Exploiting Neural Audio Codecs for Edge-to-Gateway Speech Processing

1st Stefano Ciapponi
Fondazione Bruno Kessler, University of Trento
Trento, Italy
sciapponi@fbk.eu

2nd Elisabetta Farella
Fondazione Bruno Kessler
Trento, Italy
efarella@fbk.eu

***Abstract*—Neural Audio Codecs have become powerful tools for audio processing, offering learnable compression methods that balance high compression ratios with perceptual quality. This paper introduces a signal processing system that utilizes the latent space of Neural Audio Codecs for signal reconstruction and feature extraction in edge computing environments. We design a lightweight NAC encoder inspired by SoundStream, optimized for resource-constrained devices. Our evaluation on speech recognition and classification tasks highlights the system's adaptability to Internet of Things applications. The proposed design achieves a 40× audio waveform compression with only a 3% increase in word error rate for transcription tasks and a 94.6% accuracy on end-to-end intent classification, demonstrating its practicality for real-world deployment. Additionally, the encoder operates at a real-time factor of 1.77 on an ARM Cortex-A53 using a single thread for intra/inter-operation, ensuring efficient real-time compression and 12-8 times less energy consumption compared to the original model encoder.**

***Index Terms*—Edge AI, Internet of Things, Speech Processing, Neural Audio Codecs**

I. Introduction

In the era of the Internet of Things (IoT) and pervasive, distributed intelligence, speech processing has become a key enabler for numerous applications. Advancements in generative AI have further accelerated the adoption of voice-driven technologies, powering smart assistants, real-time transcription services, hands-free human-computer interaction, and accessibility solutions. Thus, transmitting high-quality speech data under constrained bandwidth conditions has become a pressing challenge. Neural Audio Codecs (NACs) are a promising solution in this context, implementing learnable compression algorithms that achieve superior compression ratios while maintaining high perceptual quality and, compared to traditional coding methods [1], are capable of extracting non-linear and global information from the audio signal.

NACs have progressed considerably from initial autoencoder architectures. EnCodec [2] established a foundation with residual vector quantization and a loss balancer to stabilize training, rather than focusing on adversarial training. SoundStream [3] introduced a GAN-based end-to-end streaming neural audio codec optimized for general audio, using residual vector quantization to support multiple bitrates. Subsequently, DAC [4] improved upon EnCodec by addressing codebook collapse and modifying activation functions to better reconstruct periodic signals like speech and music, while AudioDec [5] provided a real-time, streamable neural audio codec with strong performance across compression, latency, and reconstruction quality. Contemporary research has prioritized computational efficiency. HILCodec [6] is noteworthy for its lightweight, time-domain convolutional architecture, adapted from SEANet, with optimized quantization strategies and a latency-aware design. Xu et al. [7] address IoT-specific constraints by achieving intelligible speech communication at 0.5 kbps using an intra-lstm-based design and power-efficient inference optimized for NBIoT deployments.

While their primary advantage lies in enabling high-fidelity reconstruction, many downstream tasks—such as automatic speech recognition or audio-based classification—do not necessarily require perfect audio quality, but rather an intelligible representation that retains relevant features. Instead, performing compression locally at the very edge brings a significant advantage by drastically reducing bandwidth requirements, enabling efficient data transmission and real-time processing in constrained environments.

To the best of our knowledge, no study has explored NAC deployment on edge devices or evaluated their performance with downstream applications at the gateway level, highlighting a gap between theoretical innovation and practical implementation. For this reason, this work presents the design and implementation of a signal processing system that exploits the dual nature of NACs' latent space representation, enabling both signal reconstruction and feature extraction in resource-constrained edge computing scenarios. The system architecture implements a lightweight NAC encoder inspired by SoundStream [3], with optimizations for efficient signal processing on edge devices. Our implementation encompasses the full signal processing pipeline, from acoustic sensing through compression to gateway processing, with particular attention to computational efficiency and processing constraints. The processing chain is evaluated across two distinct speech processing tasks: audio reconstruction for speech recognition and direct classification using the compressed latent representations. The testbed enables systematic characterization of the signal processing chain and performance evaluation at various levels of latent space residual transmission. This dual-purpose architecture showcases the versatility of NAC-based processing in audio reconstruction and feature analysis, while efficiently utilizing resources at the edge.

