Interspeech 2022
18-22 September 2022, Incheon, Korea

Muskits: an End-to-End Music Processing Toolkit for Singing Voice Synthesis

Jiatong Shi[1], *Shuai Guo*[2], *Tao Qian*[2], *Nan Huo*[3], *Tomoki Hayashi*[4,5], *Yuning Wu*[2], *Frank Xu*[1], *Xuankai Chang*[1], *Huazhe Li*[6], *Peter Wu*[7], *Shinji Watanabe*[1], *Qin Jin*[2†]

[1]Carnegie Mellon University, [2]Renmin University of China, [3]The University of Hong Kong, [4]Human Dataware Lab. Co., Ltd., [5]Nagoya University, [6]Tsinghua University, [7]University of California, Berkeley

jiatongs@cs.cmu.edu, {shuaiguo, qiantao, qjin}@ruc.edu.cn, shinjiw@ieee.org

Abstract

This paper introduces a new open-source platform named ***Muskits*** for end-to-end music processing, which mainly focuses on end-to-end singing voice synthesis (E2E-SVS). Muskits supports state-of-the-art SVS models, including RNN SVS, transformer SVS, and XiaoiceSing. The design of Muskits follows the style of widely-used speech processing toolkits, ESPnet and Kaldi, for data prepossessing, training, and recipe pipelines. To the best of our knowledge, this toolkit is the first platform that allows a fair and highly-reproducible comparison between several published works in SVS. In addition, we also demonstrate several advanced usages based on the toolkit functionalities, including multilingual training and transfer learning. This paper describes the major framework of Muskits, its functionalities, and experimental results in single-singer, multi-singer, multilingual, and transfer learning scenarios. The toolkit is publicly available at `https://github.com/SJTMusicTeam/Muskits`.

Index Terms: Singing voice synthesis, end-to-end, open-source toolkit

1. Introduction

Singing voice synthesis (SVS) uses music score and lyrics to generate natural singing voices of a target singer. Recently, SVS has attracted much research attention in the speech and music processing communities. However, due to high data annotation costs and strict musical copyright policies, there are limited public databases available for SVS. The shortage of high-quality data further leads to limited benchmarks in SVS, compared to various open-source activities in other generative tasks such as text-to-speech (TTS) and voice conversion (VC) [1–4].

This paper introduces a new open-source toolkit called ***Muskits*** that aims to provide a neural network-based end-to-end platform for music processing, specifically SVS. Unlike existing open-source SVS repositories [5–11], Muskits provides different architectures for SVS in an end-to-end manner and offers a wide-range comparison between different architectures for singing voice synthesis. Muskits inherits the base framework from ESPnet [2, 12] and follows ESPnet styles in data processing scripts and training recipes, demonstrating a complete setup for SVS. The contributions of this paper include: (1) a new platform for reproducible SVS modeling; (2) benchmarks on several public databases in both single- and multi-singer scenarios; (3) exploration on multilingual training for SVS; (4) studies on transfer learning on a multi-style low-resource database.

†Corresponding Author.

2. Related Works

The initial effort on SVS starts with unit-selection models, where pre-recorded singing segments are concatenated to form the singing voice [13, 14]. However, because of the need for large corpora, this method is not flexible to achieve reasonable sound in low-resource scenarios. Later, statistical parametric methods (e.g., the hidden Markov model (HMM)), were proposed for singing synthesis [5, 15]. They were shown to flexibly synthesize the singing voice, but also led to loss of naturalness. Deep neural networks (DNNs) have become dominant across several generation tasks in the recent decade. SVS is not an exception. Initially, several works applied basic structures, including fully-connected layers, convolutional neural networks (CNNs), and recurrent neural networks (RNNs) [16–19], which demonstrated reasonable quality and naturalness for singing generation. Some more advanced architectures, including generative adversarial networks (GANs), encoder-decoder, and diffusion-based models, also showed some quality improvements over specific databases [20–26].

