

Building a comic book pipeline with AI

We built a code-driven production pipeline for a hand-inked, twelve-issue comic in one working session with an AI agent. Scripts become lettering, panels become layers, and the same layers become print pages and motion pieces. This article says how it came together, how it works, and how another agent can build the same thing for another artist by interviewing them first. It also says plainly what does not work yet.

What is real and what isn't

Everything here was built and run in one working session of roughly five to six hours, mostly by an AI agent working from the artist's scripts and instructions. **No real panel art has gone through the pipeline yet.** The scripts exist; the drawings do not. That is the most important fact in this article, so it comes first.

The table is the boundary of every claim that follows. When a later section says something works, it means this table says so.

Piece	Status	What we checked	What is missing
Deterministic renderer and a 16-second teaser	Built and verified	Renders 9:16, 1:1 and 16:9 with music. The same frame rendered twice gave the same file hash. Loop seam checked on stills. Audio measured at -14.0 LUFS, true peak -1.1 dBFS	The teaser's own critique log scores its hook 7 out of 10, so it is not ship-ready. Its picture is a code-drawn placeholder. The scores are the agent's judgement, not an outside reviewer's
Beat measuring (librosa)	Built, lightly tested	Measured 89.3 BPM on music we generated at 90 BPM; beat spacing varied by 6.2 ms	Only run on generated music, never on a supplied track. The tracker skipped the first beat and read about 15 ms late

Piece	Status	What we checked	What is missing
Lettering manifest	Built and verified	12 scripts gave 264 story pages, 24 back-matter pages and 1,537 lettering items. Six of the seven lettering types match an independent grep count exactly. Plain balloons (699) were checked another way: a loose pattern found 1,541 cue-like lines against 1,537 items, and the four extras were not lettering	Only as good as the script format it parses
Layout push to a desktop-publishing app (DesignCraft)	Built, partly verified	Run on all 12 issues: 5 exit clean, 7 are flagged for dialogue that overflows its default panel. Pages 1–4 of issue 1 inspected as rendered images	Never opened in the app's own window. Positions are a default reading-order stack, not a composition. Balloon tails are generic wedges. One page needs a hand layout
Art ingest (scan or PSD to layers)	Built, lightly tested	Run on one flat test image	The PSD path, which reads layers by name, has not been run on a real PSD. No real art
Colour lab	Built	Eight mood recipes, a dirty-gradient painter, swatch files. PhotoCraft's import code was read to confirm it accepts .ase and .aco	Judged only on a stand-in ink. We never ran PhotoCraft itself

Piece	Status	What we checked	What is missing
Test font, version 0.1	Built, lightly tested	283 glyphs; three alternates for every letter and digit. fontspector: 65 pass, 11 warn, 0 fail. A shaping test found 0 identical neighbours across 14,027 nearby same-letter pairs. DesignCraft applied the alternates in a test render	The agent wrote the outlines. Capitals are good, lowercase is weak, strokes are thin at print size. No print proof. Not traced from the artist's ink
Hand-lettering checklist	Built, needs review	191 candidates found by text pattern: 31 sound effects, 46 appearances of one recurring mark, 35 signs, 79 handwriting items	Pattern matching, not reading. Some false hits remain
Research report on open fonts	Done, with gaps	Five research strands with citations	The report flags its own unverified claims: design-press trends, several licences, one Google Fonts rule
Pilot-page kit	Built	Generated for one page: brief, print template, composition guide, study sheet, log	Not used yet. Nobody has drawn the pilot page
Voice-over tool	Written, never run	Nothing	Needs an ElevenLabs key we did not use

An art bible of about 9,000 words and three diagrams were written in the same session by separate agents. The bible sorts every choice into DECIDED, WORKING ASSUMPTION or OPEN. The diagrams passed their tool's validation gates. We did not audit the bible line by line for this article, and passing a gate says nothing about whether a picture is good.

What we did not test at all: Windows or Linux, any app other than DesignCraft's command-line tool, and printed output. The section on limits near the end repeats this with more detail.

The idea in one page

The system around the prompt does most of the work. We took that from Ray Fu's guide, [Motion Design Studio with Claude Opus 5.5](#), which argues that a good AI-made motion graphic comes from infrastructure, house rules and a review loop, with the prompt as the small part. We paraphrase it here and do not reproduce it.

What we adopted from that guide

- Every frame is a pure function of time. A page calls `seek(t)`, a headless browser screenshots it hundreds of times, and ffmpeg encodes the frames. Same time in, same pixels out.
- House rules live in a `CLAUDE.md` file the agent reads first, and a linter enforces the mechanical ones.
- Motion uses closed-form damped springs, stacked: when a target changes, add a spring that starts at that moment instead of restarting the old one.
- Beats are measured from the audio with librosa, and visual changes land on the measured beats.
- The agent reviews its own frames as contact sheets, scores them, fixes the three worst problems and repeats.

What we added, because a comic is not a product video

- Panels as layers (background, colour, ink), fed from hand-inked art that the artist scans and paints.
- An ingest step that turns a scan or a layered PSD into those layers.
- A lettering manifest: one JSON file per issue extracted from the scripts, so the script stays the single source of truth for every balloon, caption and chat line, in print and in motion.
- A push of that manifest into a desktop-publishing app to build lettered pages.
- A colour lab, a font workflow, a checklist of everything that must be hand-lettered, a pilot-page kit, and an art bible that records what is decided and what is open.

Why a comic suits this

A comic is a large pile of structured text plus a lot of hand-made images. The tedious parts are bookkeeping and consistency: which line goes in which balloon, whether the viewer count on page 9 matches the script, whether a sign repeats the same way. Code is good at that. The drawing is not bookkeeping, and the system is built so the artist's hand stays the thing it serves.

Division of labour

The artist draws, paints and decides. The agent builds tools, enforces rules, does the repetitive extraction and layout, and says what it did not verify. We label every choice in the project one of three ways: DECIDED by the artist, WORKING ASSUMPTION set by the agent so something could be built, or OPEN. That labelling is the cheapest habit in this article and the one we would keep first.

How it came together

This is the order we built things in. Each step says what we tried, what broke and what we kept. The whole build was one session; the file timestamps run from about midday to early evening.

1. The studio and its house rules

We wrote the rules down before the first scene. A `CLAUDE.md` file holds the one law (every frame a pure function of time), a list of banned looks, a rhythm rule (something new every two to four seconds), the motion rules, a two-typeface limit and a workflow with gates. Then came a renderer of about 100 lines, a runtime library of about 115 lines (springs, seeded noise, beat snapping) and a linter that refuses to render a scene that breaks the mechanical rules.

Fonts had to be vendored into the project. A deterministic render cannot depend on a network. We kept the idea that rules belong in a file the agent reads first and a linter the renderer runs.