II. System Architecture and Methodology

This section details the experimental setup and testbed, including the developed models, system architecture, and key performance metrics, focusing on performance degradation at different bitrates.

A. General Overview

The system architecture, illustrated in Figure 1, implements a two-tier edge computing infrastructure. The first tier consists of distributed edge nodes, implemented on Raspberry Pi 3B+ platforms, which are selected to demonstrate functionality on resource-constrained hardware commonly found in practical deployments. These nodes, equipped with MEMS microphones for audio data acquisition, perform initial preprocessing before publishing the captured audio data to a designated 'audiolog' channel, along with device identification and temporal metadata for traceability. The second tier comprises gateway devices based on Nvidia Jetson Xavier platforms, which serve as the intermediate computational layer. These devices perform computationally intensive operations, primarily running transformer-based models for audio processing, while also facilitating communication between edge nodes and cloud services when additional processing capabilities are required. Communication between system components is managed through a MQTT broker, which can be hosted on the gateway or on a separate dedicated device. MQTT's publish-subscribe model enables flexible system scaling allowing us to add or remove devices without reprogramming the gateway. The MQTT broker handles data transmission with configurable Quality of Service (QoS) levels to ensure appropriate reliability based on application requirements.

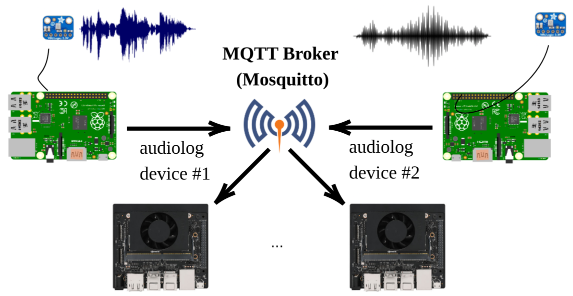

Fig. 1. System architecture: 2-tier edge computing infrastructure, with distributed edge nodes (Raspberry Pi 3B+) for audio capture and preprocessing, and gateway devices (Nvidia Jetson Xavier) for heavy computation.

B. Tiny NAC Encoder

Our work focused on developing a Tiny NAC Encoder optimized for edge deployment. We based our architecture on SoundStream [3], a fully convolutional neural audio codec that leverages 1D convolutions—operations widely optimized on edge-oriented deep learning frameworks. SoundStream's architecture incorporates two key elements: a discriminator-based loss trained through adversarial learning to preserve perceptual quality and learn phase information, and Residual Vector Quantization (RVQ) for precise bitrate control. Building upon this foundation, we prioritized minimizing the encoder's computational requirements while maintaining audio fidelity suitable for downstream processing. To achieve this optimization, we redesigned the encoder using PhiNets [8], a neural architecture that builds upon inverted residual blocks [9]. PhiNets have demonstrated remarkable efficiency in audio processing tasks [10]–[12], offering a flexible architecture controlled by four key parameters: expansion factor, number of convolutional blocks, shape factor, and width multiplier. By implementing a causal variant of PhiNet, we achieved substantial model compression, reducing the encoder size from 19MB to 2MB. This optimization involved the following architectural modifications, where C is the number of channels and D is the latent space dimension:

TABLE I
Encoder Configurations Comparison

Configuration	**C**	**D**	**Stride Sequence**
Original (19MB)	32	128	[2, 4, 5, 8]
Optimized (2MB)	32	256	[4, 5, 16]

We further optimized memory usage by reducing the RVQ codebook from 1024 to 256 entries, resulting in a 4x reduction in memory requirements and a corresponding decrease in per-sample bitrate (8 bits per codebook index). For a complete evaluation, we trained the model using 16 codebooks in the quantization process, allowing the analysis of the quantization effects on the performance of downstream tasks. The complete codebook ensemble requires approximately 22MB of memory, making the encoder and RVQ weight around 24.5 MB.

