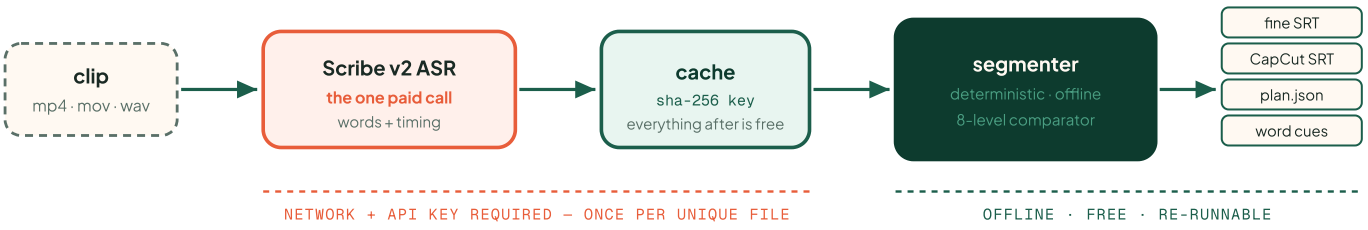
The One-Shot Subtitle Blueprint

Every component, contract, and decision rule behind a subtitle pipeline whose output you stop checking — written so you can build your own.

Ben Lim · BITSTAQ

benlimjw@bitstaq.com · bitstaq.com



What this is. A complete build specification: the components, the data that crosses each boundary, the decision rules, and the measured platform behaviour they were built against.

What it is not. Source code, or documentation for a product you have installed. Section 12 states exactly where that line falls and why.

Version 3 · 15 August 2026 · bitstaq.com/resources/one-shot-subtitle-tool

Every technical claim traces to a dated source document. The ledger is on the last page.

Questions, corrections, or a build of your own you want to argue about — **benlimjw@bitstaq.com**.

CONTENTS

What's in here

Twelve sections. The first three are the shape of the system; four through seven are the parts that were hard to get right; eight onward is how you'd build it and where it breaks.

HOW TO READ THE SOURCE MARKERS

Every factual claim carries a tag like `PROBE` naming the document it came from. Seven documents underwrite this blueprint; §12 lists them with their dates and authority level. Where a source records something as *measured*, this document says measured. Where it records something as *intent* or *uncalibrated*, this document says that too — a build spec that launders confidence is worse than no spec.

SECTION 01

The problem this shape solves

Auto-captions that need checking have not saved you anything. They have moved where you spend the time.

Every architectural decision in this document is downstream of one requirement: **the output must be trustworthy enough that you stop reading it line by line.** That sounds soft. It is not. It decomposes into four properties, and a pipeline either has all four or it has none of the benefit.

PROPERTY	WHAT IT MEANS CONCRETELY	WHICH PART OF THE SYSTEM OWNS IT
Words are right	Transcription accuracy on clean single-speaker English.	Stage 1 — the ASR engine. Bought, not built (§3).
Timing is right	Word-level start/end accurate enough that a one-word-at-a-time caption lands on the word.	Stage 1 — native word timestamps, used directly (§3).
Nothing is silently missing	No audio skipped upstream; no cue dropped downstream by the editor you import into.	Stage 1 for coverage, the CapCut-safety pass for drops (§8).
Breaks land where you'd breathe	Cues cut at clause boundaries, not wherever a character counter ran out.	Stage 2 — the segmenter. This is the part you build (§5–7).

Three constraints that follow from it

1. The segmenter must be deterministic, not a model

If the same clip can break differently on two runs, you are back to checking. The segmenter is plain rules with a stable total order over quantized keys, so its output is byte-identical across runs. SEG That is also what makes it debuggable: when a break is wrong, there is a rank that explains it, not a weight to nudge.

2. Exactly one step may cost money, and it must be cached

Everything expensive is upstream of a cache boundary; everything opinionated is downstream of it. Changing your mind about output format, preset, or line width must cost nothing and require no network and no API key. ARCH This is not a cost optimisation — it is what makes the tuning loop tight enough to actually tune.

3. Platform quirks are a separate, measurable layer

Editors and platforms mangle subtitle files in specific, discoverable ways. Those adjustments belong in one named pass that can be switched off, not smeared through the segmenter — because the same transcript must serve a consumer that needs them and one that does not. ARCH

> THE LOAD-BEARING IDEA
>
> Buy the hard AI problem. Build the deterministic layer on top of it. The ASR vendor improves its model on their schedule and your captions improve with it, with no change on your side DECK — while the part that decides *where a caption breaks* stays yours, inspectable, and identical every run.

SECTION 02

System map and data contracts

Four components, one cache boundary, five artifacts. Every edge below carries a named record — those records are the actual specification.

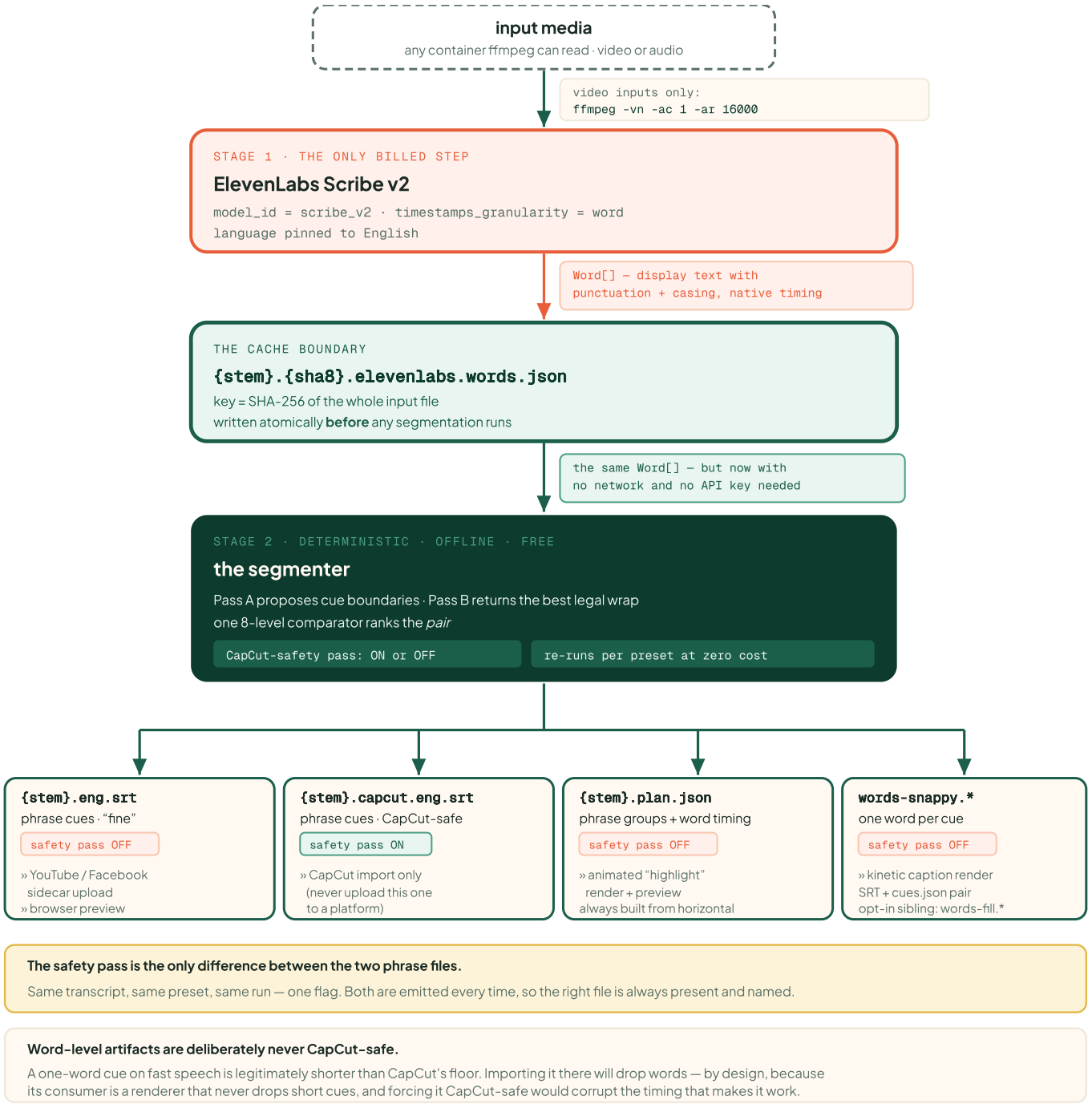


Figure 1 — the whole system, with the record on every edge. Read top to bottom: the cost boundary is the ElevenLabs box, the durability boundary is the cache, and everything below the cache re-runs for free. The five artifacts differ in exactly two dimensions: grouping (phrase vs word) and whether the CapCut-safety pass ran. `ARCH`

