Generating Vocals from Lyrics and Musical Accompaniment

Georg Streich
ETH Zurich
streichg@ethz.ch

Luca A. Lanzendörfer
ETH Zurich
lanzendoerfer@ethz.ch

Florian Grötschla
ETH Zurich
fgroetschla@ethz.ch

Roger Wattenhofer
ETH Zurich
wattenhofer@ethz.ch

Abstract

In this work, we introduce AutoSing, a novel framework designed to generate diverse and high-quality singing voices from provided lyrics and musical accompaniment. AutoSing extends an existing semantic token-based text-to-speech approach by incorporating musical accompaniment as an additional conditioning input. This enables AutoSing to synchronize its vocal output with the rhythm and melodic nuances of the accompaniment while adhering to the provided lyrics. Our contributions include a novel training scheme for autoregressive audio models applied to singing voice synthesis, as well as ablation studies to identify the best way to condition generation on musical accompaniment. We measure AutoSing's performance with subjective listening tests, demonstrating its capability to generate coherent and creative singing voices. Furthermore, we open-source our codebase to foster further research in the field of singing voice synthesis.

1 Introduction

Advances in generative audio generation have significantly improved the ability to synthesize natural-sounding speech, vocals, and music. These developments have enabled various companies to launch products that can generate full-length music tracks containing high-quality vocals from text descriptions and lyrics (1; 2; 3; 4). Although the field of singing voice synthesis (SVS) has improved greatly in recent years (5; 6; 7; 8), the proposed methods often still fall short of the diversity and sophistication exhibited by current black box commercial products.

In this work, we introduce AutoSing, a novel framework for synthesizing diverse and high-quality singing voices from provided lyrics and instrumentals. AutoSing extends WhisperSpeech (9), an existing autoregressive text-to-speech library, with conditioning on an accompanying instrumental track. Since the neural audio code used to represent speech in WhisperSpeech does not perform well on singing voices, we evaluate codecs trained on music and find that the Multi-Scale Neural Audio Codec (SNAC) (10) performs especially well considering its low bandwidth of 1.9kbps. In a further step, we experimented with training strategies that allow us to efficiently utilize SNAC's hierarchical representation. As a result, we propose a novel multiscale autoregressive training approach.

Our contributions can be summarized as follows:

- We propose a novel training scheme for autoregressive audio models and apply it to SVS. We identify scenarios where our approach improves training efficiency. In all cases, the proposed training scheme results in substantially reduced memory requirements.

Neural Information Processing Systems (NeurIPS) 2024 Workshop on AI-Driven Speech, Music, and Sound Generation.

- We narrow the gap between public research on SVS and commercial capabilities. Moreover, we open-source our codebase[1] and release a number of tools to enable further research in the domain of SVS.

Audio samples can be found online.[2]

2 Related Work

There have been various approaches to generate natural-sounding singing voices. DiffSinger (5), NaturalSpeech2 (6), MakeSinger (8) and RMSSinger (7) are diffusion-based singing voice and speech synthesis tools, which allow users to generate singing voices conditioned on music notes rather than an instrumental track.

Jukebox (11) was an early music generation model which allowed conditioning the generation on lyrics, but required multi-hour generation times (11). More recently, music generation models have been proposed that are capable of generating diverse and high-quality songs, but do not support lyrics conditioning (12; 13). Singsong (14) conditions music generation on vocal stems, addressing a complementary problem to ours. Several commercial black box products offer music generation with vocal capabilities (1; 2; 3; 4).

Discrete Audio Representations. Modeling audio as discrete codes using a Vector Quantized Variational Autoencoder (VQVAE) (15), followed by training a generative model on these codes, was introduced by (11). Subsequently, several 'neural audio codecs' (16; 17) were introduced. More recently, the Multi-Scale Neural Audio Codec (SNAC) (10) was proposed to more effectively capture and represent the varying timescales present in audio signals. Rather than encoding audio as a regular grid of codes, SNAC's quantization scheme results in a hierarchical codebook structure (cf. Fig. 2).

Semantic Tokens and Two Stage Text-to-Speech. Our vocal generation approach builds on established autoregressive, semantic token-based text-to-speech methods (18; 9). Semantic tokens serve as a discrete audio representation, analogous to the tokens of neural audio codecs. Unlike neural audio codecs, their primary goal is not to compress audio, but to encode higher-level semantic content. A number of methods have been proposed for obtaining such a 'semantic' representation. For example, (19) employs iteratively refined k-means clustering on the hidden states of a BERT-like transformer. WhisperSpeech (9) discretizes the encoder outputs of a Whisper (20) model.

Building on this concept, the authors of (21) proposed a two-stage modeling approach for audio continuation. In a first stage, semantic tokens are generated and are used to condition the generation of acoustic tokens from a neural audio codec in a second stage. They found that this produced realistic sounding speech, whereas direct unconditioned generation of acoustic tokens resulted in nonsensical output. Subsequently, (18) adapted the approach to text-to-speech generation by conditioning the semantic token generation on text.

3 Proposed Method

In the following, we present the architecture of AutoSing, a model for lyrics-conditioned vocal generation given musical accompaniment. Our approach incorporates a low-bandwidth neural audio codec to enable training and generation of 30-second music snippets.

3.1 Text and Music Conditioned Vocal Generation

Given the lyrics and an existing instrumental stem, our model is designed to generate a vocal track that recites the lyrics and matches the provided instrumental track. We explored both generating combined vocal and instrumental output, and only generating the vocal stem and combining it with the instrumental track in a post-processing step. We found that exclusively modeling the vocal track allowed us to utilize a lower bandwidth codec. Additionally, generating the vocals might be easier for the model to learn than having to output the complete track. However, generating the vocal stem

[1] `https://github.com/streichgeorg/autosing`

[2] `https://streichgeorg.github.io/autosing_samples`

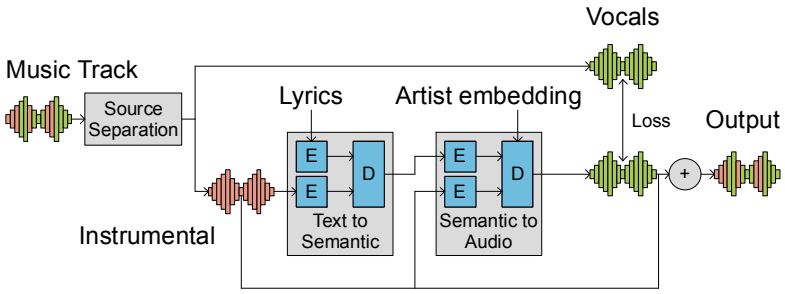

Figure 1: Overview of our proposed text-to-vocal architecture. The pipeline entails two stages (text-to-semantic, semantic-to-audio) which are both implemented using a multi-encoder transformer architecture. The text-to-semantic stage takes lyrics and the instrumental track as inputs via its encoders (E), the semantic output of its decoder (D) is then fed to the semantic-to-audio stage which also takes the instrumental track as an input. Lyrics are tokenized using byte encoding, audio signals using SNAC. To train the text-to-semantic stage, the ground truth semantic representation is extracted from the vocal track. For the semantic-to-audio stage, ground truth vocals are used as a target.

independently may come at the cost of reduced fidelity due to the source separation and remixing processes.

We build on the existing open-source text-to-speech library WhisperSpeech (9) which follows the two-stage text-to-speech approach described in Section 2. WhisperSpeech uses encoder-decoder transformer models for the text-to-semantic and semantic-to-audio stages. We use the multi-encoder approach from (22) to augment both stages with music conditioning. Specifically, we add a second cross-attention layer to each decoder block and combine the two cross-attention signals with a weighted average.

