NANOVOCO: UPSAMPLING-AWARE RANK-WIDTH REALLOCATION FOR LIGHTWEIGHT VOCODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Deploying text-to-speech (TTS) systems on resource-constrained edge devices requires lightweight neural vocoders. Under extreme compression, reducing channel width degrades synthesis quality. Although depthwise-separable convolutions (DSCs) allow wider layers at comparable parameter budgets, replacing standard convolutions (StdCs) with DSCs degrades quality even further in our experiments. We observe that this replacement involves reallocating parameters from convolutional rank to channel width, and find that the benefits of this rank-width reallocation depend strongly on the temporal upsampling rule. Replacing zero-insertion (ZI) with sample-repetition (SR) upsampling substantially improves the benefits of rank-width reallocation. We observe this interaction across different discriminator designs, generator backbones, and parameter budgets, with the strongest effect under extreme compression. Based on this observation, we propose NanoVoco, with only 72.3K parameters and UTMOS-measured synthesis quality comparable to the evaluated Parallel WaveGAN model and higher than FreGrad. Integrated into a complete TTS pipeline, its quantized implementation enables real-time streaming speech synthesis on a microcontroller using only on-chip memory.

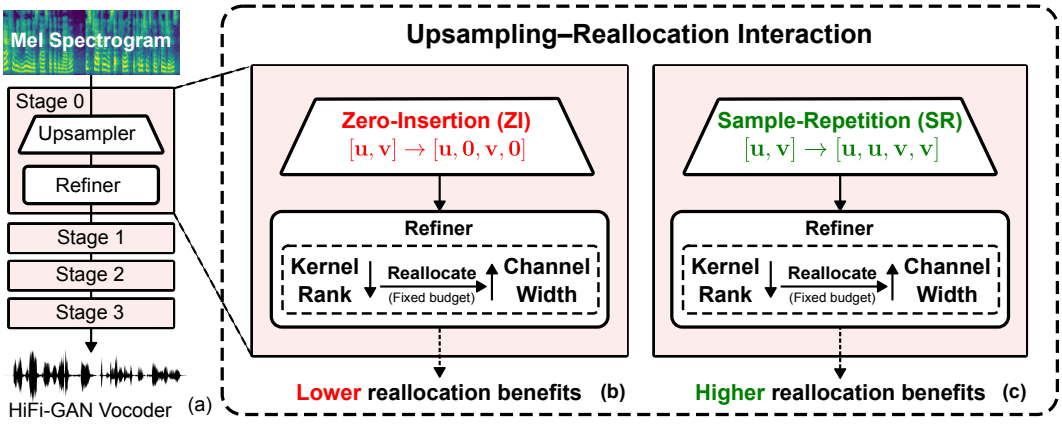


Figure 1: How upsampling affects rank-width reallocation benefits. (a) Each stage increases temporal resolution through an upsampler and then applies learned convolutions through a refiner. (b,c) Lower-rank convolutional filters save parameters that can be reinvested in additional channels. The evaluated configurations yield lower reallocation benefits under ZI and higher benefits under SR.

1 INTRODUCTION

Deploying text-to-speech (TTS) on resource-constrained edge devices requires compact models and real-time inference (Popov et al., 2020; Atienza, 2023). As lightweight acoustic models reduce the cost of mel-spectrogram generation, the neural vocoder can dominate the remaining parameter and inference budgets, making waveform synthesis a key target for compression (Atienza, 2023; De Beer

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.

& Gao, 2026). We focus on neural vocoders that alternate upsampling with learned waveform refinement (Kong et al., 2020; Lee et al., 2022; Bak et al., 2023), as illustrated in Fig. 1(a).

Depthwise-separable convolutions (DSCs) reduce the parameter cost of convolutional layers, allowing the saved parameters to be reinvested in channel width. However, this strategy does not deliver the intended benefit in our extreme-compression experiments. At comparable parameter budgets, replacing standard convolutions in the post-upsampling refinement blocks with DSCs yields wider models but substantially lower UTMOS, an automatic metric of synthesis quality (Saeki et al., 2022).

These results are obtained with zero-insertion (ZI) upsampling, which expands a sequence by inserting zeros between input features. An alternative is sample-repetition (SR) upsampling, which expands the sequence by repeating each feature. nVOC-22 combines this repetition scheme with separable convolution in its upsampling blocks (Iyer, 2023). Motivated by this precedent, we find that an SR upsampling with local DSC layers outperforms our ZI wide-DSC model. Extending DSC to the subsequent learned processing stages yields further quality gains over this nVOC-inspired configuration at a comparable parameter budget. Replacing StdC with DSC and reinvesting the saved parameters in width reallocates parameters from convolutional rank to channel width. To disentangle the interaction between upsampling and this rank-width reallocation, we compare a narrow StdC model, a same-width DSC model, and a wider DSC model under each upsampling rule. The narrow StdC and wider DSC models have comparable parameter counts. The results show that SR enhances the benefits of this reallocation by both reducing the magnitude of the rank-reduction cost and increasing the gain from widening.

We next test whether this interaction extends beyond the StdC-DSC endpoint comparison. A StdC-to-DSC sweep shows that across all evaluated reduced-rank settings, SR makes the rank-reduction cost less negative and increases the widening gain relative to ZI, turning negative reallocation benefits into positive ones. A complementary ZI-to-SR sweep reveals a rank-reduction cost that moves toward zero and a sharp increase in widening gain at the first tested nonzero interpolation coefficient, where the net benefit becomes positive. We then vary architecture and training supervision: a BigVGAN-derived setting and a HiFi-GAN backbone trained with Avocodo supervision reproduce this pattern of smaller rank-reduction losses and larger widening gains. Finally, comparisons across four parameter budgets show that SR improves reallocation benefits most strongly at approximately 72K parameters, with substantially weaker improvements at larger budgets.

We instantiate the resulting design principle in NanoVoco, combining SR with wide DSC refiners and separately evaluating DynamicTanh as an activation refinement (Zhu et al., 2025). With 72.3K parameters, NanoVoco achieves a mean UTMOS of 3.71 on LJSpeech, comparable to the evaluated Parallel WaveGAN (Yamamoto et al., 2020) model and higher than FreGrad (Nguyen et al., 2024), each over $18\times$ larger. Its quantized implementation supports a complete TTS pipeline that streams speech in real time on a microcontroller using only on-chip memory.

Audio samples are available on our anonymous demo website,[1] and code and model weights are provided in our anonymous repository.[2]

2 Related Work

Efficient neural vocoders. Prior work has reduced neural-vocoder cost through narrower generators, parameter sharing, separable convolutions, and alternative waveform representations. HiFi-GAN V2 reduces generator width (Kong et al., 2020), while SlimTTS simplifies a HiFi-GAN-based vocoder through channel and kernel-size reductions (De Beer & Gao, 2026). MISRNet reduces computation by sharing residual blocks across multiple input branches (Kaneko et al., 2022a). Other lightweight approaches include iSTFT-based vocoders (Kaneko et al., 2022b; 2023; Siuzdak, 2024; Dang et al., 2023) and flow-based SqueezeWave (Zhai et al., 2020), which avoid much of the conventional time-domain upsampling backbone. These approaches reduce vocoder cost through different architectural strategies, but do not directly study how the temporal upsampling rule affects the benefit of reallocating parameters from convolutional rank to channel width under extreme compression.

[1] https://anonymous.4open.science/w/NanoVoco-demo-6B2C/

[2] https://anonymous.4open.science/r/NanoVoco-12B0/README.md

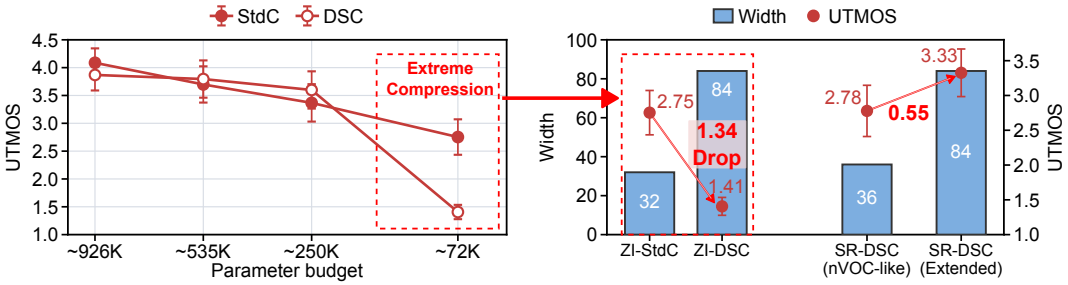


Figure 2: Motivating comparisons. Left: ZI-based StdC and DSC generators across parameter budgets. Right: initial widths (bars) and UTMOS (points) at approximately 72K parameters, including nVOC-inspired local DSC and broader DSC refiners under SR. Error bars show SD across 128 utterances.

Upsampling artifacts. Prior work has shown that temporal upsampling can introduce structured artifacts in neural audio synthesis. Sugawara et al. (2018) analyze conditions that produce checkerboard artifacts and propose zero-order-hold-based alternatives. Pons et al. (2021) study tonal and filtering artifacts in neural audio upsamplers and show that interpolation-based methods can mitigate some artifacts while introducing their own filtering effects. Tang et al. (2023) further characterize spectral repetition induced by convolutional decoders in the frequency domain. Avocodo (Bak et al., 2023) addresses vocoding artifacts through adversarial supervision of intermediate and sub-band outputs. These works establish that the upsampling operator can affect artifact structure, but do not examine how it changes the return from rank-width reallocation under a fixed parameter budget.

Upsampling-reallocation interaction. The closest architectural precedent is nVOC-22 (Iyer, 2023), which combines sample-repetition upsampling with separable convolution for efficient mobile vocoding. Its separable convolution follows the resize operation within each upsampling block, while the subsequent residual blocks use standard convolutions. We further extend separable convolution to the remaining learned processing stages and focus on the interaction between temporal upsampling and learned processing. We study how reducing convolutional rank and reallocating the saved parameters to channel width changes synthesis quality, and how the benefits of rank-width reallocation depend on the temporal upsampling rule.

3 Upsampling Changes the Benefits of Rank-Width Reallocation

We begin with compression failure under ZI and examine whether changing upsampling makes parameter reallocation more effective. Each stage is decomposed into a parameter-free temporal *upsampler* and a learned *refiner*, comprising the post-expansion convolution and all remaining learned processing in that stage. Unless otherwise stated, results for our trained vocoders use seed 1234; UTMOS means and SDs are computed across the same 128 evaluation utterances. Differences are computed before rounding unless noted otherwise.

