PREPRINT · NOT PEER REVIEWED

Distribution-Time Voice Specialization: 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

Lightweight text-to-speech (TTS) systems are typically distributed so that a device holds one shared network capable of producing any voice in a catalogue, even though many deployments ultimately use a single voice at a time. We examine **Distribution-Time Voice Specialization (DTVS)**: resolving the voice axis at *distribution* time rather than inside the network at inference time, replacing one *N*-voice deployment with *N* independently deployable single-voice specialists. DTVS is a deployment-time analogue of Mixture-of-Experts [6], though it moves in the opposite direction: it specializes before deployment and removes runtime routing entirely, so the router is the user's own choice, external to the model, and needs no gate network, while each specialist is a complete standalone artifact. The device-side cost of DTVS is the cost of *one* specialist, a quantity independent of *N*, so a single distilled voice is sufficient to characterise it. Distilling one voice from Kokoro-82M [1] into a smaller VITS/HiFi-GAN [3][4] student yields a **68.5 MB** artifact that, against the same teacher on the same Arm CPU of an NVIDIA DGX Spark GB10 [8], attains **8.6× lower real-time factor** (RTF 0.0389 vs 0.3345), **11.4× lower per-sentence latency** (82.9 ms vs 947.5 ms) and **7.5× less resident memory** (356 MB vs 2661 MB), with no GPU. We are careful about what this does and does not establish: our student changes two things at once — it drops multi-voice conditioning *and* uses a smaller architecture — so the reduction is attributable to the combination, not to specialization alone (§3.2). We state the costs plainly: DTVS converts a per-device memory cost into an aggregate catalogue cost, Kokoro on the GB10 GPU retains 2.74× lower RTF once loaded, and the student carries the quality compromise usual to a distilled model — UTMOS22 4.250 against its teacher's 4.485, a paired difference of −0.235 at $p = 3.8\times10^{-6}$ — measured from a deliberately small budget of 3.06 hours of teacher audio, 180 epochs and a 3.76 M-parameter vocoder, none of the usual scaling levers having been applied; ASR intelligibility over the same text is WER 0.0186 with 19 of 20 sentences transcribed exactly, so the compromise falls on delivery rather than on whether the words arrive. We also report the Arm-specific runtime work that made CPU-only serving viable, and two negative results. All figures are produced by scripts and re-checked by a checker that fails on drift.

Keywords — distribution-time voice specialization, 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 is distributed as a shared model checkpoint together with separate per-voice style packs, and the network accepts a per-voice reference/style tensor at inference. The distinction matters for our argument and makes it cleaner: the per-voice data is small, but *selecting one voice still requires the whole shared model*. The released checkpoint is 326 MB, and on an Arm CPU it takes close to a second to speak a sentence, making GPU inference attractive for latency-sensitive use.

So the capacity to serve any voice is paid for continuously and used intermittently. A device installs the shared network, keeps it resident, and then uses it for exactly one voice. This paper asks what happens if that axis is collapsed before the model ever reaches the device.

Contributions. (i) We state the *Distribution-Time Voice Specialization* hypothesis, give the cost model that makes its device-side claim measurable with a single voice, and identify the confound that prevents us claiming specialization alone as the cause (§3). (ii) We report a measured single-voice specialist 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 trade (§8, §9), and a claim checker that fails when prose and measurement disagree (§10).

Stated conservatively, the contribution is not that removing voices causes an 8.6× RTF improvement. It is that a single-voice specialist deployment was built and measured, and then made practical on asymmetric Arm CPUs through hardware-aware ONNX Runtime tuning.

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 input. A conventional MoE checkpoint retains the expert set as part of the deployed model while only a subset is activated per input, and the routing decision is made *inside* the network at inference time. We note that production MoE systems may sequence this differently — experts can be sharded, distributed or offloaded — so "resident" describes the common

single-node case rather than a property of MoE in general. Our proposal shares the intuition that capacity should be conditional, and differs in where the condition is applied.