C. Downstream Models

We evaluated our compression system on two downstream tasks: Automatic Speech Recognition (ASR) and Speech Intent Classification. For ASR, we assessed the quality of our reconstructed audio by transcribing it using Whisper [13], a widely adopted ASR model. For the Intent Classification task, we utilized the compressed latent space as direct input to a model. Specifically, we removed the convolutional feature extraction backbone from Wav2Vec2 (base 960h) and fed the transmitted compressed embeddings directly to the transformer component, essentially using our Tiny NAC Encoder as a feature extractor. During fine-tuning on the FluentSpeechCommands dataset [14], we kept the NAC encoder frozen and only trained the transformer blocks, allowing us to evaluate how well the pre-compressed latent representations preserve semantic information for classifying 31 different speech command intents. Both models have been chosen in a configuration suitable for deployment on the system defined in the previous sections.

III. Implementation and Experimental Setup

In this section, we will delve into the implementation details from an Edge Node and Gateway perspective. Finally, we will describe the models' training setups and define the metrics we benchmarked to characterize the system.

A. Edge Node

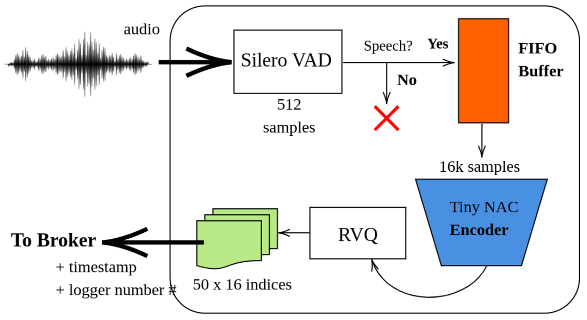

Fig. 2. Edge node processing pipeline showing audio capture, VAD, speech segment buffering, compression by Tiny NAC Encoder and data transmission to the broker. Multithreading optimizes the pipeline for efficiency.

The edge node processing pipeline is depicted in Figure 2. Audio is captured using an I2S microphone in S16LE format at 16 kHz and subsequently processed through the proposed pipeline. Initially, a Voice Activity Detector (SileroVAD[1]) analyzes the incoming audio, classifying each 512-sample segment as either speech or non-speech. Segments identified as speech—i.e., those exceeding a predefined probability threshold determined during a tuning phase—are stored in a FIFO buffer. Once the buffer accumulates one second of speech data, it is dequeued and compressed using the Tiny NAC Encoder. Compression is achieved via RVQ, resulting in a 40× compression ratio when all quantization tables are utilized. The quantized latent representations are subsequently transmitted to the broker. The implementation leverages the multithreading capabilities of the RPi3B+ to optimize pipeline efficiency. One thread manages audio capture, voice activity detection, and buffering, while a second thread handles dequeuing, compression, and transmission once the buffer contains at least 16,000 samples. Conditional variables are employed to synchronize buffer operations, ensuring that data is either enqueued or dequeued without conflicts.

B. Gateway

Figure 3 illustrates the gateway processing pipeline, which can operate in two distinct configurations. In configuration **A**, the RVQ module utilizes indices to reconstruct the encoder's latent space representation. A FIFO buffer maintains one second of latent space representations up to 10 seconds, which are then processed through the NAC decoder to reconstruct the audio waveform. Finally, the Whisper model performs speech-to-text transcription on the reconstructed audio data. Configuration **B** bypasses the NAC decoder entirely, instead feeding the encoder's latent space representations directly into a modified version of Wav2Vec. This modified Wav2Vec implementation, which excludes the feature extraction component, performs intent classification on the input data.