As SVS is receiving more and more attention, there has been an increasing need for open-source platforms so as to allow fair comparison between the proposed models. Compared to similar tasks such as TTS, SVS has faced several issues in sharing benchmarks and their platforms. Different from speech, singing voice databases usually have more strict usage guidelines and copyright concerns. Meanwhile, singing voices require a professional high-quality recording environment and generally take longer to record than that for regular speech. Resources are further limited due to the need for detailed annotation (e.g., music score information).

Even with those limitations, there are still some relevant efforts in releasing databases and code bases for some SVS models. Sinsy [5, 19], known as one of the initial open-source works in SVS, was released for research purposes. The toolkit applies the HMM-based architecture and can reach reasonable performances with low-resource singing voice corpus (e.g., 1 hour). It also demonstrates its extendability to multiple languages, including Japanese, Chinese, and English. NN-SVS [6] is a DNN-based singing voice synthesis toolkit, which follows the cascaded neural networks in [19]. Seven Japanese recipes are supported in the toolkit. We also observe several open-source efforts related to specific models, including [7–11]. They mostly include the whole training procedure and some also release the related databases [8, 11].

The aforementioned open-source efforts have greatly benefited the whole research community. However, it is difficult to provide a fair comparison of different models due to the differences in datasets, feature representations, and vocoders. In this work, our goal is to fill the gap with a new open-source toolkit,

Copyright © 2022 ISCA

10.21437/Interspeech.2022-10039

Muskits. With 12 recipes on 10 SVS databases, the toolkit has supported several choices of end-to-end SVS models with various ESPnet compatible vocoders[1]. Furthermore, we explore multilingual training, pre-training, and transfer learning on a multi-style databases using Muskits, which have not been well investigated in SVS before.

3. Functionality

Muskits consists of two major parts: a Python library of several neural network models and recipes for running complete experiments on specific databases. The library uses PyTorch for neural networks, with standard music representation, various model architectures, and a uniform training/decoding framework. The recipes provide detailed examples in all-in-one style scripts, following the style in Kaldi and ESPnet [2, 12, 27].

3.1. Music score representation

Muskits performs all pre-processing steps on the fly, including acoustic feature extraction and two standard interfaces for music score information. The two interfaces are at the frame-level and syllable-level. The frame-level features need pre-aligned linguistic units (e.g., phoneme or syllable) and music note sequences. Based on timestamps, both sequences are first expanded to the sampling rate of a given audio (e.g., 24k). Then, they are aggregated with a sliding window that has the same window size and hop length to the target acoustic features (e.g., log-Mel filter-bank features). The same or similar features are used in [8, 10, 17–19]. The syllable-level features are an extended version of XiaoiceSing feature [22], including phoneme ids, music note ids[2], tempo/beats information, and corresponding duration information. Depending on the models and application scenarios, either frame-level or syllable-level features could be selected. Note that we always use syllable-level features for models that consider duration modeling (e.g., various sequence-to-sequence models) [10, 20, 22, 28].

3.2. Models

Muskits supports three models, including a RNN-based model [10, 17], transformer-based model [20], and XiaoiceSing [22]. All models are based on the encoder-decoder architecture and support both frame and syllable-level representations discussed in Sec 3.1. The output of models is a sequence of acoustic features (e.g., Mel-filter bank features).

Similar to [10, 22, 28], in the RNN-based model, music notes and phoneme information are first passed to individual RNN encoders. Then, the combination of hidden states (either by concatenating or adding) is fed into another RNN decoder. If syllable-level representations are used for the RNN-based model, an additional duration predictor is introduced to expand the syllable-level hidden states to the frame level. The duration predictor is trained separately to predict the explicit duration of each hidden state. In training, the encoder-decoder model adopts ground truth duration. While in inference, the encoder output will be expanded by the predicted duration, and then fed into the decoder. The transformer-based model follows the architecture of [20]. It first encodes the phoneme sequence by stacked gated linear unit (GLU) layers. Then, it expands the hidden representation of phonemes into frame-level with a rule-based duration model. After adding the pitch embedding and positional encoding, the hidden states are then fed into a transformer-based decoder for final outputs. The duration of phoneme sequences is defined using rules [20]. XiaoiceSing adopts a FastSpeech-like architecture [22, 29]. The model sums over the syllable feature representations and adopts an encoder-decoder framework for singing voice generation.