3.1 Compression Failure at Comparable Budgets

We compare ZI-based StdC and DSC generators across four budgets (Fig. 2, left). All configurations are trained in-house using the common HiFi-GAN V1 protocol. Architectures and UTMOS22-strong (Saeki et al., 2022) evaluation details are given in Appendices A and B.

DSCs substantially reduce convolutional parameter cost and therefore allow wider configurations at comparable budgets. However, the additional width does not consistently translate into higher synthesis quality under extreme compression. In the smallest comparison group at approximately 72K generator parameters, the DSC generator with initial width 84 achieves a mean UTMOS of 1.41, compared with 2.75 for the StdC generator with initial width 32. The wider DSC model is 1.34 UTMOS points below the narrow StdC reference. This result shows that parameter-efficient widening alone is insufficient in this extreme-compression regime.

The combination of sample-repetition upsampling and separable convolution in nVOC-22 (Iyer, 2023) motivates examining the upsampling rule alongside convolutional structure. We evaluate an nVOC-inspired local-DSC configuration, in which each stage uses SR followed by one DSC layer, while the subsequent residual convolutions remain standard. Under the same training and evaluation protocol, this configuration achieves a UTMOS score of 2.78 with an initial width of 36. This configuration slightly outperforms ZI-StdC, although both the upsampling rule and convolutional configuration differ.

We then extend DSC to all eligible convolutions in the refiner. The resulting extended SR-DSC refiner achieves a UTMOS score of 3.33 at width 84 (Fig. 2, right). At a comparable parameter budget, extending DSC throughout the refiner and reinvesting the parameter savings in width improves mean UTMOS by 0.55 over the local-DSC configuration. The same width-84 DSC configuration, however, performs poorly under ZI. Because replacing StdC with DSC changes both the convolutional structure and channel width, we next introduce matched controls to isolate how the upsampling rule affects the benefit of parameter reallocation.

3.2 Rank-Width Reallocation in the Refiner

The transformation from StdC to a same-width DSC configuration can be characterized as a reduction in convolutional rank (Guo et al., 2018). Reinvesting the resulting parameter savings in channel width then yields a wider DSC configuration under a comparable parameter budget, effectively reallocating parameters from convolutional rank to channel width (Rank-Width Reallocation).

Consider a convolution with C_{in} input channels, C_{out} output channels, and kernel length k. Let $K_i \in \mathbb{R}^{C_{\text{out}} \times k}$ collect the temporal filters from input channel i to all output channels as its rows. Its matrix rank counts the linearly independent filters in this collection. We parameterize this slice as

$$K_i = \sum_{q=1}^{R} p_{i,q} h_{i,q}^{\top}, \qquad \operatorname{rank}(K_i) \le R, \tag{1}$$

where $h_{i,q} \in \mathbb{R}^k$ performs temporal filtering and $p_{i,q} \in \mathbb{R}^{C_{\text{out}}}$ mixes the resulting response across output channels.

At $R = \min(C_{\text{out}}, k)$, the factorization can represent an arbitrary StdC kernel slice. At $R = 1$, it recovers a conventional depthwise-pointwise convolution without an intervening nonlinearity (Guo et al., 2018). Here, R bounds the rank of each local kernel slice; it does not denote the attained rank after training or the rank of the nonlinear refiner as a whole.

Ignoring biases, the parameter costs are

$$P_{\text{StdC}} = kC_{\text{in}}C_{\text{out}}, \qquad P_R = RC_{\text{in}}(k + C_{\text{out}}). \tag{2}$$

When $P_R < P_{\text{StdC}}$, reducing rank frees parameters that can be reinvested in channel width at a comparable total budget.

3.3 Matched ZI-SR Intervention

To investigate how upsampling interacts with the benefits of rank-width reallocation, we evaluate three configurations that separate the cost of replacing StdC with DSC from the benefit of reinvesting the resulting parameter savings in channel width. Let A denote the narrow StdC generator with initial width 32, B the same-width rank-one DSC control, and C the wider rank-one DSC generator with initial width 84 at a parameter budget comparable to A. For upsampling rule u, let $Q_u(\cdot)$ denote mean UTMOS.

The $A \to B$ comparison measures the cost of rank reduction at fixed width; $B \to C$ measures the gain from widening at fixed convolutional structure. Their combination, $A \to C$, gives the net reallocation benefit at comparable parameter budgets.

Within each matched ZI-SR pair, only the parameter-free temporal expansion changes; the learned rank-width configuration and parameter count remain unchanged (Appendix A). We train all six A/B/C configurations separately using the common protocol and evaluate their final 800,000-step checkpoints (Fig. 3).

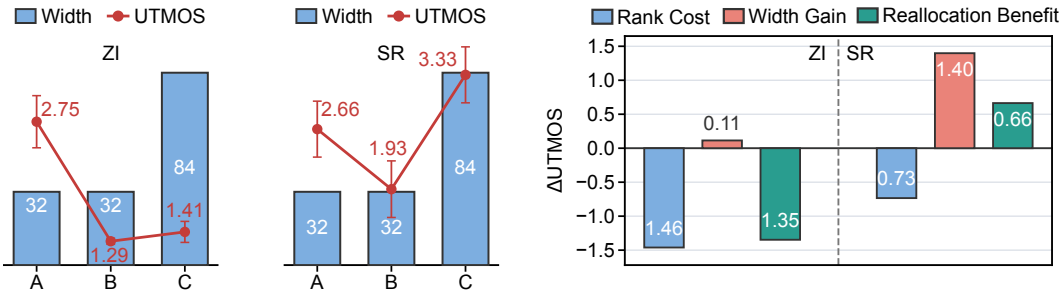


Figure 3: A/B/C controls under ZI (left) and SR (center), with initial width shown by bars and UTMOS by points. Error bars show SD across 128 utterances. Right: rank-reduction cost, widening gain, and reallocation benefit. Cost is plotted below zero; bar labels give magnitudes.

Rank-reduction cost is negative when rank reduction degrades quality. Adding it to the widening gain gives the reallocation benefit:

$$\underbrace{Q_u(C) - Q_u(A)}_{\text{Reallocation benefit}} = \underbrace{Q_u(C) - Q_u(B)}_{\text{Widening gain}} + \underbrace{\big[Q_u(B) - Q_u(A)\big]}_{\text{Rank-reduction cost}}, \qquad u \in \{\text{ZI}, \text{SR}\}. \tag{3}$$

Under ZI, the rank-reduction cost is -1.46 UTMOS points, while widening recovers only 0.11 points. Under SR, the rank-reduction cost is -0.73, closer to zero, while the widening gain increases to 1.40. The resulting reallocation benefit is -1.35 under ZI and $+0.66$ under SR.

To quantify the interaction, we compare the reallocation benefits under the two upsampling rules:

$$\underbrace{Q_{\text{SR}}(C) - Q_{\text{SR}}(A)}_{\text{Reallocation benefit under SR}} - \underbrace{\big[Q_{\text{ZI}}(C) - Q_{\text{ZI}}(A)\big]}_{\text{Reallocation benefit under ZI}}. \tag{4}$$

The increase in reallocation benefit is $+2.01$ UTMOS points, showing that SR preferentially benefits the widened, rank-constrained configuration. For these HiFi-GAN endpoint controls, we observe the smaller rank-reduction loss, larger widening gain, and sign reversal in each of the three tested training seeds (Appendix B.4).

4 Characterizing the Upsampling-Reallocation Interaction

We examine how the interaction varies with rank constraints, temporal-expansion rules, generator settings, and parameter budgets.

4.1 Rank-Width Sweep

We test whether the endpoint interaction extends to intermediate convolutional structures using the kernel-rank parameterization introduced in Sec. 3.2 (Eq. 1). We evaluate nominal rank ratios $\rho \in \{100\%, 75\%, 50\%, 25\%, 0\%\}$, where 100% denotes the dense StdC reference and 0% denotes the rank-one DSC endpoint. Layer-wise rank selection and dense-fallback rules are specified in Appendix A.

Fixed-width and comparable-budget controls. We extend the fixed-width and comparable-budget controls from Sec. 3.3 into two rank series, evaluated under both ZI and SR. The *fixed-width* series keeps the initial width at 32 while varying the nominal rank ratio. This isolates the effect of reducing convolutional rank without compensatory widening, although the generator parameter count decreases as rank is reduced. The *comparable-budget* series instead increases channel width as rank is reduced so that the generator parameter count remains approximately matched to the 71.8K-parameter narrow StdC reference. Comparing the two series at the same nominal rank ratio therefore characterizes the additional benefit obtained by reinvesting the saved parameters into width. Exact layer-wise configurations are given in Appendix A.

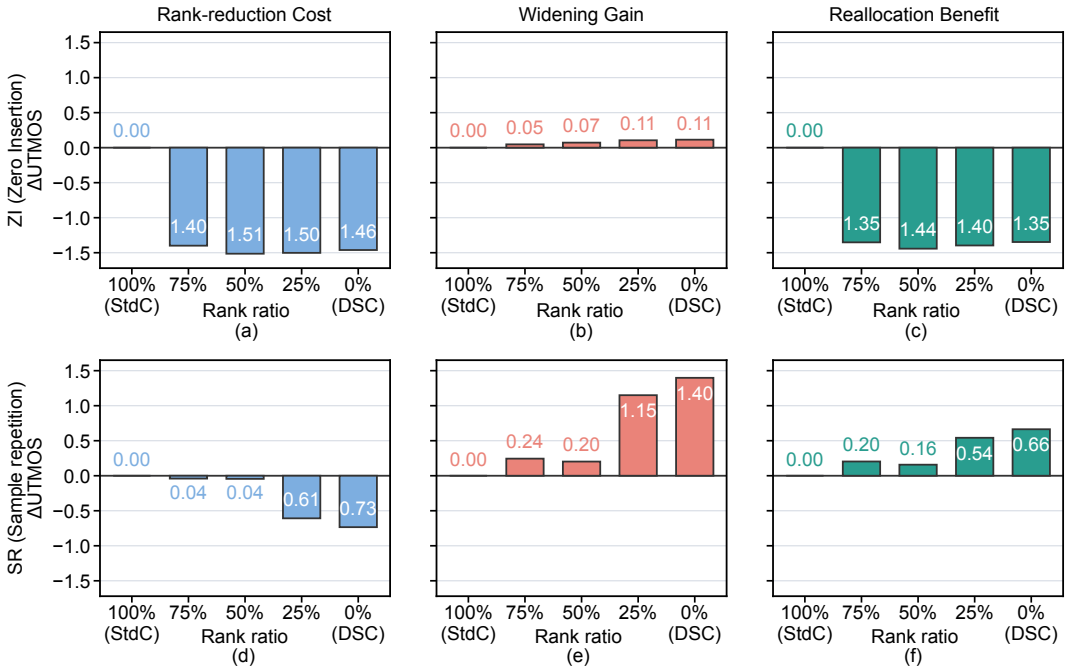


Figure 4: Decomposition across nominal rank ratios under ZI (a–c) and SR (d–f). Columns show rank-reduction cost (plotted below zero), widening gain, and reallocation benefit. The 100% endpoint is StdC and 0% is rank-one DSC. Contrasts use mean UTMOS over 128 utterances; bar labels give magnitudes.

Quality across the rank-width sweep. Under ZI, all reduced-rank configurations lose quality at fixed width, and widening fails to recover the narrow StdC reference (Fig. 4(a–c)). The rank-reduction cost ranges from -1.51 to -1.40 UTMOS points, whereas widening recovers only 0.05–0.11 points. The net benefit therefore remains negative throughout the evaluated reduced-rank series.

Under SR, the rank-reduction cost becomes more negative as rank constraints strengthen, but its magnitude remains smaller than under ZI at every matched setting. Widening more than compensates for this loss, yielding positive reallocation benefits of 0.16–0.66 UTMOS points, with the maximum at the DSC endpoint (Fig. 4(f)). The widening gain also exceeds its ZI counterpart at every evaluated reduced-rank setting. The improvement thus extends to intermediate rank constraints, but the net benefit is not monotonic in the degree of rank reduction under either upsampling rule.

4.2 Interpolation between ZI and SR

We next vary the temporal expansion continuously between ZI and SR while retaining the same learned architecture within each series.

For input features $X = (x_t)$ and upsampling factor r, we define

$$[U_\lambda(X)]_{rt+p} = \begin{cases} x_t, & p = 0, \\ \lambda x_t, & p = 1, \ldots, r-1, \end{cases} \qquad \lambda \in [0, 1]. \tag{5}$$

With the boundary convention in Appendix A, $\lambda = 0$ recovers ZI and $\lambda = 1$ gives SR. We evaluate $\lambda \in \{0, 0.25, 0.5, 0.75, 1\}$ for all three A/B/C configurations, using the same coefficient at all four stages while holding the learned configuration and training settings fixed within each series. The fixed-width DSC control allows us to track rank-reduction cost and widening gain throughout this transition.

As λ increases from zero to one, the rank-reduction cost moves toward zero, from -1.46 to -0.73 UTMOS points (Fig. 5). At the first nonzero coefficient, the widening gain rises from 0.11 at $\lambda = 0$ to 1.30 at $\lambda = 0.25$, more than offsetting the corresponding rank-reduction cost of -0.95. The

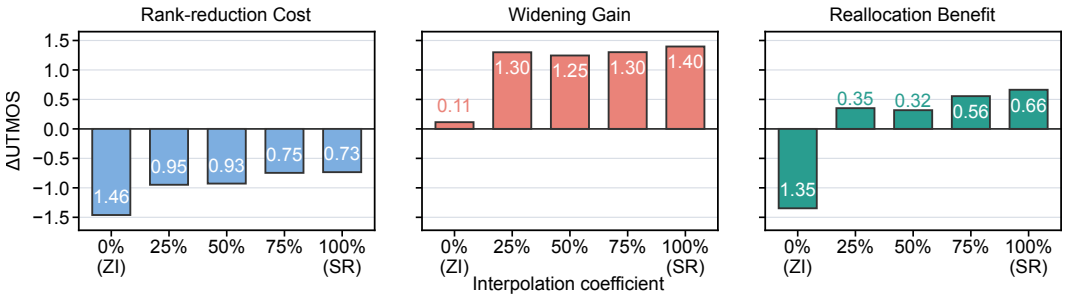


Figure 5: Decomposition along the ZI-SR interpolation. Columns show rank-reduction cost (plotted below zero), widening gain, and reallocation benefit; bar labels give magnitudes. The horizontal axis is the insertion coefficient λ, expressed as a percentage: it scales every inserted feature, rather than selecting a fraction of positions. Each A/B/C architecture is fixed across the sweep.

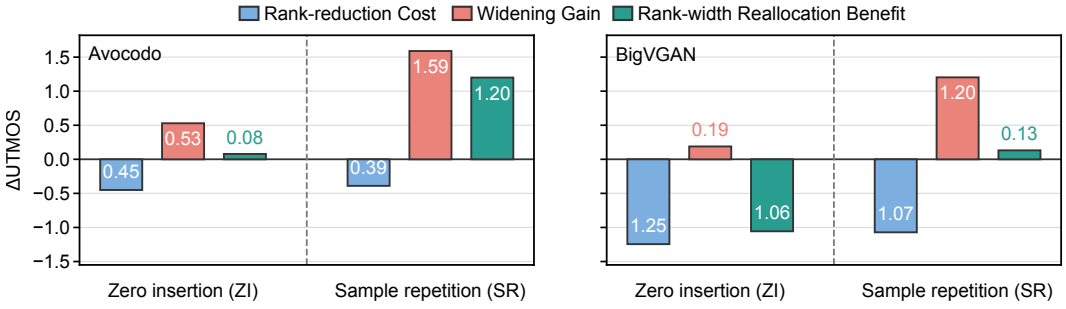


Figure 6: Transfer of the A/B/C decomposition under Avocodo supervision (left) and in a BigVGAN-derived setting (right). Rank-reduction cost is plotted below zero, alongside widening gain and reallocation benefit. Contrasts use mean UTMOS over 128 utterances; bar labels give magnitudes.

net reallocation benefit consequently changes from -1.35 to $+0.35$. At subsequent coefficients, widening gains remain between 1.25 and 1.40, while rank-reduction costs move closer to zero.

The benefit remains positive at every evaluated nonzero coefficient and reaches $+0.66$ under SR, although it is not monotonic in λ. Meanwhile, wide-DSC quality increases at every evaluated step, from 1.41 under ZI to 3.33 under SR. The widening gain thus rises sharply by the first nonzero tested coefficient, rather than increasing uniformly toward SR.

4.3 Transfer across Training and Generator Settings

We test whether the interaction persists when changing adversarial supervision or the generator backbone. The Avocodo setting retains the HiFi-GAN backbone and adopts Avocodo's discriminators and intermediate-output supervision (Bak et al., 2023). The BigVGAN-derived setting uses a different backbone, including its native activation modules and discriminator design (Lee et al., 2022). We repeat the A/B/C comparison within each setting; implementation and parameter counts are detailed in Appendix C.

Both settings reproduce the endpoint pattern from HiFi-GAN: SR makes the rank-reduction cost less negative and increases the widening gain (Fig. 6; Table 7). Under Avocodo supervision, SR strengthens an already positive net benefit, from $+0.08$ to $+1.20$ UTMOS points. In the BigVGAN-derived setting, the benefit instead changes from -1.06 to $+0.13$, reversing the relative quality ordering of wide DSC and narrow StdC.

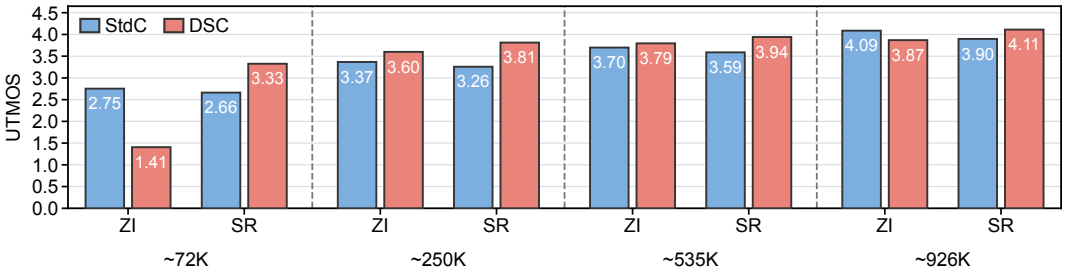


Figure 7: StdC and budget-comparable wide DSC across four parameter budgets under ZI and SR. Bars show mean UTMOS over 128 utterances. The within-rule DSC–StdC difference is the reallocation benefit; exact parameter counts and utterance-level SDs are in Table 6.

4.4 Scope across Parameter Budgets

To assess budget dependence, we evaluate both narrow StdC (A) and budget-comparable wide DSC (C) under ZI and SR at approximately 72K, 250K, 535K, and 926K parameters (Appendix B.5). Within each configuration, the ZI-SR pair has identical channel width and parameter count.

Figure 7 shows that under SR, wide DSC exceeds the corresponding StdC reference at every evaluated budget. Under ZI, the reallocation benefit is positive at 250K and 535K but negative at 72K and 926K. The increase in reallocation benefit from ZI to SR is positive at every evaluated budget: $+2.01$, $+0.32$, $+0.26$, and $+0.43$. The interaction therefore persists at larger budgets but is substantially weaker than at 72K, placing its strongest effect in the extreme-compression regime. Its magnitude does not decrease monotonically with budget.

The reference upsampling rule matters when interpreting these gains. At 926K, SR-DSC exceeds SR-StdC by 0.21 points and ZI-StdC by 0.02 points. The largest-budget result therefore preserves the interaction, but its absolute advantage over ZI-StdC is small.

5 NanoVoco: Design and Evaluation

NanoVoco implements the design principle from Sec. 3 using SR and width-84 rank-one DSC refiners (Appendix A).

5.1 Model Design and Activation Ablation

Starting from this SR-DSC configuration, we evaluate DynamicTanh (DyT) (Zhu et al., 2025) as an empirical refinement of the final model. Replacing LeakyReLU with DyT increases mean UTMOS from 3.33 to 3.71. The HiFi-GAN rank-width and upsampling controls use LeakyReLU, so their quality improvements are already present before introducing DyT. DyT changes UTMOS by -0.03 for ZI-StdC, $+0.05$ for SR-StdC, and -0.06 for ZI-DSC, compared with $+0.39$ for SR-DSC. These controls motivate DyT as a refinement of the selected SR-DSC configuration; the activation is not uniformly beneficial across the evaluated configurations. For SR-DSC, the DyT gain is positive in each of the three tested training seeds (Appendix D). Adopting DyT yields the final NanoVoco with 72.3K parameters.

