Voice Builder: A Tool for Building Text-To-Speech Voices

Pasindu De Silva[2], Theeraphol Wattanavekin[1], Tang Hao[1], Knot Pipatsrisawat[1]
[1]Google,[2]Google (on contract from Teledirect Pte Ltd)
{pasindu,twattanavekin,tanghao,thammaknot}@google.com

Abstract

We describe an opensource text-to-speech (TTS) voice building tool that focuses on simplicity, flexibility, and collaboration. Our tool allows anyone with basic computer skills to run voice training experiments and listen to the resulting synthesized voice. We hope that this tool will reduce the barrier for creating new voices and accelerate TTS research, by making experimentation faster and interdisciplinary collaboration easier. We believe that our tool can help improve TTS research, especially for low-resourced languages, where more experimentations are often needed to get the most out of the limited language resources.

Keywords: text-to-speech, voice building, opensource

1. Introduction

There is growing demand for text-to-speech (TTS) voices in various industries as the world moves towards technologies that can interact with users more naturally. Many industries including gaming, film, appliances, and accessibility (Szarkowska, 2011) now have vested interest in creating TTS voices. As a result, there is a lot of needs to produce a variety of good quality voices in different languages. The TTS problem is challenging especially for low-resourced languages, because of the lack of data. Smaller text and audio corpora imply that more experiments might need to be done in order to get the most out of the available resources. Tackling this problem for low-resourced languages usually requires close collaboration between computer scientists, who understand the algorithms and know how to utilize the tools, and linguists, who understand linguistic phenomena and can help provide linguistic resources required to improve the voices.

From our experience building voices for low-resourced languages, we have observed many pain points that prevent these researchers from conducting quick experiments and iterating on their existing baseline voices. These pain points include (i) the complexity of setting up initial voice building infrastructure, which often requires fairly strong technical background to setup and maintain (ii) the heterogeneous nature of different tools, which requires a non-trivial amount of effort to *stitch* the required pieces together to form a working pipeline for TTS research.

In this paper, we propose *Voice Builder*, a text-to-speech voice building tool, which comes with a simple web-based user-friendly interface that enables researchers to quickly build and listen to prototype voices and to iteratively improve them with ease.

2. Related Work

There are many opensource TTS systems available with underlying algorithms ranging from classification regression trees in Festival (Taylor et al., 1998; Black and Taylor, 1997), hidden Markov models in HTS (Zen et al., 2007), to neural networks in Merlin (Wu et al., 2016). Some of these tools depend on components of their predecessors. For example, Merlin requires HTS or Festival to handle the linguistic frontend processing. Some systems, such as MaryTTS (Schröder et al., 2011), try to provide more end-to-end solutions for building various components, such as recording and transcription, required for a complete TTS system. Other tools focus only on certain parts of the TTS pipeline. For example, FreeTTS (W. Walker, 2009) allows users to synthesize audio from TTS models produced by other tools, while Sparrowhawk (Google, 2018a; Ebden and Sproat, 2015) is an opensource text-normalization system.

3. Voice Builder and the Problems It Addresses

Each of the tools mentioned above has its own strengths based on the approach it uses. When experimenting with new data for a language, we often wish for a quick way to build initial voices from different approaches in order to establish a good baseline. In such cases, the quality of the voice can be quickly assessed even without all the bells and whistles that would be needed in a production system. However, connecting the aforementioned tools together to form a working pipeline and/or setting up multiple algorithms for TTS research on the same platform is a daunting task. Users often end up spending too much time setting up and maintaining end-to-end system, rather than focusing on the real research.

Voice Builder is a TTS voice building tool that was designed to allow users to quickly build and listen to initial voices. It consists of a web frontend that allows users to easily build and listen to synthesized voices, regardless of their technical ability. Voice Builder by itself is not a TTS engine, but, rather, allows arbitrary engines (such as Festival, Merlin, MaryTTS, etc) to be plugged in and used for training voice models. Voice Builder will serve as the user interface and workflow manager that connects all the necessary steps together. Once setup, users need to fill out a simple web form and click a button to initiate a voice model training process. Voice Builder focuses on only the following 3 necessary pieces of language resources to start training a voice model: audio-transcription pairs, lexicon, and phonology. Once the job finishes, users can visit Voice Builder UI to listen to audio synthesized from arbitrary

text that the users can input. Figure 1 gives a high-level overview of Voice Builder.