The two records you need to define first

Nail these two and the rest of the system is mechanical. The first is what Stage 1 hands to the cache; the second is what Stage 2 emits.

```
// The cached transcription envelope — Stage 1's entire output surface.
// One entry per spoken word, in order.
Word {
  text        : string   // DISPLAY text: punctuation and casing included
  start       : seconds  // native engine timing — not re-aligned
  end         : seconds
  confidence  : float    // ADVISORY ONLY. Never gates a break decision.
}
```

```
// What Stage 2 emits per cue, before serialization to SRT.
Cue {
  index       : int
  start, end  : seconds  // quantized to integer milliseconds before any
                         // frame arithmetic — see §8, this matters
  lines       : string[] // 1..max_lines, each <= max_chars
  words       : Word[]   // retained: the word feed and the highlight
                         // render need per-word timing inside the cue
  warnings    : enum[]   // UNSPLITTABLE_TOKEN | SHORT_DURATION |
                         // MIN_GAP_FORCED | OVERLONG_WORD
}
```

WHY `CONFIDENCE` IS ADVISORY AND NOTHING ELSE

It is tempting to break where the model is unsure. Don't. Confidence is a property of the transcription, not of the sentence, and letting it influence layout makes the output non-reproducible against a re-transcription of the same audio. It stays in the record because it is useful for diagnostics — a run that produces bad captions is either a transcription problem or a layout problem, and the confidence distribution is what tells you which. `SEG`

SECTION 03

Stage 1 — transcription

Buy this part. The interesting decision is not which engine — it is the stage you get to delete once the engine is good enough.

The engine, and how it is called

PARAMETER	VALUE	WHY
Provider	ElevenLabs Scribe	Chosen on measured word-timing quality, not on WER alone. OSS
`model_id`	`scribe_v2`	The version the rest of this document was measured against. OSS
`timestamps_granularity`	`word`	Non-negotiable. Segment-level timing cannot drive a word-level caption. OSS
Language	pinned to English	Removes language detection as a source of run-to-run variance. OSS
Punctuation + casing	required in output	This is the segmenter's *primary break signal*. An engine that returns lowercase unpunctuated text degrades §5 to pause-and-length cutting. SEG

THE DELETED STAGE — THE MOST USEFUL FINDING IN THIS SECTION

The obvious architecture is three stages: transcribe » force-align » segment, because ASR timing has historically been too loose for word-level captions. It was measured instead of assumed. Scribe's **native word timing came in at roughly 125 ms, tied with forced alignment** — so the alignment stage bought nothing and was dropped, leaving a two-stage tool. OSS

If you build this with a different engine, *re-run that comparison before you delete the stage.* The finding is about Scribe v2, not about ASR in general.

Audio preparation

Video inputs are decoded to mono 16 kHz audio before upload; audio inputs skip this. ARCH

```
ffmpeg  -vn  -ac 1  -ar 16000
drop video · downmix to mono · resample to 16 kHz — the rate speech models expect
```

This is a hard runtime dependency on `ffmpeg` (and `ffprobe`, for the frame-rate probe below). Budget for it in your install story: it is the one part of this pipeline that is not `pip install`-able. OSS

Frame rate is an input, not a detail

Probe the source's frame rate and author cues at that rate. Everything in §8 — the minimum-duration floor, the whole-frame duration arithmetic — is expressed in frames, and frames only mean something relative to a declared rate. A pipeline that authors at an assumed 30 fps and bakes onto 24 fps footage will produce cues that are individually legal and collectively wrong. The reference implementation makes this an explicit failure: a mismatch between the artifacts' recorded rate and the source's actual rate is a hard error, not a silent resample. SKILL

What Stage 1 costs, and what that means for the design

Transcription is billed by usage, not subscription, and the buyer supplies their own key. Two dated figures from the source material, which do not agree and should not be silently reconciled:

- **~$0.40 per audio-hour**, recorded 2026-06-19 in the competitive research. `OSS`
- **~$3.30–$4.00 per audio-hour**, checked against the vendor's published pricing on 2026-08-05 and stated on the companion article this blueprint accompanies.

Both are historical by the time you read this. The architectural point survives either number: **cost is per unique input file, once** — not per output format, not per preset, not per re-run. That is the cache's doing, and it is why §4 is a section rather than a footnote.

SECTION 04

The cache boundary

The cheapest section to build and the one that changes how the tool feels to use.

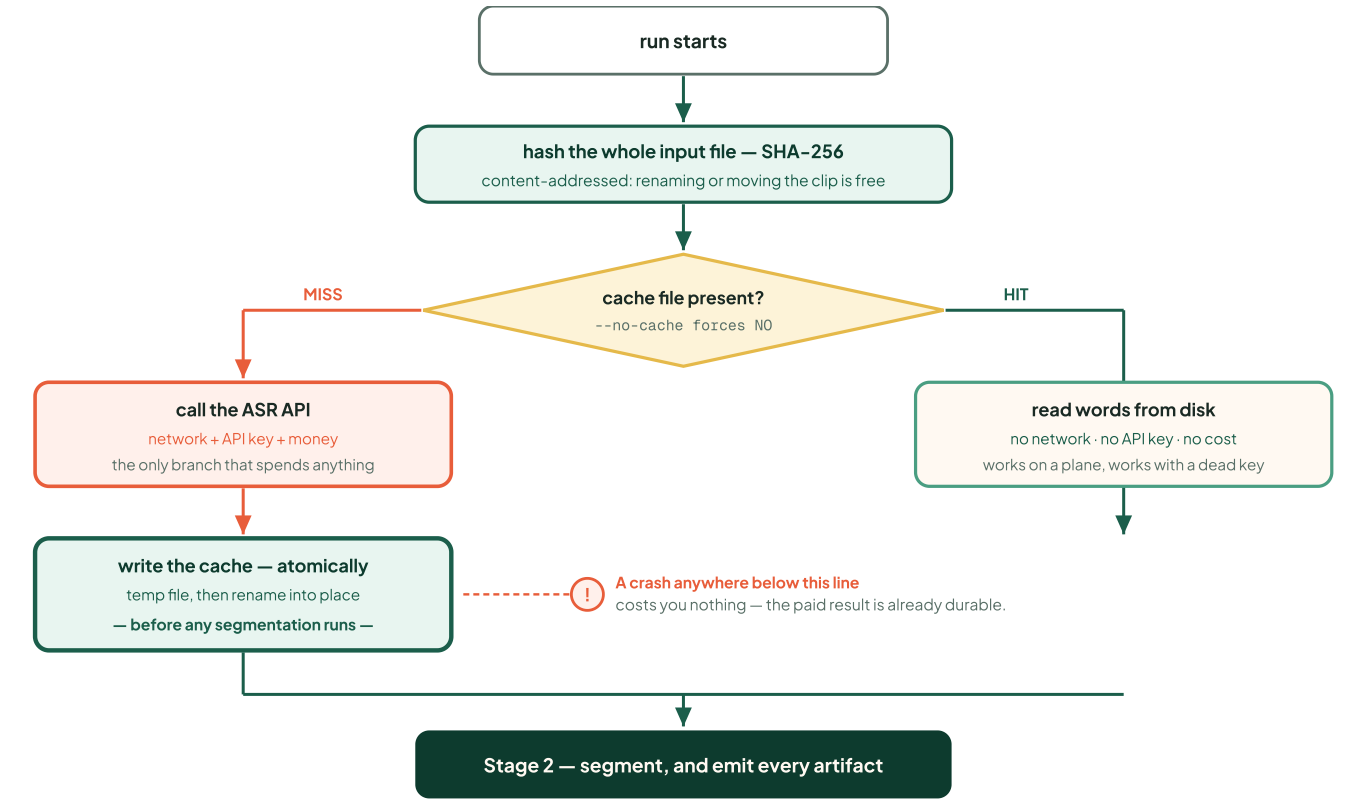


Figure 2 — the cache is an ordering constraint, not a speed optimisation. The write happens *before* segmentation specifically so that a bug in the part you are actively developing can never destroy a result you paid for. During the weeks you spend tuning §5, you will hit that path constantly. ARCH

