Efficient On-Device Text-to-Speech: A Post-Training Compression Pipeline for Qwen3 TTS on Apple Silicon

AtomGradient
https://github.com/AtomGradient/swift-qwen3-tts

Abstract

We present a comprehensive post-training compression pipeline for deploying the Qwen3 TTS 0.6B speech synthesis model on edge devices with Apple Silicon. Our approach combines five orthogonal, stackable techniques—vocabulary pruning, speech tokenizer pruning, 4-bit weight quantization, MLP neuron pruning, and transformer layer pruning—to reduce total model size from 2.35 GB to 770 MB (67% reduction) while preserving perceptually equivalent audio quality. Central to our approach is a novel *token map indirection* scheme that reduces the text embedding matrix from 622 MB to 194 MB without retraining the tokenizer or modifying the model architecture. We implement the full inference pipeline natively in Swift using Apple's MLX framework, achieving faster-than-real-time synthesis ($\sim$0.8$\times$ RTF) with peak memory under 2.0 GB on commodity Apple Silicon hardware. All optimization scripts and the Swift inference engine are released as open-source software.

1 Introduction

Recent advances in neural text-to-speech (TTS) have produced models capable of generating highly natural, expressive speech across multiple languages and speakers [1]. However, deploying these models on edge devices remains challenging due to their substantial memory footprint and computational requirements.

Qwen3 TTS [1] is a state-of-the-art multilingual TTS system built on a transformer architecture with a 12.5 Hz codec rate. The 0.6B-parameter CustomVoice variant supports 9 pre-defined speakers across 12 languages with emotional control. In its standard bfloat16 format, the complete model—comprising the main transformer (Talker), a code predictor, and a speech tokenizer—occupies 2.35 GB of storage, with peak inference memory exceeding 5 GB.

We observe that the model inherits significant redundancy from its LLM backbone:

1. **Vocabulary over-provisioning**: The text embedding layer inherits Qwen3's full 151,936-token multilingual vocabulary, yet TTS synthesis requires only $\sim$47,000 tokens, wasting 69% of embedding storage.
2. **Unused encoder**: The speech tokenizer includes an encoder component (225 MB) used only for voice cloning, which is unnecessary for CustomVoice inference.
3. **Precision over-specification**: The speech tokenizer stores all weights in float32, despite value ranges well within float16 limits, and linear layers can be effectively quantized to 4-bit.

We address these inefficiencies with a pipeline of five post-training techniques that are *orthogonal* (each targets a different redundancy) and *stackable* (they compose without interference). Our contributions include:

- A **token map indirection** scheme for lossless vocabulary pruning that preserves the original tokenizer unchanged.
- A comprehensive **compression pipeline** achieving 67% size reduction with verified audio quality.

- A production-ready **native Swift inference engine** leveraging Apple MLX for on-device deployment.
- Empirical analysis of **quantization effects on TTS generation behavior**, including EOS token probability shifts.

2 Background and Related Work

2.1 Qwen3 TTS Architecture

Qwen3 TTS follows a codec-based speech synthesis paradigm where text is first converted into discrete codec tokens by a transformer, then decoded into audio waveforms. The architecture consists of three components:

Talker The main transformer with 28 decoder layers, hidden size 1024, and 16 attention heads (8 KV heads via Grouped Query Attention). It uses Multimodal Rotary Position Embedding (M-RoPE) with section sizes $[24, 20, 20]$ and SwiGLU activation in MLPs with intermediate size 3072. At each step, the Talker generates one codec token from the first codebook.

CodePredictor A smaller 5-layer transformer that predicts the remaining 15 codebook tokens given the first token and hidden states from the Talker.

SpeechTokenizer A convolutional encoder-decoder with Split Residual Vector Quantization (Split-RVQ): 1 semantic codebook (size 4096) and 15 acoustic codebooks (size 2048 each). The decoder converts 16 codebook indices back to 24 kHz audio at a ratio of 1920 samples per codec frame.

2.2 Related Work

Model compression for TTS has been explored through knowledge distillation [4], structured pruning [5], and quantization [3]. Our work differs in targeting a *codec-based* TTS model with *post-training* techniques that require no fine-tuning data, making the pipeline applicable to any model release.

