TangoFlux Endless: Real-time Text-to-Audio Generation on Apple Silicon via CoreML-accelerated Flow Matching

Yoichi Ochiai*

February 2026

Abstract

We present **TangoFlux Endless**, a system for real-time, continuous text-to-audio generation on Apple Silicon. By converting the FluxTransformer2DModel of TangoFlux [1] to Apple's CoreML format for hybrid Apple Neural Engine (ANE) and GPU execution, we achieve a Real-Time Factor (RTF) of 0.28× for 18-second audio clips at 44.1 kHz—a 1.59× speedup over the baseline MPS (Metal Performance Shaders) backend. Our contributions include: (1) a novel CoreML-compatible refactoring of Rotary Position Embedding (RoPE) [2] that eliminates rank-6 tensor operations and ellipsis `einsum` notation; (2) a subprocess-isolated, GIL-free architecture for glitch-free audio playback during concurrent generation; (3) an N-layer staggered crossfade mechanism with $\sin^2 / \cos^2$ envelopes for seamless infinite soundscapes; and (4) a negative result demonstrating that MPS float16 inference produces audible artifacts in flow-matching models due to dtype mismatch in hardcoded intermediate tensors. We release all code, CoreML conversion scripts, and benchmark data as open-source software.

Keywords: text-to-audio generation, CoreML, Apple Neural Engine, flow matching, real-time synthesis, TangoFlux

1 Introduction

Text-to-audio (TTA) generation has advanced rapidly with diffusion [3, 4] and flow-matching [5, 6] models, producing high-fidelity environmental sounds, music, and speech from natural language descriptions. State-of-the-art models such as TangoFlux [1], AudioLDM 2 [7], and Stable Audio [8] achieve impressive quality on GPU servers, yet deploying these models on consumer edge devices remains challenging.

Interactive installations, live performances, and ambient soundscape applications demand *real-time* generation—where the system produces audio faster than it is consumed—on portable, silent hardware without GPU server access. Apple Silicon devices (M1–M4 series) integrate a CPU, GPU, and dedicated Neural Engine (ANE) capable of up to 38 TOPS [9], presenting an attractive target for on-device inference. However, the TTA model ecosystem has focused exclusively on NVIDIA GPU optimization, leaving Apple Silicon deployment unexplored.

In this work, we bridge this gap by deploying TangoFlux—a 515M-parameter flow-matching TTA model based on the Flux (MMDiT) architecture [10]—on Apple Silicon via CoreML. Our system, **TangoFlux Endless**, generates continuous soundscapes from text prompts, achieving an RTF of 0.28× (3.6× faster than real-time) on an Apple M2 Ultra, enabling true real-time, infinite audio playback.

The key technical challenge lies in converting the FluxTransformer2DModel to CoreML. Two operations in the standard Flux RoPE implementation are incompatible with CoreML's

*Digital Nature Group, University of Tsukuba / Pixie Dust Technologies, Inc. Contact: `ochiai@digitalnature.slis.tsukuba.ac.jp`

tracing requirements: (1) ellipsis-notation `einsum` (`"...n,d->...nd"`) and (2) rank-6 tensors exceeding CoreML's rank-5 limit. We propose a refactored RoPE using separated $(\cos, \sin)$ representations with direct rotation, verified to produce numerically identical outputs (max trace difference $= 0.00$).

We further document a **negative result**: attempting float16 inference on the MPS backend produces audible electrical noise artifacts. Root cause analysis reveals that TangoFlux's `inference_flow()` creates hardcoded float32 intermediate tensors that mix with float16 model weights, causing dtype mismatch that accumulates over the 25-step denoising loop. CoreML's internal float16 (via `compute_precision=FLOAT16`) avoids this problem because precision management occurs within the CoreML runtime.

The system is deployed as part of *Keisanki no Shizen* (Computational Nature), an art installation exploring the boundary between computational and natural sound environments.