3.1. Making Voice Building More Accessible

Building a new TTS voice has always been a fairly technical task. There is a lot of data processing to be done and the TTS tools available often require no-nonsense technical knowledge to harness. This technical barrier makes it difficult for the general public or language enthusiasts to contribute. After all, we cannot expect computer scientists to speak all languages, especially the long-tailed ones. Therefore, collaboration with native speakers or linguists is required in order to build voices for these languages. Voice Builder addresses this problem by providing a dead simple user interface (UI) to allow anyone with just basic computer knowledge to initiate a voice model training experiment. Figure 2 shows a portion of Voice Builder's graphical user interface for starting model training. As long as the necessary data is provided in a web form, the users can start voice building by just clicking a button; no further skill is needed. Then the user just needs to wait for the job to finish. Once model training is finished, the user can come back to the same UI to enter some text and listen to the synthesized voice. This simple process makes it easier for anyone to get involved in building voices. Moreover, Voice Builder's web UI allows collaborators across the world to work on the same project. By listening to the resulting voice, a non-technical native speaker might already be able to provide valuable feedbacks to more technical team members to further improve the voice.

3.2. Easier Experimentation

When working on TTS for low-resourced languages, one often needs to run many experiments on different data sets and to try on different training parameters. When dealing with many (often parallel) experiments, it is important to keep track of the settings and environment of each experiment for later diagnosis. Handling this manually is a potentially error-prone process; there could be many different files to keep track of. Voice Builder solves this problem by managing and isolating each experiment into its own sandbox. The configuration and data used to initiate each run is stored in a location dedicated to that experiment. These data are immutable for later analysis. Moreover, different TTS engines (or even different versions of the same engine) can be modularly added to the system, making it easy to trace back to earlier environments, if needed.

3.3. Support for Multiple Algorithms

The ability to experiment with many voice building algorithms is an important part of achieving high quality TTS voices for low-resourced languages. As a result, Voice Builder was designed with this in mind. Users will be able to easily plug in additional voice training engines into our system. We utilize Docker (Merkel, 2014) containers to encapsulate these engines. Therefore, integrating an additional engine only amounts to creating a Docker image and a configuration file.

3.4. Zero-ops Cloud Services

Creating a web-based service typically requires a substantial effort to manage and maintain. This is not the case for Voice Builder, because we chose to utilize Google Cloud Platform services for our various components. The web frontend is hosted by Google Compute Engine (Google, 2018f). All training data and configuration for each experiment are stored in the designated location on Google Cloud Storage (Google, 2018e). TTS engines and synthesizers are hosted in containers via Docker images. Lastly, we use Google Cloud Functions (Google, 2018c) to connect different stages of the pipeline together. This means that the users will get the security and reliability benefits without having to maintain any server themselves.

3.5. Custom Pre-processing

Different research groups may have different formats for various data files, depending on the tools used to obtain the data. Voice Builder was designed with this use case in mind and allows a custom data pre-processor to be used. When the user initiates a voice training experiment in the UI, a *create voice* request is sent to a (customizable) API endpoint to initiate pre-processing. The system expects the output processed data to be put in the designated job folder on Google Cloud Storage. Once all the required files are present in the folder, the system will automatically trigger the rest of the pipeline.

4. System Architecture

In this section, we explain the underlying architecture in more details. Figure 3 shows the system diagram of Voice Builder. We will now explain the components in this system from the perspectives of model training and voice synthesis.

4.1. Voice Model Training

When a user starts a voice model training process, they go through the following flow:

1. User enters information about locations of training data on the web frontend. The necessarily language resources are (i) audio-transcription pairs (ii) lexicon (iii) phonology definition. In the frontend, the user can also select the TTS engine to use and adjust the training parameters. A training specification containing all the parameters is created. The frontend also displays the status of each training job.