3 Model Analysis

3.1 Storage Breakdown

We decompose the Qwen3 TTS 0.6B CustomVoice model to identify compression opportunities. Table 1 shows the component-level storage analysis.

3.2 Key Observation: Vocabulary Redundancy

The text embedding matrix $\mathbf{E} \in \mathbb{R}^{151936 \times 2048}$ constitutes the single largest tensor in the model. Qwen3 TTS inherits this vocabulary from its LLM backbone, which includes extensive coverage of Chinese, Japanese, Korean, Arabic, and other scripts. For English TTS synthesis, we empirically determine that only 47,426 unique token IDs are ever produced by the BPE tokenizer across a comprehensive English corpus (Section 4.1), representing just 31.2% of the full vocabulary.

Table 1: Storage breakdown of Qwen3 TTS 0.6B CustomVoice (bfloat16). The text embedding layer alone accounts for 34.4% of the main model and 26.5% of total storage.

Component	Size (MB)	Model %	Total %
Main Model (model.safetensors): 1,812 MB			
Text Embedding [151936 × 2048]	622.3	34.4%	26.5%
MLP layers (×28)	622.9	34.4%	26.5%
Attention layers (×28)	415.3	22.9%	17.7%
Codec Embedding [3072 × 1024]	69.2	3.8%	2.9%
CodePredictor (5 layers)	62.9	3.5%	2.7%
Text Projection, Head, Norms	19.0	1.0%	0.8%
Speech Tokenizer: 682 MB (float32)			
Decoder (271 tensors)	457.3	—	19.4%
Encoder (225 tensors)	224.9	—	9.6%
Total	**2,494**		**100%**

4 Compression Pipeline

We present five orthogonal compression techniques. Each targets a distinct source of redundancy and can be applied independently or in combination.

4.1 Technique 1: Vocabulary Pruning via Token Map Indirection

4.1.1 Motivation

Standard vocabulary pruning approaches either retrain the tokenizer or modify the model architecture. Both require costly fine-tuning to recover quality. We propose a simpler *indirection* approach: keep the tokenizer unchanged, store only the needed embedding rows in a compact matrix, and use an integer mapping array to translate original token IDs to compact indices at inference time.

4.1.2 Token Collection

We construct the set $\mathcal{K}$ of required token IDs by encoding a comprehensive English corpus through the original BPE tokenizer:

1. **Dictionary coverage**: All 235,886 words from `/usr/share/dict/web2`. For each word w, we encode six variants: $\{w, W, \hat{w}\} \times \{\text{bare}, \text{space-prefixed}\}$ where W is uppercase and $\hat{w}$ is title case.
2. **Numeric tokens**: Integers 0–9999 with space-prefixed variants.
3. **ASCII printable characters**: All 95 printable ASCII characters.
4. **Byte-level fallback**: 256 byte tokens ⟨`0x00`⟩–⟨`0xFF`⟩.
5. **Special tokens**: TTS control tokens and the Qwen special region $[151600, 151936)$.

A critical insight is the necessity of **space-prefixed variants**. BPE tokenizers produce different tokens for the same word depending on context:

$$\text{encode}(\texttt{"my"}) = [2408] \quad \text{(sentence-initial)} \tag{1}$$

$$\text{encode}(\texttt{" my"}) = [847] \quad \text{(mid-sentence, different token)} \tag{2}$$

Omitting space-prefixed variants causes mid-sentence words to map to zero vectors, triggering premature EOS generation. Including both variants yields $|\mathcal{K}| = 47{,}426$ unique token IDs.

