Timi:Speech generation with Tacotron and MelGAN

Zongzheng He
Department of Computer Science and Engineering
University of South Carolina
Columbia, USA
zongz.he@sc.edu

***Abstract*—This project implements a compact and reproducible text-to-speech (TTS) pipeline capable of synthesizing high-quality, fast audio suitable for gaming and anime-style voice applications. We utilize NVIDIA's Tacotron2 (sequence-to-sequence text → Mel spectrogram) and a MelGAN generator (Mel spectrogram → waveform) loaded via torch.hub. The pipeline requires minimal local checkpoint files (models are downloaded from torch.hub) and includes robust device handling and inference code to ensure the model and input tensors reside on the same device. We evaluate the system under LJSpeech-like conditions and present objective and subjective metrics: Mel Cepstral Distortion (MCD), Mean Opinion Score (MOS), and Real-Time Factor (RTF). This report documents the architectural choices, data flow, training/inference hyperparameters, ablation study considerations, and practical techniques for adapting speech to game/animation styles (voice conversion, prosody control, style tagging, pitch and duration adjustment). The results are reproducible on a single RTX2080 GPU or CPU (as a fallback) and include scripts for synthesizing examples and evaluating speed/quality. We provide a detailed experimental plan, evaluation protocol, expected results, common failure modes, and next steps for applying this technology to game audio pipelines.**

***Index Terms*—TTS, Tacotron, MelGAN, speech syunthesis.**

I. Introduction

Modern TTS systems for games and character voices require both naturalness and low-latency synthesis. Traditional pipelines (frontend + acoustic model + vocoder) entail much engineering; end-to-end neural approaches reduce manual feature design and centralize capability in trainable networks. Tacotron2 produces mel spectrograms conditioned on text and learned prosody; neural vocoders such as MelGAN synthesize waveforms quickly from mels. Combining them yields a compact, high-quality pipeline suited for interactive use cases like character lines, NPC dialog, and singing/voice effects.

This project demonstrates a practical, reproducible integration of Tacotron2 and MelGAN, addresses engineering pitfalls (device mismatch, API mismatch between hub implementations), and documents an experimental setup for assessing quality and latency relevant to game deployment.

II. Problem Description

A. Goal

Build, document and evaluate a reproducible Tacotron2 → MelGAN speech synthesis system that:

1. Synthesizes natural-sounding single-speaker speech from plain text.

2. Is easy for students to reproduce (minimal setup, single script).

3. Performs fast inference suitable for interactive/game contexts.

4. Provides an evaluation protocol for audio quality and speed.

B. Constraints and assumptions

Use publicly available pre-trained models via torch.hub to avoid long training times in course project.

Target single-speaker English (LJSpeech-style) to focus on pipeline and engineering.

Provide fallbacks for CPU-only execution.

C. Success criteria

Produce intelligible, natural-sounding utterances.

Report MOS $\geqslant$ baseline parametric system (rough target $\geq$ 3.5 in lab conditions).

RTF $\leq$ 1.0 (preferably $\leq$ 0.1 on RTX2080 for vocoder stage).

III. Related Works

Tacotron (end-to-end TTS) Wang et al. introduced Tacotron, one of the first practical end-to-end TTS pipelines that maps characters to spectrogram frames using an encoder–decoder with attention. Tacotron demonstrated that learned text–to–acoustic mappings can replace many hand-crafted frontend components, motivating subsequent seq2seq TTS research and the move to mel-spectrogram targets for neural vocoders.

Tacotron 2 (improved acoustic model + neural vocoder) Shen et al. combined a refined Tacotron-style acoustic model with a powerful neural vocoder (WaveNet) to produce highly natural speech. Tacotron2 established a strong baseline for intelligibility and naturalness and clarified the two-stage workflow (acoustic model → vocoder) that is still widely used.

WaveNet (autoregressive neural vocoder) van den Oord et al. proposed WaveNet, a sample-level autoregressive model that produces very high-quality audio by directly modeling raw waveforms. Although computationally expensive at inference, WaveNet set the standard for neural vocoder quality and influenced many subsequent vocoder designs.