3.3. Training, inference, and evaluation

The training procedures in Muskits are handled with a unified task processor adapted from ESPnet, which supports multi-GPU training and dynamic batch making. On-the-fly data augmentation is supported, including pitch augmentation and mixup-augmentation [30].

In addition to the default Griffin-Lim vocoder, Muskits also supports various neural vocoders, including ParallelWaveGAN, MelGAN, and HiFiGAN [31–35]. The vocoder can be applied to both training and inference stages. In training, the vocoder could automatically generate audio from selected samples of development set for monitoring the training progress. During inference, compatible vocoders can be used directly without additionally dumping intermediate features.

We implement three objective evaluation metrics for SVS in Muskits: Mel-cepstral distortion (MCD), voiced/unvoiced error rate (VUV_E), and logarithmic rooted mean square error of the fundamental frequency (F_0RMSE). All the metrics are calculated by dynamic time-warping to consider the length mismatch between synthesized singing and ground truth singing.

3.4. Recipe flow

We provide 12 all-in-one recipes for 10 singing voice databases. All recipes follow a unified pipeline with explicit stages of preparation, training, inference, and evaluation. The stages are defined in the template `svs.sh`, as follows:

Stage 1*: Database-dependent data preparation*. The expected data generally follows Kaldi style but with additional `midi.scp` and `label` for music score and phoneme alignment information. For databases without explicit midi or MusicXML, we also provide rule-based automatic music transcription to extract related music information. Relevant functions can be found in KiSing recipe [36].

Stage 2*: Standard audio and midi formatting*. This stage formats audio into a standard format by resampling, segmentation, and dumping from pipe-style input. MIDI is also normalized and segmented at this stage. Singer ID and Language ID are also defined if needed.

Stage 3*: Filtering*. Phrases in the training and development sets are filtered by a pre-defined length threshold.

Stage 4*: Token list generation*. Muskits collects tokens from the training set and generates a corresponding token list for training.

Stage 5*: Statistics collection*. The input information of training and development sets are collected for efficient batching.

Stage 6*: Model training*. The model is optimized based on the training objectives.

Stage 7*: Model inference*. If no vocoder is provided, Muskits adopts the Griffin-Lim vocoder. Otherwise, compatible vocoders can be loaded for inference. The predicted acoustic features are also stored for external vocoders.

Stage 8*: Objective evaluation*. Objective evaluation is conducted between paired reference data and inference outputs.

Stage 9*: Model packing*. The model is packed with all the necessary elements for inferences. The packed models of our recipes are also shared with the recipes.

Most public SVS databases only have a single singer, which limits the scalability of the SVS systems built upon them. By

[1] `https://github.com/kan-bayashi/ParallelWaveGAN`

[2] The music note ids are the same as MIDI 0-128 note ids.

Table 1: *Details of experimental data: N_{src}, N_{lang}, N_{singer} stands for number of databases, languages, and singers, respectively.*

Task	N_{src}, N_{lang}, N_{singer}	Duration(h)		
		Train	Dev	Test
Single-singer	1,1,1	0.63	0.08	0.07
Multi-singer	4,1,4	3.01	0.34	0.38
Multilingual	6,4,6	11.72	0.49	1.01
Transfer	1,1,1	0.71	0.02	0.05

supporting several recipes with a standardized training interface, Muskits could easily extend to different genders, multi-singer, multilingual scenarios. To be specific, we already support two recipes on multi-singer and multilingual cases.

