Beyond RoboShaul: zero-shot speech synthesis in Hebrew

Daniel Volkov **Shahar Glam**

Abstract

We present our implementation of *'HebTTS'*, a Hebrew TTS model architecture that would work on *'native'* diacritic-free Hebrew text, with zero-shot speech synthesis capabilities.
Traditionally Hebrew uses diacritics (niqqud) to instruct readers on how to pronounce words. However, nowadays most Hebrew text is written without diacritics and thus leaves ambiguity on pronunciation. Our model uses a diacritic free approach by using a phoneme based text tokenizer. This allows us to utilize unsupervised *'real'* data.
In addition to the inherent challenges of the Hebrew language we had the added challenge of having limited resources, which challenged us to adapt the data and training routines to fit our circumstances. Our results show relative success on seen speakers uttering full sentences. On unseen speakers we achieved limited success, succeeding in uttering some words but yet full sentences. This suggests that perhaps a larger dataset and more training would allow the model to generalize better, taking into consideration our training metrics.

1 Introduction

Hebrew is a low-resource language, and as such, it presents unique challenges that constrain research and development of speech products and models. In particular, Hebrew speech and text often don't align in a one-to-one manner, due to the (lack of) presence of diacritics (niqqud). The niqqud traditionally instructs the reader how to pronounce the Hebrew text, but in modern Hebrew it is mostly omitted, and the pronunciation of text is inferred from context and prior knowledge of Hebrew.
It is thus even harder to train text-to-speech (TTS) systems for Hebrew, since they often rely heavily on phoneme-based text tokenization, which is not feasible in diacritic-free Hebrew text, which is the standard of Hebrew today.
Additionally, as a low-resource language, Hebrew suffers from a lack of public data-sets – previous attempts in creating Hebrew speech data-sets and benchmarks were quite small.
Recent advancements in the field suggest impressive capabilities of speech sythesis models which rely on discrete representations as intermediate representations (i.e. audio tokens). These audio tokens are obtained using a Residual Vector Quantization (RVQ) method, and are then processed by a Language Model for prediction of output audio tokens. The output tokens are then decoded into the desired utterance. We tackle the problem of creating a zero-shot speech synthesis model, which uses diacritic free Hebrew.

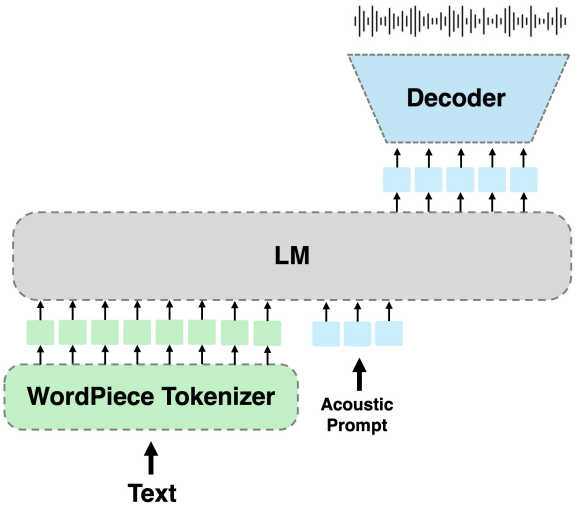

Figure 1: A high-level overview of the the proposed method. The text is first being tokenized using a word-piece tokenizer. Then an audio language model predicts a discrete sequence of audio tokens which later on will be decoded into raw waveform.

2 Related work

2.1 Audio tokenization

Since audio is usually stored as an array of `int16` encoded numbers, with a sampling frequency that

often exceeds 16Khz, treating each sample as a token is infeasible for the transformer architecture. Thus, we use neural-codec based Residual Vector Quantization methods for audio tokenization. Specifically, we use the EnCodec model presented by (Défossez et al. , 2022), which is trained in an adversarial manner to create low bandwidth discrete audio representations. The EnCodec model encodes a given audio to 8 codebooks, with the first codebook being the most *important*, due to the residual nature of the quantizer.