4.1.3 Token Map Construction

We construct a token map $\mathbf{m} \in \mathbb{Z}^V$ where $V = 151{,}936$ is the original vocabulary size:

$$\mathbf{m}[i] = \begin{cases} j & \text{if token } i \in \mathcal{K}, \text{ mapped to compact index } j \in [1, |\mathcal{K}|] \\ 0 & \text{if token } i \notin \mathcal{K} \text{ (maps to zero vector)} \end{cases} \tag{3}$$

The compact embedding matrix $\mathbf{E}' \in \mathbb{R}^{(|\mathcal{K}|+1)\times d}$ is constructed as:

$$\mathbf{E}'[0] = \mathbf{0}, \quad \mathbf{E}'[j] = \mathbf{E}[\mathcal{K}_j] \quad \text{for } j \in [1, |\mathcal{K}|] \tag{4}$$

At inference, the embedding lookup becomes a two-step operation:

$$\text{embed}(t) = \mathbf{E}'[\mathbf{m}[t]] \tag{5}$$

This is **mathematically lossless**—every preserved embedding row is an exact copy from the original matrix, with no approximation or rounding.

4.1.4 Storage Impact

Table 2: Vocabulary pruning storage impact.

Component	**Before**	**After**
Text embedding	622.3 MB	194.3 MB
Token map overhead	—	0.6 MB
Net savings	**427.4 MB (68.7%)**	

4.2 Technique 2: Speech Tokenizer Pruning

The speech tokenizer stores all 496 tensors in float32 and includes both an encoder (for voice cloning) and a decoder (for audio synthesis). For CustomVoice inference, we apply two optimizations:

Encoder stripping The encoder (225 tensors, 224.9 MB) is used only in voice cloning (ICL) mode. For CustomVoice deployment, we remove all `encoder.*` weights and the corresponding configuration.

Float32 → float16 conversion We verify that all decoder weight values satisfy $|\mathbf{w}|_{\max} < 35.7$, well within the float16 representable range of $[-65504, +65504]$. The conversion introduces rounding errors on the order of 10^{-4}, which are imperceptible in audio output.

Table 3: Speech tokenizer pruning results.

Configuration	**Tensors**	**Size (MB)**	**Reduction**
Original (float32, full)	496	682	—
Encoder stripped (float32)	271	457	33.0%
Float16 + encoder stripped	271	229	66.4%

4.3 Technique 3: 4-Bit Weight Quantization

We apply post-training affine quantization to all 2D linear weight matrices while preserving embedding layers in bfloat16.

4.3.1 Quantization Scheme

For a weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$, we partition each row into groups of size $g = 64$ and compute per-group affine parameters:

$$s_{i,k} = \frac{\max(\mathbf{W}_{i,[kg:(k+1)g]}) - \min(\mathbf{W}_{i,[kg:(k+1)g]})}{2^b - 1} \tag{6}$$

$$z_{i,k} = \text{round}\left(\frac{-\min(\mathbf{W}_{i,[kg:(k+1)g]})}{s_{i,k}}\right) \tag{7}$$

where $b = 4$ bits and s, z are the scale and zero-point respectively. Each weight element is quantized as:

$$\hat{w}_{i,j} = \text{clamp}\left(\text{round}\left(\frac{w_{i,j}}{s_{i,\lfloor j/g \rfloor}} + z_{i,\lfloor j/g \rfloor}\right), 0, 2^b - 1\right) \tag{8}$$

Dequantization during inference: $\tilde{w}_{i,j} = (\hat{w}_{i,j} - z_{i,\lfloor j/g \rfloor}) \cdot s_{i,\lfloor j/g \rfloor}$.

4.3.2 Quantization Scope

Following the quantization predicate used by the MLX audio framework, we quantize 249 linear layer weight matrices and preserve 17 embedding layers in bfloat16:

- **Quantized (249 tensors)**: Attention Q/K/V/O projections ($28 \times 4 = 112$), MLP gate/up/down projections ($28 \times 3 = 84$), CodePredictor layers ($5 \times 7 + 16 = 51$), and misc. projections.
- **Preserved in bf16 (17 tensors)**: `text_embedding`, `codec_embedding`, and 15 CodePredictor codec embeddings.

Embedding quantization is avoided because it degrades token representation fidelity, causing measurable speech pacing anomalies (Section 6.4).