4. Experiments

4.1. Experimental datasets

To demonstrate the performance of Muskits, we conduct four sets of experiments under single-singer, multi-singer, multilingual, and transfer learning scenarios. For single-singer experiments, we use the Ofuton-P database [37] that contains 56 Japanese songs (61 minutes) of a male singer. For multi-singer experiments, we use a combination of four databases, including Ofuton-P, Oniku, Natsume, and Kiritan [37–40]. For multilingual training, we adopt the four databases mentioned above, CSD databases [41], and Opencpop [11], which results in a ten-hour corpus in Japanese, English, Korean, and Mandarin. For transfer learning, we adopt the KiSing corpus, a one-hour female Mandarin corpus with multiple styles [36].

Except for Opencpop and KiSing, other databases do not have official splits of train, development, and test sets, so we apply our own split in our reproducible recipes. All the songs are segmented based on the silence between lyrics. The data sizes for each experiment are listed in Table 1. For CSD, only syllables are provided for the alignment, but the training set is not enough to cover all the syllable combinations. Therefore, we split the syllable into phonemes and evenly assign duration information to each phoneme given the duration of syllables.

4.2. Experimental setups

The experiments are in two folds based on whether the model uses ground truth (G.T.) duration in order to factorize the evaluation of duration prediction and acoustic modeling.

The RNN-based model adopts three-layer 512-dimension bi-directional long short time memory (LSTM) encoder and a five-layer 1024-dimension decoder. The transformer-based model follows the original setting in [20]. It consists of three 256-channel GLU layers with a kernel size of three as the encoder and a six-layer four-head-256-dimension transformer decoder. Local Gaussian constraints are injected into the self-attention in its decoder. The XiaoiceSing adopts the same network hyper-parameters as in [22], including an encoder and a decoder with the same setting (i.e., six transformer blocks, while each block has four-head-384-dimension self-attention block and 1536 dimensional feed-forward layer). The decoder outputs of all models are projected to the dimension of acoustic features (i.e., 80) with a postnet inherited from Tacotron2 [42]. For settings with duration prediction, the transformer-based model adopts a rule-based duration modeling as defined in [20], while the others adopt two 384-channel convolutional layers with a kernel size of three to model duration information.

For multi-singer and multilingual training, we provide the option to apply additional embeddings from one-hot singer IDs and language IDs. The embedding size is the same as the dimension of the encoder outputs for all models. In the forward process, embeddings are added directly to the hidden states of encoders. As discussed in Sec 3.3, the acoustic models can be easily used with compatible vocoders. We used the HiFi-GAN vocoder for all experiments. The vocoders are trained on the same training set from the Muskits' split introduced in Sec 4.1.

During training, the RNN-based model uses Adam optimizer with a 0.001 learning rate without learning rate schedulers to keep the same as its original paper [10]. All the other models employ Adam optimizer with a NoamLR scheduler. The warm-up steps are set to 4,000, and the peak learning rate is set to 1.0. We employ gradient clipping at the threshold of 1.0. All experiments are conducted on a single GPU and the batch size is set to 8. We apply a 0.1 dropout rate to encoder/decoder and a 0.5 dropout rate to postnets. We set the training epochs as 500 for models trained on single-singer databases and 250 for models trained on multi-singer or multilingual data.

As in Sec 3.3, we adopt MCD, V/UV Error, and F_0RMSE for objective evaluation. For each subjective evaluation conducted online, we invite 25 subjects, including both musicians and non-professionals. The same set of 15 phrases are randomly selected for each system. Subjects are asked to score the phrases in one (unintelligible) to five (excellent quality). We report our results with 95% confidence intervals. We also conduct the A/B test in the transfer learning scenario. The same 25 subjects are asked to compare two samples of the same phrase from a pair of models. 20 phrases are selected from the KiSing test set.