[1]https://github.com/snakers4/silero-vad

C. Experimental Setup

The specifications of the edge node and gateway are presented in Table II. The edge node operates on Raspberry PiOS 6.6 (64-bit), while the Jetson platform utilizes JetPack5.

TABLE II
HARDWARE SPECIFICATIONS OF EDGE NODE AND GATEWAY

Specification	Raspberry Pi 3B+	Jetson AGX Xavier
Processor	Quad-core Cortex-A53 @ 1.4 GHz	8-core ARM v8.2 + 512-core Volta GPU
Memory	1 GB LPDDR2	16 GB LPDDR4x @ 136.5 GB/s

1) NAC Training Process: The NAC training process was conducted over 35,000 steps with a batch size of 16, using the LibriTTS360 dataset for training and the Test-clean set for evaluation. Initially, training was performed for approximately 25,000 steps using an RVQ of size 1024. Subsequently, the RVQ was replaced with a size 256 RVQ for the remaining 10,000 steps. Due to computational limitations, training was halted after 35,000 steps. The training setup is analogous to the original SoundStream training, combining a vector quantization-based neural audio codec with adversarial and reconstruction losses.

2) Wav2Vec Finetuning: The Wav2vec finetuning was performed for 100 epochs, using an AdamW Optimizer, Cosine Annealing Learning Rate Scheduler and a 3e-5 learning rate. The audio is first processed in the TinyNAC Encoder and quantizer models at 1s chunks and then fed to a projection Linear Layer, which projects the features of shape 256 to 768 (Wav2Vec size); the projected features are finally fed in the Wav2Vec Transformer blocks and in a classification head.

3) Deployment: For deployment, the trained model is converted to ONNX format and executed using ONNX Runtime on the RPi. Mosquitto is used as the MQTT broker, while Hydra is employed to manage configurations in both training and deployment setups. Additionally, Hugging Face and OpenAI Whisper are used to download the pre-trained weights for Whisper and Wav2Vec.

All the training scripts, metrics and deployment packages are available in a public GitHub organization[2].

D. Evaluation Metrics

Our system evaluation employs both signal quality and task-specific performance metrics.

1) Signal Quality Metrics: PESQ (-0.5 to 4.5, higher better) assesses perceptual quality, STOI (-1 to 1, higher better) quantifies intelligibility, and MCD measures spectral distortion (lower is better).

2) Speech Recognition Performance: WER and CER are evaluated using Whisper across multiple residual quantization levels (4, 8, 12, 16 tables), tested on both raw and reconstructed audio.

3) Intent Classification Performance: Classification accuracy is analyzed relative to quantizer count.