2.2 Previous attempts at Hebrew TTS

Previous works had either used text with diacritics or used a trained models to add diacritics to diacritics-free text. The original HebTTS model (Roth et al., 2024) achieved better results than the preveious attempts.
We will use the original HebTTS model as our baseline.

3 Method

Theoretic framework. We use a pre-trained neural codec model to encode audio utterances into discrete representation (i.e. tokens). For a given utterance y, denote it's discrete representation (after encoding) as $\text{Enc}(y) = C \in \mathbb{R}^{T \times N_{CB}}$, where T is the encoded utterance length, and N_{CB} is the number of codebooks used for encoding. The encodec decoder can then decode the quantized audio, which we denote as $\text{Dec}(C) \approx y$, where the approximation holds due to the pre-training of the model we use. The TTS objective is then commonly formalized as: given a dataset $\mathcal{D} = \{x_i, y_i\}$, where y_i are utterances and x_i are their corresponding phonetic transcriptions, choose model parameters $\theta^* = \arg\max_\theta \mathbb{P}(y_i|x_i, \tilde{y}_i; \theta)$, where $\tilde{y}_i$ is a voice enrollment for y_i, from the same speaker.

3.1 Model

Text tokens. As stated earlier, the common approach uses the phonetic transcriptions of the target utterance as the text tokens. In Hebrew, this phonetic decomposition proves much more difficult, due to missing diacritics (niqqud). We thus use a different text tokenizer; a word-piece tokenizer. We use a pre-trained tokenizer, namely – the tokenizer of the AlephBert model (Seker et al. , 2021), which was trained on 98.7M Hebrew sentences.
The tokenizer is very similar to classic text tokenizers, starting with all characters as tokens, and merging tokens iteratively to minimize the amount of tokens that would appear in the training corpus; but instead of merging the most appearing pair of tokens, it merges the pair of tokens with the highest score, defined as:

$$\text{score}(w_1, w_2) = \frac{\text{freq}(w_1, w_2)}{\text{freq}(w_1) \cdot \text{freq}(w_2)}$$

Dividing by the frequencies of the individual parts prioritizes merging where the individual parts are less common on their own.
Language modeling approach. We use a similar architecture to the *Vall-E* model (Wang et al. , 2023). As noted before, the intermediate representations we work on are discrete representations generated by the neural codec. We can thus model the probability as $\mathbb{P}(y_i|x_i, \tilde{y}_i; \theta) = \mathbb{P}(C|x_i, \tilde{C}; \theta)$, where $C = \text{Enc}(y_i)$ and $\tilde{C} = \text{Enc}(\tilde{y}_i)$, only focusing on the discrete representations of the input and output utterances. We take advantage of the transformer architecture to generate discrete audio representations; for the first output codebook $C_{:,1}$, we use an auto-regressive (AR) decoder-only model. It is conditioned on text tokens x_i and the discrete representation of voice enrollment $\tilde{C}$, and is used to generate the first codebook using an iterative process.
The other $N_{CB} - 1$ codebooks are generated by a non-auto-regressive transformer decoder. For all $j \in \{2, \ldots, N_{CB}\}$, the NAR decoder is used for generating the j'th codebook $C_{:,j}$, conditioned on the text tokens x_i, the enrollment $\tilde{C}$, the previous target codebooks $C_{:,<j}$, and the codebook index j. Overall, the prediction of C is modeled as:

$$\mathbb{P}(C \mid x_i, \tilde{C}; \theta) = \mathbb{P}(C_{:,1} \mid x_i, \tilde{C}_{:,1}; \theta_{\text{AR}}) \cdot \prod_{j=2}^{8} \mathbb{P}(C_{:,j} \mid x_i, C_{:,<j}, \tilde{C}, j; \theta_{\text{NAR}})$$

This hybrid approach provides the model with a good balance between expressive power and computational efficiency: The AR decoder model allows the model to decide the utterance length, and from the residual nature of the audio quantizer we use, the first codebook is also the one which most influences the output utterance – responsible for prosody, rhythm, and intonation. The NAR decoder then predicts the remaining codebooks using one forward pass each. This approach results in $T + N_{CB}$ transformer forward passes, which is quite efficient in practice, and allows the model to

capture complex speech – since the first codebook has the highest influence on the output utterance, and is generated in an AR manner.