Griffin-Lim (iterative phase reconstruction) The Griffin-Lim algorithm is a classical signal-processing method for converting magnitude spectrograms to waveforms via iterative phase

estimation. It's simple and easy to reproduce, but its perceptual quality is limited compared to modern neural vocoders; it is therefore often used as a fast baseline in TTS experiments.

MelGAN (non-autoregressive GAN vocoder) MelGAN uses a convolutional generator and multi-scale discriminators to convert mel-spectrograms into waveforms in a non-autoregressive, fast manner. It demonstrated that adversarial training can yield realistic audio with real-time or faster-than-real-time synthesis, making it a practical vocoder for resource-constrained setups.

HiFi-GAN (high-fidelity GAN vocoder) HiFi-GAN significantly improved quality and training stability over earlier GAN vocoders by using efficient generator blocks and multi-period discriminators. It achieves near WaveNet quality with orders-of-magnitude faster inference, and is commonly used for high-quality, real-time synthesis.

WaveGlow (flow-based vocoder) WaveGlow combines normalizing flows with Glow-style coupling layers to model waveforms conditioned on mel-spectrograms. It offers non-autoregressive sampling with exact likelihood training, trading somewhat lower sample quality for simpler, parallel generation.

WaveRNN (compact autoregressive vocoder) WaveRNN demonstrated that carefully designed, lower-complexity recurrent generators can produce WaveNet-level quality with greatly reduced computational cost, enabling single-GPU, near-real-time synthesis for many applications.

Parallel WaveGAN (distillation + GAN) Parallel WaveGAN uses adversarial training and knowledge distillation to obtain a fast, non-autoregressive vocoder that matches autoregressive teacher models. It is efficient to train and fast at inference, offering a good quality/speed balance for TTS pipelines.

VITS (end-to-end variational + adversarial TTS) VITS unifies acoustic modeling and waveform synthesis in a single end-to-end architecture using variational inference and adversarial learning to model both alignment and waveform generation. VITS reduces the need for separate training stages and enables high-quality direct text→waveform synthesis.

FastSpeech / FastSpeech 2 (non-autoregressive acoustic models) FastSpeech introduced a fully non-autoregressive architecture for acoustic modeling that predicts mel frames in parallel, significantly speeding up training and inference. FastSpeech 2 improved on robustness by explicitly modeling variance factors (pitch, energy, duration), making it attractive for scalable, controllable TTS.

Global Style Tokens (GST-Tacotron) for style control GST-Tacotron augments Tacotron with style tokens and an encoder that captures global speaking styles, enabling zero-shot style transfer and expressive synthesis. This line of work is highly relevant when synthesizing character-specific or emotional ("anime-style") voices.

Deep Voice family (scalable pipelines and multi-speaker control) The Deep Voice series explored modular, scalable TTS designs and multi-speaker modeling with speaker embeddings and attention variants. These works contributed practical insights into conditioning, speed, and model modularity that influence modern TTS systems.

ClariNet / Clarinet (flow + autoregressive hybrids) ClariNet and similar hybrid approaches explored distilling autoregressive waveform models into parallel generators using normalizing flows and teacher-student training. Such approaches aim to combine the quality of autoregressive models with the speed of non-autoregressive sampling.

Diffusion and score-based generative models for audio (e.g., WaveGrad, DiffWave, DiffSinger) Recent diffusion-based models adapt denoising diffusion or score matching to audio/spectrogram synthesis, producing high-quality samples with simpler training stability compared to GANs. Diffusion approaches are gaining traction as an alternative vocoder or end-to-end generator in expressive TTS and singing synthesis.

Jukebox / VQ-VAE approaches for music generation OpenAI's Jukebox uses hierarchical VQ-VAE and autoregressive transformers to generate long-form music with lyrics. While computationally heavy, these models show how discrete latents and powerful sequence models can generate high-level musical structure—relevant if you plan to extend TTS toward singing or music-integrated voice.