5.2 Quality-Efficiency Evaluation

We next compare NanoVoco with publicly available vocoders, including RNDVoC (Li et al., 2025) and MelGAN (Kumar et al., 2019), in floating-point vocoder-only quality and CPU throughput (Table 1). Quality evaluation follows Sec. 3.1; CPU inference is single-threaded FP32 (Appendix F).

With only 72.3K parameters, NanoVoco achieves 3.71 UTMOS, comparable to the evaluated Parallel WaveGAN (3.70), which has 18.5Œ as many parameters, while substantially outperforming FreGrad (3.27), which has 24.7Œ as many parameters. HiFi-GAN V2 achieves higher quality (4.09 UTMOS) with 12.8 times as many parameters. NanoVoco's mean utterance-level word error rate (WER)

Table 1: Vocoder quality on 128 LJSpeech utterances and CPU throughput. UTMOS: mean ± SD; WER: macro-average [95% CI]. Parenthesized ratios are relative to NanoVoco.

Model	Parameters ↓	UTMOS ↑	WER (%) ↓ mean [95% CI]	CPU throughput (× real time) ↑
NanoVoco (Ours)	**72.3K (1.0×)**	3.71 ± 0.31	3.28 [2.09, 4.46]	**42.98 (1.00×)**
HiFi-GAN V2	926.0K (12.8×)	4.09 ± 0.26	3.10 [1.99, 4.21]	39.00 (0.91×)
HiFi-GAN V1	13,936.1K (192.7×)	**4.24 ± 0.22**	**2.65** [1.67, 3.63]	5.46 (0.13×)
Parallel WaveGAN	1,334.3K (18.5×)	3.70 ± 0.30	3.25 [2.09, 4.41]	1.97 (0.05×)
FreGrad	1,782.5K (24.7×)	3.27 ± 0.33	3.13 [2.04, 4.21]	1.14 (0.03×)
RNDVoC	3,572.4K (49.4×)	4.07 ± 0.25	3.42 [2.17, 4.68]	7.87 (0.18×)
MelGAN	4,266.1K (59.0×)	3.77 ± 0.32	3.19 [2.07, 4.30]	21.84 (0.51×)
BigVGAN-base	13,943.4K (192.8×)	4.18 ± 0.25	3.14 [2.00, 4.28]	1.87 (0.04×)
Ground Truth	–	4.37 ± 0.13	2.84 [1.81, 3.87]	–

is 3.28%, compared with 2.84% for ground-truth recordings. The paired difference is $+0.44$ pp (95% paired bootstrap CI: $[-0.03, +1.02]$), providing no clear evidence of increased WER on this evaluation set.

NanoVoco achieves the highest measured CPU throughput at $42.98\times$ real time, compared with $39.00\times$ for HiFi-GAN V2. Thus, the 12.8-fold difference in parameter count corresponds to only about 10% higher throughput. Parameter count does not determine runtime: computation also depends on temporal resolution, and CPU efficiency depends on memory traffic and operator implementation. Dense convolutions can benefit from efficient CPU kernels, whereas depthwise operations may be limited by memory access (Chen et al., 2023). DyT's element-wise $\tanh$ evaluation may also add overhead relative to LeakyReLU. Their contributions to this throughput difference remain unmeasured.

External checkpoints provide reference points, not matched-training comparisons. Assessing perceptual generalization requires listening tests and broader datasets (Appendix E).

5.3 End-to-End MCU Deployment

We integrate NanoVoco with Flite 2.2 and EfficientSpeech to build a complete TTS system on an Arduino Portenta H7 (STM32H747XI). The system runs on the 480-MHz Cortex-M7 using quantized neural models and only on-chip memory (2 MiB Flash and 1 MiB SRAM).

We test 128 texts five times each. The speech startup delay is 0.74 seconds (median), measured from the start of on-device text processing. After startup, the system generates audio fast enough to sustain uninterrupted playback in all 640 requests.

Quantization reduces the complete system's mean UTMOS from 3.30 to 3.23, a decrease of 0.07 points. These scores evaluate speech generated from text, whereas Table 1 evaluates the vocoder alone. Appendix G provides precise timing definitions, runtime and memory measurements, and verification that the quality-evaluation audio matches the MCU output exactly.

6 Conclusion

We show that the choice of upsampling method strongly affects rank-width reallocation benefits in compact neural vocoders. Repeating input features instead of inserting zeros improves these benefits across the evaluated convolutional ranks, with further support from intermediate upsampling configurations. This finding also holds under Avocodo supervision and in a BigVGAN-derived architecture. The improvement is strongest in our smallest models and substantially weaker at larger parameter budgets, highlighting the importance of upsampling choices under extreme compression. Guided by these findings, we develop NanoVoco, a 72.3K-parameter vocoder achieving a mean UTMOS of 3.71 on LJSpeech. Integrated into a complete TTS system, its quantized implementation streams speech in real time on a microcontroller using only on-chip memory, with a speech startup delay of 0.74 seconds (median).

AI USE STATEMENT

We used generative AI tools to assist with manuscript drafting and editing, research framing, and figure preparation. The reported measurements come from model evaluations and on-device experiments, not AI-generated data. AI-assisted material was reviewed for consistency with the experimental evidence. The authors take responsibility for the final text, claims, and artifacts.

ETHICS STATEMENT

This study uses the publicly available LJSpeech dataset. No participants were recruited, no new recordings were collected, and no human listening tests or animal experiments were conducted.

REPRODUCIBILITY STATEMENT

Appendices A–B cover architecture, training, evaluation, and seed robustness. Sections 3.3–4.4 describe the quality controls; remaining ablations and deployment protocols are detailed in Appendices C–G.

REFERENCES

Rowel Atienza. EfficientSpeech: An on-device text to speech model. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5. IEEE, 2023.

Taejun Bak, Junmo Lee, Hanbin Bae, Jinhyeok Yang, Jae-Sung Bae, and Young-Sun Joo. Avocodo: Generative adversarial network for artifact-free vocoder. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(11):12562–12570, 2023.

Jierun Chen, Shiu-hong Kao, Hao He, Weipeng Zhuo, Song Wen, Chul-Ho Lee, and S.-H. Gary Chan. Run, don't walk: Chasing higher FLOPS for faster neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12021–12031, 2023.

Dinh Son Dang, Tung Lam Nguyen, Bao Thang Ta, Tien Thanh Nguyen, Thi Ngoc Anh Nguyen, Dang Linh Le, Nhat Minh Le, and Van Hai Do. LightVoc: An upsampling-free GAN vocoder based on conformer and inverse short-time fourier transform. In *Proc. Interspeech 2023*, pp. 3043–3047, 2023.

Arne De Beer and Chang Gao. SlimTTS: Parameter- and compute-efficient neural text-to-speech synthesis for real-time inference. In *2026 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 4588–4592. IEEE, 2026.

Jianbo Guo, Yuxi Li, Weiyao Lin, Yurong Chen, and Jianguo Li. Network decoupling: From regular to depthwise separable convolutions. *arXiv preprint arXiv:1808.05517*, 2018.

Keith Ito and Linda Johnson. The LJ Speech dataset. `https://keithito.com/LJ-Speech-Dataset/`, 2017.

Rakesh Iyer. nVOC-22: A low cost mel spectrogram vocoder for mobile devices. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5. IEEE, 2023.

Won Jang, Dan Lim, Jaesam Yoon, Bongwan Kim, and Juntae Kim. UnivNet: A neural vocoder with multi-resolution spectrogram discriminators for high-fidelity waveform generation. In *Proc. Interspeech 2021*, pp. 2207–2211, 2021. doi: 10.21437/Interspeech.2021-1016.

Takuhiro Kaneko, Hirokazu Kameoka, Kou Tanaka, and Shogo Seki. MISRNet: Lightweight neural vocoder using multi-input single shared residual blocks. In *Proc. Interspeech 2022*, pp. 1631–1635, 2022a.

Takuhiro Kaneko, Kou Tanaka, Hirokazu Kameoka, and Shogo Seki. iSTFTNet: Fast and lightweight mel-spectrogram vocoder incorporating inverse short-time fourier transform. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6207–6211. IEEE, 2022b.

Takuhiro Kaneko, Hirokazu Kameoka, Kou Tanaka, and Shogo Seki. iSTFTNet2: Faster and more lightweight iSTFT-based neural vocoder using 1D-2D CNN. *arXiv preprint arXiv:2308.07117*, 2023.

Jungil Kong, Jaehyeon Kim, and Jaekyoung Bae. HiFi-GAN: Generative adversarial networks for efficient and high fidelity speech synthesis. *Advances in neural information processing systems*, 33:17022–17033, 2020.

Kundan Kumar, Rithesh Kumar, Thibault De Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre De Brebisson, Yoshua Bengio, and Aaron C Courville. MelGAN: Generative adversarial networks for conditional waveform synthesis. *Advances in neural information processing systems*, 32, 2019.

Sang-gil Lee, Wei Ping, Boris Ginsburg, Bryan Catanzaro, and Sungroh Yoon. BigVGAN: A universal neural vocoder with large-scale training. *arXiv preprint arXiv:2206.04658*, 2022.

Andong Li, Tong Lei, Zhihang Sun, Rilin Chen, Erwei Yin, Xiaodong Li, and Chengshi Zheng. Learning neural vocoder from range-null space decomposition. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pp. 8131–8140, 2025. doi: 10.24963/ijcai.2025/904.

Tan Dat Nguyen, Ji-Hoon Kim, Youngjoon Jang, Jaehun Kim, and Joon Son Chung. FreGrad: Lightweight and fast frequency-aware diffusion vocoder. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 10736–10740. IEEE, 2024.

Jordi Pons, Santiago Pascual, Giulio Cengarle, and Joan Serrà. Upsampling artifacts in neural audio synthesis. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3005–3009. IEEE, 2021.

Vadim Popov, Stanislav Kamenev, Mikhail Kudinov, Sergey Repyevsky, Tasnima Sadekova, Vitalii Bushaev, Vladimir Kryzhanovskiy, and Denis Parkhomenko. Fast and lightweight on-device tts with tacotron2 and lpcnet. In *Proc. Interspeech 2020*, pp. 220–224, 2020.

Takaaki Saeki, Detai Xin, Wataru Nakata, Tomoki Koriyama, Shinnosuke Takamichi, and Hiroshi Saruwatari. UTMOS: UTokyo-SaruLab system for VoiceMOS challenge 2022. *arXiv preprint arXiv:2204.02152*, 2022.

