Asura's Harp: Direct Latent Control of Neural Sound

Kaj Bostrom*
bostromkaj@gmail.com

Abstract

Live neural audio generation has the potential to enable new modes of performance and experimentation in music and sound art. The latest generation of proprietary audio models boast high perceptual quality, text and sample prompting, and in some cases live generation. However, these services are fundamentally limited. Remote hosting obstructs seamless interaction and customization, while prompt-based control pigeonholes outputs into existing descriptors and sounds. In this work, we describe Autoencoding Sequentially Unrolled Amortized Flow (ASUrA-Flow), a generative audio model capable of live output on local hardware. Inspired by the emergent latent codes of generative adversarial networks and variational autoencoders, ASUrA-Flow is designed to be played in real time by directly modulating its latent control vector without text or audio prompts. We train our architecture end-to-end on raw audio using amortized flow matching, a novel distribution-matching objective that provides stable training and efficient, high fidelity output directly in signal space.

1 Introduction

Achieving a particular sound with a prompt-based audio model requires a recorded sample or a description that captures the desired quality in existing aesthetic terms. Put another way, prompting presupposes that we already know what we want from the model, and that we can point to it. A system that only admits control through prompting therefore fundamentally limits the creative impulse to look beyond what we have already catalogued. Our goal is to push generative audio modeling past keyword pastiches. We seek a more responsive, less referential, more open-ended mode of control. We want models that facilitate play, not just reproduction.

In this paper, we describe Autoencoding Sequentially Unrolled Amortized Flow (ASUrA-Flow), an architecture for real time neural audio generation. We train ASUrA-Flow end-to-end with amortized flow matching (AFM), a novel two-time flow matching objective that unifies flow matching [9] and progressive mean flow self-distillation [3, 7]. We then train a control mapping to expose the latent manifold learned by the autoencoding objective as a well-behaved dense space of control vectors. This makes it possible to play ASUrA-Flow in real time with reference-free modulation along arbitrary paths in latent space. Output samples and source code are available at `bostromk.net/ASURA`.

2 Background

Treating audio as a Euclidean vector space, just one second of signal can amount to anywhere from 40,000 to 200,000 dimensions depending on sampling rate and channel count. This makes generative modeling of audio uniquely difficult, in part due to the curse of dimensionality [2] and in part due to the sheer computational footprint of neural networks with large enough receptive fields to capture audio context at perceptually relevant timescales. Two families of approaches have recently begun to yield effective audio models: flow matching over fixed-length samples [12] and autoregressive

*`bostromk.net` Work conducted independently and unaffiliated with author's employer.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: AI for Music.

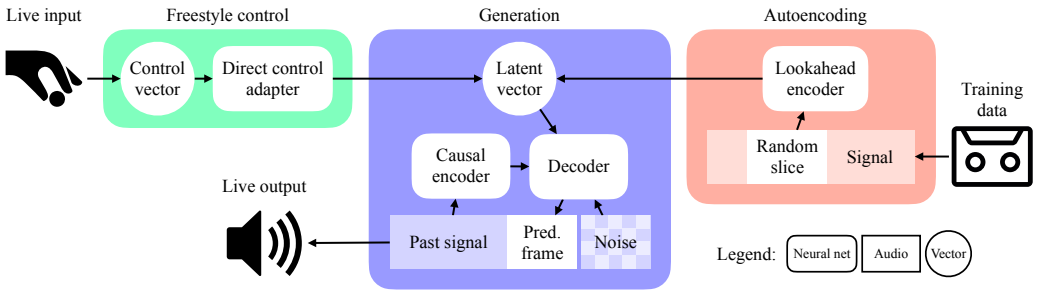

Figure 1: An overview of the ASUrA-Flow architecture. Input is provided either in the form of conditioning audio ('Autoencoding') or a manual control vector ('Freestyle control'). These inputs are mapped to a latent vector which conditions the output distribution of a frame decoder UNet trained with an amortized flow matching objective (4). Output is produced frame-by-frame and continuously re-encoded to provide autoregressive conditioning for the frame decoder.

prediction of neural codec symbols ('codebook language models') [1]. Both flow matching models and codebook LMs make tradeoffs in order to manage the inherent challenges of the problem.