2. The first teaser, and four rounds of critique

The test piece was a 16-second looping teaser for issue one, built from the script's own words: chat lines, a viewer counter and a caption that edits itself. After each render the agent looked at contact sheets, scored six axes (hook, readability, motion quality, variety, brand accuracy, sound sync), named the three worst problems and fixed them. The log records what went wrong:

- **Round 0, stills.** In 9:16 the main frame used about a quarter of the screen. The viewer counter overshot to a value the script never prints, because a bouncy spring on a counter overshoots. The title mark did not line up with the opening shape, so the loop seam jumped.
- **Round 1.** The opening sat still for 2.7 seconds, so three contact-sheet cells were identical. A struck-through phrase vanished in 16:9 because the phrase wrapped across two lines. After the in-story stream ended, the chat kept scrolling, which contradicted the script.
- **Round 2.** The poster frame landed past the loop seam, because beat snapping had moved it from 15.27 s to 15.35 s. We also added a small detail the script called for.
- **Round 3.** Scores of 7, 8, 8, 8, 8 and 8. The hook was stuck at 7 because the picture was placeholder art, and the log says so: not ship-ready.

The hook score went 4, 5, 7, 7 across the rounds. We kept the loop of render, look, score and fix, and the habit of writing the failure down.

3. The art problem

We had scripts and no drawings. We could wait, or build against a stand-in. We generated a deliberately crude test ink image by script, stamped with the words TEST INK, and wrote an ingest tool that separates ink from paper into a transparent PNG whose alpha equals ink density, so dry-brush texture survives. A panel loader then draws the layers with a camera push and parallax.

Building against a stand-in forced the real contract into the open: the artist delivers named layers. We kept that contract (`ink` , `color` , `bg`) and the labelled stand-in.

4. The colour lab

The artist paints by hand with dirty gradients, so the lab is a reference and not a substitute. It holds eight mood recipes, each tied to a specific script page, and a painter that makes warped gradients, wet-edge blooms and paper grain under the ink.

The first renders looked like grey smoke, then marble veins, then airbrush. The agent critiqued and re-rendered four times. Weak spots remain and are recorded: one recipe is so dark the silhouettes barely read, the cleanest recipe shows banding, and the texture reads as wash more than brush.

For the swatches we read PhotoCraft's source to find out which formats it imports. It accepts `.aco` and `.ase`, not `.gpl`, so we exported both. Every pass was judged on the stand-in ink.

5. The lettering manifest

The scripts are plain-text Fountain with a few house conventions. A parser turns each into JSON: pages, panels and every letter with its kind and speaker. Chat cues split into handle and message. Viewer counts quoted in panel descriptions are pulled out. The author's margin notes are stripped and listed separately, so they cannot reach a balloon.

Then we checked it against something that was not our parser. A grep count of each lettering type matched exactly for six of the seven types. We then rewired the teaser to fetch its lines by address (page and panel) and to throw when an address no longer resolves. That exposed a mistake of our own: the teaser brief had guessed page numbers for several moments, and the manifest showed they were elsewhere.

6. The push into a desktop-publishing app

First an agent read the documentation and source of four apps in the ArtCraft suite. DesignCraft has no plugin system, but it has a local control port, a headless script mode and a large command set. It has no speech-balloon primitive. Our push builds a balloon from an ellipse text frame plus a wedge-shaped tail path, uses named paragraph styles per narrator, and measures overset text with the app's own inspector. The tool exits 2 on lettering problems and still writes its output so a human can look.

Three things broke on the way, per the build agent's report: tails landed on page 1 because the path command needs an explicit page; short words split mid-word until the minimum width followed the longest word; and chat handles broke mid-handle. We re-ran all 12 issues ourselves for this article: 5 exit clean and 7 are flagged.

7. Fonts

Five research agents surveyed open fonts. The finding that mattered: open comic-lettering fonts almost never ship alternate letterforms, so repeated letters look typed. Of thirteen fonts whose features were inspected, only one, Shantell Sans, had them.

Then we built a test font by script. Hand-authored centerline strokes are expanded with a pressure-tapered brush, cleaned with boolean unions, spaced automatically, and compiled with fontmake. A `calt` lookup cycles each letter through three versions when it repeats within five glyphs. A HarfBuzz test generated nearby same-letter pairs and

checked that none were identical. The build agent reported that its first run flagged 33 identical pairs. All were repeated punctuation, which has no alternates, so the test was wrong and the font was fine. We restricted the test to letters and digits.

DesignCraft applied the alternates in a test render. The artist liked the font and wants it as the main balloon font. We pushed back on four points: strokes are thin at print size, lowercase is weak, one font cannot serve every narrator, and the agent wrote the outlines, so they are not yet the artist's.

8. The hand-lettering checklist

Some lettering lives inside the picture: sound effects, signs, marks, someone's handwriting. A script scans the panel descriptions for those and lists them with page, panel and the exact words. The first pass found 213 candidates, 94 of them "handwriting", which was too many: the pattern matched phrases like "a glass in her hand". Tightening it gave 191 and 79. One sign recurs eleven times, which tells the artist to draw one consistent hand for it.

9. The pilot kit

Every decision so far rested on a stand-in. A pilot kit generates what the artist needs to draw one page end to end: a brief built from the manifest, a print template at 150% of trim size with bleed and safe guides, a composition guide, a study sheet for a recurring mark, and a log for timing and surprises. The pilot page should settle the big format and style questions with a real page instead of a discussion.

10. How the work was run

Most of the work ran as parallel agents that each owned different files: the colour lab, the layout push, the font, five research strands, the art bible, the diagrams and the agent that wrote this article. One research agent was stopped by the artist mid-run and re-run with a narrower question. Every agent was told to report what it had not verified, and we re-ran the checks that mattered ourselves: the manifest counts, the 12-issue layout run and the font tests.

How it works

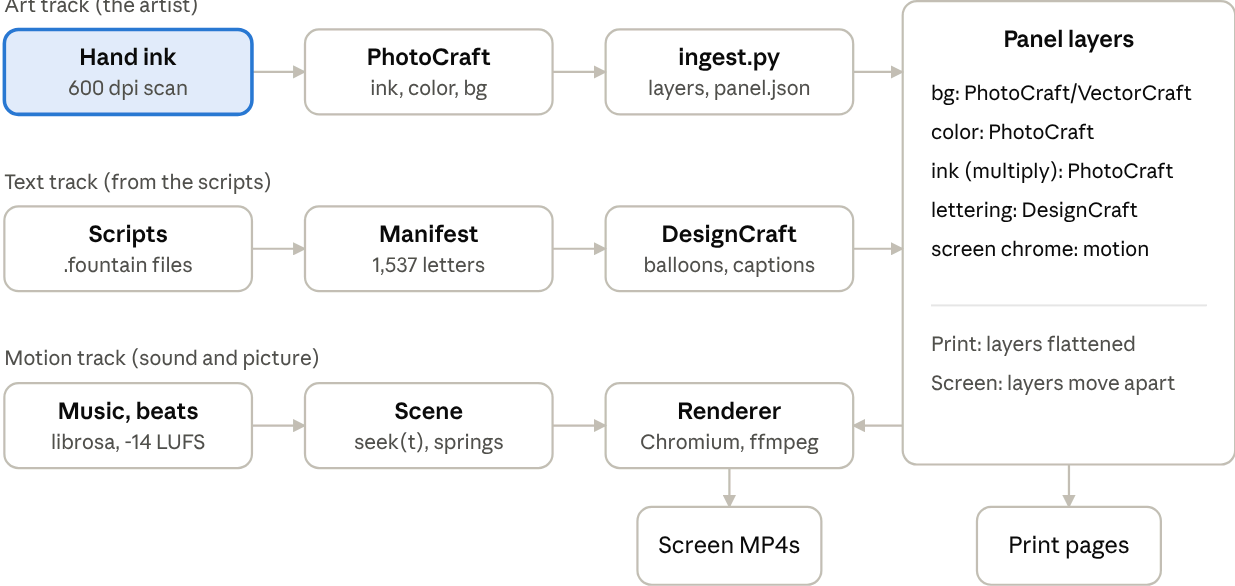
Three tracks run side by side and meet in one layer model. The art track belongs to the artist. The text track starts from the scripts. The motion track renders pictures and sound. They talk to each other only through three files, which is why each track can be built and

