Supertonic 3 is interesting for a different reason than most new TTS releases.

It is not trying to win the biggest-model contest. It is a compact, ONNX Runtime based TTS system designed to run locally across desktop, browser, mobile, and edge targets. That makes it relevant for product teams that care about offline use, privacy, cold-start time, and simple deployment more than maximum voice-clone fidelity.

We ran a bounded macOS smoke test to answer the practical question first: can Supertonic 3 run locally on Apple Silicon, how much memory does it use, and what kind of output does the default Python SDK produce?

60-second takeaway

- **It runs locally on macOS.** The Python SDK installed cleanly in a Python 3.12 `uv` environment and generated a valid WAV on Apple Silicon.
- **The default path is CPU-first.** ONNX Runtime exposed CoreML on this host, but Supertonic's package default used `CPUExecutionProvider`.
- **Memory use was modest for local TTS.** The cached short synthesis peaked at about 547 MB resident memory. Model initialization alone peaked at about 528 MB.
- **No discrete VRAM was used.** On Apple Silicon, GPU and CPU share unified memory, but this test did not use a GPU execution provider.
- **This is an on-device deployment candidate, not a replacement for fine-tuned voice-cloning models.** Put it next to Kokoro in the edge lane, not next to IndexTTS2 or VoxCPM full-SFT.

What Supertonic 3 is

The upstream project describes Supertonic as an on-device multilingual TTS system powered by ONNX Runtime. The public repository lists Python, Node.js, browser, Java, C++, C#, Go, Swift, iOS, Rust, and Flutter examples, plus a Python quick start that auto-downloads model assets on first run.

Supertonic 3, released on April 29, 2026, adds:

- 31-language support
- public ONNX assets compatible with the v2 code path

- fewer repeat and skip failures, according to upstream release notes
- 44.1 kHz 16-bit WAV output
- inline expression tags
- a compact open-weight model positioned around 99M parameters

The model weights are released separately from the sample code. Treat the code license and model license as separate production checks.

Our macOS smoke test

We tested the Python SDK path because it is the shortest path from zero to local audio output.

Host and environment

Item	Value
Host	Apple Silicon macOS, Darwin 25.4.0, arm64
Python env	uv virtualenv with CPython 3.12.13
Package	supertonic==1.3.1
Model	Supertone/supertonic-3
SDK path	from supertonic import TTS
Voice style	M1
Output text	Supertonic is running locally on macOS.
Output file	/assets/audio/tts/supertonic/supertonic_macos_smoke.wav

The first attempt failed in the sandbox because the model download could not resolve Hugging Face DNS. After allowing network access, one download attempt hit an HTTP read timeout after 2 of 26 files. Retrying with longer Hugging Face timeouts completed the download.

```
HF_HUB_DOWNLOAD_TIMEOUT=120 \
HF_HUB_ETAG_TIMEOUT=120 \
HF_HUB_DISABLE_IMPLICIT_TOKEN=1 \
HF_XET_HIGH_PERFORMANCE=1 \
python smoke_supertonic.py
```

That retry fetched all 26 files and generated the sample.

Audio sample

The sample below is intentionally boring. It is a smoke-test sentence, not a quality showcase.

The generated file was:

Property	Value
Format	WAV
Sample rate	44,100 Hz
Channels	1
Subtype	PCM 16-bit
Frames	129,024
Duration	2.86 seconds
Array shape	(1, 129024)
Array dtype	float32

The local file inspection identified it as:

```
RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, mono 44100 Hz
```

RAM, cache, and VRAM behavior

This is the part that makes Supertonic useful to evaluate separately from the GPU-heavy voice-cloning models.

Measurement	Result
Import only	~41 MB max RSS
Model initialized	~528 MB max RSS
Full short synthesis	~547 MB max RSS
Local model cache	385 MB
Smoke-test work folder	124 MB
Discrete VRAM used	None in this test

The ONNX Runtime package on this machine reported these available providers:

```
CoreMLExecutionProvider
AzureExecutionProvider
CPUExecutionProvider
```

Supertonic's default package configuration used:

CPUExecutionProvider

That matters because "runs on Apple Silicon" can mean different things. In this test it means the default SDK path generated audio locally on CPU through ONNX Runtime. It does not mean we validated a CoreML-accelerated path.

ONNX file sizes

The cached Supertonic 3 ONNX files were:

File	Size
vector_estimator.onnx	244.7 MiB
vocoder.onnx	96.7 MiB
text_encoder.onnx	34.7 MiB
duration_predictor.onnx	3.5 MiB

That explains the shape of the memory profile: small compared with 1.7B to 4.6B TTS stacks, but still not tiny enough to ignore on phones, browsers, or multi-process desktop apps.

