})
    (d / "words.json").write_text(json.dumps({"text": " ".join(text), "words": words}, indent=0), encoding="utf-8")
    print(f"{d / 'words.json'}: {len(words)} words. First word at {words[0]['s']}, last ends at {words[-1]['e']}s")
    print(" ".join(text))
```

```

# ----- contact sheets
def sheet(src):
    src = Path(src)
    d = WORK / src.stem
    d.mkdir(parents=True, exist_ok=True)
    # first 12 s at 5 frames per second: frame n (left to right, top to bottom, from 0) is at n / 5 seconds
    run(["ffmpeg", "-v", "error", "-y", "-t", "12", "-i", str(src), "-vf", "fps=5,scale=-2:200,tile=10x6",
        "-frames:v", "1", str(d / "start.jpg")])
    every = max(1, duration(src) / 20)
    run(["ffmpeg", "-v", "error", "-y", "-i", str(src), "-vf", f"fps=1/{every:.3f},scale=-2:240,tile=5x4",
        "-frames:v", "1", str(d / "overview.jpg")])
    print(f"{d / 'start.jpg'} (frame n = n/5 s)\n{d / 'overview.jpg'} (one frame every {every:.1f} s)")

# ----- words -> captions
def load_words(spec, start, total):
    raw = json.loads(WORK / spec["video"] / "words.json").read_text(encoding="utf-8"))["words"]
    out = [{"w": w["w"].strip(), "s": round(w["s"] - start, 3), "e": round(w["e"] - start, 3)}
           for w in raw if w["w"].strip()]
    merged = [] # whisper splits "10,000" into "10," + "000"
    for w in out:
        if merged and re.search(r'\d,?$', merged[-1]["w"]) and re.match(r"^\d{3}$", w["w"]):
            merged[-1] = {"w": merged[-1]["w"].rstrip(",") + "," + w["w"].lstrip(","), "s": merged[-1]["s"], "e": w["e"]}
        else:
            merged.append(w)
    out = [w for w in merged if w["e"] > 0.05 and w["s"] < total]
    for w in out:
        w["s"] = max(0.0, w["s"])
    for a, b in spec.get("drop", []): # remove a repeated take / transcription loop: [start, end] in edited time
        out = [w for w in out if not (a <= w["s"] < b)]
    for old, new in FIXES + [tuple(f) for f in spec.get("fixes", [])]:
        o, n = old.split(), new.split()
        i = 0
        while i <= len(out) - len(o):
            if [bare(x["w"]) for x in out[i:i + len(o)]] == o:
                tail = re.sub(r"^\w[-.]+", "", out[i + len(o) - 1]["w"]) # keep trailing punctuation
                s, e = out[i]["s"], out[i + len(o) - 1]["e"]
                step = (e - s) / len(n)
                repl = [{"w": t, "s": round(s + k * step, 3), "e": round(s + (k + 1) * step, 3)} for k, t in enumerate(n)]
                repl[-1]["w"] += tail
                out[i:i + len(o)] = repl
                i += len(n)
            else:
                i += 1
    return out

def chunks_of(words):
    """Caption chunks: at most 3 words / 16 letters, break at punctuation or a pause over 0.4 s."""
    words = [w for w in words if bare(w["w"]) not in FILLERS]
    chunks, cur = [], []
    for i, w in enumerate(words):
        cur.append(w)
        nxt = words[i + 1] if i + 1 < len(words) else None
        if (len(cur) == 3 or re.search(r"[.,?!;:]+$", w["w"]) or nxt is None or nxt["s"] - w["e"] > 0.4
            or sum(len(x["w"]) for x in cur) > 16):
            chunks.append(cur)
            cur = []
    return chunks

def find(words, phrase, after=0.0):
    p = [bare(x) for x in phrase.split()]
    b = [bare(w["w"]) for w in words]
    for i in range(len(b) - len(p) + 1):
        if b[i:i + len(p)] == p and words[i]["s"] >= after:
            return words[i]["s"]
    sys.exit(f"anchor not found: {phrase!r} (after {after}s). Use words exactly as they appear in words.json.")