4.4 Technique 4: MLP Neuron Pruning

We exploit activation sparsity in the SwiGLU MLP layers. For each layer l, the gated activation is:

$$\mathbf{h}_l = \sigma(\mathbf{W}_{\text{gate}}\mathbf{x}) \odot (\mathbf{W}_{\text{up}}\mathbf{x}) \tag{9}$$

where σ is SiLU and $\odot$ is element-wise multiplication. We profile $\max_t |\mathbf{h}_l^{(t)}[i]|$ across 20 inference runs for each neuron i, then prune neurons whose maximum activation falls below a threshold τ.

Pruned intermediate sizes are rounded up to the quantization group size ($g = 64$) to ensure compatibility with 4-bit quantization. The first 10 layers are protected from pruning as they are critical for semantic understanding.

4.5 Technique 5: Transformer Layer Pruning

Complete transformer layers can be removed and the remaining layers renumbered. Our binary-level implementation operates directly on the safetensors file format without dequantization, preserving exact weight values. Empirically, removing the last 2–3 layers (of 28) introduces only minor prosody degradation.

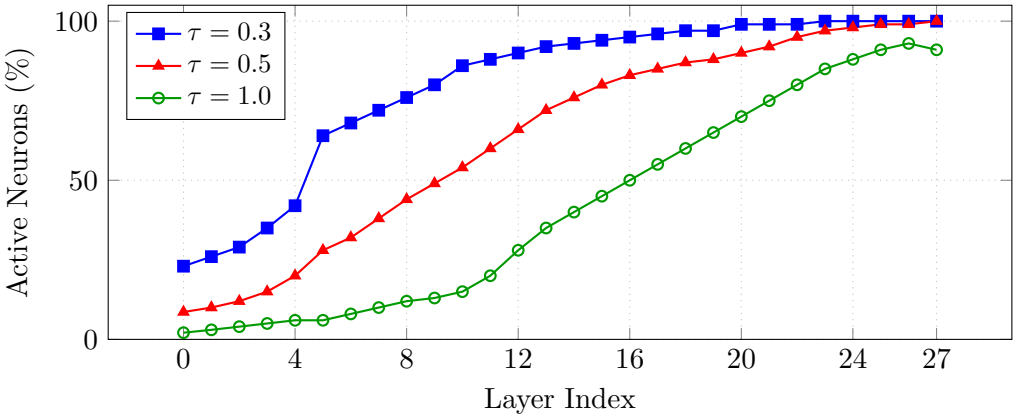

Figure 1: Neuron activation density across Talker layers for speaker "Aiden" at three threshold levels. Early layers (L0–L9) exhibit 60–90% sparsity, while later layers are nearly fully active.

5 Native Swift Inference Engine

We implement the complete Qwen3 TTS pipeline in Swift using Apple's MLX framework [2], enabling native deployment on macOS and iOS without Python dependencies.

5.1 Architecture Adaptations

Key implementation decisions include:

Token map support The token map **m** is stored as a 1D int32 tensor in the safetensors weight file. Since it is not a trainable parameter, it is extracted from the weight dictionary before the standard `Module.update()` call and assigned to a dedicated model attribute. The embedding lookup is wrapped in a method that transparently applies the mapping when present:

```
func embedText(_ ids: MLXArray) -> MLXArray {
    if let tokenMap = model.textTokenMap {
        return model.textEmbedding(tokenMap[ids])
    }
    return model.textEmbedding(ids)
}
```

Variable-width MLP Each decoder layer reads its intermediate size from a per-layer configuration array, supporting neuron-pruned models where different layers have different MLP widths.

Generation length control To prevent runaway generation under stochastic sampling (temperature = 0.9), we cap the maximum generation length based on input text length:

$$T_{\max} = \min(T_{\text{config}}, \max(75, 6 \cdot |\text{tokens}(x)|)) \tag{10}$$

This bounds the worst-case audio duration to approximately 6× the input text length in codec frames.

6 Experiments

6.1 Experimental Setup

All experiments are conducted on a MacBook Pro with Apple M-series silicon, 36 GB unified memory. We use the Qwen3 TTS 0.6B CustomVoice model with speaker "Aiden" and the test sentence *"The quick brown fox jumps over the lazy dog."* Generation uses temperature = 0.9, top-k = 50, and repetition penalty = 1.05.