tested alone.

Three tracks meet in one layer model that feeds print and screen

Highlighted: the one step only the artist can do

Art track (the artist)



tested on stand-ins only

pipeline · 3 tracks, 1 layer model

The art and text tracks each end by producing layers. The motion track reads them, along with the manifest and the measured beats. Print flattens the stack and screen keeps it apart. The artist's step is the only one no tool can do.

The three contracts

File	Written by	Read by	Holds
panel.json plus layer PNGs	the ingest tool	the motion scenes	Panel size, layer order, a depth per layer, a default camera focus
lettering/issue-NN.json	the manifest builder	the layout push and the motion scenes	Every page, panel, balloon, caption and chat line, with kind and speaker
beats.json	the beat tool	the motion scenes	Measured beat times, downbeats and strong onsets

The layer model

One panel is a stack of layers. Print flattens them. Screen keeps them apart so they can move at different depths.

Layer	What it is	Made in	In print	On screen
<code>bg</code>	Background art	PhotoCraft or VectorCraft	Flattened under everything	Slowest layer, depth 0.25
<code>color</code>	Painted gradients and washes	PhotoCraft	Flattened under the ink	Depth 0.9; coloured light can fall on it
<code>ink</code>	Brush ink, black on transparent, alpha equal to ink density	Scan, cleaned in PhotoCraft	Multiplied on top of the colour	Depth 1.0, moves with the camera
lettering	Balloons and captions set from the manifest	DesignCraft	Live type on a layer above the art	Rebuilt in motion from the same manifest
screen chrome	In-story interfaces: chat rail, LIVE bug, viewer counter	Motion studio; the rail is typeset in DesignCraft for print	A static rail whose density shows volume	Animated, with numbers from the manifest

Keep ink, colour and background on separate layers from the start. If they arrive flattened, nothing can move independently and any coloured light must be painted over the brushwork.

What each track does

- **Art track.** Hand ink, scan at 600 dpi (a working assumption), paint in PhotoCraft on layers named `ink`, `color` and `bg`, export a PSD, then run `tools/ingest.py` to write the layer PNGs and `panel.json`.
- **Text track.** `.fountain` scripts go through `tools/manifest.mjs` to `lettering/issue-NN.json`. `tools/letter-designcraft.mjs` turns that into a lettered `.designcraft` document and page PNGs. The motion scenes read the same JSON through `lib/lettering.js`.

- **Motion track.** Music is generated in code or supplied, `tools/beats.py` measures it, a scene file exports `seek(t)`, and `render.mjs` produces MP4s in three aspect ratios plus a review pack for the agent to look at.

File map

```

studio/
  CLAUDE.md          house rules the agent reads first
  render.mjs         deterministic renderer: seek(t) -> PNG frames ->
ffmpeg
  lib/
    runtime.js       springs, seeded noise, beat snapping, stage(),
scene()
  panels.js          loads ingested panels, draws layers with camera +
parallax
  lettering.js       fetches lines from the manifest by page.panel,
throws on drift
  dirty.js           dirty-gradient painter (colour lab)
  soop.css           brand tokens and shared UI styles (rename for your
project)
  tools/
    manifest.mjs     scripts -> lettering/issue-NN.json
    letter-designcraft.mjs manifest -> lettered DesignCraft pages
    handlettering.mjs manifest -> checklist of in-picture lettering
    ingest.py        scan or PSD -> layers + panel.json
    beats.py, synth.mjs, mix.sh  measure, generate and master audio
    review.mjs, lint.mjs  review pack for the agent; house-rule linter
    pilot-kit.mjs     generate a one-page pilot kit
    standin_ink.py    labelled test ink for exercising the pipeline
lab/
  fonts/            colour lab: recipes, swatches, test passes
  test font: stroke data, brush, build script,
specimen
  lettering/        generated manifests, checklist, layout output
  projects/<piece>/  scene.html, score.mjs, beats.json, brief.md,
critique.md, out/
  critique/RUBRIC.md the six scores and the three-rounds rule
  assets/           fonts and ingested panels
  incoming/         scans waiting to be ingested

```

Principles that mattered

Each principle names the mechanism that enforces it. A principle with no mechanism was advice, and advice did not survive a long session.

1. **One source of truth.** The scripts feed a manifest, and the manifest feeds print and motion. Nothing is retyped. `lib/lettering.js` fetches a line by page and panel and throws if the script no longer has it, so a stale promo fails loudly.
2. **Measure, do not assume.** `beats.py` measured our generated music at 89.3 BPM when we had designed it at 90, and showed the tracker skipped the first beat and read about 15 ms late. We only knew because we printed the numbers. Spacing in the font comes from measured ink profiles, and the font's alternates were proved with a shaping test.
3. **Enforce rules with a linter, not prose.** The linter blocks `Math.random`, timers, `requestAnimationFrame`, wall-clock time, CSS transitions and animations, glow on interface text, and a centred title on a gradient. The renderer will not run a scene that fails it. A `lint-ok` comment marks a reviewed exception.
4. **Critique with the actual pixels.** After every render the agent opens a contact sheet, a strip of the first three seconds, a phone-width sheet and a seam still, then scores six axes and fixes the three worst problems. Three rounds minimum, every score 8 or higher to ship. The agent grades its own work, so a human should still look.
5. **Label stand-ins loudly.** The test ink says TEST INK in its own pixels and `STANDIN` in its filename. The font's name table says ROUGH TEST / PLACEHOLDER. A placeholder that looks finished is how a placeholder ships.
6. **Verify counts independently.** The manifest's per-type counts were compared with a plain grep. The layout push was re-run on every issue and judged by exit code, not by the agent's summary.
7. **Fail loudly on drift.** Missing panel art renders a visible message, not a blank. The layout push exits 2 when text will not fit and still writes its output so you can see why. The renderer's file server once crashed on a missing file because it sent a success header before reading the file. We fixed that and kept the lesson: read first, then answer.
8. **Parallel agents, disjoint files.** Each agent owned a folder and was told which files it must not touch. Each report had to say what it could not verify. Disjoint ownership meant no merge conflicts and no two agents rewriting the same file.
9. **Defaults are not decisions.** Trim size, bleed, scan resolution and page grid are working assumptions with one place to change them. The art bible lists every choice as DECIDED, WORKING ASSUMPTION or OPEN, so a default never hardens into policy unnoticed.