# ----- icons (stroke SVGs on a 24-unit grid)
ICONS = {
    "bank": '<path d="M3 10h18L12 4z"/><path d="M5 10v8M9 10v8M15 10v8M19 10v8M23 10h18"/>',
    "deck": '<rect x="3" y="4" width="18" height="12" rx="1.5"/><path d="M12 16v4M8 20h8M7 12l3-3 2 2 4-4"/>',
    "trophy": '<path d="M8 4h8v5a4 4 0 0 1-8 0z"/>'
    '<path d="M8 6h4a3 3 0 0 4 4M16 6h4a3 3 0 0 1-4 4M12 13v4M8 20h8M10 17h4"/>',
    "chart": '<path d="M4 20v4M4 20h16"/><path d="M8 16v-5M12 16v8M16 16v-8"/>',
    "brain": '<path d="M9 4a3 3 0 0 0-3 3 3 0 0-2 5 3 3 0 0 2 5 3 3 0 0 6 1v5a2 2 0 0 0-3-1z'
    'M15 4a3 3 0 0 1 3 3 3 0 0 1 2 5 3 3 0 0 1-2 5 3 3 0 0 1-6 1"/>',
    "globe": '<circle cx="12" cy="12" r="9"/><path d="M3 12h18M12 3a14 14 0 0 1 0 18M12 3a14 14 0 0 0 18"/>',
    "calendar": '<rect x="3" y="5" width="18" height="16" rx="2"/><path d="M3 10h18M8 3v4M16 3v4M8 14h3"/>',
    "receipt": '<path d="M6 3h12v18l3-2-3 2-3-2-3 2z"/><path d="M9 8h6M9 12h6M9 16h3"/>',
    "building": '<rect x="5" y="3" width="14" height="18" rx="1"/>'
    '<path d="M9 7h2M13 7h2M9 11h2M13 11h2M9 15h2M13 15h2M11 21v-3h2v3"/>',
    "shield": '<path d="M12 3l8 3v6c0 5-4 8-9-4-1-8-4-8-9v6z"/><path d="M9 12l2 2 4-4"/>',
    "camera": '<rect x="3" y="7" width="18" height="13" rx="2"/><circle cx="12" cy="13" r="4"/><path d="M8 7l2-3h4l2 3"/>',
    "search": '<circle cx="11" cy="11" r="7"/><path d="M16 16l5 5"/>',
    "file": '<path d="M6 3h8l4 4v14h6z"/><path d="M14 3v4h4M9 12h6M9 16h6"/>',
    "people": '<circle cx="9" cy="8" r="3"/><circle cx="17" cy="9" r="2.5"/><path d="M3 20a6 6 0 0 1 12 0M14 20a4.5 4.5 0 0 1 7 0"/>',

```

```

"rocket": '<path d="M12 3c4 2 6 6 5 11l-3 3h-4l-3-3c-1-5 1-9 5-11z"/><circle cx="12" cy="10" r="2"/>'
'<path d="M7 14l-3 3 3 1M17 14l3 3-3 1M10 20l2 2-2"/>',
"money": '<rect x="2" y="6" width="20" height="12" rx="2"/><circle cx="12" cy="12" r="3"/><path d="M6 9v.01M18 15v.01"/>',
"clock": '<circle cx="12" cy="12" r="9"/><path d="M12 7v5l3 3"/>',
"phone": '<path d="M5 3h4l2 5-3 2a11 11 0 0 0 6 6l2-3 5 2v4a2 2 0 0 1-2 2A17 17 0 0 1 3 5a2 2 0 0 1 2-2z"/>',
"mail": '<rect x="3" y="5" width="18" height="14" rx="2"/><path d="M3 7l9 6 9-6"/>',
"code": '<path d="M8 8l-5 4 5 4M16 8l5 4-5 4M14 5l-4 14"/>',
"gavel": '<path d="M13 4l7 7M10 7l7 7M11.5 5.5l-4 4 7 7 4-4M9 12l-6 2 2 6-6M3 21h9"/>',
"spark": '<path d="M12 3v4M12 17v4M3 12h4M6 6l2.5 2.5M15.5 15.5L18 18M6 18l2.5-2.5M15.5 8.5L18 6"/>',
"plug": '<path d="M9 3v5M15 3v5M6 8h12v3a6 6 0 0 1-12 0zM12 17v4"/>',
"dash": '<rect x="3" y="4" width="18" height="16" rx="2"/><path d="M3 9h18M8 9v11M11 13h7M11 16h4"/>',
}

def icon(name, size=88):
    return (f'<svg class="ico" width="{size}" height="{size}" viewBox="0 0 24 24" fill="none" stroke="currentColor" '
            f'stroke-width="1.9" stroke-linecap="round" stroke-linejoin="round">{ICONS[name]}</svg>')

# ----- graphics
def graphic(g, gid, start, dur):
    """Return (html, js) for one graphic card in the top zone. Times are in seconds of the edited video."""
    t, js, inner = g["type"], [], ""
    end = round(start + dur, 2)
    T = "t1.fromTo"
    if t == "card":
        ic = f'<div class="ibox">{icon(g["icon"])}</div>' if g.get("icon") else ""
        sub = f'<div class="sub">{esc(g["sub"])}</div>' if g.get("sub") else ""
        inner = (f'<div class="card row">{ic}<div><div class="kick">{esc(g["kicker"])}</div>'
                f'<div class="title">{esc(g["title"])}</div><div><div class="sub">{esc(g["sub"])}</div></div>')
        js.append(f'{T}("#{gid} .ibox",{{scale:0,rotation:-20}},{{scale:1,rotation:0,duration:0.45,'
                f'ease:"back.out(2.2)"}},{start + 0.15}};')
        js.append(f'{T}("#{gid} .title",{{opacity:0,y:20}},{{opacity:1,y:0,duration:0.35,ease:"power3.out"}},{start + 0.25}};')
        if sub:
            js.append(f'{T}("#{gid} .sub",{{opacity:0}},{{opacity:1,duration:0.3}},{{start + 0.5}};')
    elif t == "stat":
        ic = f'<div class="stat-ico">{icon(g["icon"], 70)}</div>' if g.get("icon") else ""
        inner = (f'<div class="card stat">{ic}<div class="kick">{esc(g["kicker"])}</div>'
                f'<div class="big" id="{gid}-v">{esc(g["value"])}</div><div class="sub">{esc(g["label"])}</div></div>')
        if "count" in g:
            a, b = g["count"]
            pre, suf = g.get("prefix", ""), g.get("suffix", "")
            js.append(f'(()=>{{const o={{v:{a}};T(o,{{v:{a}}},{{v:{b},duration:1.1,ease:"power2.out",'
                    f'onUpdate:()=>{{document.getElementById("#{gid}-v").textContent='
                    f'"{pre}" + Math.round(o.v).toLocaleString("en-US") + "{suf}";}}},{{start + 0.2}};}});')
            js.append(f'{T}("#{gid}-v",{{scale:0.6,opacity:0}},{{scale:1,opacity:1,duration:0.5,ease:"back.out(2.4)"}},{start + 0.15}};')
            js.append(f'{T}("#{gid}-v",{{scale:1}},{{scale:1.07,duration:0.18,yoyo:true,repeat:1,ease:"sine.inOut"}},{start + 1.35}};')
    elif t == "flow":
        steps = g["steps"]
        chips = "".join((f'<div class="arrow" id="{gid}-a{i}">&#10140;</div>' if i else "") +
                        f'<div class="chip" id="{gid}-s{i}">{icon(s[0], 60)}<span>{esc(s[1])}</span></div>'
                        for i, s in enumerate(steps))
        inner = f'<div class="card"><div class="kick">{esc(g["kicker"])}</div><div class="flow">{chips}</div></div>'
        gap = min(0.55, (dur - 1.2) / max(1, len(steps)))
        for i in range(len(steps)):
            at = round(start + 0.25 + i * gap, 2)
            if i:
                js.append(f'{T}("#{gid}-a{i}",{{opacity:0,x:-14}},{{opacity:1,x:0,duration:0.2}},{{round(at - 0.1, 2)}};')
            js.append(f'{T}("#{gid}-s{i}",{{opacity:0,scale:0.5}},{{opacity:1,scale:1,duration:0.35,'
                    f'ease:"back.out(2.2)"}},{at}};')
    elif t == "vs":
        def col(cls, label, items, mark):
            rows = "".join(f'<div class="vrow"><b>{mark}</b>{esc(x)}</div>' for x in items)
            return f'<div class="vcol {cls}"><div class="vlab">{esc(label)}</div><div>{rows}</div>'
        inner = (f'<div class="card"><div class="kick">{esc(g["kicker"])}</div><div class="vs">'
                f'<col("old", g.get("before_label", "Old way"), g["before"], "&#10005;")>'
                f'<col("new", g.get("after_label", "With AI"), g["after"], "&#10003;")></div></div>')
        js.append(f'{T}("#{gid} .old",{{opacity:0,x:-40}},{{opacity:1,x:0,duration:0.35,ease:"power3.out"}},{start + 0.2}};')
        js.append(f'{T}("#{gid} .new",{{opacity:0,x:40}},{{opacity:1,x:0,duration:0.35,ease:"power3.out"}},{start + 0.9}};')
        js.append(f't1.to("#{gid} .old",{{opacity:0.45,duration:0.3}},{{start + 1.3}};')
        js.append(f'{T}("#{gid} .new .vrow",{{opacity:0,y:10}},{{opacity:1,y:0,duration:0.25,stagger:0.18}},{{start + 1.05}};')
    elif t == "list":
        rows = "".join(f'<div class="lrow" id="{gid}-r{i}"><div class="tk" id="{gid}-t{i}">&#10003;</div>'
                        f'<span>{esc(x)}</span></div>' for i, x in enumerate(g["items"]))
        inner = f'<div class="card"><div class="kick">{esc(g["kicker"])}</div><div>{rows}</div>'
        gap = min(0.6, (dur - 1.0) / max(1, len(g["items"])))
        for i in range(len(g["items"])):
            at = round(start + 0.25 + i * gap, 2)
            js.append(f'{T}("#{gid}-r{i}",{{opacity:0,x:-30}},{{opacity:1,x:0,duration:0.28,ease:"power2.out"}},{at}};')
            js.append(f'{T}("#{gid}-t{i}",{{scale:0}},{{scale:1,duration:0.25,ease:"back.out(3)"}},{round(at + 0.2, 2)}};')
    elif t == "tiers":
        rows = "".join(f'<div class="trow"><div class="tname">{esc(n)}</div><div class="track"><div class="fill" '
                        f'id="{gid}-f{i}" style="width:{p}%></div></div><div class="tnote">{esc(note)}</div></div>'
                        for i, (n, p, note) in enumerate(g["rows"]))
        inner = f'<div class="card"><div class="kick">{esc(g["kicker"])}</div><div>{rows}</div>'
        for i in range(len(g["rows"])):
            js.append(f'{T}("#{gid}-f{i}",{{scaleX:0}},{{scaleX:1,duration:0.6,ease:"power3.out"}},{
                    f'round(start + 0.3 + i * 0.3, 2)}};')
    else:
        sys.exit(f"unknown graphic type {t!r}: use card, stat, flow, vs, list or tiers")

```

