SINGLE THREAD, REAL TIME: HETEROGENEOUS QUANTIZATION FOR CPU SPEECH RECOGNITION

Anonymous authors
Paper under double-blind review

ABSTRACT

LM-based speech recognition models are among the most accurate ASR systems today, but they typically require GPUs for real-time inference, so private audio leaves the device and latency is tied to the network. We move them onto device CPUs and propose a memory-traffic model that assigns a quantized format per module. The transformer is weight-bound, so its weights are ternarized; the acoustic stack is activation-bound, so it runs entirely in INT8, which shrinks activation memory $6.2\times$ against FP32. Progressive fake-quantization keeps training from diverging, and AVX2 and NEON intrinsics accelerate both formats. Trained and deployed this way, a 1.5B recognition model is up to $4.32\times$ faster on a single thread than FP16 and $2.9\times$ smaller, and transcribes faster than real time on one thread of either an Apple M4 or an Intel Core i7-13700, for a 0.43% increase in recognition error across ten corpora relative to the FP16 baseline. The allocation transfers to other recognition and generation models with comparable gains.

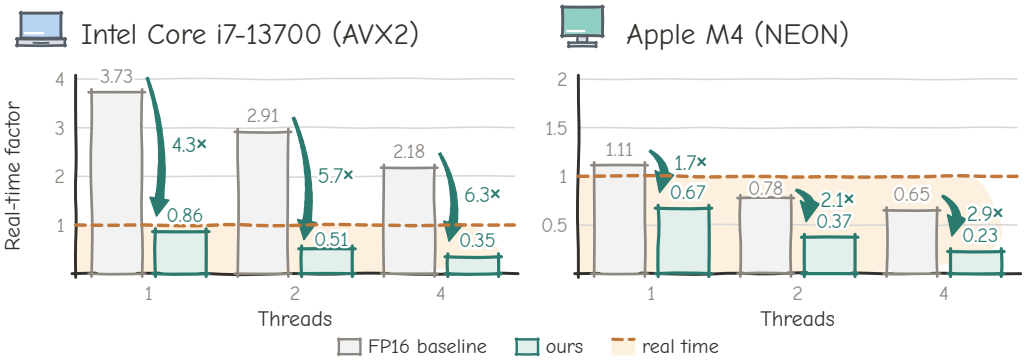

Figure 1: A 1.5B recognition model, quantized by module role, transcribes faster than real time on one thread of either consumer CPU. Grey is the same graph in FP16, our speedup on each arrow.

1 INTRODUCTION

Speech recognition has converged on one design: a non-autoregressive acoustic stack of convolutional and linear layers encodes the waveform into a low-rate latent sequence, and an autoregressive transformer transcribes that sequence (Figure 2) (Radford et al., 2023; Chu et al., 2023; Barrault et al., 2023; Peng et al., 2026). Speech generation applies the same two modules in the opposite order, a language model emitting latents that a feed-forward decoder or a flow-matching or diffusion sampler (Lipman et al., 2023; Ho et al., 2020; Sun et al., 2024) renders into a waveform (Peng et al., 2025; Hu et al., 2026; Zhou et al., 2026), so what suits a recognizer carries over to its mirror.

Most speech recognition runs in the cloud today. That sends the audio off the device, among the most sensitive data a user produces, and leaves latency to the network. Running recognition on the device itself removes both problems. An edge device, though, guarantees only a CPU, not a GPU or an NPU, with one to two orders of magnitude less bandwidth than a datacentre accelerator and no matrix unit. The target is not merely "runs on a CPU" but real time at few enough threads to leave the device free for other applications (Figure 1).

Meeting that target on a CPU means moving fewer bytes, which is what quantization does. One bit-width, though, cannot serve both halves of a recognizer. The transformer is limited by the weight

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

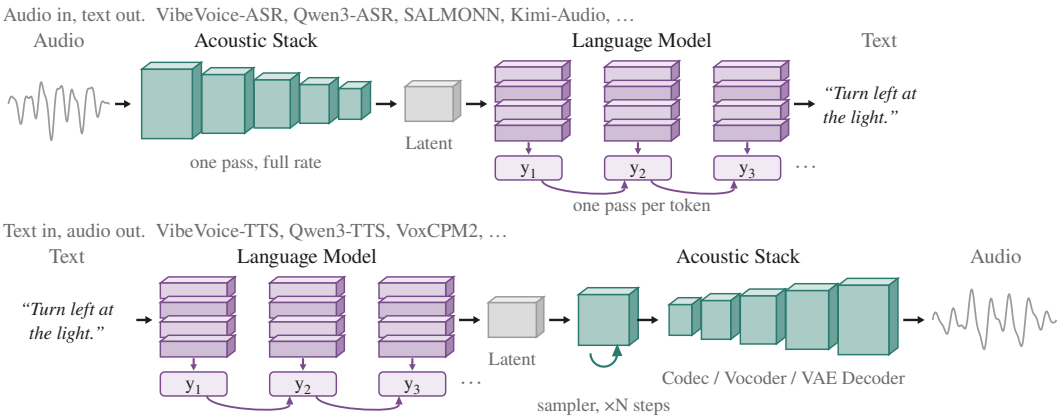

Figure 2: A recognition model runs a non-autoregressive acoustic stack at full rate and then an autoregressive language model. Generation applies the same two modules in the opposite order.

bytes it streams per token, the acoustic stack by activations that grow with audio duration. We therefore allocate precision by module role. The transformer takes ternary W2A8 in the style of BitNet b1.58 (Ma et al., 2024; 2025). The acoustic stack tolerates much less rounding, so there we keep eight bits and cut volume rather than width. Weights, activations and every buffer stay in INT8, never returning to floating point. Annealing fake-quantization from zero to full strength keeps training stable, and `ggml` kernels (Gerganov, 2023a) map both dot products onto the widest AVX2 and NEON integer instructions, fusing chains so intermediate tensors never reach memory. Our contributions are:

- **A traffic model that decides precision module by module.** We analyze a recognizer's memory traffic, find a weight-bound transformer and an activation-bound acoustic stack, and quantize each accordingly, W2A8 and full-pipeline INT8, for $2.9\times$ compression.
- **Progressive quantization-aware training for a heterogeneous format.** Annealing fake-quantization from zero to full strength trains VibeVoice-ASR in both formats at once, and holds the cost to 0.43% absolute error over ten corpora.
- **Kernels that enable real-time recognition on a CPU.** AVX2 and NEON kernels keep every buffer in INT8 and fuse memory-bound operators into each multiplication, reaching real time on a single thread. The scheme transfers to other speech models.

2 Related Work

LM-based speech recognition. Whisper made the encoder–decoder transformer the default recognizer (Radford et al., 2023); later systems swap its decoder for a pretrained language model (Chu et al., 2023; 2024; An et al., 2024; Peng et al., 2026). Speech generation mirrors the arrangement, a language model driving an acoustic decoder (Peng et al., 2025; Zhou et al., 2026). Both families run to billions of FP16 parameters and are evaluated on accelerators. Efficiency-oriented recognizers instead drop the language model for CTC or a transducer (Graves, 2012; Graves et al., 2006) or shrink the encoder (Gulati et al., 2020; Yao et al., 2024; Jeffries et al., 2024; NVIDIA NeMo Team, 2024), forgoing the accuracy it provides.

Quantization. Post-training methods for LLMs focus on weights, since batch-1 decoding is weight-bound: GPTQ and AWQ reach 3–4 bits from a calibration set (Frantar et al., 2023; Lin et al., 2024), and LLM.int8() and SmoothQuant handle the activation outliers that make W8A8 difficult (Dettmers et al., 2022; Xiao et al., 2023). Below four bits generally requires training: BitNet b1.58 matches its FP16 counterpart with ternary weights (Ma et al., 2024; 2025). Assigning different bit widths to different layers is established, searched against a hardware cost model or ranked by Hessian sensitivity (Wang et al., 2019; Dong et al., 2020), for activations as well as weights, but within one architecture. Full-integer inference for convolutional networks dates to Jacob et al. (2018); on an acoustic stack its gain is in activations rather than weights.

CPU inference engines. `ggml`, `llama.cpp` and `whisper.cpp` are the de facto substrate for CPU inference (Gerganov, 2023a;b;c), and with `sherpa-onnx` (k2-fsa, 2023) they run the recognizers a CPU can serve today, Ultravox (Fixie AI, 2024) among them, the baselines we measure against. Their k-quant formats, like bitnet.cpp's ternary lookup tables (Wang et al., 2024), quantize the weight side only: weights unpacked inside the matrix multiplication, activations quantized per dot product, outputs returned to floating point. That suits a transformer decoding one token at a time, where weights dominate the traffic (MiniCPM Team, 2025). The acoustic half is left unaddressed. No format in these engines keeps a convolutional stack in integer form across operator boundaries.

3 Quantized Inference

A recognition model has two parts (Figure 2): its encoder, an *acoustic stack*—our term for the non-autoregressive modules that run at the full audio rate—and an autoregressive transformer, one token per step. The two parts are limited by different resources, so we give each its own format (Sections 3.1.1–3.1.3) and one set of kernels serves both (Section 3.2).

3.1 Format allocation across modules

Uniform quantization gives both parts the same bit-width; we allocate by what limits each instead. We work the argument through on VibeVoice-ASR (Peng et al., 2026), a ConvNeXt encoder in front of a Qwen-family transformer (Figure 3a), the layout LM-based recognizers share and the model we later train and deploy.