Hubert Siuzdak. Vocos: Closing the gap between time-domain and fourier-based neural vocoders for high-quality audio synthesis. In *International Conference on Learning Representations*, volume 2024, pp. 25719–25733, 2024.

Yusuke Sugawara, Sayaka Shiota, and Hitoshi Kiya. Convolutional neural networks without any checkerboard artifacts. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pp. 1317–1321. IEEE, 2018.

Ling Tang, Wen Shen, Zhanpeng Zhou, Yuefeng Chen, and Quanshi Zhang. Defects of convolutional decoder networks in frequency representation. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 33758–33791. PMLR, 2023.

Ryuichi Yamamoto, Eunwoo Song, and Jae-Min Kim. Parallel WaveGAN: A fast waveform generation model based on generative adversarial networks with multi-resolution spectrogram. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6199–6203. IEEE, 2020.

Bohan Zhai, Tianren Gao, Flora Xue, Daniel Rothchild, Bichen Wu, Joseph E Gonzalez, and Kurt Keutzer. SqueezeWave: Extremely lightweight vocoders for on-device speech synthesis. *arXiv preprint arXiv:2001.05685*, 2020.

Jiachen Zhu, Xinlei Chen, Kaiming He, Yann LeCun, and Zhuang Liu. Transformers without normalization. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14901–14911. IEEE, 2025.

A HiFi-GAN Configurations

Shared architecture. Our in-repository generators are derived from HiFi-GAN V1 (Kong et al., 2020). All configurations use resblock='1', an input projection `conv_pre` (kernel size 7), four upsampling stages with rates $[8, 8, 2, 2]$ and kernel sizes $[16, 16, 4, 4]$, residual blocks with kernel sizes $[3, 7, 11]$ and dilation $(1, 3, 5)$, and a final waveform projection (`conv_post`, kernel size 7). These architectural settings are shared across all channel-width, rank, and upsampling configurations.

Upsampler-refiner decomposition. In this paper, we separate each upsampling stage into a parameter-free temporal-expansion operator, termed the *upsampler*, followed by the learned processing, termed the *refiner*. In the implementation, the V1-derived generator realizes temporal upsampling using a transposed convolution with symmetric padding. For analysis and intervention, we rewrite this operation exactly as zero insertion followed by symmetric boundary zero-padding and a stride-one convolution with zero internal padding. For a transposed convolution with kernel size k, stride r, and symmetric padding p, let

$$q = k - 1 - p \tag{6}$$

denote the amount of boundary padding applied on each side. After inserting $r - 1$ zeros between adjacent input features, the expanded sequence is padded by q zeros on both temporal boundaries and passed to a stride-one convolution with zero internal padding. With the corresponding kernel reindexing,

$$\mathrm{TConv}_{k,r,p}(X; W) = \mathrm{Conv}_{k,1,0}\left(\mathrm{Pad}_{q,q}\left(U_{\mathrm{ZI}}^{(r)}(X)\right); \widetilde{W}\right), \qquad q = k - 1 - p, \tag{7}$$

where $\widetilde{W}$ denotes the reindexed convolution kernel. Thus, the decomposition is functionally identical to the original transposed convolution and changes neither the ZI generator function nor its parameter count.

For an input of length L, ZI produces $(L - 1)r + 1$ positions and uses left and right padding (q, q). SR repeats each input feature r times, producing Lr positions, and uses padding $(q, q - r + 1)$. Reducing the right padding by $r - 1$ compensates for the additional positions at the end of the SR sequence. After the same stride-one convolution, both paths have output length $(L - 1)r + k - 2p$, which equals Lr for our stage settings.

Equivalently, ZI can be represented on a full Lr-position grid with $r - 1$ trailing zeros and padding $(q, q - r + 1)$. This convention also accommodates U_λ in Sec. 4.2: at $\lambda = 0$, the trailing zeros combine with the right padding to recover the original ZI path; at $\lambda = 1$, the operator gives SR. All matched ZI-SR configurations retain the same learned convolutions, channel widths, and rank settings, and therefore have identical generator parameter counts.

Local-DSC comparison. The nVOC-inspired configuration in Sec. 3.1 uses SR and replaces only the first post-expansion convolution in each stage with DSC; the subsequent residual convolutions remain standard. It has initial width 36, uses LeakyReLU, and contains 76,225 generator parameters after weight-normalization removal. It follows the same seed-1234 training protocol, 800,000-step final checkpoint selection, and 128-utterance evaluation as the broader SR-DSC configuration. This is an architectural comparison inspired by nVOC-22, not a reproduction of its complete published system.

Rank-constrained convolutions. Rank constraints are applied to all eligible convolutional layers. For a candidate layer ℓ with kernel size K_ℓ and output width C_ℓ, the maximum kernel-slice rank is

$$R_\ell^{\max} = \min(K_\ell, C_\ell). \tag{8}$$

We denote the progression from rank-one DSC to unconstrained StdC by a nominal rank ratio $\rho \in [0, 1]$, where $\rho = 0$ corresponds to rank-one DSC and $\rho = 1$ corresponds to StdC. The candidate integer rank is

$$\widetilde{R}_\ell = \max(1, \mathrm{round}(\rho R_\ell^{\max})), \tag{9}$$

where rounding follows the nearest-integer rule used in our implementation.

Considering kernel weights only, a dense convolution contains $C_{\mathrm{in},\ell} C_\ell K_\ell$ parameters, whereas its rank-$\widetilde{R}_\ell$ factorization contains $C_{\mathrm{in},\ell}\widetilde{R}_\ell(C_\ell + K_\ell)$ parameters. We therefore use the factorized implementation only when

$$\widetilde{R}_\ell < \frac{C_\ell K_\ell}{C_\ell + K_\ell}. \tag{10}$$

Otherwise, the layer is implemented as a dense convolution. Hence, an intermediate nominal rank ratio may produce a mixture of dense and rank-factorized layers because the break-even rank depends on both K_ℓ and C_ℓ.

At $\rho = 1$, the candidate rank is $R_\ell^{\max}$, causing every eligible layer to trigger the dense fallback and recover the StdC configuration. At $\rho = 0$, the candidate rank is one, yielding rank-one depthwise-pointwise convolutions for all eligible layers. In these HiFi-GAN configurations, the final waveform projection remains dense because its single output channel makes factorization parameter-inefficient.

Evaluated generator variants. Table 2 summarizes the HiFi-GAN generator configurations used throughout the experiments. At the approximately 72K operating point, we evaluate both a fixed-width rank series and a comparable-budget rank-width series. The fixed-width series keeps the initial width at 32, whereas the comparable-budget series increases channel width as rank is reduced. The DynamicTanh variants add the learnable activation parameters to the two endpoint configurations. The larger-budget experiments evaluate only the dense StdC and rank-one DSC endpoints. Within a matched architecture, ZI and SR use the same parameterization and parameter count.

Layer-wise rank-width configurations. For the comparable-budget series, Table 3 reports the actual layer-wise ranks and dense-fallback decisions produced by the rank-selection rule above. Numerical entries denote factorized ranks, while D denotes a dense convolution. Tuples list the four stages in input-to-output order. ZI and SR share the same layer-wise configuration at each nominal rank ratio.

B HiFi-GAN Training and Evaluation Protocol

B.1 Training objectives

The HiFi-GAN experiments use the original multi-period discriminator (MPD) and multi-scale discriminator (MSD), with adversarial, feature-matching, and mel-spectrogram losses. The generator objective is

$$\mathcal{L}_G = \mathcal{L}_{\mathrm{adv}} + 2\mathcal{L}_{\mathrm{FM}} + 45\mathcal{L}_{\mathrm{mel}}, \tag{11}$$

where $\mathcal{L}_{\mathrm{mel}}$ is the L1 loss between the target and generated mel spectrograms. The same loss definitions and weights are used for all HiFi-GAN configurations.

B.2 Training protocol

All in-repository HiFi-GAN models use LJSpeech-1.1 (Ito & Johnson, 2017) with a fixed split of 12,950 training and 150 validation utterances. Audio is sampled at 22.05 kHz and represented using 80-band mel spectrograms with an FFT size of 1024, hop size of 256, window length of 1024, and mel frequency range of 0–8 kHz. Training uses seed 1234 unless otherwise stated; both the six A/B/C endpoint controls and final NanoVoco are also trained with seeds 2345 and 3456 (Appendices B.4 and D). All runs use a batch size of 16, waveform segments of 8192 samples, and Adam with $\beta_1 = 0.8$, $\beta_2 = 0.99$, and an initial learning rate of 2×10^{-4}. The learning rate is multiplied by 0.999 after each epoch. Each configuration is trained for 800,000 steps, and the final checkpoint is used for evaluation without validation-based checkpoint selection.

B.3 Quality evaluation

Quality evaluation uses a fixed subset of 128 utterances from the validation split, disjoint from the training set of our models. We compute one UTMOS22-strong score per utterance using `tarepan/SpeechMOS:v1.2.0` and report mean $\pm$ standard deviation across utterances. Unless explicitly labeled as across-seed statistics, standard deviations reflect variation across evaluation utterances rather than training runs.

Table 2: HiFi-GAN generator variants used in the experiments. Parameter counts include the complete generator after weight-normalization removal. Matched ZI and SR configurations have identical parameter counts.

Series	Upsampling / variant	ρ	Initial width	Parameters
Comparable-budget rank-width series				
	ZI / SR	100% (StdC)	32	71,777
	ZI / SR	75%	33	70,022
	ZI / SR	50%	34	68,489
	ZI / SR	25%	52	69,773
	ZI / SR	0% (DSC)	84	69,101
Fixed-width rank series				
	ZI / SR	100% (StdC)	32	71,777
	ZI / SR	75%	32	69,361
	ZI / SR	50%	32	63,445
	ZI / SR	25%	32	32,461
	ZI / SR	0% (DSC)	32	15,813
SR replacement-scope comparison (LeakyReLU)				
nVOC-inspired	SR / local DSC	—	36	76,225
Broader refiner	SR / DSC	0% (DSC)	84	69,101
DynamicTanh variants				
	ZI / SR + DyT	100% (StdC)	32	73,058
	ZI / SR + DyT	0% (DSC)	84	72,310
Larger-budget configurations				
∼250K	ZI / SR	100% (StdC)	64	250,033
	ZI / SR	0% (DSC)	176	252,387
∼535K	ZI / SR	100% (StdC)	96	534,769
	ZI / SR	0% (DSC)	265	532,371
∼926K	ZI / SR	100% (StdC)	128	925,985
	ZI / SR	0% (DSC)	355	923,332

Local DSC replaces only the first post-expansion convolution in each stage; other convolutions remain standard, so no global ρ applies. The broader refiner uses DSC in all eligible convolutions; its row repeats the SR endpoint above for comparison.