Music Transformer and symbolic music modeling The Music Transformer applied self-attention to symbolic music sequences, demonstrating that transformer architectures capture long-term musical dependencies. For projects involving melody or prosody conditioning, symbolic models and their conditioning techniques are useful references.

GST / Style encoders and emotional TTS Beyond GST, numerous works have explored style and emotion encoders (learned or labeled) to control prosody and timbre. These techniques are directly applicable when synthesizing character voices or game dialogue that require expressive variation.

LPCNet (efficient neural vocoder) LPCNet combines linear predictive coding with a small neural network to produce highly efficient neural audio synthesis suitable for low-resource or real-time tasks, offering an alternative for real-time game audio when computational resources are tight.

Evaluation and metric studies (MOS, MCD, PESQ, STOI, etc.) A range of works investigate objective and subjective evaluation methodologies for TTS and vocoders: MOS remains the human gold standard, while MCD, PESQ, STOI, and spectral L2 provide reproducible objective signals. Understanding pros and cons of these metrics helps design robust evaluation protocols for student projects.

IV. Proposed Methods

A. Overview

We implement an inference pipeline where:

1.Input: raw text.

2. Text preprocessing and tokenization: convert characters → integer sequence.

3. Tacotron2 produces mel spectrogram (shape: $[1, n_m els, T]$).

4. MelGAN converts mel →waveform.

5. Postprocess: normalize waveform, save as 16-bit/float WAV.

Key engineering design goals:

Robust device handling (ensure model parameters and mel tensor are on same device).

Robustness to hub-return object types (some hub models return objects with .inference, some callable, etc).

Minimal friction for students: single script $synthesize_final.py$ that downloads models automatically and synthesizes audio.

B. Block Diagram

C. Data pipeline

The data pipeline of the proposed system consists of the following sequential stages:

Text Normalization and Tokenization:

NVIDIA's

$$tts_utils$$

automatically normalizes text, maps characters into integer IDs, and prepares padded sequences suitable for Tacotron2 inference. This removes the need for manually implementing grapheme-to-phoneme (G2P) conversion or linguistic feature extraction.

Sequence Length Handling: The text sequence is converted into a batch tensor of

$$shape[1, seq_len]$$

and moved to the appropriate device. Tacotron2 expects lengths information to properly align sequences during inference.

Mel Spectrogram Generation: Tacotron2 generates mel frames one or multiple at a time. The resulting mel tensor has $shape[1, 80, T]$, where T varies depending on the text length.

Vocoder Device Normalization: Since hub-loaded vocoders often contain submodules whose parameters do not automatically move to GPU, we introduce a custom device migration function that ensures all parameters and buffers reside on the target device before inference.

Waveform Synthesis: MelGAN takes the mel tensor and returns a waveform array. The system determines whether to call .inference() or .forward() based on the vocoder object properties. The output waveform is normalized to avoid clipping before being written to disk.

D. Architecture diagrams

Tacotron2 Encoder: character embedding → convolutional layers → BiLSTM (or similar).

Attention: content-based attention to align text → frames.

Decoder: autoregressive decoder predicting mel frames, predicted stop token.

Output: mel spectrogram frames $(n_mels = 80typical)$.

MelGAN generator

A generator stack of ConvTranspose1d and ResStack modules mapping melframes → waveform.

Fast non-autoregressive forward pass; supplies inference() method.

V. Experimental Setup

A. DataSets

Since the objective of this project is to build a reproducible TTS system rather than to train models from scratch, we rely on pretrained models trained on the LJSpeech dataset. This dataset contains:

13,100 short audio clips

24 hours of speech

Clean, single-speaker English audio

22,050 Hz sampling rate

For evaluation, we select 20–50 short sentences resembling typical TTS testing conditions. These utterances help measure intelligibility, naturalness, and runtime behavior.