Codebook language models discretize chunks of audio into tokens using a codec model [8]. This makes the learning problem tractable by massively compressing the signal and allowing outputs to be modeled as a series of choices over next tokens at each step. These models allow real-time generation, but large data-hungry transformer backbones are required to effectively learn the next-token distribution. Steep data and compute requirements make it impractical to train these models on self-recorded audio, posing an obstacle to the small-scale custom model paradigm artists find most exciting and ethically viable [4].

Flow matching models can generate good samples with smaller networks and less training data by factoring generation into a series of steps moving gradually from noise to signal [9]. Sadly, this sequence of steps is also what prevents flow models from generating in real time, as each step requires a costly neural function evaluation (NFE). The paths between independently sampled noise (source) and signal (target) used to train a flow model are shown in the left panel of Figure 2; paths along the resulting trained flow are shown in the center panel. Mean flow models [3, 6, 7] try to learn straight lines from source to target by averaging the velocity along these paths; one-step mean flow paths are shown in the right panel. For formal descriptions of flow matching and mean flow, see Appendix A.

Replacing the categorical next-token distribution of a discrete autoregressive transformer with a flow matching 'decoder head' yields continuous autoregressive models (CAMs) capable of sequential generation directly in continuous signal or embedding space [11]. In this work, we push CAMs to achieve real-time end-to-end audio output by amortizing the decoder head's flow into a single step.

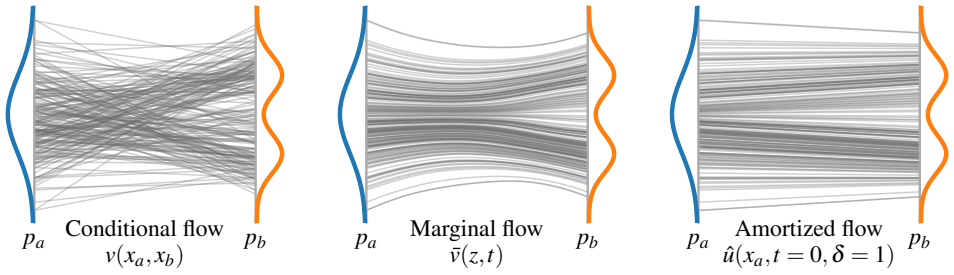

Figure 2: Paths along conditional, marginal, and amortized flows from a unimodal source distribution p_a to a bimodal target p_b.

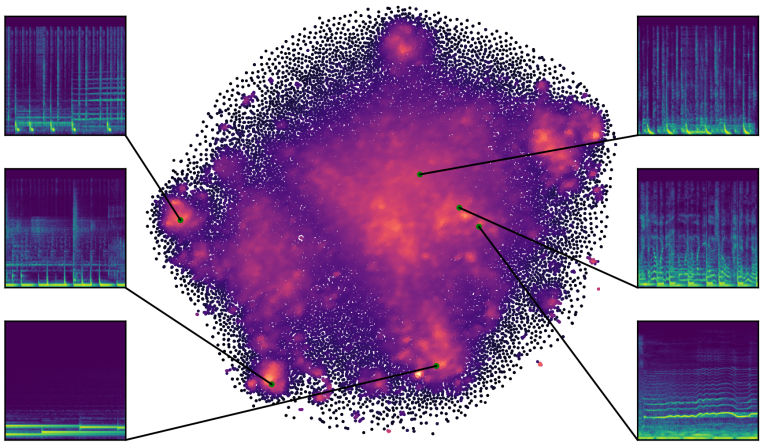

Figure 3: The latent lookahead embedding space learned end-to-end by ASUrA-Flow, visualized using t-SNE with points colored by Gaussian KDE density. Samples appear to cluster by timbre and rhythm, presumably because these factors reflect much of the variation in the next-frame distribution.