Voice selection in TTS systems. ChatAnything [14] includes a component that selects among predefined TTS tones at inference time, driven by a text description of a persona. The contrast with DTVS is instructive in both *what* is selected and *when*: that mechanism chooses a tone inside a running system and keeps the selection machinery live, whereas DTVS determines which artifact is ever installed, leaving nothing to select at run time. The two address different problems — persona expressiveness on one side, deployment footprint on the other — and they compose rather than compete.

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.

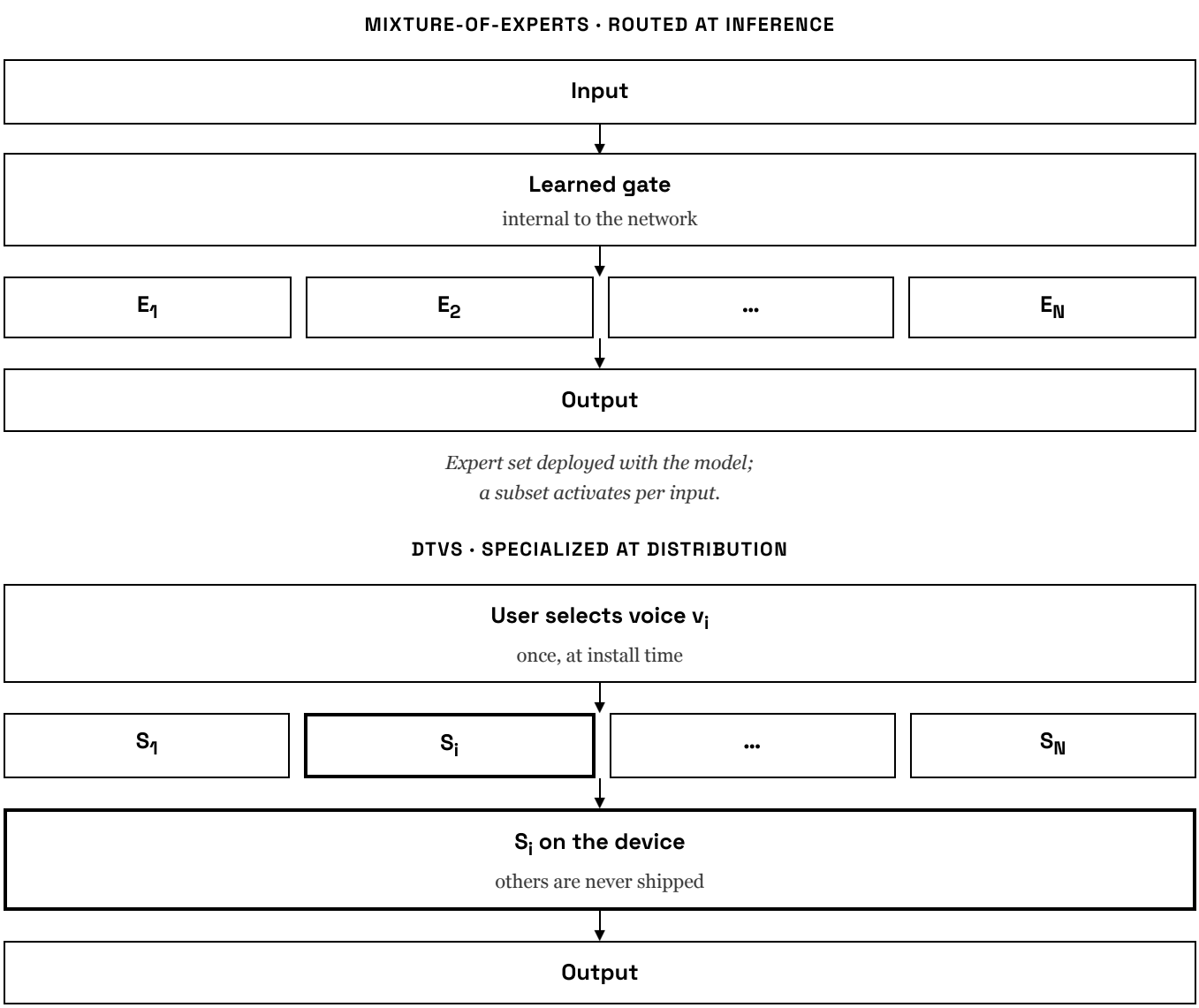
Arm inference. ONNX Runtime [9] is the execution provider for the shipped graph. Arm's KleidiAI micro-kernels [10] target exactly the operators that dominate our profile; we report their absence from our pinned runtime as measured fact and claim no speedup we did not observe.

