PREPRINT · NOT PEER REVIEWED

Mixture of Voice: Only One Voice Is Needed at a Time in On-Device Text-to-Speech

Tarun Kumar Chawdhury[1,*]

[1] DLYog Lab Research Services LLC

August 2026 · DLYOG-TR-2026-006

Code: github.com/dlyog/voiceyog-arm · Demo: youtube.com/watch?v=L4THa8PWQi4 · Submission: Devpost

ABSTRACT

Modern lightweight text-to-speech (TTS) models ship many voices in a single checkpoint, and every device that installs one carries the capacity to be *any* of those voices on *every* inference. Yet almost every deployment we are aware of uses one voice at a time. We propose **Mixture of Voice (MoV)**: resolve the voice axis at *distribution* time rather than inside the network at inference time, replacing one *N*-voice monolith with *N* independently deployable single-voice specialists. MoV is the deployment-time dual of Mixture-of-Experts [6]: the router is the user's own choice, it is external to the model, it needs no gate network, and each expert is a complete standalone artifact. We test the hypothesis where it is falsifiable. The quantity MoV depends on is the cost of *one* specialist, which is independent of *N*; so a single distilled voice is sufficient to measure it, and enumerating all *N* voices would add cost without adding evidence. Distilling one voice from Kokoro-82M [1] into a VITS/HiFi-GAN [3][4] student yields a **68.5 MB** model that, against the same teacher on the same Arm CPU of an NVIDIA DGX Spark GB10 [8], is **8.6× faster with 7.5× less resident memory** (82.9 ms vs 947.5 ms; 356 MB vs 2661 MB), needs no GPU, and reaches first audio **5.8× sooner** than the teacher does on the GB10 GPU. We state the cost plainly: MoV converts a per-device memory cost into an aggregate catalogue cost, and the GPU remains 2.7× faster per sentence once loaded. We report the Arm-specific runtime work that made CPU-only serving viable, and two negative results. All figures are produced by scripts and rechecked by a checker that fails on drift.

Keywords — mixture of voice, on-device inference, text-to-speech, knowledge distillation, mixture-of-experts, Arm CPU, asymmetric multiprocessing, ONNX Runtime, edge AI, voice banking, reproducible benchmarking

1. INTRODUCTION

A person who is losing the ability to speak — to amyotrophic lateral sclerosis, to head and neck cancer, to a stroke — may bank a recording of their voice while they still can. What they typically receive in return is a synthetic voice that lives inside somebody else's service: it needs a network, an account, a subscription, and a company that is still

trading in ten years. The engineering question underneath the human one is simple. Can a person's voice be made small enough and fast enough to live on hardware they already own, for as long as they keep the file?

Contemporary lightweight TTS makes this look close. Kokoro-82M [1] is an 82 M-parameter open-weight model, Apache-2.0 licensed and derived from StyleTTS 2 [2], that produces quality competitive with far larger systems. It also ships a set of voices in one checkpoint, which is the right decision for a general-purpose model and the wrong one for a single-user deployment: the released artifact is 326 MB, and on an Arm CPU it takes close to a second to speak a sentence, which is why it is normally run on a GPU.

We observe that the multi-voice property is paid for continuously and used intermittently. A device installs the capacity to be any voice, keeps that capacity resident, and then uses exactly one. This paper asks what happens if that axis is collapsed before the model ever reaches the device.

Contributions. (i) We state the *Mixture of Voice* hypothesis and give the cost model that makes it testable with a single voice (§3). (ii) We report a measured single-voice specialist and its performance against its own teacher on identical hardware (§5, §6). (iii) We describe the Arm-specific runtime work — topology-derived thread selection, verified against a profile — without which the CPU-only result does not hold (§7). (iv) We publish two negative results and the cost side of the MoV trade (§8, §9), and a claim checker that fails when prose and measurement disagree (§10).

2. RELATED WORK

Conditional capacity and sparsity. Sparsely-gated Mixture-of-Experts [6] established that a network may hold far more parameters than it activates, with a learned gate selecting a few experts per token. MoE resolves its routing *inside* the network at inference time; total parameters stay resident even though few are active. Our proposal shares the intuition that capacity should be conditional, and differs in where the condition is applied.