3 Methods

ASUrA-Flow is a continuous autoregressive model. We divide the signal into small time slices ('frames') and generate by using a flow-based decoder network to translate successive condition vectors $\mathbf{c}_i \in \mathbb{R}^{d_c}$ and noise frames into signal frames. We then overlap and blend the decoded frames one by one to produce an output audio stream. A causal transformer encoder network produces the next condition vector from past audio. During training, the condition vector also includes an additive contribution from a non-causal transformer encoder; this latent 'lookahead embedding' is where we later inject inference-time control.

3.1 Audio Representation

In preliminary experiments, we found that the time-frequency domain was a more effective medium than the time domain for capturing low-volume, high-frequency signal components. We represent audio in the time-frequency domain using complex short-time Fourier transform (STFT) coefficients with logarithmically-rescaled magnitudes. We choose this over other time-frequency representations like the mel spectrogram because the complex STFT is invertible, meaning we don't need a separate vocoder to produce an output waveform.

3.2 Training

We train the decoder to match the empirical distribution of successor frames by minimizing the amortized flow matching loss (Equation 4) over a corpus of training audio with respect to the encoder and decoder parameters. The amortized flow matching loss measures the error between the model's predicted mean flow velocity and a 'target' average of two substeps, just like the interval-splitting consistency loss of Boffi et al. [3][2] and Guo et al. [7]. Consistency training must also enforce the 'base case', the curved reference flow, which means spending most of training optimizing the model to predict something *other* than the mean flow. Unlike a pure consistency objective, our AFM target starts from the base flow and corrects it to straighten it in expectation. The learned mean flow is anchored to the ground-truth distribution coupling, and there is no need to trade off between flow matching and mean flow consistency. The AFM objective covers every stage of Figure 2, untangling independent sample couplings into a coherent flow and simultaneously amortizing the integration of that flow to yield a one-step mapping from noise to signal. More details on the AFM objective are presented in Appendix A.2.

[2]Interval-splitting consistency training is referred to in that work as 'progressive self-distillation'.

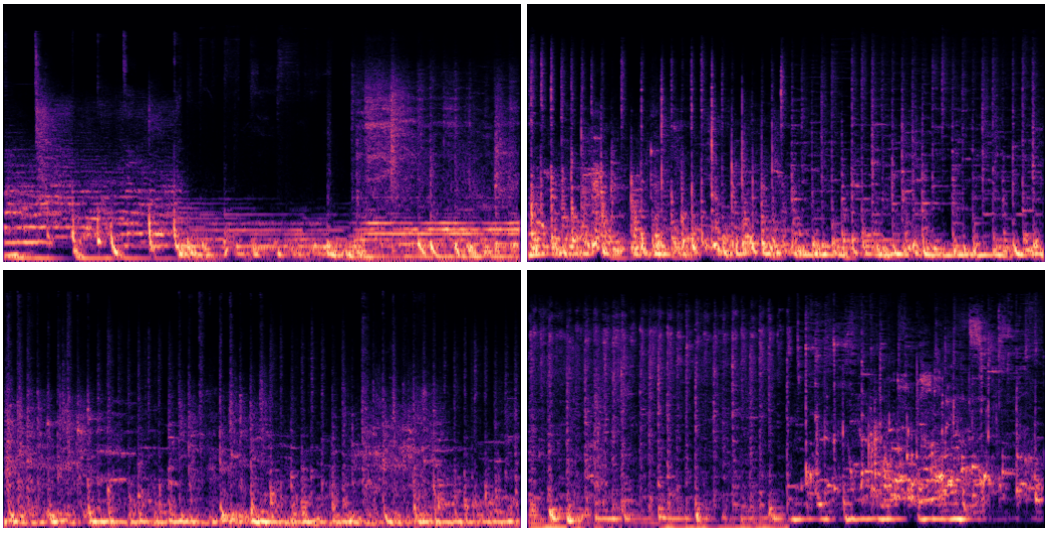

Figure 4: 6-second samples from ASUrA-Flow containing dynamic control input updates. Top row patterns: ABC, AB; bottom row: ABABAB, AB

3.3 Control

As we mentioned in Section 1, we want a way to control generation beyond prompting the model with encoded audio samples. Our solution, inspired by Preechakul et al. [13] and Esling et al. [5], is a direct control adapter that replaces the lookahead embedding from the non-causal encoder. The adapter is a feedforward neural network trained using the AFM objective to map points from a simple, dense distribution, i.e. the multivariate normal $N(0, I_{d_c})$, to the empirical distribution of lookahead embeddings. We can then interpolate and explore the model's latent manifold freely in the well-behaved input space of the direct control adapter. We note that unlike variational autoencoding schemes, the distribution-matching adapter approach avoids imposing prior regularization on the encoder that would tend to reduce the information content of the lookahead embeddings. We use t-SNE to visualize the learned lookahead embedding space for the music dataset used in our main experiment in Figure 3.

4 Experiment

We train ASUrA-Flow on a private dataset of 2,000 hours of recorded radio mixes comprising various genres of dance and electronic music. See Appendix B for hyperparameters.

Figure 4 shows generated samples following a series of control adapter input updates over the course of each clip, demonstrating that interactive play is viable with our approach. Audio versions are provided as supplemental material.

While we do not feel we would be within our rights to distribute the parameters of a generative model derived from music that is not ours to rebroadcast, we will provide full training code and encourage the community to train their own models for personal use.

5 Conclusion

We introduce ASUrA-Flow, a live neural audio model that can be trained end-to-end in a single stage. Following our position that the creative utility of prompt-based generative models is fundamentally limited, we abandon prompting and train a control adapter that makes it possible to directly modulate ASUrA-Flow's latent lookahead embedding vector in real time. This approach opens the door to new modes of play in polyphonic audio spaces. We are excited to explore these possibilities further and even more excited to share our methods and see where others take them.

Acknowledgments

Reluctant thanks to Maya Shamir for moral support. Thanks to Niloofar Mireshghallah and Taylor Berg-Kirkpatrick for feedback and advice on an early prototype.

References

[1] A. Agostinelli, T. I. Denk, Z. Borsos, J. Engel, M. Verzetti, A. Caillon, Q. Huang, A. Jansen, A. Roberts, M. Tagliasacchi, M. Sharifi, N. Zeghidour, and C. Frank. Musiclm: Generating music from text, 2023. URL https://arxiv.org/abs/2301.11325.

[2] R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton Legacy Library. Princeton University Press, 1961. ISBN 9780691079011. URL https://books.google.com/books?id=POAmAAAAMAAJ.

[3] N. M. Boffi, M. S. Albergo, and E. Vanden-Eijnden. How to build a consistency model: Learning flow maps via self-distillation, 2025. URL https://arxiv.org/abs/2505.18825.

[4] N. Bryan-Kinns, A. Wszeborowska, O. Sutskova, E. Wilson, P. Perry, R. Fiebrink, G. Vigliensoni, R. Lindell, A. Coronel, and N. N. Correia. Leveraging small datasets for ethical and responsible ai music making. In *Audio Mostly 2025*, 2025. URL https://ualresearchonline.arts.ac.uk/id/eprint/24065/.

[5] P. Esling, N. Masuda, A. Bardet, R. Despres, and A. Chemla-Romeu-Santos. Universal audio synthesizer control with normalizing flows, 2019. URL https://arxiv.org/abs/1907.00971.

[6] Z. Geng, M. Deng, X. Bai, J. Z. Kolter, and K. He. Mean flows for one-step generative modeling, 2025. URL https://arxiv.org/abs/2505.13447.

[7] Y. Guo, W. Wang, Z. Yuan, R. Cao, K. Chen, Z. Chen, Y. Huo, Y. Zhang, Y. Wang, S. Liu, and Y. Wang. Splitmeanflow: Interval splitting consistency in few-step generative modeling, 2025. URL https://arxiv.org/abs/2507.16884.

[8] R. Kumar, P. Seetharaman, A. Luebs, I. Kumar, and K. Kumar. High-fidelity audio compression with improved rvqgan, 2023. URL https://arxiv.org/abs/2306.06546.

[9] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le. Flow matching for generative modeling, 2023. URL https://arxiv.org/abs/2210.02747.

[10] I. Loshchilov and F. Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

[11] M. Pasini, J. Nistal, S. Lattner, and G. Fazekas. Continuous autoregressive models with noise augmentation avoid error accumulation, 2024. URL https://arxiv.org/abs/2411.18447.

[12] K. R. Prajwal, B. Shi, M. Lee, A. Vyas, A. Tjandra, M. Luthra, B. Guo, H. Wang, T. Afouras, D. Kant, and W.-N. Hsu. Musicflow: Cascaded flow matching for text guided music generation, 2024. URL https://arxiv.org/abs/2410.20478.

[13] K. Preechakul, N. Chatthee, S. Wizadwongsa, and S. Suwajanakorn. Diffusion autoencoders: Toward a meaningful and decodable representation, 2022. URL https://arxiv.org/abs/2111.15640.

A Flow

Flow matching (FM) [9] constructs a velocity field that transports samples from a distribution p_a to another distribution p_b over $\mathbb{R}^d$ by training a model $\hat{v}_\theta(z,t)$ to estimate the expected velocity $\bar{v}(z,t) = \mathbb{E}[x_b - x_a]$ over straight paths between samples $x_a \sim p_a, x_b \sim p_b$ that pass through the intermediate point $z = (1-t)x_a + tx_b$ for $t \in [0,1]$. These paths are visualized in the leftmost panel of Figure 2. In its basic form, the FM objective is:

$$\mathcal{L}_{\text{FM}} = \mathop{\mathbb{E}}_{\substack{x_a, x_b \\ t \sim U[0,1]}} \left[\| \hat{v}_\theta(z,t) - (x_b - x_a) \|_2^2 \right] \tag{1}$$

Optimizing a parametric model $\hat{v}_\theta(z,t)$ to minimize (1) gives an approximation of the marginal velocity field $\bar{v}(z,t)$ shown in the center pane of Figure 2. We can produce new samples from p_b by

starting at $z = x_a \sim p_a$ and following $\hat{v}_\theta(z, t)$ from $t = 0$ to $t = 1$. The bunching of paths between independently sampled x_a and x_b creates curvature in $\bar{v}$. Following a path through this field requires reevaluating the velocity many times to keep up with direction changes along the way. Following a straight velocity field would only require knowing the initial velocity, which would let us convert samples from p_a into samples from p_b in a single step.

A.1 Mean Flow

Let $\frac{\partial z}{\partial t} = \bar{v}(z, t)$, $z_0 = x_a \sim p_a$ and let z_τ be the resulting value of z at $t = \tau$. Let us define the 'mean flow map' u giving the velocity along the straight path between z_t and $z_{t+\delta}$, or equivalently the average velocity along the marginal flow path between those points:

$$u(z_t, t, \delta) = \frac{z_{t+\delta} - z_t}{\delta} = \delta^{-1} \int_t^{t+\delta} \bar{v}(z_\tau, \tau) \mathrm{d}\tau \tag{2}$$

Taking the limit as $\delta \to 0$, we recover $u(z_t, t, 0) = \bar{v}(z_t, t)$. For $t = 0, \delta = 1$, u gives us straight paths directly transporting p_a to p_b, as shown in the right panel of Figure 2.

A parametric mean flow map $\hat{u}_\theta$ can be trained from scratch using either a derivative-integral identity [6, 7, 3] or an interval-splitting averaging objective . We will pass over the former objective, as it is less stable and requires a Jacobian-vector product to compute. The interval-splitting progressive self-distillation or 'split mean flow' objective [7, 3] takes the form:

$$\begin{aligned} \mathcal{L}_{\mathrm{SMF}} &= \mathop{\mathbb{E}}_{\substack{x_a, x_b \\ t, \delta, \lambda}} \left[\left\| \hat{u}_\theta(z_t, t, \delta) - \mathrm{sg}(\lambda u_{\mathrm{A}} + (1 - \lambda) u_{\mathrm{B}}) \right\|_2^2 \right] \\ u_{\mathrm{A}} &= \hat{u}_\theta(z_t, t, \lambda\delta) \\ u_{\mathrm{B}} &= \hat{u}_\theta(z_t + \lambda\delta u_{\mathrm{A}}, t + \lambda\delta, (1 - \lambda)\delta) \end{aligned} \tag{3}$$

where sg is the stop-gradient operator. Equation 3 only enforces consistency between a network's mean velocity predictions for different values of δ, and does not actually anchor a network's velocity predictions to a target flow. To train a split mean flow map to match a target flow, Boffi et al. [3] and Guo et al. [7] add an $\mathcal{L}_{\mathrm{FM}}$ term to the loss, substituting $\hat{v}_\theta(z, t) = \hat{u}_\theta(z, t, 0)$ in Equation 1. In practice, some portion r_v of each batch is sampled with $\delta = 0$ for $\mathcal{L}_{\mathrm{FM}}$ and the remaining $(1 - r_v)$ with $\delta > 0$ for $\mathcal{L}_{\mathrm{SMF}}$. Guo et al. [7] report that it is necessary to set $r_v \geq 0.5$ to ensure convergence. We hypothesize that this underrepresents the $\delta = 1$ condition needed for single-step inference.

A.2 Amortized Flow Matching

We want to anchor $\hat{u}_\theta$ to the target flow at $\delta = 0$ without restricting $\delta > 0$ exposure. To achieve this, we unify $\mathcal{L}_{\mathrm{FM}}$ and $\mathcal{L}_{\mathrm{SMF}}$ into the amortized flow matching objective:

$$\mathcal{L}_{\mathrm{AFM}} = \mathop{\mathbb{E}}_{\substack{x_a, x_b \\ t, \delta, \lambda}} \left[\left\| \hat{u}_\theta(z_t, t, \delta) - \mathrm{sg}(\lambda u_{\mathrm{A}} + (1 - \lambda) u_{\mathrm{B}} + \hat{u}_\theta(z_t, t, 0) - (x_b - x_a)) \right\|_2^2 \right] \tag{4}$$

The target-side terms $\hat{u}_\theta(z_t, t, 0) - (x_b - x_a)$ ensure that $\mathcal{L}_{\mathrm{AFM}}$ coincides with $\mathcal{L}_{\mathrm{FM}}$ when $\delta = 0$. Assuming the network is able to match the first moment of the target velocity, $\mathbb{E}[\hat{u}_\theta(z_t, t, 0) - (x_b - x_a)] = 0$; by linearity of expectation, $\mathcal{L}_{\mathrm{AFM}}$ then also coincides with $\mathcal{L}_{\mathrm{SMF}}$ when minimized.

B Training

We use AdamW [10] as our optimizer with learning rate $1\mathrm{e}{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.95$, and weight decay $1\mathrm{e}{-4}$. We warm up the learning rate linearly over the first 10k steps. We use minibatches containing 256 target frames sampled from 8 segments (32 decoder targets per segment), with a segment length of 512 frames. We take the stereo STFT with $n_{\mathrm{FFT}} = 2048$ and hop length 512, so each segment is $\sim$6 seconds of audio sampled at 44.1khz, and each frame is 2 channels$\times$1024 bins=2048 complex coefficients, which we model with 1d convolutional frame encoder and decoder

networks as a sequence of length 1024 with 4 real-valued channels. Frame encoder networks produce frame embeddings of 512 dimensions, which is also the condition vector dimension. We compute an exponential moving average (EMA) of θ with weight 0.999, and use the EMA parameters at inference time. We train for 1m steps, totaling 1320 GPU hours across 4 NVIDIA RTX A6000 GPUs (approximately 2 wall-clock weeks).