3. THE DISTRIBUTION-TIME VOICE SPECIALIZATION 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.

DTVS 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



A conventional MoE checkpoint carries its expert set into deployment and activates a subset per input, selected by a learned gate inside the network. DTVS 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. (Production MoE systems may shard or offload experts; the left panel depicts the common single-node case.)

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(DTVS)     = |S|            independent of N
C_cat(DTVS)     = N · |S|        grows with N
```

The claim DTVS makes is about C_{dev}, and $C_{dev}(DTVS) = |S|$ 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 an open question in §11.

3.2 What our measurement does not isolate

A confound we cannot remove post hoc

Our student differs from the teacher in *two* ways at once: it drops multi-voice conditioning, *and* it is a smaller VITS/HiFi-GAN architecture trained from scratch rather than a compressed copy. The measured 326 MB → 68.5 MB reduction and the CPU speed-up are therefore attributable to that *combination*, and this experiment cannot apportion the credit between them.

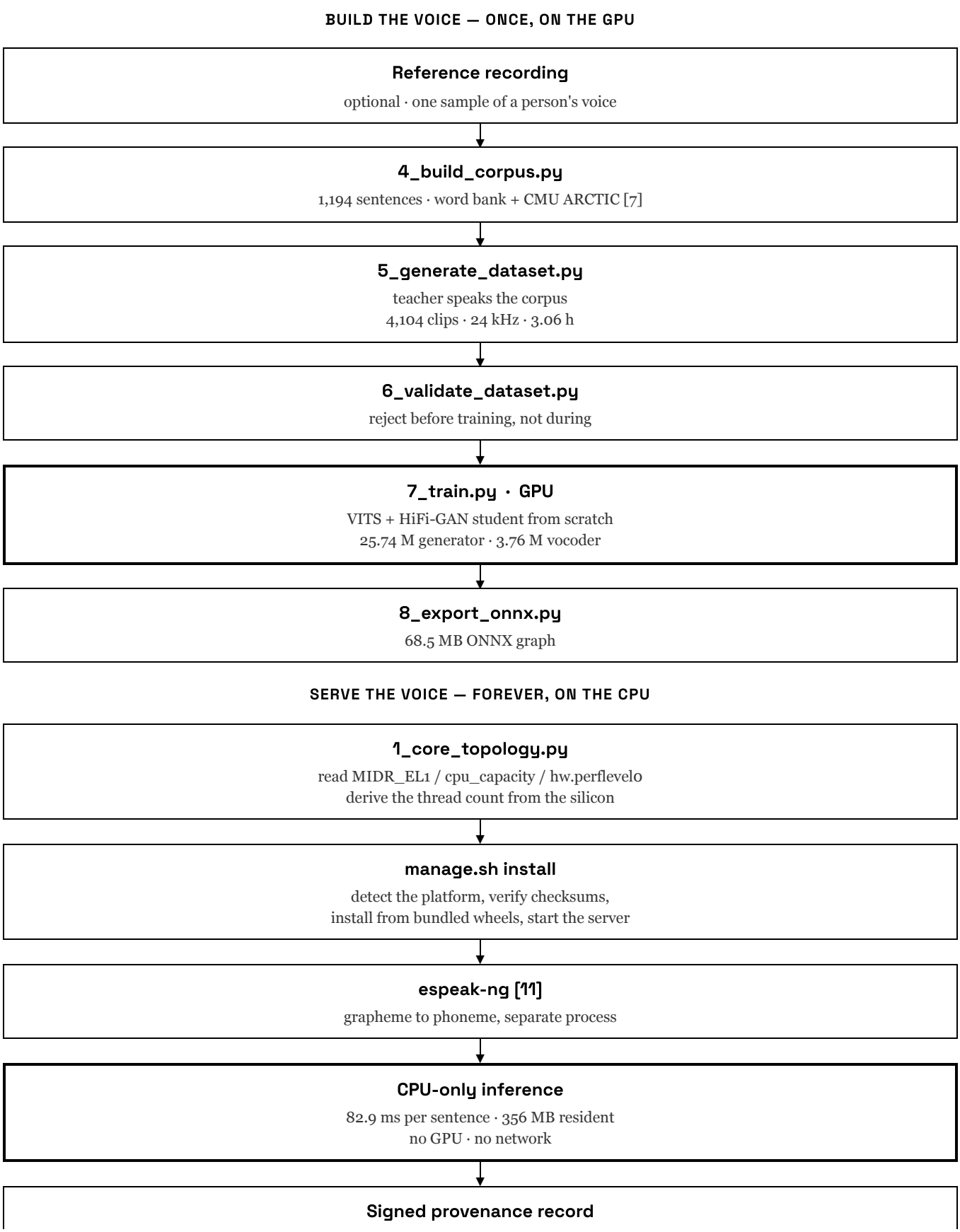
We therefore do not claim that removing the voice axis causes the reduction. The defensible claim is narrower and is the one we make throughout: a distribution-time single-voice specialization *can* yield a substantially smaller and faster deployment artifact, and we measured one that does.

The ablation that would establish causality is well defined and we did not run it: train the *same* student architecture twice, once multi-voice and once single-voice, and compare. That isolates the voice axis with architecture held constant. Until somebody runs it, the honest reading of §6 is that specialization plus a smaller architecture, together, produced the artifact — with the practical consequence unchanged, because a deployment gets both or neither.

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], which is invoked as a separate executable; VoiceYog does not link eSpeak NG code into its runtime. eSpeak NG is GPL-3.0-or-later, and we describe only that engineering arrangement — we draw no conclusion about its licensing consequences, which depend on both the mechanism and the semantics of the communication and are outside the scope of this paper. 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, so the comparison carries no hidden hardware term. The two speed ratios differ and we report both rather than picking one: **8.6× lower RTF** (0.03888 vs 0.33447) and **11.4× lower mean per-sentence latency** (82.9 ms vs 947.5 ms). They diverge because the two engines synthesised different total audio durations over the same 20 sentences, and RTF normalises by audio produced while latency does not. RTF is the more conservative figure and is the one we quote when a single number is needed; wherever this paper says "8.6×" it means RTF. Memory is **7.5× less** and the file is **4.8× smaller**.

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 runs). That gap is consistent with avoiding CUDA context initialisation and GPU-side model setup and allocation, but we did not profile initialisation, so we attribute it to those operations only as a plausible explanation rather than a measured decomposition. We specifically do *not* describe it as copying weights into discrete video memory: the GB10 exposes 128 GB of coherent unified memory [8], so the traditional host-to-device transfer is not the mechanism here.

Once both are warm, Kokoro on the GPU retains **2.74× lower RTF** (0.01418 vs 0.03888), equivalently about **2.1× lower mean sentence latency** (40.1 ms vs 82.9 ms). Both are measured and both are published; the case here rests on memory, cold start and the absence of an accelerator, not on beating a GPU at steady-state throughput.