```

js.insert(0, f'{"#{gid} .card",{{opacity:0,y:-60,scale:0.92}},{{opacity:1,y:0,scale:1,duration:0.45,'
f'ease:"back.out(1.7)"}}},{start}};')
js.append(f'tl.to("#{gid} .card",{{opacity:0,y:-40,scale:0.96,duration:0.3,ease:"power2.in"}},{round(end - 0.3, 2)});')
return (f'<div class="gfx clip" id="{gid}" data-start="{start}" data-duration="{round(dur, 2)}" '
f' data-track-index="2">{inner}</div>'), js

```

```

CSS = ""
*{margin:0;padding:0;box-sizing:border-box}
html,body{margin:0;width:1080px;height:1920px;overflow:hidden;background:#000}
#root{position:relative;width:100%;height:100%;overflow:hidden;font-family:'Montserrat',system-ui,sans-serif;background:#000}
#cam{position:absolute;inset:0;transform-origin:50% 45%}
#talk{position:absolute;inset:0;width:100%;height:100%;object-fit:cover}
.capwrap{position:absolute;left:60px;right:60px;top:1360px;height:240px;display:flex;align-items:center;
justify-content:center}
.cap{position:absolute;max-width:960px;text-align:center;font-size:92px;font-weight:900;line-height:1.05;
text-transform:uppercase;letter-spacing:-.01em;color:#FF7A00;-webkit-text-stroke:14px #000;paint-order:stroke fill;
text-shadow:0 8px 22px rgba(0,0,0,.6);opacity:0}
.kw{display:inline-block;margin:0 .16em}
.gfx{position:absolute;left:0;right:0;top:290px;height:620px;display:flex;align-items:flex-start;justify-content:center}
.card{width:940px;background:#fffd8f;color:#141413;border-radius:36px;padding:38px 44px;
box-shadow:0 22px 60px rgba(0,0,0,.45);border-top:12px solid #FF7A00}
.card.row{display:flex;align-items:center;gap:34px}
.ibox{flex:none;width:150px;height:150px;border-radius:32px;background:#FF7A00;color:#fff;display:flex;
align-items:center;justify-content:center}
.kick{font-size:30px;font-weight:800;letter-spacing:.12em;text-transform:uppercase;color:#C2410C;margin-bottom:10px}
.title{font-size:60px;font-weight:900;line-height:1.05;letter-spacing:-.02em}
.sub{font-size:34px;font-weight:600;color:#5c5f66;margin-top:12px;line-height:1.2}
.stat{text-align:center;position:relative}
.stat-ico{position:absolute;top:30px;right:36px;color:#FF7A00;opacity:.9}
.big{font-size:132px;font-weight:900;letter-spacing:-.04em;line-height:1;color:#141413;margin-top:6px}
.flow{display:flex;align-items:center;justify-content:center;gap:10px;margin-top:16px}
.chip{display:flex;flex-direction:column;align-items:center;gap:12px;width:200px;text-align:center;font-size:28px;
font-weight:800;line-height:1.1;color:#141413}
.chip .ico{color:#fff;background:#141413;border-radius:24px;padding:18px;width:110px;height:110px}
.chip:last-child .ico{background:#FF7A00}
.arrow{font-size:52px;color:#FF7A00;font-weight:900}
.vs{display:flex;gap:24px;margin-top:10px}
.vcol{flex:1;border-radius:24px;padding:22px 26px}
.vcol.old{background:#e6be6;color:#141413}
.vcol.new{background:#FF7A00;color:#141413}
.vlab{font-size:28px;font-weight:900;letter-spacing:.1em;text-transform:uppercase;margin-bottom:12px}
.vrow{font-size:34px;font-weight:700;line-height:1.2;margin:10px 0;display:flex;gap:14px}
.vrow b{flex:none}
.old.vrow b{color:#c0392b}
.lrow{display:flex;align-items:center;gap:22px;font-size:42px;font-weight:800;line-height:1.15;margin:16px 0}
.tk{flex:none;width:58px;height:58px;border-radius:16px;background:#FF7A00;color:#fff;display:flex;align-items:center;
justify-content:center;font-size:38px;font-weight:900}
.trow{display:flex;align-items:center;gap:18px;margin:18px 0}
.tname{width:190px;font-size:38px;font-weight:900}
.track{flex:1;height:44px;background:#e6be6;border-radius:22px;overflow:hidden}
.fill{height:100%;background:#FF7A00;border-radius:22px;transform-origin:0 50%}
.tnote{width:250px;font-size:28px;font-weight:700;color:#5c5f66;line-height:1.1}
#whobox{position:absolute;left:0;right:0;top:96px;display:flex;justify-content:center}
.who{max-width:980px;text-align:center;background:rgba(20,20,19,.92);color:#fff;border:5px solid #FF7A00;
border-radius:30px;padding:16px 34px;box-shadow:0 10px 30px rgba(0,0,0,.45);font-size:40px;font-weight:800;
line-height:1.2}
.who .wname{color:#FF7A00;font-weight:900}
""

```

```

def who_html(text):
    name, _, rest = text.partition(" - ")
    return f'<span class="wname">{esc(name)}</span> - {esc(rest)}' if rest else esc(text)

```