Data. We use the HebDB[1] dataset for training. This dataset consists of $\sim$ 2500 hours of speech, collected from local podcasts. This dataset is comprised of spontaneous dialogues, featuring multiple speakers discussing a wide range of topics including economy, politics, sports, culture, science, history, and music, among others. This data was originally composed of full-length episodes, but has already been pre-processed by (Turetzky et al. , 2024), using voice activity detection, segmenting the lengthy audio to short phrases, and then transcribing using Whisper-V2 Large, which shows great Hebrew ASR capabilities. This effort proved very useful for us, but we still had to pre-process the data, in order to obtain speaker-id annotations. These annotations were used for sampling $(y_i, \tilde{y}_i)$ pairs from the same speaker (Alongside the transcription x_i). To achieve this, we tried two different approaches:

- The first method we tried was clustering speaker embeddings; we tried to leverage the publicly available Resemblyzer (Wan et al. , 2020) implementation[2] to extract speaker embeddings from the samples we had, then clustering those embedding vectors and labeling each cluster as a different speaker. This approach yielded $\sim$ 400 hours of fully-annotated data, which we then tried training the model on. Sadly, this approach yielded bad results, resulting in *"garbage"* generation. This is likely due to this dataset being very *"in-the-wild"*, containing samples that have multiple speakers, music, and general noise. All these factors might have hurt the performance of the speaker embedding extractor, yielding embeddings that don't correspond to the speaker's identity.

- Our second approach leverages the fact that most recordings are single-speaker; We first filter out utterances shorter than some base threshold (We use 6 seconds), then split each utterance into two new samples; the first three seconds, and the rest. We then use a publicly available Hebrew force-alignment model[3] to partition the transcription of the utterance between the two new samples. This method yielded $\sim$ 700 hours of high quality speaker-id and transcription annotations. This method is the method we used for training.

3.2 Experiments and implementation details

We adapt the publicly available code[4] by (Roth et al., 2024), which is itself an adaptation of the publicly available VALL-E implementation[5], by Li Feinteng. As mentioned above, we use the HebDB dataset for all of our experiments and training. We train the `valle_alephbert_concat` model, which uses the AlephBert embedding layer, alongside a learned embedding layer, allowing the model more generalization capabilities. We train the AR decoder for 800K steps, and the NAR decoder for 200K steps, followed by a warm restart run on the AR decoder for 150K steps. We use the Eden scheduler, inspired by (Roth et al., 2024).

Due to limited compute capabilities, we train on only one NVIDIA TITAN XP GPU, using a batch size of $\sim$ 2500 acoustic tokens. We use gradient accumulation (of 10 batches) to increase our effective batch size, to match the existing literature on vall-e-style model training.

We use a base learning rate of 0.05, with Eden's `lr_batches` $=$ 5000 and `lr_epochs` $=$ 6. We train the AR decoder for 15 epochs (+3 at the warm restart run, and 4 epochs on the NAR decoder), with $\sim 55,000$ steps each epoch.

Mistakes. This project is our first time training an LLM, we thus faced many failed attempts and *'dumb'* mistakes (or *'fun experiments'*). For example, we started training with really small batches ($\sim$ 1000 acoustic tokens), and without any gradient accumulation, we also trained with a very (!) low learning rate ($\sim 10^{-4}$) for our first few experiments, etc. The entire training *'journey'* appears on our Weights&Biases project page[6], with some explanations in the `reports` section. Our code is available on our GitHub repository[7]. Our final trained checkpoint is available on HuggingFace[8]. Some generation samples are available on our site[9].