6.1 Predicted speech quality

Size and speed are half a result. We therefore scored both engines with UTMOS22 [18], the VoiceMOS Challenge system, on the same 20 held-out sentences — verified disjoint from all 4,104 training texts — so the comparison is paired, with every sentence appearing on both sides.

Table 3. UTMOS22 predicted MOS, paired on identical held-out text. Higher is better; 5 is the ceiling.

Engine	Predicted MOS (95% CI)	On disk	RTF, Arm CPU
Kokoro-82M (teacher)	4.485 ± 0.017	326 MB	0.33447
Ours (student)	**4.250 ± 0.126**	**68.5 MB**	**0.03888**
paired difference	**−0.235**, $p = 3.8\times10^{-6}$	4.8× smaller	8.6× lower

The student carries a measurable quality compromise, as a distilled student generally does. A Wilcoxon signed-rank test over the paired scores gives $p = 3.8\times10^{-6}$, so the difference is a real one rather than sampling noise, and we report it as a number rather than reaching for a phrase like "comparable quality". The summary is that **0.235 predicted MOS buys 4.8× smaller and 8.6× lower RTF**, at a training budget of 3.06 hours of teacher audio, 180 epochs and a 3.76 M-parameter vocoder against the teacher's 19.69 M. Whether that trade is worth making depends on the deployment, and §6.2 explains why the figure should be read as a starting point rather than a property of the method.

Two cautions on reading the absolute numbers. UTMOS is a *predictor* trained on human ratings, not a human study; and it assigns roughly 3.3 to pure silence, so the usable range is compressed and a score of 4.25 should not be read as "85% of perfect". The paired delta is the quantity we trust. We also expected run-to-run variation from stochastic synthesis and found none: three independent generations produced identical scores, so the exported ONNX path is deterministic.

Intelligibility. Predicted MOS speaks to naturalness, not to whether the words survive. Synthesizing the same held-out sentences, transcribing them back with Whisper (`small.en`, temperature 0) and scoring against the input text gives **WER 0.0186** and **CER 0.0245**, with **19 of 20** sentences transcribed exactly.

The single imperfect transcript is worth naming, because it is not a synthesis failure. The reference reads "Set the oven to three hundred fifty degrees" and Whisper returned "Set the oven to 350 degrees" — the model pronounced the number correctly and the ASR chose to write it in digits, which our normalizer scores as three substitutions. We report 0.0186 as measured rather than extending the normalizer to expand numerals after seeing which sentence failed; the honest reading is that intelligibility on this set is essentially perfect and the residual is a formatting artifact.

Read together with §6.1's MOS result, the two numbers separate cleanly: the words arrive intact, and what the student gives up is in the delivery. That is the expected shape of a distillation compromise, and it is the shape that matters for the uses this deployment targets, where being understood is the requirement and polish is a preference.

The compromise is not uniform across the set. The two largest differences fall on the two sentences carrying the most intonation — a question ("Can you hear me clearly on this call?", −0.89) and an exclamation ("Wow, I did not expect to see you here!", −0.67) — which is where a 3.76 M-parameter vocoder and a smaller duration and flow predictor would be expected to give ground first, and which therefore points at vocoder capacity as the first lever to try.

6.2 The student was deliberately undertrained