2 Related Work

2.1 Text-to-Audio Generation

The TTA landscape has evolved from autoregressive token prediction to diffusion and flow-matching paradigms. AudioGen [11] and MusicGen [12] use autoregressive transformers over discrete audio tokens from neural codecs like EnCodec [13], achieving high quality but with sequential computation overhead. AudioLDM [14] pioneered latent diffusion for TTA using CLAP [15] contrastive embeddings, operating in compressed latent space. AudioLDM 2 [7] extended this to a unified framework for speech, music, and sound effects. Make-An-Audio [16] addressed data scarcity through pseudo prompt enhancement. Tango [17] demonstrated competitive quality with 63× less training data using instruction-tuned Flan-T5 conditioning.

More recent approaches leverage transformer architectures and flow matching. EzAudio [18] uses an optimized DiT for 1D waveform VAE latent representations. GenAu [19] scales transformer-based audio generation to 1.25B parameters. Stable Audio [8] employs a 1057M-parameter DiT for variable-length 44.1 kHz stereo audio. TangoFlux [1], our base model, is a 515M-parameter flow-matching model using the Flux (MMDiT) architecture [10] with CLAP-Ranked Preference Optimization (CRPO), generating up to 30 s audio at 44.1 kHz.

2.2 Efficient Inference for Diffusion Models

Reducing the number of function evaluations (NFE) is a primary acceleration strategy. DDIM [20] introduced deterministic sampling for 10–50× speedup. Consistency Models [21] map noise to data in a single step. Latent Consistency Models (LCM) [22] distill LDMs for 1–4 step inference. Progressive distillation [23] iteratively halves the step count. For audio specifically, SoundCTM [24] achieves flexible 1-step to multi-step generation via consistency trajectory models. FlashAudio [25] uses rectified flows with Bifocal Samplers for one-step generation at 400× real-time on RTX 4090.

Our approach is orthogonal: rather than reducing NFE, we accelerate each function evaluation by deploying on specialized hardware (ANE/GPU) via CoreML, while maintaining the full 25-step schedule. These two strategies are complementary and could be combined in future work.

2.3 Edge Deployment of Generative Models

Apple's work on deploying Stable Diffusion via CoreML [26] demonstrated the feasibility of running U-Net-based image diffusion on ANE, achieving significant speedups over CPU-only inference. The ANE Transformers technical report [27] provides guidelines for ANE-compatible

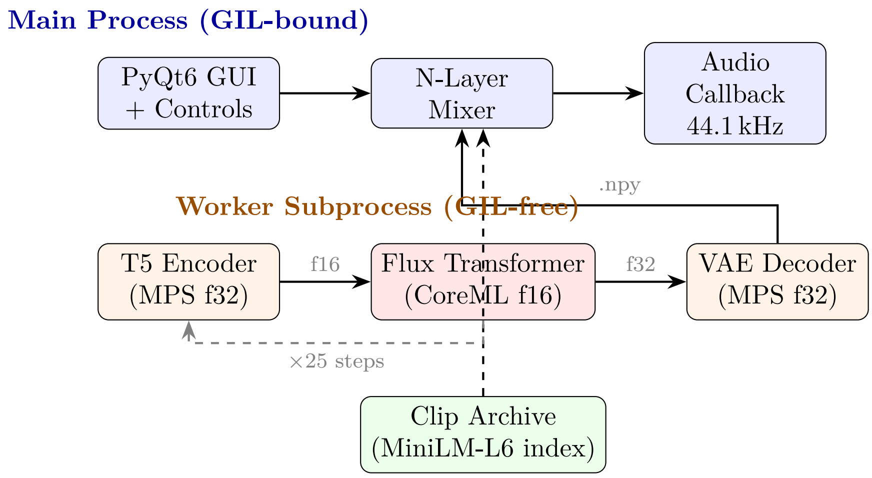

Figure 1: System architecture of TangoFlux Endless. The inference pipeline runs in a separate subprocess to avoid GIL contention with the audio callback. The FluxTransformer2DModel is executed via CoreML (ANE/GPU, float16), while T5 encoding and VAE decoding remain in PyTorch (MPS, float32). Generated clips are exchanged as `.npy` files via a temporary directory.

operations. WWDC 2024 [9] introduced mixed-bit palettization and W8A8 quantization on M4-class chips.

Our work extends CoreML deployment from U-Net image diffusion to Flux-architecture audio diffusion—a more recent transformer design requiring distinct tensor shape handling and RoPE compatibility modifications.