[2]https://github.com/orgs/Exploiting-NACs/repositories

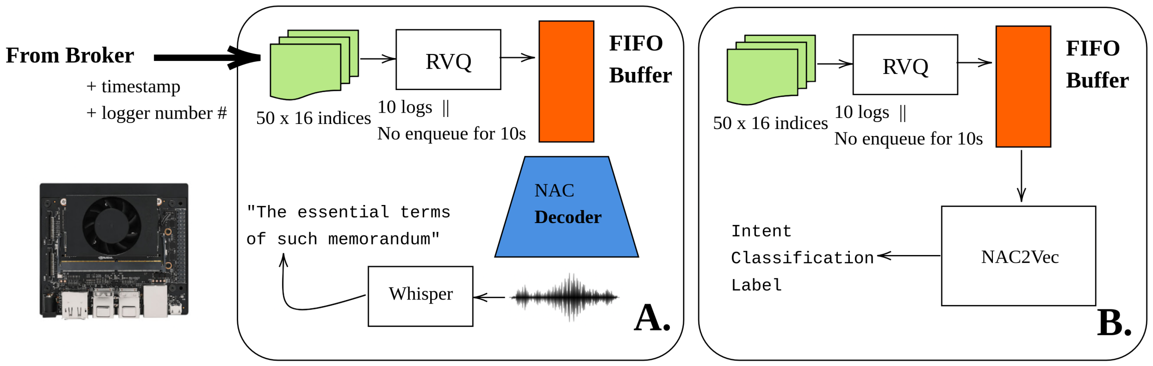

Fig. 3. Gateway processing pipeline, with 2 configurations: Config A. The NAC decoder processes the latent space for audio reconstruction, followed by speech-to-text transcription using Whisper. Config B. bypasses the NAC decoder and uses modified Wav2Vec for intent classification.

4) Deployment Efficiency: Model memory footprint on Raspberry Pi and Real-Time Factor (RTF)/inference time are measured under both automatic thread management and single-thread operation, simulating multi-model deployment constraints.

IV. Results and Discussion

A. Signal Quality Metrics

The training process was completed after 35,000 steps, as outlined in the experimental setup, with the primary goal of ensuring that the reconstructed audio remained suitable for downstream tasks. Evaluation on the LibriTTS test-clean dataset yielded a PESQ score of 1.35, an STOI score of 0.76, and an MCD score of 14.63. The PESQ score indicates moderate perceptual quality, suggesting some audible distortions, but still within a range that is acceptable for use. Similarly, the STOI score of 0.76 indicates that the reconstructed speech retains a reasonable level of clarity. However, the MCD score of 14.63 points to significant spectral distortion, reflecting a notable divergence from the reference audio in terms of spectral characteristics. This distortion is likely a result of early stopping, which may have prevented full convergence and hindered optimal spectral fidelity. While the perceptual and intelligibility metrics show that the reconstructed audio remains functional for downstream applications, the elevated MCD score highlights the potential benefit of further refining the model, possibly through extended training, to improve spectral accuracy and reduce distortion.

B. Speech Recognition Metrics

Table III and Figure 4 present the WER and CER across various quantizer settings, illustrating the impact of compression on transcription accuracy. In real-world speech recognition applications, a WER below 10% is generally considered acceptable for reliable performance.

At 6.4 kbps (16q) and 4.8 kbps (12q), the WER increases from baseline 4% to 7.6% and 8.4%, respectively, while the CER increases from 1.3% to 3.5% and 3.7%. These values remain within the acceptable range, suggesting that such compression levels can be deployed without severely impacting transcription quality. At 3.2 kbps (8q), WER reaches 9.4% and CER 4.4%, nearing the upper limit of usability. The most significant degradation occurs at 1.6 kbps (4q), where WER jumps to 22% and CER to 12.9%, indicating substantial information loss and making accurate transcription impractical. While bitrates of 6.4 kbps and 4.8 kbps ($\leq 40\times$ compression) maintain intelligible speech with moderate WER increases, further compression below 3.2 kbps results in severe degradation, surpassing the acceptable threshold for real-world deployments. These findings emphasize the trade-off between compression efficiency and ASR performance, highlighting the need to balance bitrate reduction with transcription accuracy in practical applications.

TABLE III
WER/CER w.r.t. number of quantizers

WER/CER	**16q (6.4kbps)**	**12q (4.8kbps)**	**8q (3.2kbps)**	**4q (1.6kbps)**
Baseline (WER/CER)	0.04/0.013			
WER	0.076	0.084	0.094	0.22
CER	0.035	0.037	0.044	0.129

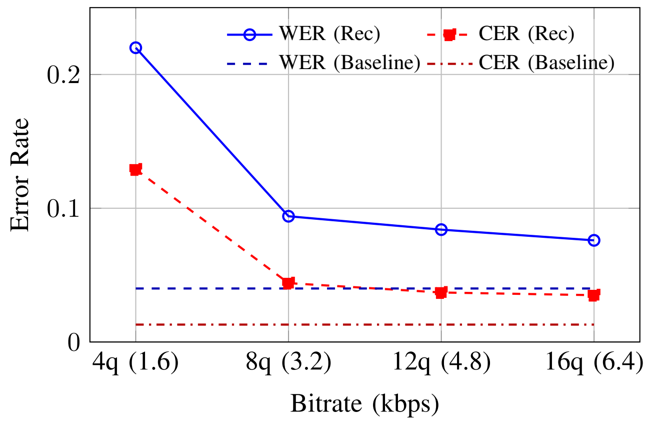

Fig. 4. WER and CER across different bitrates, with baselines. The metrics remain below 10% error rate across bitrates higher than 1.6kbps, making them suitable for deployments in real world scenarios.

C. Intent Classification Metrics

Table IV presents the intent classification accuracy across varying numbers of quantizers. The baseline model, which fine-tunes the entire Wav2Vec framework including its feature extraction backbone, achieves the highest accuracy of 99.70%. In contrast, all other results (quantized configurations) are obtained while keeping the feature extractor frozen. As the number of quantizers is reduced (from 16q to 4q), a corresponding decline in accuracy is observed, with the 4q configuration (1.6 kbps) yielding the lowest accuracy at 85.10%. While this trend aligns with expectations, it is noteworthy that halving the transmission rate results in only an approximate 3% reduction in accuracy. These findings demonstrate that the intent classification accuracy remains sufficiently robust for practical applications, even at lower bitrates, while maintaining the feature extractor in a frozen state for compatibility with other tasks, such as configuration A at the gateway level.

TABLE IV
INTENT CLASSIFICATION ACCURACY W.R.T. NUMBER OF QUANTIZERS

Quantizers	Intent Classification Accuracy
Baseline	99.70%
16q (6.4kbps)	**94.58%**
12q (4.8kbps)	93.95%
8q (3.2kbps)	91.87%
4q (1.6kbps)	85.10%

TABLE V
ON DEVICE BENCHMARKS

MODELS:	*pretrained*	*tiny encoder*
Parameters	17.4M	**2.5M**
Memory (MB)	85.9	24.5
MACC (M)	4.21	1.05
Inference time auto (ms)	3376	392
Inference Time 1 thread (ms)	6594	567
RTF auto	0.3	**2.56**
RTF 1-thread	0.16	1.77
Energy auto/1-thread	4.1/6.0 mWh	0.5/0.49 mWh

D. Deployment efficiency metrics

Table V compares the deployment efficiency metrics between our approach (*tiny encoder*) and the chosen variant of SoundStream (*pretrained*) model from the Python library. The *tiny encoder* features fewer parameters (2.5M vs. 17.4M) and lower memory usage (24.5 MB vs. 85.9 MB), making it more suitable for resource-constrained devices. It also achieves faster inference times (392 ms vs. 3376 ms in auto mode) and a lower RTF (2.56 vs. 0.3). In contrast, the *pretrained* model, has higher computational complexity (4.21 MACC vs. 1.05) but may offer better performance at the cost of higher resource consumption. Moreover, as highlighted in the last row of the table, a 600 Mhz clock benchmark on Rpi3 revealed that our tiny model implementation is 8-12 times more energy efficient on 1 thread and multi-thread configurations, respectively. Energy consumption has been reported for inference on 1 hour of audio data. Both models were exported in ONNX format and tested on a RPi3 using a 1-second audio clip, with 10 warm-up trials followed by 100 benchmarking trials.

V. CONCLUSION

This work demonstrates that NACs provide an efficient solution for edge-to-gateway compression and downstream processing, particularly for tasks that rely on speech-relevant features rather than exact speech reconstruction. Notably, NACs serve both as compression algorithms and feature extractors, enabling a wide range of downstream applications and making the system highly adaptable. Our compact NAC design results in only a minor performance trade-off, with word error rates (WER) of 7.6% and 4% for reconstructed and original audio, respectively. Additionally, the lightweight encoder supports real-time processing on an ARM Cortex-A53, achieving a real-time factor of 1.77 in a single-thread setup—significantly improving energy efficiency compared to the original, more computationally demanding model.