That gap should be read against how little was spent producing it. The student was trained on **3.06 hours** of teacher audio for **180 epochs**, on the order of ten hours on a single GB10. That is a small corpus and a short run by any standard, and it was stopped there on purpose: the object of this work is the deployment argument in §3, not a quality record, and a stronger student would not have changed the cost model.

The consequence is that 0.235 characterises *this* training budget, not the method: it is the figure obtained with every conventional scaling lever left in its lowest position. Those levers are the ordinary ones — more teacher audio than 3.06 hours, more than 180 epochs, and a vocoder larger than 3.76 M parameters against the teacher's 19.69 M — and each of them costs GPU time during training and nothing at inference, so applying them would not disturb any result in §6 or §7. We have not run that experiment and so predict no particular figure; establishing the scaling curve is future work, and until it exists the number above should be read as a starting point.

> **On informal listening**
>
> The authors and their immediate family have listened to the output and find it clearly intelligible and pleasant enough for personal narration, tutorials and similar uses. We record that only to say what our own impression was, and it should carry no evidential weight whatever: the listeners were neither blind, independent, nor sampled, and the group is small. It is not a listening study and we do not present it as one. §11 says what an actual one would require.

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 lower-capacity core can therefore become the straggler for every other thread, on every operator, for the whole graph — so on these parts, adding cores can subtract performance. We deliberately do not quantify the per-core gap from clock frequency: the Cortex-X925 and Cortex-A725 are different microarchitectures, and a ratio of advertised clocks is not a ratio of workload throughput. The evidence for the effect is the sweep in Table 2, which measures it end to end rather than inferring it.

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 our DGX Spark test system Linux enumerated the performance cores as `5–9,15–19`, interleaved across clusters rather than contiguous, so a range chosen by inspection would place half the threads on efficiency cores. Logical CPU numbering is a property of what the OS reported on this machine, not an architectural guarantee, which is precisely why the runtime queries it instead of hard-coding it.

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 stage instrumentation totals 24.8 ms: 14.2 ms of Arm CPU prefix, a 0.12 ms handoff, and 10.4 ms of GPU decoder. *Percentages in this paragraph are shares of that 24.8 ms instrumented total, not of the 42.2 ms end-to-end latency in Table 1*, which additionally contains untimed setup — so the prefix is 57.4% and the handoff 0.49% on that basis. The prefix runs strictly before the GPU stage, so each processor idles while the other works. The handoff is not the problem, and the result is reported rather than dropped.

9. DISCUSSION: WHAT DTVS COSTS

DTVS 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	**DTVS**
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

DTVS 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.

9.1 Deployment identity and provenance (motivation and future work)

A second consequence of DTVS is structural rather than numerical, and we raise it as motivation and future work rather than as a result. Regulation is moving toward persistent provenance for synthetic audio: California's AI Transparency Act [16], as amended by AB 853 [17] and operative 2 August 2026, requires covered providers of generative-AI systems to attach a *latent*, tamper-resistant disclosure to AI-generated audio, to offer a manifest disclosure, and to publish a free detection tool.