Where it fits in the TTS decision tree

Supertonic belongs in the same broad deployment lane as Kokoro, but not for exactly the same reason.

Need	Better starting point	Why
CPU-first local app TTS	Supertonic 3	ONNX Runtime path, broad language coverage, no GPU needed
Smallest possible model footprint	Kokoro	Smaller model class and simpler low-compute positioning
Realtime voice agent	Qwen3-TTS	Stronger existing Instavar latency evidence
Fine-tuned branded voice	VoxCPM, Qwen3-TTS, IndexTTS2	We have first-party fine-tuning evidence for these paths
Lightweight cloning experiment	F5-TTS	Better fit when the goal is cloning experiments, not edge
Batch narration quality	CosyVoice 3 or VoxCPM	Better fit when quality matters more than local footprint

The practical rule: use Supertonic when local runtime constraints are the reason you are choosing a model. Do not choose it just because it is new.

What we would test next

This smoke test answers "does it run and what does it cost on macOS?" It does not answer production quality.

Before using Supertonic 3 in a shipped voice feature, we would still run:

1. **Long-form stability tests.** Generate 30-second, 2-minute, and 5-minute passages and check for skips, repeats, punctuation failures, and prosody fatigue.
2. **Language coverage checks.** Pick the actual target languages and run native-speaker review, not only English.
3. **Expression tag tests.** Verify whether tags such as laughter, breath, or sighs are controllable enough for product use.
4. **CoreML or WebGPU provider tests.** The CPU path is enough to prove local feasibility, but not enough to optimize Apple or browser deployment.
5. **Voice Builder workflow review.** If you need a branded or cloned voice, evaluate Supertone's Voice Builder path separately from the open SDK quick start.
6. **License and consent review.** Confirm the model license, generated-voice rights, custom voice ownership, and data provenance before commercial use.

Production caveats

1. "No GPU" does not mean "free"

A 547 MB peak RSS process is light next to multi-billion-parameter TTS, but it is still meaningful in a desktop app, Electron wrapper, browser tab, or mobile environment. If you plan to run many workers, budget memory per process.

2. Local generation does not remove voice consent work

Keeping synthesis on-device helps with privacy and vendor dependence. It does not solve speaker consent, custom voice ownership, or content policy.

3. The first run needs a model download

The Python SDK auto-downloads assets. That is ergonomic in a notebook and awkward in production unless you pre-bundle, prefetch, or manage cache state explicitly.

4. It is not yet part of our IMDA NSC fine-tuning benchmark

Our strongest Instavar evidence for voice-cloning quality still comes from the IMDA NSC benchmark series around Qwen3-TTS, VoxCPM, IndexTTS2, and CosyVoice. Supertonic has a different evidence tier here: local inference smoke test, not fine-tuned voice-cloning benchmark.

Bottom line

Supertonic 3 is worth tracking because it makes a different product promise: local, compact, multilingual TTS with a straightforward ONNX Runtime path.

Our macOS run supports that basic claim. It installed, downloaded the model, generated a valid 44.1 kHz WAV, and reran from cache quickly with a peak memory footprint around 547 MB.

That is enough to add it to the Instavar TTS decision tree as a CPU-first on-device option. It is not enough to call it a best voice-cloning model, a long-form narration winner, or a drop-in replacement for the fine-tuned GPU-backed models we have already benchmarked.

Related Instavar TTS coverage

- Which TTS Model Should You Use? A Decision Tree (2026) - where Supertonic now fits in the edge and CPU-first lane.
- Best Open-Source TTS Models for Production in 2026 - first-party comparison across the main voice-cloning benchmark models.
- Voice Cloning on a 24GB GPU - What Actually Works in 2026 - GPU and VRAM constraints for fine-tuned models.
- LoRA Finetuning Qwen3-TTS for Custom Voices - the stronger Instavar evidence base for realtime and LoRA-controlled voice cloning.
- F5-TTS Fine-Tuning Voice Cloning Guide - the lightweight cloning-experiment lane.

Sources

- Supertonic GitHub repository
- Supertonic 3 demo and release page
- SupertonicTTS paper on arXiv
- First-party macOS smoke test artifact: /private/tmp/supertonic-macos-smoke/smoke_supertonic.py
- First-party audio sample: /public/assets/audio/tts/supertonic/supertonic_macos_smoke.wav