Distillation. Hinton et al. [5] showed that a compact student can absorb the behaviour of a larger teacher. Our student is not a compressed copy of the teacher: it is a smaller architecture trained from scratch on the teacher's output for one voice, so the multi-voice conditioning capacity is not quantized or pruned — it is absent by construction.

Neural TTS. VITS [3] is an end-to-end conditional VAE with adversarial training and normalizing flows; HiFi-GAN [4] supplies the waveform decoder. StyleTTS 2 [2] models style as a latent variable via diffusion and underlies Kokoro [1], the teacher and baseline used here.

Arm inference. ONNX Runtime [9] is the execution provider for the shipped graph. Arm's KleidiAI micro-kernels [10] target exactly the operators that dominate our profile; we report their absence from our pinned runtime as measured fact and claim no speedup we did not observe.

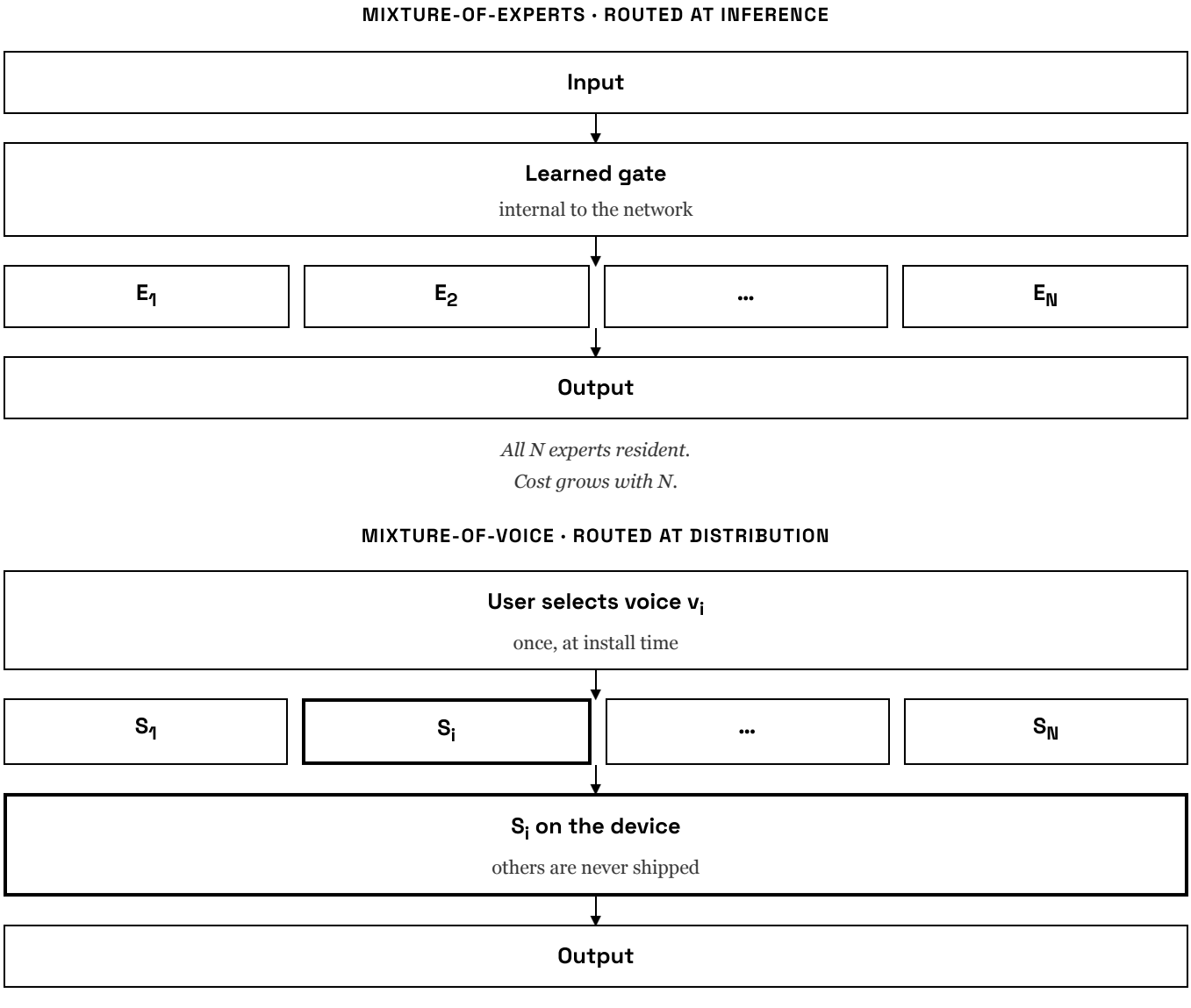
3. THE MIXTURE OF VOICE HYPOTHESIS

Let M_N be a TTS model supporting voices $v_1 \ldots v_N$. Serving voice v_i requires the whole of M_N to be installed and resident, because the voice identity is an input to the network rather than a property of the artifact.