Existing provenance schemes answer "was this generated, and by which system?" They answer "*which voice*?" less well. In a shared N-voice network the voice is a runtime argument, so the artifact hash identifies the system rather than the persona, and recovering the voice means trusting a field the runtime reported about itself. Under DTVS that distinction collapses: one voice is one artifact, so the model checksum already present in the record is a stable per-voice identifier, because there is exactly one voice the artifact can produce. **Voice identity becomes a property of the artifact rather than a parameter of the call.** That alignment costs nothing extra — it falls out of the deployment structure — which is why we think it is worth stating even though we did not build for it.

What we implement is modest and we describe it precisely. Each clip is accompanied by a signed record naming the model, its checksum, the runtime and the platform, verifiable offline by a recipient holding neither the model nor the private key, and tamper-evident: our tests confirm that altering a single bit of the audio makes verification fail.

What this explicitly does not do

The record is *detached*, not embedded. Nothing is written into the audio signal, so this is not a watermark, the record can be stripped by anyone redistributing the audio, and the waveform alone carries no identity. It therefore does **not** allow a listener to identify a voice by hearing it, and it does **not** satisfy a latent-disclosure requirement. We are also not a covered provider under the statute, which reaches systems above a one-million-monthly-user threshold, so we make no compliance claim of any kind. Stable identity would further depend on key management and distribution discipline that this paper does not address.

The obvious next step follows from the gap rather than from the result: pairing DTVS with an acoustic watermark would supply the half we lack, with the artifact providing the identity and the watermark providing survivability through redistribution. We have not built or evaluated that, and we claim nothing about it here.

This section discusses regulation as engineering motivation only. It is a research reading of a moving legal landscape, may contain inaccuracies, and is not legal advice; anyone with an actual compliance obligation should consult counsel and the statute itself.

10. REPRODUCIBILITY

Every number above is written to JSON by the script that measured it, and a checker [13] 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 DTVS depends on; it does not show that every voice distils equally well, and voices with unusual prosody or limited data may not.

The reduction is not isolated to specialization. As set out in §3.2, our student drops multi-voice conditioning and changes architecture simultaneously. The ablation that separates them — the same architecture trained multi-voice and single-voice — remains unrun, and until it is, no causal claim about the voice axis alone is supported by this work.

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. DTVS addresses footprint and cold start.

No human listening study. §6.1 reports an automatic predictor, which is not the same thing. UTMOS is calibrated on VoiceMOS Challenge systems that need not resemble ours, and 20 sentences is a small sample even paired. The ASR intelligibility figures in §6.1 cover a different axis and are reproducible from the repository, but they measure whether words arrive, not how the speech sounds. What remains absent is a speaker-similarity score against the teacher and a MUSHRA or ACR panel of 20–50 independent listeners with confidence intervals — neither of which we ran. The informal impressions noted in §6.2 are anecdote, not data, and nothing in this paper should be read as a claim about perceptual parity.

