

Synthetic Voices as Validation Instruments for Automated Oral-Reading-Fluency Scorers: Method, Demonstration, and Limits

Jasper Vande Moortel, Orion-Solution CommV Version 0.1 — 10 July 2026

Abstract

Validating an automated oral-reading-fluency scorer requires audio whose miscues are known exactly. Real children's recordings, the gold standard, are scarce, privacy-constrained, and expensive to annotate, and their annotations carry their own disagreement. This paper develops the complementary instrument: synthetic speech with planted, position-seeded miscues, giving machine-exact ground truth at negligible marginal cost. We specify the battery protocol (variant taxonomy covering omissions, insertions, substitutions, repetitions, self-corrections, and controlled mispronunciations planted through SSML phoneme overrides), empirically verify the enabling mechanism in Dutch and English against a production speech engine, and demonstrate the method's yield: in a single day and 432 engine calls it exposed systematic miscue-labeling failures in a widely deployed commercial engine, drove a published two-round improvement of an open miscue classifier with identical-data before-and-after evidence, and caught a designer's error in a proposed rule before it shipped. We state the method's limits with equal care: fluent adult synthetic speech validates measurement mechanics, never child-speech accuracy; planted errors are cleaner than real miscues; no Dutch child TTS voice currently exists; and thresholds tuned on synthetic data inherit battery-specific guarantees that must not be mistaken for properties of reading. In the literature pass behind this work no published validation of an ORF scorer against synthetic audio was found, so the method's external validity is itself untested; we register the transfer study directly, running the identical protocol on annotated real child recordings through the same adapter interface and comparing the conclusions the two routes reach.

1. Introduction

Every validation study of an automated oral-reading-fluency (ORF) scorer faces the same question: against what truth? The field's answer has been human judgment over real recordings, either expert scorers rating the same audio (Bolaños, Cole, Ward, Tindal, Hasbrouck, & Schwanenflugel, 2013) or measures derived from human transcripts (Harmsen, van Hout, Cucchiari, & Strik, 2025). That answer is right and insufficient. It is right because children's real reading is the construct. It is insufficient for three practical reasons: annotated child recordings are scarce and privacy-constrained, so most developers cannot obtain them at all;

human annotation itself disagrees, at rates that re-examination sometimes finds worse than corpus documentation suggests (Gao, Tejedor-García, Cucchiaroni, & Strik, 2025); and agreement statistics over aggregate scores can hide systematic classification errors, since a scorer can match human WCPM closely while mislabeling the miscue types underneath.

Synthetic speech offers a complementary instrument. If a text-to-speech system reads a passage with deliberately planted deviations (a skipped word, an inserted one, a duplicated one, a forced mispronunciation) the ground truth is known by construction, exactly, at every position, for the cost of an API call. Validation against machine-known truth catches a class of defect that human-agreement designs cannot even see, and it is repeatable at every rule change for the cost of compute.

In the research pass behind this work, no published validation of an ORF scorer against synthetic audio was located in any language searched; we state this as a search finding rather than a proven absence. The adjacent traditions exist (Section 2) but none combines synthetic voices, planted reading miscues, and ORF-scorer validation. This paper develops that combination as a method:

1. **The battery protocol** (Section 3): variant taxonomy, position seeding, ground-truth construction, the corpus-adaptor interface that lets the identical analysis re-run on real recordings, and a reporting template.
2. **The enabling mechanism, verified** (Section 3.3): controlled mispronunciations planted through SSML phoneme overrides, empirically confirmed against a production engine in both Dutch and English, including the severity behavior that shapes what such plants can test.
3. **A demonstration of yield** (Section 4): what the method found in one day against a production engine and an open classifier, drawn from the two studies reported in full in the fifth paper of this series (Vande Moortel, 2026e).
4. **The limits, stated as carefully as the method** (Section 5), and **a registered transfer-validity study** (Section 6) that will test the method itself against real child audio.

2. Background

Synthetic speech as ASR data. Using TTS output to train or evaluate speech recognizers is established practice in low-resource settings; cross-lingual TTS and voice-conversion augmentation pipelines report usable gains from very little real target speech (Casanova et al., 2023, at secondary confidence). This tradition targets recognizer training, not assessment validation, and none of it involves reading miscues; its relevance here is the demonstrated viability of synthetic audio as a stand-in under stated conditions.

Synthetic child speech. A smaller niche synthesizes child-like voices specifically, typically by transfer-learning adult multi-speaker TTS on child corpora, motivated as much by privacy regulation as by data scarcity (Farooq et al., 2023; Rolland et al., 2025, both at abstract confidence). The niche is effectively English-only: no published Dutch child TTS work was found, and no Dutch child voice exists in the major commercial TTS catalogs (Microsoft, n.d.-e).

Planted errors in pronunciation training. The computer-assisted pronunciation training literature has synthesized errors on purpose: TTS-generated words with correct and incorrect stress patterns to train error detectors (Korzekwa et al., 2022, at secondary confidence), and signal-level augmentation frameworks that blend mispronunciation variants without full re-synthesis (El Kheir et al., 2022). These target isolated adult second-language pronunciation errors, mostly at word level. Passage-level child reading miscues (omissions, repetitions, self-corrections in connected text) are a different object, and no work was found applying planted-error synthesis to them.

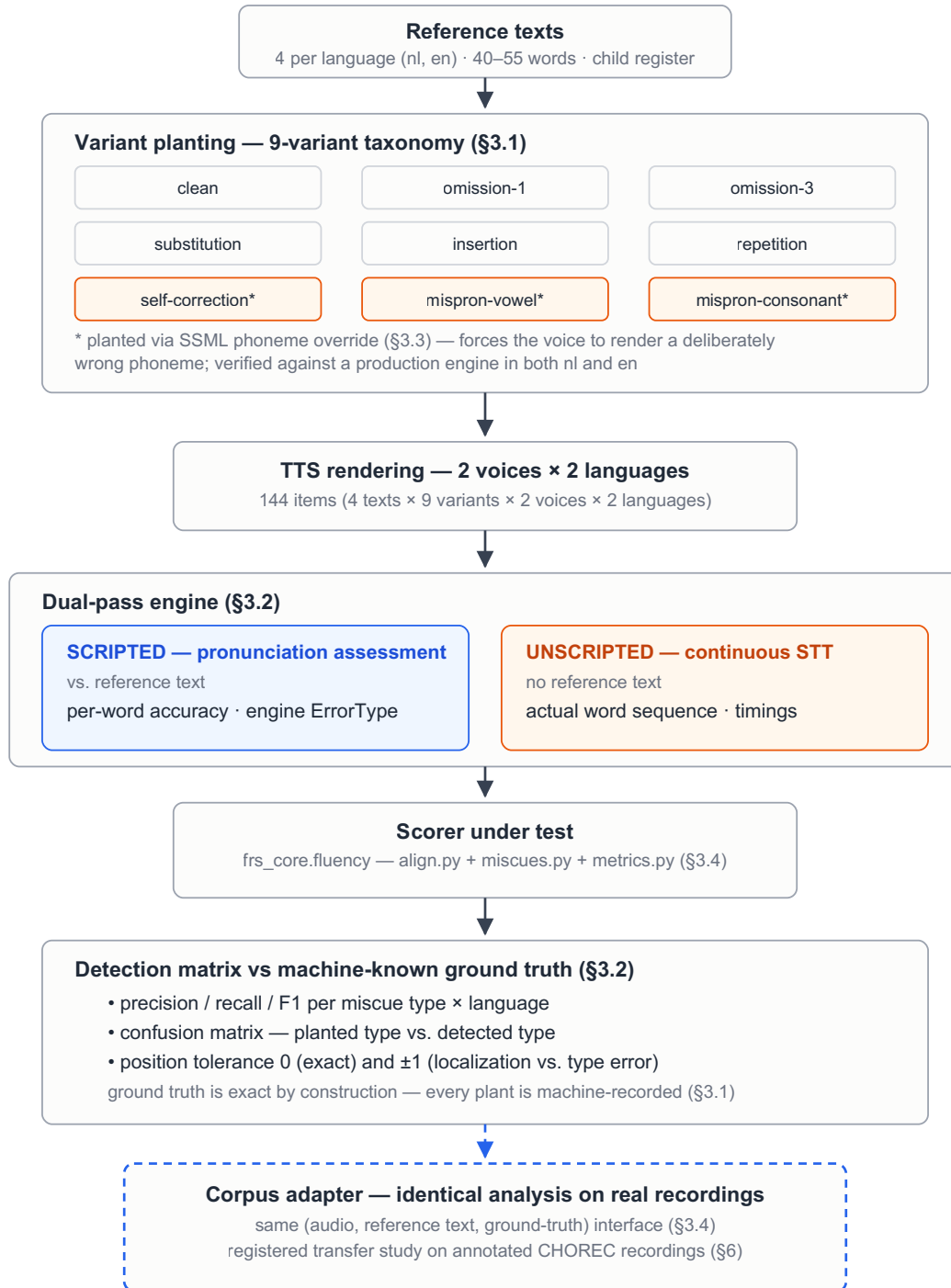
ORF validation practice. The recent Dutch validations, all primary-verified for this series, all use real recordings: against human raters (Molenaar, Tejedor-Garcia, Strik, & Cucchiari, 2023), against human transcripts (Harmsen et al., 2025), and against test-battery criteria at scale (van der Velde et al., 2025). The gap this paper fills sits precisely between these literatures.

3. The method

3.1 Battery construction

A battery is built from reference texts in the target languages (for FRS: four per language, 40 to 80 words, child-appropriate register, varied punctuation), each expanded into a fixed variant set:

- **clean** — the text as written (the false-positive floor);
- **omission-1** and **omission-3** — one and three non-adjacent words removed (candidates are selected positionally and by minimum length, not by part of speech; the FRS generator excludes first and last words and words under three characters);
- **substitution** — one word replaced by a different real word of similar length;
- **insertion** — one extra word inserted;
- **repetition** — one word duplicated;
- **self-correction** — one word rendered once with a wrong phoneme override, immediately followed by its correct form;
- **mispronunciation-vowel** and **mispronunciation-consonant** — one word with a single forced phoneme deviation of each class.

Figure 1 — The synthetic validation battery pipeline

The synthetic generator (top) and the corpus adapter (bottom) both implement the identical item interface, so the analysis re-runs unchanged on real recordings once an annotated corpus is available (§3.4).

Planted positions are drawn once per text with a fixed seed and recorded; the ground truth is therefore exact by construction: every variant carries a machine-readable annotation of which reference position carries which miscue type. Each variant is rendered by at least two voices per language to expose voice effects. For FRS this yields 144 items; the marginal cost of the full battery, rendered and scored twice through a commercial engine, was under one hour of audio in total.

3.2 Ground truth, scoring, and reporting

Detection is scored per miscue type and language: a planted miscue counts as detected when the scorer reports the correct type at the exact planted position, with a secondary ± 1 -position tolerance analysis to separate localization error from type error. The reporting template (Appendix C) requires: precision, recall, and F1 per type and language at both tolerances; the planted-versus-reported type confusions; the clean-variant false-positive inventory reported separately from planted-variant false positives, since the clean variants are the method's false-accusation floor (the FRS reference implementation currently pools these and derives the clean inventory by manual inspection; Appendix C marks it as a required template item precisely because of that gap); and the scorer's suppression or uncertainty counters, so that verdicts a scorer deliberately withheld remain visible. Aggregate-score deviations (WCPM, accuracy against the clean baseline) complete the picture: they should grow with planted-error count, and in the FRS demonstration they did (-2.9 WCPM at one planted error, -5.6 at three; with only two non-trivial points the shape beyond monotone growth is not established).

3.3 The enabling mechanism: forced mispronunciations, verified

Omissions, insertions, repetitions and substitutions are planted by editing the rendered text. Mispronunciations cannot be; they require the voice to say the *right word wrongly*. The mechanism is the SSML phoneme override: wrapping one word in a phoneme element carrying a deliberately altered IPA transcription. Whether commercial TTS honors such overrides, particularly outside English, was an open question (a community bug report suggested Dutch support might be broken).

We verified the mechanism empirically against the Azure engine in both languages (Vande Moortel, 2026e, Study 1). The overrides render: a Dutch word forced from /fɔnt/ to /fiant/ was scored 31 of 100 by the engine's scripted assessment and flagged as a mispronunciation, and the unscripted recognizer independently transcribed it as the *other real word* the altered phonemes spell; the English case scored 4 of 100. Two methodological consequences follow. First, the mechanism works and is controllable at single-phoneme granularity, in both languages, with documented phoneme tables (Microsoft, n.d.-f). Second, the severity behavior matters: a phoneme-level plant severe enough to change the perceived word surfaces in transcripts as a substitution, which is exactly how severe decoding errors behave in real readings, and any scorer's mispronunciation-versus-substitution disambiguation is therefore testable with this mechanism.

3.4 The corpus-adaptor interface

The battery's analysis consumes items through a deliberately minimal interface: an item is (audio, reference text, ground-truth miscue annotations), regardless of origin. A synthetic generator implements it today; a corpus adaptor implements it for annotated real recordings later, and the identical analysis re-runs unchanged. This is not an implementation convenience

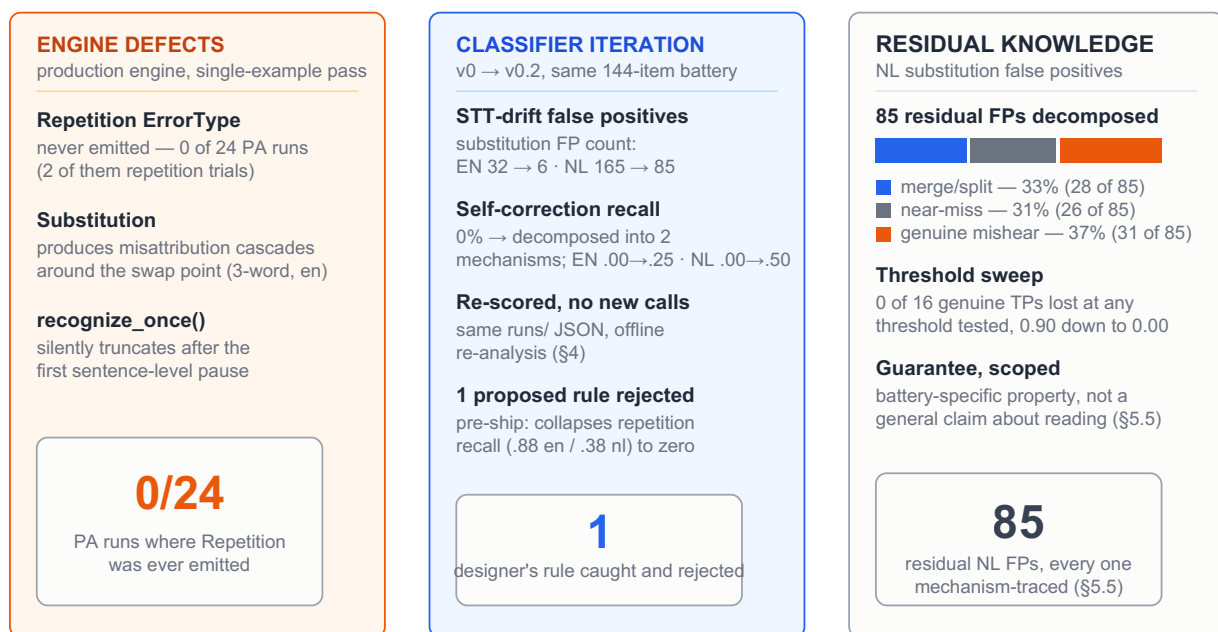
but the method's scientific design: it makes the transfer-validity study of Section 6 a drop-in comparison rather than a new experiment.

4. Demonstration: what the method caught

The method's value is best stated as its yield in the FRS program, reported in full in the fifth paper of this series (Vande Moortel, 2026e) and summarized here as evidence about the *method* (Figure 2).

Figure 2 — What the method caught

144-item synthetic battery, one day, 432 engine calls — three finding classes an aggregate agreement statistic cannot see (§4)



Detection failures are as informative as detections: exact ground truth lets every false positive be traced to a named mechanism rather than absorbed into an aggregate score (§4–5).

Source: experiments/fluency_synthetic_battery/REPORT.md, experiments/fluency_instrument/REPORT.md.

Engine defects invisible to agreement studies. A 12-item pilot battery against the production engine underlying Microsoft Reading Progress established that the engine never emitted its own repetition label in any run (a planted duplicate was labeled an insertion in Dutch and silently absorbed in English), that planted substitutions produced cascading misattributions to unrelated neighboring words, and that single-shot recognition truncates multi-sentence audio silently. None of these facts is visible in an aggregate agreement statistic; all of them shape what any product built on the engine can honestly report.

Classifier defects, and a designer's error. The full 144-item battery scored an open mis-cue classifier and returned per-type precision and recall against exact ground truth. It exposed the recognizer's transcription drift as a false-accusation engine (correctly read words transcribed as close variants and then flagged as substitutions), quantified a total self-correction detection failure and decomposed its mechanisms, and, after a corroboration-rule revision, re-

scored the identical stored responses to give a before-and-after table with no new data collection. In the same iteration the battery rejected a proposed rule outright: a self-correction trigger that re-analysis showed would collapse genuine repetition recall to zero. The method caught the designer's error, not just the code's, before either shipped.

The residual as knowledge. Where detection stayed poor (Dutch substitution precision), the battery's exact ground truth allowed a mechanism-level decomposition of every false positive (recognizer merges, short-word similarity compression, genuine mishears) and a threshold sweep whose safety guarantee could be precisely scoped as battery-specific rather than general, preventing an overfit "fix." The method's failures are as informative as its successes because the truth is exact.

5. Limits

The method validates measurement mechanics. It does not validate performance on children, and the following limits are constitutive, not incidental.

1. **The speech is fluent, adult, and synthetic.** Child speech differs acoustically and prosodically from adult speech in developmentally varying ways, and recognition degrades on it substantially; conclusions about mechanics do not transfer to accuracy claims on children. Any battery result must be labeled as mechanics validation.
2. **Planted errors are discrete and clean.** Real miscues include partial attempts, drawn-out decoding, dialect features, and compounded disfluencies. The battery's error model is a lower bound on difficulty, suitable for falsifying a scorer (a scorer that fails on clean planted errors will fail on real ones) but not for certifying one.
3. **No Dutch child voice exists** in any major TTS catalog (Microsoft, n.d.-e); a Dutch battery necessarily uses adult voices, and pitch manipulation toward child-like registers introduces its own artifacts. The English-only child-TTS niche does not currently change this.
4. **Synthetic-tuned thresholds inherit battery-specific guarantees.** In the FRS demonstration, every planted substitution happened to carry a corroborating engine verdict, so no similarity threshold could cost a true positive on that data; that is a property of the battery, not of reading, and was documented as such rather than exploited. Threshold calibration belongs to real data.
5. **Voice and engine coverage bound generality.** Two voices per language and one engine were used; observed voice effects were minor but real, and the engine-specificity of every finding is inherited by the method's conclusions.

6. The registered transfer-validity study

The method itself has an untested premise: that validation conclusions reached on synthetic audio agree with those reached on real child recordings, where both are available. No study of that agreement exists for ORF scoring, in our search. We register it directly.

Design. Run the identical battery analysis, through the same adapter interface, on miscue-annotated real child recordings (the CHOREC corpus of Flemish children's oral reading, with reading-error and reading-strategy annotation tiers; license requested July 2026 (Cleuren, Duchateau, Ghesquière, & Van hamme, 2008)), and compare three things between the synthetic and real routes: the per-type detection ranking (which miscue types are easy and hard), the false-positive mechanism inventory, and the rule-revision decisions each route would have driven. Agreement on those three would license synthetic batteries as a first-line validation instrument with real-audio confirmation reserved for release gates; disagreement would map exactly where the synthetic route misleads. Annotation reliability itself is treated as a measured quantity in this design, given the documented gap between corpus-reported and re-examined agreement on Dutch miscue annotation (Gao et al., 2025).

Status. Pre-registered here in the Paper 4 tradition of this series (Vande Moortel, 2026a): the design is stated before the data exists, and the comparison will be published whether it flatters the method or not.

7. Conclusion

Synthetic voices with planted miscues give ORF-scorer developers what they otherwise lack: exact ground truth, unlimited repetition, zero privacy exposure, and per-rule-change re-validation at compute cost. In one day the method exposed production-engine failures, drove an evidenced classifier iteration, and caught a design error before it shipped. It is not a substitute for real children's audio and must never be reported as one; it is the instrument you use so that, when the scarce real data arrives, you spend it answering the only question it can answer: how the scorer behaves on children, not whether its mechanics work at all.

References

- Casanova, E., et al. (2023). ASR data augmentation in low-resource settings using cross-lingual multi-speaker TTS and cross-lingual voice conversion. In *Proceedings of INTERSPEECH 2023*. https://www.isca-archive.org/interspeech_2023/casanova23_interspeech.pdf [Full author list under verification.]
- Cleuren, L., Duchateau, J., Ghesquière, P., & Van hamme, H. (2008). Children's Oral Reading Corpus (CHOREC): Description and assessment of annotator agreement. In *Proceedings of the Sixth Interna-*

- tional Conference on Language Resources and Evaluation (LREC 2008)*. European Language Resources Association. <https://aclanthology.org/L08-1205/>
- El Kheir, Y., et al. (2022). SpeechBlender: Speech augmentation framework for mispronunciation data generation [Preprint]. arXiv. <https://arxiv.org/abs/2211.00923> [Full author list under verification.]
- Farooq, et al. (2023). Synthetic speaking children: Why we need them and how to make them. In *Proceedings of SpeD 2023*. <https://arxiv.org/abs/2311.06307> [Full author list and initials under verification.]
- Gao, L., Tejedor-García, C., Cucchiarini, C., & Strik, H. (2025). Rethinking reading miscue annotation protocol: Insights from re-examining Dutch child reading annotation. In *Proceedings of the 10th Workshop on Speech and Language Technology in Education (SLaTE 2025)* (pp. 116–117). https://www.isca-archive.org/slate_2025/gao25_slate.pdf
- Harmsen, W., van Hout, R., Cucchiarini, C., & Strik, H. (2025). Can ASR generate valid measures of child reading fluency? In *Proceedings of INTERSPEECH 2025* (pp. 2395–2399). <https://doi.org/10.21437/Interspeech.2025-306>
- Korzekwa, D., et al. (2022). Computer-assisted pronunciation training: Speech synthesis is almost all you need. *Speech Communication*. <https://arxiv.org/pdf/2207.00774> [Full author list and pagination under verification.]
- Microsoft. (n.d.-e). *Language support: Text-to-speech voices* [Azure AI Speech documentation]. Microsoft Learn. Retrieved July 10, 2026, from <https://learn.microsoft.com/azure/ai-services/speech-service/language-support>
- Microsoft. (n.d.-f). *SSML phonetic sets* [Azure AI Speech documentation]. Microsoft Learn. Retrieved July 10, 2026, from <https://learn.microsoft.com/azure/ai-services/speech-service/speech-ssml-phonetic-sets>
- Molenaar, B., Tejedor-Garcia, C., Strik, H., & Cucchiarini, C. (2023). Automatic assessment of oral reading accuracy for reading diagnostics. In *Proceedings of INTERSPEECH 2023* (pp. 5232–5236). <https://doi.org/10.21437/Interspeech.2023-1681>
- Rolland, et al. (2025). Personalised children's automatic speech recognition using text-to-speech. In *Proceedings of WOCCI 2025*. https://www.isca-archive.org/wocci_2025/rolland25_wocci.pdf [Full author list and initials under verification.]
- van der Velde, M., Harmsen, W., Veldkamp, S., Feskens, R., Keuning, J., & Swart, N. M. (2025). Speech enabled reading fluency assessment: A validation study. *International Journal of Artificial Intelligence in Education*. <https://pmc.ncbi.nlm.nih.gov/articles/PMC12686063/>
- Vande Moortel, J. (2026a). *A living standard: Crowdsourced comparative judgment and batched recalibration of an open reading-difficulty scale (vo.2)* [Working paper]. Orion-Solution. <https://frs.orion-solution.net/research>
- Vande Moortel, J. (2026e). *Measuring oral reading fluency with a dual-pass ASR architecture: Design, instrument characterization, and synthetic validation of the FRS fluency meter (vo.1)* [Working paper]. Orion-Solution. <https://frs.orion-solution.net/research>
- Bolaños, D., Cole, R. A., Ward, W. H., Tindal, G. A., Hasbrouck, J., & Schwanenflugel, P. J. (2013). Human and automated assessment of oral reading fluency. *Journal of Educational Psychology*, 105(4), 1142–1151. <https://doi.org/10.1037/a0031479>

Appendix A. Variant construction rules

Verbatim planting procedure from
`experiments/fluency_synthetic_battery/generator.py` and `texts.py`, given as reimplementable pseudocode. Nothing below is invented; every choice the code makes arbitrarily rather than on principle is flagged as such, per the task that produced this appendix.

A.1 Setup

- `SEED = 42`. For each reference text (`text_id` in `nl1..nl4`, `en1..en4`) a dedicated pseudo-random stream is created as `rng = random.Random(f"{SEED}:{text_id}")`. Same seed and same text corpus reproduce an identical manifest on every run -- the determinism convention `frs_core` uses, extended to this experiment.
- Every text is tokenized twice: `words_raw = text.split()` (whitespace split, punctuation still attached to the neighboring word) and `words_core = tokenize_words(text)` (whitespace split plus leading/trailing punctuation stripped from each token, matching `frs_core.fluency.align.tokenize_reference`'s tokenization exactly). The generator asserts these two lists are the same length before doing anything else; a text containing a standalone punctuation token would break this and misalign every later by-index edit.
- Four "specials" per text are hand-curated in `texts.py`, not seed-drawn: the `mispron_vowel` word, the `mispron_consonant` word, the `selfcorrection` word, and the `substitution` target/replacement pair. `texts.py`'s own module docstring states why: picking "a clearly wrong" IPA override or "a different real word of similar length" needs human judgment a random index pick cannot supply, and the file says so rather than silently presenting these as randomized.
- These four specials' word-indices in `words_core` form a reserved set (`_reserved_indices`), subtracted from the pool the seed-drawn ("structural") variants can select from.

A.2 Candidate pool for the seed-drawn (structural) variants

```

candidates = [
    i for i in range(1, len(words_core) - 1)    # not the first or last word
    if len(words_core[i]) >= 3                  # MIN_CANDIDATE_LEN
    and i not in reserved                       # not already a "special"
]

```

Arbitrary choice, stated plainly: the paper body (Section 3.1) and the code's own comments call these "content words," but the filter above is not a part-of-speech or lexical-content

test -- it is purely positional (excludes only the first and last token) and length-based (3+ characters). A short function word of three or more letters (Dutch "een," "het"; English "and," "was") is eligible whenever it is not otherwise reserved. `MIN_CANDIDATE_LEN = 3` and the first/last-word exclusion are implementation guards against alignment edge-effects at sentence boundaries, not a linguistic content-word definition. The generator makes this trade-off without a part-of-speech tagger and does not claim otherwise anywhere in the source; the paper's own wording ("content words," Section 3.1) is looser than what the code enforces.

A.3 Draw sequence (per text, shared across both voices)

All four structural variants for a text are drawn from the same `rng` stream, in this fixed order, once per text -- not once per voice. Both voices for a text therefore share identical planted positions; only the audio, and hence the STT/PA behavior, differs between voices.

1. **omission-1 position:** `om1_idx = rng.choice(candidates)`.
2. **omission-3 positions:** up to 500 attempts of `rng.sample(candidates, 3)`, sorted, accepted the first time every adjacent pair in the sorted triple differs by 2 or more indices (no two of the three removed words are adjacent). Raises a hard error if 500 attempts find no valid triple -- never triggered on this 4-text/2-language corpus.
3. **repetition position:** `rep_idx = rng.choice(candidates)` -- a fresh draw from the same pool. The code does not exclude indices already drawn for omission or elsewhere, so `rep_idx` can coincide with `om1_idx` or a member of `om3_idx`s.
4. **insertion position and word:** `ins_idx = rng.choice(candidates)`, then `ins_word = rng.choice(insert_words)`, where `insert_words` is a fixed, hand-picked, four-item list per language (Dutch: hee, erg, best, echt; English: very, really, quite, so) -- an arbitrary small editorial bank, not derived from any corpus or frequency list.

A.4 Per-variant text construction

- **omission-1 / omission-3:** operate on `words_raw`, deleting the target index/indices in descending order (highest index first) so earlier deletions do not shift the indices still to be removed, then rejoin with single spaces.
- **substitution:** not a single-word swap. The generator replaces a hand-curated multi-word anchor phrase with its anchor-replacement phrase via one `text.replace(anchor, anchor_replacement, 1)` call -- e.g. `nl1` replaces the two-word phrase "grijze kat" with "grijze hond," not "kat" with "hond" in isolation, which preserves surrounding article/adjective agreement. The recorded ground-truth `ref_index` is the tokenized-word index of the specials' target word (e.g. "kat") in the ORIGINAL text, not the replacement.
- **insertion:** `ins_word` is spliced into `words_raw` immediately after the word at `ins_idx` (between `words_raw[ins_idx]` and `words_raw[ins_idx + 1]`), then

rejoined. The recorded `ref_index` is `ins_idx` -- the reference-text position the inserted word now trails.

- **repetition:** the word at `rep_idx` in `words_raw` is duplicated immediately before itself (`[..., w, w, ...]`), then rejoined. `ref_index = rep_idx`.
- **self-correction:** the text is not split by index for this variant. The hand-curated special word's literal surface string is duplicated in place via `text.replace(word, f"{word}{word}", 1)` -- the first occurrence becomes two consecutive copies of the same word. The SSML builder (Appendix B) is then asked to wrap `word` with its `wrong_ipa`; because its match regex (`\bword\b`) takes only the first (leftmost) match with `count=1`, it is the FIRST of the two now-adjacent copies that receives the wrong-phoneme override, and the second, correctly rendered copy is left as plain text. This wrong-then-correct ordering is a side effect of "leftmost regex match on now-duplicated text," not a separately coded rule; it only works because the duplication happens before the SSML step runs. `ref_index` is the special word's tokenized index in the original, single-occurrence text.
- **mispron_vowel / mispron_consonant:** the reference text is left completely unedited (`variant_text = text`, identical to the `clean` variant's text). Only the SSML wraps the relevant hand-curated word with its `wrong_ipa` (Appendix B). Two independent hand-curated words per text carry these two variants -- one designated a "vowel" target, one a "consonant" target, by the author's own classification in `texts.py`, not by an automated phonetic analysis.

A.5 Item count

4 texts x 9 variants (clean, omission1, omission3, substitution, insertion, repetition, selfcorrection, mispron_vowel, mispron_consonant) x 2 voices x 2 languages = 144 items, asserted by `generator.py`'s own `main()` before writing `manifest.json`.

Appendix B. The SSML phoneme-override mechanism

B.1 The mechanism, verbatim

Both `experiments/fluency_synthetic_battery/generator.py` and the earlier pilot's `experiments/fluency_instrument/reference_texts.py` implement the identical `build_ssml()` pattern:

```
def build_ssml(locale, voice, text, mispron_word=None, mispron_ipa=None):
    body = text
    if mispron_word and mispron_ipa:
        pattern = r"\b" + re.escape(mispron_word) + r"\b"
        replacement = f'<phoneme alphabet="ipa" ph="{mispron_ipa}">{mispron_word}</phoneme>'
        body, n = re.subn(pattern, replacement, text, count=1)
        if n != 1:
            raise ValueError(...) # exactly one match required
    return (
        '<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" '
        f'xml:lang="{locale}"><voice name="{voice}">{body}</voice></speak>'
    )
```

Only the first whole-word regex match is wrapped (`count=1`); the function raises rather than silently guessing if the target word does not appear exactly once in the base text -- an explicit correctness guard. The self-correction variant (Appendix A.4) deliberately exploits this "first match wins" behavior after duplicating the word itself.

B.2 Verbatim example markup -- one Dutch, one English planted mispronunciation

These are the two cases the pilot instrument (`experiments/fluency_instrument`) empirically verified against the production Azure engine, and that Section 3.3 of this paper cites (Vande Moortel, 2026e, Study 1). Reconstructed directly from `reference_texts.py`'s `NL_TEXT / EN_TEXT` plus `NL_MISPRON_WORD="hond"`, `NL_MISPRON_IPA="ɦant"`, `EN_MISPRON_WORD="dog"`, `EN_MISPRON_IPA="di:g"`:

Dutch (`nl-NL`, voice `nl-NL-FennaNeural`):

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="nl-NL"><voice
name="nl-NL-FennaNeural">De kleine <phoneme alphabet="ipa" ph="ɦant">hond</phoneme> rende
vrolijk door het park. Hij zag een vogel, een kat en een eekhoorn tussen de bomen. Plots begon
het hard te regenen, dus hij rende snel terug naar huis. Zijn vacht was helemaal nat, maar hij
was toch heel erg blij.</voice></speak>
```

English (`en-US`, voice `en-US-JennyNeural`):

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-US"><voice
name="en-US-JennyNeural">The little <phoneme alphabet="ipa" ph="di:g">dog</phoneme> ran
happily through the park. He saw a bird, a cat, and a squirrel between the trees. Suddenly it
started raining hard, so he ran quickly back home. His fur was completely wet, but he was
still very happy.</voice></speak>
```

B.3 IPA alteration rules

Every override in the corpus (`texts.py` 's `SPECIALS` dict, 8 texts x 2 targets each, plus the two self-correction words and the pilot's two words) follows the same hand-applied method, documented in `texts.py` 's own module docstring as "best-effort... not independently verified by listening" -- the battery's job is to measure whether the engine flags the override, not to guarantee the override sounds wrong to a human ear:

- **Vowel-target words:** swap the word's stressed/core vowel for a distinctly different vowel quality, never a subtle allophonic shift. Examples in the corpus: hond /*hɔnt*/->/*hant*/ (short o to short a), hek /*hɛk*/->/*hiak*/, sla /*sla*/->/*slo*/, dog "dawg" /*dɔ:g*/->/*di:g*/ (long o to long e, spelled "deeg"), fence /*fɛns*/->/*fi:ns*/, lettuce /*'lɛtɪs*/->/*'li:tɪs*/.
- **Consonant-target words:** swap only the word's initial/onset consonant for an unrelated consonant, leaving the rest of the word's phonology unchanged. Examples: vond /*vɔnt*/->/*bɔnt*/ (v to b), wortels /*'vɔrtɛls*/->/*'bɔrtɛls*/, veren /*'ve:rɛn*/->/*'be:rɛn*/, bushes /*'bʊʃɪz*/->/*'dʊʃɪz*/ (b to d), carrots /*'kærəts*/->/*'tærəts*/ (k to t), feathers /*'fɛðərz*/->/*'sɛðərz*/ (f to s).
- **Self-correction words** use the same "clearly wrong" method applied to a whole word (both vowel and consonant positions can shift), e.g. meteen forced to /*ma:'ta:n*/, Luckily forced to /*'lʌkɪli*/.
- No systematic rule (minimal-pair distance, a phoneme-confusability matrix, etc.) selects these substitutions; each is a one-off human judgment call recorded directly in `SPECIALS` , and the file documents this rather than presenting it as a principled algorithm.

B.4 Vendor phoneme table pointers

The `alphabet="ipa"` attribute and the accepted IPA symbol set per Azure locale are documented at Microsoft (n.d.-f), "SSML phonetic sets" -- already cited in this paper's reference list, and the same source both `generator.py` 's and `reference_texts.py` 's overrides were built against. No separate phoneme table exists in the code itself, only inline comments recording correct-vs-wrong IPA per word.

B.5 Observed severity behavior (cross-checked against runs/*.json, not summary prose alone)

	Dutch (<code>nl_phoneme_mispron</code>)	English (<code>en_phoneme_mispron</code>)
Target word	hond	dog
Forced IPA	hant	di:g

	Dutch (nl_phoneme_mispron)	English (en_phoneme_mispron)
Scripted PA AccuracyScore (runs/ nl_phoneme_mispron_pa.json / runs/ en_phoneme_mispron_pa.json)	31.0	4.0
Scripted PA ErrorType	Mispronunciation	Mispronunciation
Unscripted continuous-STT transcript (runs/ *_stt.json)	"De kleine hand rende vrolijk door het park."	"The little D ran happily through the park."

- Dutch: the engine's own transcript substitutes a different, real Dutch word ("hand," not "hond") -- the override is audible enough to flip recognition to an unrelated real lexical item, exactly as Section 3.3 states.
- English: the engine's transcript does not produce another real word at all -- it collapses "dog" to the single letter "D," dropping the rest of the word. This is a truncation/degeneration, not a real-word confusion. Section 3.3's claim that the recognizer "transcribed it as the other real word the altered phonemes spell" is written about the Dutch case specifically and is accurate for it; the paper does not make that claim about the English case, and the English case would not support it if it were. Worth stating explicitly here since a reader skimming the sentence could conflate the two languages' STT behavior.
- The severity gap (31 vs. 4) is an engine-internal outcome, not a graded input the method controls: both overrides were built by the identical "clearly different vowel quality" hand-curation method (Appendix B.3); nothing in the generation procedure predicts which language's engine will score an override closer to zero.

Appendix C. Reporting template

Column structures below are read directly off `analyze_results.py`'s `build_results()` function -- the function that actually produces `results.json` / `results_v02.json` -- not off the paper body's description of what a battery "should" report. Where the two disagree, the disagreement is stated rather than smoothed over. One example row per table is filled in from the real FRS v0.2 run, labeled explicitly as example data.

C.1 Precision / recall / F1 per miscue type x language x tolerance

Source field: `precision_recall_table` (list of `{type, lang, tolerance, tp, fn, fp, precision, recall, f1}`). Computed once per tolerance value, `TOLERANCES = [0, 1]`.

Type	Lang	Tolerance	TP	FN	FP	Precision	Recall	F1
<i>(type)</i>	<i>(lang)</i>	<i>(0 or 1)</i>						

[EXAMPLE -- vo.2, real FRS run]

Type	Lang	Tolerance	TP	FN	FP	Precision	Recall	F1
omission	nl	0	29	3	2	0.94	0.91	0.92

C.2 Planted-vs-reported confusion matrix

Source field: `top_confusions` -- a `Counter` of `(planted_type, detected_type)` pairs, same-position matches at tolerance 0 only, reported as `.most_common(15)`. This is not a full type x type matrix and is not split by language: it is the 15 most frequent observed confusions pooled across nl and en. A true confusion matrix (every planted type x every detected type, per language) would need to be derived from `per_item` directly; the script itself only ever emits the sparse top-15 list.

Planted type	Detected as	Count
<i>(type)</i>	<i>(type)</i>	

[EXAMPLE -- vo.2, real FRS run, top confusion]

Planted type	Detected as	Count
omission	substitution	4

(The identical table's top row in the vo run, before STT-drift suppression, was `mispronunciation -> substitution`, count 8 -- included here only to note that the template's "top" row is not stable across classifier versions, not as a second example row.)

C.3 Clean-variant false-positive inventory

This is the one item the paper body's Section 3.2 lists as a reporting-template requirement that `analyze_results.py` does not actually compute as a distinct output. Clean-item detections are not tagged or separated anywhere in `build_results()`; they are folded, indistinguishably from non-clean items' spurious detections, into the general `precision_recall_table` FP counts (a clean item has an empty `planted` list, so every detected event on it becomes an unmatched, counted FP for its type and language). The per-clean-item

breakdown REPORT.md's "root-cause spot check" narrative uses was done by hand, once, against one item, not by a reusable script path.

A reusable version of this table is derivable from the `per_item` list (filter `variant == "clean"`, list every `detected` tuple) but does not exist as a script output today.

Item ID	Lang	Voice	Detected ref-index	Detected type
<i>(item_id)</i>				

[EXAMPLE -- vo, results.json, item n11_clean_n1-NL-FennaNeural, zero planted errors]

Item ID	Lang	Voice	Detected ref-index	Detected type
n11_clean_n1-NL-FennaNeural	nl	nl-NL-FennaNeural	0	substitution
n11_clean_n1-NL-FennaNeural	nl	nl-NL-FennaNeural	8	substitution

(These are the "Sara"->"Sarah" and "speelde"->"speelden" STT transcription-drift false positives Section 4's Demonstration and REPORT.md's root-cause spot check both discuss.)

C.4 Suppression / uncertainty counters

Source fields: `boundary_suppressed_distribution` (vo, unchanged in vo.2 -- the `flu_boundary_suppressed_count` boundary-noise guard) and `stt_drift_suppressed_distribution` (vo.2 only -- `flu_stt_drift_suppressed_count`, absent as a key before the vo.2 classifier rules existed).

Counter	Count value	N items (v0)	N items (v0.2)
<i>(counter name)</i>			

[EXAMPLE -- real FRS run]

Counter	Count value	N items (v0)	N items (v0.2)
flu_boundary_suppressed_count	0	133	133
flu_boundary_suppressed_count	1	11	11
flu_stt_drift_suppressed_count	0	-- (key absent pre-v0.2)	63
flu_stt_drift_suppressed_count	1	--	35

C.5 Aggregate WCPM / accuracy-pct deviation vs. planted-error count

Source fields: `deviation_by_error_count` (keyed by integer planted-error count) and `deviation_by_variant` (keyed by variant name), each `{n_items, mean_wcpm_dev, mean_accuracy_dev}`, computed against each item's own `(text_id, voice)` clean baseline.

By planted-error count:

Planted errors (n)	N items	Mean WCPM deviation	Mean accuracy-pct deviation
<i>(n)</i>			

[EXAMPLE -- vo, real FRS run]

Planted errors (n)	N items	Mean WCPM deviation	Mean accuracy-pct deviation
1	112	-2.85	-1.47
3	16	-5.64	-7.89

By variant:

Variant	N items	Mean WCPM deviation	Mean accuracy-pct deviation
<i>(variant)</i>			

[EXAMPLE -- vo, real FRS run]

Variant	N items	Mean WCPM deviation	Mean accuracy-pct deviation
omission1	16	0.74	-2.93

C.6 Reuse note

Every table above is produced twice by design: once via `main()` -> `results.json` (classifier defaults at the time of scoring) and once via `main_v02()` -> `results_v02.json` (re-analysis of the SAME captured `runs/` JSON through updated `frs_core.fluency` defaults, no new API calls). A future adopter of this template gets a before/after comparison for free by re-running `analyze_results.py --v02` after any classifier change, per the corpus-adapter design in Section 3.4 -- the identical column structures apply unchanged to a real-recording adapter's output (e.g. CHOREC, Section 6) once one exists.