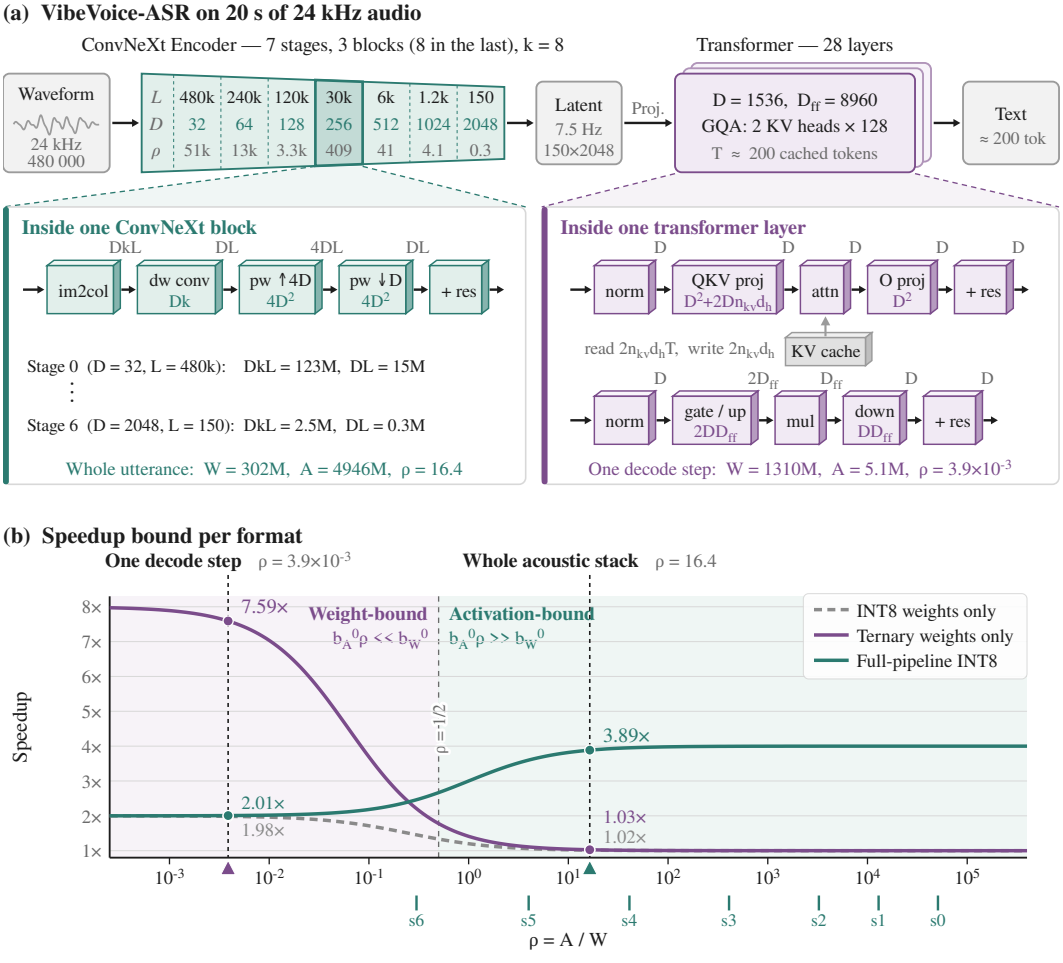

Figure 3: (a) VibeVoice-ASR, with the parameters the analysis uses and the traffic each half moves for 20 s of audio. (b) The speedup bound for the three formats a CPU engine offers, with the two halves at $\rho = 3.9 \times 10^{-3}$ and 16.4, a factor of 4×10^3 apart.

3.1.1 TRAFFIC MODEL AND SPEEDUP BOUND

Traffic per invocation. One invocation of a module reads W weight values, and its operators exchange A activation values across their boundaries, each counted once written and once read. Only boundaries count, so quantizing an activation *inside* a matrix multiplication, as weight-only formats do, leaves b_A unchanged. At b_W bits per weight and b_A bits per activation it moves

$$Q(b_W, b_A) = b_W W + b_A A \tag{1}$$

bits, and on a CPU without a matrix unit this traffic largely sets the time: a matrix–vector product performs one FLOP per FP16 byte, and the acoustic stack moves almost all of its traffic in stages with many frames and few channels, whose intensity is of the same order (Appendix A).

Two regimes. With time $\propto Q$ and $\rho = A/W$, the activation values a module moves per weight value, a change of format speeds it up by at most

$$S(\rho) \;=\; \frac{Q(b_W^0,\, b_A^0)}{Q(b_W,\, b_A)} \;=\; \frac{b_W^0 + b_A^0\,\rho}{b_W + b_A\,\rho}, \qquad S(0) = \frac{b_W^0}{b_W}, \qquad \lim_{\rho\to\infty} S(\rho) = \frac{b_A^0}{b_A}, \tag{2}$$

monotone in ρ between the two limits. A module is *weight-bound* when $b_A^0\rho \ll b_W^0$ and *activation-bound* when $b_A^0\rho \gg b_W^0$, crossing over at $\rho = b_W^0/b_A^0$. Our baseline is FP16 weights with FP32 arithmetic, $b_W^0 = 16$ and $b_A^0 = 32$, as `ggml` stores and computes them. Allocating precision is then placing each module on this one axis (Figure 3b).

The transformer sits at $\rho \ll 1$. For a decoder layer with hidden size D, n_{kv} key/value heads of width d_h, feed-forward width D_{ff} and context length T, one decode step reads W_{layer} weights and exchanges A_{layer} activation values,

$$\begin{aligned} W_{\text{layer}} &= 2D^2 + 2Dn_{kv}d_h + 3DD_{\text{ff}}, \\ A_{\text{layer}}(T) &= 2n_{kv}d_hT + 2n_{kv}d_h + 16D + 6D_{\text{ff}}, \end{aligned} \tag{3}$$

the first activation term being the KV cache read, the second the keys and values the step appends, and the rest the vectors that cross operator boundaries inside it (Appendix A). The weight term is quadratic in D, the activation term linear in D and T, so ρ stays small unless the context is enormous: for VibeVoice-ASR it is 3.9×10^{-3} at the context a 20 s utterance occupies.

The acoustic stack sits at $\rho \gg 1$. The stack is shallow but sees the signal at full rate. For a ConvNeXt block (Liu et al., 2022) with D channels, depthwise kernel width k and sequence length L, the weights are the depthwise mixer and a $4\times$-expansion feed-forward network; the tensors crossing boundaries are the `im2col` expansion and the pointwise layers' inputs and outputs:

$$W = Dk + 8D^2, \quad A = 2DkL + 12DL, \quad \text{hence} \quad \rho_{\text{block}} = \frac{(2k+12)\,L}{k+8D} \;\approx\; \frac{2k+12}{8}\cdot\frac{L}{D}. \tag{4}$$

A block's ρ is thus its aspect ratio L/D up to a constant. Each stage doubles D while downsampling L, so ρ falls monotonically down the stack, graded rather than uniformly activation-bound, from 5.1×10^4 at VibeVoice-ASR's first encoder stage to below one at the last, and 16.4 over the whole stack (Appendix A).

Format allocation. The two modules fall on opposite sides of the crossover, so no single format serves both. The traffic decides which operand to narrow, and accuracy decides how far. Round-to-nearest and GPTQ leave ternary weights unusable, while quantization-aware training returns them to nearly FP16 accuracy. On the stack eight-bit activations cost almost no accuracy beyond the weights, while four bits destroy the output (Appendix B). We therefore give the transformer ternary weights (Section 3.1.3) and the stack full-pipeline INT8 (Section 3.1.2).

Equation 2 then prices each choice. Ternary weights give $7.59\times$ on the transformer against $1.01\times$ for narrower activations, INT8 activations $3.88\times$ on the stack against $1.03\times$. The allocation follows from architecture, not this checkpoint, so models built the same way inherit it.

3.1.2 FULL-PIPELINE INT8 IN THE ACOUSTIC STACK

The stack is bound by activations, so they must be narrow everywhere. Dynamic W8A8 quantizes each matrix multiplication's inputs but returns its output to FP32, leaving every surrounding operator at 32 bits. We keep every tensor between operators in INT8 instead. The producing kernel's epilogue

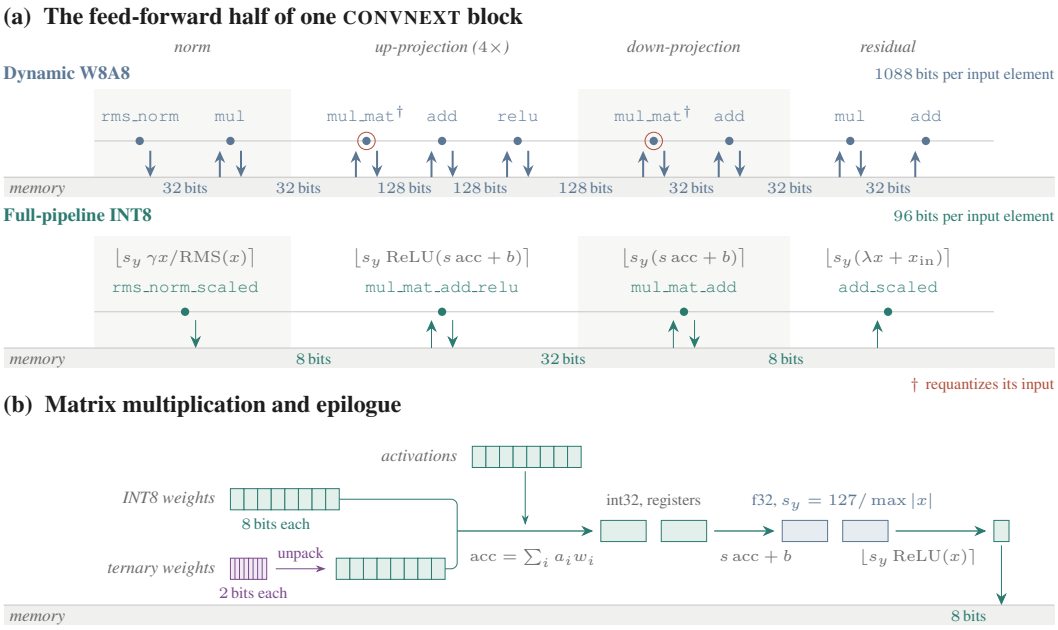

Figure 4: (a) The feed-forward half of a CONVNEXT block, nine FP32 operators against four fused INT8 ones. Arrows carry the bits moved per input element, 1088 in total against 96. (b) Both formats reach one int32 accumulator, whose epilogue rescales, adds bias, applies ReLU and requantizes.

applies bias, nonlinearity and output scale, so the next operator reads 8 bits, and operators that only rescale or add fold into their producer. Nine FP32 operators become four in a block's feed-forward half (Figure 4). Five fused operators cover the encoder (`im2col_asym`, `mul_mat_add`, `mul_mat_add_relu`, `add_scaled` and `rms_norm_scaled`).