4.3. Results and discussion

Single singer: Table 2 presents the results of the single-singer setting. Among models using G.T. duration, the RNN-based model achieves the best performance over the subjective MOS metric; the Transformer-based model achieves the best MCD; XiaoiceSing achieves the best VUV_E and F_0RMSE. For models without G.T. duration, the RNN-based model achieves the best performance on both objective and subjective metrics. We observe that there is a huge gap between models with and without G.T. duration. The primary reason could be the limited size of the Ofuton-P database (37.8 minutes of singing for training), which causes the difficulty of learning reasonable duration predictors. Moreover, the results in the subjective evaluation suggest that the RNN-based model is better than the other two, and the Transformer-based model is better than XiaoiceSing, in our single-singer scenario. This may indicate that simpler structures are more suitable for the task with low data-resource.

Multiple singer: Table 2 presents the experiment results of the multi-singer experiments setting. Even though multi-singer corpus has larger data size compared to single-singer, it may encounter additional issues like multiple styles of singing and singer variation. For example, we find singing voices in Oniku database mostly use falsetto, while the other databases use mostly modal voice. As in Table 2, evaluation metrics on multi-singer may not be necessarily better than single-singer's. However, as we compare the MOS between single and multi-singer scenarios[3], we observe that the RNN-based model has some degradation on MOS, while the other two have some improvements. It might indicate that the transformer-based model and XiaoiceSing have a better modeling capacity and need a certain amount of data to converge. The RNN-based model, on the other hand, can efficiently converge on small data, but has the

[3]Other metrics are not suitable for comparison, given that the test data is different.

Table 2: *: Single- and multi-singer experiments: The experimental databases are Ofuton-P database for single-singer experiments and a combination of four databases as described in Sec 4.1; the models are discussed in Sec 4.2; the metrics are defined in Sec 3.3.*

Model	G.T. Dur.	Single-Singer MCD↓	VUV_E↓	F_0RMSE↓	MOS↑	Multi-Singer MCD↓	VUV_E↓	F_0RMSE↓	MOS↑
RNN [10]	✓	6.52	2.30	0.105	**3.47 ± 0.11**	7.15	3.87	0.195	3.25 ± 0.13
Transformer [20]	✓	**6.46**	2.67	0.118	3.23 ± 0.11	**6.68**	3.90	**0.184**	**3.40 ± 0.12**
XiaoiceSing [22]	✓	6.88	**2.22**	**0.103**	3.11 ± 0.11	6.81	**3.86**	0.190	3.23 ± 0.12
RNN [10]	/	**6.19**	**2.18**	**0.109**	**3.43 ± 0.11**	7.05	**3.55**	0.226	2.84 ± 0.12
Transformer [20]	/	6.95	5.61	0.172	2.39 ± 0.11	6.87	3.70	0.186	**3.33 ± 0.12**
XiaoiceSing [22]	/	7.19	2.68	0.120	1.96 ± 0.08	**6.79**	3.85	**0.154**	2.83 ± 0.11
G.T.	/	/	/	/	4.89 ± 0.05	/	/	/	4.65 ± 0.10

Table 3: *Multilingual experiments on Ofuton-P. All the models use the same XiaoiceSing architecture. The pre-trained models are conducted over the multilingual data described in Sec. 4.1.*

Model	MCD↓	VUV_E↓	F_0RMSE↓	MOS↑
Baseline	6.88	**2.22**	**0.103**	3.05 ± 0.12
Pre-train	**6.17**	2.51	0.113	2.99 ± 0.12
+ LangID	6.32	2.94	0.123	2.90 ± 0.12
Fine-tune	6.23	2.75	0.106	**3.49 ± 0.12**
+ LangID	6.34	2.55	0.110	3.09 ± 0.12
G.T.	/	/	/	4.77 ± 0.07

capacity issue when extending to multi-singer scenarios.