[1] https://huggingface.co/datasets/SLPRL-HUJI/HebDB

[2] https://github.com/resemble-ai/Resemblyzer

[3] https://github.com/imvladikon/wav2vec2-hebrew

[4] https://github.com/slp-rl/HebTTS

[5] https://github.com/lifeiteng/vall-e

[6] https://wandb.ai/the_magnivim/HebrewTTS%20recreation/reportlist

[7] https://github.com/D4niel0s/HebTTS

[8] https://huggingface.co/D4niel0s/HebTTS_implementation

[9] https://dasvolkov9.wixsite.com/my-site-4

	Objective Metric	
Model	**WER**	**CER**
Reference (seen speaker)	0.19	0.08
Reference (unseen speaker)	0.19	0.08
Ours (seen speaker)	1.02	0.65
Ours (unseen speaker)	1.43	0.72

Table 1: Evaluation metrics of our model, against the baseline HebTTS model

3.3 Evaluation metrics

To evaluate our model, we use the Word Error Rate (WER) and Character Error Rate (CER) metrics. The WER measures the model's ability to match the desired output on a word level, while the CER measures the character level (mis)match. Both score the output based on it's edit distance from the ground truth output, where the edit distance is measured with respect to words or characters, respectively.
We evaluated out model on a random set of 500 sentences generated by chat-GPT. For each speaker in our speakers set (some seen in training and others not) we iterated through each sentence and generated the matching audio for the target text using our model. We then transcribed the resulting speech using *Whisper-v2-large*, normalized both target and transcribed texts by removing punctuations, etc. and calculated WER, CER.

4 Results & Discussion

We have obviously achieved worse results than the original HebTTS model. We believe this is mainly due to our limited resources. Additionally we notice that relatively, our results on unseen speakers are worse than those on seen speakers, this further supports our belief that the insufficient results are due to insufficient training rather than a bug or an architecture mistake. As of writing these lines our model validation loss is still decreasing!

Note . We successfully completed the training phase of our model, but were unable to perform the planned evaluation before the project deadline due to unexpected process interruptions. Above are the metrics from the partial evaluation we did manage to perform.

5 Future work

Although the architecture we trained does achieve impressive results, it is far from a universal solution – changing languages means re-training the model and potentially changing the architecture, as in the case of Hebrew. A future effort could be made to make a *multi-lingual* speech synthesis model, which would have zero-shot synthesis capabilities across different (and hopefully all) languages. To achieve this goal without expanding the datasets in low-resource languages, we think that an effort should be invested in text tokenizers which exploit linguistic phenomena. For example – if one were able to break a given input string into the corresponding set of phonemes in all languages, we think a model such as the one we trained could generalize to be multi-lingual.
Another goal is avoiding impersonation or *spoofing* – we think that it is possible to create a watermarking scheme (such as the ones deployed in LLMs today) for speech synthesis models – which would alleviate some of the privacy concerns.
Another research direction (which already achieved impresssive results[10]) would be leveraging acoustic tokens for other tasks (i.e. conversational models, ASR, etc.). The transformer architecture has shown it's strength over the years and it's impressive results in various fields, we thus think it could be adapted into a lot of audio-related tasks, and maybe make some *'universal model'*, similarly to the prevalence of GPTs in the field of NLP.

