Client-Side ASR - Whisper In The Browser - Engineering Cheat Sheet

1 - Untangle the name (the #1 mistake)

```
'faster-whisper' = CTranslate2 server/desktop library - NOT a browser engine.
Real browser engines: whisper.cpp (WebAssembly) + Transformers.js (WebGPU).
Built-in Web Speech API sends audio to Google by default - not on-device.
```

2 - The in-browser pipeline (all in the tab)

```
Mic -> getUserMedia -> resample to 16 kHz mono -> VAD gate -> Whisper -> caption.
No audio leaves the device. The VAD gate skips silence and saves the battery.
```

3 - Two engines: WebAssembly vs WebGPU

```
WebAssembly (whisper.cpp): runs on the CPU; SIMD + threads; the fallback.
  -> threads need cross-origin isolation (COOP + COEP headers) or they crawl.
WebGPU (Transformers.js): runs on the GPU; up to ~100x faster; default late 2025.
```

4 - The cost ledger (no cloud bill - cost moves to the device)

```
Download (once, cached): tiny 75 / base 142 / small 466 MB; quantized 31 / 57 / 182.
Memory: a few hundred MB for tiny, more for larger. Battery: only while running.
Sweet spot for live: tiny/base, quantized, VAD-gated, WebGPU when available.
```

5 - Does it keep up live? The real-time factor (RTF)

```
RTF = processing time / audio length.  Below 1.0 = keeps up;  above 1.0 = falls behind.
Example: base on CPU 1.2 s / 1 s = 1.2 (too slow);  on WebGPU 0.25 s / 1 s = 0.25 (fine).
Always measure on the device your users actually have. Moonshine suits tight live loops.
```

6 - Client or server? Checklist

- [ ] Audio must stay private / must work offline / 1:1 or single user -> run on the CLIENT.
- [ ] Group call you record, search, or translate / need top accuracy -> run on the SERVER.
- [ ] Prefer WebGPU; fall back to WebAssembly; ship COOP + COEP headers for Wasm threads.
- [ ] Size the model to your WORST supported device - not the developer's laptop.
- [ ] Recover accuracy with VAD + noise suppression, not with a bigger model.