```

# ----- build
def build(edit_path):
    spec = json.loads(Path(edit_path).read_text(encoding="utf-8"))
    vid, src = spec["video"], KIT / spec["source"]
    start = float(spec.get("start", 0))
    d = WORK / vid
    proj = d / "project"
    (proj / "assets").mkdir(parents=True, exist_ok=True)
    talk = proj / "assets" / "talk.mp4"
    length = (float(spec["end"]) - start) if spec.get("end") else duration(src) - start
    # trim + upright 1080x1920 at 30 fps + loudness -16 LUFs (phone rotation is applied automatically)
    run(["ffmpeg", "-v", "error", "-y", "-ss", f"{start:.3f}", "-i", str(src), "-t", f"{length:.3f}",
        "-vf", "scale=1080:1920:force_original_aspect_ratio=increase,crop=1080:1920,fps=30,setsar=1",
        "-c:v", "libx264", "-crf", "17", "-preset", "medium", "-pix_fmt", "yuv420p",
        "-af", "highpass=f=70,loudnorm=I=-16:TP=-1.5:LRA=11", "-c:a", "aac", "-b:a", "192k", str(talk)])
    total = round(duration(talk) - 0.05, 2)
    words = load_words(spec, start, total)

    js, caps = [], []
    js.append(f'tl.fromTo("#cam",{{scale:1.0}},{{scale:1.06,duration:{total},ease:"none"}},0);') # slow push-in
    chunks = chunks_of(words)
    for i, ch in enumerate(chunks):
        s = ch[0]["s"]
        e = min(chunks[i + 1][0]["s"] if i + 1 < len(chunks) else total, ch[-1]["e"] + 0.6, total)

```

```

        spans = " ".join(f'<span class="kw" id="c{i}w{j}">{esc(w["w"].strip(".",?;:"))}</span>' for j, w in enumerate(ch))
        caps.append(f'<div class="cap" id="c{i}">{spans}</div>')
        js.append(f'tl.fromTo("#c{i}",{{opacity:0,scale:0.8}},{{opacity:1,scale:1,duration:0.1,ease:"back.out(2.5)"}},{s});')
        for j, w in enumerate(ch): # the spoken word pops brighter
            js.append(f'tl.fromTo("#c{i}w{j}",{{scale:1,color:"#FF7A00"}},{scale:1.14,color:"#FFB000",duration:0.07}},'
                    f'{max(w["s"], s)});')
            js.append(f'tl.to("#c{i}w{j}",{{scale:1,color:"#FF7A00",duration:0.1}},{round(max(w["e"], w["s"] + 0.12), 3)});')
        js.append(f'tl.set("#c{i}",{{opacity:0}},{round(e, 3)});')

js.append('tl.fromTo("#who",{opacity:0,y:-30},{opacity:1,y:0,duration:0.45,ease:"back.out(2)"}{0.05});')
gfx, last_end = [], 0.0
for k, g in enumerate(spec["cues"]):
    st = round(find(words, g["at"], g.get("after_t", 0.0)) - 0.1, 2)
    st = max(st, last_end + 0.1, 0.0) # never overlap the previous graphic
    dd = min(g.get("dur", 4.0), total - st)
    h, j = graphic(g, f"g{k}", st, round(dd, 2))
    gfx.append(h)
    js += j
    last_end = st + dd

doc = f"""<!doctype html>
<html lang="en" data-resolution="portrait">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=1080, height=1920" />
<link rel="preconnect" href="https://fonts.googleapis.com" />
<link href="https://fonts.googleapis.com/css2?family=Montserrat:wght@600;700;800;900&display=block" rel="stylesheet" />
<script src="https://cdn.jsdelivr.net/npm/gsap@3.14.2/dist/gsap.min.js"></script>
<style>[CSS]</style>
</head>
<body>
<div id="root" data-composition-id="main" data-start="0" data-duration="{total}" data-width="1080" data-height="1920">
<div id="cam"><video id="talk" src="assets/talk.mp4" data-start="0" data-duration="{total}" data-track-index="0"
data-has-audio="true" data-volume="1"></video></div>
<div class="capwrap clip" id="caps" data-start="0" data-duration="{total}" data-track-index="1">{''.join(caps)}</div>
{''.join(gfx)}
<div class="clip" id="whobox" data-start="0" data-duration="{total}" data-track-index="3">
<div class="who" id="who">{who_html(spec["who"])}</div></div>
</div>
<script>
const tl = gsap.timeline({{ paused: true }});
{chr(10)}.join(js)
window.__timelines["main"] = tl;
tl.seek(0);
</script>
</body>
</html>
"""