Weights stay at eight bits. They are 3% of the stack's traffic (Table 7), so taking them to two bits raises the bound of Equation 2 only from 3.88× to 4.06×. That is too little to pay for unpacking in every kernel, or for the accuracy cost, which Appendix B measures.

3.1.3 TERNARY WEIGHTS IN THE TRANSFORMER

At the transformer's ρ the only quantity worth reducing is b_W, and the floor for a trainable format is ternary, which no post-training method reaches (Appendix B). We adopt the BitNet b1.58 formulation (Ma et al., 2024), which constrains weights to $\{-1, 0, +1\}$, and pack four of them to a byte. A weight then costs 2 bits against 16, an 8× reduction in exactly the traffic that sets step time.

Activations are INT8 only where they are matrix-multiplication operands; the residual stream stays in FP32, because narrowing it raises the bound of Equation 2 only from 7.59× to 7.94×. Integer operands are required in any case, and INT8 is what the acoustic stack already produces.

3.2 SIMD KERNEL DESIGN

Both formats reduce to the same integer dot product (Figure 4b). On AVX2 `_mm256_sign_epi8` moves the weight sign onto the activation so that `_mm256_maddubs_epi16` can multiply unsigned–signed byte pairs, and `_mm256_madd_epi16` widens the products into an int32 accumulator; NEON reaches the same accumulator with `sdot`, or `smull` and `sadalp` on ARMv8.0 cores that lack it. One 256-bit load holds 32 INT8 weights or 128 packed ternary codes, and half as many on NEON's 128-bit registers. Ternary weights only add a prologue that masks the four two-bit codes out of each byte, so one code path per instruction set serves both formats. Kernels tile the operands and parallelize over whichever side has room (weight rows when the activation is a single vector, activation columns otherwise), so the reused side stays in cache. Fusion removes the round trips of Figure 4a. Rescale, bias, nonlinearity and rounding happen in the producing kernel, which reads s_y off the accumulator it holds, so no intermediate returns to floating point.

4 TRAINING OPTIMIZATION

Ternary weights and full-pipeline INT8 both have to be learned rather than calibrated. Applied after training, the ternary half emits no words at all and the INT8 half raises error on every corpus, while trained together the two recover the accuracy of the floating-point checkpoint they started from (Appendix B). We train VibeVoice-ASR, starting with its two ConvNeXt encoders, an acoustic one trained on reconstruction and a semantic one supervised through a frozen LM teacher, and moving on to the transformer. The encoder needs fake-quantization nodes on its weights and its activations, since it carries a byte-wide tensor at every operator boundary, while the transformer needs them on its weights alone. We train through them with the straight-through estimator (Bengio et al., 2013) and per-tensor scales in the standard formulation (Nagel et al., 2021; Esser et al., 2020). Enabling all of these nodes at full strength from the start does not train, the loss staying above 3.1 (Figure 5b). Neither half adapts to quantization imposed all at once.

We therefore bring quantization in gradually, so that the two halves converge. Every quantized tensor is a blend of its full-precision and its fake-quantized value,

$$\hat{x} = (1-\alpha)\,x + \alpha\,\mathrm{FakeQuant}(x),$$
$$\mathrm{FakeQuant}(x) = s\,\mathrm{clamp}(\lfloor x/s \rceil, q_{\min}, q_{\max}), \qquad s = \frac{\max|x|}{q_{\max}}. \tag{5}$$

The scale s is recomputed at every step from the tensor it quantizes, so it follows the weights and activations as they move. INT8 tensors take $q_{\max} = 127$, one scale for a whole weight tensor and one for each block of an activation's last two axes. Ternary weights take $q_{\max} = 1$ and $s = \mathrm{mean}|W|$. One global α is scheduled in three stages, and rises linearly from 0 to 1 in the middle one. At $\alpha = 0$ we replace GELU with ReLU throughout the encoder and finetune. Non-negative activations make the epilogue cheaper, a clamp at zero rather than a transcendental per element, and the finetune recovers the accuracy of the GELU encoder, so the swap costs nothing of its own before quantization begins. The ramp $\alpha : 0 \to 1$ then lets scales and weights co-adapt while gradients still flow through a mostly linear path, and at $\alpha = 1$ the network trains fully quantized, reaching 0.92 where the unstaged run stays above 3.1.

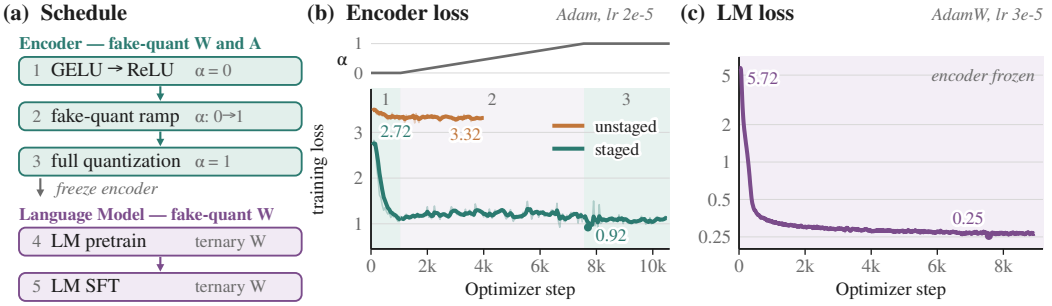

Figure 5: Quantization-aware training of VibeVoice-ASR in both of its formats. (a) the five stages in order, and what each one fake-quantizes. (b) the semantic encoder's loss with the staged ramp against the same encoder without it. (c) the transformer's loss once the encoder is frozen.

The encoder is then frozen and the transformer trained against it, first pretraining and then supervised finetuning, so it sees the encoder's output exactly as deployment produces it and not an output that is still changing underneath it. This stage fake-quantizes weights only: I2_S inference keeps the residual path and the transformer's activations in high precision (Section 3.1.3), so there is nothing for an activation node to model.

5 EXPERIMENTS

5.1 SETUP

Hardware. One server socket, an AMD EPYC 7V13 (24 cores, AVX2, Linux), and two consumer devices, an Apple M4 (4P + 6E cores, NEON, macOS) and an Intel Core i7-13700 (8P + 8E cores, AVX2, Windows 11). CPU only, no GPU involved anywhere.

Protocol. RTF $= t_{\text{compute}}/t_{\text{audio}}$, so RTF < 1 means the system keeps up with the signal. The baseline is always the same graph with FP16 weights and FP32 accumulation, on the same code path, schedule and thread pool, so a comparison isolates the formats. Warm-up, repetitions, file mapping and the benchmark inputs are in Appendix C.

Models. Table 1 lists the five deployed systems, which module of each gets which format, and the resulting file sizes. VibeVoice-ASR is the one trained under the schedule of Section 4; the other four are converted from released checkpoints without retraining.

Benchmarks. Recognition accuracy uses ten corpora of read, conversational and meeting speech in five languages, from MLC-SLM (MLC-SLM Challenge Organizers, 2025) to LibriSpeech (Panayotov et al., 2015); Appendix C lists them all.

Table 1: The allocation on five systems, two recognition and three generation. "LM" is the autoregressive transformer, ternary weights carry INT8 activations (W2A8), and sizes sum every file loaded at inference.

System	Task	Precision allocation		Size (GiB)		
		I2_S	Full-pipeline INT8	FP16	Ours	Ratio
VibeVoice-ASR[1]	ASR	Qwen2.5 LM*	VAE Encoder	4.62	1.58	2.9
Qwen3-ASR[2]	ASR	Qwen3 LM*, AuT Layers	Conv Downsampler	3.80	0.66	5.8
VibeVoice-TTS[3]	TTS	Qwen2.5 LM	Diffusion Head, Acoustic VAE	4.40	1.25	3.5
Qwen3-TTS[4]	TTS	Talker, Code Predictor	Tokenizer Decoder	1.92	1.01	1.9
VoxCPM2[5]	TTS	TSLM, RALM, LocDiT	VAE Decoder	4.35	0.82	5.3

*The token embedding stays at Q6_K rather than ternary, as in BitNet b1.58.
[1]Peng et al. (2026) [2]Shi et al. (2026) [3]Peng et al. (2025) [4]Hu et al. (2026) [5]Zhou et al. (2026)

5.2 End-to-end speed

On the EPYC (Table 2) the FP16 graph needs eight threads to reach real time, at an RTF of 0.836, while quantized it is real time on two and reaches 0.272 on eight. The ratio grows with thread count, because adding cores makes bandwidth scarcer and the format moving fewer bytes gains more.

Table 2: End-to-end real-time factor of VibeVoice-ASR on three machines. Values below real time are set in bold, and "Ratio" is FP16 over ours at the same thread count.