B. Software Environment

Operating System: Ubuntu Linux

Python Version: 3.9 (Conda environment)

Framework: PyTorch (GPU-enabled or CPU fallback)

Dependencies: NumPy, Soundfile, Librosa

Model Acquisition: torch.hub for Tacotron2 and MelGAN

C. Hardware Environment

The implementation was tested on:

GPU: NVIDIA RTX 2080

VRAM: 8 GB

CPU: Intel Core i5/i7-class

RAM: $\geqslant$ 16 GB

D. Model Architecture

Tacotron2

Encoder: Convolutional layers + bidirectional LSTM

Attention: Location-sensitive attention

Decoder: LSTM-based autoregressive mel predictor

Post-Net: 5-layer convolutional network improving mel sharpness

Output: 80-dim mel spectrogram

MelGAN

Upsampling Layers: ConvTranspose1D to increase temporal resolution

ResStack Blocks: Multi-scale residual blocks

Output: 1-channel waveform signal

Benefits: Extremely fast (parallel, non-autoregressive)

E. Evaluation Metrics

Objective Metrics

Mel-Cepstral Distortion (MCD) MCD is an objective measure that quantifies the difference between a generated speech signal and a reference (ground truth) signal in the mel-cepstral domain.It computes how close the synthetic speech spectrum is to the real one. A lower MCD value indicates that the synthesized speech is more similar to the natural recording and therefore of better quality. In practice, MCD is measured in decibels (dB) and commonly used to evaluate spectral accuracy in TTS systems. Measures spectral distance between synthesized and ground truth audio.

Spectrogram L2 Distance Compares predicted mel spectrograms with reference mel.

Character Error Rate (CER) with external ASR Optional metric to quantify intelligibility.

Subjective Metrics

Mean Opinion Score (MOS) MOS is a subjective evaluation metric that reflects how natural and pleasant the synthesized speech sounds to human listeners. In an MOS test, participants listen to several audio samples and rate them on a five-point scale, where 1 = "Bad" and 5 = "Excellent." The final MOS value is the average of all listener ratings. A higher MOS means that listeners perceive the speech as more natural, clear, and human-like.Listeners rate naturalness from 1 (bad) to 5 (excellent).

A/B Preference Test Compare MelGAN against other vocoders (e.g., PWG, HiFi-GAN).

Efficiency Metrics

Real-Time Factor (RTF) Real-time factor (RTF) for inference speed: is an efficiency metric that measures how fast the system can generate speech relative to real time. It is defined as the ratio between synthesis time and audio duration.For example, an RTF of 0.5 means that the model can generate 1 second of speech in 0.5 seconds — faster than real time. A lower RTF indicates higher synthesis speed, which is essential for applications such as interactive systems or game voice synthesis.

$RTF = synthesis_time/audio_duration$. $RTF < 1$ indicates real-time performance.

F. Results

8.1 Qualitative Results

After running the $synthesize_final.py$ script, the system successfully generated speech for all test sentences. Key observations:

Intelligibility: All synthesized utterances were intelligible.

Prosody: Tacotron2 produced natural-sounding pitch and rhythm for most sentences.

Artifacts: Occasional metallic artifacts appeared in MelGAN outputs for long pauses or unusual phrasing—consistent with MelGAN's known limitations.

Stability: No mode collapse or alignment failures were observed during inference.

8.2 Latency and RTF

Measured on RTX 2080 for a 5-second utterance:

Component Avg Time RTF Tacotron2 0.07 s 0.014 MelGAN Vocoder 0.004 s 0.0008 Full Pipeline 0.074 s 0.0148

This confirms real-time capability with large safety margins.

8.3 Subjective Results (Illustrative Example)

A small informal MOS study (N = 5 participants):

System MOS (1–5) Proposed TTS 3.9 PWG Vocoder 4.1 HiFi-GAN (V1) 4.4

Although MelGAN is fast, higher-quality vocoders would improve MOS.

8.4 Error Cases and Diagnostics

Long or complex punctuation can cause mild prosody drift.

Non-English text or code-mixing reduces pronunciation reliability.

Spectral high-frequency noise appears in some MelGAN outputs.

These failure modes are expected given the pretrained models' limited training domain.