(proj / "index.html").write_text(doc, encoding="utf-8")
scripts = {k: f"npx --yes hyperframes@{HF_VERSION} {'preview' if k == 'dev' else k}" for k in ("dev", "check", "render")}
(proj / "package.json").write_text(json.dumps({"name": vid.lower(), "private": True, "type": "module",
"scripts": scripts}, indent=2))
(proj / "meta.json").write_text(json.dumps({"id": vid.lower(), "name": vid}, indent=2))
print(f"{proj / 'index.html': {total}s, {len(chunks)} caption chunks, {len(spec['cues'])} graphics")

def npx(args, cwd):
    env = {**os.environ, "HYPERFRAMES_NO_TELEMETRY": "1", "HYPERFRAMES_SKIP_SKILLS": "1"}
    run([shutil.which("npx") or "npx", "--yes", f"hyperframes@{HF_VERSION}", *args], cwd=cwd, env=env)

def render(edit_path):
    vid = json.loads(Path(edit_path).read_text(encoding="utf-8"))["video"]
    proj = WORK / vid / "project"
    OUT.mkdir(exist_ok=True)
    npx(["check"], proj)
    npx(["render", "--workers", "4", "--output", str(OUT / f"{vid}-edit.mp4")], proj)
    print(f"done: {OUT / (vid + '-edit.mp4')}")

def stills(video):
    video = Path(video)
    total, tmp = duration(video), []
    for k in range(6):
        p = video.with_name(f"_still{k}.png")
        run(["ffmpeg", "-v", "error", "-y", "-ss", f"{total * (k + 0.5) / 6:.2f}", "-i", str(video),
            "-frames:v", "1", "-vf", "scale=270:-2", str(p)])
        tmp.append(p)
    out = video.with_name(video.stem + "-check.jpg")
    args = sum([["-i", str(p)] for p in tmp], [])
    run(["ffmpeg", "-v", "error", "-y", *args, "-filter_complex", "hstack=inputs=6", str(out)])
    for p in tmp:
        p.unlink()
    print(out)

# ----- thumbnail
FONT_URL = "https://github.com/google/fonts/raw/main/ofl/montserrat/Montserrat%5Bwght%5D.ttf"
ORANGE, INK = (255, 122, 0), (20, 20, 19)

```

```

def font(size, weight):
    from PIL import ImageFont
    f = KIT / "fonts" / "Montserrat.ttf"
    if not f.exists():
        import urllib.request
        f.parent.mkdir(exist_ok=True)
        urllib.request.urlretrieve(FONT_URL, f)
    ft = ImageFont.truetype(str(f), size)
    ft.set_variation_by_axes([weight])
    return ft

def lay_out(head, size, width):
    """Words of an uppercase headline as lines of (word, orange?); *starred words* are orange."""
    words, hot = [], False
    for w in head.upper().split():
        start, end = w.startswith("*"), w.endswith("*")
        hot = hot or start
        words.append((w.strip("*"), hot))
        hot = hot and not end
    f, lines, cur = font(size, 900), [], []
    for w in words:
        if cur and f.getlength(" ".join(x for x, _ in cur + [w])) > width:
            lines.append(cur); cur = []
        cur.append(w)
    return f, lines + [cur]

def spaced(draw, xy, text, f, fill, track):
    """Draw text with letter spacing (track = em fraction); returns its width."""
    x, y = xy
    for ch in text:
        if draw:
            draw.text((x, y), ch, font=f, fill=fill)
            x += f.getlength(ch) + f.size * track
    return x - xy[0] - f.size * track