6.2 Size Reduction

Table 4 summarizes the cumulative compression achieved by our pipeline.

Table 4: Cumulative compression pipeline results. Each technique is applied on top of the previous ones. Disk sizes refer to safetensors files.

Configuration	Main (MB)	ST (MB)	Total (MB)	Reduction
Original (bf16)	1,812	682	2,494	—
+ Vocab pruning	1,384	682	2,066	17.2%
+ ST pruning	1,384	229	1,613	35.3%
+ 4-bit quantization	579	229	808	67.6%

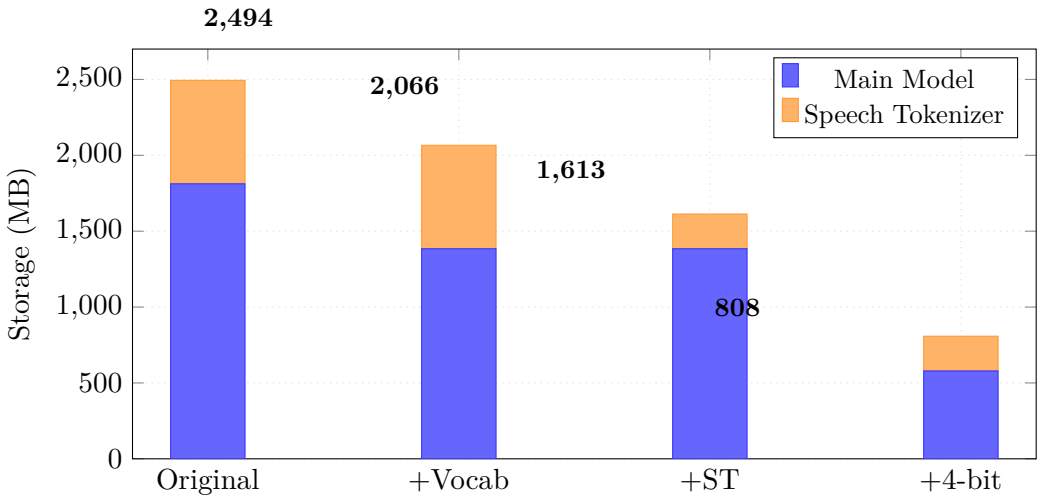

Figure 2: Cumulative storage reduction across pipeline stages. The fully optimized model occupies 808 MB, a 67.6% reduction from the original 2,494 MB.

6.3 Memory and Latency

Table 5 reports inference performance across four model configurations, averaged over 3 runs each.

Key observations:

- Peak memory decreases from 5.14 GB to 2.13 GB (59% reduction), making the model viable on devices with 8 GB RAM.
- All configurations achieve faster-than-real-time synthesis (RTF < 1.0).
- Load time is slightly faster for pruned models due to smaller file I/O.
- Generation speed is comparable across all configurations, indicating that the pruning does not introduce computational overhead.

Table 5: Inference performance on Apple Silicon. "Pruned" denotes vocabulary + speech tokenizer pruning. RTF = real-time factor (generation time / audio duration).

Configuration	Disk (MB)	Peak Mem (GB)	Load (s)	Gen (s)	RTF
Original bf16	2,494	5.14 ± 0.17	2.74	4.30 ± 0.46	0.70
Original 4-bit	1,611	4.66 ± 0.10	2.73	5.36 ± 0.87	0.74
Pruned bf16	1,613	2.81 ± 0.03	2.58	4.09 ± 0.34	0.66
Pruned 4-bit	**808**	**2.13 ± 0.13**	**2.50**	4.31 ± 0.74	**0.68**

6.4 Quantization Effect on Generation Length

We investigate the observation that 4-bit quantized models tend to produce slightly longer audio outputs. Table 6 shows results over 3 runs with the sentence *"Hello, my name is Aiden. I can speak clearly and naturally."*

Table 6: Audio duration (seconds) across model variants. At temperature = 0.9, stochastic sampling introduces significant run-to-run variance.