REFERENCES

[1] D. O'Shaughnessy, "Review of methods for coding of speech signals," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2023, no. 1, Feb. 2023.

[2] A. Défossez, J. Copet, G. Synnaeve, and Y. Adi, "High Fidelity Neural Audio Compression," Oct. 2022, arXiv:2210.13438 [eess].

[3] N. Zeghidour, A. Luebs, A. Omran, J. Skoglund, and M. Tagliasacchi, "Soundstream: An end-to-end neural audio codec," *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, vol. 30, p. 495–507, Nov. 2021. [Online]. Available: https://doi.org/10.1109/TASLP.2021.3129994

[4] R. Kumar, P. Seetharaman, A. Luebs, I. Kumar, and K. Kumar, "High-fidelity audio compression with improved RVQGAN," in *Proceedings of the 37th Int. Conference on Neural Information Processing Systems*, ser. NIPS '23. Red Hook, NY, USA: Curran Associates Inc., Dec. 2023, pp. 27 980–27 993.

[5] Y.-C. Wu, I. D. Gebru, D. Marković, and A. Richard, "Audiodec: An Open-Source Streaming High-Fidelity Neural Audio Codec," in *ICASSP 2023 - 2023 IEEE Int. Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Jun. 2023, pp. 1–5, iSSN: 2379-190X.

[6] S. Ahn, B. J. Woo, M. H. Han, C. Moon, and N. S. Kim, "HILCodec: High-Fidelity and Lightweight Neural Audio Codec," *IEEE Journal of Selected Topics in Signal Processing*, vol. 18, no. 8, pp. 1517–1530, Dec. 2024, conference Name: IEEE Journal of Selected Topics in Signal Processing.

[7] Q. Xu, X. Yuan, X. Zhu, M. Ouyang, and G. Liao, "An Ultra-Low Bitrate Neural Audio Codec Under NB-IoT," in *2024 5th Int. Conference on Electronic Communication and Artificial Intelligence (ICECAI)*, May 2024, pp. 259–265.

[8] F. Paissan, A. Ancilotto, and E. Farella, "PhiNets: A Scalable Backbone for Low-power AI at the Edge," *ACM Trans. Embed. Comput. Syst.*, vol. 21, no. 5, pp. 53:1–53:18, Dec. 2022.

[9] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks." IEEE Computer Society, Jun. 2018, pp. 4510–4520.

[10] S. Ciapponi, F. Paissan, A. Ancilotto, and E. Farella, "TinyVocos: Neural Vocoders on MCUs," in *2024 IEEE 5th Int. Symposium on the Internet of Sounds (IS2)*, Sep. 2024, pp. 1–10.

[11] F. Paissan, A. Ancilotto, A. Brutti, and E. Farella, "Scalable Neural Architectures for End-to-End Environmental Sound Classification," in *ICASSP 2022 - 2022 IEEE Int. Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2022, pp. 641–645, iSSN: 2379-190X.

[12] A. Brutti, F. Paissan, A. Ancilotto, and E. Farella, "Optimizing PhiNet architectures for the detection of urban sounds on low-end devices," in *2022 30th European Signal Processing Conference (EUSIPCO)*, Aug. 2022, pp. 1121–1125, iSSN: 2076-1465.

[13] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust Speech Recognition via Large-Scale Weak Supervision."

[14] L. Lugosch, M. Ravanelli, P. Ignoto, V. S. Tomar, and Y. Bengio, "Speech Model Pre-Training for End-to-End Spoken Language Understanding," in *Interspeech 2019*. ISCA, Sep. 2019, pp. 814–818.