The four rules

1. **Key on the whole file's SHA-256, not the path or mtime.** Content-addressing means a renamed, moved, or re-copied clip is still a hit — and a re-encoded clip correctly is not. ARCH
2. **Write atomically, before segmentation.** Temp file plus rename. The paid artifact must be durable before any code you are still changing gets to run. ARCH
3. **A cache hit must need no API key at all.** Not "a key that isn't used" — no key. This is what makes re-segmentation genuinely free, and it is a good self-test: if your tool refuses to start without a key on a cached clip, the boundary is in the wrong place. ARCH
4. **An empty word list is a valid, cacheable result.** A silent clip transcribes to nothing; cache that. Otherwise every re-run of a silent clip re-hits the API to rediscover silence. ARCH

Give the user exactly one escape hatch — a flag that forces a miss — and make it the only way to spend money again. ARCH Everything else in the tool should be safe to run in a loop.

> WHAT THIS BUYS YOU ARCHITECTURALLY
>
> Once re-segmentation is free and offline, presets stop being a configuration burden and become a *free re-run*. You do not need an interactive per-caption editor, because getting a different result costs one command and no money. That is why the tool has presets and not a UI — the cache is what makes that trade work. SEG

SECTION 05

Stage 2 — the segmenter

The part you actually build. Roughly two hundred lines, one third-party dependency, and one structural decision that everything else hangs off.

The segmenter turns an ordered `Word[]` into cues. Its only external dependency is a subtitle I/O library — `pysubs2` in the reference implementation — for reading and writing SRT. `SEG` No tagger, no model, no network.

The structure: a jointly-scored, two-pass candidate search

Two passes, but **not two independent stages.** Pass A proposes where a cue should *end*. Pass B decides how the text inside a cue should *wrap* across lines. The naive design runs A, fixes a boundary, then asks B to do its best inside it. That design is wrong, and understanding why is the single most valuable thing in this document.

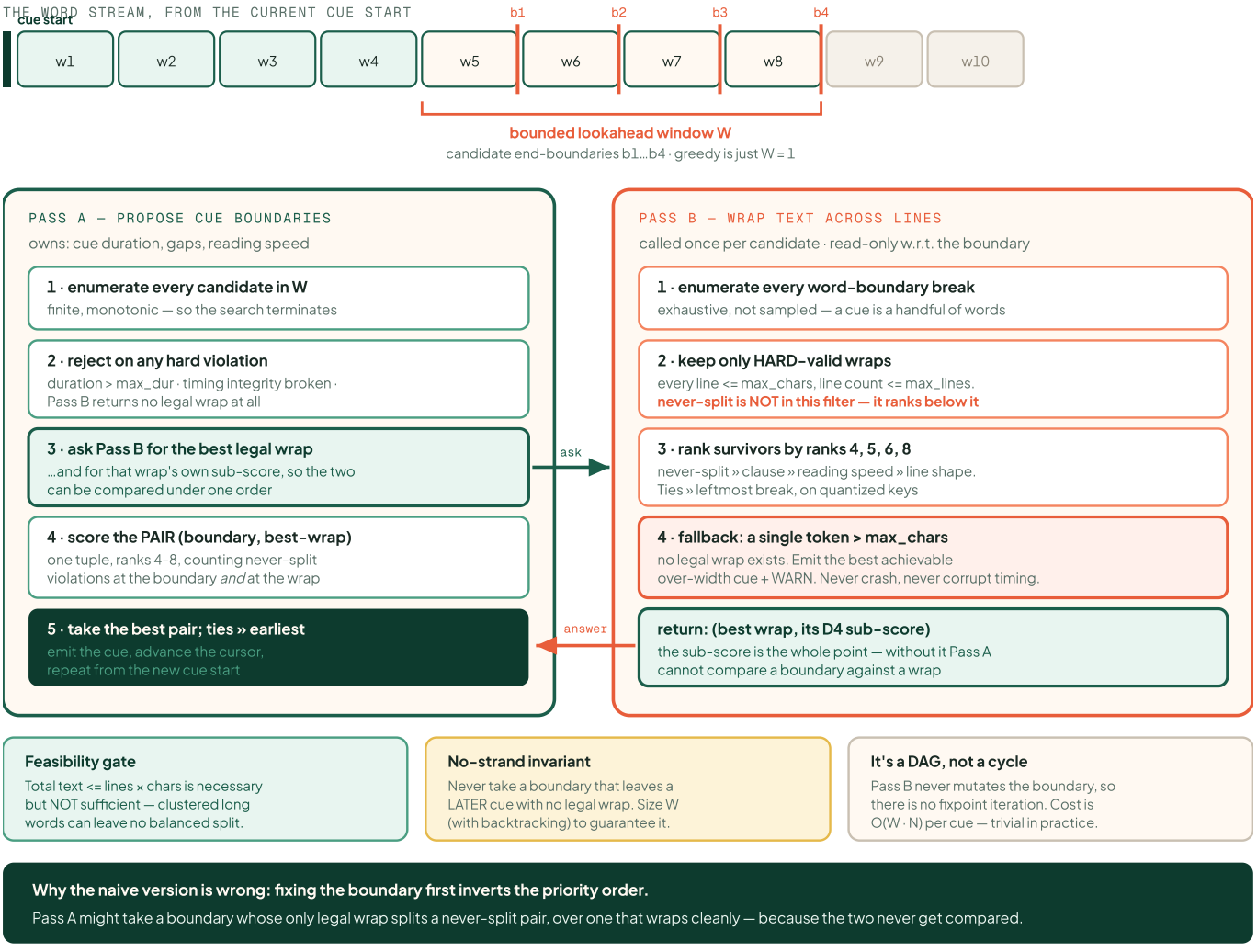


Figure 3 — Pass A and Pass B are one search, not two. The joint scoring is what keeps a single priority order governing both the cue boundary and the line wrap. Because Pass B is read-only with respect to the candidate, the dependency is a DAG and the cost is negligible — so there is no reason to accept the inverted-priority version. `SEG`

BUILD NOTE

If you build only one thing from this document correctly, make it this: **Pass B must return a score, not just a wrap.** A Pass B that returns only its chosen line break will compile, run, and produce captions that look fine most of the time — and will systematically pick the wrong boundary in exactly the cases the priority order exists to handle. It is a silent defect, and it is the one an independent review pass caught in the original design. SEG

SECTION 06

The comparator

Eight ranks. Three are hard filters that reject. Five are a tuple compared left to right. It is not a score, and that is deliberate.