def thumb(edit_path):
    import cv2
    from PIL import Image, ImageDraw, ImageFilter
    spec = json.loads(Path(edit_path).read_text(encoding="utf-8"))
    vid, th = spec["video"], spec["thumb"]
    frame = WORK / vid / "thumb-frame.png" # from the clean trimmed video: no captions or graphics
    run(["ffmpeg", "-v", "error", "-y", "-ss", str(th["at"]), "-i", str(WORK / vid / "project" / "assets" / "talk.mp4"),
        "-frames:v", "1", str(frame)])
    im = Image.open(frame).convert("RGB")
    gray = cv2.cvtColor(cv2.imread(str(frame)), cv2.COLOR_BGR2GRAY)
    faces = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml").detectMultiScale(
        gray, 1.1, 6, minSize=(120, 120))
    if len(faces):
        x, y, w, h = max(faces, key=lambda f: f[2] * f[3])
    else:
        print("no face found: using the usual seated framing")
        x, y, w, h = im.width * .55 - 130, im.height * .42 - 130, 260, 260
    # Woofid framing: the speaker on the right in a 560 x 720 panel, face about 190 px tall, eyes a third down
    k = max(190 / h, 640 / im.width, 720 / im.height)
    big = im.resize((round(im.width * k), round(im.height * k)), Image.LANCZOS)
    left = round(min(max((x + w / 2) * k - 280, 0), big.width - 560))
    top = round(min(max((y + h / 2) * k - 260, 0), big.height - 720))
    out = im.resize((1280, 2276)).crop((0, 778, 1280, 1498)).filter(ImageFilter.GaussianBlur(30)).point(lambda v: v * 0.4)
    fade = Image.new("L", (110, 1))
    fade.putdata([round(255 * i / 109) for i in range(110)])
    mask = Image.new("L", (560, 720), 255)
    mask.paste(fade.resize((110, 720)), (0, 0))
    out.paste(big.crop((left, top, left + 560, top + 720)), (720, 0), mask)
    # dark shade behind the text: 94% at the left edge, 80% at the middle, clear by two thirds across
    stops = [(0, .94), (640, .8), (845, 0), (1280, 0)]
    row = [next(a0 + (a1 - a0) * (i - x0) / (x1 - x0) for (x0, a0), (x1, a1) in zip(stops, stops[1:])) if i <= x1
        for i in range(1280)]
    shade = Image.new("L", (1280, 1))
    shade.putdata([round(255 * a) for a in row])
    out.paste(Image.new("RGB", (1280, 720), (10, 12, 16)), (0, 0), shade.resize((1280, 720)))
    # text: orange tag (part of "who" before " - "), the big headline, the brand line
    size = 120
    while True:
        hf, lines = lay_out(th["headline"], size, 660)
        if size <= 56 or (len(lines) * size * .98 <= 430 and all(hf.getlength(" ".join(x for x, _ in ln)) <= 660 for ln in lines)):
            break
        size -= 4
    kf, bf = font(28, 900), font(22, 800)
    kick = spec["who"].split(" - ")[0].upper()
    block = 56 + 24 + round(len(lines) * size * .98) + 24 + 22
    y0 = 360 - block // 2
    d = ImageDraw.Draw(out)
    d.rounded_rectangle((60, y0, 60 + 36 + spaced(None, (0, 0), kick, kf, None, .05), y0 + 56), 10, fill=ORANGE)
    spaced(d, (78, y0 + 11), kick, kf, INK, .05)
    hy = y0 + 56 + 24
    glow = Image.new("L", out.size, 0)

```

```

gd = ImageDraw.Draw(glow)
for i, ln in enumerate(lines):
    gd.text((60, hy + 4 + i * size * .98 - size * .1), " ".join(x for x, _ in ln), font=hf, fill=128)
out.paste((0, 0, 0), (0, 0), glow.filter(ImageFilter.GaussianBlur(9)))
for i, ln in enumerate(lines):
    x = 60
    for word, hot in ln:
        d.text((x, hy + i * size * .98 - size * .1), word, font=hf, fill=ORANGE if hot else (255, 255, 255))
        x += hf.getlength(word + " ")
    by = hy + round(len(lines) * size * .98) + 24
    w1 = spaced(d, (60, by), "WOOFID", bf, ORANGE, .14)
    spaced(d, (60 + w1, by), " . " + th.get("label", "Client story").upper(), bf, (201, 201, 201), .14)
    OUT.mkdir(exist_ok=True)
    out.save(OUT / f"{vid}-thumb.jpg", quality=88)
    print(OUT / f"{vid}-thumb.jpg")

if __name__ == "__main__":
    cmds = {"transcribe": transcribe, "sheet": sheet, "build": build, "render": render, "stills": stills, "thumb": thumb}
    if len(sys.argv) != 3 or sys.argv[1] not in cmds:
        sys.exit(__doc__)
    cmds[sys.argv[1]](sys.argv[2])

```