MoV replaces M_N with a set of specialists $S_i = distil(M_N, v_i)$, each a complete model that can produce exactly one voice. Selection becomes a function of the installation, not of the forward pass: the user chooses once, the device

holds one artifact.

FIG 1 — WHERE THE VOICE AXIS IS RESOLVED



MoE keeps every expert resident and selects per token with a learned, internal gate. MoV selects once, before deployment, with a router that is the user's own choice and lives outside the model. The double-ruled boxes are the only artifacts a device ever holds.

3.1 Why one voice is a sufficient test

Write C_{dev} for what a single device must install and hold resident, and C_{cat} for what the publisher must store and distribute in aggregate. Then

```
C_dev(monolith) = |M_N|          grows with N
C_dev(MoV)      = |S|            independent of N
C_cat(MoV)      = N · |S|        grows with N
```

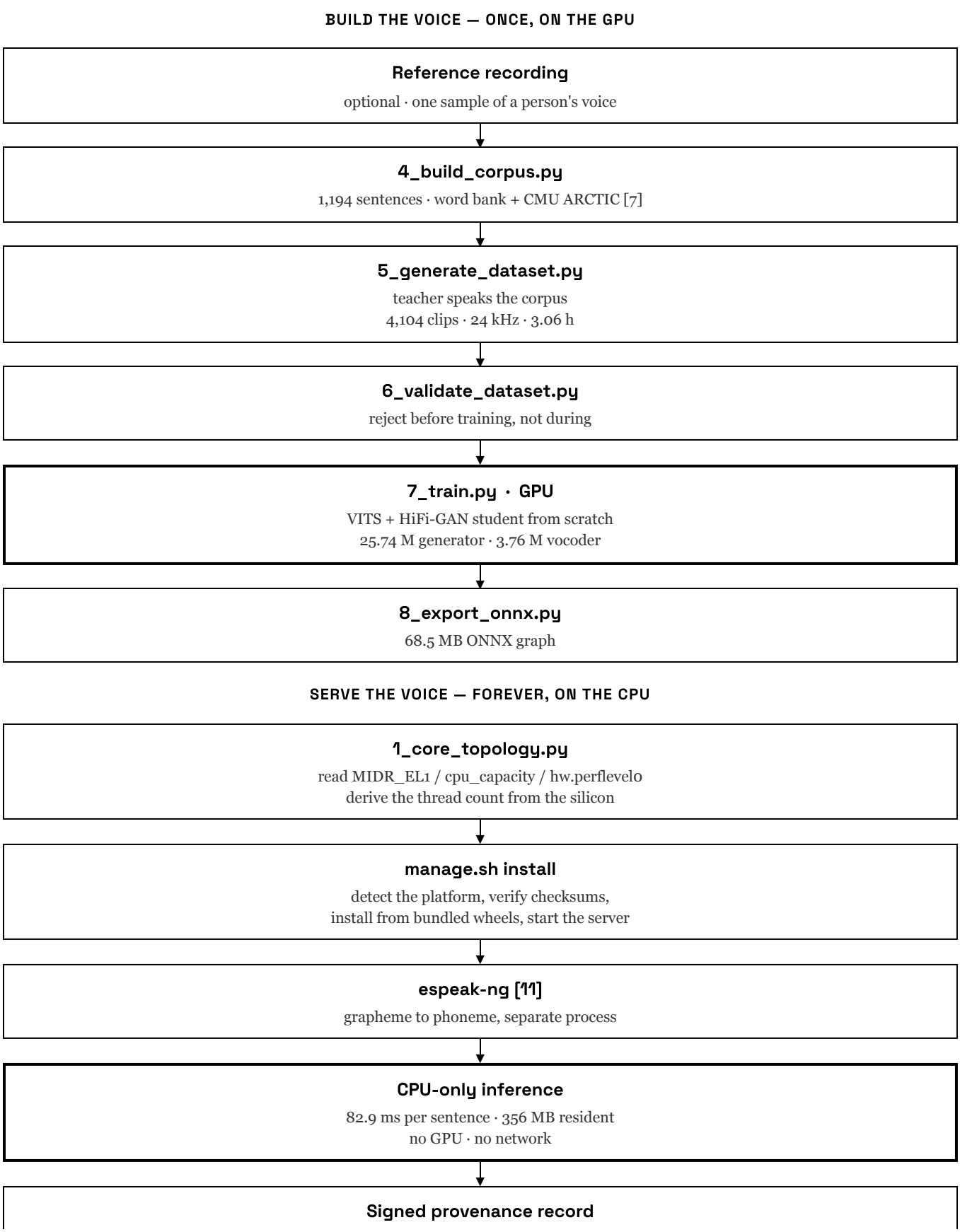
The claim MoV makes is about C_{dev}, and $C_{dev}(MoV) = |S|$ contains no term in N. It is therefore measured completely by building one specialist and characterising it. Training twenty specialists would produce twenty samples of a quantity whose dependence on N is nil; it would raise our compute bill and leave the hypothesis exactly as well or as badly supported.