2.4 Architectural Foundations

TangoFlux builds on several architectural foundations: the transformer [28], DiT (Diffusion Transformer) [29], the MMDiT dual-stream design of Flux/SD3 [10], RoPE [2] for positional encoding, and flow matching [5] with rectified flow [6] for training and inference.

3 System Architecture

TangoFlux Endless consists of three main components: (1) a hybrid CoreML/PyTorch inference pipeline, (2) a subprocess-isolated generator worker, and (3) an N-layer crossfade playback engine. Figure 1 shows the overall system design.

3.1 Hybrid Inference Pipeline

The TangoFlux inference pipeline consists of three stages:

1. **Text encoding**: Flan-T5-Large encodes the text prompt into `encoder_hidden_states` and `pooled_projection` tensors (PyTorch, MPS, float32).

2. **Flow-matching denoising**: The FluxTransformer2DModel iteratively denoises latent representations over 25 Euler steps using the FlowMatchEulerDiscreteScheduler (CoreML, ANE/GPU, internally float16).

3. **Waveform decoding**: AutoencoderOobleck decodes the final latents to a 44.1 kHz waveform (PyTorch, MPS, float32).

The hybrid design exploits CoreML's ANE/GPU scheduling for the computationally dominant transformer while keeping the T5 encoder and VAE decoder in PyTorch MPS, which avoids the overhead of converting models with variable-length text inputs.

3.2 Subprocess Isolation

Python's Global Interpreter Lock (GIL) causes CPU-bound PyTorch inference and I/O-bound audio callbacks to contend for execution time. In a threaded model, this produces audible pop/click artifacts at callback deadlines. We eliminate GIL contention by running the generator in a separate process via `subprocess.Popen`, communicating through `.npy` files in a shared temporary directory.

3.3 N-Layer Staggered Crossfade

Multiple audio clips ($N = 3$ by default) play simultaneously with staggered phase offsets. Each clip uses $\sin^2 / \cos^2$ fade envelopes:

$$\mathrm{env}(t) = \begin{cases} \sin^2\left(\frac{\pi t}{2T_f}\right) & t < T_f \text{ (fade-in)} \\ 1 & T_f \le t \le T - T_f \text{ (sustain)} \\ \cos^2\left(\frac{\pi(t-T+T_f)}{2T_f}\right) & t > T - T_f \text{ (fade-out)} \end{cases} \tag{1}$$

where T is the clip duration and T_f is the fade duration. The output is normalized by $1/N$ to prevent clipping:

$$y(t) = \frac{1}{N}\sum_{i=1}^{N} x_i(t) \cdot \mathrm{env}_i(t - \phi_i) \tag{2}$$

where ϕ_i is the phase offset for layer i.

3.4 Archive Fallback

When generation cannot keep pace with playback (buffer underrun), the system retrieves previously generated clips using semantic similarity. Current prompt embeddings (all-MiniLM-L6-v2, 384-dim [30]) are compared against an archive index via cosine similarity, and a random selection from the top-3 candidates ensures variety.

4 CoreML Conversion of Flux RoPE

The primary technical contribution is the CoreML-compatible conversion of TangoFlux's Flux-Transformer2DModel. The standard Flux implementation [10] contains two operations incompatible with CoreML's `coremltools.convert()` tracing:

Problem 1: Ellipsis `einsum`. The RoPE frequency computation uses:

```
out = torch.einsum("...n,d->...nd", pos, omega)
```

CoreML's tracing infrastructure (`coremltools` 8.x) cannot handle ellipsis notation in `einsum` operations.

Solution: Replace with equivalent basic tensor operations:

```
out = pos.unsqueeze(-1) * omega.unsqueeze(0).unsqueeze(0)
```

Problem 2: Rank-6 tensors. The original RoPE applies rotation via a 2×2 matrix representation:

```
stacked = torch.stack([cos, -sin, sin, cos], dim=-1)
out = rearrange(out, "b n d (i j) -> b n d i j", i=2, j=2)
return emb.unsqueeze(1)  # rank 6!
```

CoreML limits tensors to rank 5.