Moreover, we enable conditioning on vocal features by training a custom embedding model (cf. Appendix B).

We evaluated existing low-bandwidth neural codecs on vocal stems and found that at 1.9kbps, the 32kHz version of SNAC (10) provided the best trade-off between bandwidth and audio quality. Additionally, we also encode the accompaniment music using SNAC. We refer to Fig. 1 for an overview of our architecture.

3.2 Multiscale Autoregressive Training

Current neural audio codecs encode audio using hundreds of codes per second (17; 10). Although this is a significant improvement compared to waveform and spectrogram representations, it still imposes large memory and compute requirements. This is especially the case for singing or music in general, given the increased need to model long-range dependencies and the higher bandwidth requirements compared to speech generation. Previous works used sparse attention (11) or multi-stage architectures (21; 18) to handle long sequence lengths.

Recently, MusicGen (12) showed that good results can be achieved with a single dense transformer that predicts codes in parallel. Instead of predicting residual codes for a single frame simultaneously, the authors find that delaying the prediction for fine-grained codes mitigates the performance degradation that would otherwise occur with semi-autoregressive modelling.

To further reduce sequence lengths during training, we propose a multiscale training approach for the acoustic modeling from MusicGen (12) adapted to the SNAC audio codec. We introduce what we call ‘compressed’ and ‘unrolled’ codebook interleaving patterns (cf. Fig. 2). We perform normal autoregressive training, but, show the model samples encoded using either the compressed or unrolled pattern. When shown a compressed sequence, the model needs to predict multiple adjacent codebooks in parallel but has access to a long temporal context. On the other hand, when given an unrolled sequence, the model learns dependencies between fine-grained codes while only having access to a shorter context. At the end of training, we fine-tune the model on full-length unrolled sequences for a few steps. Apart from requiring per sample position information, no changes to the underlying transformer architecture are required for our approach.

4 Experiments and Results

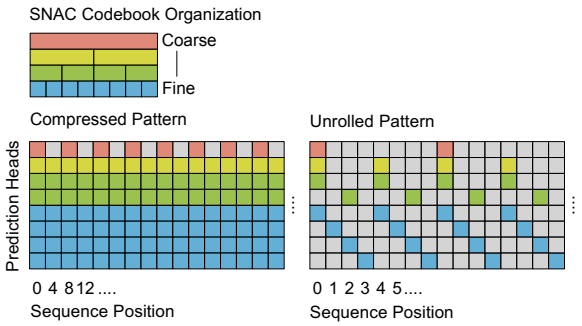

Figure 2: Top: Depiction of SNAC's codebook structure, where fine-grained codebooks have an increased frame-rate. The frame rate for coarse codebooks is 10Hz. Bottom: 'Compressed' and 'unrolled' patterns before applying codebook delays. We train the model on compressed and unrolled sequences simultaneously. Compressed sequences allow for efficiently learning on a long context, while unrolled sequences enable the model to learn dependencies between fine-grained codes.

For our experiments, we use an internal dataset consisting of 35k hours of music, which, after extracting sections that contain singing, amounted to 20k hours of training data. Furthermore, we added 10k hours of speech data from LibriLight (23) to our training set. We use the Mel-Band Roformer model from (24) to separate music into vocal and instrumental stems. WhisperSpeech leverages Whisper to extract a semantic representation from audio (cf. Section 2). Because the original Whisper model-family and subsequently this semantic encoder have subpar performance on singing, we fine-tuned Whisper on vocal stems. This reduced word error rates from 42% to 23% on our validation set. Using the fine-tuned Whisper model, we then follow the methodology from WhisperSpeech to train a custom semantic encoder which performs well on singing.

We follow WhisperSpeech and use the training and initialization strategy proposed by (25) which allows for hyperparameter transfer across model sizes. For our experiments, we report base learning rates, which are adapted to a specific model configuration through (25). For details, we refer to the source code. We use cosine learning rate schedules with a warm-up phase and during inference, temperature sampling with a temperature coefficient of 0.7 is used for both the text-to-semantic and semantic-to-audio stages. We use 15-second audio segments for ablations and train our large model on 30-second segments.

To evaluate our method, we measure FAD scores (26) using the VGGish (27) model. Additionally, we conduct a user study where we present samples to participants and have them rate the samples on a scale of 1 to 5 in three categories: overall (OVL.), creativity (CRT.) and harmony (HRM.). For the harmony category, participants were asked to rate how well the vocal performance complemented the musical accompaniment. In general, participants were asked not to consider the quality of the musical accompaniment. We further investigate the effect that masking on the input has in **??**.

4.1 Performance of Multiscale Training

We derive an estimate for the reduction of floating-point operations per training step yielded by our multiscale training approach. We follow (28) and approximate computation to be linear in the number of parameters and sequence length. We keep the ratio between encoder and decoder depths constant at $1/3$ and use the same number of attention heads for both the encoders and decoder. Let N be the number of parameters of the decoder. For the number of operations per step C, it follows that $C \sim N(\frac{1}{3}L_S + \frac{1}{3}L_M + L_A)$, where L_S, L_M, L_A are the sequence lengths of semantic music conditioning and the acoustic token output. In our case, with 15-second segments, we have $L_S = 375$, $L_M = 320$ and $L_A = 320$ for 'compressed' or 'cropped unrolled', and $L_A = 1280$ for full-length unrolled length sequences. Hence, our approach results in a $2.74\times$ reduction in computation per step. This approximation does not take into account the reduced attention costs of our approach.

We train two 900M parameter models for 11,500 and 4,300 steps, respectively, using both multiscale and standard autoregressive training. Additionally, we fine-tune the model trained with the multiscale approach for 45 steps on full-length unrolled sequences. For the given model size and compute budget, our approach results in a lower perplexity, FAD score and performs better in our subjective listening study (cf. Table 1).

Additionally, we investigate whether training on compressed and unrolled patterns is actually necessary, or if similar results can be obtained by solely training on compressed sequences. We train two 350M parameter models for 6k steps. For the first model we use our multiscale approach as

Table 1: Performance of multiscale training compared to normal autoregressive training. For the given model size and a fixed compute budget, multiscale training results in better performance across all considered metrics.

Training	PPL ↓	FAD ↓	OVL. ↑	CRT. ↑	HRM. ↑
Autoregressive	81.7	3.59	2.9	3.2	2.5
Multiscale	**76.4**	**2.72**	**3.1**	**3.5**	**2.8**

Table 2: Perplexity scores on compressed (PPL A), cropped (PPL B) and full-length sequences (PPL C) for multiscale and compressed sequence only training.

Training	PPL A ↓	PPL B ↓	PPL C ↓
Compressed Only	**123.97**	-	167.33
Multiscale	128.90	132.70	**113.75**

described in Section 3.2. For the second model, we train only on compressed sequences. After the initial training phase, we fine-tune both models on full-length unrolled sequences. In Table 2, we give perplexity scores on 'compressed' and 'unrolled' sequences for the models, and the perplexity on full-length unrolled sequences. When using multiscale training, the resulting model performs well on full-length unrolled sequences, notably it performs better than on either of the patterns it was trained on. Whereas, when only training on compressed sequences, the model does not generalize to full-length unrolled sequences.

4.2 AutoSing Large

We scale our method to a 1B parameter text-to-semantic and a 2.3B parameter semantic-to-audio model, which we train on the combined 30k hours of singing and speech data. We split training across 32 A100 40GB GPUs for both the text-to-semantic and semantic-to-audio stage. When training the text-to-semantic stage, we let the model predict semantic tokens without text conditioning for songs where lyrics are not available. We train the text-to-semantic stage with a batch size of 256 and a base learning rate of $1.5 \cdot 10^{-3}$ for 46k steps split across 32 A100 GPUs. We train the semantic-to-audio stage for 70k steps using our multiscale approach, and then fine-tune it for 100 steps on full-length unrolled sequences. We use a batch size of 512, a base learning rate of $4 \cdot 10^{-3}$, and a 3 to 1 ratio of compressed and unrolled sequences. During fine-tuning, we use a base learning rate of 10^{-4}.

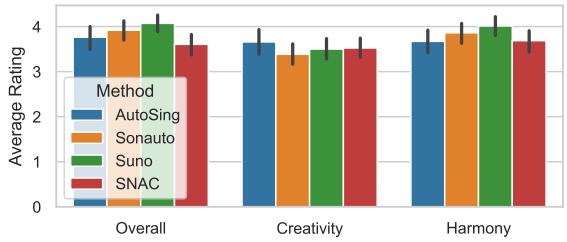

Figure 3: Performance comparison between AutoSing and commercial music generation models. We note that AutoSing displays competitive performance and even outperforms other methods in terms of creativity. For the SNAC baseline, we re-encoded random samples from our dataset using SNAC.

In our listening study, we compare AutoSing with two commercial music models, Suno and Sonauto, along with ground truth examples encoded using SNAC tokens. We present the results in Fig. 3. The results for the SNAC baseline (cf. Fig. 3) reveal a performance bottleneck due to the codec's low bitrate. Therefore, adding a diffusion-based upsampling step, as used in other audio generation systems (29; 9; 30), could be a future step to improve performance.

5 Conclusion

In this work, we introduced AutoSing, an SVS system capable of producing diverse and high-quality vocals using a novel multiscale training scheme for autoregressive audio models. Our approach decouples sequence lengths from the codec's frame rate, improving training efficiency. Our ablation study reveals that conditioning the text-to-semantic stage on source-separated instrumental stems degrades performance, and we propose a strategy to mitigate this. Although, presumably, AutoSing was trained with considerably less data and compute, it still demonstrates strong vocal generation capabilities, comparable to leading commercial music generation models.

Acknowledgment

We would like to thank LAION and JUWELS for providing their high-performance computation infrastructure. Furthermore, we thank Christoph Schuhmann, Jenia Jitsev, and Marianna Nezhurina for their support. The AutoSing codebase is built on `WhisperSpeech`.[3]

References

[1] Suno, "Suno." `https://suno.com/`. Accessed: 2024-08-13.

[2] Udio, "Udio." `https://www.udio.com/`. Accessed: 2024-08-13.

[3] S. Forsgren and H. Martiros, "Riffusion - Stable diffusion for real-time music generation," 2022.

[4] Sonauto, "Home," 2024. Accessed: 2024-08-21.

[5] J. Liu, C. Li, Y. Ren, F. Chen, and Z. Zhao, "Diffsinger: Singing voice synthesis via shallow diffusion mechanism," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 36, pp. 11020–11028, 2022.

[6] K. Shen, Z. Ju, X. Tan, Y. Liu, Y. Leng, L. He, T. Qin, S. Zhao, and J. Bian, "Naturalspeech 2: Latent diffusion models are natural and zero-shot speech and singing synthesizers," *arXiv preprint arXiv:2304.09116*, 2023.

[7] J. He, J. Liu, Z. Ye, R. Huang, C. Cui, H. Liu, and Z. Zhao, "Rmssinger: realistic-music-score based singing voice synthesis," *arXiv preprint arXiv:2305.10686*, 2023.

[8] S. Kim, M. Jeong, H. Lee, M. Kim, B. J. Choi, and N. S. Kim, "Makesinger: A semi-supervised training method for data-efficient singing voice synthesis via classifier-free diffusion guidance," *arXiv preprint arXiv:2406.05965*, 2024.

[9] J. P. Cłapa, "Whisperspeech." `https://github.com/collabora/WhisperSpeech`, 2024. Accessed: 2024-08-12.

[10] H. Siuzdak, F. Grötschla, and L. A. Lanzendörfer, "Snac: Multi-scale neural audio codec," in *Audio Imagination: NeurIPS 2024 Workshop AI-Driven Speech, Music, and Sound Generation*, 2024.

[11] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A generative model for music," *arXiv preprint arXiv:2005.00341*, 2020.

[12] J. Copet, F. Kreuk, I. Gat, T. Remez, D. Kant, G. Synnaeve, Y. Adi, and A. Défossez, "Simple and controllable music generation," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[13] J. Nistal, M. Pasini, C. Aouameur, M. Grachten, and S. Lattner, "Diff-a-riff: Musical accompaniment co-creation via latent diffusion models," *arXiv preprint arXiv:2406.08384*, 2024.

[14] C. Donahue, A. Caillon, A. Roberts, E. Manilow, P. Esling, A. Agostinelli, M. Verzetti, I. Simon, O. Pietquin, N. Zeghidour, *et al.*, "Singsong: Generating musical accompaniments from singing," *arXiv preprint arXiv:2301.12662*, 2023.

[15] A. Van Den Oord, O. Vinyals, *et al.*, "Neural discrete representation learning," *Advances in neural information processing systems*, vol. 30, 2017.

[16] N. Zeghidour, A. Luebs, A. Omran, J. Skoglund, and M. Tagliasacchi, "Soundstream: An end-to-end neural audio codec," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 30, pp. 495–507, 2021.

[17] A. Défossez, J. Copet, G. Synnaeve, and Y. Adi, "High fidelity neural audio compression," *arXiv preprint arXiv:2210.13438*, 2022.

[3] `https://github.com/collabora/WhisperSpeech`

[18] E. Kharitonov, D. Vincent, Z. Borsos, R. Marinier, S. Girgin, O. Pietquin, M. Sharifi, M. Tagliasacchi, and N. Zeghidour, "Speak, read and prompt: High-fidelity text-to-speech with minimal supervision," *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 1703–1718, 2023.

[19] W.-N. Hsu, B. Bolte, Y.-H. H. Tsai, K. Lakhotia, R. Salakhutdinov, and A. Mohamed, "Hubert: Self-supervised speech representation learning by masked prediction of hidden units," *IEEE/ACM transactions on audio, speech, and language processing*, vol. 29, pp. 3451–3460, 2021.

[20] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision. arxiv 2022," *arXiv preprint arXiv:2212.04356*, vol. 10, 2022.

[21] Z. Borsos, R. Marinier, D. Vincent, E. Kharitonov, O. Pietquin, M. Sharifi, D. Roblek, O. Teboul, D. Grangier, M. Tagliasacchi, *et al.*, "Audiolm: a language modeling approach to audio generation," *IEEE/ACM transactions on audio, speech, and language processing*, vol. 31, pp. 2523–2533, 2023.

[22] T. Lohrenz, Z. Li, and T. Fingscheidt, "Multi-encoder learning and stream fusion for transformer-based end-to-end automatic speech recognition," *arXiv preprint arXiv:2104.00120*, 2021.

[23] J. Kahn, M. Riviere, W. Zheng, E. Kharitonov, Q. Xu, P.-E. Mazaré, J. Karadayi, V. Liptchinsky, R. Collobert, C. Fuegen, *et al.*, "Libri-light: A benchmark for asr with limited or no supervision," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 7669–7673, IEEE, 2020.

[24] J.-C. Wang, W.-T. Lu, and M. Won, "Mel-band roformer for music source separation," *arXiv preprint arXiv:2310.01809*, 2023.

[25] G. Yang, E. J. Hu, I. Babuschkin, S. Sidor, X. Liu, D. Farhi, N. Ryder, J. Pachocki, W. Chen, and J. Gao, "Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer," *arXiv preprint arXiv:2203.03466*, 2022.

[26] K. Kilgour, M. Zuluaga, D. Roblek, and M. Sharifi, "Fr\'echet audio distance: A metric for evaluating music enhancement algorithms," *arXiv preprint arXiv:1812.08466*, 2018.

[27] S. Hershey, S. Chaudhuri, D. P. W. Ellis, J. F. Gemmeke, A. Jansen, C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, M. Slaney, R. Weiss, and K. Wilson, "Cnn architectures for large-scale audio classification," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017.

[28] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[29] J. Betker, "Better speech synthesis through scaling," *arXiv preprint arXiv:2305.07243*, 2023.

[30] R. San Roman, Y. Adi, A. Deleforge, R. Serizel, G. Synnaeve, and A. Défossez, "From discrete tokens to high-fidelity audio using multi-band diffusion," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[31] M. Ravanelli, T. Parcollet, P. Plantinga, A. Rouhe, S. Cornell, L. Lugosch, C. Subakan, N. Dawalatabad, A. Heba, J. Zhong, J.-C. Chou, S.-L. Yeh, S.-W. Fu, C.-F. Liao, E. Rastorgueva, F. Grondin, W. Aris, H. Na, Y. Gao, R. D. Mori, and Y. Bengio, "SpeechBrain: A general-purpose speech toolkit," 2021. arXiv:2106.04624.

Table 3: Performance for different masking strategies. Masking the two most fine-grained codebooks results in the best performance across evaluated strategies.

Masking Strategy	FAD ↓	OVL. ↑	CRT. ↑	HRM. ↑
Two Most Fine-grained	2.17	**2.9**	**3.1**	**2.8**
Most Fine-grained	**2.13**	2.8	2.9	2.6
No Music Conditioning	2.38	2.7	3.0	2.5
No Masking	2.18	2.7	2.9	2.6

A Masking Music Conditioning

Since we are using source separation to extract vocals and instrumentals from existing music, we need to mitigate the effect of leftover vocal artifacts in the instrumental stem. To address this, we propose masking the fine-grained codebooks of the instrumental track before feeding it into the text-to-semantic stage. We do not apply masking for the semantic-to-audio stage, since we found that information leakage is mainly a problem at the text-to-semantic stage, and full access to the instrumental track is crucial for the semantic-to-audio stage. The authors of (14) observed an analogous problem when using source separated vocal stems as a conditioning signal.

We evaluate the effect of masking fine-grained instrumental tokens during the text-to-semantic stage. Specifically, the following four strategies S: masking the two most fine-grained codebooks (S1, green and blue in Fig. 2), masking only the most fine-grained codebooks (S2, blue in Fig. 2), completely removing music conditioning (S3), and applying no masking (S4). We fine-tune a text-to-semantic model that was trained without masking using each strategy for 7k steps. Our findings in Table 3 indicate that masking the text-to-semantic stage's music input enhances performance.

B Artist Embedding

To make singing style controllable, we leverage a custom artist embedding model, which we train using a classification objective. Unlike popular speaker and audio embedding models which are typically based on spectrogram inputs (31), we achieve promising results by reusing the decoder implementation from the semantic-to-audio stage for this task. To enhance performance, we incorporated an auxiliary autoregressive loss. The model is trained on vocal stems, in order to minimize the inclusion of instrumental features in the representation.

We train the artist embedding model using our internal singing dataset. As a backbone, we use a 76M parameter decoder-only transformer model and add a two-layer classification head on the last attention layer's output. We use the activations of the classification head's hidden layer as 256 dimensional embeddings. The training objective is given by $L = 0.3 \cdot L_{\text{class}} + L_{\text{AR}}$ where L_{class} represents the cross-entropy loss for the artist classification task, and L_{AR} is an autoregressive loss. For the classification loss, a label-smoothing term with a strength of 0.1 is applied. We use a base learning rate of 10^{-3}. The effect of the artist embeddings on generation can be heard on our sample page.[2]