10. **The artist's hand is what the system serves.** Ink is separated as alpha so brush texture survives. Real art is cropped and moved, never redrawn. The agent does not invent art decisions, and the generated font stays a placeholder until its shapes come from the artist's own ink.
11. **Verify tools before promising them.** We found pip packages that do not exist (`htletterspacer` and `kernmaker` both returned no matching distribution), an `ffmpeg` build with no `drawtext` filter (so our review sheets have no timestamps), and an editor that needs a newer Node than the machine had. Install and smoke-test each tool before it goes into a plan.

Build it yourself: the playbook

This is written for an AI agent that has shell access and can run sub-agents, and for a human who wants to follow along. Work in phases, and do not start a phase until the previous one passes its "done when" test. Each phase gives the goal, what to build, the key spec, the test and the most likely failure. File names are ours; rename them for your project.

Phase 0. Interview the artist

Goal. Turn the artist's intentions into a labelled decision table before writing any code. The full question list is in the section called The interview.

Build. An art bible with a master decision table. Every row is labelled DECIDED (the artist said it or wrote it), WORKING ASSUMPTION (you chose a default so something could be built) or OPEN.

Done when. Every row is labelled, the artist has read the table, and none of the art decisions were invented by you.

Likely failure. The agent fills a gap with its own taste and later treats that taste as the artist's decision.

Phase 1. Repository and house rules

Goal. A project where the agent reads the rules first and cannot break the mechanical ones.

Build. A folder with `npm init` and Playwright, a Python virtual environment with `librosa`, vendored fonts, a `CLAUDE.md`, and a linter. The rules file needs: the one law (every frame a pure function of time), banned looks, a rhythm rule, motion rules, a two-typeface limit, the

brand tokens, the workflow gates, and a short list of failures you have already hit.

Key spec. Lint rules, each a regular expression run over the scene file:

```
const RULES = [
  [/Math\.random/, 'use a seeded generator'],
  [/setTimeout|setInterval/, 'timers: every frame must be a function of t'],
  [/requestAnimationFrame/, 'the scene wrapper handles live preview'],
  [/Date\.now|performance\.now|new Date\(/, 'wall-clock time: use t'],
  [/transition\s*:/, 'CSS transition: animate inside seek(t)'],
  [/animation\s*:@keyframes/, 'CSS animation: animate inside seek(t)'],
  // plus: glow on interface text, centred title on a gradient
];
```

Done when. The linter passes a clean scene and fails a file containing `Math.random()`.

Likely failure. Rules written as taste ("make it feel premium"). Write rules a program can check.

Phase 2. A deterministic renderer

Goal. A scene file becomes an MP4, and the same input always gives the same pixels.

Build. A static file server, a headless Chromium page per worker, and ffmpeg. A scene registers itself and the renderer drives it:

```
// in the scene
scene({ duration: 16, fps: 30, poster: 15.2, seek(t) { /* paint the frame for
time t */ } });
```

```
// in the renderer: several pages in parallel, each takes every Nth frame
await page.evaluate((t) => window.__scene.seek(t), i / fps);
await page.screenshot({ path: `f${String(i).padStart(5, '0')}.png` });
```

```
ffmpeg -framerate 30 -i f%05d.png -i mix.wav -c:v libx264 -pix_fmt yuv420p \
-crf 16 -preset slow -movflags +faststart -c:a aac out.mp4
```

Read the viewport size from the page URL (`?w=1080&h=1920`) so one scene serves 9:16, 1:1 and 16:9. Our 480-frame teaser rendered in about 27 s at 9:16, 16 s at 1:1 and 26 s at 16:9, with six pages in parallel on a Mac.

The motion library is small. Closed-form damped spring, stacked springs, and a seeded hash:

```

export function springStep(dt, { w = 14, z = 0.62 } = {}) {
  if (dt <= 0) return 0;
  if (z >= 1) return 1 - Math.exp(-w * dt) * (1 + w * dt);
  const wd = w * Math.sqrt(1 - z * z);
  return 1 - Math.exp(-z * w * dt) * (Math.cos(wd * dt) + (z * w / wd) *
Math.sin(wd * dt));
}
// one spring per target change, each starting at its own time (simplified)
export function springTrack(t, keys, opts) {
  let v = keys[0][1];
  for (let i = 1; i < keys.length; i++) v += (keys[i][1] - keys[i - 1][1]) *
springStep(t - keys[i][0], opts);
  return v;
}
export function hash(n) {
  n = (n | 0) ^ 0x9e3779b9;
  n = Math.imul(n ^ (n >>> 16), 0x85ebca6b);
  n = Math.imul(n ^ (n >>> 13), 0xc2b2ae35);
  return ((n ^ (n >>> 16)) >>> 0) / 4294967296;
}

```

Our presets: heavy {w: 9, z: 0.78}, snap {w: 22, z: 0.7}, loose {w: 11, z: 0.45}, critical {w: 16, z: 1}. Use critical damping for any number that is read as a number.

Done when. Rendering one frame twice gives an identical file hash, and all three formats render.

Likely failure. A bouncy spring on a counter reads a value nobody wrote. Text wraps differently per format. Any timer sneaks in.

Phase 3. Audio and beats

Goal. Visual changes land on measured beats, and the audio is mastered to a streaming level.

Build. Music either supplied or generated in code with seeded noise (kick, tick, hat, drone, riser, hit, blip), a beat measurer, and a mixer.

```

{ "source": "music.wav", "duration": 16.0, "tempo": 89.29,
  "beats": [0.6827, 1.344, 2.016], "downbeats": [0.6827], "onsets": [1.344,
2.6773] }

```

The measurer is librosa's `onset_strength`, then `beat_track`, with an optional `--bpm` hint when it locks to half or double time. In a scene, `onBeat(t, tol = 0.15)` returns the nearest measured beat if one is within 0.15 s, otherwise `t` unchanged. Mastering is two-pass loudness normalisation:

```
ffmpeg -i premix.wav -af loudnorm=I=-14:TP=-1.5:LRA=11:print_format=json -f
null -
ffmpeg -i premix.wav -af
"loudnorm=I=-14:TP=-1.5:LRA=11:measured_I=$I:measured_TP=$TP:measured_LRA=$LR
A:measured_thresh=$TH:offset=$OFF:linear=true" mix.wav
```

Our single pass read -13.7 LUFS. The two-pass result was -14.0 LUFS with a true peak of -1.1 dBFS, which is above the -1.5 we asked for and still under -1.0.

Done when. The measured tempo is within a few percent of the intended one and the mix is within half a LUFS of target.

Likely failure. The tracker skips beat 0 and reads a few milliseconds late. Snap only cuts and hits; never snap a time that must stay put, such as a poster frame or a loop seam.

Phase 4. Review pack and critique loop

Goal. The agent looks at its own output and improves it, with the scores written down.

Build. A script that writes five things for each render: a contact sheet at one frame per second, a strip of the first three seconds at four frames per second, a phone-width sheet, a still of the last frame beside the first, and a video of the film played twice.

```
ffmpeg -i out.mp4 -vf "fps=1,scale=270:480,tile=6x3:padding=6" -frames:v 1
contact.png
ffmpeg -t 3 -i out.mp4 -vf "fps=4,scale=270:480,tile=6x2:padding=6" -frames:v
1 hook.png
printf "file 'out.mp4'\nfile 'out.mp4'\n" > list.txt && ffmpeg -f concat -
safe 0 -i list.txt -c copy loopcheck.mp4
```

The loop: render, open the images, score six axes from 1 to 10 (hook, readability, motion quality, variety, brand accuracy, sound sync), name the three worst problems with timestamps, fix them, render again. Three rounds minimum. Ship when every score is 8 or higher. Log each round.

Done when. The log shows at least three honest rounds. Ours stopped at round three with the hook at 7 and said so.

Likely failure. Scoring without opening the images, or checking only one aspect ratio. The 16:9 and 1:1 cuts broke in different places from the 9:16 one.

Phase 5. Art ingest and the layer model

Goal. A flat scan and a layered PSD both become the same stack of layers a scene can draw.

Build. An ingest script. For a flat scan it finds the paper and ink levels, converts darkness to alpha, and writes black ink on transparent. For a PSD it reads layers by name.

```
g = grayscale(scan) / 255
paper = percentile(g, 92); ink = percentile(g, 0.5)           # automatic
levels
alpha = clip((paper - g) / (paper - ink), 0, 1) ** 0.85      # ink density;
dry-brush texture survives
alpha[alpha < 0.06] = 0                                     # drop paper
grain and dust
ink_png = rgba(colour=(13, 13, 15), alpha=alpha)            # brand ink,
transparent elsewhere
```

PSD layer names are matched case-insensitively: `ink`, `line` or `lineart` is ink; `color`, `paint`, `flats` or `gradient` is colour; `bg`, `background` or `sky` is background. Each layer gets a depth for parallax: background 0.25, colour 0.9, ink 1.0, with ink blended by multiply. The result is a `panel.json` beside the PNGs:

```
{ "id": "issue-01/p02-1", "width": 4096, "height": 3072,
  "layers": [
    { "name": "bg",      "file": "bg.png",      "depth": 0.25, "blend": "normal"
  },
    { "name": "color", "file": "color.png", "depth": 0.9,  "blend": "normal"
  },
    { "name": "ink",    "file": "ink.png",    "depth": 1.0,  "blend":
"multiply" } ],
  "focus": [0.5, 0.5] }
```

The loader draws a layer of depth `d` with zoom `lerp(1, zoom, d)` and focus `lerp(0.5, focus, d)`, so far layers move less than near ones, and it clamps so the art never leaves the frame. Re-ingesting keeps hand-edited fields such as `focus` and depths. The longest side is capped at 4096 px for motion, because scans are often far larger.

Done when. Both a flat scan and a layered PSD produce the same folder layout and a scene draws them.

Likely failure. The artist delivers flattened files. Ask for layers early. Also note that we have only run the flat-scan branch, on a test image. Run the PSD branch on a real file before trusting it.

Phase 6. Colour lab

Goal. Give the artist guides to react to. Do not paint the book for them.

Build. A recipes file where each recipe has a name, the script page it comes from, three to six stops from shadow to light, an accent, the direction of the light and a few rules. Add a painter that makes a dirty gradient: a gradient along an angle, warped by low-frequency noise so the bands wander, with blooms where pigment pools, grain, and paper tooth. Keep it seeded and deterministic so the same recipe can later drift or flicker in motion. Render each recipe under the ink with multiply, and put the passes on one sheet with the recipe's source line beside it.

Export swatches in the format the artist's paint app imports. Read that app's source or docs to find out; ours accepts `.ase` and `.aco` and rejects `.gpl`.

Done when. The artist has looked at a sheet and said which recipes feel right and which do not.

Likely failure. Judging colour on stand-in ink, or accepting the agent's own opinion that a pass "looks like paint". Only the artist can say that.

Phase 7. Lettering manifest

Goal. The scripts become JSON that every other tool reads.

Build. A parser for your script format. For Fountain with a few house conventions, the grammar is small: sections are `## PAGE ONE` through `## BACK MATTER TWO`; a panel is a paragraph starting `PANEL n.`; a page header such as "Nine panels." declares the count; a cue is an all-caps speaker line with an optional type, and its text sits on the lines directly below.

```
const CUE = /^[A-Z][A-Z0-9'."&\/ -]*?)(?: \((([A-Za-z ]+)\))?)?$/; // SPEAKER
or SPEAKER (TYPE); simplified from ours
const KIND = { CAPTION: 'caption', THOUGHT: 'thought', SMALL: 'small',
               OFF: 'off-panel', FEED: 'chat', BROADCAST: 'broadcast',
               SCREEN: 'screen' };
// no type = a plain balloon; chat lines split as "handle: message"; [[margin
notes]] are stripped and listed
```

```
{ issue, title,
  pages: [{ n, layout, declaredPanels,
            panels: [{ n, description, counters,
                      letters: [{ id, kind, speaker, text } | { id, kind:
'chat', speaker, lines: [{ handle, text }] }] }] },
  backMatter: [{ n, typography, body }] }
```

Validate the structure (declared panel count equals found panel count, 22 story pages and 2 back-matter pages per issue in our format). Then verify the content with something that is not your parser:

```
grep -cE "^[A-Z][A-Z0-9' .-]* \((CAPTION\)\$" issues/*.fountain # repeat per
type; compare with the manifest
```

Done when. Per-type counts match the independent count, and you can explain every difference. Ours: six of seven types matched exactly, and a deliberately loose pattern found 1,541 cue-like lines against 1,537 items. Three of the four extras were description lines ending in a capital letter and a full stop. The fourth was a line in a back-matter page.

Likely failure. The parser silently drops text, or mistakes an all-caps line in a description for a speaker.

Phase 8. Layout push to a desktop-publishing app

Goal. Lettered pages built by script from the manifest.

Build. Pick a layout app with a scriptable interface. DesignCraft has a headless mode (`designcraft-cli script file.dcs --save out.designcraft`) and a local control port. Our push does this:

1. Create a document at trim size plus bleed, one page per story page.
2. Add layers, bottom to top: Placeholder, Art (left empty for the artist), Panels, Lettering, Layout Flags, Notes.

3. Create named paragraph styles per narrator and per balloon type, so restyling the book means editing a style.
4. Draw a default panel grid only when the page header is a plain "Nine panels.", "Six panels." or "Four panels." (3x3, 2 columns by 3 rows, 2x2) or a one-panel splash, and the panel count agrees. Any other layout gets an equal-height stack or a default grid and a visible red flag saying it needs a manual layout.
5. Place every letter in reading order. The app has no balloon primitive, so a balloon is an ellipse text frame plus a wedge-shaped tail path. Grow shapes and rebuild, up to six rounds, until no text is overset.
6. Ask the app's own inspector for overset text and compare against the panel bounds.
7. Exit 0 when clean, 1 on a build failure and 2 on lettering problems, and still write the document and page PNGs on 2.
8. Export PNGs and look at them.

Done when. All issues run, you can explain every non-zero exit, and you have viewed rendered pages.

Likely failure. Sizing by character count instead of the app's own measurement, shapes landing on the wrong page, and a fixed grid that cannot hold a dialogue-heavy panel. Seven of our twelve issues overflow the default grid. The fix we have not built is to size panels from the text each one carries.

Phase 9. Fonts

Goal. Lettering that looks hand-made, without anyone retyping every letter. Go in this order.

Step 1: use existing open fonts, and check each licence at its source. On Google Fonts the licence is a field in each family's `METADATA.pb` file in the `google/fonts` repository. We confirmed `OFL` for Anton, JetBrains Mono, Shantell Sans and Patrick Hand there. Some popular marker fonts, such as Permanent Marker and Rock Salt, are Apache-2.0 instead. Both allow commercial use. The SIL Open Font License also allows embedding the font in a PDF or a printed book, but not selling the font file by itself, and a modified version cannot use a declared reserved name. This is not legal advice.

One technical finding drove the next step. Repeated letters look typed unless the font cycles alternates. The old OpenType `rand` feature is reportedly poorly supported (forum evidence only), so the fonts we inspected use `calt` rules instead. Of thirteen comic-style fonts we inspected, only Shantell Sans had alternates.

Step 2: a generated test font as a placeholder. The pipeline:

1. Store each glyph as hand-authored centerline strokes (3 to 12 points each, joined with a centripetal Catmull-Rom spline), in font units with cap height 700 and x-height 500.
2. Expand strokes with a pressure-tapered elliptical brush. Jitter position, slant, width and endpoint overshoot from a seed so every build is byte-for-byte repeatable. Union the stamps with `skia-pathops`.
3. Make three versions of every letter and digit (`A` , `A.alt1` , `A.alt2`), space each glyph from the depth of its ink profile, and write the feature file.
4. Compile:

```
fontmake -u build/Font.ufo -o ttf otf --overlaps-backend pathops
```

The `calt` rule, in words: for each letter, look back up to five glyphs for the same letter. If the nearest earlier one is the default, this one becomes `.alt1`. If that was `.alt1`, this becomes `.alt2`. If that was `.alt2`, it returns to the default. With no earlier match, the previous glyph's group picks the starting version, so the same letter at the start of different words does not always use the default.

Test it with a shaper, not by eye. Shape a string with and without `calt` using HarfBuzz and compare glyph names. Then run a property test: generate random strings, and for every pair of identical letters within five glyphs, the glyph ids must differ. Ours: 14,027 pairs, 0 identical. The build agent's first run reported 33 identical pairs, which were repeated punctuation. Restrict the test to the letters that have alternates.

Run a font checker too. `fontspecter` gave 65 pass, 11 warn and 0 fail on ours, and the warnings were minor.

Step 3: your own hand, later. Replace the stroke data with strokes traced from the artist's ink and feed three traced takes per glyph as the three versions instead of seeded jitter. The rest of the pipeline (spacing, features, build, checks) does not change. We have not done this.

Done when. The layout app shows the alternates in a real balloon and you have a printed proof at the real size.

Likely failure. Thin strokes at print size (ours are), weak lowercase (ours is squeezed small caps), and an app that ignores the alternates. Authorship matters too: if an agent wrote the outlines, do not release the font under an open licence until the shapes come from the

artist's ink. Research found US copyright generally does not register typeface designs but can protect font software, and that AI authorship is unsettled. Check this with a lawyer if you plan to sell or publish a font.

Phase 10. The hand-lettering checklist

Goal. A list of everything that must be drawn inside the picture, so the artist can plan inking sessions.

Build. Scan panel descriptions and back-matter directions for four patterns and list each hit with page, panel, the exact words and the script's own sentence:

- sound effects: a sentence with a "lettered across / small / along" phrase and quoted text
- a recurring mark: appearances of the recurring symbol, tagged by material
- signs: a sentence with sign, board, poster or banner plus quoted text
- handwriting: handwritten, scrawled, in pencil, chalk, and similar

Also list text that recurs, since one hand has to come back each time, and the sound-effect vocabulary. Ours found 191 candidates across 12 issues: 31 sound effects with 26 distinct words, 46 mark appearances, 35 signs and 79 handwriting items. One sign recurs 11 times.

Done when. The artist has reviewed the list and dropped the false hits.

Likely failure. Patterns that are too broad. Our first pass found 94 handwriting items because it matched "a glass in her hand". Tightening the pattern gave 79.

Phase 11. The pilot page

Goal. Take one page through every stage before drawing the rest of the book.

Build. A script that generates a kit for any page from the manifest and the recipes: a brief that quotes the script, lists every element to draw and the in-picture lettering, and names the colour recipe; a print template at 150% of trim size with bleed, trim and safe guides (ours is 10.3125 by 15.75 inches including bleed, drawn at 150% of a 6.625 by 10.25 inch trim, and prints on 11 by 17); a composition guide with numbered back, middle and foreground zones as a suggestion; a study sheet for the recurring mark; and a log table for time and surprises at each stage.

The stages are: pencils, ink, scan, layers in the paint app, PSD, ingest, motion preview, lettering, proof.

Done when. The log is filled in and the format and style decisions are written down.

Likely failure. Treating the kit's defaults as decisions. They are working assumptions. Nobody has used our kit yet, so we cannot say how well it works.

Phase 12. The art bible

Goal. One document that holds what is decided, what is assumed and what is open, so a new person or agent can start cold.

Build. Sections: scope; the labelling scheme and a master decision table; visual language; the lettering system; the pipeline; characters and world, limited to what the scripts already settle; prioritised open questions, each with options and a clearly marked recommendation; the roadmap; appendices with a glossary, file map and commands.

Done when. Someone who has never seen the project can find "what is decided" in under a minute.

Likely failure. Invented canon or invented looks. Trace every statement to a file or to the artist, and say so when two files disagree.

Phase 13. Motion pieces at scale

Goal. Teasers, trailers and loops from real panels.

Build. For each piece: a brief, a score, a scene, a critique log. Scenes load panels with `loadPanels` and take every word and number from the manifest by page and panel. Chat or interface elements follow the script's own states, such as a thin rail when few people watch and a flood when many do, with the viewer counter's numbers quoted exactly.

Done when. A piece with real art scores 8 or higher on all six axes after at least three rounds.

Likely failure. Polishing procedural stand-ins. Our teaser's hook is stuck at 7 for exactly this reason, and no amount of tuning fixes placeholder art.