Multilingual singing: We conduct five experiments on multilingual SVS and evaluate the performances on Ofuton-P database. The baseline is the XiaoiceSing without duration prediction presented in Table 2. The pre-train models are trained on the multilingual data introduced in Sec 4.1 with the same XiaoiceSing architecture. As in Sec 4.2, we examine models with or without language IDs. Based on models trained on multilingual data, we also fine-tune the multilingual models on Ofuton-P database for another 500 epochs. As in Table 3, the multilingual models clearly demonstrate a large gain over the MCD objective scores, but do not improve the V/UV error rate and F_0RMSE. In subjective evaluation, directly using multilingual models does not bring improvements on MOS. However, pre-trained models could be useful when applying fine-tuning on the same database. Meanwhile, given that the model with language IDs always getting worse objective and subjective scores, it might indicate that additional language IDs could not improve the singing quality. One potential reason is that language IDs may restrict the model to transfer knowledge from multilingual data especially when the data size is in low-resource.

Transfer learning: In this experiment, we explore to use transfer learning on KiSing, a low-resource multi-style database. Although KiSing has 42.6 minutes of singing for training, it has content duplication in most songs, resulting in limited variances in terms of phoneme sequences and melodies. Furthermore, the singer has adopted both a Chinese folk style (a variant of the Peking opera) and a pop style in songs, which has more frequent changes in rhythm, harmony, and the usage of falsetto. We conducted three experiments on this database: Baseline is trained only on KiSing. Two pre-trained models Multilingual$*$ and Opencpop$*$ are first pre-trained on multilingual data and Opencpop data, respectively. And then, they are fine-tuned on KiSing for another 500 epochs. As discussed in Sec. 4.1, Opencpop is 5.2-hour Mandarin pop singing voice database, which has the same language coverage with KiSing. As shown

Table 4: *Objective evaluation for transfer learning experiments on KiSing (a low-resource multi-style singing database). All the models use the same XiaoiceSing architecture. Models with $*$ are first pre-trained on the corresponding database and then fine-tuned on KiSing.*

Model	MCD↓	VUV_E↓	F_0RMSE↓
Baseline	8.74	3.74	0.214
Opencpop$*$	8.52	**3.54**	0.190
Multilingual$*$	**8.25**	3.66	**0.177**

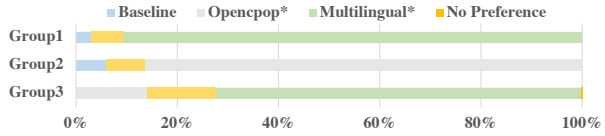

Figure 1: *A/B test for transfer learning experiments on KiSing.*

in Table 4, Multilingual$*$ achieves the best objective scores of MCD and F_0RMSE, while Opencpop$*$ achieves the best VUV_E. For the subjective evaluation, we observe that both pre-trained models outperform the Baseline, while Multilingual$*$ is better than Opencpop$*$. The result indicates that larger data size and a larger coverage of phonemes could benefit the low-resource multi-style SVS.

5. Conclusions

In this work, we introduce a new E2E-SVS toolkit, called Muskits. This toolkit is developed as an open platform which can support fair comparison of different SVS models, and we expect it to facilitate the research field of SVS. The toolkit is based on the ESPnet framework with 12 recipes covering over 10 public databases in 4 languages. We conduct experiments under four scenarios: single-singer, multi-singer, multilingual SVS, and transfer learning on a multi-style database. In single- and multi-singer scenarios, we compare three published works on SVS with the same music score representation and vocoder. In addition, we explore multilingual SVS by combining six databases in four languages. From multilingual pre-training, we demonstrate subjective and objective gain on a single-singer database, which could alleviate the low-resource issue in SVS. Further experiments on transfer learning also show the effectiveness of our framework on a multi-style low-resource database. Full documentation and real-time demos can be found at `https://github.com/SJTMusicTeam/Muskits`.

6. Acknowledgement

This work was supported by the National Natural Science Foundation of China (No. 62072462) and the National Key R&D Program of China (No. 2020AAA0108600).

7. References

[1] M. Schröder, M. Charfuelan, S. Pammi, and I. Steiner, "Open source voice creation toolkit for the MARY TTS Platform," in *Interspeech*, 2011, pp. 3253–3256.