6 Limitations & Broader impact

Speech synthesis models have recently become prolific in literature and in the *main-stream* market. This rise in popularity was caused by the increased capabilities of neural speech-synthesis models and by recent advances in research in the field. However, these impressive capabilities come with significant ethical and security concerns: zero-shot synthesis models can clone a speaker's voice from only a few seconds of audio, enabling highly convincing impersonations without explicit consent.
The ability to recreate one's voice exposes individuals to privacy violations, fraud, and reputa-

[10] https://www.sesame.com/research/crossing_the_uncanny_valley_of_voice#demo

tional harm, as adversaries can "steal" voices to bypass authentication, defame public figures, or craft targeted deepfake scams. As these models become more accessible through open-source libraries and cloud APIs, the societal risk of audio deepfakes—ranging from financial fraud to political disinformation—grows exponentially, challenging regulators and developers to establish consent frameworks, watermarking standards, and robust detection tools to safeguard personal and public interests.

References

Amit Roth, Arnon Turetzky, Yossi Adi. *A Language Modeling Approach to Diacritic-Free Hebrew TTS*. 2024.

Chengyi Wang, Sanyuan Chen, Yu Wu, Ziqiang Zhang, Long Zhou, Shujie Liu, Zhuo Chen, Yanqing Liu, Huaming Wang, Jinyu Li, Lei He, Sheng Zhao, Furu Wei. *Neural Codec Language Models are Zero-Shot Text to Speech Synthesizers*. 2023.

Amit Seker, Elron Bandel, Dan Bareket, Idan Brusilovsky, Refael Shaked Greenfeld, Reut Tsarfaty. *AlephBERT: A Hebrew Large Pre-Trained Language Model to Start-off your Hebrew NLP Application With*. 2021.

Arnon Turetzky, Or Tal, Yael Segal-Feldman, Yehoshua Dissen, Ella Zeldes, Amit Roth, Eyal Cohen, Yosi Shrem, Bronya R. Chernyak, Olga Seleznova, Joseph Keshet, Yossi Adi. *HEBDB: a Weakly Supervised Dataset for Hebrew Speech Processing*. 2024.

Alexandre Défossez, Jade Copet, Gabriel Synnaeve, Yossi Adi. *High Fidelity Neural Audio Compression*. 2022.

Li Wan, Quan Wang, Alan Papir, Ignacio Lopez Moreno. *Generalized End-to-End Loss for Speaker Verification*. 2020.

A Convergence plots for AR decoder

The first run (in Red) was a very low lr run. We thus ran another run (in Blue). The green run is the warm restart run.

Figure 2: Total loss convergence plots for the AR decoder

Figure 3: Validation loss convergence plots for the AR decoder

B Convergence plots for NAR decoder

The first run (in Red) was a very low lr run. We thus ran another run (in Blue).

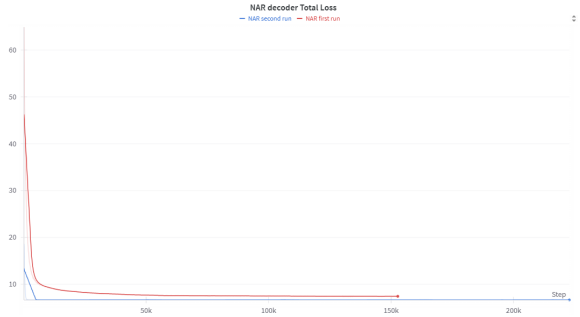

Figure 4: Total loss convergence plots for the NAR decoder

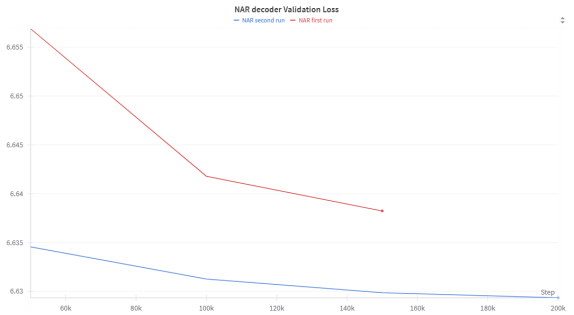

Figure 5: Validation loss convergence plots for the NAR decoder

Note: For more details and interactive plots, see our Weights&Biases project page[11].

[11] https://wandb.ai/the_magnivim/HebrewTTS%20recreation/reportlist