Solution: Separate the rotation into $(\cos, \sin)$ components and apply directly:

$$\texttt{rope}(\mathbf{p}, d, \theta) \to (\cos(\mathbf{p} \cdot \boldsymbol{\omega}),\ \sin(\mathbf{p} \cdot \boldsymbol{\omega})) \quad (3)$$

$$\texttt{apply\_rope}(\mathbf{x}, \mathbf{c}, \mathbf{s}) : \mathbf{x}_{\text{out}} = [\mathbf{c} \cdot x_r - \mathbf{s} \cdot x_i,\ \mathbf{s} \cdot x_r + \mathbf{c} \cdot x_i] \quad (4)$$

where x_r, x_i denote the real and imaginary components of the paired RoPE dimensions. This representation maintains a maximum tensor rank of 5.

Verification. We verify numerical equivalence by comparing traced outputs before and after refactoring on identical inputs. The maximum absolute difference is 0.00, confirming that the refactoring is lossless.

5 Experiments

All experiments are conducted on an Apple Mac Studio with M2 Ultra (24-core CPU, 76-core GPU, 32-core Neural Engine, 192 GB unified memory) running macOS 15.3, PyTorch 2.4.0, and `coremltools` 8.1.

5.1 Latency: CoreML vs. MPS

Table 1 reports generation latency for 18-second audio clips at 25 denoising steps, measured over 10 runs after 2 warmup iterations.

Table 1: Generation latency for 18 s audio at 25 steps (10 runs, 2 warmup discarded).

Backend	Mean (s)	Std (s)	Min (s)	Max (s)	P95 (s)	RTF
MPS (f32)	8.049	0.273	7.652	8.350	8.328	0.447×
CoreML (f16)	**5.059**	0.397	4.749	5.765	5.699	**0.281×**
Speedup (MPS ÷ CoreML)						**1.59×**

CoreML achieves a mean RTF of 0.281×, meaning generation completes in 28.1% of the audio's real-time duration. The 1.59× speedup over MPS comes from ANE/GPU hybrid execution of the transformer's matrix operations, which dominate the 25-step denoising loop.

5.2 Step Count Sweep

Table 2 shows generation time and spectral characteristics as a function of denoising steps, using the CoreML backend.

Table 2: Step count sweep (CoreML backend, 5 runs each). Spectral metrics computed on generated audio for the prompt "A gentle rain falling on leaves with distant thunder."

Steps	Time (s)	RTF	RMS	Centroid (Hz)	Flatness	Silence
10	2.827	0.157×	0.033	1467	0.054	0.000
15	3.552	0.197×	0.057	897	0.027	0.000
20	4.252	0.236×	0.060	623	0.015	0.000
25	**5.003**	**0.278×**	0.057	1412	0.038	0.000
30	5.900	0.328×	0.048	2562	0.026	0.000
50	8.898	0.494×	0.058	775	0.189	0.167

Generation time scales linearly with step count ($\approx$ 0.18 s/step), consistent with the flow-matching Euler scheduler performing one transformer forward pass per step. The default 25 steps offers the best trade-off: sufficient quality (low flatness, zero silence ratio) while maintaining RTF $< 0.3\times$. At 50 steps, a silence ratio of 16.7% emerges along with elevated spectral flatness (0.189), suggesting over-denoising.

5.3 Prompt Variance

Table 3 evaluates generation consistency across five semantically distinct prompts, with 3 runs each using the CoreML backend at 25 steps.

Table 3: Prompt variance analysis (CoreML, 25 steps, 3 runs per prompt).

Prompt	Time (s)	RTF	RMS	Centroid (Hz)	Silence
Rain + thunder	4.960	0.276×	0.050	1183	0.000
Ocean waves + seagulls	5.205	0.289×	0.084	670	0.034
Busy café	5.004	0.278×	0.079	627	0.000
Wind + birdsong	4.999	0.278×	0.044	347	0.061
Campfire	5.118	0.284×	0.022	869	0.198
Mean ± Std	5.06 ± 0.10	0.281×	—	—	—

Latency is prompt-independent: the standard deviation across prompts is only 0.10 s (2.0% of mean), confirming that the fixed-shape CoreML model produces deterministic computation time regardless of semantic content. In contrast, **spectral characteristics vary significantly by prompt**: RMS ranges from 0.022 (campfire) to 0.084 (ocean waves), and spectral centroid spans 347–1183 Hz, reflecting the model's ability to produce acoustically diverse outputs.