2. The training configuration is sent over to an optional pre-processing unit. This unit is responsible for processing and outputting all the necessary files to the cloud storage location specified in the configuration. If this unit is not present, all training data is gathered directly into the cloud storage location by the frontend.

3. Once all the required files are present, a cloud function will automatically trigger the training process by spawning a TTS engine in a container. The engine spawned is determined by the parameters in the configuration.

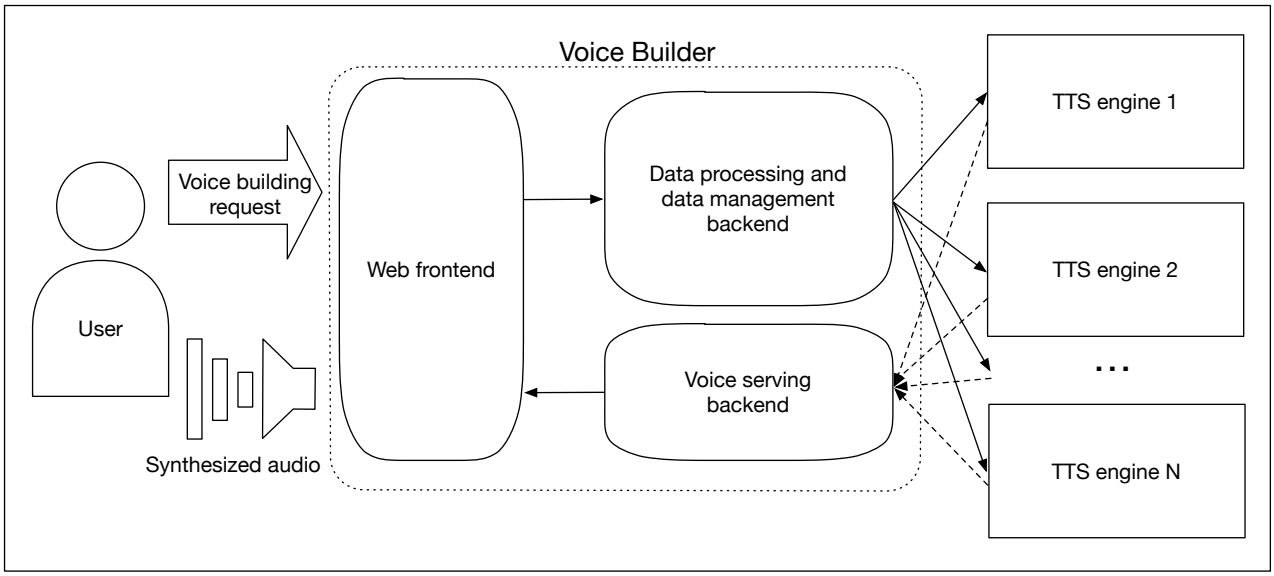

Figure 1: An overview diagram of Voice Builder.

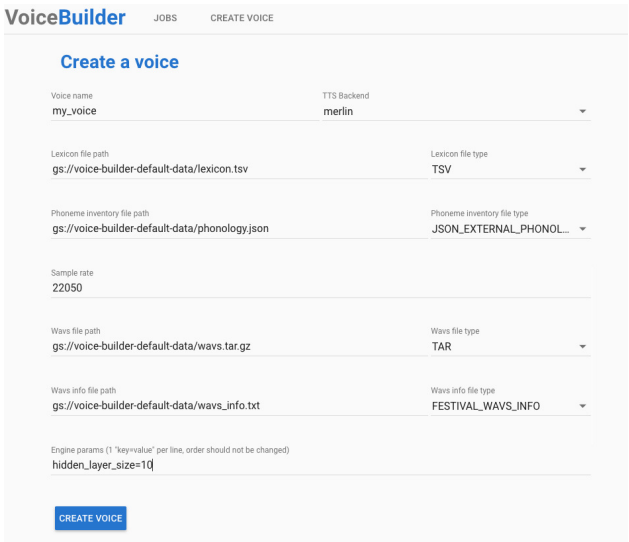

Figure 2: A screenshot of Voice Builder's graphical user interface for training a new voice.

