MOSS-TTS is one of the fastest-rising open TTS repos this month, but this post is intentionally a first technical read, not a final benchmark verdict.

The right posture right now is: separate what is public and runnable from what is still a claim, then run our own bounded 24GB feasibility smoke test before making deployment calls.

> **Status note (as of February 17, 2026):**
> MOSS-TTS has public code, model cards, Hugging Face checkpoints, and demo apps.
> GitHub repo metadata is also clear: created on February 7, 2026, with no releases or tags yet.
> Treat this as an early-release system with high momentum and active issue traffic.

60-second takeaway

- **Coverage is broad:** OpenMOSS ships a family (single-speaker TTS, dialogue, voice generation, realtime, sound effects), not one narrow model.
- **The flagship numbers are promising but still author-reported:** README/model cards report strong Seed-TTS-eval results for MossTTSDelay and MossTTSLocal.
- **24GB feasibility is still unknown for our stack:** checkpoint footprints suggest Local is the practical first target; Delay likely needs tighter memory strategy.
- **Production reality check is mixed:** setup is available, but there are still open install/runtime friction points in issues (especially around environment and platform behavior).
- **Next step is explicit:** publish this first read now, then append benchmark evidence after a bounded 24GB smoke test.

What is actually released today

OpenMOSS announced the MOSS-TTS Family release on **February 10, 2026** with public repo + model pages + demo surfaces.

Family scope from their docs includes:

- MOSS-TTS (flagship single-speaker and voice cloning)
- MOSS-TTSD-v1.0 (multi-speaker dialogue)
- MOSS-VoiceGenerator (text-to-voice design without reference speech)
- MOSS-TTS-Realtime
- MOSS-SoundEffect

For MOSS-TTS specifically, the model card describes two released architectures:

- **MossTTSDelay-8B** (positioned for production and long-form stability)
- **MossTTSLocal-1.7B** (positioned for research/evaluation and lighter runs)

Parameter and artifact reality check

The published checkpoints are substantial. From Hugging Face model.safetensors.index.json metadata:

Component	Reported total_size	Practical implication
MOSS-TTS (Delay)	16,979,683,328 bytes (~15.81 GiB)	Heavy core model footprint before runtime overhead.
MOSS-TTS-Local-Transformer	6,121,212,928 bytes (~5.70 GiB)	More plausible starting point for constrained GPUs.
MOSS-Audio-Tokenizer	7,098,265,600 bytes (~6.61 GiB)	Must be budgeted alongside the synthesis model.

Inference from these artifacts: an end-to-end 8B path plus tokenizer can cross 22 GiB in weight memory alone, so a 24GB card may need careful dtype/attention/offload strategy.

Reported benchmark snapshot (author-reported)

README/model-card tables report the following Seed-TTS-eval results:

Model	EN WER (%)	EN SIM (%)	ZH CER (%)	ZH SIM (%)
MossTTSDelay (8B)	1.79	71.46	1.32	77.05
MossTTSLocal (1.7B)	1.85	73.42	1.20	78.82

These are useful directional signals, but still **maintainer-reported** metrics. No external reproduction package was identified in the repo at time of writing.

Production reality check (what matters before adoption)

1) Environment stack is pinned and non-trivial

pyproject.toml pins a specific stack (torch==2.9.1+cu128, torchaudio==2.9.1+cu128, transformers==5.0.0, torchcodec==0.8.1), with optional FlashAttention 2.

That is workable, but it raises setup fragility risk across machines.

2) Open issues already surface practical friction

As of **February 17, 2026**, GitHub shows active issue volume around:

- install/setup failures
- Windows runtime speed and dependency behavior
- MOSS-TTS-Local app-path errors
- fine-tuning roadmap questions

This does not invalidate the model quality claims, but it does signal that operational polish is still in-progress.

3) Release engineering is still early

Repo metadata currently shows:

- **0 releases**
- **0 tags**

That is normal for an early launch, but teams should pin specific commit SHAs and checkpoint revisions instead of assuming stable release semantics.

Where this fits in our Instavar TTS coverage

This post sits between two layers of our current cluster:

- benchmark/run-driven posts (VoxCPM, Qwen3-TTS, IndexTTS2, CosyVoice)
- technical-report/deep-read posts (GLM-TTS, ReStyle-TTS)

MOSS-TTS adds a new angle worth covering now: an integrated family approach that combines cloning, dialogue, voice design, and realtime variants under one

open umbrella.

24GB feasibility smoke test update (February 17, 2026)

We ran two bounded timed smoke passes on desktop_ethernet (RTX 3090 Ti, 24,564 MiB):

- moss_tts_local_smoke (OpenMOSS-Team/MOSS-TTS-Local-Transformer)
- moss_tts_delay_smoke (OpenMOSS-Team/MOSS-TTS)

Setup used for this run

- Base Python env: /mnt/work/chee-wei-jie/voice-models/Maya1/venv
- Compatibility overlay for MOSS processors: transformers==5.0.0 and dependencies under /tmp/moss_tts_pydeps
- Smoke harness: /tmp/moss_tts_smoke.py (loads processor + model to CUDA, attempts short generation, logs structured result)

Measured outcomes

Model	Status	Runtime (benchmark wrapper)	Processor load	Peak GPU memory seen in nvidia-smi log	Outcome detail
MOSS-TTS-Local-Transformer	FAIL	13.409 s	6.393 s	24,111 MiB	torch.OutOfMemoryError while moving model to CUDA (.to(device))
MOSS-TTS (Delay 8B)	FAIL	20.143 s	13.634 s	24,111 MiB	torch.OutOfMemoryError while moving model to CUDA (.to(device))

Important caveat for interpretation

At test time, the same GPU already had a co-tenant process (VLLM::EngineCore from a vllm serve /models/DotsOCR_1_5 container) using about **18,558 MiB**.

So these results confirm:

- MOSS Local and Delay both fail under the current shared-GPU load profile.
- This is not yet a clean-room dedicated-GPU feasibility verdict for 24GB.

Next rerun required for final 24GB verdict

To produce a definitive 24GB fit call, rerun the same harness on an idle GPU window, then append:

1. dedicated-GPU Local vs Delay pass/fail status,
2. peak VRAM and runtime without co-tenant load,
3. one short audio quality sanity sample per model.

Related Instavar TTS coverage

- IMDA NSC Voice Cloning Finetuning Benchmark 2026 - run-specific benchmark summary across current baselines.
- VoxCPM 1.5 LoRA Finetuning on IMDA NSC FEMALE_01 - practical LoRA run notes and checkpoint choices.
- LoRA Fine-Tuning Qwen3-TTS for Custom Voices - configuration and quality notes from our Qwen3-TTS run.
- IndexTTS2 Finetuning on IMDA NSC FEMALE_01 - full-SFT baseline behavior and operational notes.
- GLM-TTS Technical Report for Production Zero-Shot TTS - architecture and evidence-tier breakdown for another recent open release.
- ReStyle-TTS and Relative Style Control in Zero-Shot TTS - style-control-focused research brief.

Sources

- OpenMOSS/MOSS-TTS (GitHub)
- MOSS-TTS README (release notes, architecture, eval tables)
- MOSS-TTS model card
- MOSS-TTSD model card
- MOSS-TTS Hugging Face model
- MOSS-TTS-Local-Transformer Hugging Face model
- MOSS-Audio-Tokenizer Hugging Face model
- MOSS-TTS GitHub issues