FlashFoley: Fast Interactive Sketch2Audio Generation

Zachary Novack[1,2]*, **Koichi Saito**[3], **Zhi Zhong**[2], **Takashi Shibuya**[3], **Shuyang Cui**[2], **Julian McAuley**[1], **Taylor Berg-Kirkpatrick**[1], **Christian Simon**[2], **Shusuke Takahashi**[2], **Yuki Mitsufuji**[2,3]

[1]UC – San Diego [2]Sony Group Corporation, Japan [3]Sony AI, USA

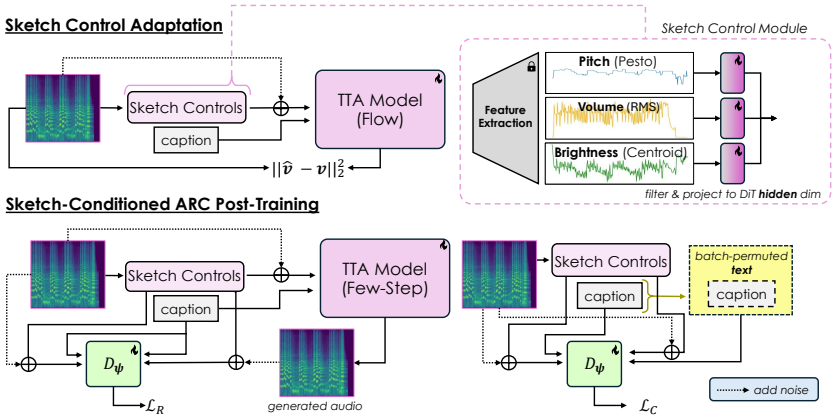

Figure 1: FlashFoley finetuning, including sketch control adaptation and ARC post-training.

Abstract

Despite the growth of Text-to-Audio (TTA) models for creative applications like sound design and live jamming, existing systems, particularly in the open-source, lack the ability for flexible fine-grained control (such as vocal "sketches") while maintaining fast inference speeds for real-time interaction. We address this unnecessary tradeoff between speed and control through **FlashFoley**, the first open-source, *accelerated* sketch2audio model. With FlashFoley, we extend the Sketch2Sound framework, wherein we finetune TTA models with pitch, volume, and brightness controls through simple linear adaptation, to adversarial post-training, allowing the model to generate 11s samples from audio sketches in 75ms. We combine this with a novel zero-shot chunked streaming algorithm, enabling real-time interactive generation while maintaining high-quality fast offline sampling. Audio examples can be found at `https://anonaudiogen.github.io/web`.

1 Introduction

Generative Text-to-Audio (TTA) models [1, 2, 3, 4, 5, 6] have achieved a modern renaissance, with systems capable of generating rich soundscapes for a range of creative tasks, including multimedia sound design, Foley generation [7], and real-time jamming [8, 9]. Such focus for creative applications has shifted research directions in two orthogonal research thrusts: (1) designing systems that allow for controllable generation beyond broad text for fine-grained controls [10, 11, 12], and in particular, audio "sketches" [7], and (2) building algorithmic/hardware accelerations to overcome the slow and non-interactive inference in TTA flow-based models [13, 14, 4, 9, 15] (We refer to Appendix A for a review of related work.). However, such developments in fine-grained controllability and fast sampling (i.e. <0.5s) have proceeded in isolation: broadly, **controllable models are not fast** [7, 10, 11, 16], and **fast models are not controllable** [13, 4, 14, 15].

*Work done during the internship at Sony.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: The First Workshop on Generative and Protective AI for Content Creation.

We present **FlashFoley**: the first open-source[2], *accelerated real-time interactive* sketch2audio model. FlashFoley extends the control setup from Sketch2Sound [7], where we extract time-varying, median-filtered pitch [17], volume, and brightness controls and condition a pre-trained TTA model using simple linear adaptation. This allows the model to both turn vocal "sketches" into realistic audio and perform audio-to-audio style transfer. We then develop the Adversarial Relativistic Contrastive (ARC) Post-Training [4] acceleration method for time-varying controllable generation, allowing generation of 11.88s of stereo 44.1kHz audio in 75ms, 10x faster than existing controllable TTA systems. While these contributions alone enable fast, controllable generation, they cannot by themselves enable *real-time interactivity*, as non-autoregressive TTA models' bidirectional context prevents streaming use cases like live jamming [9]. To circumvent this, we propose a **zero-shot block-autoregressive** algorithm that bestows FlashFoley with semi-streaming generation without training causally, allowing the initial generated audio to be streamed while the user is actively inputting sketch controls.

2 Method

2.1 Background: Text-to-Audio Rectified Flows

FlashFoley is based on the rectified flow (RF) paradigm [18, 19], which is equivalent to diffusion [20] but enjoys better empirical performance [19]. Given a text prompt $\mathbf{c}_{\text{txt}}$ and stereo audio $\mathbf{x}_0 \in \mathbb{R}^{2\times Sf_s}$, (i.e. S seconds at f_s sampling rate), we first embed the audio in some pretrained VAE latent space $\mathbf{z}_0 = \mathcal{E}(\mathbf{x}_0) \in \mathbb{R}^{D\times N}$. RFs [21] learn the *velocity* of the flow from a standard Gaussian p_1 to the data distribution p_0 given $\mathbf{c}_{\text{txt}}$. By defining the forward corruption process (from data to noise) as $\mathbf{z}_t = (1-t)\mathbf{z}_0 + t\boldsymbol{\epsilon}$ where $t \in [0,1]$, $t \sim p_{\text{gen}}(t)$, and $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$[3]. We can derive the ordinary differential equation (ODE) for the *reverse* process (i.e. noise to data) as $\mathrm{d}\mathbf{z}_t = -\mathbf{v}\mathrm{d}t$, where $\mathbf{v} = \boldsymbol{\epsilon} - \mathbf{z}_0$ is the velocity of the ODE. Thus, our goal is to learn $v_{\boldsymbol{\theta}}(\mathbf{z}_t, t, \mathbf{c}_{\text{txt}}) \approx \mathbf{v}$, which can be achieved with a simulation-free objective that simply adds noise to samples and predicts the velocity:

$$\arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{z}_0, \mathbf{c}_{\text{txt}} \sim \mathcal{X}, \boldsymbol{\epsilon} \sim \mathcal{N}(0,\boldsymbol{I}), t \sim p_{\text{gen}}(t)} [\|\mathbf{v} - v_{\boldsymbol{\theta}}(\mathbf{z}_t, t, \mathbf{c}_{\text{txt}})\|_2^2]. \tag{1}$$

Samples can then be generated by integrating the ODE with $v_{\boldsymbol{\theta}}$ using any numerical ODE solver. As no existing TTA models to our knowledge are both fast and controllable through fine-grained signals, it is initially unclear how to achieve both simultaneously. We start from insight from [22] and [13]: it is better to add controls and *then* distill, rather than vice versa or trying both simultaneously.

2.2 Finetuning with Local Sketch Controls through Pre-Transformer Projection

Our goal is to finetune our TTA model with a set of l additional features $\mathbf{F} = \{\mathbf{f}_i\}_{i=1}^{l}$ where $\mathbf{f}_i \in \mathbb{R}^{K_i \times N}$, i.e. *time-varying* local controls. As existing methods for finetuning generally require long tuning runs and introduce 10s-100s of millions of new parameters [23, 10], we instead use the method from Sketch2Sound [7], which we refer to as *Pre-Transformer Projection* (PTP).

The top of Figure 1 describes our PTP-based sketch control adaptation. Given a Diffusion Transformer (DiT) [24]-based model with hidden dimension H, the *initial* hidden state of the model given some noisy latent input $\mathbf{z}_t$ is $\mathbf{h}_{\text{init}} := \text{ProjIn}_{\boldsymbol{\theta}}(\mathbf{z}_t) \in \mathbb{R}^{H\times N}$, where $\text{ProjIn}_{\boldsymbol{\theta}}$ is the input projector of the model, generally comprised of a chain of linear layers with residual connections. Notably, $\mathbf{h}_{\text{init}}$ occurs before any DiT blocks. PTP works by learning only a *single* linear operator $W_i \in \mathbb{R}^{K_i \times H}$ per control, and projecting it directly to $\mathbf{h}_{\text{init}}$, making the new input to the DiT blocks $\mathbf{h}'_{\text{init}} = \mathbf{h}_{\text{init}} + \sum_i W_i^\top \mathbf{f}_i$. Training involves finetuning the entire model with Eq. 1, though in order to ensure that the model does not overfit to the spectral features present in the controls, the controls are convolved with randomly-sized median filters. This filtering enables flexible control at inference-time between rendering the controls exactly (small filters) or using inputs as broad "sketches" (large filters). We additionally discovered further practical benefits to PTP, as noted in Appendix B.

In this work, we focus on three main features [7]: **i). Volume**: the root-mean-squared (RMS) in decibels (dB) of A-weighted magnitude spectra, **ii). Pitch**: the 384-bin pitch probabilities predicted by PESTO [17], dropping out probabilities below 0.1, **iii). Brightness**: the frequency center of mass for each time frame (i.e. the spectral centroid), rescaled to a logarithmic range 0-127 and then to 0-1, where higher values denote more higher harmonics, and thus a perceptually "brighter" sound.

[2]Note that leading sketch2audio models like Sketch2Sound [7] are fully closed source.

[3]We denote this process as sampling from the forward distribution $q(\mathbf{z}_t \mid \mathbf{z}_0)$

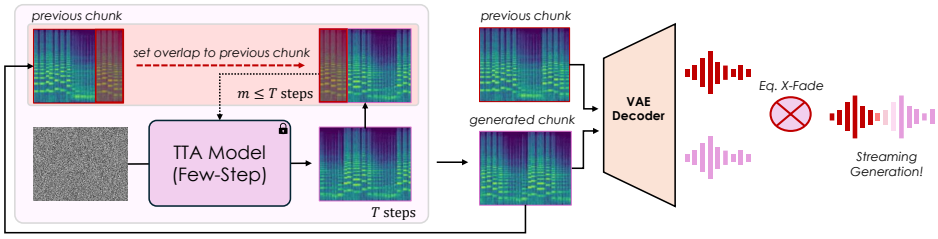

Figure 2: FlashFoley Zero-Shot chunked streaming inference.

2.3 Locally-Conditioned ARC Post Training

Given our sketch-controlled model, we now aim to turn this model into a generator $G_{\boldsymbol{\phi}}$ that can generate with considerably fewer (e.g. 1-8) steps. We begin with following the ARC Post-Training [4] paradigm, show in Figure 1 (bottom). ARC replaces the velocity loss with an adversarial one (using discriminator $D_{\boldsymbol{\psi}}$ initialized from the TTA model itself) to drastically improve output realism at any noise level, thus needing fewer steps during sampling and accelerating inference. ARC is mainly driven by a *relativistic* adversarial [25] objective: Given *paired* real $\mathbf{z}_0$ and generated $\hat{\mathbf{z}}_0$ samples and $\mathcal{C} = \{\mathbf{c}_{\text{txt}}, \mathbf{F}\}$, $\hat{\mathbf{z}}_0 = G_{\boldsymbol{\phi}}(\mathbf{z}_t, t, \mathcal{C})$ is generated from a noisy version of the real sample, and the $G_{\boldsymbol{\phi}}, D_{\boldsymbol{\psi}}$ are trained to minimize/maximize the **relative** difference between discriminator scores:

$$\min_{\boldsymbol{\phi}} \max_{\boldsymbol{\psi}} \mathcal{L}_{\text{R}}(\boldsymbol{\phi}, \boldsymbol{\psi}) = \mathbb{E}_{\mathbf{z}_0, \mathcal{C}, s, \hat{\mathbf{z}}_0} \left[f\left(D_{\boldsymbol{\psi}}(q(\hat{\mathbf{z}}_s \mid \hat{\mathbf{z}}_0), s, \mathcal{C}) - D_{\boldsymbol{\psi}}(q(\mathbf{z}_s \mid \mathbf{z}_0), s, \mathcal{C}) \right) \right], \tag{2}$$

where $f(x) = -\log(1 + e^{-x})$. This relativistic objective on paired samples forms a decision boundary around each real sample, preventing mode collapse and improving quality relative to standard adversarial objectives. As post-training objectives lack direct supervision from teacher models (which is core to distillation methods) and thus suffer on control following, ARC also involves a *contrastive* objective, where $D_{\boldsymbol{\psi}}$ is jointly trained to maximize the same relativistic objective from Eq. 2, but between real samples with correct conditions and real samples with conditions *batch-shuffled* by some random permutation operator $\mathcal{P}$, forcing $D_{\boldsymbol{\psi}}$ to consider the semantic content of the input controls and not focus too much on high-frequency features (as common in GANs [26]):

$$\max_{\boldsymbol{\psi}} \mathcal{L}_{\text{C}}(\boldsymbol{\psi}) = \mathbb{E}_{\mathbf{z}_0, \mathcal{C}, s} \left[f\left(D_{\boldsymbol{\psi}}(q(\mathbf{z}_s \mid \mathbf{z}_0), s, \mathcal{P}[\mathcal{C}]) - D_{\boldsymbol{\psi}}(q(\mathbf{z}_s \mid \mathbf{z}_0), s, \mathcal{C}) \right) \right]. \tag{3}$$

$G_{\boldsymbol{\phi}}$ then samples by iteratively denoising the output and *re-noising* to some lower noise level.

In adapting ARC to time-varying local controls $\mathbf{L}$, though the relativistic loss can remain largely unchanged (as the local conditions only strengthen the pairing between real and generated samples), the contrastive loss raises an important question: ***which* features, between the text and the varied local conditions, should we train $D_{\boldsymbol{\psi}}$ to pay attention to?** While [4] would suggest we should contrast on *all* controls, the different semantic granularity between text vs. time-varying controls changes the difficulty of the contrastive objective for $D_{\boldsymbol{\psi}}$. We ablate this design choice in Section 3 and show that $\mathcal{L}_C$ should only be calculated with randomized *text* inputs, as randomizing the local conditions functionally breaks the discriminators capacity to focus on anything but the local controls.

2.4 Zero-Shot Block Autoregressive Generation

While control finetuning and ARC post-training enable a fast and controllable TTA *offline* system, $G_{\boldsymbol{\phi}}$ is not inherently streamable. This poses an issue particularly for interactive applications like real-time timbre transfer or sound design, as the *entire* sequence of sketch controls must be captured before starting generation. While streamable architectures (like codec LMs [27, 9]) do exist, turning flow models into causal variants remains unexplored for audio systems.

Instead, we designed an approximate block-causal streaming algorithm that works *without any finetuning*, shown in Figure 2. This works by performing generation in chunks of B frames with a stride of k, where the first $B - k$ frames of the current chunk are set to the last $B - k$ frames of the previous chunk during sampling, combined with an *equal-power* crossfade in the audio domain. We provide a detailed algorithm in Appendix D. With respect to capturing the *input controls*, audio output can be streamed with a latency of $BS/N + \delta$ seconds (where $\delta \approx 75$ms accounts for latency from inference and other overhead, which is negligible compared to B), while offline sampling requires a full $S + \delta$ seconds. Though this approach is essentially the "naïve" outpainting method

Table 1: Experimental Results on VimSketch Dataset

Method	Filter Size	Steps	Control L1 (↓) RMS	Centroid	Pitch	Chroma	FAD (↓)	CLAP (↑)	GL (↓)	SL (↓)
SAOS	N/A	50	15.81	15.92	15.19	3.68	0.23	0.32	0.63	12.52
+ controls	5	50	4.89	4.14	10.05	2.87	0.33	0.26	0.63	12.52
FlashFoley	5	8	4.08	3.21	8.02	2.55	0.35	0.23	0.08	11.96
+ BAR	5	8	4.22	3.27	8.58	2.70	0.44	0.21	0.08	6.02
+ sketch $\mathcal{L}_C$	5	8	3.80	2.88	7.52	2.50	0.53	0.13	0.08	11.96

[11, 16] commonly maligned for its reported harsh artifacts at chunk boundaries, there are a few key differences: (1) the constant stream of local control features gives the model fine-grained local supervision, and (2) the audio-domain crossfade explicitly accounts for artifacts. Our results show that this surprisingly simple solution enables streaming generation without significant audio degradation.

3 Experiments

3.1 Setup

FlashFoley is built from the recent Stable Audio Open Small (Base) [4], or SAOS, as our base model. We detail further training information in Appendix C. For evaluation, we use the VimSketch dataset [28], which contains ≈10k vocal imitations of 500 audio classes. We generate 10k samples from each imitation, and evaluate these generations along: (1) sketch control accuracy by measuring L1 distance for all controls, with RMS in dB, and centroid and pitch in semitones (as well as measuring chroma, or pitch class, in semitones) [7] (2) audio quality, using Frechet Distance (FAD) with the CLAP audio backbone [29], and (3) text adherence using CLAP cosine similarity. We also report the Generation Latency (GL, the time it takes to generate output) and the Streaming Latency (SL, the time it takes for the first audio sample to be played given streaming input sketch controls).

3.2 Results

We display our results in Table 1, comparing between our base model (SAOS), the model with finetuned sketch controls (+ controls), sketch controls + ARC Post-Training (FlashFoley), running FlashFoley with our block-AR sampling (+ BAR), and performing ARC with $\mathcal{L}_C$ shuffling *all* conditions, not only text (+ sketch $\mathcal{L}_C$). Despite the degradation in quality / text adherence from adding controls (as expected from [7]), the near 10x acceleration from ARC Post-Training in FlashFoley did not lead to a large drop in these metrics, and even *improved* control accuracy in all controls. Using the zero-shot Block-AR sampling halves the streaming latency, and while this led to degraded audio quality, it is not significantly worse, and notably is still better than using the sketch-based $\mathcal{L}_C$, which severely degraded FAD and CLAP score. In particular, we found that contrasting on all controls caused the discriminator to overfit to the sketch controls, making the generator largely ignore the text inputs and negatively affecting its ability to model higher frequency timbral information, as seen in Figure 3.

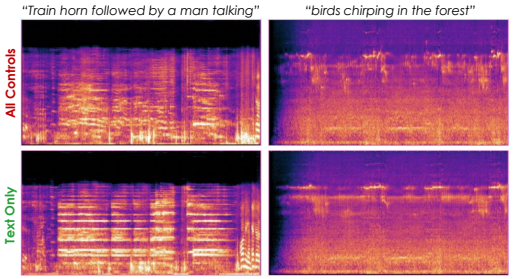

Figure 3: Differences between using $\mathcal{L}_C$ for *all* inputs (top) vs. only *text* inputs (bottom). D_ψ contrasting sketch controls severely impacts text following and quality.

4 Conclusion and Future Work

We have presented FlashFoley, the first open-source, accelerated sketch2audio model, capable of both offline generation in 75ms and streaming generation with a latency of 6s as sketch controls stream in. We hope FlashFoley lays the groundwork for future work both in achieving even faster and higher quality streaming generation, but also learning how such sketch2audio systems can be used in real creative audio generation workflows.

References

[1] Zach Evans, Julian D Parker, CJ Carr, Zack Zukowski, Josiah Taylor, and Jordi Pons. Stable audio open. *arXiv:2407.14358*, 2024.

[2] Zach Evans, CJ Carr, Josiah Taylor, Scott H. Hawley, and Jordi Pons. Fast timing-conditioned latent audio diffusion. In *ICML*, 2024.

[3] Zach Evans, Julian Parker, CJ Carr, Zack Zukowski, Josiah Taylor, and Jordi Pons. Long-form music generation with latent diffusion. *arXiv:2404.10301*, 2024.

[4] Zachary Novack, Zach Evans, Zack Zukowski, Josiah Taylor, CJ Carr, Julian Parker, Adnan Al-Sinan, Gian Marco Iodice, Julian McAuley, Taylor Berg-Kirkpatrick, and Jordi Pons. Fast text-to-audio generation with adversarial post-training. In *WASPAA*, 2025.

[5] Haohe Liu, Zehua Chen, Yi Yuan, Xinhao Mei, Xubo Liu, Danilo Mandic, Wenwu Wang, and Mark D Plumbley. AudioLDM: Text-to-audio generation with latent diffusion models. In *ICML*, 2023.

[6] Haohe Liu, Yi Yuan, Xubo Liu, Xinhao Mei, Qiuqiang Kong, Qiao Tian, Yuping Wang, Wenwu Wang, Yuxuan Wang, and Mark D. Plumbley. Audioldm 2: Learning holistic audio generation with self-supervised pretraining. *TASLP*, 2024.

[7] Hugo Flores García, Oriol Nieto, Justin Salamon, Bryan Pardo, and Prem Seetharaman. Sketch2sound: Controllable audio generation via time-varying signals and sonic imitations. In *ICASSP*. IEEE, 2025.

[8] Hugo Flores Garcia, Prem Seetharaman, Rithesh Kumar, and Bryan Pardo. VampNet: Music generation via masked acoustic token modeling. In *ISMIR*, 2023.

[9] Lyria Team, Antoine Caillon, Brian McWilliams, Cassie Tarakajian, Ian Simon, Ilaria Manco, Jesse Engel, Noah Constant, Pen Li, Timo I Denk, et al. Live music models. *arXiv preprint arXiv:2508.04651*, 2025.

[10] Shih-Lun Wu, Chris Donahue, Shinji Watanabe, and Nicholas J. Bryan. Music ControlNet: Multiple time-varying controls for music generation. *TASLP*, 2024.

[11] Zachary Novack, Julian McAuley, Taylor Berg-Kirkpatrick, and Nicholas J. Bryan. DITTO: Diffusion inference-time T-optimization for music generation. In *ICML*, 2024.

[12] Zachary Novack, Julian McAuley, Taylor Berg-Kirkpatrick, and Nicholas J. Bryan. DITTO-2: Distilled diffusion inference-time t-optimization for music generation. In *ISMIR*, 2024.

[13] Zachary Novack, Ge Zhu, Jonah Casebeer, Julian McAuley, Taylor Berg-Kirkpatrick, and Nicholas J. Bryan. Presto! distilling steps and layers for accelerating music generation. In *ICLR*, 2025.

[14] Yatong Bai, Trung Dang, Dung Tran, Kazuhito Koishida, and Somayeh Sojoudi. Accelerating diffusion-based text-to-audio generation with consistency distillation. In *Interspeech*, 2024.

[15] Koichi Saito, Dongjun Kim, Takashi Shibuya, Chieh-Hsin Lai, Zhi Zhong, Yuhta Takida, and Yuki Mitsufuji. SoundCTM: Unifying score-based and consistency models for full-band text-to-sound generation. In *ICLR*, 2025.

[16] Fang-Duo Tsai, Shih-Lun Wu, Weijaw Lee, Sheng-Ping Yang, Bo-Rui Chen, Hao-Chung Cheng, and Yi-Hsuan Yang. Musecontrollite: Multifunctional music generation with lightweight conditioners. *arXiv preprint arXiv:2506.18729*, 2025.

[17] Alain Riou, Stefan Lattner, Gaëtan Hadjeres, and Geoffroy Peeters. Pesto: Pitch estimation with self-supervised transposition-equivariant objective. In *ISMIR*, 2023.

[18] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv:2209.03003*, 2022.

[19] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *ICML*, 2024.

[20] Diederik Kingma and Ruiqi Gao. Understanding diffusion objectives as the elbo with simple data augmentation. *NeurIPS*, 36, 2023.

[21] Xingchao Liu, Chengyue Gong, and qiang liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *ICLR*, 2023.

[22] Kangfu Mei, Mauricio Delbracio, Hossein Talebi, Zhengzhong Tu, Vishal M Patel, and Peyman Milanfar. Codi: Conditional diffusion distillation for higher-fidelity and faster image generation. In *CVPR*, pages 9048–9058, 2024.

[23] Tom Baker and Javier Nistal. Lilac: A lightweight latent controlnet for musical audio generation. *arXiv preprint arXiv:2506.11476*, 2025.

[24] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *IEEE/CVF International Conference on Computer Visio (ICCV)*, 2023.

[25] Nick Huang, Aaron Gokaslan, Volodymyr Kuleshov, and James Tompkin. The gan is dead; long live the gan! a modern baseline gan. In *ICML Workshop on Structured Probabilistic Inference and Generative Modeling*, 2024.

[26] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *NeurIPS*, volume 27. Curran Associates, Inc., 2014.

[27] Alexandre Défossez, Jade Copet, Gabriel Synnaeve, and Yossi Adi. High fidelity neural audio compression. *arXiv:2210.13438*, 2022.

[28] Bongjun Kim, Mark Cartwright, Fatemeh Pishdadian, and Bryan Pardo. Vimsketch dataset, March 2019.

[29] Yusong Wu, Ke Chen, Tianyu Zhang, Yuchen Hui, Taylor Berg-Kirkpatrick, and Shlomo Dubnov. Large-scale contrastive language-audio pretraining with feature fusion and keyword-to-caption augmentation. In *ICASSP*, 2023.

[30] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *ICLR*, 2020.

[31] Ho Kei Cheng, Masato Ishii, Akio Hayakawa, Takashi Shibuya, Alexander Schwing, and Yuki Mitsufuji. Taming multimodal joint training for high-quality video-to-audio synthesis. *arXiv*, 2024.

[32] Ziyang Chen, Prem Seetharaman, Bryan Russell, Oriol Nieto, David Bourgin, Andrew Owens, and Justin Salamon. Video-guided foley sound generation with multimodal controls. In *CVPR*, 2025.

[33] Yiming Zhang, Yicheng Gu, Yanhong Zeng, Zhening Xing, Yuancheng Wang, Zhizheng Wu, and Kai Chen. Foleycrafter: Bring silent videos to life with lifelike and synchronized sounds. *arXiv preprint arXiv:2407.01494*, 2024.

[34] Junqi Zhao, Jinzheng Zhao, Haohe Liu, Yun Chen, Lu Han, Xubo Liu, Mark Plumbley, and Wenwu Wang. Audioturbo: Fast text-to-audio generation with rectified diffusion. *arXiv preprint arXiv:2505.22106*, 2025.

[35] Xinhao Mei, Chutong Meng, Haohe Liu, Qiuqiang Kong, Tom Ko, Chengqi Zhao, Mark D. Plumbley, Yuexian Zou, and Wenwu Wang. WavCaps: A ChatGPT-assisted weakly-labelled audio captioning dataset for audio-language multimodal research. *TASLP*, pages 1–15, 2024.

[36] Junsong Chen, Shuchen Xue, Yuyang Zhao, Jincheng Yu, Sayak Paul, Junyu Chen, Han Cai, Enze Xie, and Song Han. Sana-sprint: One-step diffusion with continuous-time consistency distillation. *arXiv:2503.09641*, 2025.

A Related Work

Growth of TTA systems has been driven largely by the advent of diffusion models [30] for audio generation [5, 6, 2, 3, 1, 4]. As broad text controls offer limited controllability for generating creative audio scenes, there has been growing interest in imbuing TTA systems with more flexible, fine-grained controls [7, 10], such as text-queried *video*-to-audio generation [31, 32, 33]. Our work is inspired by Sketch2Sound [7], which instead extracts sketch-like signals (volume, pitch, and spectral brightness) from audio signals as conditioning, allowing control of TTA systems with flexible, time-varying signals with minimal finetuning cost. However, Sketch2Sound is both fully closed-source and operates on standard slow diffusion models, restricting its practical use case for real-time creativity.

There has also been considerable research into the acceleration of *text-only* TTA systems, through distillation approaches such as consistency-based methods [14, 15, 12], rectification [34], or distribution matching [13]. In this work, we instead focus on the recent ARC Post-Training [4] method, which is the current SOTA for TTA acceleration and notably avoids the need for a strong teacher model, and extend it to time-varying controllable TTA generation with real-time interaction.

B Analysis of PTP Dynamics

PTP is particularly beneficial for easy finetuning beyond the clear parameter efficiency for a number of reasons: (1) PTP minimizes changes in the computational graph of the model, which makes PTP exceedingly easy to implement in complex DiT implementations without breaking the pretrained model, (2) it learns *independent* parameters for each control *only*, with no parameter sharing across controls or new parameters for the noisy latent input. We found this latter point to be critical when designing FlashFoley: Other conditioning methods, like channel-wise concatenation or projection to the *VAE space* D (rather than the hidden dimension) resulted in training instability and degraded audio quality. To expand more upon this latter point, consider three possible methods for injecting local conditions into a pre-trained RF model:

1. **PTP**, where we project the conditions into the *initial DiT hidden state*.
2. **Input Addition**, where we project the conditions instead onto the *noisy VAE latent* before entering the model.
3. **Channel concatenation**, where we *append* the local conditions to the input VAE latent, before entering the model.

While the differences between these methods may appear unclear, we will show below that they can lead to very different practical implementations of local conditioning depending on the structure of $\mathrm{ProjIn}_{\boldsymbol{\theta}}$ that may effect training dynamics.

First, consider the case where $\mathrm{ProjIn}_{\boldsymbol{\theta}}(\mathbf{z}) := W_{\mathbf{z}}^{\top}\mathbf{z}$, i.e. we transfer from the VAE space D to the hidden state of the model H with a single linear layer. In this case, **PTP and channel concatenation are equivalent**: Denote our input (noisy) VAE latent $\mathbf{z} \in \mathbb{R}^{D\times N}$, and W.L.O.G. single local feature $\mathbf{f} \in \mathbb{R}^{K\times N}$. For channel concatenation, we must modify our pretrained $\mathrm{ProjIn}_{\boldsymbol{\theta}}$ to accept double the channel count, which means initializing the new weight matrix as:

$$W'_{\mathbf{z}} = \begin{bmatrix} W_{\mathbf{z}} \\ W_{\mathbf{f}} \end{bmatrix} \in \mathbb{R}^{(K+D)\times H},$$

where $W_{\mathbf{f}} \in \mathbb{R}^{K\times H}$. We denote this modified operator as $\mathrm{ProjIn}_{\boldsymbol{\theta}'}$. If we denote the concatenated input as $\mathbf{z}' = [\mathbf{z} \quad \mathbf{f}]^{\top} \in \mathbb{R}^{(K+D)\times N}$, then:

$$\begin{aligned}
\mathrm{ProjIn}_{\boldsymbol{\theta}'}(\mathbf{z}') &= W_{\mathbf{z}}'^{\top}\mathbf{z}' \\
&= \begin{bmatrix} W_{\mathbf{z}}^{\top} & W_{\mathbf{f}}^{\top} \end{bmatrix} \begin{bmatrix} \mathbf{z} \\ \mathbf{f} \end{bmatrix} \\
&= W_{\mathbf{z}}^{\top}\mathbf{z} + W_{\mathbf{f}}^{\top}\mathbf{f} \\
&= \mathrm{ProjIn}_{\boldsymbol{\theta}}(\mathbf{z}) + W_{\mathbf{f}}^{\top}\mathbf{f},
\end{aligned}$$

which is equivalent to PTP. Contrast this with input addition, where instead $W_{\mathbf{f}} \in \mathbb{R}^{K \times D}$, $W_{\mathbf{z}}$ is left unmodified, and $\mathbf{z}' = \mathbf{z} + W_{\mathbf{f}}^\top \mathbf{f}$. Then:

$$\begin{aligned}\mathrm{ProjIn}_{\boldsymbol{\theta}'}(\mathbf{z}') &= W_{\mathbf{z}}^\top \mathbf{z}' \\ &= W_{\mathbf{z}}^\top (\mathbf{z} + W_{\mathbf{f}}^\top \mathbf{f}) \\ &= \mathrm{ProjIn}_{\boldsymbol{\theta}}(\mathbf{z}) + W_{\mathbf{z}}^\top W_{\mathbf{f}}^\top \mathbf{f}\end{aligned}$$

It is clear that input addition is not equivalent to PTP (and thus channel concatenation), but importantly, it is different in that $W_{\mathbf{z}}$ is now *shared* between $\mathbf{z}$ and $\mathbf{f}$ for projection into the DiT hidden space. This may explain the empirical instability observed for input addition: while PTP/channel concatenation let each matrix independently learn to project the latent and control distributions separately, input addition prevents $W_{\mathbf{z}}$ from solely focusing on the latent distribution.

However, now consider the case where $\mathrm{ProjIn}_{\boldsymbol{\theta}}$ is more complicated, which is the case for the architecture of FlashFoley (building off of SAO-Small). Here, $\mathrm{ProjIn}_{\boldsymbol{\theta}}(\mathbf{z}) := W_{\mathbf{z}}^\top (W_{\mathbf{zz}}^\top \mathbf{z} + \mathbf{z})$, i.e. we transfer from the VAE space to the hidden state by first using a square matrix transformation $W_{\mathbf{zz}} \in \mathbb{R}^{D \times D}$ on $\mathbf{z}$ with a residual connection, and *then* up-projecting to the hidden state. While PTP still only learns a single linear layer $W_{\mathbf{f}}$, channel concatenation must now modify *both* $W_{\mathbf{z}}$ and $W_{\mathbf{zz}}$ to accomodate the higher channel count, giving us:

$$W'_{\mathbf{zz}} = \begin{bmatrix} W_{\mathbf{zz}} & W_{\mathbf{zf}} \\ W_{\mathbf{fz}} & W_{\mathbf{ff}} \end{bmatrix} \in \mathbb{R}^{(K+D)\times(K+D)} \quad W'_{\mathbf{z}} = \begin{bmatrix} W_{\mathbf{z}} \\ W_{\mathbf{f}} \end{bmatrix} \in \mathbb{R}^{(K+D)\times H}$$

If we consider the concatenated input as $\mathbf{z}' = [\mathbf{z} \quad \mathbf{f}]^\top \in \mathbb{R}^{(K+D)\times N}$ once more, then

$$\begin{aligned}\mathrm{ProjIn}_{\boldsymbol{\theta}'}(\mathbf{z}') &= W'^\top_{\mathbf{z}} (W'^\top_{\mathbf{zz}} \mathbf{z}' + \mathbf{z}') \\ &= \begin{bmatrix} W_{\mathbf{z}}^\top & W_{\mathbf{f}}^\top \end{bmatrix} \left(\begin{bmatrix} W_{\mathbf{zz}}^\top & W_{\mathbf{fz}}^\top \\ W_{\mathbf{zf}}^\top & W_{\mathbf{ff}}^\top \end{bmatrix} \begin{bmatrix} \mathbf{z} \\ \mathbf{f} \end{bmatrix} + \begin{bmatrix} \mathbf{z} \\ \mathbf{f} \end{bmatrix} \right) \\ &= \begin{bmatrix} W_{\mathbf{z}}^\top & W_{\mathbf{f}}^\top \end{bmatrix} \begin{bmatrix} W_{\mathbf{zz}}^\top \mathbf{z} + W_{\mathbf{fz}}^\top \mathbf{f} + \mathbf{z} \\ W_{\mathbf{zf}}^\top \mathbf{z} + W_{\mathbf{ff}}^\top \mathbf{f} + \mathbf{f} \end{bmatrix} \\ &= W_{\mathbf{z}}^\top W_{\mathbf{zz}}^\top \mathbf{z} + W_{\mathbf{z}}^\top W_{\mathbf{fz}}^\top \mathbf{f} + W_{\mathbf{z}}^\top \mathbf{z} + W_{\mathbf{f}}^\top W_{\mathbf{zf}}^\top \mathbf{z} + W_{\mathbf{f}}^\top W_{\mathbf{ff}}^\top \mathbf{f} + W_{\mathbf{f}}^\top \mathbf{f} \\ &= \mathrm{ProjIn}_{\boldsymbol{\theta}}(\mathbf{z}) + W_{\mathbf{z}}^\top W_{\mathbf{fz}}^\top \mathbf{f} + W_{\mathbf{f}}^\top W_{\mathbf{zf}}^\top \mathbf{z} + W_{\mathbf{f}}^\top W_{\mathbf{ff}}^\top \mathbf{f} + W_{\mathbf{f}}^\top \mathbf{f},\end{aligned}$$

which is drastically different than PTP, as PTP maintains its form independent of the architecture of $\mathrm{ProjIn}_{\boldsymbol{\theta}}$. In particular, in this scenario channel concatenation *also* introduces weight-sharing on the final projection matrices like in input addition, and while the representative capacity for channel concatenation is clearly larger than in PTP, this behavior may explain the issues with finetuning from a pre-trained checkpoint. Broadly, this means that **PTP can be seen as channel concatenation with block-diagonal projection matrices** (if one represents all of the control matrices as their single product matrix), where all feature projection is performed *independently* before pooling to the hidden state.

C Experimental Details

We use the recent Stable Audio Open Small (Base) [4], or SAOS, as our base model, which is a 340M parameter DiT with both text and total seconds conditioning. SAOS operates within a stereo 44.1kHz VAE with 2048x temporal compression (i.e. 21.5 Hz) on 256 latent frames, or about 11.88s of audio. For both finetuning stages, we use the WavCaps dataset [35], which is comprised of 400K samples of general audio at varying sampling rates (mostly 32 kHz and 48 kHz), which we resample to 44.1kHz and truncate to 11.88 seconds. Following Sketch2Sound[7], we perform control finetuning for 40K steps, and perform ARC Post-Training for 70K steps, both with a batch size of 256 across 4 H100s.

For inference with the base SAOS model and SAOS+sketch controls, we use the Flow-DPM solver [36] with 50 steps and a CFG weight of 7, while all FlashFoley experiments use 8 steps with no CFG. For SAOS+sketch controls, we follow [7] in their CFG design, where the "conditional" branch receives both sketch and text controls, and the "unconditional" branch receives only the sketch controls. For all experiments with sketch controls, we use a median filter width of 5, which roughly matches the 0.25s filter width from [7]. For FlashFoley with the Block-AR sampling, we test with $B = 128$ (i.e. ≈6s of latent frames), $k = 64$ (3s of latents), and a sampling depth of $m = 5$, which controls how many of the sampling steps we apply the overlap operation.

D Block Autoregressive Sampling Algorithm

Algorithm 1 FlashFoley Zero-Shot Block Autoregressive Sampling

input : Model $G_{\boldsymbol{\phi}}$, prompt $\mathbf{c}_{\text{txt}}$, stream of sketch controls $\mathbf{L}$, block size B, stride k, sampling depth m, number of sampling steps T, timestep schedule $\{t_i\}_{i=0}^{T}$, max generation length $N_{\max}$, VAE Decoder $\mathcal{D}$, output audio buffer $\mathbf{a}$, XFADE function.

$\mathbf{z}_{\text{buff}}, \mathbf{x}_{\text{buff}} = \emptyset, \emptyset$ // `Initiliaze chunk buffer and # of blocks`
$N_B = \lceil (N_{\max} - B)/k + 1 \rceil$
for $b = 0$ to N_B **do**
  $\mathbf{z}^{(b)} \sim \mathcal{N}(0, \boldsymbol{I})$ // `Initialize noise & current block sketch controls`
  $\mathbf{L}_b = \mathbf{L}[bk : bk + B]$
  for $i = 0$ to T **do**
    $\hat{\mathbf{z}}_0^{(b)} = G_{\boldsymbol{\phi}}(\mathbf{z}^{(b)}, t_i, \mathbf{c}_{\text{txt}}, \mathbf{L}_b)$ // `Predict clean latent`
    if $\mathbf{z}_{\text{buff}} \neq \emptyset$ and $i < m$ **then**
      $\hat{\mathbf{z}}_0^{(b)}[: B - k] = \mathbf{z}_{\text{buff}}[-(B - k) :]$ // `Set overlap to previous chunk`
    end if
    $\mathbf{z}^{(b)} = (1 - t_{i+1})\hat{\mathbf{z}}_0^{(b)} + t_{i+1}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$ // `Re-noise to lower level`
  end for
  $\mathbf{x}^{(b)} = \mathcal{D}(\hat{\mathbf{z}}_0^{(b)})$ // `Decode to audio`
  if $\mathbf{x}_{\text{buff}} \neq \emptyset$ **then**
    $\mathbf{a}[(b - 1)k : bk + B] = \text{XFADE}(\mathbf{x}_{\text{buff}}, \mathbf{x}^{(b)})$ // `Crossfade and write to buffer`
  end if
  $\mathbf{z}_{\text{buff}}, \mathbf{x}_{\text{buff}} = \hat{\mathbf{z}}_0^{(b)}, \mathbf{x}^{(b)}$
end for

output : $\mathbf{a}$