4. The specified training engine is created in a container and initiates training based on the given specifications. Once the process finishes, the output model is saved at the designated cloud storage location.

5. Once the model is present, a cloud function will trigger a process to deploy a voice synthesis server using a container. This server is now ready to serve voice synthesis requests.

4.2. Voice Synthesis

When a user wants to listen to the voice synthesized by a trained model, they go through the following flow:

1. The user visits the frontend page that corresponds to the finished training job. On that page, the user can enter arbitrary text to be synthesized.

2. The request is sent to the voice synthesis server associated with that job.

3. Audio is synthesized and returned to be played on the frontend.

Notice that we utilize many Google Cloud Services that are publicly available in our system. The reasons for using cloud services, as opposed to building own servers, are as follow:

- **Encapsulation of different training engines.** Container technology allows us to be flexible and extensible in our support of various model training engines. It enables us to easily wrap the engines and integrate them with other parts of our system. Google Cloud Platform (GCP) (Google, 2018d) has various container solutions, two of which are used by our system:

    1. *Genomics Pipeline* (Google, 2018g) is used for voice model training. It provides an easy way to run and monitor any command. Input and output files are copied from and to Google Cloud Storage and are fully managed by the pipeline. It also has a simple template to help users plug in any model training engine easily.

    2. *Containers on Compute Engine* (Google, 2018b) is used to serve our frontend as well as the synthesis servers. Containers make our components platform-agnostic, which means that they can be run anywhere. Users can download a model to run locally on their machines or send to others to run using the same container image. In most cases, the same container images used for model training can also be used here. These characteristics make our system more portable, easier to deploy and maintain at scale.

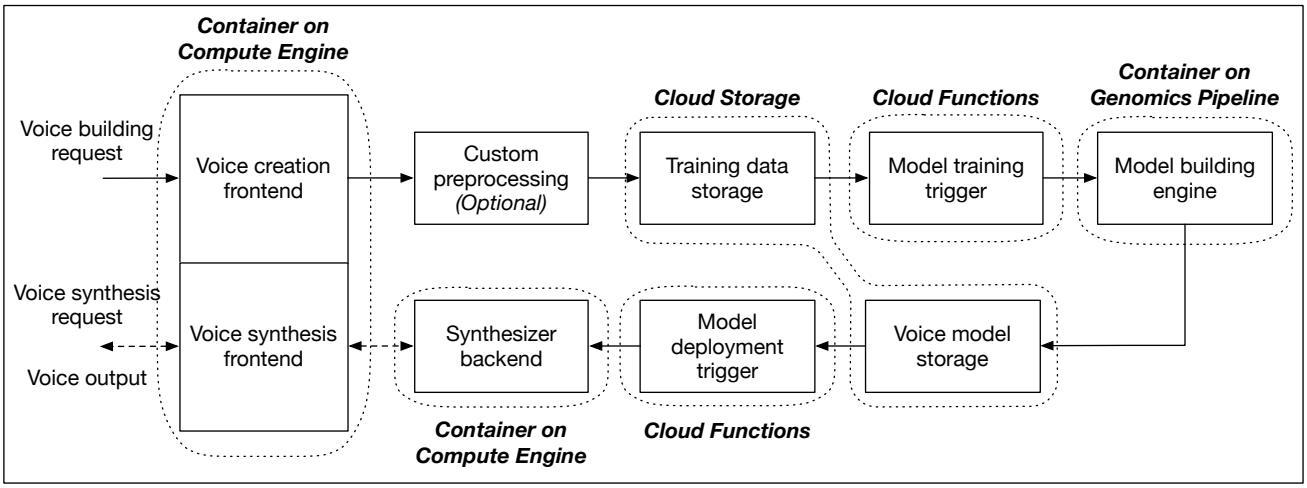

Figure 3: System diagram of Voice Builder. Each block corresponds to a system component. Each dotted bubble indicates a cloud service used for the enclosed component. Voice model training flow is indicated by solid arrows, while voice synthesis flow is indicated by dashed arrows.

