```yaml
url: "https://seimei.do/en/blog/stable-ts-faster-whisper-local-transcription-comparison/"
title: "stable-ts vs Faster-Whisper: picking a local Japanese transcription backend by
  ↪measurement"
description: "Choosing the transcription backend for local lecture/webinar notes by measuring
  ↪processing time, timestamp deltas, and error patterns on a Mac CPU, instead of trusting
  ↪CUDA-based published benchmarks."
pubDate: 2026-07-11
tags: ["transcription", "whisper", "stable-ts", "faster-whisper", "local-ai"]
series: "constella-local-transcription"
related: ["smartphone-telephoto-taper-measurement"]
graphLabel: "stable-ts vs Faster-Whisper"
graphWeight: 1
ogImage: "/blog/stable-ts-faster-whisper-local-transcription-comparison/og.jpg"
lang: en
```


https://seimei.do/en/blog/stable-ts-faster-whisper-local-transcription-comparison/

Goal

Transcribing webinar and lecture recordings locally, without sending audio to any external API, is a routine need here. The local default backend of my transcription pipeline (a home-built video-intake and transcription tool) is stable-ts (stable_whisper), but published benchmarks keep saying Faster-Whisper is faster and more accurate. Most of those benchmarks assume CUDA on NVIDIA hardware, though, and there is no guarantee they transfer to the CPU path on my Mac. So I measured both on the same audio set and decided where the operational default should live.

If you only want the conclusion, jump to Verdict.

Same policy as the taper angle measurement: decide by measuring on your own hardware, not by published numbers.

Conditions

The shared audio set has three pieces:

- Smoke test: English synthetic speech (macOS `say`, 4.011 s, converted to 16 kHz mono)
- Japanese benchmark: Japanese synthetic speech (macOS `say -v Kyoko`, 9.527 s; three cases: clean, pink-noise-mixed at 10 dB SNR, and a variant with the company name written in hiragana to force the correct reading)
- Production-like parity check: a 60-second slice of real lecture audio (300 – 360 s region of an existing recording, 16 kHz mono, 1 ch)

For reference, this is the Japanese benchmark audio (`say -v Kyoko`) converted into what Whisper-family

backends actually receive.

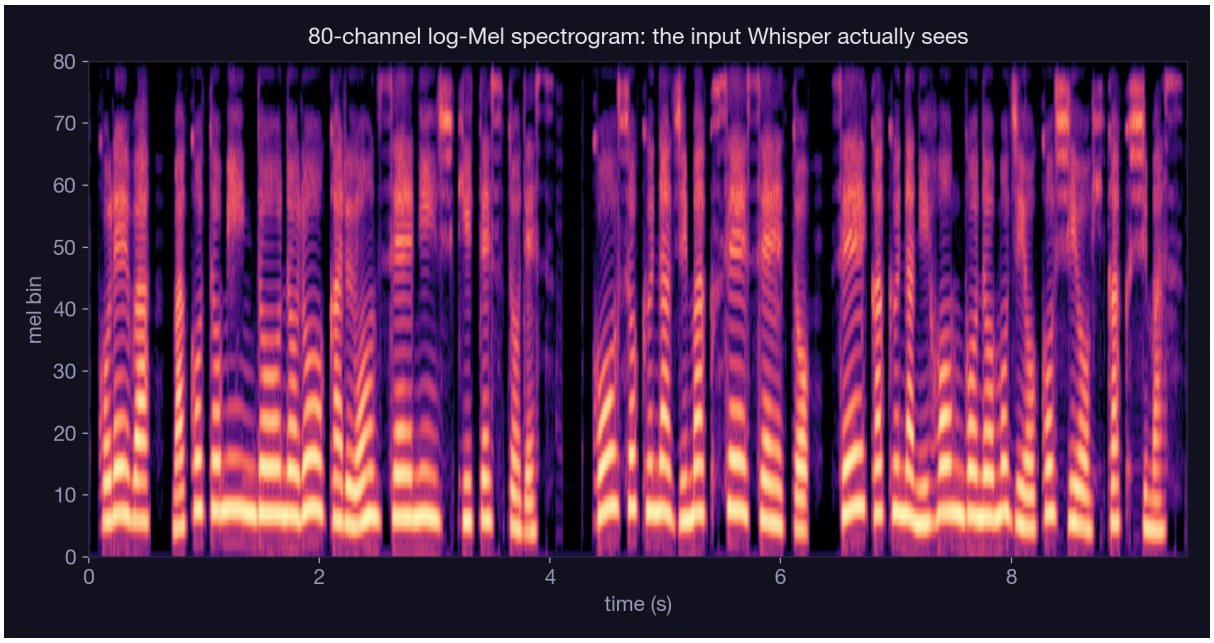

図 1　80-channel log-Mel spectrogram of the 9.53-second Japanese synthetic audio, the actual input representation for Whisper-family backends

Environment and versions (pinned at measurement time):

- Machine: Mac (Apple Silicon, M3 Max), CPU path throughout; no GPU / CUDA
- Python 3.11.6 (the pipeline' s dedicated virtualenv)
- `stable-ts` 2.19.1 / `openai-whisper` 20250625 (the pipeline' s lock pins)
- `faster-whisper` 1.2.1 during the smoke test (isolated venv), then 1.1.1 + `ctranslate2` 4.5.0 when adopted into the pipeline (the Japanese benchmark in this article ran on the adopted 1.1.1)
- Faster-Whisper model: `Systran/faster-whisper-large-v2` (CTranslate2 conversion, snapshot `f0fe8156`), `compute_type=int8`, `local_files_only=True`, offline flags (`HF_HUB_OFFLINE=1`, `TRANSFORMERS_OFFLINE=1`)
- stable-ts models: `base` / `large` / `large-v3-turbo` (local cache under `~/.cache/whisper/`)
- Measured model file sizes: `base.pt` 145,262,807 bytes, `large-v3-turbo.pt` 1,617,941,637 bytes, faster-whisper large-v2 `model.bin` 3,086,912,962 bytes

Metrics

- **Processing time**: model load time and transcription time measured separately
- **Timestamp accuracy**: max start/end delta between the two backends' segment boundaries; for subtitles and span references this matters more than raw speed
- **Error patterns**: normalized similarity against reference text, plus what exactly went wrong

(proper nouns in particular)

- **Output shape**: agreement in segment count, word count, and subtitle layout block count

Hallucination (boilerplate repetition over silence and the like) was not measured as an independent automated metric. A later re-inspection of both backends' output on the real-domain 60-second slice found zero consecutive duplicate segments on either backend, zero short in-segment string repetitions (3+ consecutive repeats), and zero cases of an implausible sentence appearing right after a silence gap of 3 s or longer.

Results

Japanese synthetic benchmark

9.527 s, clean/noise-mixed, with reference text.

Path	Load	Transcribe	Similarity	Notes
stable_whisper `base` (clean)	0.471s	1.518s	0.7957	fast; proper nouns and technical terms break
stable_whisper `base` (noise)	0.471s	0.427s	0.7872	same
stable_whisper `large-v3-turbo` (clean)	3.509s	1.943s	0.8958	good Japanese
stable_whisper `large-v3-turbo` (noise)	3.509s	1.745s	0.9149	good Japanese
Faster-Whisper `large-v2` direct (clean)	1.882s	8.665s	0.8958	text on par with turbo
Faster-Whisper `large-v2` direct (noise)	1.882s	8.585s	0.9149	text on par with turbo

The similarities run low across the board because, as the next section shows, they include the reading error made on the speech-synthesis side.

Proper nouns

The reference sentence for the Japanese benchmark is:

これは、誠明堂のローカル音声認識テストです。高速な字幕生成と、正確な日本語の文字起こしを比較します。

Re-measuring with the real company name in the sentence revealed that proper nouns break at two separate layers.

First, the speech-synthesis (TTS) layer breaks. macOS `say -v Kyoko` misreads the kanji 誠明堂 as "makoto-mindō" , and every backend faithfully transcribed that misreading (large-v3-turbo wrote「マコトミンドウ」, Faster-Whisper「まことみんどう」, and so on). That is why the similarities in the table above run low: the company name was already broken in the input audio, before ASR ever saw it.

Second, even with the correct sound, the recognition (ASR) layer breaks. In the variant where the name was written in hiragana to force the correct reading, base wrote 精明堂, large-v3-turbo wrote 声明道, and Faster-Whisper wrote 声明堂: three different kanji guesses, none reaching 誠明堂. Which kanji to assign to that sound is information the audio does not carry. The English smoke test behaved the same way: "Seimeido" came back as "CMEED" on both Faster-Whisper paths and "CME DAO" on stable_whisper `base`.

Proper-noun spelling, in other words, is not a problem you solve by choosing a backend, on either the TTS or the ASR side. It belongs to the dictionary and post-processing (cleansing) layer, so we exclude it from the backend verdict.

Real-domain lecture audio, 60-second slice (the key test)

The closest condition to production.

Metric	stable-whisper	faster-whisper
Elapsed time	14.407s	117.599s
Segments	15	14
Words	291	297
Layout blocks	16	15

Normalized text similarity was 0.97318 (not identical), max timestamp delta start 6.08s / end 6.22s. **On this CPU path Faster-Whisper was roughly 8x slower**, with text and timing close but not the same.

Verdict

- **The operational default is stable-ts (stable_whisper) + `large-v3-turbo`**. On a Mac CPU it has the best balance of speed and Japanese quality, with a large quality gain over `base`. About 14 s for 60 s of real audio is fine for daily use.

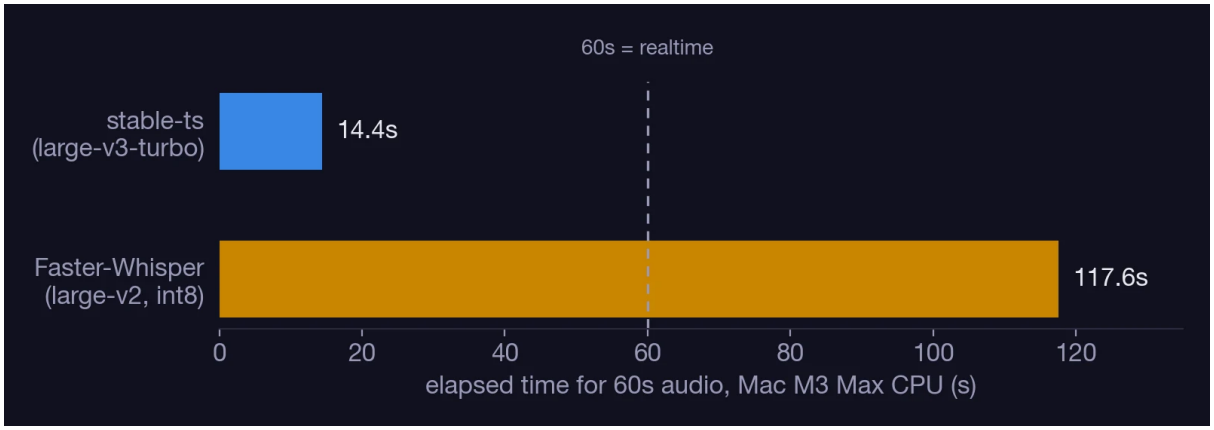

図 2　Elapsed time comparison for the 60-second real-domain audio: stable-ts 14.4s vs Faster-Whisper 117.6s, with a 60s realtime reference line

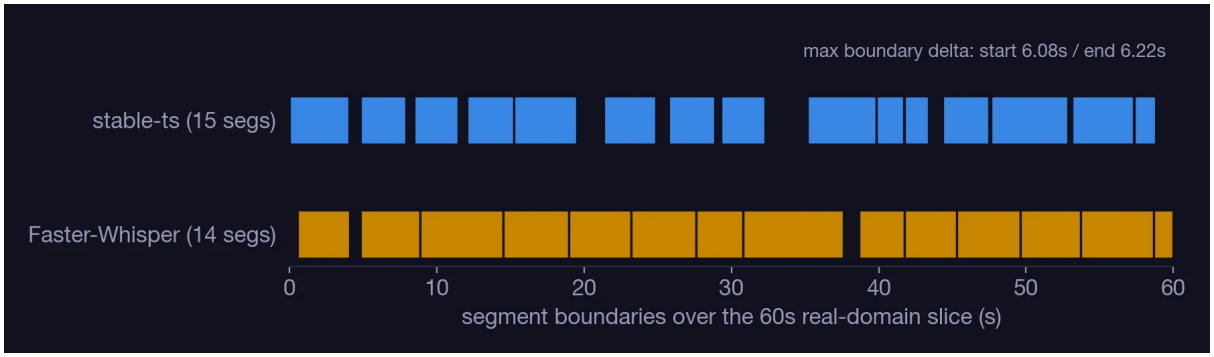

図 3　Segment boundaries of both backends laid out on the 60-second timeline: stable-ts 15 segments, Faster-Whisper 14 segments

- **Faster-Whisper does not become the default.** The published "up to 4x faster" assumes CUDA; on this Mac' s CPU + int8 path it was roughly 8x slower instead. As a second lineage with different behavior, it stays available as an **opt-in backend for comparison and regression checks**.
- The default would only change after re-measurement under GPU or a different runtime, or a clear quality win on representative real-domain audio.

Reproduction

Minimal reproduction commands (CPU, local cache, offline):

```
# stable-ts (default path)
import stable_whisper
model = stable_whisper.load_model("large-v3-turbo", device="cpu")
result = model.transcribe("audio_16k_mono.wav", language="ja")
```

```python
# Faster-Whisper (comparison path, local snapshot, offline)
from faster_whisper import WhisperModel
model = WhisperModel(
    "<local-hf-snapshot-path>/models--Systran--faster-whisper-large-v2/snapshots/<snapshot>",
    device="cpu", compute_type="int8", local_files_only=True,
)
segments, info = model.transcribe("audio_16k_mono.wav", language="ja")
```

For comparisons, record wall time (load and transcribe separately), text, segment/word counts, and segment start/end deltas as JSON. Setting `HF_HUB_OFFLINE=1` / `TRANSFORMERS_OFFLINE=1` pins the reproduction conditions.

One more path exists: stable-ts itself can take Faster-Whisper as its inference backend via `stable_whisper.load_faster_whisper(...)`. That path also worked in the English smoke test (load 1.693 s / transcribe 6.653 s). But `refine()`, stable-ts' s signature feature, is not implemented on that path, so "Faster-Whisper speed plus stable-ts post-processing" is not currently on the table. The real choice is a three-way one: (1) stable-ts alone, (2) Faster-Whisper alone, (3) the stable-ts + Faster-Whisper integration. The operational default (1) was chosen out of those three.