What a single voice cannot establish is uniformity — whether every voice in a catalogue distils equally well. We do not claim it, and we mark it as the principal open question in §11.

4. METHOD

The pipeline is eight scripts. A teacher speaks a fixed corpus in the target voice; the resulting pairs train a smaller student from scratch; the student is exported to ONNX and served on the CPU. Only the teacher step differs between distilling a catalogue voice and cloning a person's own voice from a single reference recording.

FIG 2 — PIPELINE. THE GPU IS USED ONCE; THE CPU SERVES FOREVER.



model, checksum, runtime, platform [12]

The right-hand column is all a device
ever runs. Nothing above it is needed again.

Every stage is a script in the released repository, and each figure shown is the value that script printed on the DGX Spark. Only the teacher in `5_generate_dataset.py` differs between distilling a catalogue voice and cloning a person's own.

Grapheme-to-phoneme conversion is delegated to espeak-ng [11], invoked as a separate process so that its licence does not attach to the shipped runtime. The prompt corpus is built offline from a phonetically-motivated word bank and the CMU ARCTIC prompts [7].

5. EXPERIMENTAL SETUP

The primary machine is an NVIDIA DGX Spark [8] built on the GB10 Grace Blackwell superchip: a 20-core Arm CPU comprising ten Cortex-X925 cores at 3.90 GHz and ten Cortex-A725 cores at 2.81 GHz, with a Blackwell GPU sharing 128 GB of unified LPDDR5X memory. A second target, an Apple M1 Max with 8 performance and 2 efficiency cores, is used to test that the runtime logic generalises across asymmetric Arm parts.

The benchmark of record uses 20 held-out sentences, five warm-up sentences, and a separate process per engine, with the GPU verified idle at start. Peak RSS is the process maximum. Real-time factor (RTF) is synthesis time divided by audio duration; lower is better.

6. RESULTS

Table 1. Benchmark of record — DGX Spark GB10, idle GPU, 20 held-out sentences, one process per engine.

Engine	RTF	Latency	Peak RSS	On disk
Ours — Arm CPU only	**0.03888**	**82.9 ms**	**356 MB**	**68.5 MB**
Ours — Arm CPU → GPU	0.01957	42.2 ms	1898 MB	68.5 MB
Kokoro-82M — GPU	0.01418	40.1 ms	3464 MB	326 MB
Kokoro-82M — Arm CPU	0.33447	947.5 ms	2661 MB	326 MB

Like for like. Rows one and four are the same task on the same silicon. The specialist is **8.6× faster** by RTF and uses **7.5× less** resident memory, in a file **4.8× smaller**. No accelerator is involved on either side, so the comparison carries no hidden hardware term.