5.4 Spectral Analysis

The spectral analysis reveals content-appropriate acoustic signatures:

- **Ocean waves** (highest RMS 0.084, centroid 670 Hz): broadband energy distribution consistent with surf sounds.
- **Wind + birdsong** (lowest centroid 347 Hz): low-frequency dominance characteristic of wind with narrow tonal components.
- **Campfire** (highest silence ratio 0.198, lowest RMS 0.022): intermittent crackling pattern with quiet intervals, acoustically appropriate for campfire sounds.
- **Rain** (highest centroid 1183 Hz): high-frequency energy consistent with rain noise spectrum.

6 Negative Results

We document two negative results that inform future TTA deployment efforts.

6.1 Float16 on MPS Produces Audible Artifacts

Attempting float16 inference on Apple's MPS backend produces audible electrical noise artifacts. The root cause is TangoFlux's `inference_flow()` method, which creates hardcoded float32 intermediate tensors:

```
torch.tensor(float("nan"))   # float32
torch.randn(...)              # float32
torch.zeros(...)              # float32
```

These mix with float16 model weights during the 25-step denoising loop, causing dtype promotion mismatches. The iterative nature of flow matching amplifies small precision errors at each step.

CoreML's internal float16 (via `compute_precision=ct.precision.FLOAT16`) does not exhibit this problem because the CoreML runtime manages precision conversions internally, without Python-level tensor dtype conflicts.

6.2 Disabling Classifier-Free Guidance

Setting `guidance_scale=0` (no CFG) reduces per-step computation but severely degrades text adherence. Generated audio becomes generic ambient noise unrelated to the prompt. Combined with float16 on MPS, CFG removal produces compound quality degradation.

7 Discussion

Comparison with GPU-side SOTA. FlashAudio [25] achieves 400× real-time on RTX 4090 with one-step generation. Our system achieves 3.6× real-time on Apple Silicon—two orders of magnitude slower in absolute terms, but operating on consumer hardware without discrete GPU. The comparison highlights that on-device deployment on Apple Silicon is viable for real-time applications (RTF < 1.0) but cannot match datacenter GPU throughput.

Future acceleration paths. Several techniques could further reduce our RTF: (1) Latent Consistency Model distillation [22] to reduce the 25-step loop to 2–4 steps, potentially achieving RTF $< 0.1\times$; (2) mixed-bit palettization [9] (4-bit weights) to reduce memory bandwidth and enable more ANE parallelism; (3) `torch.compile` with Metal backend, currently immature but promising for future PyTorch releases.

Limitations. Our system currently generates fixed-duration 18 s clips and achieves continuous playback through crossfade rather than true streaming generation. The ∼5 s generation latency creates a startup delay before first audio output. The archive fallback adds variety but cannot produce truly novel content when the generator falls behind.

8 Conclusion

We have demonstrated that real-time text-to-audio generation on Apple Silicon is achievable through a combination of CoreML conversion, subprocess isolation, and staggered crossfade playback. Our CoreML-compatible RoPE refactoring enables deployment of Flux-architecture transformers on the Apple Neural Engine, achieving a 1.59× speedup over MPS with an RTF of 0.281×. The negative result on float16 MPS inference provides guidance for future TTA deployment efforts. We release all source code, patches, and benchmark data to support reproducible research at the intersection of generative audio and edge deployment.

Acknowledgments

This work is part of *Keisanki no Shizen* (Computational Nature), an installation at the National Museum of Emerging Science and Innovation (Miraikan), Tokyo. The author thanks the Digital Nature Group at the University of Tsukuba and Pixie Dust Technologies for their support.

References

[1] Chia-Yu Hung, Navonil Majumder, Zhifeng Kong, Ambuj Mehrish, Amir Zadeh, Chuan Li, Rafael Valle, Bryan Catanzaro, and Soujanya Poria. TangoFlux: Super fast and faithful text to audio generation with flow matching and CLAP-ranked preference optimization. *arXiv preprint arXiv:2412.21037*, 2024. Accepted at ICLR 2026.

[2] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 2024.

[3] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Proc. NeurIPS*, 2020.

[4] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proc. CVPR*, 2022.

[5] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *Proc. ICLR*, 2023.

[6] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *Proc. ICLR (Spotlight)*, 2023.

[7] Haohe Liu, Yi Yuan, Xubo Liu, Xinhao Mei, Qiuqiang Kong, Qiao Tian, Yuping Wang, Mark Plumbley, and Wenwu Wang. AudioLDM 2: Learning holistic audio generation with self-supervised pretraining. *arXiv preprint arXiv:2308.05734*, 2023.

[8] Zach Evans, Julian D. Parker, CJ Carr, Zack Zukowski, Josiah Taylor, and Jordi Pons. Stable audio open. *arXiv preprint arXiv:2407.14358*, 2024.

[9] Apple. Deploy machine learning and AI models on-device with Core ML — WWDC 2024. Session 10161, 2024.

[10] Patrick Esser, Sumith Kulal, Andreas Blattmann, et al. Scaling rectified flow transformers for high-resolution image synthesis. *Proc. ICML*, 2024.

[11] Felix Kreuk, Gabriel Synnaeve, Adam Polyak, Uriel Singer, Alexandre Defossez, Jade Copet, Devi Parikh, Yaniv Taigman, and Yossi Adi. AudioGen: Textually guided audio generation. In *Proc. ICLR*, 2023.

[12] Jade Copet, Felix Kreuk, Itai Gat, et al. Simple and controllable music generation. In *Proc. NeurIPS*, 2023.

[13] Alexandre Défossez, Jade Copet, Gabriel Synnaeve, and Yossi Adi. High fidelity neural audio compression. *arXiv preprint arXiv:2210.13438*, 2022.

[14] Haohe Liu, Zehua Chen, Yi Yuan, Xinhao Mei, Xubo Liu, Danilo Mandic, Wenwu Wang, and Mark D. Plumbley. AudioLDM: Text-to-audio generation with latent diffusion models. In *Proc. ICML*, 2023.

[15] Benjamin Elizalde, Soham Deshmukh, Mahmoud Al Ismail, and Huaming Wang. CLAP: Learning audio concepts from natural language supervision. In *Proc. ICASSP*, 2023.

[16] Rongjie Huang, Jiawei Huang, Dongchao Yang, Yi Ren, et al. Make-An-Audio: Text-to-audio generation with prompt-enhanced diffusion models. *arXiv preprint arXiv:2301.12661*, 2023.

[17] Deepanway Ghosal, Navonil Majumder, Ambuj Mehrish, and Soujanya Poria. Text-to-audio generation using instruction-tuned LLM and latent diffusion model. *arXiv preprint arXiv:2304.13731*, 2023.

[18] Jiarui Hai, Yong Yang, Jiatong Zhang, Jionghao Lu, and Changli Wang. EzAudio: Enhancing text-to-audio generation with efficient diffusion transformer. *arXiv preprint arXiv:2409.10819*, 2024.

[19] Mohamed Haji-Ali, Apoorv Hedge, Fabian-Robert Stöter, et al. Taming data and transformers for audio generation. *arXiv preprint arXiv:2406.19388*, 2024.

[20] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *Proc. ICLR*, 2021.

[21] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *Proc. ICML*, 2023.

[22] Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference. *arXiv preprint arXiv:2310.04378*, 2023.

[23] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *Proc. ICLR*, 2022.

[24] Koichi Saito, Dongjun Kim, Takashi Shibuya, Chieh-Hsin Lai, Zhi Zhong, Yuhta Takida, and Yuki Mitsufuji. SoundCTM: Unifying score-based and consistency models for text-to-sound generation. In *Proc. ICLR*, 2025.

[25] Huadai Zhu et al. FlashAudio: Rectified flows for fast and high-fidelity text-to-audio generation. *Proc. ACL*, 2025. arXiv:2410.12266.

[26] Apple Machine Learning Research. Stable diffusion with Core ML on Apple Silicon. `https://github.com/apple/ml-stable-diffusion`, 2022.

[27] Apple Machine Learning Research. Deploying transformers on the Apple Neural Engine. `https://machinelearning.apple.com/research/neural-engine-transformers`, 2022.

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. Attention is all you need. In *Proc. NeurIPS*, 2017.

[29] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proc. ICCV*, 2023.

[30] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proc. EMNLP*, 2019.