[2] T. Hayashi, R. Yamamoto, T. Yoshimura *et al.*, "ESPnet2-TTS: Extending the Edge of TTS Research," *arXiv preprint arXiv:2110.07840*, 2021.

[3] Mozilla, "Text-to-speech for all," 1 2021. [Online]. Available: https://github.com/mozilla/TTS

[4] C. AI, "Coqui TTS," 3 2022. [Online]. Available: https://github.com/coqui-ai/TTS

[5] K. Oura, A. Mase, T. Yamada *et al.*, "Recent development of the HMM-based singing voice synthesis system—Sinsy," in *Seventh ISCA Workshop on Speech Synthesis*, 2010.

[6] R. Yamamoto, "Neural network-based singing voice synthesis library for research," 3 2022. [Online]. Available: https://github.com/r9y9/nnsvs

[7] P. Chandna, M. Blaauw, J. Bonada *et al.*, "WGANSing: A Multi-Voice Singing Voice Synthesizer Based on the Wasserstein-GAN," in *EUSIPCO*, 2019, pp. 1–5.

[8] J. Tae, H. Kim, and Y. Lee, "MLP Singer: Towards Rapid Parallel Korean Singing Voice Synthesis," in *2021 IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2021, pp. 1–6.

[9] S. Choi, W. Kim, S. Park *et al.*, "Korean singing voice synthesis based on auto-regressive boundary equilibrium gan," in *ICASSP*, 2020, pp. 7234–7238.

[10] J. Shi, S. Guo, N. Huo *et al.*, "Sequence-to-sequence singing voice synthesis with perceptual entropy loss," in *ICASSP*, 2021, pp. 76–80.

[11] Y. Wang, X. Wang, P. Zhu *et al.*, "Opencpop: A High-Quality Open Source Chinese Popular Song Corpus for Singing Voice Synthesis," *arXiv preprint arXiv:2201.07429*, 2022.

[12] S. Watanabe, T. Hori, S. Karita *et al.*, "ESPnet: End-to-End Speech Processing Toolkit," *Interspeech*, pp. 2207–2211, 2018.

[13] H. Kenmochi and H. Ohshita, "VOCALOID-commercial singing synthesizer based on sample concatenation." in *Interspeech*, vol. 2007, 2007, pp. 4009–4010.

[14] J. Bonada, M. Umbert Morist, and M. Blaauw, "Expressive Singing Synthesis Based on Unit Selection for the Singing Synthesis Challenge 2016," in *Interspeech*, 2016, pp. 1230–1234.

[15] K. Saino, H. Zen, Y. Nankaku *et al.*, "An HMM-based singing voice synthesis system," in *SLT*, 2006.

[16] M. Nishimura, K. Hashimoto, K. Oura *et al.*, "Singing Voice Synthesis Based on Deep Neural Networks." in *Interspeech*, 2016, pp. 2478–2482.

[17] J. Kim, H. Choi, J. Park *et al.*, "Korean Singing Voice Synthesis System based on an LSTM Recurrent Neural Network," in *Interspeech*, 2018, pp. 1551–1555.

[18] K. Nakamura, K. Hashimoto *et al.*, "Singing voice synthesis based on convolutional neural networks," *arXiv preprint arXiv:1904.06868*, 2019.

[19] Y. Hono, S. Murata, K. Nakamura *et al.*, "Recent development of the DNN-based singing voice synthesis system—sinsy," in *APSIPA*, 2018, pp. 1003–1009.

[20] M. Blaauw and J. Bonada, "Sequence-to-sequence singing synthesis using the feed-forward transformer," in *ICASSP*, 2020, pp. 7229–7233.

[21] Y. Gu, X. Yin, Y. Rao *et al.*, "ByteSing: A Chinese Singing Voice Synthesis System Using Duration Allocated Encoder-Decoder Acoustic Models and WaveRNN Vocoders," in *ISCSLP*, 2021, pp. 1–5.