Against the accelerator. Compared with the teacher on the GB10 GPU, the specialist holds **9.7× less** memory and reaches first audio **5.8× sooner** from cold (0.94 s vs 5.42 s, medians of three), because there is no CUDA context to build and no 326 MB to move into device memory. Once both are warm, the GPU is **2.7× faster per sentence**. Both statements are measured and both are published; the case for MoV rests on memory, cold start and the absence of an accelerator, not on beating a GPU at steady-state throughput.

7. MAKING THE CPU PATH VIABLE

A 68.5 MB graph does not by itself produce 82.9 ms. Before tuning anything we profiled the shipped workload with Arm Performix (`code_hotspots`, 41,593 samples): 94.0% of Arm CPU time is inside ONNX Runtime's kernels, 3.1% in OpenBLAS, 2.0% in libc, and Python interpreter overhead is under 1%. The tuning knob is therefore attached to nearly all of the workload rather than to a wrapper around it.

Both targets are asymmetric. ONNX Runtime parallelises an operator across intra-op threads and joins them, and that join is a barrier: the operator finishes when its slowest thread finishes. A thread placed on a core 28% slower delays every other thread, on every operator, for the whole graph — so on these parts, adding cores can subtract performance.

The runtime therefore reads the core layout at startup — from `MIDR_EL1` and `cpu_capacity` on Linux, `hw.perflevel0` on macOS — and derives a thread count instead of accepting one. On the GB10 the ten performance cores are `5–9,15–19`: interleaved across clusters, so a contiguous range chosen by inspection would place half the threads on efficiency cores.

Table 2. Topology-derived threads against the two obvious defaults. The prediction is written to the results file before the sweep runs, so it cannot be fitted afterwards.

Machine	Cores	Chosen	vs all cores	vs ORT default
DGX Spark GB10	20	9	1.39×	1.65×
Apple M1 Max	10	8	2.21×	1.20×

On both machines the topology's prediction was the measured optimum. The same binary and the same graph retune themselves per part with no recompilation and no configuration.

8. NEGATIVE RESULTS

INT8 quantization produced an artifact that does not run. Dynamic quantization made the file 3.3× smaller and rewrote 183 convolutions into `ConvInteger`, for which ONNX Runtime's CPU provider had no aarch64 kernel in this configuration. The same failure occurred at the same node on both targets. A smaller file is not an optimization if the artifact cannot be loaded.

Splitting the graph across CPU and GPU lost. A cooperative path that runs the encoder prefix on the Arm CPU and the decoder on the GPU reaches RTF 0.0196 — **0.72× Kokoro on the GPU, i.e. slower** — while using 1.8× less memory. The Arm CPU prefix is 57.4% of that pipeline and runs strictly before the GPU stage, so each processor idles while the other works; the unified-memory handoff itself costs 0.12 ms, or 0.49% of the pipeline. The handoff is not the problem, and the result is reported rather than dropped.

9. DISCUSSION: WHAT MOV COSTS

MoV is not free, and the cost is exactly the one the model in §3 predicts. Table 3 works it through for a catalogue of twenty voices, using our measured single-voice artifact and the released teacher.

Table 3. Projected cost at $N = 20$. Per-device rows are measured; the aggregate row is arithmetic on the measured artifact size, not a second experiment.

	Monolith	MoV
Shipped to one device	326 MB	**68.5 MB**
Resident during inference	2661 MB	**356 MB**
Accelerator for practical latency	GPU	**none**
Aggregate catalogue	**326 MB**	1,370 MB
Voices available per device	**N**	1

MoV trades an aggregate storage cost, paid once by a publisher where storage is cheap, against a per-device memory cost, paid continuously by every user on hardware where memory is the binding constraint. It is the wrong architecture for a product whose value is switching voices freely, and the right one for a person who needs one voice to keep working on a machine they own. The single-voice measurement is what makes that trade quantitative rather than rhetorical.

10. REPRODUCIBILITY

Every number above is written to JSON by the script that measured it, and a checker compares the prose against those files and exits non-zero on disagreement — 43 claims at the time of writing. It requires Python 3 and nothing else: no model, no virtual environment, no network. The thread sweep re-runs in about a minute on the reader's own machine and will report where their hardware disagrees with ours.