Machine	Format	Real-time factor ↓					
		1T	2T	3T	4T	6T	8T
EPYC 7V13 (AVX2)	FP16	3.617	2.057	1.460	1.277	1.017	**0.836**
	Ours	1.580	**0.844**	**0.554**	**0.455**	**0.327**	**0.272**
	Ratio	2.29	2.44	2.64	2.81	3.11	3.07
Apple M4 (NEON)	FP16	1.114	**0.777**	**0.702**	**0.651**	**0.618**	**0.632**
	Ours	**0.673**	**0.374**	**0.262**	**0.227**	**0.245**	**0.217**
	Ratio	1.66	2.08	2.68	2.87	2.52	2.91
Core i7-13700 (AVX2)	FP16	3.728	2.912	2.441	2.177	1.873	2.200
	Ours	**0.863**	**0.510**	**0.408**	**0.348**	**0.291**	**0.278**
	Ratio	4.32	5.71	5.98	6.26	6.44	7.91

Both consumer CPUs are real time on a single thread, 0.673 on the M4 over NEON and 0.863 on the i7 over AVX2, and the M4 stops gaining past four threads, regressing at six where the pool starts scheduling onto efficiency cores. On one thread quantization gives 1.66× on the M4 and 4.32× on the i7, whose FP16 baseline never crosses real time at any thread count; at eight the two reach 2.91 and 7.91×, the latter inflated by the i7's own FP16 baseline regressing past six threads. The other four systems gain 2.20–2.70× on one EPYC thread and 2.29–3.05× on eight, where all of them reach real time. The strategy is not specific to VibeVoice-ASR. Since ρ decides the allocation, any model built from an acoustic stack and an autoregressive transformer inherits it, and these four did so without retraining. Appendix D sweeps all five systems over every machine and thread count.

Table 3 puts three CPU engines and ours on the same clip and machine, each in the fastest quantized format it offers. We are 2.1–2.5× faster than `whisper.cpp` and reach real time at two threads

where it needs four; the `sherpa-onnx` graph needs six, and so does Ultravox, whose language model is quantized but whose audio encoder is not and which spends 85–90% of its time in that encoder.

Table 3: Three CPU engines and ours on one 30 s clip on the EPYC 7V13, greedy, median of three warm runs with model load excluded. The three baselines quantize weights only. Size is every file loaded at inference. Bold is real time.

Engine	Model	Format	Size (GB)	Real-time factor ↓					
				1T	2T	3T	4T	6T	8T
Ours	VibeVoice-ASR	Ternary + INT8	1.58	1.302	**0.685**	**0.479**	**0.378**	**0.274**	**0.228**
`whisper.cpp`	Whisper large-v3-turbo	q4_k	0.44	3.191	1.641	1.167	**0.902**	**0.617**	**0.472**
`sherpa-onnx`	Whisper large-v3	int8	1.65	1.616	1.210	1.114	1.023	**0.928**	**0.919**
`llama.cpp`	Ultravox-1B	Q4_K_M	2.03	4.954	2.549	1.756	1.328	**0.925**	**0.720**

5.3 Module-level speedups

Figure 6 splits the recognition model into its three modules. The encoder holds 69% of FP16 time, so quantizing the transformer alone bounds the gain at 1.4×. The two formats also scale with threads in opposite directions. The encoder gains 2.16× at one thread and 3.37× at eight, since adding cores exhausts bandwidth sooner for the format that moves four times the bytes, while ternary prefill moves from 2.74× to 2.46×, a batched GEMM already near compute-bound. Neither reaches the bound of Section 3.1.1. The encoder realizes 3.37 of 3.88×, ternary decode only 2.77 of 7.59×, and three effects separate the two. Equation 2 prices bytes at a fixed dataflow, while our kernels fuse operators and cut A itself (Section 3.2), so bound and measurement are not over the same graph. Unpacking ternary weights leaves a compute floor that 8× less traffic cannot cross. And tensors too small to amortize a per-call cost pay it in either format. Appendix E reports what the allocation earns module by module in every system. Which module dominates differs in each, so a scheme that accelerates only transformers or only convolutions cannot generalize.

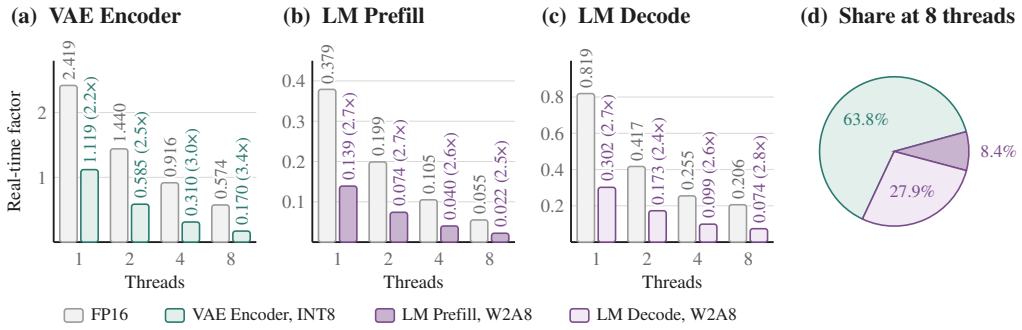

Figure 6: (a)–(c) VibeVoice-ASR by module on the EPYC 7V13, FP16 against the allocated format, ratio in brackets, a vertical scale per panel. (d) Share of quantized time at eight threads.

5.4 Activation handling and peak memory

Dynamic W8A8 returns every intermediate to FP32, so the additions, normalizations and layout changes around each GEMM still move 32 bits per element; full-pipeline INT8 carries them at 8. Over identical INT8 weights (Table 4) that is worth 1.17× end to end at one thread and 1.34× at eight on 15 s of audio.

Peak memory matters more, since it decides whether the model runs at all on a memory-limited device. The compute arena, scratch rather than weights, shrinks 4.3× against W8A8 and 6.2× against FP32, and peak resident set on 15 s falls from 15.1 to 5.0 GB. We also provide a streaming mode that holds the input window to about three seconds, so peak memory follows the window rather than the utterance. Its resident set is near 3 GB, within most consumer machines.

5.5 Accuracy

Table 5 sets our 1.5B model against GPU reference systems of comparable size and against a floating-point control; Appendix C gives the corpora and the scoring rules. The comparison that

Table 4: Activation handling over identical INT8 weights. "Arena" is compute-scratch bytes per PCM sample at 24 kHz, the one column independent of duration; "RSS" is peak resident set.

Variant	Duration (s)	End-to-end RTF ↓			Memory	
		1T	4T	8T	Arena (B)	RSS (MB)
Full-pipeline INT8 (ours)	1	2.091	0.677	0.407	8915	2320
	5	1.499	0.458	0.264		3090
	15	1.404	0.403	0.232		5025
Dynamic W8A8	1	2.252	0.784	0.493	37960	2993
	5	1.707	0.552	0.338		6461
	15	1.649	0.508	0.311		15138
FP32	1	2.886	1.068	0.711	55120	4046
	5	2.521	0.982	0.648		9047
	15	2.576	0.975	0.636		21560

Table 5: Recognition accuracy: character error rate on MLC-KO and Fleurs-zh, word error rate elsewhere. "—" marks a language a system does not claim. Each reference system runs on GPU in the precision its framework loads by default (Appendix C).

Corpus	WER / CER (%) ↓						
	Reference systems (GPU)					Ours (CPU)	
	Parakeet[1]	Whisper[2]	Turbo[3]	SenseVoice[4]	FunASR[5]	FP16	Quantized
MLC-EN	8.45	9.36	9.45	11.21	9.75	8.56	8.88
MLC-FR	60.53	27.48	27.35	—	—	20.03	20.98
MLC-IT	16.13	16.50	17.33	—	—	17.90	17.32
MLC-KO	—	12.81	14.09	14.59	—	12.32	13.13
Fleurs-en	6.61	5.64	6.07	9.11	6.34	6.27	6.56
Fleurs-zh	—	7.48	8.03	7.61	6.42	7.21	7.53
Libri-clean	1.93	1.91	2.00	3.41	1.73	2.02	2.08
Libri-other	3.60	3.58	3.92	7.40	4.31	4.10	4.73
TED-LIUM	2.78	3.86	3.49	4.19	3.49	3.16	3.47
AMI-ihm	21.09	23.32	23.51	24.95	24.97	24.20	25.35

[1] Parakeet-TDT-0.6B-v3 (NVIDIA NeMo Team, 2024) [2] Whisper large-v3 (Radford et al., 2023) [3] Whisper large-v3-turbo (Radford et al., 2023) [4] SenseVoice-Small (An et al., 2024) [5] Fun-ASR-Nano-2512 (Gao et al., 2023)

matters is the last two columns, one checkpoint executed quantized or in FP16. Quantization costs 0.43% averaged over the ten corpora, worst case $+1.15\%$ on AMI-ihm, and on MLC-IT the quantized model is 0.58% better—for a model $2.9\times$ smaller and $2.3\times$ faster. Over the same ten our FP16 checkpoint is 0.62% ahead of Whisper large-v3 and the quantized model 0.19% ahead, on a CPU rather than a GPU.

6 Conclusion and Limitations

Our target is real-time speech recognition on a device CPU. We find that LM-based recognition decomposes into a weight-bound autoregressive transformer and an activation-bound acoustic encoder, and that the resource limiting each sets its precision. We allocate W2A8 to the transformer and full-pipeline INT8 to the acoustic stack, train both under an annealed fake-quantization schedule and serve them with fused AVX2 and NEON kernels over one shared integer dot product. Returning no intermediate to floating point shrinks the compute arena $6.2\times$ against FP32. Across several architectures it gives up to a $4.32\times$ single-thread speedup and a $5.8\times$ size reduction; on consumer hardware it reaches real time on one thread at 0.43% more error over ten corpora than the same checkpoint in floating point. The allocation carries to other recognition and generation systems of the same shape, evidence that the strategy generalizes beyond one architecture.

Limitations. (1) Only VibeVoice-ASR went through the schedule of Section 4, so for the other four systems we report speed and size but no accuracy. (2) Some modules fall short of the predicted gain: near the crossover of Section 3.1.1 a module has little traffic left to remove, and once a kernel turns compute-bound the bytes it saves no longer translate into time.

AI USE STATEMENT

We used a large language model as a writing assistant for this manuscript: drafting and copy-editing prose from author-supplied outlines, measurement logs and result tables, and assembling the bibliography. Every number, claim and citation in the paper was checked by the authors against their own experimental records and against the cited sources, and the text was revised by the authors throughout. Generative AI was not used to generate the research idea, the method, or any experimental result: the quantization scheme, the training schedule, the kernels and all measurements are the authors' own work.