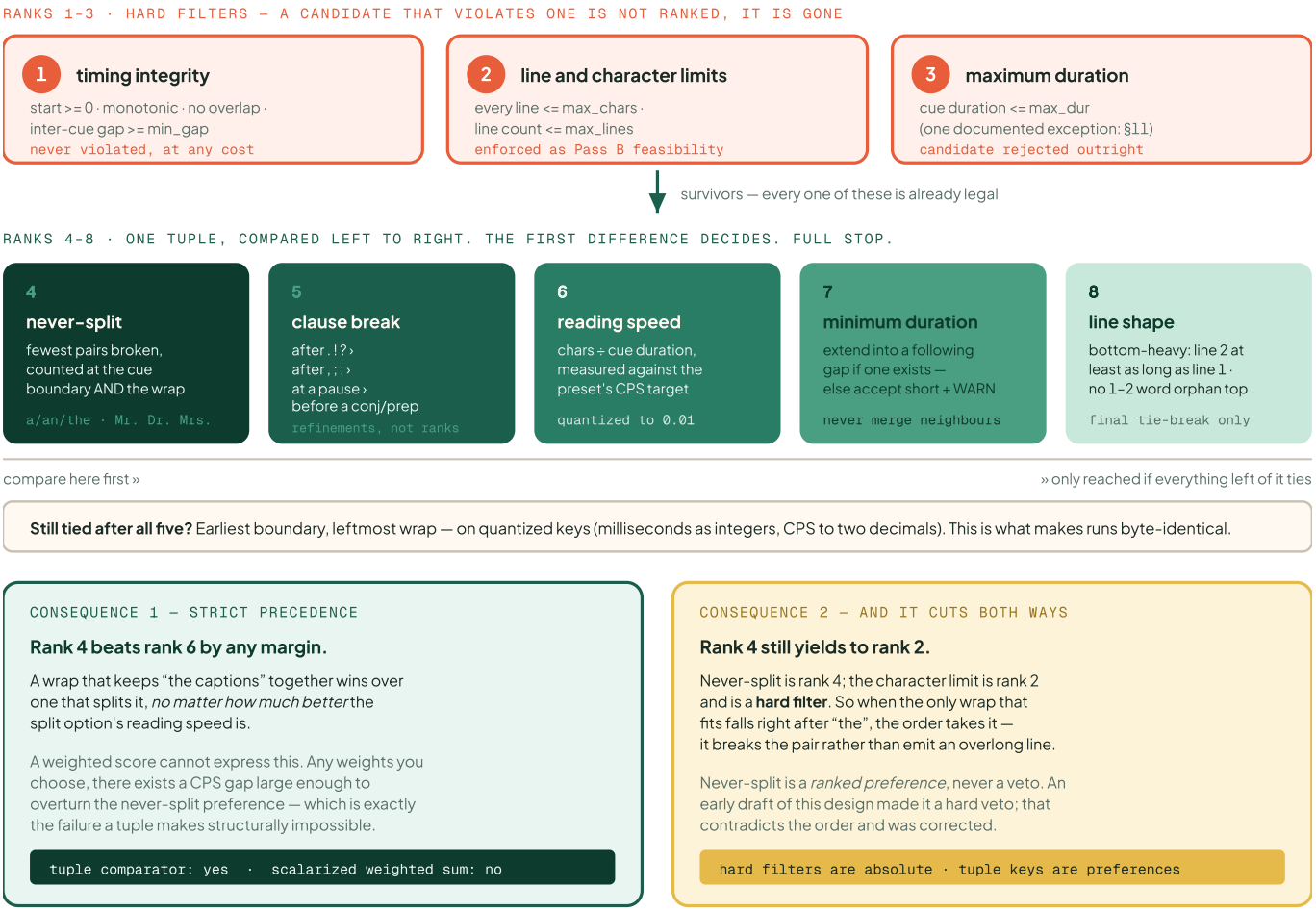


Figure 4 — the comparator, and the two things people get wrong about it. The rejected alternative here is not a strawman: a well-regarded open-source segmenter uses a tuned weighted-sum cost, and it is a perfectly good design for a system without strict precedence requirements. It just cannot express "never-split always outranks reading speed." SEG

What goes in the never-split list — and what deliberately does not

The industry style guides (Netflix, BBC, TED) converge on six pairs that should not be split across a break. **Only two of the six can be enforced by a closed word list.** SEG Getting this boundary right is what separates a list that helps from one that quietly ruins your breaks.

RULE	IN THE LIST?	WHY
article + noun	**Yes** — `a`, `an`, `the`	Closed class. A left-anchored "don't break after this word" rule is safe: an article is essentially always followed by its noun phrase.
title + name	**Yes** — `Mr.` `Mrs.` `Ms.` `Dr.` `St.` `Prof.`	Closed class, same reasoning.
adjective + noun	No	These need part-of-speech or named-entity tagging. A naive left-anchor veto *over-blocks*: forbid a break after every pronoun and you kill the perfectly legal break in "he was / right." They are protected only incidentally, by never being a *preferred* break. On clean narration that is an acceptable, documented gap.
first name + last name	No	
pronoun + verb	No	
auxiliary + verb	No	

DO NOT REACH FOR A STOPWORD LIST

The obvious shortcut — use spaCy's or NLTK's stopword list as the never-split set — is wrong in a way that is hard to notice. **Those lists contain content words.** They were built for information retrieval, where dropping common words is the goal; using one as a break-veto blocks large numbers of legitimate cuts and degrades every caption in the file. The reference implementation this design ports from deliberately omits pronouns, prepositions, and auxiliaries from its own no-break list for exactly this reason. SEG

The soft preferences that sit alongside it

- **Prefer to break *before* a conjunction** — the FANBOYS set plus common subordinators. SEG
- **Prefer to break *before* a preposition** — the standard closed list, around seventy items. SEG
- **Prefer to break *after* a comma.** SEG

These live in the ranking, where over-blocking is structurally impossible — a preference that loses simply does not win a tie. That is the general principle worth carrying out of this section: **put linguistic knowledge in the ranking, not in the filter.** A wrong preference costs you one suboptimal break. A wrong veto costs you every break it touches.

The trailing-silence budget

Three separate mechanisms want to consume the same resource: the silence between the end of one cue and the start of the next. Left unspecified, they fight, and the result depends on evaluation order — which is exactly the kind of accidental non-determinism this design exists to avoid. SEG

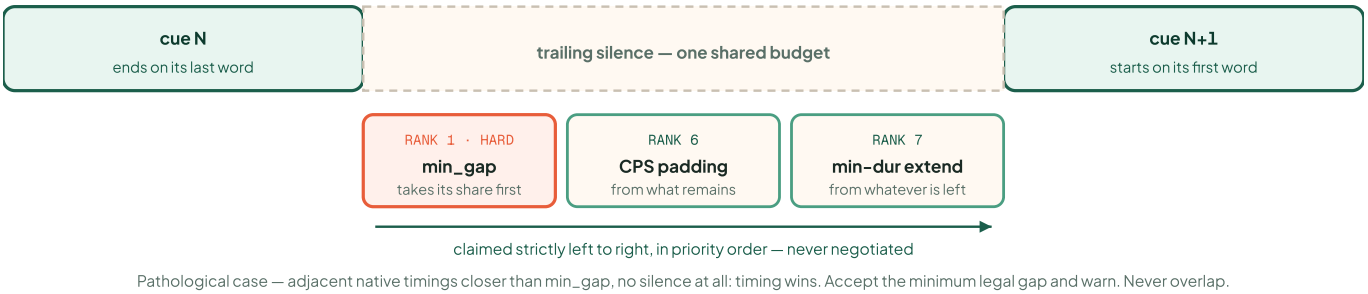


Figure 5 — one budget, three claimants, one order. Note the second-order effect this makes visible: cutting at a pause can *raise* measured reading speed, because it removes the slack silence that was padding the cue's duration. The two levers against that are a boundary with fewer characters in the same span, or padding the cue's end into the silence that follows it. SEG

THE RE-VALIDATION RULE THAT IS EASY TO SKIP

When a gap fix pulls an earlier cue's end backwards, that cue's reading speed and minimum-duration status **have both just changed**. Re-evaluate them and correct the warning before emitting. The invariant: *no cue's emitted verdict predates its final timing.* Skip this and your diagnostics will confidently report on timings the file does not contain. `SEG`

SECTION 07

Pacing, presets, and numbers

A phrase caption track and a Shorts caption track are the same algorithm with one bit flipped. Everything else is parameters.

It is tempting to write a second segmenter for short-form. Don't. The difference between "fuller" phrase pacing and "snappier" vertical pacing is not a different mechanism — it is **whether optional clause cuts are admitted to the candidate set at all.** SEG

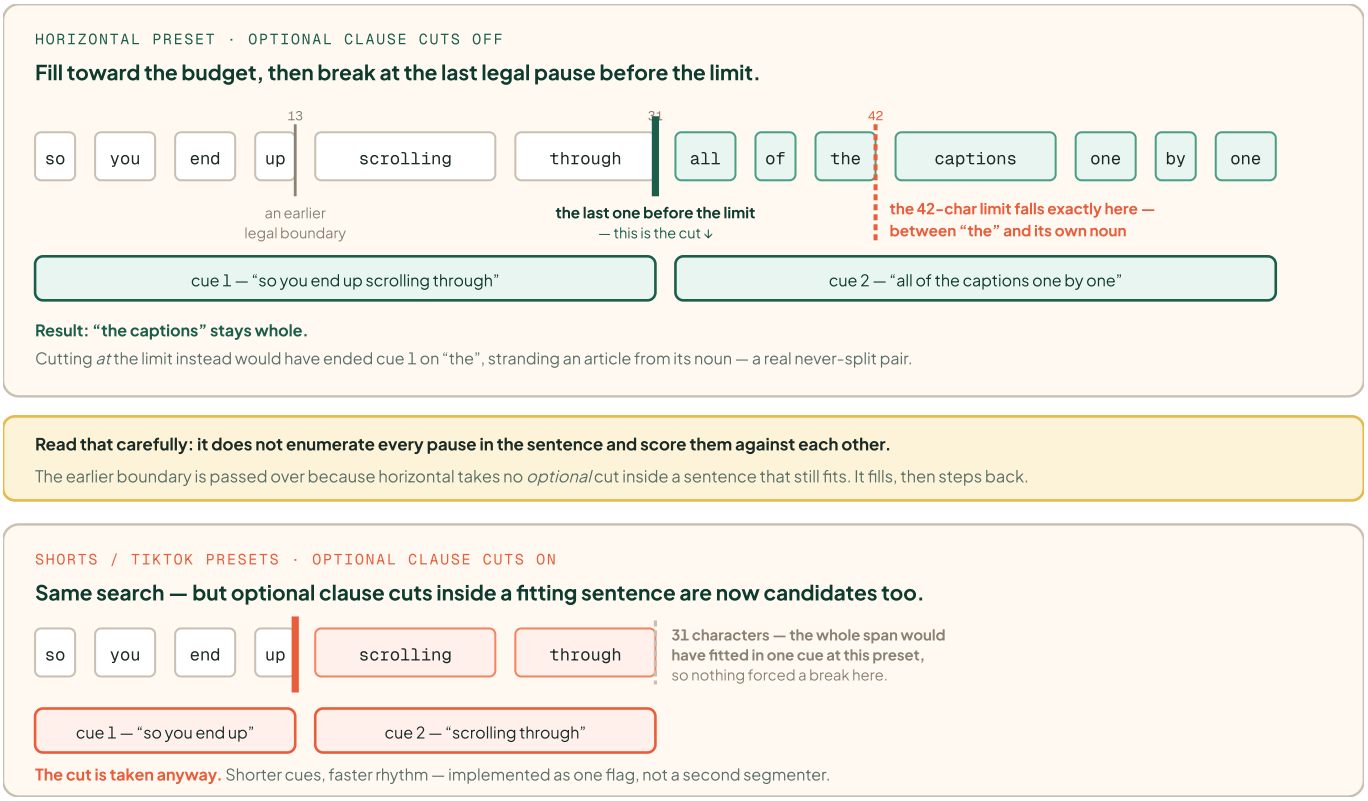


Figure 6 — pacing is emergent, not a parameter. The example sentence is engineered: at a 42-character limit the cut lands exactly after "the", which is a real never-split pair. That is what makes the horizontal behaviour visible — the naive cut and the correct one differ by an article. SEG DECK

The preset table

PRESET	CHARS / LINE	CPS TARGET	MAX LINES	OPTIONAL CLAUSE CUTS	INTENDED OUTPUT
`horizontal`	42	17	2	off — fuller pacing	16:9 phrase track; the default
`shorts`	32	20	2	on — snappier	vertical short-form
`tiktok`	27	20	2	on — snappier	tighter vertical

SHARED ACROSS ALL PRESETS	VALUE	STATUS
minimum cue duration	0.833 s	**Spec-anchored.** Font-independent, from the broadcast style guides. Not a calibration target.
maximum cue duration	7 s	**Spec-anchored.** Same.
minimum inter-cue gap	100 ms	**Spec-anchored.** Same.

THE PER-PRESET NUMBERS ARE STARTING VALUES, NOT TUNED CONSTANTS

The chars-per-line and CPS figures above are **research-backed starting values that have not been calibrated**, and the source document is emphatic that a reader must not mistake "we chose these" for "these are final." SEG The only data that can settle chars-per-line is a manual import check at the real rendering font and size — chars-per-line is font-dependent, and no font data existed when they were chosen. Treat them as parameters in your implementation, not constants: they are expected to move. Even `max_lines = 2` is a `vl` default kept explicitly tunable, since three lines may turn out better for vertical. SEG

Two further notes on presets, both of which will bite you if unstated:

- **The character limit is per *line*, not per cue.** A two-line cue at the horizontal preset can carry up to 84 characters. The single-limit picture in Figure 6 shows one break decision, not a cue budget. DECK
- **Switching preset is a free re-run, not a re-transcription.** This is the cache paying off (§4). It is also why there is no interactive per-caption prompt in this design: the cheap escape from a bad preset is running it again. SEG

What was surveyed, and what was taken from each

None of this is novel. It is a deliberate port of a proven structure, and knowing which structure — and which to avoid — is most of the design work. Six reference implementations were read from source. SEG

IMPLEMENTATION	WHAT IT DOES	VERDICT
SubtitleEdit AutoBreakLine / TextSplit	Enumerate every word-boundary split » filter through a no-break list » rank lexicographically (sentence-end early, comma early, then balance). Falls back to all splits when no clean break exists.	**Adopted as the structure.** Its enumerate » filter » rank shape maps onto a lexicographic precedence by construction. Its real English no-break list is only 19 entries — articles plus title abbreviations.
auto-subs word_segmenter.py	Two-pass, with a weighted-sum dynamic program in pass 2: base cost, length preference, soft reading-speed penalty, punctuation bonuses, a silence term.	**Signals taken, cost function rejected.** The punctuation-bonus ordering and pause preference are good. The scalarization is the thing §6 exists to avoid — note the rejection is of collapsing ranks into one number, not of dynamic programming as a technique.
stable-ts regroup mini-language	Composable split/merge operations with sticky locks; default recipe splits on sentence punctuation, then gaps over 0.5 s, then commas, then length.	**Confirms the shape; one path rejected.** Merging short neighbours to fix minimum duration changes cue boundaries and risks reading speed — extend-into-gap-or-warn is cleaner. No reading-speed handling and no line balancing.
whisper-family writers	Greedy or halving heuristics, punctuation-only, mostly no reading-speed handling. One uses a 3.0 s pause trigger; another tracks a deferred best cut point at the last punctuation seen.	**Validates simplicity.** The deferred-cut idea is a good signal. No line balancing anywhere.
srt_equalizer	Balances line lengths of pre-formed cues by halving, with a comma-proximity boost.	**Confirms comma proximity** as a break signal. Needs cues to already exist, so it cannot be the segmenter.
Fixed-window word slicers	Slice words into fixed-size kinetic cues.	**Rejected as a model** for phrase captions — but note this *is* essentially what the word-level output does (§9). Two different jobs, correctly kept separate.

Two findings from the literature worth knowing before you reach for a tagger

- **Part-of-speech features are low value on clean, punctuated input.** A conditional-random-field subtitle-segmentation study concluded that POS information should be removed *because of the compute time it*

costs. Cite that at its real precision: it is a CRF result and partly a performance argument, not a general proof that POS is useless. Combined with an ASR engine that already returns punctuation and casing, the primary break signal is present in the text before you do anything. SEG

- **Viewers prioritise meaning over geometry.** Eye-tracking work on subtitles found readers care about clause boundaries more than line shape — which is why clause break sits at rank 5 and line shape sits at rank 8, and not the other way around. Every "make the lines look balanced" instinct you have should lose to a better clause break. SEG

SECTION 08

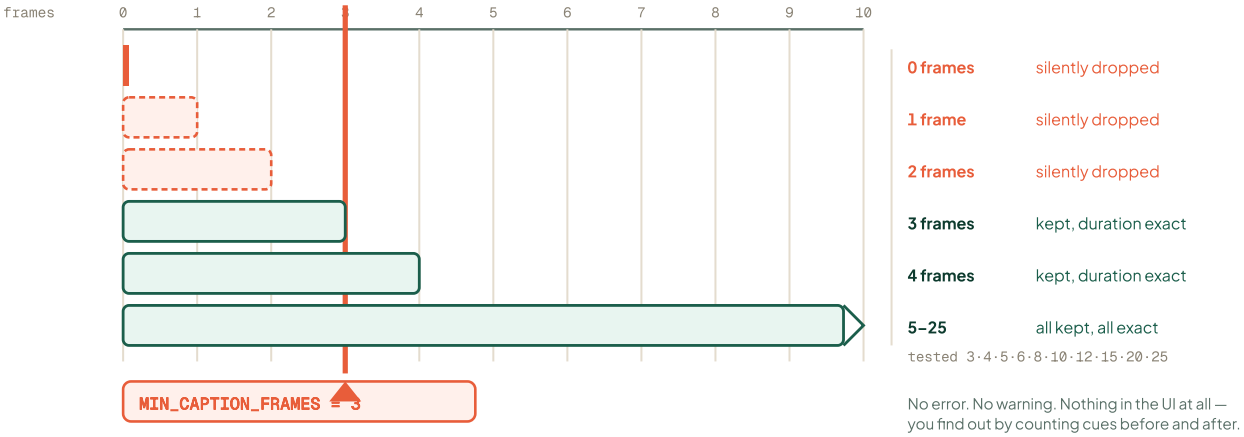
CapCut, measured

The most repeated claim about CapCut and subtitle imports is wrong. Here is what a controlled probe actually found, how it was run, and what it still does not cover.

READ THIS BEFORE YOU BUILD AGAINST IT

A widely-held belief — including in earlier drafts of this very pipeline — was that *CapCut drops one of two captions whose spans collide.* It lived as a code comment and a correlational observation: snap some cues, notice they collide, notice a user reported a missing caption. It was never tested directly. **A controlled probe on 2026-08-01 disproved it.** Colliding captions are legal and both survive. The real constraint is a minimum caption *duration*, and it is a completely different mechanism to design against. PROBE

PROBE 2 — DURATION LADDER · ONE CAPTION PER RUNG, EACH ISOLATED BY A FULL SECOND OF SILENCE



AND IT IS COUNTED ON CAPCUT'S OWN GRID — NOT IN ELAPSED TIME

```
frames = round(end × fps) - round(start × fps)
```

never (end – start) × fps — a cue whose elapsed time looks like 3 frames can still count as 2, and vanish

PROBE 1 — COLLISION · THE CASE EVERYONE GETS WRONG

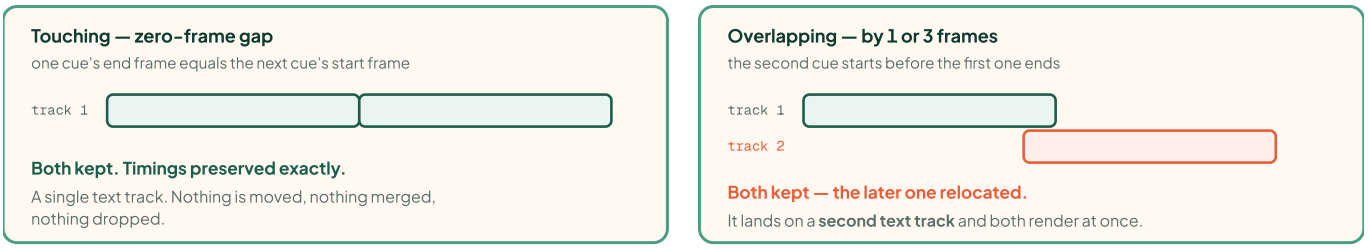


Figure 7 — what a fresh 30 fps project actually contained after import. These numbers were not read off the timeline by eye — each was read directly out of the imported project's own saved project file, which is why they are exact rather than approximate. That distinction is what turned a three-year-old assumption into a measured constraint. PROBE

How the probe was run — so you can re-run it

Reproduce this before trusting it. A future update to the editor can move the floor, and a build spec that cannot be re-verified is folklore with a date on it.

1. **Generate three isolating SRTs.** One with a touching pair and two overlapping pairs (−1 and −3 frames); one duration ladder with a caption at each of 0/1/2/3/4/5/6/8/10/12/15/20/25 frames, *each separated by a full*

second of silence so no other condition can confound the reading; one with a single caption ending exactly on a half-frame boundary, well clear of any neighbour. PROBE

2. **Import each into its own fresh project** at a known frame rate. One probe per project — never reuse a project, or an earlier probe's captions leak into a later reading. PROBE
3. **Read back the saved project file, not the timeline.** Enumerate every caption the editor actually stored, with its track index and frame range, and diff that against what you sent. What is missing was dropped. PROBE

TWO LIMITS ON ALL OF THE ABOVE — DO NOT OVERCLAIM THEM

1 · The frame rate. Everything was measured on 30 fps projects. The floor is 100 ms at 30 fps, so "3 frames" and "100 ms" are indistinguishable there — they coincide at exactly that rate and nowhere else. Whether the editor's real internal rule is a frame count or a wall-clock duration is **unresolved**. The safe implementation takes the larger of the two, which is correct under either interpretation. PROBE

2 · Storage, not rendering. Every number above describes what the editor *stores in the project after import*. A caption present in the project is assumed to render; that was **not verified against an exported video.** PROBE

The safety pass, then

Given the above, the CapCut-bound adjustment is narrow and easy to state: **extend or absorb any cue whose duration falls under the floor, and do nothing about collisions.** ARCH It is a pass over the emitted cues, it runs only for the CapCut-bound artifact, and it is the entire difference between the two phrase files in §9.

```
floor_seconds = max( MIN_CAPTION_FRAMES / fps , 0.100 )
takes the safe side of the frames-vs-milliseconds ambiguity above — correct under either reading
```

One detail that decides whether this works at all: **compute the frame count on values that have already been through whatever millisecond quantization your emitter applies**, not on raw floats. PROBE SRT carries timestamps at millisecond resolution, so your cue times get rounded on the way out — and a value can shift by a couple of milliseconds in a way that flips which frame it rounds to. Check the floor against the numbers you are actually going to write, not the ones you have in memory.

Why two files instead of one flag

An earlier design shipped one SRT plus a `--capcut-safe` flag and a warning banner. It was replaced, and the reasoning generalises well beyond subtitles: **that design made correctness depend on a user reading a warning.** ARCH

The shipped design emits both named files on every run. The right artifact is always present, its filename says what it is for, and the run summary names both and flags which one is not for CapCut. The accepted cost is that the plainly-named file changed meaning between versions — mitigated by the summary, and judged worth it. ARCH

A SEPARATE FRAME-GRID FACT — RELEVANT ONLY IF YOU WRITE PROJECT FILES DIRECTLY

If you bypass SRT import and write timings straight into the editor's project JSON, there is a second grid subtlety. **CapCut converts a frame index to microseconds by truncation; a naive implementation rounds.** At 30 fps the two disagree by exactly one microsecond on one third of all frame boundaries — the ones where the frame index is 2 mod 3 — and never by more. CapCut rewrites the project onto its own grid the first time it opens the file, so a validator comparing raw microseconds will report drift on a project nobody touched. `GRID`

The fix is not a tolerance. Accept *both legal spellings of the same frame* — an exact two-value set membership test — because a real edit is a whole frame away from both, while a ±half-frame numeric tolerance silently accepts values that encode no frame at all. And compute the truncation as one floored division of the integer numerator, not by pre-dividing to a per-frame microsecond constant: the pre-divided form is wrong at 24 and 48 fps starting at frame 99, rates you reach in practice with screen capture and some export presets. `GRID`

Scope: this is the draft-writing path, not SRT import. Whether the 3-frame drop also applies to captions written directly into a project file is explicitly **unverified** — plausible, since such markers tend to be short, but not measured. `PROBE`

One last measured curiosity, stated at its real weight: a caption ending exactly on a half-frame boundary resolved *downward* to the lower frame — the same direction Python's banker's rounding goes. **One case was tested.** Whether that is the editor's internal tie rule or a coincidence of this single measurement is unknown. `PROBE` Do not build a tie-breaking strategy on it.

SECTION 09

The output set

Five artifacts from one run. They differ in exactly two dimensions, and the table below is the whole policy.

ARTIFACT	GROUPING	SAFETY PASS	CONSUMER, AND WHY
`{stem}.eng.srt` the "fine" one	phrase	**OFF**	Direct sidecar upload to YouTube or Facebook, and browser preview. Neither frame-snaps captions nor drops short ones, so the pass would only distort timing they handle correctly. Resolution-independent; the tool never renders it.
`{stem}.capcut.eng.srt`	phrase	**ON**	CapCut import, and nothing else.
`{stem}.plan.json`	phrase + per-word timing	**OFF**	The animated highlight render and its preview. Built *always* from the horizontal preset regardless of what you asked for — see the warning below.
`{stem}.words-snappy.*` SRT + cues.json	one word per cue	**OFF**	Kinetic one-word-at-a-time captions. Deliberately never CapCut-safe.
`{stem}.words-fill.*` opt-in	leans two words per cue	**OFF**	Same consumer, looser grouping. Emitted only when explicitly requested — a default run produces five word/plan files, not seven.

On a multi-preset run the names disambiguate as `{stem}.{preset}.eng.srt` and `{stem}.{preset}.capcut.eng.srt` — the `.capcut` tag sits after the preset tag and before `.eng`. ARCH

THE TRAP IN ROW THREE

`plan.json` is **always** built from the horizontal preset, no matter which preset you requested. ARCH That is deliberate — the highlight render must use the same fine groups as the sidecar file — but it means a vertical-preset SRT and the highlight render will not agree on where phrases break. If you build this, print a note when a run emits no horizontal phrase preset. Silent disagreement between two artifacts of the same run is the worst kind of bug: everything looks right in isolation.

Two things called "a fix" — keep them apart

This is a naming trap the original codebase fell into, and it is worth stating explicitly because the two live in different layers and fix different problems. ARCH

	THE CAPCUT-SAFETY PASS	THE RENDER-PATH FIXES
Lives in	the segmenter	the composition builder
Fixes	captions vanishing on import into one specific editor	visual artifacts in the *rendered* video — a blank-word hard cut, a lingering highlight pill
Relationship	**None.** The render fixes do not make word-level output CapCut-safe, and the safety pass does nothing for render quality. Conflating them leads to applying one where you needed the other.	

Snappy is not Highlight

Two word-level presentations, two entirely different mechanics and code paths — not two settings of one thing. ARCH

SNAPPY

One word alone on screen at a time, swapping with a fade or hard cut. Your own composition. Reads the word cues file.

HIGHLIGHT

The whole phrase shown statically, with a coloured pill sweeping word to word as each is spoken. A third-party component. Reads `plan.json`.

SECTION 10

The optional render path

Everything above stops at files. If you also want captions burned into the video, this is the shape — and the one bug that will get you.

This is a genuinely separate tool from the segmenter, with a heavier dependency footprint (a browser-based renderer and a Node toolchain on top of ffmpeg). Build it second, or not at all — the SRT path is useful alone. OSS

1. **Probe the source's *display* geometry.** Strip rotation, correct for non-square pixels, round to even dimensions, and classify the aspect ratio. Carry the frame rate and the colour tags forward as a frozen record. ARCH
2. **Resolve the artifact bundle.** Glob the subtitle directory, parse and verify what you find, and match the transcription cache by the input's hash. Fail loudly if the bundle is incomplete rather than rendering something partial. ARCH
3. **Scaffold a re-bakeable project directory** that copies the clip, ships the caption component, and self-hosts its font. Make it fingerprint-aware so a re-bake after editing cues is cheap. ARCH
4. **Build the caption composition at the display canvas size** — data in, HTML out. Wait for fonts to be ready before measuring any glyphs. ARCH
5. **Render to a transparent overlay, then composite with ffmpeg** at the display dimensions: overlay at origin, carry the colour tags, clear the rotation tag, set the output duration from the source. ARCH

> THE BUG THIS DESIGN EXISTS TO FIX
>
> Size the canvas from the source's **display** dimensions, not its **stored** dimensions. A phone clip carries a rotation flag: it is stored 1920×1080 and displayed 1080×1920. Size from the stored dimensions and you get a canvas rotated ninety degrees from the footage, which produces captions that are confidently, precisely wrong. ARCH This is the single most likely thing to go wrong in this section, and it will only show up on phone footage.

Encoder selection

The reference implementation defaults to a hardware encoder on macOS and offers a universal software encoder as an explicit flag. If you are targeting more than one platform, make the software path the default off-macOS and keep the hardware path opt-in — a hardware encoder that does not exist on the host is a confusing failure, not a fast one. OSS

Give it real exit codes

A bake that fails should say which layer failed. The reference implementation distinguishes: usage error · unparseable or missing artifact bundle · frame-rate mismatch between artifacts and source · renderer or ffmpeg failure (including a hardlink failure when the working directory is on a different volume from the clip) · parameter out of range. SKILL Five distinguishable failures is worth the afternoon it takes.

> ON PATCHING THIRD-PARTY COMPONENTS
>
> The highlight render drives a vendored third-party caption component with the pipeline's own timing. Those overrides are applied **on read** — the vendored file is never edited in place. ARCH It costs a few lines and it means upgrading the component is a file replacement rather than an archaeology exercise.

SECTION 11

Build order and failure modes

A working order that keeps you honest, then every place this design still breaks.

Build order

#	BUILD	DONE WHEN	WHY THIS ORDER
1	Audio prep + one ASR call + the `Word[]` record	A clip produces a word list with punctuation, casing, and per-word timing.	Everything downstream is a pure function of this record. Get its shape wrong and you rewrite twice.
2	The cache	Second run on the same file makes no network call and needs no key. Deleting the cache and re-running reproduces it.	Build it *before* the segmenter, not after. You are about to iterate on §5 hundreds of times, and each iteration should be free.
3	Pass B alone — wrapping a fixed cue	Given a word list and limits, returns the best legal wrap *and a comparable score*.	It is the feasibility oracle Pass A depends on. Testable in isolation with hand-written inputs.
4	Pass A, joint-scored	Whole clips segment. Same input, byte-identical output, twice.	Byte-identical is the acceptance test for the whole determinism story. If it does not hold, find out now.
5	SRT emission + a validator	Every cue is legal, or carries a warning that names why.	The validator is how you tune §7's numbers without reading files by eye.
6	The CapCut-safety pass + the second file	Re-import the ladder probe; nothing is missing.	Isolated and last, because it is the only editor-specific code in the system.
7	Word-level outputs	One-word and two-word groupings emit from the same cached words.	Nearly free once §4 exists — a different grouping over the same record.
8	The render path (§10), if you want it	Captions bake correctly onto a rotated phone clip.	Heaviest dependencies, least essential. The rotated clip is the test that matters.

The parameter surface a tool like this needs

Not our invocation — the shape any implementation converges on, and why each one has to exist.

PARAMETER	WHY IT IS NOT OPTIONAL
input path	The one positional argument. Everything is keyed off its hash.
output directory	One run writes five to seven files. They belong together, and not next to your source footage.
frame rate	Must be the source's real rate, probed rather than assumed (§3). The floor arithmetic in §8 is meaningless without it.
preset (repeatable)	Because re-running is free, multiple presets in one invocation costs nothing but a loop.
force re-transcribe	The single, explicit escape hatch that spends money. Nothing else should be able to.

Where it breaks

Proper names — the gap that does not close

Names the model has never seen — people, products, anything outside common vocabulary — come out wrong, and no amount of segmentation fixes a wrong word. This is not a rough edge; it is the single most-reported complaint in the captioning market generally, `MON` and it is the one thing this architecture cannot answer. There are two real remedies, both outside the pipeline: a custom dictionary of the names you use often, or the underlying model improving. A human pass before publishing stays part of the workflow. Say so

plainly wherever you present this — the audience most likely to build it is the audience most likely to test it on their own names.

Degraded or absent punctuation

The whole design assumes an engine returning clean punctuation and casing. Without it, clause detection loses its primary signal and falls back to pause-gap and length cuts: still deterministic, still legal, noticeably worse breaks. **Graceful degradation, not a supported mode.** SEG If your engine does not punctuate, this design is the wrong starting point.

The two documented over-limit escapes

- **A single token longer than the character limit.** No word-boundary split can help. Emit the best achievable over-width cue and warn — never crash, never corrupt timing. This is the *only* way a multi-token cue should ever exceed the limit, which is precisely what makes the distinction testable: over-width with one token is a warning; over-width with several is a bug. SEG
- **A single word longer than the maximum duration.** A hard limit, but a lone over-long word cannot be split in time without fabricating timing. Emit it at its native span and warn. Vanishingly rare at normal speech rates — named so it is not a silent gap. SEG

Dense back-to-back speech

Multiple unavoidable short-duration warnings are *expected*, not a failure — the cues are legible, there is simply no silence to extend into. Resist the instinct to merge neighbours to clear the warning: it changes cue boundaries and risks reading speed, and the extend-or-warn rule is cleaner. SEG

Everything §8 does not cover

Non-30 fps timelines. Export and render behaviour as opposed to what is stored after import. Captions written directly into a project file rather than imported as SRT. All flagged in place; all still open. PROBE

What is genuinely unoccupied about this shape

Worth knowing before you spend the weeks: a survey of the caption tooling landscape found the market splits into two camps, and this combination sits in the empty intersection. OSS

- **Styled caption SaaS** exports subtitle files, but segments by word or character count rather than clause, and offers no editor-import guarantee.
- **Open-source ASR wrappers** emit clean files but split naively — an open issue on one of the most popular describes the exact failure this segmenter removes: full stops appearing mid-sentence, segments splitting people's names, random sentence cuts once length limits are applied.
- **Commercial ASR APIs** offer characters-per-caption or rows-per-screen constraints. None of the surveyed ones do clause- or conjunction-aware breaking.

The defensible position is the *intersection*, not any single feature: a clean, editable subtitle file, with linguistically-principled breaks, that survives import into the editor you actually use. OSS That is also why the segmenter is the part worth building yourself and the transcription is the part worth buying.

SECTION 12

Bill of materials · the line · sources

Exactly what this is built on, exactly what this document withholds and why, and where every claim above came from.

Bill of materials

COMPONENT	WHAT WE USE	NOTES AND ALTERNATIVES
ASR engine	ElevenLabs Scribe, `model_id=scribe_v2`	Word-level timestamps are the hard requirement. Chosen on measured timing quality; the alignment stage was deleted *because of* that measurement (§3), so swapping engines means re-running it. `OSS`
Subtitle I/O	`pysubs2`	The segmenter's only third-party dependency. Reading and writing SRT only — it does no segmentation. `SEG`
Audio + video	`ffmpeg`, `ffprobe`	Hard runtime dependency, not a Python package. Audio extraction, geometry probing, final compositing. Check its licensing for your distribution. `OSS`
Segmenter	yours — around 200 lines	No tagger, no model, no network. §5–7 is its complete specification.
Render engine optional, §10	HyperFrames (`hyperframes@0.7.3` at time of writing)	Browser-based; brings Node and a headless Chrome into your install story. Any renderer that takes timed HTML and emits a transparent video works here. `ARCH` `SKILL`
Encoders optional, §10	`h264_videotoolbox` on macOS · `libx264` everywhere	Hardware path is Apple-only. Make the portable one the default off-macOS. `ARCH` `OSS`
Orchestration	a Claude Code skill wrapping the CLI	Deliberately thin. The engine is a command-line tool, so a script, a scheduled job, or another agent can call it as one step in something larger — which is the actual argument for building it as a CLI rather than a skill. `SKILL`

What this document includes, and what it doesn't

This blueprint is the free rung of a two-part split: **share the thinking, gate the leverage.** That line is drawn deliberately, and stating it precisely is more useful to you than pretending it isn't there.

IN HERE — THE THINKING

- Every component and the record on every edge
- The complete decision rules, in priority order
- The exact contents of the never-split list, and what is deliberately excluded
- Every named library and service
- The measured platform behaviour *and the method*, so you can re-verify it
- The alternatives that were rejected, and why
- Every limit, gap, and uncalibrated number, labelled as such

NOT IN HERE — THE LEVERAGE

- **No implementation source.** No function bodies, nothing you could paste into a file and run.
- **No invocation reference.** Earlier versions of this document listed the tool's own commands — removed. They are useless without the tool, and their presence made a build spec read like product documentation. `§11` gives the parameter surface and the reasoning instead.
- **No packaged artifact** — install path, dependency pinning, calibrated presets, or the render project scaffolding.

Two formulas do appear, because they are findings rather than implementation: the frame-count expression in §8 and the duration floor. Withholding a one-line arithmetic fact would make this document less correct

without protecting anything — you would derive both from the measurements in the same section anyway, and the whole point of publishing the measurements is that you should not have to rediscover them.

> IF YOU WOULD RATHER NOT BUILD IT
>
> The blueprint above is the whole architecture. I've only wired it up as a skill on my own machine, and I haven't had the time to clean it up for anyone else. If you're interested, email **benlimjw@bitstaq.com** and let me know. If there's enough interest, I'll put the time in and package it properly. If you build your own from this instead, that is a completely legitimate outcome. It is why the specification is this detailed.

Source ledger

Seven documents underwrite every claim in this blueprint. Each is marked with its authority level, because "measured" and "intended" are not the same kind of fact and a build spec should never blur them.

MARKER	DOCUMENT	AUTHORITY	UNDERWRITES
PROBE	`capcut-caption-behaviour.md` measured 2026-08-01	**Measured.** Read out of the saved project file, not off the timeline.	Everything in §8: the 3-frame floor, the collision results, the grid arithmetic, the probe method, the half-frame case, and both stated limits.
ARCH	`english-subtitles-architecture.md` as-built, 2026-06-25	**As-built**, generated from the shipped code.	The system map and edges (§2), the cache rules (§4), the artifact policy and the two-files decision (§8–9), the render path (§10).
SEG	`subtitle-segmenter-algorithm.md` research, 2026-06-17	**Intent.** Authoritative for *why we decided*, not for current code.	The whole of §5–7: the two-pass joint search, the comparator, the never-split scope, the trailing-silence budget, the preset values and their uncalibrated status, the reference-implementation survey.
GRID	`capcut-frame-grid-rounding-drift.md` research, 2026-07-29	**Measured root cause**, high confidence, three independent measurements agreeing.	The truncate-versus-round microsecond grid note in §8, and the two-legal-spellings comparison rule.
OSS	`english-subtitles-open-sourcing-research.md` research, 2026-06-19	**Intent** + competitive survey; several claims flagged by the source as unverified.	Engine parameters (§3), the deleted alignment stage, the dependency and licensing notes, the market-gap analysis in §11.
SKILL	`create-subtitles/SKILL.md`	**Operational**, the shipped orchestration contract.	Frame-rate probing as a first step (§3), the exit-code taxonomy (§10), the skill-versus-engine split (§12).
DECK	`one-shot-subtitles/content-plan.md` 2026-08-06	**Editorial**, records which claims were corrected against measurement and why.	The horizontal-pacing correction and its example sentence (§7), the per-line-versus-per-cue clarification, the framing rule this document follows.

One further source, marked `MON`, appears once: the tools-monetization research dated 2026-08-06, for the market-wide observation that proper-name errors are the most-reported captioning complaint (§11).

A note on how this is written

This document describes what each stage *does*, never who wrote it, and never frames any one component as the only part that was built. Choosing the components, deciding what hands off to what, and wiring the chain *is* building the tool. If you take this and build your own, the same will be true of yours.

Ben Lim · benlimjw@bitstaq.com

BITSTAQ · bitstaq.com/resources/one-shot-subtitle-tool · Version 3, 15 August 2026