Artifacts

Code and evidence: github.com/dlyog/voiceyog-arm
Demo video: youtube.com/watch?v=L4THa8PWQi4
Submission: devpost.com/software/voiceyog-arm-optimized-tts-for-dgx-spark-and-apple-silicon

Every generated clip additionally carries a signed provenance record naming the model, its checksum, the runtime and the platform, verifiable offline by a recipient holding neither the model nor the private key; the design follows the shape of C2PA Content Credentials [12] without claiming conformance to that specification.

11. LIMITATIONS

One voice, not a catalogue. We measured a single specialist. §3.1 argues why that measures the quantity MoV depends on; it does not show that every voice distils equally well, and voices with unusual prosody or limited data may not.

No listening study. We report size, latency, memory and cold start. We do not report MOS or any human evaluation of naturalness, so no claim of perceptual parity with the teacher is made anywhere in this paper.

Two machines. The topology result holds on a GB10 and an M1 Max. Other asymmetric Arm parts are expected to behave similarly and were not tested.

Steady-state throughput. A GPU running the teacher remains faster per sentence once loaded. MoV addresses footprint and cold start.

Unmeasured next step. Our pinned ONNX Runtime 1.20.1 contains no KleidiAI [10] symbols, while 1.28.0 on the same machine contains eleven `kai_run_matmul_*` symbols. Those kernels target the operators that dominate our profile, which makes upgrading a measurable next step and not a claimed result.

12. CONCLUSION

A multi-voice TTS model asks every device to carry the ability to be any voice so that it can be one. Mixture of Voice moves that choice out of the forward pass and into distribution, and the cost model says the device-side saving does not depend on how many voices the catalogue holds — which is why one distilled voice is enough to test it. That voice is 68.5 MB, is 8.6× faster in 7.5× less memory than its own teacher on the same Arm CPU, and needs no accelerator. For the person who banked one voice and wants it to still work in ten years, on hardware they own, offline, that is the difference between a service and a file.

REFERENCES

1. hexgrad. *Kokoro-82M* — open-weight TTS model, 82 M parameters, Apache-2.0, built on StyleTTS 2. Hugging Face. [HF]
2. Li, Y. A., Han, C., Raghavan, V. S., Mischler, G., & Mesgarani, N. (2023). StyleTTS 2: Towards Human-Level Text-to-Speech through Style Diffusion and Adversarial Training with Large Speech Language Models. *NeurIPS*, 36, 19594–19621. arXiv:2306.07691. [arXiv]
3. Kim, J., Kong, J., & Son, J. (2021). Conditional Variational Autoencoder with Adversarial Learning for End-to-End Text-to-Speech. arXiv:2106.06103. [arXiv]
4. Kong, J., Kim, J., & Bae, J. (2020). HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis. arXiv:2010.05646. [arXiv]
5. Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the Knowledge in a Neural Network. arXiv:1503.02531. [arXiv]
6. Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q. V., Hinton, G., & Dean, J. (2017). Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *ICLR 2017*. arXiv:1701.06538. [arXiv]
7. Kominek, J., & Black, A. W. (2004). The CMU Arctic Speech Databases. *5th ISCA Speech Synthesis Workshop (SSW5)*. [ISCA]
8. NVIDIA. *DGX Spark* — GB10 Grace Blackwell Superchip; 20-core Arm CPU (10× Cortex-X925, 10× Cortex-A725), 128 GB unified LPDDR5X. [NVIDIA]
9. ONNX Runtime — cross-platform inference accelerator. [onnxruntime.ai]
10. Arm. *KleidiAI* — micro-kernels for AI workloads on Arm CPUs, integrated into ONNX Runtime's MLAS library. [GitHub]
11. eSpeak NG — open-source speech synthesizer and grapheme-to-phoneme front end. [GitHub]
12. C2PA. *Content Credentials: C2PA Technical Specification 2.1* (2024). [C2PA]
13. Chawdhury, T. K. (2026). *VoiceYog: Arm-Optimized TTS for DGX Spark and Apple Silicon* — code, evidence and claim checker. [GitHub]