[22] P. Lu, J. Wu, J. Luan *et al.*, "XiaoiceSing: A High-Quality and Integrated Singing Voice Synthesis System," *Interspeech*, pp. 1306–1310, 2020.

[23] Y. Hono, K. Hashimoto, K. Oura *et al.*, "Singing Voice Synthesis based on Generative Adversarial Networks," in *ICASSP*, 2019, pp. 6955–6959.

[24] J. Lee, H.-S. Choi, C.-B. Jeon *et al.*, "Adversarially Trained End-to-End Korean Singing Voice Synthesis System," *Interspeech*, pp. 2588–2592, 2019.

[25] J. Liu, C. Li, Y. Ren *et al.*, "Diffsinger: Singing voice synthesis via shallow diffusion mechanism," *arXiv preprint arXiv:2105.02446*, 2021.

[26] Y. Zhang, J. Cong, H. Xue *et al.*, "VISinger: Variational Inference with Adversarial Learning for End-to-End Singing Voice Synthesis," *arXiv preprint arXiv:2110.08813*, 2021.

[27] D. Povey, A. Ghoshal, G. Boulianne *et al.*, "The Kaldi speech recognition toolkit," in *ASRU*, 2011.

[28] J. Chen, X. Tan, J. Luan *et al.*, "HiFiSinger: Towards High-Fidelity Neural Singing Voice Synthesis," *arXiv preprint arXiv:2009.01776*, 2020.

[29] Y. Ren, C. Hu, X. Tan *et al.*, "FastSpeech 2: Fast and High-Quality End-to-End Text to Speech," in *ICLR*, 2020.

[30] S. Guo, J. Shi, T. Qian *et al.*, "Singaug: Data augmentation for singing voice synthesis with cycle-consistent training strategy," *arXiv preprint arXiv:2203.17001*, 2022.

[31] R. Yamamoto, E. Song, and J.-M. Kim, "Parallel WaveGAN: A fast waveform generation model based on generative adversarial networks with multi-resolution spectrogram," in *ICASSP*, 2020, pp. 6199–6203.

[32] K. Kumar, R. Kumar, and o. de Boissiere, "MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis," *NIPS*, vol. 32, 2019.

[33] G. Yang, S. Yang, K. Liu *et al.*, "Multi-band MelGAN: Faster waveform generation for high-quality text-to-speech," in *SLT*, 2021, pp. 492–498.

[34] A. Mustafa, N. Pia, and G. Fuchs, "Stylemelgan: An efficient high-fidelity adversarial vocoder with temporal adaptive normalization," in *ICASSP*, 2021, pp. 6034–6038.

[35] J. Kong, J. Kim, and J. Bae, "HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis," *NIPS*, vol. 33, pp. 17 022–17 033, 2020.

[36] J. Shi, K. Zhang, and Z. Yao, "KiSing," http://shijt.site/index.php/2021/05/16/kising-the-first-open-source-mandarin-singing-voice-synthesis-corpus/, accessed: 2020.10.15.

[37] P. Futon, "DB Production: Futon P," https://sites.google.com/view/oftn-utagoedb/%E3%83%9B%E3%83%BC%E3%83%A0, accessed: 2022.03.10.

[38] K. Otanmiya, "Otanmiya Kurumi Singing Voice Database," http://onikuru.info/db-download/, accessed: 2022.03.10.

[39] Amanokei and K. Sota, "Yuri Natsume Singing Voice Database," https://amanokei.hatenablog.com/entry/2020/04/30/230003, accessed: 2022.03.10.

[40] I. Ogawa and M. Morise, "Tohoku Kiritan singing database: A singing database for statistical parametric singing synthesis using Japanese pop songs," *Acoustical Science and Technology*, vol. 42, no. 3, pp. 140–145, 2021.

[41] S. Choi, W. Kim, S. Park *et al.*, "Children's song dataset for singing voice research," in *ISMIR*, 2020.

[42] J. Shen, R. Pang, R. J. Weiss *et al.*, "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions," in *ICASSP*, 2018, pp. 4779–4783.