Generative AI was additionally used as a coding assistant in parts of the accompanying implementation. All such code was read, compiled and validated by the authors against the corresponding reference implementations before any number in this paper was taken from it. We have reviewed all AI-assisted work and take responsibility for the final content of this paper, including text, claims and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

The motivation of this work is a privacy property: running recognition and generation on the device keeps speech—among the most sensitive data a user produces—from leaving it. We see no ethical concern in the efficiency contribution itself. Two adjacent risks are worth naming. First, all three generation systems we quantize are released checkpoints capable of voice cloning from a short prompt, and making them run on commodity CPUs lowers the cost of misuse as much as it lowers the cost of legitimate use; we add no capability to these models, and we encourage deployments to pair them with the provenance and watermarking measures their authors recommend. Second, the recognition accuracy we report is not uniform across languages and conditions—our error rate in Table 5 ranges from 2.08% to 25.35%—so a deployment that treats the English number as representative would serve some users considerably worse than others. All corpora used here are public research datasets, used under their own licences and for evaluation only; no new speech data was collected, and no human subjects were involved.

REPRODUCIBILITY STATEMENT

The measurement protocol—hardware, repetitions, warm-up, file mapping and how the real-time factor is defined—is stated in Section 5.1 and Appendix C, and the quantization formats and kernel structure in Sections 3.1.2–3.2 in enough detail to reimplement them. The precision assignment for all five architectures is tabulated per module in Table 1, and the training schedule, including the three stages and the blending rule of Equation 5, in Section 4, whose wall-clock cost is in Appendix F. The accuracy experiments use one corpus list, one slicing and normalization rule, and the same loop-detection threshold and generation cap across the eight systems compared. Appendices D and E report the complete thread sweeps behind every speedup quoted in the paper, so that each aggregate number can be traced to the timings it came from. The quantization and conversion scripts and the kernel and inference implementations are included in the supplementary material; the accuracy experiments are reproducible from the corpus list and the slicing, normalization and decoding rules of Appendix C rather than from released code.

REFERENCES

Keyu An, Qian Chen, Chong Deng, Zhihao Du, Changfeng Gao, et al. FunAudioLLM: Voice understanding and generation foundation models for natural interaction between humans and LLMs. *arXiv preprint arXiv:2407.04051*, 2024.

Loïc Barrault, Yu-An Chung, Mariano Coria Meglioli, David Dale, Ning Dong, et al. SeamlessM4T: Massively multilingual and multimodal machine translation. *arXiv preprint arXiv:2308.11596*, 2023.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Jean Carletta, Simone Ashby, Sebastien Bourban, Mike Flynn, Mael Guillemot, Thomas Hain, Jaroslav Kadlec, Vasilis Karaiskos, Wessel Kraaij, Melissa Kronenthal, Guillaume Lathoud, Mike Lincoln, Agnes Lisowska, Iain McCowan, Wilfried Post, Dennis Reidsma, and Pierre Wellner. The AMI meeting corpus: A pre-announcement. In *Machine Learning for Multimodal Interaction*, 2005.

Yunfei Chu, Jin Xu, Xiaohuan Zhou, Qian Yang, Shiliang Zhang, Zhijie Yan, Chang Zhou, and Jingren Zhou. Qwen-Audio: Advancing universal audio understanding via unified large-scale audio-language models. *arXiv preprint arXiv:2311.07919*, 2023.

Yunfei Chu, Jin Xu, Qian Yang, Haojie Wei, Xipin Wei, Zhifang Guo, Yichong Leng, Yuanjun Lv, Jinzheng He, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen2-Audio technical report. *arXiv preprint arXiv:2407.10759*, 2024.

Alexis Conneau, Min Ma, Simran Khanuja, Yu Zhang, Vera Axelrod, Siddharth Dalmia, Jason Riesa, Clara Rivera, and Ankur Bapna. FLEURS: Few-shot learning evaluation of universal representations of speech. In *Proceedings of the IEEE Spoken Language Technology Workshop*, 2023.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. LLM.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*, 2022.

Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. HAWQ-V2: Hessian aware trace-weighted quantization of neural networks. In *Advances in Neural Information Processing Systems*, 2020.

Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. In *Proceedings of the International Conference on Learning Representations*, 2020.

Fixie AI. Ultravox: A fast multimodal LLM for real-time voice, 2024. URL https://github.com/fixie-ai/ultravox.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *Proceedings of the International Conference on Learning Representations*, 2023.

Sanchit Gandhi, Patrick von Platen, and Alexander M. Rush. ESB: A benchmark for multi-domain end-to-end speech recognition. *arXiv preprint arXiv:2210.13352*, 2022.

Zhifu Gao, Zerui Li, Jiaming Wang, Haoneng Luo, Xian Shi, Mengzhe Chen, Yabin Li, Lingyun Zuo, Zhihao Du, Zhangyu Xiao, and Shiliang Zhang. FunASR: A fundamental end-to-end speech recognition toolkit. In *Proceedings of Interspeech*, 2023.

Georgi Gerganov. ggml: Tensor library for machine learning, 2023a. URL https://github.com/ggml-org/ggml.

Georgi Gerganov. llama.cpp: LLM inference in C/C++, 2023b. URL https://github.com/ggml-org/llama.cpp.

Georgi Gerganov. whisper.cpp: Port of OpenAI's Whisper model in C/C++, 2023c. URL https://github.com/ggml-org/whisper.cpp.

Alex Graves. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*, 2012.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the International Conference on Machine Learning*, 2006.

Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruoming Pang. Conformer: Convolution-augmented transformer for speech recognition. In *Proceedings of Interspeech*, 2020.

François Hernandez, Vincent Nguyen, Sahar Ghannay, Natalia Tomashenko, and Yannick Estève. TED-LIUM 3: Twice as much data and corpus repartition for experiments on speaker adaptation. In *Speech and Computer (SPECOM)*, 2018.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, 2020.

Hangrui Hu, Xinfa Zhu, Ting He, Dake Guo, Bin Zhang, Xiong Wang, Zhifang Guo, Ziyue Jiang, Hongkun Hao, Zishan Guo, Xinyu Zhang, Pei Zhang, Baosong Yang, Jin Xu, Jingren Zhou, and Junyang Lin. Qwen3-TTS technical report. *arXiv preprint arXiv:2601.15621*, 2026.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

Nat Jeffries, Evan King, Manjunath Kudlur, Guy Nicholson, James Wang, and Pete Warden. Moonshine: Speech recognition for live transcription and voice commands. *arXiv preprint arXiv:2410.15608*, 2024.

k2-fsa. sherpa-onnx: Speech-to-text and text-to-speech with ONNX Runtime, 2023. URL https://github.com/k2-fsa/sherpa-onnx.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of Machine Learning and Systems*, 2024.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *Proceedings of the International Conference on Learning Representations*, 2023.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. The era of 1-bit LLMs: All large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764*, 2024.

Shuming Ma, Hongyu Wang, Shaohan Huang, Xingxing Zhang, Ying Hu, Ting Song, Yan Xia, and Furu Wei. BitNet b1.58 2B4T technical report. *arXiv preprint arXiv:2504.12285*, 2025.

MiniCPM Team. MiniCPM4: Ultra-efficient LLMs on end devices. *arXiv preprint arXiv:2506.07900*, 2025.

MLC-SLM Challenge Organizers. The multilingual conversational speech language model (MLC-SLM) challenge, 2025. Interspeech 2025 challenge; development set used here.

Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021.

NVIDIA NeMo Team. Parakeet: A family of large, powerful and multilingual ASR models, 2024. NVIDIA Technical Blog.

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: An ASR corpus based on public domain audio books. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2015.

Zhiliang Peng, Jianwei Yu, Wenhui Wang, Yaoyao Chang, Yutao Sun, Li Dong, Yi Zhu, Weijiang Xu, Hangbo Bao, Zehua Wang, Shaohan Huang, Yan Xia, and Furu Wei. VibeVoice technical report. *arXiv preprint arXiv:2508.19205*, 2025.

Zhiliang Peng, Jianwei Yu, Yaoyao Chang, Zilong Wang, Li Dong, Yingbo Hao, Yujie Tu, Chenyu Yang, Wenhui Wang, Songchen Xu, Yutao Sun, Hangbo Bao, Weijiang Xu, Yi Zhu, Zehua Wang, Ting Song, Yan Xia, Zewen Chi, Shaohan Huang, Liang Wang, Chuang Ding, Shuai Wang, Xie Chen, and Furu Wei. VibeVoice-ASR technical report. *arXiv preprint arXiv:2601.18184*, 2026.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *Proceedings of the International Conference on Machine Learning*, 2023.

Xian Shi, Xiong Wang, Zhifang Guo, Yongqi Wang, Pei Zhang, Xinyu Zhang, Zishan Guo, Hongkun Hao, Yu Xi, Baosong Yang, Jin Xu, Jingren Zhou, and Junyang Lin. Qwen3-ASR technical report. *arXiv preprint arXiv:2601.21337*, 2026.

Yutao Sun, Hangbo Bao, Wenhui Wang, Zhiliang Peng, Li Dong, Shaohan Huang, Jianyong Wang, and Furu Wei. Multimodal latent language modeling with next-token diffusion. *arXiv preprint arXiv:2412.08635*, 2024.

Jinheng Wang, Hansong Zhou, Ting Song, Shaoguang Mao, Shuming Ma, Hongyu Wang, Yan Xia, and Furu Wei. 1-bit AI infra: Part 1.1, fast and lossless BitNet b1.58 inference on CPUs. *arXiv preprint arXiv:2410.16144*, 2024.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the International Conference on Machine Learning*, 2023.

Zengwei Yao, Liyong Guo, Xiaoyu Yang, Wei Kang, Fangjun Kuang, Yifan Yang, Zengrui Jin, Long Lin, and Daniel Povey. Zipformer: A faster and better encoder for automatic speech recognition. In *Proceedings of the International Conference on Learning Representations*, 2024.

Yixuan Zhou, Guoyang Zeng, Xin Liu, Xiang Li, Renjie Yu, Jiancheng Gui, Jiaheng Wu, Ziyang Wang, Xudong Shen, Runchuan Ye, Zhisheng Zhang, Jiuyang Zhou, Bingsong Bai, Weiyue Sun, Mengyuan Deng, Qundong Shi, Zhiyong Wu, and Zhiyuan Liu. VoxCPM2 technical report. *arXiv preprint arXiv:2606.06928*, 2026.

A TRAFFIC ACCOUNTING

Equation 1 charges b_W bits per weight value a module reads and b_A bits per activation value crossing an operator boundary, counted once on write and once on read; values that stay inside a kernel are free, and a weight-only format changes b_W alone. Every term below is derived symbolically, then evaluated for VibeVoice-ASR on 20 s of 24 kHz audio, the setting of Figure 3(a).

Taking time proportional to Q assumes traffic, not arithmetic, is the limit, which arithmetic intensity tests. Weights applied at N positions perform $2WN$ FLOPs and move $(b_W W + b_A A)/8$ bytes, so with $\rho = A/W$

$$I = \frac{2WN}{(b_W W + b_A A)/8} = \frac{16N}{b_W + b_A \rho} \quad \text{FLOP/byte.} \tag{6}$$

A decode step reuses no weight, $N = 1$, and $\rho_{\mathrm{LM}} \ll 1$ leaves $I \approx 16/b_W$, one FLOP per byte at FP16 and below the ratio of any core of Section 5.1, so the transformer is bandwidth-bound. In the acoustic stack a pointwise weight is reused at every position, $N = L$, and intensity climbs from 4.7 at stage 0 of Table 6 to 130 at stage 5, so the deep stages are compute-bound. Q stays predictive anyway: the low-intensity stages hold most of the activation traffic Equation 2 removes, and INT8 widens the compute axis as well; Section 5.3 measures 3.37× of the 3.88× bound.

A.1 TRANSFORMER LAYER

The weights of a layer are the four attention projections and the three feed-forward ones. Under grouped-query attention Q and O are $D \times D$ while K and V are $D \times n_{kv} d_h$:

$$W_{\mathrm{layer}} = D^2 + 2Dn_{kv}d_h + D^2 + 2DD_{\mathrm{ff}} + DD_{\mathrm{ff}} = 2D^2 + 2Dn_{kv}d_h + 3DD_{\mathrm{ff}}, \tag{7}$$

in the order Q, K and V, O, gate and up, down.

A decode step processes one token, so every tensor inside the layer is a single vector: eight of width D and three of width D_{ff}, each written by one operator and read by the next, plus the keys and values the step appends and the cache it reads back,

$$A_{\mathrm{layer}}(T) = 2\,(8D + 3D_{\mathrm{ff}}) + 2n_{kv}d_h + 2n_{kv}d_h T = 16D + 6D_{\mathrm{ff}} + 2n_{kv}d_h(1 + T). \tag{8}$$

At VibeVoice-ASR's $D = 1536$, $n_{kv}d_h = 256$, $D_{\mathrm{ff}} = 8960$ and 28 layers, a 20 s utterance occupying $T \approx 200$ tokens gives 46.8 M weights and 1.81×10^5 activation values per layer, so 1310 M against 5.1 M over the stack and $\rho_{\mathrm{LM}} = 3.9 \times 10^{-3}$. Feed-forward weights are seven eighths of the total, and the cache read, more than half the activations, is the only term that grows with context; it reaches the weights at $T^\star = (W_{\mathrm{layer}} - 16D - 6D_{\mathrm{ff}} - 2n_{kv}d_h)/2n_{kv}d_h \approx 9.1 \times 10^4$ tokens, some two hours of speech. Two conventions push the true ratio lower: `ggml` holds the cache in FP16, and a step's 180 K activation values fit in L2, so most of A never reaches DRAM. Counting every boundary tensor overstates ρ_{LM}, against our own conclusion.

A.2 ENCODER STAGE

A block is a depthwise convolution of kernel width k, executed as an `im2col` expansion followed by a contraction, then an inverted bottleneck through $4D$, so with the depthwise kernel and the two pointwise matrices on the weight side and the `im2col` output, the pointwise input, the expanded tensor and the block output on the activation side,

$$W_{\mathrm{block}} = Dk + 4D^2 + 4D^2 = Dk + 8D^2, \tag{9}$$

$$A_{\mathrm{block}} = 2\,(DkL + DL + 4DL + DL) = 2DkL + 12DL, \tag{10}$$

whose ratio is the ρ_{block} of Equation 4, equal to $3.5\,L/D$ at $k = 8$; a stage of n_b identical blocks scales both sides and leaves the ratio alone. Table 6 evaluates it per stage. Downsampling doubles D and shortens L, so ρ falls monotonically and spans five orders of magnitude: the weights end up almost entirely in the last stage, the traffic in the first three. The stack still sits at $\rho_{\mathrm{stack}} = 16.4$, far above the crossover at $\rho = 1/2$, so one activation-bound format is right despite that spread. Omitted from A_{block} are the block input, the normalization output and the residual sum, $\approx 6DL$ more, which would raise ρ_{stack} against our own claim.

Table 6: Equation 4 per encoder stage, with n_b blocks each. I is the FP16 arithmetic intensity of Equation 6, evaluated at $N = L$.

Stage	D	L	n_b	Weights (M)	Activations (M)	A/W	I (FLOP/byte)
0	32	480 000	3	0.03	1290	50 909	4.7
1	64	240 000	3	0.10	1290	12 923	9.3
2	128	120 000	3	0.40	1290	3 256	18
3	256	30 000	3	1.6	645	409	37
4	512	6 000	3	6.3	258	41	72
5	1024	1 200	3	25.2	103	4.1	130
6	2048	150	8	268.6	69	0.3	99
Total			26	302	4 946	16.4	23

A.3 BOUNDS

Table 7 substitutes the two ratios into Equation 2 against the FP16-weight, FP32-arithmetic baseline $b_W^0 = 16$, $b_A^0 = 32$, for ternary weights ($b_W = 2$, $b_A = 32$) on the transformer and full-pipeline INT8 ($b_W = b_A = 8$) on the stack. The same format is worth $7.59\times$ on one half of the model and $1.03\times$ on the other, which is the whole content of the allocation.

The same split holds in all five systems: every module that runs one token or one sampler step reads $\rho \leq 2.2 \times 10^{-2}$ and bounds at 6.2 to $7.7\times$ in ternary, every module that runs at the audio rate reads $\rho \geq 1.42$ and bounds at 3.2 to $4.0\times$ in the INT8 pipeline, with nearly two orders of magnitude between the two groups.

Table 7: Traffic and the FP16 speedup bound for all five systems, from Equations 1–2, per step or per 20 s of audio as each row states. "Weights" is the weight share; bold marks the assigned format.

Module	Volume (M)		$\rho = A/W$	Weights	Bound on S		
	W	A			INT8 W	Ternary W	INT8 pipeline
VibeVoice-ASR							
Transformer, decode step	1310	5.1	3.9×10^{-3}	99.2%	1.98	**7.59**	2.01
Acoustic stack, 20 s	302	4946	16.4	3.0%	1.02	1.03	**3.88**
Qwen3-ASR							
Transformer, decode step	1409	20.1	1.4×10^{-2}	97.2%	1.95	**6.70**	2.03
Acoustic stack, 20 s	317	449	1.42	26.1%	1.15	1.30	**3.17**
VibeVoice-TTS							
LM, decode step	1310	4.6	3.5×10^{-3}	99.3%	1.99	**7.63**	2.01
VAE decoder, 20 s	344	5542	16.1	3.0%	1.02	1.03	**3.88**
Qwen3-TTS							
Talker, decode step	440	9.7	2.2×10^{-2}	95.8%	1.92	**6.18**	2.04
Code predictor, one step	79	0.3	4.4×10^{-3}	99.1%	1.98	**7.54**	2.01
Tokenizer decoder, 20 s	104	7257	69.8	0.7%	1.00	1.01	**3.97**
VoxCPM2							
TSLM + RALM, decode step	1699	4.4	2.6×10^{-3}	99.5%	1.99	**7.72**	2.01
CFM LocDiT, one step	208	0.5	2.5×10^{-3}	99.5%	1.99	**7.73**	2.01
VAE decoder, 20 s	46	19252	421	0.1%	1.00	1.00	**4.00**

B BIT-WIDTH SELECTION AND THE NEED FOR TRAINING

We quantize the same VibeVoice-ASR fine-tune after training, without any quantization-aware training, and score 200 utterances per corpus under the pipeline of Appendix C with a 512-token generation limit. The two modules are measured separately, Table 8 varying the language model and Table 9 the encoder, each with the other half at FP16 except in the trained row.

Round-to-nearest fails at ternary weights. Rounding each weight to $\{-1, 0, 1\}$ leaves a model that emits only the punctuation of its output format, so every hypothesis normalizes to an empty string. Four-bit round-to-nearest stays intelligible but still loses accuracy against FP16 (row 2).

GPTQ does not close the gap. Fitted with one scale per output channel against a Hessian from held-out audio, the ternary model still produces unusable text and decodes to the generation limit on every utterance. At four bits GPTQ is worse than round-to-nearest, row 4 reaching 17.42% on LibriSpeech test-clean against 4.54% for row 2.

Activation width in the encoder. Quantizing the weights costs accuracy (row 2), and moving the activations to INT8 as well changes little (row 3), so eight bits are effectively free, while INT4 activations destroy the output (row 4). The instruction set points the same way. Integer SIMD arithmetic is defined on bytes, so a sub-byte value must be widened before the dot product reads it: amortized over a block on the weight side, but repeated in the inner loop for an activation every output channel reads, or paid as a byte-wide copy that gives the arena back. We fix the encoder at INT8 weights and activations and recover the loss with quantization-aware training (row 5).

Table 8: Rows 1–5 vary the LM weight format, the encoder at FP16 and LM activations FP32. Size is the LM GGUF alone, rates capped at 100%, and row 6 the released model, both halves quantized.

#	LM weights	Method	Size (GiB)	LibriSpeech test-clean	TED-LIUM 3	FLEURS Chinese
				WER (%) ↓	WER (%) ↓	CER (%) ↓
Quantization applied after training						
1	FP16	—	3.32	2.24	3.47	10.88
2	Q4_K_M	Round-to-nearest	1.04	4.54	5.65	30.17
3	I2_S	Round-to-nearest	0.92	100.00	100.00	100.00
4	INT4	GPTQ	1.01	17.42	8.66	32.10
5	I2_S	GPTQ	0.92	100.00	100.00	100.00
Quantization-aware training (Section 4)						
6	I2_S	Ours	0.92	2.21	3.61	11.54

Table 9: Rows 1–4 vary the encoder format, with the LM at FP16. Size is the encoder GGUF alone, rates capped at 100%, and row 5 is the released model, both halves quantized.

#	Encoder weights	Encoder activations	Size (GiB)	LibriSpeech test-clean	TED-LIUM 3	FLEURS Chinese
				WER (%) ↓	WER (%) ↓	CER (%) ↓
Quantization applied after training						
1	FP16	FP32	1.31	2.24	3.47	10.88
2	INT8	FP32	0.65	4.22	6.77	29.04
3	INT8	INT8	0.65	4.20	6.26	30.45
4	INT8	INT4	0.65	100.00	100.00	100.00
Quantization-aware training (Section 4)						
5	INT8	INT8	0.65	2.21	3.61	11.54

C MEASUREMENT PROTOCOL

Platforms. The AMD EPYC 7V13 is one socket with 24 cores available to the process, 216 GB of RAM and AVX2 but neither AVX-512 nor VNNI, under Linux. The Apple M4 has 4 performance and 6 efficiency cores, 16 GB of unified memory and NEON, under macOS. The Core i7-13700 has 8 performance and 8 efficiency cores, 32 GB of RAM and AVX2, under Windows 11 built with MinGW GCC. All three run Release builds, and no GPU is involved anywhere.

Timing. Every measurement follows a warm-up pass with model files memory-mapped; both matter, since a cold first pass pays page faults on the compute arena and inflates the apparent encoder speedup severalfold. Each number is the median of 15 runs, and the baseline is the same graph with FP16 weights and FP32 accumulation—same code path, same thread pool. Because every intermediate stays in FP32 either way, an FP16 build reserves the same compute arena as the FP32 row of Table 4, 55137 against 55120 bytes per sample. Every input is 20.0 s of audio and the output length is pinned per system, 70 and 86 tokens for VibeVoice-ASR and Qwen3-ASR and 150, 250 and 125 decoder steps for VibeVoice-TTS, Qwen3-TTS and VoxCPM2, so a ratio compares compute at equal work even where an untrained conversion decodes wrongly.

Baselines. Table 3 is a separate measurement on a 30 s clip, so that a Whisper encoder fills exactly one window, greedy everywhere and with each engine's model-load time subtracted; each number is the median of three runs after a warm-up pass, one process per cell with the thread count set through the engine's own option and `OMP_NUM_THREADS`. We swept all ten formats `whisper.cpp`'s `quantize` tool emits and report the fastest, q4_k at RTF 0.472 on eight threads. The GPU reference systems run in the precision their framework loads by default, bf16 for Fun-ASR-Nano, FP16 for Whisper and FP32 for Parakeet and SenseVoice.

Corpora and scoring. Table 5 covers ten sets: the English, French, Italian and Korean development sets of MLC-SLM (MLC-SLM Challenge Organizers, 2025), FLEURS English and Mandarin (Conneau et al., 2023), both LibriSpeech test splits (Panayotov et al., 2015), TED-LIUM 3 in its ESB form (Hernandez et al., 2018; Gandhi et al., 2022) and the individual-headset condition of AMI (Carletta et al., 2005). One manifest fixes the file and the start and end time of every input, and each system is handed it unchanged; MLC-SLM and AMI are cut at segment boundaries into slices of at most 30 s, long conversational chunks being what drives a decoder into repetition loops. Slices are units of inference, not of scoring: hypotheses are concatenated per recording and every rate is corpus-level, total edits over total reference length. One normalization pipeline runs on both sides, after which English uses Whisper's English normalizer and word-level WER, French and Italian its basic normalizer, and Mandarin and Korean are scored per character.

D END-TO-END PERFORMANCE

Tables 10–14 are the full sweeps behind Section 5.2, over every system, machine and thread count. On the EPYC we also quantize each half of the allocation on its own, the acoustic half or the transformer, to show how the two combine. On the M4, six threads are slower than four in every quantized system, since the chip has only four performance cores.

Where the gain is small. The two smallest ratios on the M4 are 0.98–$1.45\times$ and 0.97–$1.90\times$, both marginally slower than FP16 at one thread. A single M4 core has enough bandwidth to keep an FP16 graph fed, so there is little traffic left for a narrower format to remove. Adding cores makes bandwidth scarce again, which is why the ratios grow. On NEON the AuT layers, whose ρ sits just above the crossover, run at 0.8 to $1.2\times$, adding nothing to what the ternary decode gains.

Table 10: VibeVoice-ASR: end-to-end RTF against thread count.

Machine	Quantized	Real-time factor ↓					
		1T	2T	3T	4T	6T	8T
EPYC 7V13	None (FP16)	3.617	2.057	1.460	1.277	1.017	**0.836**
	+ INT8 stack	2.283	1.223	**0.844**	**0.685**	**0.501**	**0.435**
	+ Ternary transformer	2.540	1.543	1.139	**0.952**	**0.737**	**0.629**
	All	1.580	**0.844**	**0.554**	**0.455**	**0.327**	**0.272**
	Ratio	2.29	2.44	2.64	2.81	3.11	3.07
Apple M4	None (FP16)	1.114	**0.777**	**0.702**	**0.651**	**0.618**	**0.632**
	All	**0.673**	**0.374**	**0.262**	**0.227**	**0.245**	**0.217**
	Ratio	1.66	2.08	2.68	2.87	2.52	2.91
Core i7-13700	None (FP16)	3.728	2.912	2.441	2.177	1.873	2.200
	All	**0.863**	**0.510**	**0.408**	**0.348**	**0.291**	**0.278**
	Ratio	4.32	5.71	5.98	6.26	6.44	7.91

Table 11: Qwen3-ASR: end-to-end RTF against thread count.

Machine	Quantized	Real-time factor ↓					
		1T	2T	3T	4T	6T	8T
EPYC 7V13	None (FP16)	1.680	**0.902**	**0.650**	**0.535**	**0.404**	**0.356**
	+ Encoder	1.586	**0.845**	**0.600**	**0.493**	**0.381**	**0.340**
	+ LM	**0.724**	**0.395**	**0.285**	**0.231**	**0.172**	**0.145**
	All	**0.623**	**0.346**	**0.242**	**0.197**	**0.149**	**0.129**
	Ratio	2.70	2.61	2.68	2.72	2.72	2.76
Apple M4	None (FP16)	**0.510**	**0.361**	**0.336**	**0.294**	**0.273**	**0.261**
	All	**0.528**	**0.265**	**0.191**	**0.155**	**0.174**	**0.142**
	Ratio	0.97	1.36	1.76	1.90	1.57	1.84
Core i7-13700	None (FP16)	4.153	2.115	1.502	1.274	**0.996**	1.068
	All	1.546	**0.689**	**0.455**	**0.385**	**0.319**	**0.338**
	Ratio	2.69	3.07	3.30	3.31	3.13	3.16

Table 12: VibeVoice-TTS: end-to-end RTF against thread count.

Machine	Quantized	Real-time factor ↓					
		1T	2T	3T	4T	6T	8T
EPYC 7V13	None (FP16)	6.163	3.402	2.419	2.074	1.608	1.450
	+ INT8 stack	4.744	2.618	1.829	1.500	1.168	1.068
	+ Ternary transformer	3.913	2.229	1.590	1.356	1.013	**0.889**
	All	2.806	1.531	1.028	**0.845**	**0.610**	**0.508**
	Ratio	2.20	2.22	2.35	2.46	2.63	2.85
Apple M4	None (FP16)	2.520	1.710	1.560	1.490	1.280	1.350
	All	1.550	**0.890**	**0.700**	**0.620**	**0.740**	**0.640**
	Ratio	1.63	1.92	2.23	2.40	1.73	2.11
Core i7-13700	None (FP16)	4.731	3.237	2.978	2.771	2.635	2.617
	All	1.919	1.209	1.062	1.009	**0.958**	**0.977**
	Ratio	2.47	2.68	2.80	2.75	2.75	2.68

Table 13: Qwen3-TTS: end-to-end RTF against thread count.

Machine	Quantized	Real-time factor ↓					
		1T	2T	3T	4T	6T	8T
EPYC 7V13	None (FP16)	4.656	2.592	1.838	1.542	1.174	1.058
	+ INT8 stack	3.265	1.784	1.267	1.029	**0.796**	**0.706**
	+ Ternary transformer	2.609	1.571	1.252	1.019	**0.748**	**0.661**
	All	1.755	**0.983**	**0.686**	**0.566**	**0.422**	**0.347**
	Ratio	2.65	2.64	2.68	2.72	2.78	3.05
Apple M4	None (FP16)	1.568	1.105	1.066	**0.959**	**0.887**	**0.906**
	All	1.013	**0.555**	**0.393**	**0.330**	**0.405**	**0.352**
	Ratio	1.55	1.99	2.71	2.91	2.19	2.57
Core i7-13700	None (FP16)	3.438	2.337	2.041	1.806	1.660	1.664
	All	**0.970**	**0.583**	**0.499**	**0.461**	**0.439**	**0.492**
	Ratio	3.54	4.01	4.09	3.92	3.78	3.38

Table 14: VoxCPM2: end-to-end RTF against thread count.

Machine	Quantized	Real-time factor ↓					
		1T	2T	3T	4T	6T	8T
EPYC 7V13	None (FP16)	10.829	5.723	3.908	3.110	2.225	1.818
	+ INT8 stack	10.421	5.514	3.807	3.039	2.189	1.780
	+ Ternary transformer	4.630	2.570	1.809	1.433	1.026	**0.826**
	All	4.411	2.477	1.724	1.368	**0.990**	**0.793**
	Ratio	2.45	2.31	2.27	2.27	2.25	2.29
Apple M4	None (FP16)	3.653	2.130	1.630	1.468	1.816	1.611
	All	3.729	1.958	1.364	1.114	1.353	1.109
	Ratio	0.98	1.09	1.20	1.32	1.34	1.45
Core i7-13700	None (FP16)	8.430	4.790	3.679	3.165	2.620	2.457
	All	3.310	1.843	1.450	1.258	1.057	1.041
	Ratio	2.55	2.60	2.54	2.52	2.48	2.36

Time to first output. Table 15 measures the wait for the first output unit, which no pipelining can hide, with generation stopped after one unit. Quantization improves it by 2.0 to 6.5×, so the three generation systems answer in 85–603 ms at eight threads and streaming VibeVoice-ASR in 949, its first token waiting on an encoder pass over a 2.93 s window. That still fits inside the window where FP16 at 6177 ms does not. Qwen3-ASR encodes the whole input before it emits anything and waits 1330 ms, 2.0 to 2.2× better than FP16 over thread counts.

Table 15: Time to first output on the EPYC 7V13, the first text token for recognition and the audio sample otherwise. VibeVoice-ASR waits on a 2.93 s streaming window, Qwen3-ASR on all 20 s.

System	Format	Time to first output (ms) ↓					
		1T	2T	3T	4T	6T	8T
VibeVoice-ASR	FP16	12606	8786	6878	6493	5386	6177
	Ours	4853	2706	1888	1548	1112	949
	Ratio	2.60	3.25	3.64	4.19	4.84	6.51
Qwen3-ASR	FP16	16339	8598	6048	4756	3293	2604
	Ours	7444	4034	2796	2227	1615	1330
	Ratio	2.19	2.13	2.16	2.14	2.04	1.96
VibeVoice-TTS	FP16	6514	3571	2557	2085	1560	1362
	Ours	2604	1509	1077	896	705	603
	Ratio	2.50	2.37	2.37	2.33	2.21	2.26
Qwen3-TTS	FP16	935	575	455	371	328	305
	Ours	339	202	152	123	103	85
	Ratio	2.76	2.85	2.99	3.02	3.17	3.61
VoxCPM2	FP16	1826	989	677	546	390	327
	Ours	780	443	298	234	169	138
	Ratio	2.34	2.23	2.27	2.33	2.30	2.37

E MODULE-LEVEL TIMING BREAKDOWN

Tables 16–20 give one table per system, all on the EPYC 7V13, with each module's real-time factor per thread count, its FP16-to-quantized ratio, and its share of module time at eight threads. Every module gains 1.4 to 4.4×, the ternary decode paths most.

Table 16: VibeVoice-ASR by module on the EPYC 7V13. Real-time factor against thread count, the FP16-to-quantized ratio, and each module's share of module time at eight threads.

Module	Format	Real-time factor ↓					Share
		1	2	4	6	8	at 8T (%)
VAE Encoder	FP16	2.419	1.440	0.916	0.731	0.574	68.7
	INT8	1.119	0.585	0.310	0.217	0.170	63.8
	Ratio	2.16	2.46	2.96	3.37	3.37	
LM Prefill	FP16	0.379	0.199	0.105	0.072	0.055	6.6
	I2_S	0.139	0.074	0.040	0.028	0.022	8.4
	Ratio	2.74	2.68	2.62	2.55	2.46	
LM Decode	FP16	0.819	0.417	0.255	0.214	0.206	24.7
	I2_S	0.302	0.173	0.099	0.076	0.074	27.9
	Ratio	2.71	2.42	2.58	2.81	2.77	

Table 17: Qwen3-ASR by module on the EPYC 7V13. Real-time factor against thread count, the FP16-to-quantized ratio, and each module's share of module time at eight threads.

Module	Format	Real-time factor ↓					Share
		1	2	4	6	8	at 8T (%)
Conv downsampler	FP16	0.110	0.060	0.040	0.026	0.022	6.1
	INT8	0.072	0.037	0.021	0.016	0.013	10.6
	Ratio	1.53	1.64	1.90	1.60	1.62	
AuT layers	FP16	0.135	0.070	0.038	0.026	0.021	5.9
	I2_S	0.072	0.038	0.020	0.015	0.012	9.8
	Ratio	1.87	1.86	1.88	1.76	1.71	
LM Prefill	FP16	0.569	0.295	0.155	0.108	0.083	23.6
	I2_S	0.228	0.122	0.065	0.046	0.036	29.3
	Ratio	2.50	2.41	2.41	2.36	2.28	
LM Decode	FP16	0.863	0.472	0.298	0.238	0.226	64.3
	I2_S	0.250	0.138	0.085	0.068	0.062	50.4
	Ratio	3.45	3.42	3.52	3.50	3.65	

Table 18: VibeVoice-TTS by module on the EPYC 7V13. Real-time factor against thread count, the FP16-to-quantized ratio, and each module's share of module time at eight threads.

Module	Format	Real-time factor ↓					Share
		1	2	4	6	8	at 8T (%)
LM Prefill	FP16	0.259	0.134	0.071	0.048	0.038	2.6
	I2_S	0.098	0.052	0.028	0.019	0.015	3.0
	Ratio	2.64	2.60	2.53	2.57	2.48	
LM Decode	FP16	2.821	1.576	0.957	0.737	0.698	48.1
	I2_S	0.781	0.459	0.260	0.196	0.167	33.0
	Ratio	3.61	3.43	3.68	3.77	4.17	
Diffusion Head	FP16	1.936	1.007	0.591	0.447	0.404	27.9
	INT8	1.336	0.703	0.383	0.268	0.223	44.0
	Ratio	1.45	1.43	1.54	1.67	1.81	
VAE Decoder	FP16	1.145	0.684	0.454	0.375	0.309	21.4
	INT8	0.588	0.316	0.173	0.127	0.101	20.0
	Ratio	1.95	2.16	2.62	2.95	3.05	

Table 19: Qwen3-TTS by module on the EPYC 7V13. Real-time factor against thread count, the FP16-to-quantized ratio, and each module's share of module time at eight threads.

Module	Format	Real-time factor ↓					Share
		1	2	4	6	8	at 8T (%)
Text Prefill	FP16	0.059	0.034	0.018	0.013	0.011	1.0
	I2_S	0.023	0.015	0.009	0.007	0.006	1.7
	Ratio	2.63	2.31	1.99	1.85	1.80	
Talker	FP16	0.788	0.464	0.282	0.223	0.203	19.2
	I2_S	0.209	0.130	0.081	0.065	0.057	16.5
	Ratio	3.77	3.57	3.48	3.43	3.54	
Code Predictor	FP16	1.466	0.864	0.486	0.381	0.348	32.9
	I2_S	0.393	0.248	0.159	0.133	0.117	33.7
	Ratio	3.73	3.48	3.05	2.87	2.98	
Tokenizer Decoder	FP16	2.336	1.228	0.754	0.557	0.496	46.9
	INT8	1.125	0.588	0.315	0.217	0.166	48.0
	Ratio	2.08	2.09	2.39	2.57	2.98	

Table 20: VoxCPM2 by module on the EPYC 7V13. Real-time factor against thread count, the FP16-to-quantized ratio, and each module's share of module time at eight threads.

Module	Format	Real-time factor ↓					Share
		1	2	4	6	8	at 8T (%)
Prefill	FP16	0.056	0.032	0.019	0.015	0.014	0.8
	I2_S	0.028	0.015	0.009	0.007	0.005	0.7
	Ratio	2.05	2.12	2.07	2.20	2.61	
TSLM + RALM	FP16	1.296	0.721	0.439	0.352	0.344	18.9
	I2_S	0.328	0.189	0.114	0.089	0.078	9.8
	Ratio	3.95	3.81	3.86	3.96	4.41	
CFM LocDiT	FP16	8.809	4.610	2.455	1.718	1.353	74.5
	I2_S	3.665	2.061	1.129	0.808	0.641	80.9
	Ratio	2.40	2.24	2.17	2.13	2.11	
VAE Decoder	FP16	0.663	0.357	0.195	0.138	0.106	5.8
	INT8	0.390	0.211	0.115	0.086	0.068	8.6
	Ratio	1.70	1.69	1.69	1.61	1.56	

F TRAINING COST

Section 4 gives the schedule; this section gives its price, as wall-clock from the training logs of the runs that produced the released checkpoint.

Encoders. Both encoders are ConvNeXt VAEs of the same shape, trained separately in bf16 on one A100 80 GB each with Adam on a cosine schedule peaking at a few times 10^{-5}. The acoustic one is trained on waveform reconstruction; the semantic one is initialized from it and trained on next-token cross-entropy through a linear projection into the embedding space of a frozen Qwen2.5-1.5B teacher, so its loss is in nats per token and Figure 5b is that stage. The acoustic stage runs for 10.5k updates and the semantic one for 12k, together ≈ 34 A100-hours.

Transformer. The transformer stage runs in bf16 on 64 A100 80 GB GPUs, eight nodes of eight, with the encoders frozen, gradients and optimizer state sharded across ranks, and a global batch of 256 under AdamW at a peak learning rate of 3×10^{-5}. We train for 6000 steps, which takes 47.2 hours, 1.5M samples and ≈ 3000 A100-hours, as a fine-tune of an existing bf16 checkpoint rather than a pretraining run. The other four systems are converted without training.