The interview

An agent can lift this section as it stands. Run it before writing code, in the groups below, a few questions at a time. Record answers in the artist's own words. Then label every answer in the decision table:

- **DECIDED:** the artist said it, or wrote it in their own notes or scripts.

- **WORKING ASSUMPTION:** you chose a default so something could be built.
- **OPEN:** nobody has decided.

Never invent an art decision for the artist. If they have not decided how stylised the characters are, the answer is OPEN, and you build around the gap with a clearly labelled stand-in. Show the finished table back to the artist and let them correct it.

Format and output

#	Question	Why it matters	What it drives
1	Print first, screen first, or both?	Sets page shape and how much weight motion gets	Phase order; what to build first
2	If print: trim size, bleed, and the printer's spec?	Template and layout document size	Trim, bleed and safe constants in the pilot kit and the layout push
3	How many issues and pages, and is the count fixed?	Validation and workload	Manifest checks (ours: 22 story pages plus 2 back matter per issue)
4	What will you publish: print, webcomic, social video, motion comic, several?	Decides which outputs exist	Render targets
5	Which video shapes and lengths do you need, and should they loop?	Layouts per aspect ratio; the loop seam; the first-second rule	<code>render.mjs</code> formats; each piece's brief

Art process and files

#	Question	Why it matters	What it drives
6	What media and tools do you draw with, and at what physical size?	Working size sets template and scan size	Template scale (ours: 150% of print)
7	How do you scan or photograph, at what resolution?	Ingest levels and file sizes	<code>ingest.py</code> options

#	Question	Why it matters	What it drives
8	How do you colour: flats, painted gradients, texture, digital or traditional?	Shapes the colour lab	<code>recipes.json</code> , painter parameters
9	Can you deliver separate ink, colour and background layers, and from which app?	Parallax and separate lighting depend on it	Layer names; the PSD path of ingest
10	Which apps do you prefer, and must any be open source?	Tool and file-format choices	Swatch formats; the stack
11	Where do files live and how are they named?	Stable paths for ingest and scenes	Panel ids such as <code>issue-01/p02-1</code>

Style and characters

#	Question	Why it matters	What it drives
12	How stylised should the characters be? Two or three references, for grammar only	Biggest effect on acting and silhouettes	Art bible, visual language
13	Linework rules: brush weight, spotted blacks, border style?	Ingest levels, panel borders	Art bible; template
14	Is there a fixed palette or a colour rule?	Brand tokens and the brand score	Tokens file; critique rubric
15	Which recurring motifs need one canonical drawing?	Consistency across every medium	Study sheet; checklist
16	Which characters and places need model sheets first, and what exists?	Production order	Roadmap; choice of pilot page

Lettering and type

#	Question	Why it matters	What it drives
17	Hand-lettered, set in fonts, or both? Which parts are which?	Splits the work between artist and tools	Checklist scope; font phase
18	All caps or mixed case?	Lowercase quality matters only for mixed case	Font build; text styles
19	Which narrators or voices need visibly distinct text?	One style per register	Named paragraph styles
20	Do you have a typeface preference or a sample of your own lettering?	Direction for fonts; material for a traced font	Font phase
21	How should in-story interfaces look (chat, tickers, news)?	They recur on many pages	Screen-chrome layer; shared styles

Screen and motion

#	Question	Why it matters	What it drives
22	Which in-story screens appear, and what numbers do they show?	Counters and feeds can carry the story	Manifest counters; scene scripts
23	Music: supplied, generated or none? Tempo and mood?	Input for beat measuring	<code>score.mjs</code> , <code>beats.json</code>
24	Voice-over? Whose voice, and what permission?	Optional tool; consent	<code>vo.mjs</code> (we never ran it)
25	Which looks must be avoided, and what is the hook rule?	Becomes the house rules	<code>CLAUDE.md</code> ; linter

Scripts and text

#	Question	Why it matters	What it drives
26	What format are the scripts in, and which conventions mark captions, balloons, effects, chat?	The parser's grammar	<code>manifest.mjs</code>
27	Where must text be quoted exactly: numbers, signs, names?	Exact quoting needs lookups that fail loudly	<code>lettering.js</code>
28	What is the spoiler policy for promos?	Limits what a teaser may show	Each brief's "spoilers stop at" line

Tools and constraints

#	Question	Why it matters	What it drives
29	Operating system, installed tools, any offline limits?	Decides whether the stack runs at all	Setup and smoke tests
30	How much time per page, and how many pages a week?	Keeps the roadmap honest	Roadmap. Do not invent speeds

Ownership and legal

#	Question	Why it matters	What it drives
31	What licence do you want for outputs and for any font made for the book? Are you comfortable with AI-generated font outlines?	Decides whether a font can be released	Font phase; licence choice
32	Which works must you not borrow from?	Safe references	Art bible reference rules

Limits, risks and what we would do differently

The pipeline is a working skeleton, not a finished product. Here is where it stops.

Limits

- **No real art has gone through it.** The scan, the paint, the PSD and the ingest of real files are the largest unknown. Everything on screen so far is a stand-in.
- **Only one layout app, and only its command line.** We never opened a generated document in DesignCraft's window. The alternates were confirmed only there. Other apps are untested, and the old `rand` feature is poorly supported according to forum evidence we could not confirm at the source.
- **No printed proof.** The test font is thin at 7.6 pt on screen renders. We have not handled CMYK, colour profiles or PDF/X.
- **Layout is a default stack.** Seven of twelve issues overflow the default grid. Tails are generic wedges and are not aimed at speakers. Only pages whose header is a plain nine, six or four panel grid, or a single splash, get an automatic grid. Every other layout is flagged for a hand layout.
- **Motion is one teaser.** It runs on placeholder art, its hook scores 7 out of 10, and beat measuring has only been tried on music we generated ourselves.
- **The critique loop is self-graded.** The agent scores its own frames. A human should review at least the final round.
- **One machine.** macOS, Node 22, Python 3.14. The ffmpeg build we used has no `drawtext` filter, so our review sheets carry no timestamps.
- **Weak sources in the research.** We could not read several design-press pages, so the trend section rests on blog posts. Comic suitability of the fonts is inference from letterforms. A Google Fonts rule about reserved names rests on secondary sources, and the claim that Google asks for AI disclosure rests on one forum comment.
- **Cost unknown.** We did not track time per step or the cost of the agent's work. Treat any figure you see elsewhere as unmeasured.

Risks

- **Defaults hardening into policy.** The decision labels exist to prevent this. Use them.
- **Authorship and licence of generated outlines.** If an agent wrote a font's shapes, releasing it under an open licence is unsettled. Wait for shapes derived from the artist's ink, and take legal advice before selling or publishing a font.

- **Trusting pattern-matched lists.** The hand-lettering checklist finds candidates. It does not read.
- **Agent summaries that overstate.** We re-ran the checks that mattered. Do the same.
- **Spoilers in promo material.** Briefs should say where spoilers stop, and the scenes should pull only approved text.
- **Young software.** The layout and paint apps are in active development. DesignCraft's local control port has no authentication, so keep it on the loopback interface and do not expose it to a network.

What we would do differently

1. **See real hand work first.** We built a colour lab and a font before seeing any of the artist's inked pages, and had to caveat both. Ask for one inked page and one page of lettering before building either.
2. **Run the pilot page earlier.** It should come before the font and the lab, because it settles the format and style questions they depend on.
3. **Size panels from the text.** A fixed grid failed seven issues. Let each panel's lettering load set its size.
4. **Build the manifest before the first teaser.** We hard-coded the teaser's text first, then rewired it, and the rewiring exposed our own errors.
5. **Ask agents narrower questions.** One broad research task had to be stopped and re-run with a tighter scope.
6. **Record time and cost from the start.**
7. **Put a human in the critique loop sooner.**

Stack, licences and cost

Versions are the ones installed on the machine we used. Licences are listed only where we read them in the package metadata or the repository; every other row says "not checked". This is not legal advice, so check each licence before you ship or sell anything.

Component	Version	Role	Licence
Node.js	22.23.1	Renderer and tools	Not checked

Component	Version	Role	Licence
Playwright	1.64.1	Headless Chromium driver	Apache-2.0 (package metadata); the browser's own licences not checked
ffmpeg	8.1.2	Frame encoding, audio mastering, review sheets	Our build is configured with <code>--enable-gpl</code> , so that build is GPL. Check yours
Python	3.14.6	Beats, ingest, font build	Not checked
librosa	1.0.0	Beat and onset measuring	ISC (package metadata)
Pillow	12.3.0	Image handling	MIT-CMU (package metadata)
numpy, scipy, psd-tools, soundfile	2.5.4, 1.18.1, 1.25.0, 0.14.0	Image, audio and PSD support	Not checked
fontTools	4.67.0	Font building blocks	MIT (package metadata)
fontmake	3.12.2	Compiles UFO to TTF and OTF	Apache-2.0 (package metadata)
ufo2ft, ufoLib2, skia-pathops	3.9.1, 0.19.1, 0.9.2	Font compilation and outline unions	Not checked
uharfbuzz	0.56.3	Shaping tests for alternates	Apache-2.0 (package metadata)
fontspecter	1.9.0	Font quality checks, installed with cargo	Not checked
ScriptCraft, VectorCraft, DesignCraft, PhotoCraft	Built from local clones; versions not recorded	Script editor, vector art, page layout, image editing	MIT OR Apache-2.0, from each repository's <code>Cargo.toml</code> and licence files

Component	Version	Role	Licence
Anton, JetBrains Mono, Shantell Sans, Patrick Hand	Current Google Fonts files	Display, interface and candidate lettering fonts	OFL, from each family's METADATA.pb
ElevenLabs	Optional	Voice-over	Paid account. We wrote the tool and never ran it

Cost and time. The session ran roughly five to six hours. Individual steps were quick: the layout app compiled in 48 seconds from source, the test font builds in about 10 seconds, and the teaser renders in under 30 seconds per format. We did not track the cost of the AI agent's work, so we cannot give a price. The agent itself is a paid service, and the only other paid option here is voice-over.

Appendix

A brief to hand to your own agent

Copy this into your agent's first message, together with this article.

You are building a code-driven production pipeline for a hand-drawn comic. The attached article, "Building a comic book pipeline with AI", is your playbook.

1. Interview the artist before writing code. Use the interview section's question list, a few questions at a time. Record answers in their own words.
2. Build a decision table. Label every row DECIDED (the artist said it), WORKING ASSUMPTION (you chose a default) or OPEN. Never invent an art decision or a character look. Show the table to the artist and let them correct it.
3. Set up the project. Install and smoke-test every tool before you plan around it. Vendor fonts locally. Write a rules file and a linter before the first scene.
4. Follow the phases in order: renderer, audio and beats, review pack, art ingest, colour lab, lettering manifest, layout push, fonts, hand-lettering checklist,

pilot page, art bible, motion pieces. Do not start a phase until the previous phase passes its "done when" test.

5. Rules that never bend: every frame is a pure function of time; label stand-ins loudly; verify counts with something that is not your own parser; fail loudly when a reference no longer resolves; look at your own renders before scoring them; never claim you reviewed something you did not open.

6. When you finish a phase, update a status table with three columns: built and verified, built but lightly tested, not done. Say what you did not test.

7. Stop and ask the artist before any decision that is theirs to make.

Acceptance checklist

- ☐ A decision table exists, every row is labelled, and the artist has read it
- ☐ The linter fails a scene containing `Math.random()` and passes a clean scene
- ☐ One frame rendered twice gives an identical file hash
- ☐ A piece renders in every format you promised, with audio at the target loudness
- ☐ The review pack exists and the critique log shows at least three rounds
- ☐ Ingest has been run on a real scan and, if you use PSDs, a real layered PSD
- ☐ The manifest's per-type counts match an independent count, and every difference is explained
- ☐ The layout push has been run on every issue and each non-zero exit is understood
- ☐ Rendered pages have been viewed, and a printed proof has been checked at real size
- ☐ Every stand-in is labelled, and a status table says what is real and what is not

Glossary

Term	Meaning
Pure function of time	A frame depends only on the time <code>t</code> , with no randomness or timers, so a render is repeatable
<code>seek(t)</code>	The function a scene exports; the renderer calls it once per frame

Term	Meaning
Stacked springs	One spring per change of target, each starting at its own time, so nothing restarts
Contact sheet	A grid of one frame per second that shows pacing and repetition at a glance
Beat snapping	Moving an intended time to the nearest measured beat within a tolerance
Manifest	JSON extracted from the scripts that lists every page, panel and piece of lettering
Ingest	Turning a scan or layered PSD into transparent layers plus a <code>panel.json</code>
Stand-in	A placeholder that lets work continue, always labelled as one
Screen chrome	In-story interface drawn over a picture: a chat rail, a LIVE bug, a counter
<code>calt</code>	OpenType contextual alternates, the feature that picks a different letter version by context
Overset	Text that does not fit its frame
Recipe	A named colour mood with stops, an accent and rules, tied to a script page
Pilot page	One page taken through every stage to find what breaks before the rest are drawn
DECIDED, WORKING ASSUMPTION, OPEN	The three labels for every choice

Links and credit

- Ray Fu, [Motion Design Studio with Claude Opus 5.5](#): the starting idea. We adopted its rendering approach and review loop and added everything specific to comics.
- [google/fonts](#): where we checked each font's licence field.
- The layout, paint and script apps we used belong to the open-source ArtCraft suite. Search for "ArtCraft Crafting Apps" and check each repository's licence file.

Where this stands

The skeleton works end to end on stand-ins. Scripts become lettering, panels become layers, and the same text and layers drive print pages and motion pieces. What it has never done is carry a real drawing. The next step is the one only the artist can take: draw, scan and paint one page, then run it through the pipeline and write down what breaks. Until that happens, read every claim in this article against the status table at the top.