No attestation, and no compliance claim. The provenance discussion in §9.1 is motivation, not a contribution. Our record is detached rather than embedded in the audio, so it is not a watermark and cannot survive a recipient who discards it; we neither implement nor evaluate acoustic watermarking, speaker attestation, or any mechanism that would let a voice be identified from the waveform alone.

Unmeasured next step. Our pinned ONNX Runtime 1.20.1 contains no KleidiAI [10] symbols, while a newer build on the same machine contains eleven `kai_run_matmul_*` symbols. That build reports version **1.28.0** with `git-commit-id 45de2a8b06` (`cp312-manylinux_2_28_aarch64`); it is a pre-release wheel and was not, at the time of writing, in the stable PyPI channel [15], so we identify it by commit rather than by version number alone. Those kernels target the operators that dominate our profile, which makes upgrading a measurable next step and not a claimed result.

12. CONCLUSION

A shared multi-voice network asks every device to carry the ability to serve any voice so that it can serve one. Distribution-Time Voice Specialization 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 specialist is enough to characterise it. The specialist we built is 68.5 MB, attains 8.6× lower RTF and 11.4× lower per-sentence latency in 7.5× less memory than its own teacher on the same Arm CPU, and needs no accelerator. We cannot say how much of that came from dropping the voice axis and how much from the smaller architecture, and we report no quality evaluation; what we can say is that the artifact exists, was measured, and was made practical on asymmetric Arm CPUs by reading the silicon rather than guessing at it. 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]
14. Zhao, Y., Yuan, X., Gao, S., Lin, Z., Hou, Q., Feng, J., & Zhou, D. (2023). ChatAnything: Facetime Chat with LLM-Enhanced Personas. arXiv:2311.06772. *Cited in §2 for its inference-time selection among predefined TTS tones, contrasted with the distribution-time approach taken here.* [arXiv]
15. Saeki, T., Xin, D., Nakata, W., Koriyama, T., Takamichi, S., & Saruwatari, H. (2022). UTMOS: UTokyo-SaruLab System for VoiceMOS Challenge 2022. *Interspeech 2022*. arXiv:2204.02152. Used here via the `utmos22_strong` predictor. [arXiv]
16. California SB 942 (2024), *California AI Transparency Act*. Requires covered providers of generative-AI systems — those publicly accessible in California with more than one million monthly users — to apply latent and manifest disclosures to AI-generated image, video and audio content and to provide a free public detection tool. [leginfo]
17. California AB 853 (2025), *California AI Transparency Act* (amending SB 942). Signed 13 October 2025; moves the operative date to 2 August 2026 and extends later obligations to generative-AI hosting platforms, large online platforms and capture-device manufacturers from 2027. [leginfo]
18. ONNX Runtime release history, PyPI. *Used to establish that the 1.28.0 build scanned in §11 was a pre-release wheel (`git-commit-id 45de2a8b06`) rather than a stable-channel release at the time of writing.* [PyPI]

VoiceYog is released open-source under Apache-2.0 · Code: github.com/dlyog/voiceyog-arm

Preprint, not peer reviewed. This is research work and may contain inaccuracies. Nothing in it is legal advice, and the discussion of regulation in §9.1 is engineering motivation rather than a compliance assessment.

No human listening study is reported here; no claim of perceptual parity with the teacher model is made.