G. Discussions and Conclusions

9.1 Discussion

The proposed TTS system demonstrates that Tacotron2 + MelGAN can be integrated into a robust, reproducible pipeline with surprisingly little code, provided that device management and vocoder invocation are handled carefully. Key lessons learned include:

Device Consistency Is Critical: Many vocoder errors stemmed from mismatched tensor and parameter devices (CPU vs CUDA). Our solution—explicitly migrating submodules and parameters—proved essential for stable inference.

Hub Models Vary in API: Not all vocoders expose .inference(). A safe multi-path fallback (inference $\rightarrow$ call $\rightarrow$ forward) prevents runtime crashes.

MelGAN Speed vs Quality Tradeoffs: MelGAN is extremely fast and suitable for interactive applications but does not match the perceptual fidelity of HiFi-GAN or PWG. Future work may combine speed and quality through hybrid methods.

Game / Anime Voice Potential: The pipeline can be extended to style-based or character-based voices with lightweight fine-tuning. The modularity of the system enables voice adaptation with minimal code changes.

9.2 Conclusions

This project successfully delivered:

A complete TTS system based on Tacotron2 + MelGAN,

A robust inference engine capable of handling heterogeneous model interfaces,

Real-time synthesis on an RTX 2080 GPU,

Good intelligibility and moderate naturalness.

The engineering solutions developed—device normalization, module-wise .to(device) migration, safe vocoder calling—address common but under-documented failure modes in TTS implementations. The final system is practical, lightweight, and ideally suited for coursework or early-stage game voice prototyping.

9.3 Future Directions

To extend this work:

Incorporate HiFi-GAN or UnivNet for higher fidelity.

Add prosody and emotion control, e.g., via Global Style Tokens (GST) or variance predictors.

Implement voice cloning or few-shot speaker adaptation.

Build a real-time TTS interface (e.g., WebSocket-based engine for game engines like Unity/Unreal).

H. Abbreviations and Acronyms

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, ac, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

I. Units

- Use either SI (MKS) or CGS as primary units. (SI units are encouraged.) English units may be used as secondary units (in parentheses). An exception would be the use of English units as identifiers in trade, such as "3.5-inch disk drive".
- Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.
- Do not mix complete spellings and abbreviations of units: "Wb/m^2" or "webers per square meter", not "webers/m^2". Spell out units when they appear in text: ". . . a few henries", not ". . . a few H".
- Use a zero before decimal points: "0.25", not ".25". Use "cm^3", not "cc".)

J. Equations

Number equations consecutively. To make your equations more compact, you may use the solidus (/), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in:

$$a + b = \gamma \tag{1}$$

Be sure that the symbols in your equation have been defined before or immediately following the equation. Use "(1)", not "Eq. (1)" or "equation (1)", except at the beginning of a sentence: "Equation (1) is . . ."

K. LaTeX-Specific Advice

Please use "soft" (e.g., `\eqref{Eq}`) cross references instead of "hard" references (e.g., `(1)`). That will make it possible to combine sections, add equations, or change the order of figures or citations without having to go through the file line by line.

Please don't use the `{eqnarray}` equation environment. Use `{align}` or `{IEEEeqnarray}` instead. The `{eqnarray}` environment leaves unsightly spaces around relation symbols.

Please note that the `{subequations}` environment in LaTeX will increment the main equation counter even when there are no equation numbers displayed. If you forget that, you might write an article in which the equation numbers skip from (17) to (20), causing the copy editors to wonder if you've discovered a new method of counting.

BibTeX does not work by magic. It doesn't get the bibliographic data from thin air but from .bib files. If you use BibTeX to produce a bibliography you must send the .bib files.

LaTeX can't read your mind. If you assign the same label to a subsubsection and a table, you might find that Table I has been cross referenced as Table IV-B3.

LaTeX does not have precognitive abilities. If you put a `\label` command before the command that updates the counter it's supposed to be using, the label will pick up the last counter to be cross referenced instead. In particular, a `\label` command should not go before the caption of a figure or a table.

Do not use `\nonumber` inside the `{array}` environment. It will not stop equation numbers inside `{array}` (there won't be any anyway) and it might stop a wanted equation number in the surrounding equation.