- **Simpler connective layer.** Creating a business logic pipeline that stitches all components together is not a trivial task. Google Cloud Platform provides smooth integration between different products via Cloud Functions. Instead of managing our own event messages, Cloud Functions give us a powerful native triggering mechanism. For example, Cloud Functions can respond to change notification from Google Cloud Storage by automatically deploying a voice model after all the required files have been placed in the designated location on Google Cloud Storage.
- **Public availability and ease of use.** All GCP products in our system are available for the public. Setting up our system requires virtually no other additional software, making the installation process easy.

These cloud services help make our system flexible, reliable, and easy to set up.

5. Available Resources

We have opensourced the Voice Builder source code under the Apache 2.0 license. The code is available at https://github.com/google/voice-builder with setup instructions and some examples. In this release, we provide integrations with Festival and Merlin as a starting point for users.

6. Future Work

In the future, we would like to make Voice Builder even more useful for researchers to evaluate voices and perform diagnosis. Our hope is to make evaluation and diagnosis tools available right in the web frontend. For example, the frontend could allow users to synthesize a batch of sentences from a pair of models and present them for side-by-side listening test. For diagnosis, we hope to be able to provide basic alignment information used during training so that linguists could identify problematic data easily.

7. Conclusions

In this paper, we present a tool for creating TTS voices. The goals of this work is to make creating new TTS voices simpler, more flexible, and accessible by everyone. We designed the system to be user-friendly by providing an easy-to-use web frontend. Voice Builder allows for easy integration with publicly available TTS engines. Moreover, we utilize cloud services to run our system, thus, making our tool globally accessible to collaborators without requiring them to maintain their own servers. We believe that this tool could help move TTS research forward, especially for low-resourced languages, by facilitating experimentation and encouraging collaboration among researchers, no matter where they are in the world.

8. Bibliographical References

Black, A. W. and Taylor, P. A. (1997). The Festival Speech Synthesis System: System Documentation. Technical Report HCRC/TR-83, Human Communciation Research Centre, University of Edinburgh, Scotland, UK. Available at http://www.cstr.ed.ac.uk/projects/festival.html.

Ebden, P. and Sproat, R. (2015). The kestrel tts text normalization system. *Natural Language Engineering*, 21(3):333–353.

Google. (2018a). https://github.com/google/sparrowhawk. Accessed: 2018-02-19.

Google. (2018b). Containers on compute engine. https://cloud.google.com/compute/docs/containers/. Accessed: 2018-02-12.

Google. (2018c). Google cloud functions. https://cloud.google.com/functions/. Accessed: 2018-02-12.

Google. (2018d). Google cloud platform. https://cloud.google.com/. Accessed: 2018-02-12.

Google. (2018e). Google cloud storage. https://cloud.google.com/storage/. Accessed: 2018-02-12.

Google. (2018f). Google compute engine. https://cloud.google.com/compute/. Accessed: 2018-02-12.

Google. (2018g). Google genomics pipeline. https://cloud.google.com/genomics/reference/rest/v1alpha2/pipelines. Accessed: 2018-02-12.

Merkel, D. (2014). Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.*, 2014(239), March.

Schröder, M., Oliva, M. C., Pammi, S., and Steiner, I. (2011). Open source voice creation toolkit for the MARY TTS Platform. In *Proceedings of Interspeech 2011*. ISCA, August.

Szarkowska, A. (2011). Text-to-speech Audio Description: Towards Wider Availability of AD. 15:142–163, January.

Taylor, P. A., Black, A., and Caley, R. (1998). The Architecture of the Festival Speech Synthesis System. In *The Third ESCA Workshop in Speech Synthesis*, pages 147–151, Jenolan Caves, Australia.

W. Walker, P. Lamere, P. K. (2009). A speech synthesizer written entirely in the Java(TM) programming language. https://freetts.sourceforge.io/, March. Accessed: 2018-02-12.

Wu, Z., Watts, O., and King, S., (2016). *Merlin: An Open Source Neural Network Speech Synthesis System*, pages 218–223. ISCA, September.

Zen, H., Nose, T., Yamagishi, J., Sako, S., Masuko, T., Black, A. W., and Tokuda, K. (2007). The HMM-based speech synthesis system (HTS) version 2.0. In *SSW*.