```yaml
url: "https://seimei.do/zh/blog/stable-ts-faster-whisper-local-transcription-comparison/"
title: "stable-ts 与 Faster-Whisper 对比实测：用实测数据选择本地日语转写后端"
description: "为讲座/网络研讨会的本地转写选择后端：不依赖基于 CUDA 的公开基准，而是在 Mac CPU 上实测处理时间、时间戳偏差和误识别模式后做出选择的记录。"
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: zh
```

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

目的

把网络研讨会和讲座录像在本地转写成参考笔记、不把音频发送给任何外部 API，是这里的日常需求。我的转写工具（自制的视频摄取与转写管线）本地默认后端是 stable-ts（stable_whisper），但公开基准反复声称 Faster-Whisper 更快更准。问题是那些基准大多以 CUDA / NVIDIA 为前提，不能保证适用于我这台 Mac 的 CPU 路径。于是在同一套音源上实测两者，决定运行默认值放在哪边。

只想看结论的话，请跳转到取舍。

和锥度角测量一样：不信公开数值，用自己手上的实测来决定。

条件

同一套音源共三种：

- 冒烟测试用：英语合成语音（macOS `say`，4.011 秒，转 16 kHz 单声道）
- 日语基准用：日语合成语音（macOS `say -v Kyoko`，9.527 秒。三种情况：干净版、混入粉红噪声版（10 dB SNR）、以及把公司名写成平假名以强制正确读音的变体）
- 接近实际运行的 parity 校验用：真实讲座音频的 60 秒切片（既有录像的 300 - 360 秒区间，16 kHz 单声道，1 ch）

作为参考，下面是把日语基准音源（`say -v Kyoko`）转换成 Whisper 系后端实际接收的形式。

运行环境与版本（实测时的固定值）：

- 机器：Mac（Apple Silicon, M3 Max），全程 CPU 路径，未使用 GPU / CUDA
- Python 3.11.6（管线专用虚拟环境）
- `stable-ts` 2.19.1 / `openai-whisper` 20250625（管线 lock 固定值）
- `faster-whisper` 冒烟时为 1.2.1（隔离 venv），之后正式并入管线时为 1.1.1 + `ctranslate2` 4.5.0