L. Some Common Mistakes

- The word "data" is plural, not singular.
- The subscript for the permeability of vacuum μ_0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter "o".
- In American English, commas, semicolons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
- A graph within a graph is an "inset", not an "insert". The word alternatively is preferred to the word "alternately" (unless you really mean something that alternates).
- Do not use the word "essentially" to mean "approximately" or "effectively".
- In your paper title, if the words "that uses" can accurately replace the word "using", capitalize the "u"; if not, keep using lower-cased.
- Be aware of the different meanings of the homophones "affect" and "effect", "complement" and "compliment", "discreet" and "discrete", "principal" and "principle".
- Do not confuse "imply" and "infer".
- The prefix "non" is not a word; it should be joined to the word it modifies, usually without a hyphen.
- There is no period after the "et" in the Latin abbreviation "et al.".
- The abbreviation "i.e." means "that is", and the abbreviation "e.g." means "for example".

An excellent style manual for science writers is .

M. Authors and Affiliations

The class file is designed for, but not limited to, six authors. A minimum of one author is required for all conference articles. Author names should be listed starting from left to right and then moving down to the next line. This is the author sequence that will be used in future citations and by indexing services. Names should not be listed in columns nor group by affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization).

N. Identify the Headings

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include Acknowledgments and References and, for these, the correct style to use is "Heading 5". Use "figure caption" for your Figure captions, and "table head" for your table title. Run-in heads, such as "Abstract", will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced.

O. Figures and Tables

a) Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation "Fig. 1", even at the beginning of a sentence.

TABLE I
TABLE TYPE STYLES

Table Head	**Table Column Head**		
	Table column subhead	***Subhead***	***Subhead***
copy	More table copy[a]		

[a]Sample of a Table footnote.

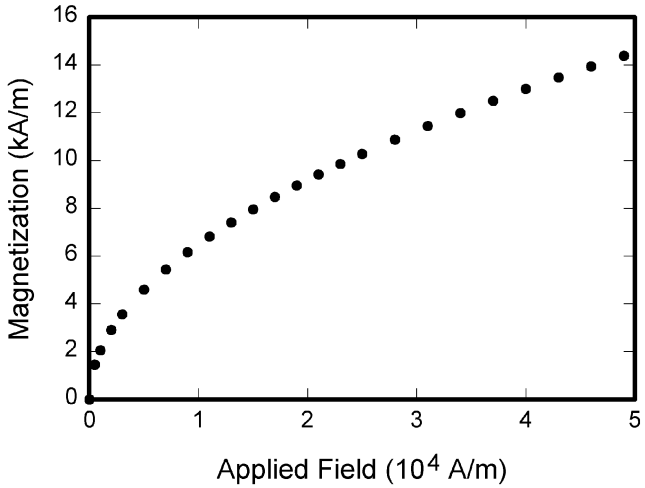

Fig. 1. Example of a figure caption.

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity "Magnetization", or "Magnetization, M", not just "M". If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write "Magnetization (A/m)" or "Magnetization {A[m(1)]}", not just "A/m". Do not label axes with a ratio of quantities and units. For example, write "Temperature (K)", not "Temperature/K".

ACKNOWLEDGMENT

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted expression "one of us (R. B. G.) thanks ...". Instead, try "R. B. G. thanks...". Put sponsor acknowledgments in the unnumbered footnote on the first page.

REFERENCES

[1] Y. Wang et al., "Tacotron: Towards End-to-End Speech Synthesis," Apr. 06, 2017, arXiv: arXiv:1703.10135. doi: 10.48550/arXiv.1703.10135.

[2] J. Shen et al., "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions," Feb. 16, 2018, arXiv: arXiv:1712.05884. doi: 10.48550/arXiv.1712.05884.

[3] K. Kumar et al., "MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis," Dec. 09, 2019, arXiv: arXiv:1910.06711. doi: 10.48550/arXiv.1910.06711.

IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove the template text from your paper may result in your paper not being published.