Table 3: Layer-wise configurations for the comparable-budget rank-width series. D denotes a dense convolution; numerical entries denote factorized ranks. Tuples list the four stages in input-to-output order.

Nominal rank ratio ρ	100%	75%	50%	25%	0%
`conv_pre`, $K = 7$	D	5	4	2	1
Post-expansion convs., $K = [16, 16, 4, 4]$	(D, D, D, D)	(D, D, D, D)	(8, 4, D, 1)	(4, 3, 1, 1)	(1, 1, 1, 1)
Residual blocks, $K = [3, 3, 3, 3]$	(D, D, D, D)	(2, 2, D, D)	(2, 2, D, 1)	(1, 1, 1, 1)	(1, 1, 1, 1)
Residual blocks, $K = [7, 7, 7, 7]$	(D, D, D, D)	(D, D, D, D)	(4, D, 2, 1)	(2, 2, 2, 1)	(1, 1, 1, 1)
Residual blocks, $K = [11, 11, 11, 11]$	(D, D, D, D)	(D, D, D, D)	(6, 4, 2, 1)	(3, 3, 2, 1)	(1, 1, 1, 1)

For Table 1, we also measure word error rate (WER) on the same 128 utterances per model and their ground-truth recordings. We transcribe the audio with Whisper large-v3 via `faster-whisper` using int8 inference and compare each transcription with its corresponding LJSpeech reference text. For each utterance i, we compute $w_i = (S_i + D_i + I_i)/N_i$ using `jiwer`, where S_i, D_i, and I_i count word substitutions, deletions, and insertions, and N_i is the number of reference words. We report the unweighted mean $\bar{w} = n^{-1} \sum_{i=1}^{n} w_i$ over $n = 128$ utterances (macro-average), rather than pooling word counts across utterances. Approximate 95% confidence intervals use the normal approximation $\bar{w} \pm 1.96\, s/\sqrt{n}$, where s is the sample standard deviation of the utterance-level WERs computed with denominator $n - 1$. Means and interval endpoints are expressed as percentages. The Ground Truth

row applies the same evaluation to the original recordings. These intervals describe uncertainty across utterances for each fixed model and recognizer; they do not estimate paired between-model differences.

To compare NanoVoco with ground-truth recordings, we pair their utterance-level WERs by utterance identifier, using the same reference text for each pair. We compute $d_i = w_i^{\mathrm{Nano}} - w_i^{\mathrm{GT}}$ and estimate a 95% paired bootstrap interval for the mean difference by resampling the 128 utterance pairs with replacement 10,000 times. The mean difference is $+0.44$ percentage points, with a 95% interval of $[-0.03, +1.02]$ percentage points. This interval includes zero; it does not establish equivalence or non-inferiority to ground-truth recordings.

B.4 Three-seed endpoint controls

We repeat the six HiFi-GAN A/B/C controls with training seeds 1234, 2345, and 3456, using the same training protocol and 128 evaluation utterances. A is the width-32 StdC reference, B is width-32 DSC, and C is width-84 DSC; all six controls use LeakyReLU. Table 4 reports each trained model's mean and standard deviation across utterances.

For each seed, we compute the rank-reduction cost, widening gain, and reallocation benefit from that seed's configuration means using Eq. 3, and then compute the increase in reallocation benefit from ZI to SR. Table 5 summarizes these contrasts. Its final column reports mean $\pm$ sample SD across the three training seeds, describing training variability on the fixed evaluation set. This differs from the utterance-level SD in Table 4; neither is a confidence interval.

The reallocation benefit is negative under ZI and positive under SR in each of the three tested seeds. SR also makes rank-reduction cost less negative and increases widening gain in each of these seeds, with a mean interaction of $+2.007 \pm 0.03702$ UTMOS points. These repeats provide a descriptive check of the 72K HiFi-GAN endpoint interaction on a fixed evaluation set. The observed agreement is limited to these three seeds and this setting; intermediate sweeps, transfer experiments, and larger-budget comparisons use seed 1234. Three-seed results for the final DyT model are reported in Appendix D.

Table 4: HiFi-GAN endpoint quality across three training seeds. Each entry is UTMOS mean $\pm$ SD across the same 128 utterances. A and B have width 32; C has width 84.

Configuration	Seed 1234	Seed 2345	Seed 3456
ZI: A (StdC)	2.753 ± 0.3187	2.689 ± 0.3757	2.685 ± 0.2921
ZI: B (DSC)	1.292 ± 0.03700	1.290 ± 0.05030	1.287 ± 0.03370
ZI: C (wide DSC)	1.406 ± 0.1285	1.403 ± 0.1361	1.393 ± 0.1051
SR: A (StdC)	2.662 ± 0.3420	2.589 ± 0.3569	2.607 ± 0.3381
SR: B (DSC)	1.928 ± 0.3460	1.874 ± 0.3703	1.860 ± 0.3125
SR: C (wide DSC)	3.326 ± 0.3421	3.345 ± 0.4167	3.283 ± 0.3413

Table 5: Endpoint contrasts in UTMOS points, computed within each training seed. The final column is mean $\pm$ sample SD across three seeds. The final row reports the increase in reallocation benefit from ZI to SR. Differences are computed before display rounding.

Contrast	Seed 1234	Seed 2345	Seed 3456	Across seeds
ZI: Rank-reduction cost	-1.461	-1.399	-1.398	-1.420 ± 0.03611
ZI: Widening gain	0.1141	0.1131	0.1064	0.1112 ± 0.004187
ZI: Reallocation benefit	-1.347	-1.286	-1.292	-1.308 ± 0.03374
SR: Rank-reduction cost	-0.7342	-0.7149	-0.7462	-0.7318 ± 0.01579
SR: Widening gain	1.398	1.471	1.422	1.430 ± 0.03701
SR: Reallocation benefit	0.6638	0.7558	0.6761	0.6986 ± 0.04995
Benefit increase (SR–ZI)	2.011	2.042	1.968	2.007 ± 0.03702

B.5 EXTENSION TO LARGER PARAMETER BUDGETS

Table 6 compares narrow StdC (A) and budget-comparable wide DSC (C) under both ZI and SR at approximately 72K, 250K, 535K, and 926K parameters. Within each configuration, the ZI-SR pair has identical channel width and parameter count. All configurations in this budget comparison are trained in-house, including the width-128 StdC reference at approximately 926K parameters. Each uses training seed 1234 and LeakyReLU, is evaluated at the final 800,000-step checkpoint, and is scored on the same 128 utterances under the protocol in Appendix B.

Table 6: Quality across parameter budgets. UTMOS is mean ± SD across evaluation utterances. Parameter counts include the complete generator after weight-normalization removal.

Budget	Refiner	Upsampling	Initial width	Parameters	UTMOS
∼72K	StdC	ZI	32	71,777	2.753 ± 0.3187
	StdC	SR	32	71,777	2.662 ± 0.3420
	DSC	ZI	84	69,101	1.406 ± 0.1285
	DSC	SR	84	69,101	3.326 ± 0.3421
∼250K	StdC	ZI	64	250,033	3.367 ± 0.3362
	StdC	SR	64	250,033	3.257 ± 0.3352
	DSC	ZI	176	252,387	3.599 ± 0.3348
	DSC	SR	176	252,387	3.813 ± 0.3492
∼535K	StdC	ZI	96	534,769	3.698 ± 0.3256
	StdC	SR	96	534,769	3.588 ± 0.3346
	DSC	ZI	265	532,371	3.795 ± 0.3349
	DSC	SR	265	532,371	3.941 ± 0.2931
∼926K	StdC	ZI	128	925,985	4.087 ± 0.2592
	StdC	SR	128	925,985	3.897 ± 0.2442
	DSC	ZI	355	923,332	3.869 ± 0.2791
	DSC	SR	355	923,332	4.112 ± 0.2558

At each budget, we compute the reallocation benefit as $Q_u(C) - Q_u(A)$ and compare this benefit between SR and ZI. For 72K, 250K, 535K, and 926K, respectively, the reallocation benefit under ZI is -1.347, $+0.2324$, $+0.09690$, and -0.2179, whereas the benefit under SR is $+0.6638$, $+0.5562$, $+0.3534$, and $+0.2148$. The corresponding interactions are $+2.011$, $+0.3238$, $+0.2565$, and $+0.4327$ UTMOS points, computed before display rounding. These are seed-1234 contrasts; the three-seed endpoint results at 72K are reported separately in Appendix B.4.

Replacing ZI with SR raises wide-DSC mean UTMOS by 1.920, 0.2138, 0.1465, and 0.2427 points, while changing the corresponding StdC means by -0.09070, -0.1100, -0.1100, and -0.1900. The interaction remains positive at every evaluated budget but is substantially weaker at 250K–926K than at 72K. At 250K and 535K, ZI-DSC already exceeds ZI-StdC, so the severe compression failure at 72K does not generalize to all budgets. At 926K, SR-DSC exceeds SR-StdC by 0.2148 points and ZI-StdC by 0.02480 points.

The StdC and wide-DSC configurations identify the reallocation benefit and its change with upsampling. The larger-budget runs do not include the same-width DSC control B, so they do not separately identify rank-reduction cost or widening gain. The displayed SDs describe variation across utterances; no confidence interval or significance claim for these contrasts is inferred from them.

C TRANSFER EXPERIMENTS

Both transfer settings use the A/B/C design from Sec. 3.3: A is the narrow StdC configuration with initial width 32, B is the same-width rank-one DSC configuration, and C is the wider rank-one DSC configuration at a generator budget comparable to A (initial width 84 for Avocodo and 83 for BigVGAN). The corresponding ZI and SR variants differ only in the temporal-expansion rule unless otherwise specified below.

Table 7: Transfer quality results. UTMOS is reported as mean ± standard deviation across evaluation utterances. A and C have comparable generator parameter budgets; A and B share initial width. The lower rows are differences computed before display rounding, using Eq. 3.

	Avocodo supervision		BigVGAN-derived	
Refiner configuration	ZI	SR	ZI	SR
A: narrow StdC	1.733 ± 0.3171	1.849 ± 0.3373	2.500 ± 0.3304	2.376 ± 0.3430
B: same-width DSC	1.283 ± 0.03790	1.459 ± 0.2089	1.255 ± 0.02900	1.304 ± 0.08080
C: wide DSC	1.812 ± 0.3496	3.048 ± 0.3288	1.443 ± 0.1709	2.507 ± 0.3694
Rank-reduction cost	-0.4504	-0.3895	-1.245	-1.071
Widening gain	0.5290	1.589	0.1887	1.203
Reallocation benefit	$+0.07860$	$+1.200$	-1.056	$+0.1315$

Table 8: Parameter counts for the Avocodo transfer configurations. Training-time counts include the auxiliary waveform-projection heads, which are removed for inference.

Configuration	Training params.	Auxiliary heads	Inference params.
A: StdC, $w = 32$	71,855	70	71,785
B: DSC, $w = 32$	15,919	94	15,825
C: DSC, $w = 84$	69,370	239	69,131

Table 7 reports UTMOS means and utterance-level SDs for the seed-1234 transfer controls, together with their cost/gain decomposition.

C.1 Avocodo Supervision

The Avocodo generators retain the HiFi-GAN stage structure, upsampling factors, residual-block configuration, and backbone rank-width settings defined in Appendix A. The final waveform projection follows the Avocodo architecture and uses kernel size 11 rather than the kernel size 7 used in the HiFi-GAN configurations. Configuration A retains a dense final projection, whereas B and C use DSC for the final projection retained at inference. Training replaces the HiFi-GAN adversarial supervision with the Avocodo scheme, including its artifact-aware discriminators and two learnable intermediate waveform-projection heads. These auxiliary heads participate in training but are removed for inference. All remaining training and quality-evaluation settings follow Appendix B. Table 8 reports training- and inference-time generator parameter counts. The main-text budget comparisons use the inference-time counts. The larger projection kernel and, for B and C, the DSC output projection increase these counts by 8, 12, and 30 parameters for A, B, and C, respectively, relative to their HiFi-GAN counterparts.

C.2 BigVGAN-Derived Generators

The BigVGAN-derived experiments retain the official BigVGAN-v1 generator backbone, consisting of `conv_pre`, four upsampling-AMPBlock1 stages, `activation_post`, and `conv_post`. The AMPBlock1 structure, Snake/SnakeBeta activations with anti-aliasing through `Activation1d`, and the BigVGAN-v1 discriminator design are retained. Configuration A uses dense convolutions, whereas B and C replace convolutions constructed through the shared `make_conv1d` path with rank-one depthwise-separable convolutions. This replacement covers `conv_pre`, the learned post-expansion convolutions, both convolution stacks in each AMPBlock1, and `conv_post`. ZI and SR have identical generator parameter counts within each A/B/C configuration.

Training follows Appendix B except for the discriminator and optimization settings. BigVGAN retains the MPD but replaces the MSD with the UnivNet-style multi-resolution discriminator (MRD) (Jang et al., 2021), using resolutions $[[1024, 120, 600], [2048, 240, 1200], [512, 50, 240]]$. Training uses a batch size of 32, an initial learning rate of 10^{-4}, and `ExponentialLR` with $\gamma = 0.9999996$ updated every iteration.

Table 9: Generator parameter counts for the BigVGAN-derived transfer configurations.

Configuration	Generator params.
A: StdC, $w = 32$	72,861
B: DSC, $w = 32$	16,901
C: DSC, $w = 83$	68,982

Table 9 reports generator parameter counts after weight-normalization removal. BigVGAN has no auxiliary training-only waveform heads, so the training- and inference-time generator counts coincide. The counts need not match the corresponding HiFi-GAN configurations exactly because BigVGAN retains learnable Snake/SnakeBeta activations and uses a different DSC implementation and replacement scope. The A–C comparison is therefore budget-comparable rather than identical.

D Activation Ablation

Activation implementation. We replace each intermediate LeakyReLU with an independent DynamicTanh (DyT) module (Zhu et al., 2025), leaving activation locations and the final output Tanh unchanged. For an input feature vector $\mathbf{x}$,

$$\mathrm{DyT}(\mathbf{x}) = \boldsymbol{\gamma} \odot \tanh(\alpha \mathbf{x}) + \boldsymbol{\beta}, \tag{12}$$

where α is a learnable scalar and $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learnable per-channel affine parameters. Each module is initialized with $\alpha = 1$, $\boldsymbol{\gamma} = \mathbf{1}$, and $\boldsymbol{\beta} = \mathbf{0}$. Within each evaluated activation pair, the remaining architectural choices are held fixed.

Table 10: Activation ablation. Each score is UTMOS mean $\pm$ SD across 128 utterances for the indicated training seed. Δ is DyT minus LeakyReLU, computed before display rounding. The SR-DSC + DyT configurations are the final NanoVoco.

Configuration	Width	Seed	LeakyReLU	DyT	Δ UTMOS
ZI-StdC	32	1234	2.753 ± 0.3187	2.726 ± 0.3616	-0.0266
SR-StdC	32	1234	2.662 ± 0.3420	2.713 ± 0.3016	$+0.0503$
ZI-DSC	84	1234	1.406 ± 0.1285	1.349 ± 0.0761	-0.0571
SR-DSC	84	1234	3.326 ± 0.3421	3.711 ± 0.3124	$+0.3851$
SR-DSC	84	2345	3.345 ± 0.4167	3.651 ± 0.3564	$+0.3057$
SR-DSC	84	3456	3.283 ± 0.3413	3.621 ± 0.3016	$+0.3380$

Dependence on configuration. The largest observed gain occurs with SR and wide DSC; replacing LeakyReLU with DyT does not improve either ZI configuration (Table 10). These comparisons concern the evaluated width-32 StdC and width-84 DSC models, rather than isolating convolution type at fixed width. They support DyT as a refinement of the selected SR-DSC architecture, not as a uniformly beneficial activation replacement.

Activation gains across training seeds. For SR-DSC, DyT improves mean UTMOS in each of the three tested seeds (Table 10). The within-seed gains are $+0.3851$, $+0.3057$, and $+0.3380$ points, with mean $+0.3429 \pm 0.03993$ across seeds. The final NanoVoco achieves 3.661 ± 0.04611 UTMOS, compared with 3.318 ± 0.03190 for its LeakyReLU counterpart. These summaries give the mean and sample SD of the three seed-level means or contrasts on the fixed 128-utterance evaluation set; they do not denote utterance-level SD or confidence intervals.

The quality-efficiency comparison in Table 1 retains the seed-1234 NanoVoco checkpoint used for its UTMOS, WER, and CPU measurements. Its 3.71 ± 0.31 UTMOS entry is a single-checkpoint utterance summary, distinct from the three-seed summary above. Additional seeds are evaluated here for UTMOS only. The positive DyT gains describe within-architecture comparisons; they do not establish an advantage over external vocoders across training seeds.

E External Baselines and Evaluation Scope

The external baselines in Table 1 are evaluated using their official pretrained checkpoints and native conditioning pipelines. We include only models with publicly available official implementations and pretrained weights. Lightweight alternatives such as nVOC-22 (Iyer, 2023) and MISRNet (Kaneko et al., 2022a) are excluded because we could not obtain both resources for this evaluation and could not verify the fidelity of an independent reimplementation. NanoVoco is trained using the protocol in Appendix B.

Two caveats govern the interpretation of Table 1. First, quality is assessed using automatic metrics on 128 LJSpeech utterances; listening tests and broader datasets are needed to establish whether the observed UTMOS gains translate into perceptual improvements and generalize beyond this setting. Second, the external checkpoints retain their original training and validation splits, training settings, and frontends. Evaluating them on the same utterances does not ensure that these utterances were held out during every checkpoint's training. Their rows therefore serve as reference points for the quality-efficiency comparison, rather than controlled tests of architectural differences under a common training protocol.

F CPU Benchmark Protocol

Execution environment. CPU inference is benchmarked on an AMD Ryzen 9 9950X3D running Linux. Each inference process is pinned to a single logical CPU. PyTorch intra-op and inter-op thread counts, together with all BLAS/OpenMP thread counts, are fixed to one. All models are evaluated in FP32 with batch size one and in evaluation/inference mode. CUDA execution, dynamic threading, quantization, and additional JIT compilation are disabled.

Input and timing boundary. All models are benchmarked on the same fixed-length evaluation segment using 100 consecutive conditioning frames from a 22.05-kHz LJSpeech evaluation recording. Each model retains its native conditioning transform to preserve compatibility with its released implementation or checkpoint. Timing covers the complete model-specific conditioning-to-waveform generation path, including stochastic noise generation and model-specific reconstruction operations where applicable. Model loading, conditioning-feature extraction and normalization, file I/O, and input preparation are excluded from timing. FreGrad is evaluated using its six-step fast-sampling procedure.

Repetitions and aggregation. For each model, we launch three independent worker processes. Each worker performs 10 warm-up calls followed by 30 sequential timed inference runs, yielding 90 timed runs per model. Models are executed sequentially, and their execution order is independently shuffled across the three measurement rounds. We report the median wall-clock latency over all 90 timed runs. CPU throughput is computed as the actual generated audio duration divided by this median latency; $1\times$ therefore corresponds to real-time synthesis, and higher values indicate faster inference.

G MCU Deployment and Evaluation

G.1 Experimental setup and timing boundaries

The deployed pipeline is ASCII text $\rightarrow$ Flite 2.2 CMU lexicon/LTS $\rightarrow$ EfficientSpeech $\rightarrow$ NanoVoco $\rightarrow$ PCM16 $\rightarrow$ I^2S DMA. The acoustic model and vocoder contain 266,417 and 72,310 parameters, respectively. Both neural models use A16W8 convolution with 16-bit activations, 8-bit weight precision, and 32-bit accumulation. Inference runs on the Arduino Portenta H7's STM32H747XI Cortex-M7 at 480 MHz with batch size one; DMA transmission overlaps synthesis. Model state, inference buffers, and the audio ring all reside on chip.

After one untimed warm-up, we run five passes over the same 128 held-out LJSpeech texts used for quality evaluation. Inputs span 11–112 phonemes, 89–795 mel frames, and 1.033–9.230 s of audio at the nominal 22.05-kHz sample rate. The repetitions measure repeatability on one board, not 640 independent utterances. The timing origin for request latency is the start of processing with

the complete newline-terminated text already in MCU memory; host-to-MCU transfer and byte reception are excluded.

Compute time and request latency. For utterance i in pass p, t_{ip}^{comp} is the sum of mutually exclusive DWT-timed segments covering Flite, the acoustic model, the vocoder including its per-request initialization, and explicit handoff and PCM16 conversion work. The conversion segment also includes PCM CRC updates and cache maintenance. Untimed request control, I^{2}S setup, buffer-space waits, and the final DMA drain are not included in this sum. Short segments use unsigned 32-bit cycle differences accumulated in 64-bit counters, avoiding the 8.948-s single-window limit at 480 MHz. With N_{ip} generated samples and $d_{ip} = N_{ip}/22{,}050$, we define

$$r_{ip}^{\mathrm{comp}} = \frac{t_{ip}^{\mathrm{comp}}}{d_{ip}}, \qquad R_p^{\mathrm{comp}} = \frac{\sum_{i=1}^{128} t_{ip}^{\mathrm{comp}}}{\sum_{i=1}^{128} d_{ip}}. \tag{13}$$

The primary compute RTF is the median of R_p^{comp} over the five passes, and its reciprocal is compute throughput. Each pass generates 733.9 s of audio under the nominal sample rate.

Separately, an enclosing elapsed-time measurement runs from the request origin through transmission of the final I^{2}S frame, including all intervening processing, control, initialization, and output waits. Elapsed timestamps are corrected by $5/6$ for the firmware's change from the core's 400-MHz timing base to 480 MHz. This request-completion interval is distinct from compute time and from the startup latency defined below.

G.2 Throughput and continuous output

The median corpus compute RTF is 0.9876 ($1.013\times$ compute throughput). The five pass values range from 0.987633 to 0.987638. All 128 utterances remain below compute RTF one in every pass; the maximum individual-run value is 0.9917. All 640 requests complete with zero reported I^{2}S underflows, clipped PCM samples, or incomplete transmissions. Output PCM16 CRCs agree across all five passes for every utterance, and no run uses external SDRAM. These observations establish continuous digital audio output on the evaluated workload after the configured startup buffer is filled.

To distinguish across-text variation from run-to-run repeatability, we compute a 95% utterance-bootstrap interval for the primary corpus RTF. Each of 10,000 resamples draws 128 utterance identifiers with replacement, retains all five measurements for each selected identifier, recomputes each pass's duration-weighted corpus RTF, and takes the median across passes. The 2.5th and 97.5th percentiles are $[0.9873, 0.9879]$ (random seed 1729). This interval describes utterance-resampling uncertainty for the fixed model and board, not variation across hardware or training runs.

Table 11 decomposes pass 2, whose corpus RTF equals the median. NanoVoco accounts for 92.82% of timed computation; handoff and conversion contribute a further 0.3659%. The median corpus request-completion ratio, computed as total elapsed time through final-frame transmission divided by total nominal audio duration in each pass, is 1.129. This ratio includes startup followed by playback at the output clock rate; it is not the compute RTF and need not be below one for underflow-free streaming.

Table 11: Timed computation in the pass with median corpus compute RTF. Contributions share the generated-duration denominator; values are rounded independently.

Component	Time/audio second (s/s)	Share (%)
Flite frontend	1.294×10^{-5}	0.001310
EfficientSpeech	0.06727	6.812
NanoVoco	0.9167	92.82
Handoff, conversion, and other timed overhead	0.003614	0.3659
Total	0.9876	100.0

G.3 STARTUP AND PLAYBACK LATENCY

We distinguish initial PCM availability from the start of continuous output. The first event occurs when 256 PCM16 frames are ready. Formal time to first audio (TTFA) ends when the output buffer reaches the configured 8192-frame threshold and I^2S DMA starts. The *speech startup delay* reported in the main text refers to this TTFA, rounded to 0.74 seconds for its median. Both are elapsed intervals from the same request origin and include intervening control and initialization. The 9216-frame DMA ring holds duplicated mono samples in two 16-bit channels. The initial reserve absorbs the vocoder's bursty output schedule while synthesis continues during transmission. TTFA excludes DAC, amplifier, loudspeaker, and physical acoustic-onset latency. The measured mean I^2S frame rate is 22,059.66 Hz; compute RTF consistently uses the nominal 22,050 Hz.

For each utterance, we take the median over its five repetitions, then summarize the resulting 128 values. The median first-256-frame latency is 364.4 ms. Formal TTFA has a median of 744.1 ms, P90 of 786.6 ms, P95 of 803.1 ms, and a range of 644.0–815.2 ms. These are workload quantiles, not confidence intervals.

G.4 MEMORY FOOTPRINT AND DEPLOYMENT REPRESENTATION

The full pipeline fits in the device's internal Flash and data SRAM without external SDRAM (Table 12). The firmware occupies 1,823,784 B, or 99.39% of the 1,835,008-B M7 application partition. Static data-SRAM allocation is 991,616 B across four banks, out of a combined capacity of 1,015,808 B (992 KiB). These allocations include reserved stacks, heap, and inference buffers. The AXI allocation includes a 1,024-B exception/main-stack reservation, and D2 includes a fixed 16-KiB Mbed Ethernet/lwIP section. The audio ring reuses the existing decoder/output scratch allocation. Because buffers have bank-specific placement, the aggregate free space cannot be treated as one shared memory pool.

Table 12: Firmware footprint and linker-allocated data SRAM, in bytes. Flash capacity refers to the M7 application partition; runtime reservations are included in the SRAM allocations.

Region	Allocated	Capacity	Unallocated
Flash application partition	1,823,784	1,835,008	11,224
DTCM	125,440	131,072	5,632
D1 AXI SRAM	524,288	524,288	0
D2 SRAM	278,528	294,912	16,384
D3 SRAM4	63,360	65,536	2,176
Data SRAM total	991,616	1,015,808	24,192
External SDRAM	0	—	—

From parameters to deployment bytes. The 72.3K parameter count describes the learned vocoder, not its complete deployment footprint. Tightly packing the quantized weights alone would require substantially less storage than the deployed representation. Our implementation instead trades storage for faster inference through weight rearrangement and duplication, precomputed depthwise data, and lookup tables (Table 13). In particular, W8 specifies numerical precision: quantized vocoder weights use `int16` containers to support paired DSP multiply-accumulate operations. DyT lookup tables occupy 576,666 B, and the full NanoVoco representation occupies 870,632 B of Flash. Together with EfficientSpeech, the two model representations occupy 1,164,776 B. The SRAM allocations additionally cover intermediate activations, inference scratch space, audio buffering, and runtime reservations; they are not a measure of weight storage.

The resulting pipeline sustains real-time streaming on the evaluated workload while nearly filling the available Flash and SRAM and operating with little compute headroom. These narrow resource margins illustrate the practical value of extreme vocoder compression: a compact learned model must leave room for the execution data and buffers needed to run the complete TTS system within the MCU's storage and timing constraints.

Table 13: Flash usage of the complete firmware and the NanoVoco representation. Panel (b) partitions the NanoVoco entry in panel (a); it is already included in the firmware total.

(a) Firmware Component	Bytes
NanoVoco deployment data	870,632
EfficientSpeech deployment data	294,144
Frontend, program, and other data	659,008
Total	1,823,784
(b) NanoVoco deployment data	
Stored weights outside folded depthwise blocks	114,650
Biases, scales, and requantization data	34,932
DyT lookup tables	576,666
Folded depthwise data	110,916
Alignment padding	33,468
Subtotal	870,632

G.5 Quantization and synthesis quality

We assess the quality cost of quantization through paired host-side FP32 and A16W8 inference on the same 128 texts, holding Flite 2.2 CMU frontend outputs and decoding settings fixed. On this evaluation workload, the host-side A16W8 PCM16 output is bitwise identical to the PCM16 output of the I^{2}S firmware. The host-side quality scores therefore also characterize the MCU-generated digital audio. We compute one UTMOS22-strong score per utterance using `tarepan/SpeechMOS:v1.2.0`, and obtain WER and CER with Whisper large-v3 through `faster-whisper` 1.2.1. WER and CER are averaged over utterances, rather than pooled over reference words or characters.

Table 14: Host-side quantization evaluation on 128 utterances. Values are mean $\pm$ standard deviation across utterances. Panel (a) isolates each neural component; panel (b) evaluates the complete TTS pipeline. Δ is A16W8 minus FP32, computed from the reported means before rounding them for display. UTMOS differences are score points, and WER/CER differences are percentage points (pp).

(a) Component-level differences Component	Metric	Result
EfficientSpeech	DTW-aligned mel L1	0.1323 ± 0.0249
	Normalized mel error (%)	2.790 ± 0.545
NanoVoco	Relative waveform RMSE (%)	13.68 ± 3.08

(b) End-to-end synthesis quality Metric	FP32	A16W8	Δ
UTMOS	3.299 ± 0.3286	3.227 ± 0.3137	-0.0720 points
WER (%)	9.504 ± 11.07	9.599 ± 11.08	$+0.095$ pp
CER (%)	4.478 ± 6.655	4.653 ± 7.160	$+0.175$ pp

Component-level effects. For EfficientSpeech, we measure the mean absolute mel difference after DTW alignment and normalize it by the mean absolute magnitude of the FP32 reference mel. For NanoVoco, both implementations receive identical FP32 mel conditioning, and relative waveform RMSE for each utterance is

$$E_{\mathrm{rel}} = \frac{\mathrm{RMSE}(x_{\mathrm{A16W8}}, x_{\mathrm{FP32}})}{\mathrm{RMS}(x_{\mathrm{FP32}})} = \sqrt{\frac{\sum_n (x_{\mathrm{A16W8},n} - x_{\mathrm{FP32},n})^2}{\sum_n x_{\mathrm{FP32},n}^2}}. \tag{14}$$

The normalized mel error is 2.79%, while relative waveform RMSE is 13.68% (Table 14(a)). These metrics characterize differences in their respective signal domains; waveform error is not itself a measure of perceptual quality.

End-to-end effects. With both neural components quantized, mean UTMOS decreases from 3.299 to 3.227, while mean utterance WER changes from 9.504% to 9.599% and CER from 4.478% to 4.653% (Table 14(b)). The corresponding changes are approximately -0.0720 UTMOS points, $+0.095$ WER percentage points, and $+0.175$ CER percentage points. These are descriptive effect sizes; the reported standard deviations characterize variation across utterances, not uncertainty in the paired change. We do not infer perceptual equivalence or statistical non-inferiority from them. The A16W8 UTMOS of 3.227 measures the complete text-to-waveform pipeline, including acoustic-model predictions; the 3.71 UTMOS in Table 1 instead measures the floating-point vocoder in isolation.