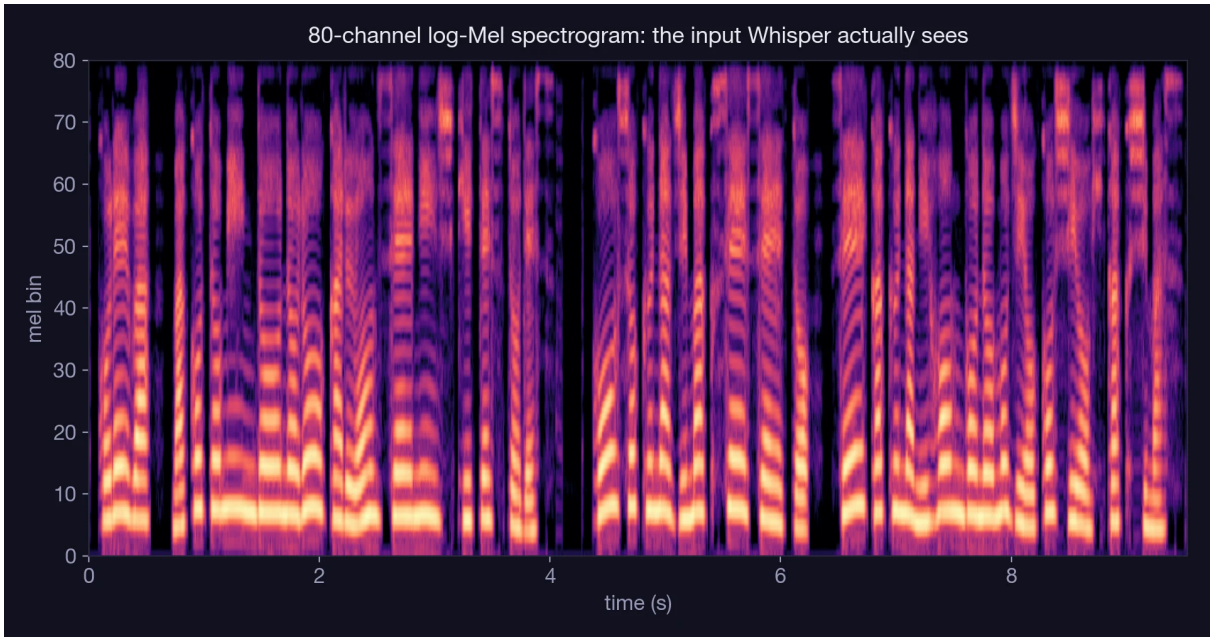

图 1　9.53 秒日语合成语音的 80 通道 log-Mel 频谱图，即 Whisper 系后端的实际输入表示

- Faster-Whisper 模型：`Systran/faster-whisper-large-v2`（CTranslate2 转换，snapshot `f0fe8156`），`compute_type=int8`，`local_files_only=True`，离线开关（`HF_HUB_OFFLINE=1`，`TRANSFORMERS_OFFLINE=1`）
- stable-ts 侧模型：`base` / `large` / `large-v3-turbo`（`~/.cache/whisper/` 本地缓存）
- 模型文件实测大小：`base.pt` 145,262,807 字节，`large-v3-turbo.pt` 1,617,941,637 字节，faster-whisper large-v2 `model.bin` 3,086,912,962 字节

指标

- 处理时间：模型加载时间与转写本体耗时分开实测
- 时间戳精度：两个后端分段起止时刻的最大差（max start/end delta）。对字幕与区间引用而言，这比速度更关键
- 误识别模式：除与参考文本的归一化相似度外，还记录具体错在哪里（尤其是专有名词）
- 输出形状：分段数、词数、字幕排版块数是否一致

幻觉（静音区间的固定句反复等）没有作为独立自动指标测量。但事后复查了真实 60 秒切片上两个后端的输出：连续重复分段两个后端均为 0 件，分段内短字符串反复（连续 3 次以上）也为 0 件，3 秒以上静音间隙之后紧跟不自然句子的情况同样为 0 件。

结果

日语合成语音基准
9.527 秒，干净/混噪，带参考文本。

路径	加载	转写	相似度	备注
stable_whisper `base`（干净）	0.471s	1.518s	0.7957	快，但专有名词与技术词汇易碎
stable_whisper `base`（噪声）	0.471s	0.427s	0.7872	同上
stable_whisper `large-v3-turbo`（干净）	3.509s	1.943s	0.8958	日语质量良好
stable_whisper `large-v3-turbo`（噪声）	3.509s	1.745s	0.9149	日语质量良好
Faster-Whisper `large-v2` direct（干净）	1.882s	8.665s	0.8958	文本与 turbo 相当
Faster-Whisper `large-v2` direct（噪声）	1.882s	8.585s	0.9149	文本与 turbo 相当

相似度整体偏低的原因见下一节：其中包含了语音合成侧的读音错误。

专有名词的处理
日语基准的朗读原文如下。

> 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。

用真实公司名重新测量后发现，专有名词在两个层面同时碎裂。

第一，语音合成（TTS）层先碎。macOS `say -v Kyoko` 把汉字「誠明堂」误读成「まことみんどう」（makoto-mindō），所有后端都忠实转写了这个误读（large-v3-turbo 写出「マコトミンドウ」，Faster-Whisper 写出「まことみんどう」等）。上表相似度整体偏低正是因此：在 ASR 看到音频之前，公司名就已经在输入侧碎掉了。

第二，即使读音正确，识别（ASR）层也会碎。在把公司名写成平假名以强制正确读音的变体中，base 写出「精明堂」，large-v3-turbo 写出「声明道」，Faster-Whisper 写出「声明堂」：三种不同的汉字猜测，没有一个到

达「誠明堂」。这个读音对应哪个汉字，本来就不是音频里包含的信息。英语冒烟测试同理：“Seimeido”在两条 Faster-Whisper 路径上变成“CMEED”，在 stable_whisper `base` 上变成“CME DAO”。

也就是说，无论 TTS 侧还是 ASR 侧，专有名词的写法都不是靠选后端能解决的问题，它属于词典与后处理（清洗）层。因此本文把它排除在后端判定依据之外。

真实讲座音频 60 秒切片（最关键的测试）
最接近实际运行的条件。

指标	stable-whisper	faster-whisper
耗时	14.407s	117.599s
分段数	15	14
词数	291	297
排版块数	16	15