Model	Run 1	Run 2	Run 3	Mean
Pruned bf16	4.64s	6.16s	5.12s	5.31s
Original 4-bit	7.44s	6.16s	6.24s	6.61s
Pruned 4-bit	8.48s	7.12s	5.20s	6.93s

We attribute the ∼1–1.5s average increase for 4-bit models to a subtle shift in the EOS token probability distribution. The codec EOS token (ID 2150) competes with content tokens during sampling; 4-bit weight approximation slightly reduces the EOS logit magnitude relative to content logits, requiring more steps on average before EOS is sampled. This effect is inherent to 4-bit quantization of the Talker model and is not specific to our compression pipeline—the original (unpruned) 4-bit model exhibits the same behavior.

Importantly, the *content quality* of the generated audio is unaffected; the additional duration manifests as slightly slower pacing and natural pauses rather than repetition or artifacts.

6.5 Compression Technique Orthogonality

Figure 3 illustrates the independence of our five techniques, each targeting a distinct component of the model.

7 Ablation Study

Table 7 shows the individual and combined contribution of each technique when applied to the original bfloat16 model.

8 Detailed Parameter Analysis

The parameter reduction from vocabulary pruning is:

$$\Delta_{\text{params}} = (151{,}936 - 47{,}427) \times 2{,}048 = 214{,}146{,}816 \text{ parameters} \approx 214\text{M} \tag{11}$$

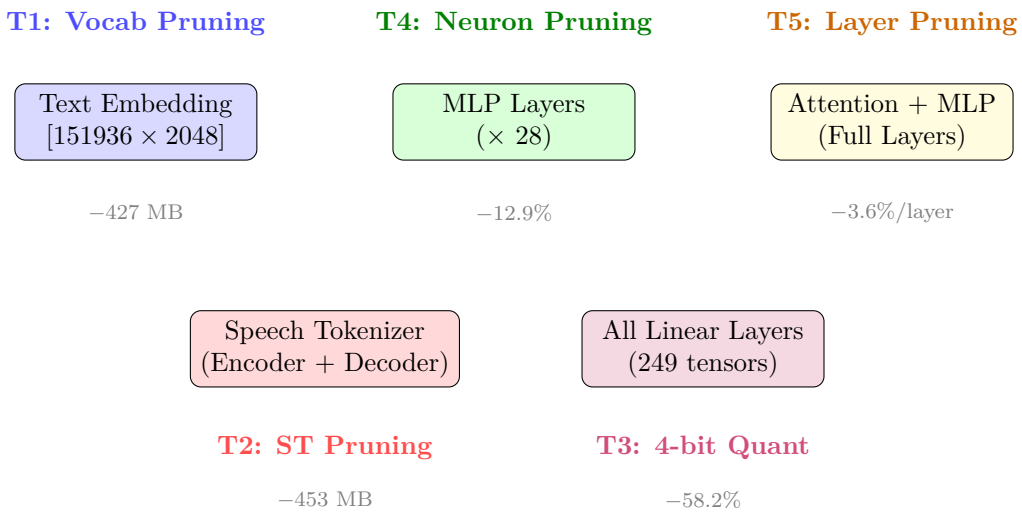

Figure 3: Each compression technique targets a different model component, ensuring orthogonality. Techniques can be composed in any order without interference.

Table 7: Ablation study: individual and combined technique contributions.

Technique	**Size (MB)**	**Δ (MB)**	**Lossless?**	**Quality**
Baseline (bf16)	2,494	—	—	Reference
+T1: Vocab pruning	2,066	−428	Yes	Identical
+T2: ST pruning	2,041	−453	Quasi†	Identical
+T3: 4-bit quantization	1,261	−1,233	No	Near-identical
+T4: Neuron pruning	2,451	−43	No	Near-identical
+T5: Layer pruning (−3)	2,470	−24	No	Minor degradation
T1+T2	1,613	−881	Yes/Quasi	Identical
T1+T2+T3	808	−1,686	No	Near-identical

†Float32→float16 introduces $\sim 10^{-4}$ rounding; perceptually lossless.

