Real-Time Streamable Generative Speech Restoration with Flow Matching

Simon Welker ⓘ, Bunlong Lay ⓘ, Maris Hillemann ⓘ, Tal Peer ⓘ, Timo Gerkmann ⓘ, *Senior Member, IEEE*

Abstract—Diffusion-based generative models have greatly impacted the speech processing field in recent years, exhibiting high speech naturalness and spawning a new research direction. Their application in real-time communication is, however, still lagging behind due to their computation-heavy nature involving multiple calls of large DNNs.

Here, we present Stream.FM, a frame-causal flow-based generative model with an algorithmic latency of 32 milliseconds (ms) and a total latency of 48 ms, paving the way for generative speech processing in real-time communication. We propose a buffered streaming inference scheme and an optimized DNN architecture, show how learned few-step numerical solvers can boost output quality at a fixed compute budget, explore model weight compression to find favorable points along a compute/quality tradeoff, and contribute a model variant with 24 ms total latency for the speech enhancement task.

Our work looks beyond theoretical latencies, showing that high-quality streaming generative speech processing can be realized on consumer GPUs available today. Stream.FM can solve a variety of speech processing tasks in a streaming fashion: speech enhancement, dereverberation, codec post-filtering, bandwidth extension, STFT phase retrieval, and Mel vocoding. As we verify through comprehensive evaluations and a MUSHRA listening test, Stream.FM establishes a state-of-the-art for generative streaming speech restoration, exhibits only a reasonable reduction in quality compared to a non-streaming variant, and outperforms our recent work (Diffusion Buffer) on generative streaming speech enhancement while operating at a lower latency.

Index Terms—speech enhancement, speech restoration, diffusion models, generative models, flow matching, real-time

I. INTRODUCTION

Real-time speech processing refers to a type of speech processing that can be performed in an online fashion, where a model produces new parts of the processed output signal as soon as possible after new parts of the input signal arrive. Most applications allow for a fixed latency ℓ, e.g., few tens of milliseconds (ms) for voice-over-IP (VoIP) communication. Compared to offline speech processing methods, this fixed latency also requires the system to have a fixed amount of lookahead, i.e., a limited future context. Depending on the task, this may result in a performance degradation compared to an offline method which has potentially all future context available.

While the task of neural speech enhancement (SE) has traditionally been considered a discriminative task, works from recent years, in particular SGMSE+ [1], have shown that treating SE as a generative task instead can have several advantages, particularly in perceived speech quality and model generalization. Furthermore, other speech processing tasks such as bandwidth extension (BWE) or Mel vocoding are naturally well-suited to be treated as generative tasks, since significant amounts of information are missing from the input signal and must be *regenerated* given only the remaining information [2]. Other tasks which have been successfully tackled from this point of view include neural codec post-filtering [3], [4], dereverberation [1], [2], [5], short-time Fourier transform (STFT) phase retrieval (PR) [6], and binaural speech synthesis [7].

However, a key downside to such generative methods is that they are computationally intensive due to the involved numerical solvers evaluating a large deep neural network (DNN) multiple times. This seems to preclude the use of these methods for many real-time scenarios [8]. However, we show here that real-time and multi-step inference are not necessarily at odds with each other and can be realized on available consumer hardware, given that one uses appropriately buffered inference schemes and matching network architectures.

While many prior works describe their methods as *real-time* [7], [9], [10], this is often either **(1)** not supported through timings on real hardware, making it unclear whether a streaming implementation is realistically attainable, or **(2)** measured based on the real-time factor (RTF) estimated using offline processing, i.e., using an input utterance duration longer than a single frame, and dividing by the processing time the model takes for the full utterance. This neglects that, in offline processing, the inference process can make full use of parallelism across the time dimension of the input signal, as well as CUDA kernels which are typically optimized to process single large tensors as quickly as possible rather than many smaller tensors one-by-one. Such reported offline RTFs may therefore severely underestimate the actual RTF for streaming inference, making it unclear which methods are practically real-time capable in a streaming setting.

With this work, we aim to close this gap and make modern generative streaming-capable methods available to the research community. We propose Stream.FM, a real-time capable streaming generative model based on flow matching [11] that can solve various speech restoration problems with a total latency below 50 ms, bringing the high-quality capabilities of diffusion-based generative models to real-time speech processing. We expand upon our previous conference submission [12] for real-time Mel vocoding in the following ways: **(1)** we investigate five additional speech processing tasks beyond Mel vocoding: speech enhancement, dereverberation, bandwidth extension, codec post-filtering, and STFT phase retrieval; **(2)** for SE, we make use of a predictive-generative model combination as proposed in StoRM [5]; **(3)** we newly propose and investigate the use of learned Runge-Kutta ordinary differential equation (ODE) solvers to boost model output quality under a fixed computational budget; **(4)** we explore the use of model weight compression towards a flexible and favorable choice along the computation/quality tradeoff.

The predictive-generative SE model detailed here was presented in a real-time demonstration at the ITG Conference on Speech Communication [13]. Code[1] and audio examples are available online[2].

Ref. 12 saved as Mel_vecoding

II. BACKGROUND

In this section, we will detail the necessary background and notation for this work. We will denote time-domain sequences as lowercase $s, y \in \mathbb{R}^n$ and their STFT frame sequences as uppercase

All authors are with the Signal Processing Group, Dept. of Informatics, Universität Hamburg, 22527 Hamburg, Germany (e-mail: firstname.lastname@uni-hamburg.de).

$S,Y \in \mathbb{C}^{T \times F}$ with T frames, each having F frequencies. s, S refer to clean audio and y, Y refer to corrupted audio, $\hat{S}$ indicates an estimate of S, and $\hat{s} := \mathrm{iSTFT}(\hat{S})$. We use the indexing notation $Y[t]$ to indicate the t-th frame of Y, and the slicing notation $Y[t_0 : t_1]$ to indicate the range of frames in Y from t_0 to t_1 (inclusive).

A. Diffusion- and flow-based speech processing

Diffusion-based speech enhancement was originally introduced in [14], [15] and first achieved state-of-the-art quality with SGMSE+ [1]. Song et al. [16] originally proposed to define *diffusion models* for generative modeling via time-continuous *forward stochastic differential equations (SDEs)* that model a continuous mapping from data to noise. Each forward SDE has a corresponding reverse SDE resulting directly from mathematical theory [16], which can be used to map from tractable noise samples to newly generated data. The reverse SDE involves the *score function* $\nabla_x \log(p(x))$ of the data distribution $p(x)$, which is intractable in general but can be learned by a neural network called a *score model* [16].

For speech enhancement, SGMSE [15] and SGMSE+ [1] propose specific modified SDEs, modeling speech corruption by interpolating between clean speech s and corrupted speech y and incrementally adding Gaussian white noise. To produce enhanced speech, one draws a Gaussian white noise sample, adds it to the corrupted speech, and then numerically solves the reverse SDE starting from this point. We refer the reader to [1] for the full detailed description of the training and inference.

Later works introduce flow matching (FM) [11], which is closely related to diffusion models but takes a perspective based on ODEs rather than SDEs. The idea is to learn a model to transport samples from a tractable distribution $q_0(X_0)$, e.g., a multivariate Gaussian, to an intractable data distribution $q_1(X_1) = p_{\mathrm{data}}$ by solving the ODE

$$\frac{\mathrm{d}}{\mathrm{d}\tau}\phi(\tau,X) = u(\tau,\phi(\tau,X)), \quad \phi(0,X) = X_0 \tag{1}$$

starting from a random sample $X_0 \sim q_0$. $\phi : [0,1] \times \mathbb{R}^n \to \mathbb{R}^n$ is called the *flow* with the associated *time-dependent vector field* $u : [0,1] \times \mathbb{R}^n \to \mathbb{R}^n$, generating a *probability density path* $p_\tau : \mathbb{R}^n \to \mathbb{R}_+$, i.e., a probability density function that depends on an artificial time coordinate $\tau \in [0,1]$ with $p_0 = q_0$ and $p_1 = q_1$. One can learn a neural network v_θ called a *flow model* to approximate $u(\tau,\cdot)$ with the *conditional flow matching* loss [11, Eq. 9]:

$$\mathcal{L}_{\mathrm{CFM}} = \mathbb{E}_{X,\tau,(X_\tau|X)}\left[\|v_\theta(\tau,X_\tau) - u(\tau,X_\tau|X)\|_2^2\right] \tag{2}$$

where $\tau \sim \mathcal{U}(0,1), X \sim q_1$ is clean data sampled from a training corpus, $\mathbb{E}_a[b(a)] = \int_{-\infty}^{\infty} b(a)p(a)\mathrm{d}a$ denotes the expectation of $b(a)$ with respect to the distribution $p(a)$ of a, and θ denotes the set of DNN parameters. Note that, during training, the expectation is approximated by an empirical average over the training data. The objective Eq. (2), where u is *conditional* on the clean X, has the same gradients as an intractable objective where u is unconditional [11, Eq. 5], and results in the correct probability path $p_\tau(X_\tau)$ and flow field $u(\tau,X_\tau)$ [11, Sec. 3.1, 3.2]. FlowDec [4] showed that, similar to SGMSE [15], [1], FM can be modified to interpolate between the distributions of clean and corrupted data, enabling the use of FM for generative signal enhancement. Concretely, we choose the following path, which linearly interpolates from a clean sample S to a corrupted sample Y and adds increasing amounts of Gaussian noise:

$$p_\tau(X_\tau|S,Y) := \mathcal{N}(X_\tau; (1-\tau)Y + \tau(S-Y), \Sigma_\tau) \tag{3}$$

where $\Sigma_\tau = ((1-\tau)\Sigma_y + \tau\Sigma_{\min})^2$ and $\Sigma_y, \Sigma_{\min}$ denote covariance matrices which we assume to be scalar or diagonal. We denote a scalar covariance by σ_y^2, i.e., $\Sigma_y = \sigma_y^2 I$ with the identity matrix I. The flow model v_θ can be trained via the *joint flow matching* loss [4]:

$$\mathcal{L}_{\mathrm{JFM}} = \mathbb{E}_{\tau,(S,Y),(X_\tau|S,Y)}\left[\|v_\theta(\tau,X_\tau,Y) - (X_1-X_0)\|_2^2\right] \tag{4}$$

where (S,Y) are paired data of clean and corrupted audio sampled from a training corpus. Note that we let each set of X_1, X_0 and X_τ share a single Gaussian noise sample $\varepsilon \sim \mathcal{N}(0,I)$, as a simple form of training variance reduction through minibatch coupling [17]. While [4] did not use $\Sigma_{\min}$, we reintroduce it here similar to [11] as we found this to slightly increase training stability.

The trained network v_θ can then be plugged into the flow ODE Eq. (1) in place of u, and this neural ODE can be solved numerically starting from a sample $X_0 \sim p_0$ to perform signal enhancement. This typically requires multiple calls of the network v_θ, where the number of calls is referred to as number of function evaluations (NFE).

B. Latency definitions

We define the algorithmic latency ℓ_{alg} as the attainable latency on infinitely fast hardware, and the overall latency of a system as $\ell_{\mathrm{tot}} = \ell_{\mathrm{alg}} + \ell_{\mathrm{proc}}$, where ℓ_{proc} is the processing time per frame. For a frame-causal STFT-based method with causal (right-hand) window alignment such as Stream.FM, the algorithmic latency ℓ_{alg} is the synthesis window length W_{syn} minus one sample divided by the sampling rate f_s, i.e., $\ell_{\mathrm{alg}} = \frac{W_{\mathrm{syn}}-1}{f_s}$ [18], [19]. In typical frame-by-frame processing implementations, ℓ_{proc} is effectively given exactly via the synthesis frame hop H_{syn}, due to the synthesis side waiting for each synthesis frame hop to be complete before producing audio samples. This allows the processing model up to $\ell_{\mathrm{proc}} \le \frac{H_{\mathrm{syn}}}{f_s}$ of processing time. We use the same configurations for analysis and synthesis here, hence the analysis window $W_{\mathrm{ana}} = W_{\mathrm{syn}}$ and hop $H_{\mathrm{ana}} = H_{\mathrm{syn}}$, so $\ell_{\mathrm{tot}} = \frac{W_{\mathrm{ana}}-1+H_{\mathrm{ana}}}{f_s}$ for our models.

C. Explicit Runge-Kutta ODE solvers

To solve ODEs such as Eq. (1) numerically, a simple method is the Euler method [20]. It starts from X_0 and $\tau = 0$, discretizes the time interval $\tau \in [0,1]$ into uniformly spaced points, and iterates the following equation for N iterations until $\tau = 1$ using $\Delta\tau = \frac{1}{N}$:

$$X_{\tau+\Delta\tau} := X_\tau + \Delta\tau \cdot u(\tau,X_\tau) \approx X_\tau + \Delta\tau \cdot v_\theta(\tau,X_\tau), \tag{5}$$

where we inserted the learned flow model v_θ for u in the approximate equality. The Euler solver thus requires NFE $= N$ calls of the DNN v_θ, but has a relatively large approximation error for a given NFE budget compared to more flexible solvers [20]. For high-quality real-time inference, we want our solvers to have a fixed small NFE and attain high quality under this budget. To this end, we use explicit Runge-Kutta solvers [20], which can be parameterized by a set of coefficients $\mathbf{A} \in \mathbb{R}^{r \times r}, \mathbf{b} \in \mathbb{R}^r, \mathbf{c} \in \mathbb{R}^r$ [21] where r is the NFE per step, and $\mathbf{A}$ is a strictly lower triangular matrix so the solver is explicit [20]. While these coefficients are usually predefined, in Section III-E, we propose to learn task-optimized $\{\mathbf{A},\mathbf{b},\mathbf{c}\}$ from data. The iterated equation of such a solver is:

$$G_1 := v_\theta(\tau,X_\tau) \tag{6}$$

$$G_{i>1} := v_\theta\left(\tau + c_i\Delta\tau, X_\tau + \Delta\tau\sum_{j=1}^{i-1} a_{ij}G_j\right), \tag{7}$$

$$X_{\tau+\Delta\tau} := X_\tau + \Delta\tau\left(\sum_{i=1}^{q} b_i G_i\right) \tag{8}$$

where $a_{ij} = \mathbf{A}[i,j]$. We set $N = 1$ and $\Delta\tau = 1$ in the following so that we perform a single evaluation of Eqs. (6) to (8) at $\tau = 0$, stepping directly to $\tau = 1$. This yields NFE $= r$, with each $G_i \in \{G_1,...,G_r\}$ contributing one evaluation of v_θ.

D. Related work

In 2020, Défossez et al. [22] proposed a predictive waveform-domain model for causal real-time SE, which we refer to as *DEMUCS* here. The recent work aTENNuate [23] is a predictive state-space model for real-time SE. DEMUCS and aTTENuate operate at a similar, slightly larger algorithmic latency than Stream.FM, but no streaming implementation of aTTd ENuate has been published, hence we only compare against an offline variant of it. Dmitrieva and Kaledin [24] recently proposed HiFi-Stream for streaming SE, which uses a block-wise inference scheme at the cost of possible block-edge discontinuities and a higher latency.

Richter et al. [25] first proposed the use of causal DNNs for diffusion-based SE and achieved convincing results, but did not target real-time capability on real hardware, and used heavy down- and upsampling along time which leads to large latencies [26], [27].

Lay et al. [8] proposed the Diffusion Buffer (DB), which realizes a real-time streaming SE diffusion model on consumer hardware. DB couples the diffusion time τ to physical time t in a fixed-size frame buffer. The buffer is progressively denoised frame-by-frame using one DNN call each time, and each time the oldest—then fully denoised—frame in the buffer is output. This realizes a fixed algorithmic latency equal to the total buffer duration. A key downside of DB is that the algorithmic latency, while sub-second, is still relatively large: a buffer of fewer than 20 frames did not yield satisfying quality, resulting in an effective minimum algorithmic latency of 320 ms in their configuration and around 180 ms in a follow-up preprint [27].

Liang et al. [7] describe a similar approach to streaming FM as detailed here, but only investigate monaural-to-binaural speech upmixing whereas we treat six different speech tasks. The authors determine RTFs on a 0.683-second snippet which may underestimate streaming RTF significantly, see Section V-F. To ensure reproducibility and enable future developments, we provide a more extensive description of the buffered multi-step inference scheme, see Section III-A, as well as a public code repository.

A concurrent work [9] investigates streaming FM models for speech restoration, similarly using a U-Net architecture without time-wise downsampling as proposed here. The authors report an algorithmic latency of 20 ms, but do not provide real-hardware timings, making it unclear whether a streaming model can be practically realized.

III. METHODS

A. Multi-step streaming diffusion

When we use a DNN in an inference setting for diffusion- or flow-based signal enhancement, processing a noisy frame sequence Y into a clean estimate $\hat{S}$, we use ODE or SDE solvers. For FM, ODE solvers are used. Within these solvers, we call the learned DNN, v_θ, N times in sequence, starting from a noisy sequence $Y_0 = Y + \varepsilon$ with some independent noise sample ε. We assume here that v_θ has a finite receptive field of size R and is frame-causal, i.e., there exists no t such that the frame $\hat{S}[t]$ depends on any input frame $Y[t+n], n > 0$. For ease of illustration, we use the equidistant Euler solver Eq. (5) for ODEs with N steps in the following and assume the model v_θ was
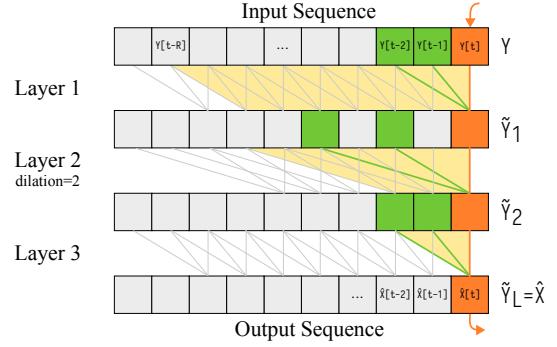

Fig. 1: Inference for one new frame (orange) in a simplified frame-causal DNN. While the output frame has a receptive field size (yellow) of 9 in the input, only 3 frames must be evaluated in each layer since all required past results (green) can be stored in a buffer $\mathbf{B}$.

trained with the joint flow matching (JFM) objective Eq. (4). This inference process generates intermediate partially-denoised sequences $Y_1, Y_2,...,Y_{N-1} \in \mathbb{R}^{T \times F}$ and finally the fully denoised sequence Y_N, which each depend on their respective prior sequence as:

$$Y_1[t] = Y_0[t] + \Delta\tau \cdot v_\theta(0, Y_0[t-R:t]), \tag{9}$$

$$Y_2[t] = Y_1[t] + \Delta\tau \cdot v_\theta(\Delta\tau, Y_1[t-R:t]), \quad ..., \tag{10}$$

$$Y_N[t] = Y_{N-1}[t] + \Delta\tau \cdot v_\theta((N-1)\Delta\tau, Y_{N-1}[t-R:t]), \tag{11}$$

where the first parameter $\tau \in [0,1]$ to $v_\theta(\tau, \cdot)$ is the continuous diffusion time and $\Delta\tau = \frac{1}{N}$ is the discretized diffusion timestep. After completing this process, Y_N is fully denoised and we can treat it as the clean sequence estimate $\hat{S} := Y_N$. Since Y_0 is just the addition of Y and Gaussian noise ε, recursively collapsed, this implies the following effective dependency of $\hat{S}$ on Y:

$$\hat{S}[t] = V_\theta(Y[t - N \cdot R : t]), \tag{12}$$

where V_θ represents the evaluation of the whole procedure in Eqs. (9) to (11).Given this, it may seem that real-time streaming diffusion inference with a large DNN v_θ is not viable: to generate a single output frame $\hat{S}[t]$ we must run the entire diffusion process backwards, doing so we must process all NR frames, and we cannot exploit time-wise parallelism for efficiency since frames come in one-by-one.

One approach to circumvent this is the Diffusion Buffer [8], see Section II-D. In a follow-up preprint [27], the authors show that with a different training loss and direct-prediction inference, frames can be output earlier to reduce latency. However, this inference approximates the reverse process that the DNN was trained for, with the quality of the approximation decreasing as one decreases the latency. This results in significant output quality degradations when targeting the same algorithmic latency as our method, see Table II ($d = 0$), suggesting the need for a different approach in low-latency settings.

B. Efficient real-time streaming diffusion inference

We provide a scheme that can perform streaming frame-wise inference while incurring no algorithmic latency and avoiding any redundant computations. For this purpose, we assume that v_θ has a specific form, namely that of a causal convolutional neural network containing L stacked causal convolution layers $\mathcal{C}_l$ each with stride 1, dilation d_l, and a kernel $K_l \in \mathbb{R}^{c_{o,l} \times c_{i,l} \times k_l}$ of size k_l with input channels $c_{i,l}$ and output channels $c_{o,l}$. The DNN may also contain any operations that are point-wise in time, e.g., nonlinearities and specific

types of normalization layers. Each such layer has receptive field size $R_l = (k_l - 1)d_l + 1$, and the overall receptive field size of v_θ is $R = 1 + \sum_{l=1}^{L} R_l - 1$. The first key observation here is that a causal convolution of kernel size k and dilation d depends on its input sequence only at k time points in $Y[t-(k-1)d:t]$, and immediately when its inputs are available, its output is fixed. Even when stacking multiple convolutions, the past outputs of $\mathcal{C}_{l-1}$ never need to be recomputed. This lets us perform local caching throughout the DNN, where every layer $\mathcal{C}_l$ keeps an internal rolling buffer $B_l \in \mathbb{R}^{c_{i,l} \times (R_l - 1)}$ which contains only $R_l - 1$ past input frames, with $R_l - 1 \ll R$.

When we receive a new input frame $Y[t]$ to the DNN after previously having seen the input frames $Y[0],...,Y[t-1]$, we only need to evaluate each layer's convolution kernel K_l at the single latest time index t on the buffer B_l concatenated with the newest frame, producing a single output frame $\tilde{Y}_l$ per layer, see Fig. 1:

$$\tilde{Y}_1[t] = \varphi_1(K_1 \star \underbrace{Y[t-R_1:t-1}_{=B_1},t]), \tag{13}$$

$$\tilde{Y}_2[t] = \varphi_2(K_2 \star \underbrace{\tilde{Y}_1[t-R_2:t-1}_{=B_2},t]), \quad ..., \tag{14}$$

$$\tilde{Y}_L[t] = \varphi_L(K_L \star \underbrace{\tilde{Y}_{L-1}[t-R_L:t-1}_{=B_L},t]), \tag{15}$$

where $\star$ is the (possibly dilated) convolution evaluated only for a single output frame, yielding an output in $\mathbb{R}^{c_{o,l} \times 1}$, and each φ_l represents arbitrary extra operations between convolutions assumed to be point-wise in time. We can then set $\tilde{Y}_L[t]$ as the output frame of this single DNN call. Note that all k_l, d_l can be chosen arbitrarily and independently without affecting the efficiency of this scheme. This idea has also been explored for real-time video processing [26]. The authors note that strided convolutions incur a delay to their respective downstream module and hence incur algorithmic latency, so we avoid their use here and set all time-wise convolution strides to 1.

The second key observation is that this scheme can be straightforwardly extended to diffusion/flow model inference with multiple DNN calls by tracking N independent collections of cache buffers, one for each DNN call, resulting in $(N \cdot L)$ cache buffers in total. We denote this collection of buffers as $\mathbf{B}$ where $\mathbf{B}_{n,l} \in \mathbb{R}^{(R_l - 1) \times F}$, $n = 1,...,N$ and $l = 1,...,L$. Starting from the initial value $Y_0[t]$, we can then process the following sequences $\tilde{Y}_{n,l}$ for each n-th model call and l-th layer:

$$\left.\begin{aligned}\tilde{Y}_{1,1}[t] &= \varphi_1(K_1 \star [\mathbf{B}_{1,1}, Y_0[t]]), \quad ..., \\ Y_1[t] := \tilde{Y}_{1,L}[t] &= \varphi_L(K_L \star [\mathbf{B}_{1,L}, \tilde{Y}_{1,L-1}[t]]),\end{aligned}\right\} \text{Call 1}$$

$$\left.\begin{aligned}\tilde{Y}_{2,1}[t] &= \varphi_1(K_1 \star [\mathbf{B}_{2,1}, Y_1[t]]), \quad ..., \\ Y_2[t] := \tilde{Y}_{2,L}[t] &= \varphi_L(K_L \star [\mathbf{B}_{2,L}, \tilde{Y}_{2,L-1}[t]]),\end{aligned}\right\} \text{Call 2}$$

$$\vdots$$

$$Y_N[t] := \tilde{Y}_{N,L}[t] = \varphi_L(K_L \star [\mathbf{B}_{N,L}, \tilde{Y}_{N,L-1}[t]]),$$

where $[\cdot,\cdot]$ denotes concatenation along the time dimension. For notational simplicity, we conceptually include the ODE solver step, see Eqs. (9) to (11), in each φ_L. After each operation, the respective buffer $\mathbf{B}_{n,l}$ is shifted to drop the oldest frame and the new frame is inserted on the right, see Fig. 1. The buffers are all initialized with zeros, matching the training if all convolution layers use zero-padding.

This scheme now performs the same computations as an offline implementation, only spread across physical time. Importantly, it

yields exactly the same result—up to tiny numerical floating-point errors—as feeding the entire sequence to the causal model at once in an offline, batch-wise fashion. This fact enables the use of standard batched training and the subsequent approximation-free use of the trained weights for streaming inference, in contrast to, e.g., the Diffusion Buffer when outputting frames early for lower latency [27].

For future research, we note that this scheme can also be used with SDEs with one minor modification. For SDE solvers, a new noise sample ε_n is drawn at every solver step $n = 1,...,N$. By redefining

$$Y_n[t] := \tilde{Y}_{n,L}[t] + \tilde{\sigma}_n \varepsilon_n[t] \tag{16}$$

where $\tilde{\sigma}_n$ is the level of noise added back in according to the SDE solver and diffusion schedule, and treating this as another point-wise operation in time, no other changes to the inference are required. However, we only investigate ODE solvers in the following since we work with FM models.

C. DNN architecture

We design a custom frame-causal CNN without strided convolutions, as a modified variant of the NCSN++ architecture [16]. NCSN++ is a non-causal 2D U-Net architecture with multiple residual blocks per level, a progressive down/upsampling path, as well as non-causal attention layers. [1] provides a detailed description of the specific NCSN++ configuration often used for SE. As in [1], our DNN receives the audio signal in the form of a complex-valued STFT (a 2D time-frequency signal), with the real and imaginary parts mapped to two real-valued DNN input/output channels. We perform the following modifications and simplifications:

1) We perform down- and upsampling only along frequency, never along time. To increase time context, we replace time-strided with time-dilated causal convolutions (dilation of 2).
2) We keep the FIR filters for anti-aliased down- and upsampling [16] along frequency, but remove the FIR filtering along time.
3) We remove all attention layers. It is possible to include causal attention layers, but we do not follow this here for simplicity.
4) We replace the original GroupNorm [28] with a custom sub-band grouped BatchNorm (*SGBatchNorm*), inspired by [29]. SGBatchNorm performs joint grouping along channels and frequencies and normalizes each group. We use four frequency groups, and the channel grouping of NCSN++ [16]. Prior work [25] used time-cumulative group normalization, which determines statistics frame-by-frame as a time-varying IIR filter. In contrast, SGBatchNorm determines running-average batch statistics during training which are frozen and thus time-invariant during inference.
5) In contrast to [25], we keep the progressive input down/upsampling paths [16], by simply using causal convolutions and the SGBatchNorm described above.
6) We remove an extra residual block on each level on the upward path, which was used to process a skip connection from the *input* of each corresponding block on the downward path. We only keep the skip connection from the *output* of the corresponding downward block.
7) To fuse skip connections, we always employ addition, while NCSN++ [16] used addition on the downward and channel concatenation on the upward path. Our proposed addition significantly reduces the number of features on the upward path.

8) We do not use an exponential moving average (EMA) of the weights, since we found our models to perform well without it.

D. Predictive-generative speech enhancement

Inspired by Lemercier et al. [5], for our SE task we use a joint predictive-generative approach, where an *initial predictor* network D_η with parameters η is first trained to map Y to an initial estimate $Z := D_\phi(y)$, and Z is then used instead of Y for defining and training the generative model. For D_η, we use the same DNN architecture as for the FM model, but remove all diffusion time conditioning layers. To train D_η, we use the following loss:

$$\mathcal{L}_{\text{pred}}(z,s) := \frac{1}{2}\|z-s\|_1 + \frac{1}{2}\mathcal{L}_{\text{MR-STFT}}(z,s), \tag{17}$$

$$\mathcal{L}_{\text{MR-STFT}}(z,s) := \sum_{w=1}^{N_w} \big\| \, |\text{STFT}_w(z)| - |\text{STFT}_w(s)| \, \big\|_1, \tag{18}$$

where $z := \text{iSTFT}(Z)$ and $\mathcal{L}_{\text{MR-STFT}}$ is a multi-resolution magnitude STFT L_1 loss similar to [30], using a set of STFT_w with different windows w with N_w window configurations. We use $N_w = 4$ Hann windows with $W_{\text{ana}} \in \{256, 512, 768, 1024\}$ and 50% overlap.

E. Custom learned low-NFE ODE solvers

Given a fixed budget of DNN evaluations per frame (NFE), we develop specialized learned Runge-Kutta ODE solvers (see Section II-C) to optimize the achievable quality without model retraining or finetuning, at virtually no increase of computations during inference. We train the scheme's parameters $\{\mathbf{A}, \mathbf{b}, \mathbf{c}\}$, given a pretrained flow matching model v_θ with frozen weights θ, by solving the ODE (1) with the current Runge-Kutta (RK) scheme and treating the final output as the clean speech estimate via $\hat{s} := \text{iSTFT}(X_1)$. To optimize $\{\mathbf{A}, \mathbf{b}, \mathbf{c}\}$, we backpropagate through the whole solved ODE path and multiple DNN calls to determine the gradients, using the following loss:

$$\mathcal{L}_{\text{RK}} := -\text{SpeechBERTScore}(\hat{s},s) \tag{19}$$

$$+ 0.001 \cdot \text{MRLogSpecMSE}(\hat{s},s) \tag{20}$$

where the motivation for using negative SpeechBERTScore [31] is to reduce phonetic hallucinations/confusions that generative methods can suffer from [25], [32]. MRLogSpecMSE refers to a multi-resolution log-magnitude STFT mean squared error (MSE) loss similar to [30] which we use to increase fine high-frequency detail, which FM models tend to lose in low-NFE settings [4], see the log-spectral distance (LSD) metric in Table II. For this loss, we use window sizes $W_{\text{ana}} \in \{320, 512, 640\}$ and 75% overlap.

We ensure by construction of the optimized parameters that $\sum_j a_{ij} = c_i, \sum_i b_i = 1$, and $0.05 \leq b_i \leq 1$ for all $1 \leq i,j \leq r$ to ensure that all model calls contribute to the final estimate. We clip $c_i \leq 0.85$ to avoid evaluating the FM model close to $\tau \approx 1$ where the training target (4) is unstable, and put a quadratic penalty loss on each a_{ij} outside the range $[-2,2]$. We train $\{\mathbf{A}, \mathbf{b}, \mathbf{c}\}$ with the Adam optimizer at a learning rate of 10^{-3}, a batch size of 10, and 2-second audio snippets, for 25,000 steps on our training dataset. For the 4 NFE available under our runtime budget in the SE task, we initialize $\{\mathbf{A}, \mathbf{b}, \mathbf{c}\}$ from Kutta's four-stage 3/8 scheme [20]. For all other tasks, we have five NFE available due to the lack of a predictive DNN, and we use one Ralston-2 step followed by one Ralston-3 step [33] for

TABLE I: Corruption and feature representation operators for our speech restoration tasks. $*$ indicates convolution, $(\cdot \downarrow\downarrow f)$ and $(\cdot \uparrow\uparrow f)$ indicate down/upsampling by a factor f, respectively, Dec/Enc refer to the encoder/decoder of an audio codec, and M is a per-frame STFT→Mel matrix with $M^\dagger$ as its Moore-Penrose pseudoinverse. $+0j$ indicates embedding of real numbers into the complex plane.

Task	Problem	Corruption model
1	Speech Enhancement	$Y = \text{STFT}\{x+n\}$
2	Dereverberation	$Y = \text{STFT}\{x*h\}$
3	Codec Post-Filtering	$Y = \text{STFT}\{\text{Dec}(\text{Enc}(x))\}$
4	Bandwidth Extension	$Y = \text{STFT}\{(x \downarrow\downarrow f) \uparrow\uparrow f\}$
5	STFT Phase Retrieval	$Y = |\text{STFT}\{x\}| + 0j$
6	Mel Vocoding	$Y = \big| M^\dagger \big(M|\text{STFT}\{x\}|\big)\big| + 0j$

initialization. Our learned schemes do not guarantee a convergence order since their coefficients do not necessarily follow the required algebraic conditions [20, Eq. (2.21)], which we argue is acceptable since our solvers do not need to be general-purpose. We list all RK parameters learned for each task in the supplementary material.

F. Model compression through weight decoupling

We make use of DNN compression methods to reduce the computational costs of each network call. Using this, we aim to improve the quality given a fixed computational budget, or to decrease the number of computations at some reasonable decrease in output quality. There are various methods to this end including weight pruning, quantization, or model distillation, but here we specifically follow the decoupling approach of Guo et al. [34] to approximate the large 2D convolution weight tensors $\mathbf{W} \in \mathbb{R}^{n_o \times n_i \times k_h \times k_w}$ with output/input channels n_o, n_i and kernel size $k_h \times k_w$, since such convolutions incur most of the computational effort in our networks.

The authors first show that a 2D convolution can be losslessly decomposed into one depthwise and one pointwise convolution without increasing the algorithmic complexity. They further show a direct connection between these two layers and the singular value decomposition (SVD) of the weights for each input channel, $\mathbf{W}_{:,i} \in \mathbb{R}^{n_o \times k_h \times k_w}$, reshaped to a matrix $\widetilde{\mathbf{W}}_{:,i} \in \mathbb{R}^{n_o \times (k_h k_w)} = USV^\top$. U and $SV^\top$ are determined via the SVD and map directly to the weight tensors for the depthwise and the pointwise convolution, respectively. This leads to a SVD-based compression of pretrained convolution layers, by truncating the SVD to rank $J \leq K = \min(n_o, k_h k_w)$ where $J = K$ indicates no compression. The truncated matrices also map directly to the weights of smaller depthwise and pointwise convolution layers.

We apply this weight compression to all 3×3 Conv2d layers with $n_o \geq 9$ in our network. As also proposed in [34] we then fine-tune the compressed models for 25,000 training steps. We deviate slightly from the authors' representation, choosing $U\sqrt{S}$ for the depthwise and $\sqrt{S}V^\top$ for the pointwise weights to spread the singular values across the two layers and improve fine-tuning stability.

IV. EXPERIMENTS

A. Speech restoration tasks

We provide here a description of the six speech restoration tasks we investigate. See Table I for the full list of corruption and feature representation operators we use in practice.

1) Speech Enhancement: For our speech enhancement (SE) task, the signal corruption model is $y = s + n$, where s is the clean audio and n is some uncorrelated background noise. As the dataset, we use EARS-WHAM v2[3] [35], downsampled to 16 kHz. We also use the EARS-WHAM v2 clean utterances as the dataset for all following tasks except dereverberation.

2) Dereverberation: We investigate speech dereverberation using the EARS-Reverb v2 dataset [35]. The signal corruption model is $y = s * h$, where $*$ indicates time-domain convolution and $h \in \mathbb{R}^{l_h}$ is a sampled room impulse response (RIR).

3) Codec Post-Filtering: Inspired by ScoreDec [3] and FlowDec [4], we investigate the use of Stream.FM as a post-filter for a low-bitrate speech codec. FlowDec [4] introduces FM-based generative post-filtering, proposing a non-adversarially trained variant of the neural codec DAC [36]. Since DAC is non-causal and computationally expensive, we use the Lyra V2 codec[4] instead, which is built for streaming speech coding on consumer devices. Lyra V2 produces one frame every 20 ms at our chosen bitrate of 3.2 kbit/s. To align the codec frames with our model, we change the STFT parameters to use 40 ms windows and 20 ms hops.

4) Bandwidth Extension: We train a model to perform BWE from speech downsampled to sampling frequencies of 8 kHz and 4 kHz, leading to a frequency cutoff at 4 kHz and 2 kHz, respectively. To generate y, we downsample each s randomly to either 8 or 4 kHz.

5) STFT Phase Retrieval: Peer et al. have shown with DiffPhase [6] that SGMSE+ [1] can be modified to solve an STFT PR task, resulting in very high reconstruction quality. We extend this idea to our streaming setting, using only 50% STFT overlap instead of the 75% overlap used in DiffPhase. We compare our method against a family of streaming STFT PR algorithms proposed by Peer et al. [37].

6) Mel Vocoding: As shown in [12], the ideas of DiffPhase [6] can be extended to streaming Mel vocoding through a small change in the corruption model. Instead of treating the phaseless STFT magnitudes $|X|$ as the corrupted signal [6], we additionally subject them to a lossy Mel compression as follows:

$$X_{\text{mel}}[t] = \left| M^\dagger \left(M \left| X[t] \right| \right) \right| + 0j \qquad (21)$$

where $M \in \mathbb{R}^{F_{\text{mel}} \times F_{\text{STFT}}}$ is the Mel matrix mapping STFT frames to Mel frames, $M^\dagger$ is its Moore-Penrose pseudoinverse, and $X[t]$ denotes the single magnitude spectrogram frame at frame index t. We follow the Mel configuration of HiFi-GAN [38] in the 16 kHz variant from SpeechBrain [39], but to reduce the latency, we use 32 ms windows instead of 64 ms while keeping the 16 ms hop length.

B. Data and data representation

We use the EARS dataset [35] as the basis of all our problem variants and model trainings, resampled to a sampling frequency of $f_s = 16$ kHz. Unless otherwise noted, we use an STFT with a 512-point periodic $\sqrt{\text{Hann}}$-window (32 ms), a 256-point hop length (16 ms, 50% overlap), and magnitude compression with exponent $\alpha = 0.5$ as in [15], [1]. Different from these works, we use an orthonormal STFT and do not apply an additional scaling. Since a 512-point window leads to 257 frequency bins, for ease of DNN processing, we discard the Nyquist band to retrieve frames with 256 frequency bins, and pad it back with zeros before applying the inverse STFT.

For SE, similar to [25], during training we peak-normalize s and y independently and apply a random negative gain between -12 and 0 dB to y, so that the model learns to perform automatic gain control. For all other tasks, we normalize s and y jointly based on the peak magnitude of y, assuming that a roughly constant input-output level relationship is available in these tasks. As the FM process hyperparameter Σ_y (3), we empirically set scalar $\sigma_y = 0.05$ for SE, $\sigma_y = 0.25$ for STFT PR and Mel vocoding, and $\sigma_y = 0.35$ for dereverberation and codec post-filtering. For BWE, we follow [4] and determine a heuristic per-frequency-band diagonal covariance matrix Σ_y to avoid adding noise in the preserved low-frequency bands and to allow easier regeneration of the low-energy high-frequency bands. We set $\sigma_{\min} = 0.001$ for all tasks except BWE where $\Sigma_{\min} = 0.001 \cdot \Sigma_y$.

C. DNN configuration and training

For Stream.FM, we parameterize our architecture described in Section III-C with two residual blocks per level and four U-Net levels with [128, 256, 256, 256] channels, respectively, leading to 27.9 M parameters when used as an FM backbone DNN. For the initial predictor network D_ϕ in the SE task, we remove all time-conditioning layers and reduce the complex input channels from two to one, resulting in 24.6 M parameters. For the non-causal FM baseline models, we use the original NCSN++ architecture [16], here also parameterized with four U-Net levels in the same channel configuration and also two residual blocks per level (38.7 M parameters).

For each task, we train an FM model v_θ using Eq. (4) for 150,000 steps, 2 GPUs, and 2-second random snippets with a batch size of 12 per GPU using the SOAP optimizer [40] which was recently found to perform well for diffusion model training [41]. We use a cosine annealing learning rate schedule with a maximum learning rate of $\lambda = 5 \times 10^{-4}$ and linear warmup for the first 1,000 steps, clamping the scheduled λ to a minimum value of 10^{-6}. We use gradient clipping with a maximum norm of $\|\nabla\|_{\max} = 3$ for all tasks except for SE with $\|\nabla\|_{\max} = 1$ and codec post-filtering with $\|\nabla\|_{\max} = 5$, based on empirical gradient norm inspection.

To train the SE model, we first train the initial predictor D_ϕ for 150,000 steps using Eq. (17) with the SOAP optimizer [40] at a constant learning rate of $\lambda = 3 \times 10^{-3}$. We then freeze the initial predictor during FM model training. For SE, we also train a lower-latency joint predictive-generative Stream.FM model, reconfiguring the STFT with 16 ms frames and 8 ms hops for a total latency of only 24 ms, keeping all other settings the same.

D. Evaluation

1) Baseline methods: As streaming-capable baseline methods for SE, we use DEMUCS [22], DeepFilterNet3 [42], HiFi-Stream [24], and Diffusion Buffer [27]. We further include the aTENNuate [23] method for which no streaming model variant has been published, hence we only evaluate its offline performance. We use the official streaming implementations for DEMUCS[5], HiFi-Stream[6] and DeepFilterNet3[7], and the official Python package for aTENNuate[8].


(a) Speech enhancement (b) Bandwidth extension
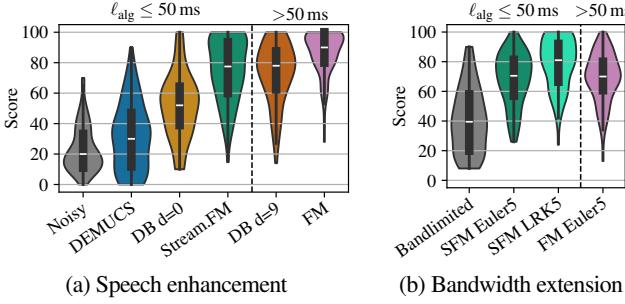
Fig. 2: Violin plots of the scores listeners assigned to examples from each method in the listening experiments for **(a)** speech enhancement and **(b)** bandwidth extension. *SFM* is Stream.FM, *FM* is the flow matching baseline, and *DB* is the Diffusion Buffer [27]. Note that FM has $\ell_{alg} = \infty$ and DB with $d = 9$ has $\ell_{alg} \approx 180$ ms.

2) Latency determination: Theoretical derivations of latencies can be misleading in complex systems. We thus determine ℓ_{alg} in an end-to-end fashion: At every single index in a 2-second example input waveform, we set the value (only at this index) to IEEE 754 `NaN` (not a number), and let each model produce an enhanced output waveform. `NaN`s are infectious, i.e., any operation involving a `NaN` produces a `NaN` output, a fact we use to detect dependencies of every output sample on each input sample. We sweep across all input indices and determine the maximum index difference between the affected input index and the first index for which the output is also `NaN`, giving us ℓ_{alg}. We report $\ell_{alg} = \infty$ if we find that the latency grows arbitrarily large with the input waveform length.

3) Metric evaluation: As intrusive metrics, we report wideband PESQ [43], ESTOI [44], SI-SDR [45] and LSD[9] [46]. We further report the word error rate (WER) using the QuartzNet15x5Base-En model [47] from the NeMo toolkit [48] as the speech recognition backend, using the model's transcripts of the clean audios as the reference. As non-intrusive metrics, we report NISQA [49], WVMOS [50], and DistillMOS [51] which we refer to as *DiMOS* for brevity.

4) Listening experiments: We conduct two MUSHRA-like listening experiments [52], one for SE and one for BWE, each with 12 participants who gave informed consent. We asked participants to rate the overall quality (0–100) of 8 randomly sampled utterances, as reconstructed by each method. For SE, we compare predictive-generative Stream.FM (SFM) against non-streaming FM, both with 4 Euler steps, Diffusion Buffer [27] at $d \in \{0, 9\}$, and DEMUCS [22]. For BWE, we compare SFM against non-streaming FM, both with 5 Euler steps, against SFM with a learned RK5 solver. We use the noisy/downsampled utterance as the low anchor for SE/BWE, respectively.

5) Runtime performance evaluation: We perform all model runtime evaluations using a single laptop with an *NVIDIA RTX 4080 Laptop* GPU. We measure the number of floating-point operations (FLOPs) using the PyTorch `torch.utils.flop_counter` module, and the wall clock timings using `torch.cuda.Event`.

V. RESULTS AND DISCUSSION

A. Speech enhancement

We show the metric results for the SE task in Table II. We find that all SFM variants attain the highest values among the

[9]as in https://github.com/haoheliu/ssr_eval, 32 ms Hann window, 75% overlap.

streaming-capable baselines in almost all metrics and exhibit particularly strong improvements in non-intrusive DistillMOS, WVMOS and NISQA, though WVMOS rates DEMUCS [22] on-par. When the Diffusion Buffer (DB) baseline [27] is configured to use the same low algorithmic latency ($d = 0$) as SFM, where DB uses only a single diffusion step per frame, SFM with four Euler steps shows strong advantages over DB. SFM also performs similar to or better than the higher-latency DB variant ($d = 9$) with 10 diffusion steps, the main model proposed in [27]. HiFi-Stream [24] and aTTENuate [23] do not achieve clear improvements over the noisy mixtures and in particular worsen the WER. This may indicate an overfitting to their respective training datasets, which can be seen as reasonable under their small DNN parameter budgets.

Comparing different ODE solvers for Stream.FM, we see that a single Euler step (*Euler1*) shows excellent performance in this task, but 4 Euler steps (*Euler4*) slightly improve most non-intrusive metrics and LSD, at some decrease in intrusive metric scores. The learned RK solver (*LRK4*) strongly improves PESQ, WVMOS and WER over Euler4, but does not yield a metric improvement across the board. The non-streaming FM baseline outperforms SFM, but the quality degradation of the streaming models is expected due to the small look-ahead, and is on acceptable levels. Our $\ell_{alg} = 16$ ms lower-latency SFM variant exhibits a relatively minor quality reduction compared to the main SFM model, proving the viability of our method for SE in even lower-latency settings ($\ell_{tot} \approx 24$ ms). In Table III, we also show metrics on the classic VoiceBank-DEMAND benchmark [53]. SFM exhibits the generalization capabilities expected of diffusion-based SE models [1], and attains the best PESQ, ESTOI and DistillMOS among streaming methods.

In the listening experiment results, depicted in Fig. 2a, SFM is clearly preferred over all low-latency baselines and has similar median ratings as the higher-latency Diffusion Buffer (DB) method ($d = 9$, $\ell_{alg} \approx 180$ ms), with a slightly higher upward and downward spread of scores. The non-streaming FM baseline receives an excellent median score around 90. DEMUCS [22] is not rated well in comparison, possibly indicating a failure to generalize to different data.

B. Dereverberation

For dereverberation, the results are shown in Table IV. SFM achieves a consistent improvement over the reverberant audio, with Euler5 generally performing best and the learned RK5 solver not yielding a clear improvement. The non-streaming FM baseline shows a clear advantage, particularly in PESQ and WER. We argue that this is expectable, since future information is especially useful to estimate RIR characteristics and suppress reverberation.

C. Codec post-filtering

For codec post-filtering, we list the results in Table V. SFM can greatly improve both intrusive and non-intrusive metrics over the plain Lyra V2 decoder, at the cost of a small increase in WER. The learned Runge-Kutta scheme fails to yield an improvement here except in LSD. We conjecture that (19) is suboptimal here since the Lyra decoder outputs already have good phoneme and high-frequency preservation, but we leave other loss choices to future work. Notably, the non-streaming FM baseline shows only marginal quality improvements over SFM at five Euler steps here, which may be related to the streaming nature of the Lyra V2 codec itself.

TABLE II: Mean metrics for speech enhancement on EARS-WHAM v2 (16 kHz) [35]. Our main model (SFM) is evaluated using different ODE solvers (*Euler1, Euler4*, etc.) with the number indicating the number of solver steps, and compared against several streaming and non-streaming baselines. "LRK4" refers to a four-stage learned Runge-Kutta solver, see Section III-E. In the NFE column, *1+n* indicates a single call for the initial predictor and n calls for the flow model. Methods marked with * use author-provided model checkpoints trained on different data. ℓ_{alg} is the algorithmic latency, see Section IV-D2. Best within a group in **bold**, second best underlined, worse than input in red.

Method	NFE	PESQ	ESTOI	SI-SDR	DiMOS	WVMOS	NISQA	WER↓	LSD↓	ℓ_{alg} (ms)	Params
Noisy	-	1.24	0.64	5.36	2.58	1.20	1.95	32.8%	2.24	-	-
STREAM.FM (32 ms)											
SFM Euler1	1+1	<u>2.18</u>	**0.84**	**15.2**	**3.91**	2.65	<u>4.43</u>	**19.5%**	1.39	32	52.5M
SFM Euler4	1+4	2.09	<u>0.83</u>	14.3	<u>3.88</u>	2.72	**4.50**	21.8%	**1.29**	32	52.5M
SFM Midpoint2	1+4	2.02	0.82	13.3	3.66	2.72	4.28	23.6%	1.38	32	52.5M
SFM LRK4	1+4	**2.24**	0.82	14.0	3.81	**3.00**	4.06	<u>20.2%</u>	1.42	32	52.5M
STREAMING BASELINES											
Diffusion Buffer $d=0$ [27]	1	1.75	0.75	11.1	2.81	2.08	2.49	27.3%	<u>1.37</u>	32	22.2M
Diffusion Buffer $d=9$ [27]	1	2.06	0.81	<u>14.4</u>	3.64	2.44	3.78	21.7%	1.47	176	22.2M
DEMUCS* [22]	1	1.95	0.79	13.0	3.46	**3.00**	2.61	27.1%	1.55	41	33.5M
DeepFilterNet3* [42]	1	1.76	0.70	8.8	3.29	<u>2.81</u>	3.37	37.0%	2.46	40	2.14M
HiFi-Stream* [24]	1	1.21	0.37	-3.8	1.94	1.49	1.50	69.7%	1.57	256	1.6M
LOWER-LATENCY STREAM.FM (16 MS)											
SFM 16ms Euler1	1+1	**2.07**	**0.83**	**14.8**	<u>3.72</u>	<u>2.59</u>	<u>4.43</u>	**20.5%**	**1.42**	16	52.5M
SFM 16ms Euler2	1+2	<u>2.01</u>	<u>0.82</u>	<u>14.7</u>	**3.77**	**2.63**	**4.45**	<u>21.7%</u>	<u>1.44</u>	16	52.5M
NON-STREAMING METHODS											
FM Euler1	1+1	<u>2.36</u>	**0.86**	**16.8**	4.20	2.73	4.37	**16.4%**	1.53	∞	73.7M
FM Euler4	1+4	**2.41**	**0.86**	16.1	**4.34**	<u>2.82</u>	**4.50**	<u>18.4%</u>	**1.31**	∞	73.7M
FM Midpoint2	1+4	2.35	<u>0.85</u>	15.0	<u>4.29</u>	<u>2.82</u>	<u>4.39</u>	19.4%	1.44	∞	73.7M
aTENNuate* [23]	1	1.86	0.72	9.0	2.95	**2.86**	2.37	33.1%	<u>1.42</u>	∞	0.8M

TABLE III: VoiceBank-DEMAND speech enhancement benchmark [53]. "DB" refers to the Diffusion Buffer [27].

Method	PESQ	ESTOI	SI-SDR	DiMOS	ℓ_{alg} (ms)
Noisy	1.97	0.79	8.4	2.58	-
SFM Euler4	**2.72**	**0.85**	13.4	**3.88**	32
SFM LRK4	<u>2.69</u>	<u>0.84</u>	13.3	<u>3.81</u>	32
DB $d=0$ [27]	2.42	0.81	13.1	2.75	32
DB $d=9$ [27]	2.45	<u>0.84</u>	14.5	3.66	176
DEMUCS [22]	2.60	**0.85**	<u>15.1</u>	3.46	41
DeepFilterNet3 [42]	<u>2.71</u>	<u>0.84</u>	**17.3**	3.34	40
HiFi-Stream [24]	2.48	0.83	14.1	3.35	256
aTENNuate [23]	**2.97**	0.84	16.2	2.95	∞
FM Euler4	2.86	**0.86**	14.1	**4.34**	∞

D. Bandwidth extension

We list the bandwidth extension metrics in Table VI. The Euler1 solver is now clearly the worst option, with a significant gap in non-intrusive metrics and WER compared to higher-NFE solvers. Notably, the learned RK5 solver performs best, with a clear improvement in all metrics except PESQ and SI-SDR. LRK5 also clearly improves WER, suggesting that it reconstructs the semantic content better. This confirms the usefulness of our proposal in this highly ambiguous restoration problem. PESQ may be misleading here, as the bandlimited audios receive the best PESQ scores. The improvements from the learned solver are also supported by the listening experiment Fig. 2b, where it receives the highest scores.

E. STFT phase retrieval / Mel vocoding

For STFT PR and Mel vocoding, see Tables VII and VIII, which are both highly ill-posed non-linear inverse problems, the behaviors of the solvers are similar as for BWE, but the differences are more pronounced. Euler1 produces unusable estimates here, with worse metrics even than naive zero-phase estimates, and the learned solvers show a clear advantage, particularly in WER and LSD.

F. Computations and runtimes

In Table IX we list the determined GFLOPs per second, the streaming RTF defined as $\frac{t_{\text{proc,fr}}}{H_{\text{syn}}/f_s}$ for each single frame where f_s is the sampling frequency and $t_{\text{proc,fr}}$ is the processing time per frame [22], and offline RTF $\frac{t_{\text{proc}}}{1\text{sec}}$ where t_{proc} is the time taken to process one 1-second utterance in a single model call. We can see that offline RTF consistently and severely underestimates the streaming RTF, and that FLOPs have no simple relation to streaming RTF, e.g., the Diffusion Buffer [27] attains smaller streaming and offline RTFs than Stream.FM for 5 NFE but spends substantially more FLOPs.

G. Model weight compression for Mel vocoding

In Fig. 3, we analyze the model weight compression ideas introduced in Section III-F, using the Mel vocoding task as an example with the Euler solver at different NFE. We can see that more compressed models (smaller ranks $T < K$) reduce all metrics when keeping NFE = 5 constant, and linearly decrease GFLOPs per frame, both as expected. However, when increasing the NFE to the respective maximum possible number for each compressed model on our hardware, we find that $T = 6$ at NFE = 7 is better across all metrics than the uncompressed $T = K = 9$ at NFE = 5, while also slightly decreasing the GFLOPs per frame. This suggests that slight model compression may be preferable over no compression if it allows to increase the NFE, confirming the usefulness of our proposal.

VI. CONCLUSION

In this work, we have presented Stream.FM, a streaming generative method for general speech restoration tasks which can

TABLE IV: Dereverberation task (EARS-Reverb v2 test set). SFM has $\ell_{alg} = 32$ ms, $\ell_{tot} \approx 48$ ms; FM has $\ell_{alg} = \infty$ and 38.7M parameters. Best within a group in **bold**, second best <u>underlined</u>, worse than input in red.

Method	NFE	PESQ	ESTOI	SI-SDR	DiMOS	WVMOS	NISQA	WER↓	LSD↓
Reverberant	-	1.32	0.58	-16.6	3.02	2.02	2.11	20.1%	1.21
SFM Euler1	1	1.63	0.73	-14.2	3.29	2.17	3.03	23.6%	1.49
SFM Euler5	5	**2.01**	**0.79**	**-13.3**	**3.76**	**2.49**	<u>3.59</u>	<u>16.8%</u>	1.12
SFM Midpoint2	4	<u>1.94</u>	0.78	-14.3	<u>3.70</u>	2.35	**3.62**	17.3%	**0.98**
SFM LRK5	5	1.91	0.78	<u>-13.5</u>	3.49	<u>2.44</u>	3.29	**16.6%**	<u>1.01</u>
FM Euler5	5	2.31	0.85	-11.7	3.77	2.43	3.47	11.4%	1.01

TABLE V: Lyra V2 codec post-filtering task on EARS-WHAM v2 test set clean utterances. SFM has $\ell_{alg} = 40$ ms, $\ell_{tot} \approx 60$ ms and 27.9M parameters; FM has $\ell_{alg} = \infty$ and 38.7M parameters. Best within a group in **bold**, second best <u>underlined</u>, worse than input in red.

Method	NFE	PESQ	ESTOI	SI-SDR	DiMOS	WVMOS	NISQA	WER↓	LSD↓
Decoded	-	2.00	0.76	1.6	3.08	2.60	2.68	**13.6%**	1.01
SFM Euler1	1	1.80	0.63	**4.9**	2.18	1.81	2.45	44.3%	5.84
SFM Euler5	5	**2.55**	**0.80**	<u>3.1</u>	<u>4.00</u>	**2.93**	<u>3.96</u>	16.3%	1.49
SFM Midpoint2	4	<u>2.38</u>	<u>0.79</u>	1.9	**4.09**	<u>2.80</u>	**4.11**	<u>16.0%</u>	<u>1.25</u>
SFM LRK5	5	2.27	0.77	1.8	3.94	2.71	3.90	16.3%	**1.09**
FM Euler5	5	2.56	0.80	3.0	4.14	2.92	3.97	16.1%	1.41

TABLE VI: Bandwidth extension task ($\{2,4\}$ kHz $\to 8$ kHz of frequency content) on EARS-WHAM v2 test set clean utterances. SFM has $\ell_{alg} = 32$ ms, $\ell_{tot} \approx 48$ ms and 27.9M parameters; FM has $\ell_{alg} = \infty$ and 38.7M parameters. Best within a group in **bold**, second best <u>underlined</u>, worse than input in red.

Method	NFE	PESQ	ESTOI	SI-SDR	DiMOS	WVMOS	NISQA	WER↓	LSD↓
Bandlimited	-	**3.51**	0.84	15.9	3.09	2.21	2.93	19.4%	2.24
SFM Euler1	1	3.22	<u>0.92</u>	**16.8**	3.43	2.41	3.32	15.3%	1.95
SFM Euler5	5	<u>3.37</u>	**0.94**	<u>16.5</u>	4.07	2.96	3.76	12.3%	1.26
SFM Midpoint2	4	3.10	**0.94**	16.0	<u>4.18</u>	<u>3.01</u>	<u>3.97</u>	<u>12.0%</u>	<u>1.10</u>
SFM LRK5	5	3.02	**0.94**	15.3	**4.19**	**3.02**	**3.99**	**10.5%**	**1.00**
FM Euler5	5	3.52	0.94	16.3	4.28	3.00	3.77	11.8%	1.29

TABLE VII: STFT phase retrieval task with 50% overlap, no lookahead, on EARS-WHAM v2 test set clean utterances. RTISI-DM refers to [37] using the Difference Map algorithm [54] with hyperparameter $\beta = 1.75$, which we grid-searched as the optimum for 50 iterations. RTISI has no parameters. For RTISI and RTISI-DM, *NFE* refers to the number of algorithm iterations. Best within a group in **bold**, second best <u>underlined</u>, worse than input in red.

Method	NFE	PESQ	ESTOI	SI-SDR	DiMOS	WVMOS	NISQA	WER↓	LSD↓	ℓ_{alg} (ms)	Params
Zero-phase	-	1.31	0.68	-34.6	1.73	1.86	1.42	14.9%	1.19	-	-
SFM Euler1	1	1.58	0.58	**1.7**	1.90	1.70	2.05	61.8%	6.72	32	27.9M
SFM Euler5	5	**4.24**	**0.97**	<u>-1.7</u>	<u>4.3</u>	<u>3.26</u>	4.13	<u>3.7%</u>	0.76	32	27.9M
SFM Midpoint2	4	4.05	<u>0.96</u>	-2.5	<u>4.3</u>	3.15	**4.15**	<u>3.7%</u>	<u>0.68</u>	32	27.9M
SFM LRK5	5	<u>4.22</u>	**0.97**	-2.3	**4.4**	**3.27**	4.11	**3.0%**	**0.61**	32	27.9M
RTISI [55]	50	3.08	0.90	-28.3	3.28	2.76	3.03	5.2%	0.71	32	0
RTISI-DM [37]	50	3.35	0.91	-27.0	3.86	2.83	3.56	4.9%	0.73	32	1
FM Euler5	5	4.38	0.98	-1.2	4.37	3.25	4.10	2.9%	0.65	∞	38.7M

TABLE VIII: Mel vocoding task on EARS-WHAM v2 test set clean utterances. *HiFi-GAN* refers to the 16 kHz model trained on LibriTTS data [56], available in SpeechBrain [39]. For RTISI-DM, *NFE* means the number of algorithm iterations. Best within a group in **bold**, second best <u>underlined</u>, worse than input in red. We refer the reader to our prior work [12] for a more in-depth evaluation on this task.

Method	NFE	PESQ	ESTOI	SI-SDR	DiMOS	WVMOS	NISQA	WER↓	LSD↓	ℓ_{alg} (ms)	Params
$M^\dagger$ + Zero-phase	-	1.28	0.63	-38.9	1.43	1.07	1.21	35.1%	2.22	-	
SFM Euler1	1	1.35	0.36	**-5.6**	1.18	1.23	1.35	86.7%	7.82	32	27.9M
SFM Euler5	5	4.10	**0.96**	<u>-10.1</u>	<u>4.31</u>	<u>3.11</u>	4.14	4.7%	1.01	32	27.9M
SFM Midpoint2	4	3.92	0.94	-10.2	4.28	2.99	**4.17**	<u>4.5%</u>	0.89	32	27.9M
SFM LRK5	5	**4.14**	**0.96**	-10.4	**4.34**	**3.15**	<u>4.15</u>	**3.6%**	**0.80**	32	27.9M
$M^\dagger$ + RTISI-DM [37]	50	2.97	0.88	-29.7	2.51	1.92	2.58	5.8%	0.82	32	1
FM Euler5	5	4.34	0.97	-9.9	4.35	3.05	4.15	4.3%	1.04	∞	38.7M
HiFi-GAN [38]	1	2.99	0.90	-29.9	4.21	3.02	3.91	5.4%	0.77	236	13.9M

TABLE IX: FLOP and RTF measurements on an RTX 4080 Laptop GPU. N is the number of DNN calls per frame. FLOPs are per-frame multiplied with the number of frames per second, streaming RTF is relative to each model's per-frame runtime budget (frame shift), and offline RTF is determined for 1-second inputs.

Method	GFLOPs/sec.	Streaming RTF	Offline RTF
Stream.FM	$N \cdot 282.0$	$N \cdot 0.174$	$N \cdot 0.021$
Diffusion Buffer [27]	8810.4	0.646	0.007
DEMUCS [22]	10.2	0.221	0.004
HiFi-Stream [24]	6.7	0.098	0.012
DeepFilterNet3 [42]	0.3	0.321	0.008
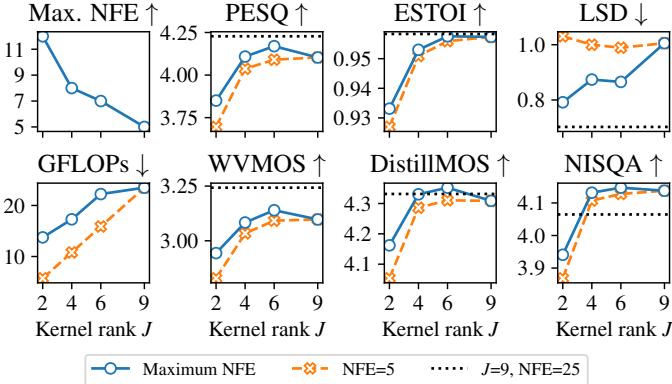
Fig. 3: Metrics of compressed Stream.FM models for Mel vocoding using kernel ranks $J \in \{2,4,6,9\}$ where $J = 9$ is uncompressed, using the Euler solver. We compare the maximum NFE for each J under our runtime budget against constant NFE $= 5$ and a high-NFE variant $K = 9$, NFE $= 25$. Reported GFLOPs are per-frame.

run in real-time on a consumer GPU. We detailed our architecture, inference scheme, and other optimizations needed to achieve this real-time capability, showed state-of-the-art performance for generative streaming methods through metrics and listening experiments, and proposed novel ideas to optimize quality or compute in a low-NFE setting. We hope that our contributions and public codebase can help to close the gap between generative and predictive speech restoration models in real-time settings, and support the research community towards further developments in this field.

VII. ACKNOWLEDGEMENTS

REFERENCES

[1] J. Richter, S. Welker, J.-M. Lemercier, B. Lay, and T. Gerkmann, "Speech enhancement and dereverberation with diffusion-based generative models," *IEEE Trans. on Audio, Speech, and Lang. Proc. (TASLP)*, vol. 31, pp. 2351–2364, 2023.

[2] J.-M. Lemercier, J. Richter, S. Welker, and T. Gerkmann, "Analysing diffusion-based generative approaches versus discriminative approaches for speech restoration," in *IEEE Int. Conf. on Acoustics, Speech and Signal Proc. (ICASSP)*, 2023.

[3] Y.-C. Wu, D. Marković, S. Krenn, I. D. Gebru, and A. Richard, "ScoreDec: A phase-preserving high-fidelity audio codec with a generalized score-based diffusion post-filter," in *IEEE Int. Conf. on Acoustics, Speech and Signal Proc. (ICASSP)*. IEEE, 2024.

[4] S. Welker, M. Le, R. T. Q. Chen, W.-N. