

Honest Automated Subtitle Repair: A Reproducible Defect-Injection Benchmark of the VideoText Subtitle Quality Engine

Santhosh Guntupalli

VideoText (videotext.io)

Technical report v1.0 · 27 October 2026 · Licensed CC BY 4.0

ABSTRACT

Subtitle files in SRT and WebVTT often reach publication with overlapping cues, impossible timestamps, over-long lines, and cues that are too fast to read. Automated fixers can correct many of these errors, but they can also make a file worse without saying so, for example by trimming an overlap until the cue is too short to read. We describe the deterministic subtitle checker and fixer in VideoText (videotext.io) and evaluate it with a reproducible defect-injection benchmark. The benchmark has 2,000 SRT files (118,803 cues) built from public-domain text, and 11,597 controlled defects in five classes, plus three stress conditions where no in-place repair exists. An independent rule oracle, written for this study, scores every output. We compare the engine with two simple baseline fixers. The engine detected all 11,597 injected defects (lower 95% Wilson bound 99.97%). It raised no false positive on 107,206 cues that were not modified. On defects that can be repaired, 1,190 of 1,200 files (99.2%) came out fully compliant. The simpler of the two baselines reached 50.0%, and a stronger baseline that also extends cue durations reached 100%. Where cue text overflowed two lines, the engine produced compliant output for all 200 files and both baselines for none. In the two dense stress conditions, no fixer produced a compliant file. The baselines turned 1,200 overlaps into unreadable cues without reporting them. The engine left those cues unchanged, reported each of its 2,412 remaining violations, and marked every affected file for review. All 10,322 fixes the engine claimed were confirmed in the exported files. The engine never changed the words of a cue, never moved a cue it did not need to move, and gave the same output when run a second time on all 2,000 files. Median processing time was 1.1 ms per 60-cue file. The free public *check_subtitles* tool uses a separate, lighter checker. Before a fix made for this report it missed every three-line cue (86.2% recall on the corpus). With the fix, it detects all 11,597 defects (§5.5). We publish the generator, corpus, oracle, and per-file results so that other subtitle tools can be measured the same way.

Keywords: subtitles; closed captions; SRT; WebVTT; subtitle quality control; reading speed; characters per second (CPS); overlapping subtitles; automated repair; benchmark; reproducibility

Scope of the results. Sections 5.1–5.4 measure the subtitle engine behind VideoText's Fix Subtitles tool, its API, and the `fix_subtitles` tool for AI assistants. The free `check_subtitles` tool for AI assistants is a separate checker. On 27 October 2026 it did not enforce the two-line limit. That is fixed in code, and the fixed checker is measured in §5.5 and Table 4. Do not quote engine figures for the public checker, and do not quote Table 4's 100% for the live tool until the fix is deployed and checked again.

1. Introduction

Subtitles reach viewers through many hands. A speech recognizer drafts them, an editor retimes them, a translator rewrites them, and a platform ingests them. Each step can introduce structural errors. Two cues can overlap. A cue can end before it starts. A line can be too long for the screen. A cue can show too many characters for the time it is visible. Players and platforms deal with these errors in different ways. Some reject the file, some render it wrongly, and some display it as it is, so the viewer gets text they cannot read in time.

Many tools offer to "fix" subtitle files automatically. The obvious fixes are easy to write. An overlap can be removed by moving the earlier cue's end time to the next cue's start, and a long line can be wrapped. But these simple fixes can create new defects. Trimming an overlap can leave a 0.3-second cue with forty char-

acters, which no viewer can read. Wrapping a long cue can leave four lines on screen. A fixer that reports success in these cases is worse than no fixer, because the user stops checking.

This report makes three contributions:

1. We describe the design of the deterministic subtitle quality engine in VideoText (§3). Its central rule is that a fix is reported only when the exported file actually has it, and that any remaining defect must be reported.
2. We define a reproducible defect-injection benchmark (§4). It has five common defect classes and three stress conditions, an oracle that does not call the engine's code, two baselines, and metrics for detection, repair, collateral change, and honesty of reporting.
3. We report results for the engine and the baselines, including the cases the engine cannot fix (§5), and we discuss the threats to validity (§6). The corpus, code, and per-file results are released so that the benchmark can be used for other tools.

The scope is limited on purpose. This study does not measure speech-recognition accuracy, translation quality, or viewer comprehension. It measures whether a subtitle file meets explicit structural and readability rules after repair, and whether the tool reports the file's state truthfully.

2. Background

2.1 Subtitle formats

SubRip (SRT) is an informal plain-text format of numbered cues, each with a `start --> end` timestamp line and one or more lines of text. WebVTT is a W3C format with similar cue timing plus cue settings and identifiers [1]. Neither format stops a file from having overlapping or zero-length cues. Overlaps are sometimes intended, for example two positioned speakers on screen at once, so a repair tool must tell intended overlaps from accidental ones.

2.2 Readability rules

Broadcaster and streaming style guides limit the line length, the number of lines, and the reading speed. A widely used profile for English is at most 42 characters per line, at most two lines per cue, and a reading speed of about 17–20 characters per second (CPS) for adult content [2]. Guides also set a minimum time on screen. In this study we use the profile the engine enforces by default: 42 characters per line, 2 lines, 20 CPS, and at least 1.5 s on screen for any cue over 20 characters. When we count characters for reading speed, formatting markup is ignored and a line break counts as one space.

2.3 Automated repair and its risks

Repair is a constrained problem. A cue's start time is usually tied to speech, so it should not move without evidence. The time available to a cue is bounded by its neighbours. And the text is the author's, so a structural fixer should not change it. When these constraints conflict, any repair gives something up. We argue that a fixer must make that choice visible to the user.

3. The VideoText subtitle quality engine

VideoText is a web platform for video transcription and subtitle work. Its tools turn video into transcripts and subtitles, and translate, check, fix, burn in, and convert subtitle files. Uploaded media goes through an API to a queue (Bull on Redis) and is processed by background workers. Speech recognition uses a hosted model, with ElevenLabs Scribe v2 as the primary model and OpenAI whisper-1 as the fallback. Subtitle checking and fixing, the subject of this report, runs in the worker on the subtitle text alone. This part is de-

terministic and does not call any machine-learning model. An optional AI grammar pass and an optional filler-word removal step exist, but both are off by default and were not used in this study.

3.1 Pipeline

For each file the engine does the following:

1. It parses SRT or VTT, keeping VTT identifiers and settings.
2. It checks for drift against the media duration when that duration is known.
3. It combines cues that start at the same moment (within 0.1 s), dropping exact duplicates.
4. It repairs cues that end at or before their start.
5. It resolves overlaps, in time order, including a cue nested inside another.
6. If the user enabled it, it rewraps lines to the 42-character, 2-line limit. A cue that needs more than two lines is split into several cues, which share the original interval.
7. If the user enabled it, it extends cues that are too short or too fast to read, without crossing the next cue and with a 10 s maximum.
8. It serializes the file, then parses the exported text again and validates it.

3.2 Design rules

- **Claims are checked against the export.** Every finding has a `fixed` flag. After serialization, the engine parses the file the user will download and checks it again. Any finding still present is set to `fixed = false`, and a remaining problem that was not already listed is added as a new finding. The file's status is passed only when the export meets every default check and no changed timing needs to be verified by a person.
- **No unreadable slivers.** An overlap is resolved by trimming only when the trimmed cue is still a real cue. A cue over 20 characters must keep at least 1 s on screen and stay under twice the reading-speed limit. Otherwise the overlap is left in place and reported, with a suggestion for what the user can do.
- **Text is not changed.** Unless the user enables the grammar or filler options, the words of the file are checked against the source, and the user is warned if they differ.
- **Start times stay where they are.** Repairs move end times. Start times move only for an explicit offset or drift re-sync, which the user must request.
- **Author line breaks are kept.** When a cue's lines are each 42 characters or less but there are too many of them, the engine splits the cue at the author's line breaks instead of rewrapping the text.
- **Positioned captions keep their own lanes.** Files with VTT cue settings or identifiers are not treated as having accidental overlaps.

4. Benchmark method

4.1 Clean corpus

The source text is the body of five public-domain books from Project Gutenberg [3]: *Pride and Prejudice* (#1342), *Alice's Adventures in Wonderland* (#11), *Frankenstein* (#84), *Moby Dick* (#2701), and *The Adventures of Sherlock Holmes* (#1661). The SHA-256 of each downloaded file is recorded in the results. A seeded generator (mulberry32, seed 20261027) cuts the text into cues of 4–14 words, ending a cue early at a sentence end. It balances each cue over at most two lines of 42 characters. Each cue's duration comes from a reading speed drawn uniformly from 11–17 CPS. Durations are at least 1.2 s, at least 1.55 s for cues over 20 characters, and at most 7 s. The gap between cues is 0.12–0.9 s. Each file has 60 cues. By construction, every clean file passes the oracle (§4.3), and we checked that it does: no clean file has a violation.

4.2 Defect injection

Each defective file starts as a fresh clean file. Defects are injected into randomly chosen cues that are eligible for that defect, and no two injected cues are next to each other, so each finding has a single cause. A single-class file has 6 defects. A mixed file has 10, two of each class. Table 1 lists the classes. The three stress conditions are built so that the defect cannot be repaired in place. A correct fixer must split the cue, borrow time from a neighbour, or report the defect.

Condition	Injection	Files	Defects
Clean	none (control)	200	0
Overlap	end moved 0.3–1.5 s past the next cue's start	200	1,200
Long line	two-line cue (>48 chars) joined into one line	200	1,200
3+ lines	cue of ≥ 6 words broken into three lines	200	1,200
Fast reading	duration cut to give 25–40 CPS	200	1,200
Invalid timing	end $\leq$ start (30% equal, 70% 0.05–1 s before)	200	1,200
Mixed	two of each class above	200	2,000
<i>Overflow text*</i>	<i>two cues merged into one line of over 88 characters</i>	<i>200</i>	<i>1,197</i>
<i>Dense overlap*</i>	<i>next cue starts 0.3–0.8 s after a cue of over 30 characters</i>	<i>200</i>	<i>1,200</i>
<i>Dense fast*</i>	<i>25–40 CPS cue, next cue starts 0.04 s after it ends</i>	<i>200</i>	<i>1,200</i>
Total		2,000	11,597

Table 1. Benchmark conditions. Rows marked * are stress conditions, where the defect has no in-place repair. In 3 of the 1,200 overflow attempts no eligible cue pair was found, so that condition has 1,197 defects.

4.3 Oracle

We wrote a rule checker for this study that does not import any engine code. For every cue it checks the following: end $>$ start; reading speed ≤ 20 CPS; at least 1.5 s on screen if the cue has more than 20 characters (the minimum-duration rule); at most 2 lines; no line over 42 characters; and no overlap in time with any other cue. A file is *compliant* if the oracle finds no violation in it.

4.4 Systems compared

- **VideoText engine** at commit 50cf3f0, through `fixSubtitleContent` with *Fix timing* and *Line breaks* enabled, and with grammar, filler removal, media duration, and re-sync off. For detection we use the check step: `fixSubtitleIssues` with every repair option off, plus `validateSubtitleEntries`.
- **Baseline A (clamp + wrap)**. A cue with end $\leq$ start gets end = start + 1 s. Each cue's end is clamped to the next cue's start. Each cue is wrapped greedily at 42 characters, with no limit on the number of lines.
- **Baseline B (clamp + wrap + extend)**. Baseline A, then each cue is extended toward the duration its reading speed and minimum-duration rule require, without crossing the next cue's start.

We wrote both baselines to represent the simple fixes described in §1. They are not reimplementations of any commercial product.

4.5 Metrics

- **Detection recall**: the share of injected defects for which the check step reported a finding of the matching type on that cue.
- **False-positive rate**: the share of cues that were not modified (all cues in clean files, plus cues that were not injected in defective files) that received any finding.

- **Compliance:** the share of files that are compliant after repair, and the number of violations remaining.
- **Claim accuracy:** the share of findings marked fixed whose violation is absent from the exported file at that cue, and the share of findings left open that the oracle confirms.
- **Unreported residuals:** the number of violations left in the export that are not named by an open finding or a warning on that cue.
- **Silent passes:** the number of files with status passed that the oracle finds non-compliant.
- **Text preservation:** the share of files where the output word sequence, in time order, equals the input word sequence.
- **Collateral timing change:** the share of cues that were neither injected nor next to an injected cue whose start or end moved by more than 1.5 ms. Cues are matched by text, and texts that occur more than once are excluded.
- **Idempotence:** the share of files where fixing the fixed file again returns identical bytes.
- **Runtime:** wall-clock time of one `fixSubtitleContent` call, covering parse, check, fix, serialize, and re-validation.

Proportions are reported with Wilson 95% intervals where the interval adds information. The entire run is deterministic. Running it twice gave byte-identical corpora and identical results, apart from timestamps and runtimes.

5. Results

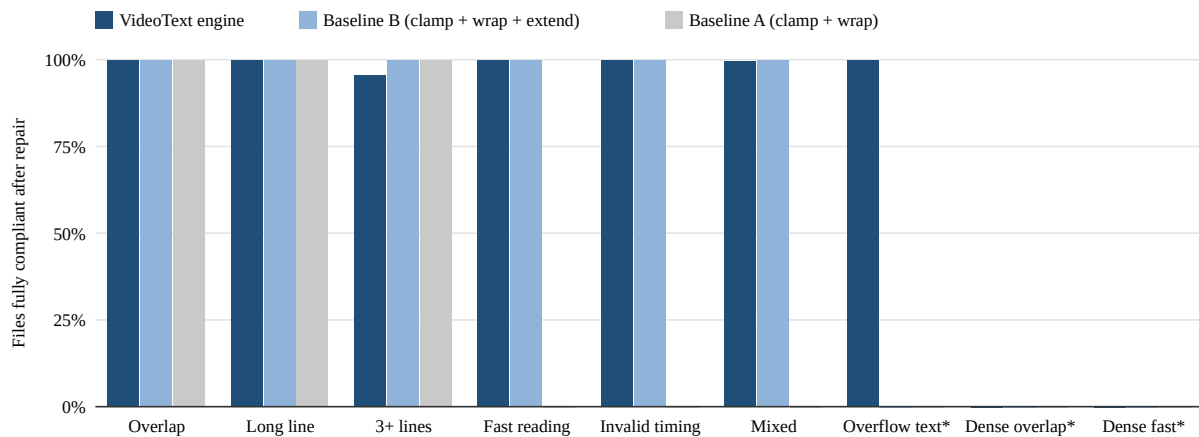
5.1 Detection

The check step reported a finding of the matching type for all 11,597 injected defects, in every condition (recall 100%; Wilson 95% CI 99.97–100%; for each single condition with $n = 1,200$, 99.68–100%). It reported no finding on any of the 12,000 cues in clean files or the 95,206 unmodified cues in defective files, so the false-positive rate was 0 of 107,206.

5.2 Repair

Condition	Engine compliant	Engine residual	Baseline A compliant	A residual	Baseline B compliant	B residual
Clean	200/200	0	200/200	0	200/200	0
Overlap	200/200	0	200/200	0	200/200	0
Long line	200/200	0	200/200	0	200/200	0
3+ lines	191/200	10	200/200	0	200/200	0
Fast reading	200/200	0	0/200	1,200	200/200	0
Invalid timing	200/200	0	0/200	1,072	200/200	0
Mixed	199/200	2	0/200	738	200/200	0
<i>Overflow text*</i>	<i>200/200</i>	<i>0</i>	<i>0/200</i>	<i>1,197</i>	<i>0/200</i>	<i>1,197</i>
<i>Dense overlap*</i>	<i>0/200</i>	<i>1,200</i>	<i>0/200</i>	<i>1,200</i>	<i>0/200</i>	<i>1,200</i>
<i>Dense fast*</i>	<i>0/200</i>	<i>1,200</i>	<i>0/200</i>	<i>1,200</i>	<i>0/200</i>	<i>1,200</i>

Table 2. Files that are compliant after repair, and oracle violations remaining in them. Rows marked * are stress conditions.



* Stress conditions: no room to repair in place. $n = 200$ files per condition, 60 cues per file before injection.

Figure 1. Share of files fully compliant after repair. The engine leads alone on overflow text. Baseline B matches or slightly beats it where there is room to repair. No fixer can make the dense stress conditions compliant without moving cue start times.

Repairable defects (the six non-stress defective conditions: 1,200 files, 8,000 defects). The engine made 1,190 files compliant (99.2%; CI 98.5–99.5%). Baseline A made 600 compliant (50.0%) and Baseline B 1,200 (100%). Baseline A fails on fast reading because it never extends cues, and on invalid timing because its 1 s repair is often too short for the cue's text. Baseline B adds that extension and fixes every case where the neighbouring gap has room. The engine's 12 remaining violations are 11 minimum-duration violations and 1 reading-speed violation. Ten of them come from the engine keeping the author's line breaks (§3.2). A three-line cue whose text would fit on two rewrapped lines is split into two cues at the author's line break, and occasionally one of the two pieces gets less than 1.5 s. Baseline B rewraps the text instead and keeps a single cue. The engine reported every one of the 12 remaining violations to the user (§5.3).

Overflow text. When a cue's text needs more than two 42-character lines, rewrapping cannot fix it. The engine split all 1,197 overflow cues into compliant cues within the original interval (200/200 files). Both baselines wrapped them into three or more lines (0/200).

Dense conditions. In *dense overlap*, the next cue starts 0.3–0.8 s after a cue of more than 30 characters. Clamping the earlier cue leaves a cue at 38 CPS or more, nearly twice the 20 CPS limit. Both baselines clamped anyway. They turned all 1,200 overlaps into 1,200 unreadable cues and reported nothing. The engine refused to create these cues. It kept the 1,200 overlaps, reported each one with a suggested action, and set all 200 files to `needs_review`. In *dense fast*, the gap to the next cue is 0.04 s, so there is no room to extend. No fixer made a change, and only the engine reported the 1,200 cues. Making these files compliant would require moving the next cue's start time or rewriting text, and the engine does neither without being asked (§3.2).

5.3 Honesty of reporting

Measure	Result
Findings marked <code>fixed</code> and confirmed fixed in the export	10,322 / 10,322 (100%)
Open findings confirmed by the oracle on that cue	2,412 / 2,413 (99.96%)
Violations left in the export without any finding or warning	0 / 2,412
Files with status <code>passed</code> that are not compliant (silent passes)	0 / 1,565
Compliant files marked <code>needs_review</code> (conservative)	25 / 1,590
Files with unchanged word sequence	2,000 / 2,000
Cues away from any defect whose timing changed	0 / 85,289
Files that are idempotent (fixing again changes nothing)	2,000 / 2,000

Table 3. Engine reporting and side effects over all 2,000 files. The baselines report nothing, so all 3,597 (A) and 6,607 (B) of their remaining violations are unreported.

The one open finding the oracle did not confirm is a split cue in the 3+ lines condition that is on screen for 1.499 s, against the 1.5 s minimum. The oracle's 1.5 ms rounding tolerance accepted the cue, and the engine, which is stricter, reported it. In that case the engine over-reported. It never under-reported. The 25 compliant files marked for review are files where a repaired end time could not be anchored to the original file, for example after an invalid timestamp, so the engine asks the user to check it against the video.

5.4 Runtime

One full fix call took a median of 1.13 ms per 60-cue file (95th percentile 1.89 ms, maximum 8.4 ms; $n = 2,000$), single-threaded on Node.js 22.22 in a 4-vCPU Intel Xeon 2.1 GHz cloud container. That is roughly 50,000 cues per second, so in practice the cost of the deterministic stage is upload and queueing, not computation.

5.5 The public `check_subtitles` tool

VideoText also offers a free `check_subtitles` tool for AI assistants, served through the Model Context Protocol and a ChatGPT app. It is **not** the engine evaluated above. It is a separate, lightweight checker (`chatgpt-app/src/subtitles.ts`) that reports problems and does not repair them.

Spot check. On 27 October 2026 we sent one benchmark file (`mixed_001.srt`) to the live tool. It reported 8 of the 10 injected defects, at the same cues and with the same types as the engine. It missed both three-line cues. The checker had no rule limiting the number of lines.

Full-corpus measurement and fix. We then ran the checker's code over all 2,000 benchmark files, using the same ground truth and the same matching rule as in §5.1. Before the fix it detected 9,997 of 11,597 defects (86.2%). It detected none of the 1,600 three-line cues and every defect in the other seven classes. We added the two-line rule to the checker, with a unit test, in the same repository revision as this report. With the rule added, the checker detected all 11,597 defects and flagged none of the 107,206 unmodified cues (Table 4).

Public checker version	Defects detected	3+ lines	Other seven classes	False positives
Before the fix (live on 27 Oct 2026)	9,997 / 11,597 (86.2%)	0 / 1,600	9,997 / 9,997	0 / 107,206
With the two-line rule	11,597 / 11,597 (100%)	1,600 / 1,600	9,997 / 9,997	0 / 107,206

Table 4. Detection by the public checker's code on the benchmark corpus (`results/public_checker_parity.json`). These figures measure code, not the live service. The live tool matches the second row only once that version is deployed.

Remaining differences. Two rule differences between the public checker and the engine remain. The corpus does not exercise either of them:

- The public checker has no minimum on-screen duration rule. The engine requires at least 1.5 s for any cue over 20 characters.
- The public checker compares each cue only with the next cue in file order. So in an unsorted file, it can miss an overlap with a later cue that is not adjacent. The engine checks in time order and finds nested cues.

Claims about the public checking tool should therefore cite Table 4 and its date, not the engine results in §5.1–5.4. The fixing tool (`fix_subtitles`) calls the engine itself.

6. Discussion

6.1 What the results show

On defects that have an obvious repair, a careful simple fixer (Baseline B) is enough, and on this corpus it is slightly ahead of the engine. Its advantage comes from ignoring the author's line breaks. The engine does better in two kinds of case: (i) text that does not fit in two lines and must be split, and (ii) cases with no valid in-place repair. In the second kind, the clamp-style fixes common in simple tools hide one defect by creating another. Of the three systems, the engine is the only one whose output status can be trusted. No file it marked passed was non-compliant, and every remaining violation was reported on the cue that has it.

6.2 Limitations and threats to validity

- **Synthetic defects.** Real files contain correlated defects, markup, multiple speakers, VTT positioning, and encoding faults, and none of these were tested. The injected distributions are our own choices.
- **Shared rules.** The oracle is independent code, but it encodes the same thresholds the engine uses. Detection recall therefore measures implementation correctness against the declared rules, not whether those rules are right for every audience or language.
- **Baselines.** The baselines are simple fixers we wrote ourselves. They are not commercial products, and we make no claim about any other vendor's tool.
- **Timing ground truth.** No audio was used. A real fix should be judged against speech timing, and this study cannot measure that. That is why the engine's choice to keep start times fixed cannot be scored here.
- **Language.** The corpus is English text only. CPS limits and line lengths differ for other scripts.
- **Two checkers.** The engine and the public `check_subtitles` tool are separate implementations of the same rules. They differed on one rule when this study ran, and they still differ on the two rules listed in §5.5. Results for one do not transfer automatically to the other.
- **Conflict of interest.** The author develops VideoText. To limit bias, all code, data, and per-file outputs are released, the stress conditions where the engine does not succeed are reported in full, and the conditions where a baseline matches or beats the engine are stated.

6.3 Future work

Planned extensions are: real subtitle files with human-verified corrections; VTT files with positioning; non-English scripts; evaluation of timing against forced alignment of the audio; and inclusion of other subtitle tools that offer an automated or exportable workflow, run on the same released corpus. Separately, the VideoText team is running a transcription-accuracy pilot on LibriSpeech test-clean [4] with an open Whisper baseline [5]. Its results are not results for the VideoText pipeline and are not reported here.

7. Conclusion

Automated subtitle repair is easy where there is room to repair and hard where there is not. The useful question about a fixer is whether its report of success can be believed. In a reproducible benchmark of 2,000 files and 11,597 injected defects, the VideoText subtitle engine detected every defect with no false positives. It repaired 99.2% of the files that could be repaired and split every overflowing cue. Where no valid repair existed, it reported the problem instead of hiding it. Every one of its 10,322 claimed fixes was present in the exported file. We release the benchmark so that these properties can be measured for any subtitle tool.

Data and code availability

The archive that accompanies this report contains the following:

- `scripts/run_benchmark.ts`: the generator, injector, oracle, baselines, and scorer.
- `scripts/fetch_sources.sh`: downloads the source texts.
- `scripts/build_figure.py`: draws Figure 1.
- `corpus.zip`: all 2,000 SRT inputs.
- `results/summary.json` and `results/per_file.csv`: the full results.
- `results/injections.jsonl`: which cue received which defect, for scoring any other checker.
- `scripts/public_checker_parity.mts` and `results/public_checker_parity.json`: the public checker measurement in Table 4.

The engine itself is part of the VideoText product. It can be run on any SRT file through the Fix Subtitles tool at videotext.io, and through the public API and MCP `fix_subtitles` tool. Reproduction instructions are in the archive's `README.md`.

Acknowledgments and AI disclosure

The benchmark code and the first draft of this manuscript were produced with the help of an AI coding assistant (Anthropic Claude). The author designed the study, reviewed the code, checked the numbers against the released result files, and takes responsibility for the content. Source texts are courtesy of Project Gutenberg.

References

1. W3C. *WebVTT: The Web Video Text Tracks Format*. W3C Candidate Recommendation, 2019. <https://www.w3.org/TR/webvtt1/>
2. Netflix. *English (USA) Timed Text Style Guide*. Netflix Partner Help Center. <https://partnerhelp.netflixstudios.com/>
3. Project Gutenberg. eBooks #1342, #11, #84, #2701, #1661. <https://www.gutenberg.org/>
4. V. Panayotov, G. Chen, D. Povey, S. Khudanpur. LibriSpeech: an ASR corpus based on public domain audio books. *Proc. IEEE ICASSP*, 2015, pp. 5206–5210.
5. A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, I. Sutskever. Robust speech recognition via large-scale weak supervision. *Proc. ICML*, 2023.

How to cite: Guntupalli, S. (2026). *Honest Automated Subtitle Repair: A Reproducible Defect-Injection Benchmark of the VideoText Subtitle Quality Engine* (Technical report v1.0). Zenodo. <https://doi.org/10.5281/zenodo.23701131>