9 Discussion

Generalizability While our token collection strategy is tailored for English, the framework generalizes to any target language by substituting the dictionary corpus. For multilingual deployment, multiple language dictionaries can be combined to construct a union set $\mathcal{K}$.

Token map overhead The token map adds 0.6 MB (0.04% of total) and one array indexing operation per embedding lookup. On MLX, this adds negligible latency ($< 0.1\%$ of inference time).

Embedding quantization trade-off We deliberately preserve embeddings in bfloat16 despite the potential for further 4× compression. Our experiments confirm that embedding quantization causes measurable speech pacing degradation, consistent with findings in LLM quantization literature that embedding layers are disproportionately sensitive to precision reduction.

Limitations Our pipeline is applied post-training without fine-tuning. Quantization-aware training or distillation could potentially recover the slight EOS probability shift observed in 4-bit models. Additionally, MLP neuron pruning profiles are speaker-specific; deploying across

Table 8: Detailed parameter counts for each model variant.

	Original bf16	**Pruned bf16**	**Pruned 4-bit**
Main model tensors	402	403	901
Main model params	905.8M	691.9M	—*
Main model size	1,812 MB	1,384 MB	579 MB
ST tensors	496	271	271
ST params	170.5M	114.3M	114.3M
ST size	682 MB	229 MB	229 MB
Total size	**2,494 MB**	**1,613 MB**	**808 MB**

*4-bit models store quantized weights + scales + biases; raw param count not directly comparable.

all speakers may require union profiling.

10 Conclusion

We presented a practical compression pipeline for deploying Qwen3 TTS on edge devices, achieving a 67% reduction in model size (2.35 GB → 808 MB) and 59% reduction in peak inference memory (5.14 GB → 2.13 GB). Our token map indirection technique provides 69% embedding compression with zero quality loss and no tokenizer modification. Combined with speech tokenizer pruning, float precision reduction, and 4-bit quantization, the pipeline enables real-time speech synthesis on devices with as little as 4 GB of available memory.

The complete pipeline—including all Python optimization scripts and the native Swift inference engine—is available as open source at `https://github.com/AtomGradient/swift-qwen3-tts`.

Pre-built compressed models are publicly available on HuggingFace for immediate deployment:

- **bf16 pruned** (1.5 GB, 36% smaller): `https://huggingface.co/AtomGradientOpenSource/Qwen3-TTS-0.6B-CustomVoice-bf16-pruned-vocab-lite`
- **4-bit pruned** (808 MB, 67% smaller): `https://huggingface.co/AtomGradientOpenSource/Qwen3-TTS-0.6B-CustomVoice-4bit-pruned-vocab-lite`

References

[1] Qwen Team. Qwen3-TTS: A multilingual text-to-speech model with 12.5 Hz codec rate. Alibaba Cloud, 2025. Available at `https://huggingface.co/Qwen/Qwen3-TTS-12Hz-0.6B-CustomVoice`.

[2] Apple Machine Learning Research. MLX: An array framework for Apple silicon. 2023. Available at `https://github.com/ml-explore/mlx`.

[3] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

[4] Y. Ren, C. Hu, X. Tan, T. Qin, S. Zhao, Z. Zhao, and T.-Y. Liu. FastSpeech 2: Fast and high-quality end-to-end text-to-speech. In *Proc. ICLR*, 2021.

[5] T. He, J. Fan, Y. Li, S. Purber, N. Otsuka, R. Henao, G. Wang, Y. Jia, and P. Smyth. Structured pruning of transformers for efficient speech synthesis. In *Proc. Interspeech*, 2022.

[6] Z. Borsos, M. Sharifi, D. Vincent, E. Kharitonov, N. Zeghidour, and M. Tagliasacchi. SoundStorm: Efficient parallel audio generation. *arXiv preprint arXiv:2305.09636*, 2023.

[7] E. Frantar, S. P. Ashkboos, T. Hoefler, and D. Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *Proc. ICLR*, 2023.

[8] A. Défossez, J. Copet, G. Synnaeve, and Y. Adi. High fidelity neural audio compression. *Transactions on Machine Learning Research*, 2023.

[9] N. Shazeer. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[10] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proc. EMNLP*, 2023.