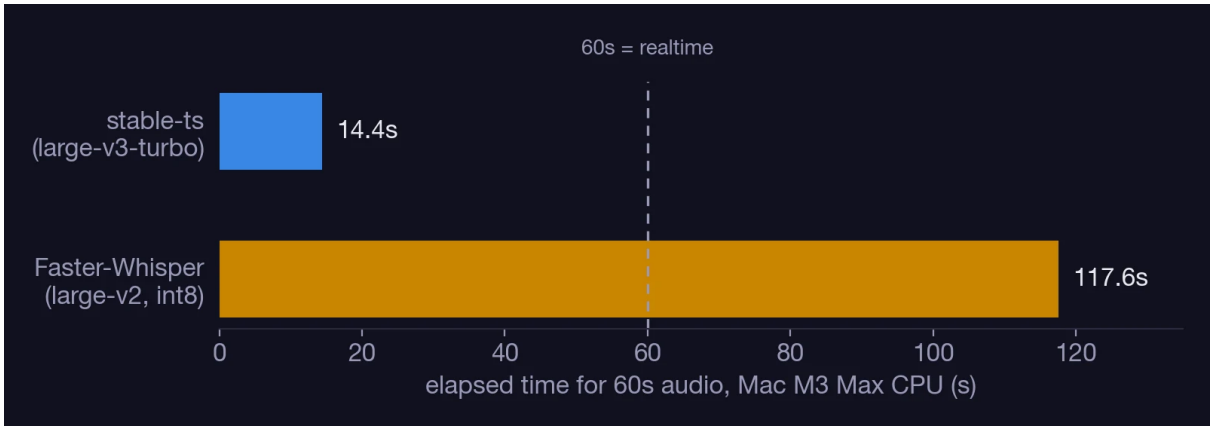

图 2　真实 60 秒音频的处理时间对比：stable-ts 14.4 秒、Faster-Whisper 117.6 秒，含 60 秒=实时的基准线

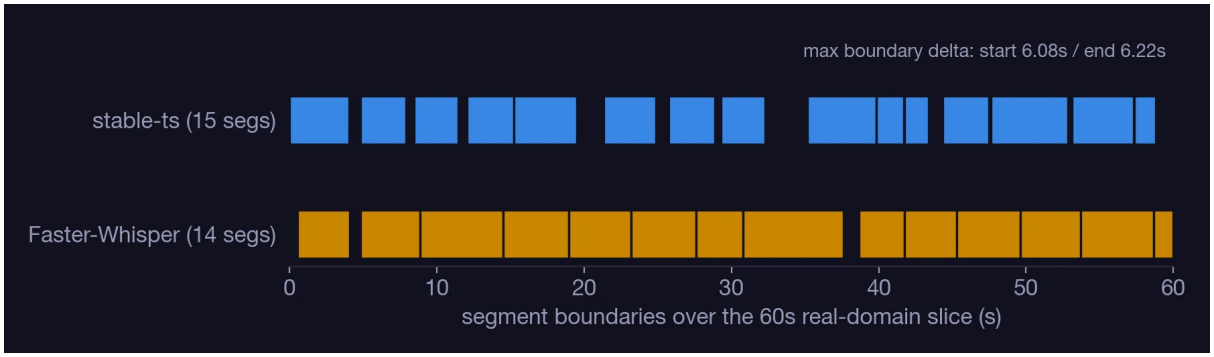

图 3　两个后端的分段边界在 60 秒时间轴上的对比：stable-ts 15 段、Faster-Whisper 14 段

归一化文本相似度 0.97318（并不完全一致），时间戳最大差 start 6.08s / end 6.22s。在这条 **CPU** 路径上 **Faster-Whisper** 慢了约 **8** 倍，文本与时间轴接近但不相同。

取舍

- 运行默认值是 **stable-ts（stable_whisper）+ `large-v3-turbo`**。在 Mac CPU 上速度与日语质量的平衡最好，相对 `base` 的质量提升幅度大。真实 60 秒音频约 14 秒的处理时间足以支撑日常使用。
- **Faster-Whisper** 不设为默认。公开基准的「最高快 4 倍」以 CUDA 为前提；在这台 Mac 的 CPU + int8 路径上反而慢了约 8 倍。作为行为不同的第二谱系，保留为对比与回归校验用的可选后端。
- 只有在 GPU/其他运行时条件下重新实测、或在代表性真实音源上出现明确质量优势时，才会更换默认值。

复现步骤

最小复现命令（CPU、本地缓存、离线）：

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

```
# Faster-Whisper（对比路径，本地 snapshot、离线）
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")
```

对比时把 wall time（加载与转写分开）、文本、分段/词数、分段起止时刻差记录为 JSON。设置 `HF_HUB_OFFLINE=1` / `TRANSFORMERS_OFFLINE=1` 可固定复现条件。

另外，stable-ts 本身也能通过 `stable_whisper.load_faster_whisper(...)` 把 Faster-Whisper 用作推理后端。这条整合路径在英语冒烟测试中同样跑通了（加载 1.693 秒 / 转写 6.653 秒）。但 stable-ts 的招牌功能 `refine()` 在该路径上尚未实现，「Faster-Whisper 的速度加 stable-ts 的后处理」目前无法兼得。也就是说实际的选项不是二选一，而是三选一：(1) 仅 stable-ts，(2) 仅 Faster-Whisper，(3) stable-ts + Faster-Whisper 整合路径。本文的运行默认值 (1) 正是从这三者中选出的结果。