Speech Synthesis Using Piper TTS on a Resource-Constrained Edge Device

Mateusz Jankowski
Poznan University of Technology
Poznan, POLAND
Email: mateusz.jankowski@student.put.poznan.pl

Damian Cetnarowicz
Poznan University of Technology
Institute of Automatic Control and Robotics
Poznan, POLAND
Email: damian.cetnarowicz@put.poznan.pl

Aleksandra Świetlicka
Poznan University of Technology
Institute of Automatic Control and Robotics
Poznań, POLAND
Email: aleksandra.swietlicka@put.poznan.pl

Abstract — This paper evaluates Piper TTS as a lightweight mechanism for local speech synthesis intended for offline applications and edge devices. A custom Polish voice model was trained for 5000 epochs and exported to an ONNX file of 60.569 MB containing 15,650,459 parameters. The same model was evaluated in three execution variants: ONNX Runtime on a CPU under Microsoft Windows 10, ONNX Runtime with CUDA acceleration under Microsoft Windows 10, and synthesis on an ARM processor under Raspberry Pi OS Lite on a Raspberry Pi 3B+. Seven test utterances were synthesized three times. On the desktop computer, the average steady-state Real-Time Factor was 0.0377 for CPU and 0.0322 for CUDA. On the Raspberry Pi 3B+, Piper reported an average synthesis-only RTF of 0.982, whereas the end-to-end time RTF increased to 1.718 when the model was reloaded for each request. The results indicate that model loading cost is significant on resource-constrained devices and that interactive applications require a persistently initialized model kept in memory.

Keywords — edge computing, local speech synthesis, ONNX Runtime, Piper TTS, Real-Time Factor, Raspberry Pi

I. INTRODUCTION

Local text-to-speech (TTS) synthesis is useful in systems that require offline operation, predictable response time, and independence from cloud services. Examples include voice announcements, assistive interfaces, communication bots, and mobile or robotic devices. In such applications, voice naturalness is important, but it is not the only engineering criterion. Computational cost, memory requirements, model initialization time, and the ability to run on typical hardware must also be considered.

In this paper, Piper is adopted as a tool for local speech synthesis because it combines a neural voice model with simple execution on CPU, CUDA, and ARM platforms. From a deployment perspective, it is important that the voice model is exported to the ONNX format and that its execution can be moved between different ONNX Runtime backends [1]-[4].

The aim of this work is not to develop a new synthesis architecture. Instead, the study experimentally verifies whether a small custom voice model can be deployed locally on a resource-constrained edge device. The contribution is a controlled comparison of CPU and CUDA execution with Microsoft Windows 10 operating system and a practical test on a Raspbian operating system. The analysis separates steady-state synthesis time from the latency of the entire process, because model loading can dominate overall time of processing short sentences.

II. SELECTED TTS SOLUTIONS

A. Compared Approaches

Before selecting Piper, several groups of text-to-speech solutions were considered. eSpeak NG is a classic and very lightweight formant synthesizer: it works offline, supports many languages, and has low requirements, but as a standalone rule-based synthesizer it offers lower naturalness than the neural Piper pipeline [5]. Tacotron 2 represents the text -> mel-spectrogram -> vocoder approach and can generate high-quality speech, but it requires a more complex pipeline and a higher computational cost [9]. FastSpeech 2 mitigates the limitations of autoregressive models through parallel generation and explicit duration, pitch, and energy modeling, but it still requires a carefully prepared model and vocoder configuration [10]. Coqui TTS is a broad research toolkit with many models, fine-tuning tools, and multilingual support, but it is more demanding than the solution needed for simple offline deployment [11].

Piper provides a compromise between the naturalness of neural synthesis, deployment simplicity, small model size, and the ability to run without cloud services. The attractiveness of this compromise motivated the authors to investigate more detailed implementation properties.

This research was financed with: 0211/SBAD/0226

III. PIPER ARCHITECTURE AND VOICE MODEL

A. Text-to-Speech Pipeline

Piper is a local neural speech synthesis system in which the input text is first normalized and converted into a phonemic representation. In the used pipeline, phonemization is performed by eSpeak NG or by the piper-phonemize component [1], [5]. This means that the model does not directly receive a raw character string, but a sequence of phonetic units together with voice configuration information. This separation is important for Polish because it reduces part of the orthographic ambiguity and allows the neural model to learn the mapping between the phoneme-level representation of the input text and the corresponding acoustic speech waveform.

From the architectural perspective, Piper uses voice models following the practical flow of Variational Inference with adversarial learning for end-to-end Text-to-Speech (VITS). VITS is a single-stage end-to-end system that combines a conditional variational autoencoder, normalizing flows, and adversarial learning [8]. In this view, the text encoder creates a phoneme representation, the alignment and duration prediction mechanism models the length of individual utterance fragments, and the generator acting as a vocoder transforms the latent representation directly into audio samples. The stochastic duration predictor enables the modeling of different rhythms and speaking rates for the same text [8].

B. Export to ONNX and Model Execution

After training, the voice model is exported to an ONNX file [3]. In the evaluated case, the ONNX file had a size of 60.569 MB and contained 15,650,459 parameters. The accompanying JSON file of 6.934 KB stored voice settings, including the sampling rate and parameters used by Piper during synthesis. ONNX stores computations as an operator graph, which makes it possible to execute the same model on CPU, CUDA, or another backend supported by ONNX Runtime [3], [4]. In the experiment, CPUExecutionProvider and CUDAExecutionProvider were compared, and execution on the Raspberry Pi 3B+ was tested separately.

IV. SYSTEM AND METHODOLOGY

A. Experimental Pipeline

The experimental pipeline covered data preparation, model training, and measurement of local speech synthesis. The training data set consisted of WAV files, each containing a short utterance. Prior to training, the WAV files were verified with respect to recording duration, sampling rate, RMS level, peak amplitude, silence ratio, and clipping, defined as saturation of the signal at the limits of the representable amplitude range. The main properties of the resulting training data set are summarized in Table I.

The voice model was trained in the Piper environment on a workstation equipped with an NVIDIA GeForce RTX 3060 Ti. After training, the model was exported to ONNX format, and a JSON configuration file was also used for execution. During synthesis, the input text was first converted into a phonemic representation using eSpeak NG and then passed to the neural Piper model, which generated the audio signal.

The benchmark was performed for seven test sentences of different lengths. Each benchmark sentence was synthesized on each hardware configuration. A single warm-up run, understood as a preliminary synthesis run not included in the final measurements, was performed before the measured repetitions. These three synthesis execution variants were compared: Windows with CPUExecutionProvider, Windows with CUDAExecutionProvider, and Raspberry Pi 3B+ as a resource-constrained platform. On the Windows computer, the model was loaded once and kept in memory, whereas for the Raspberry Pi, the total process time and the synthesis time after model initialization were distinguished.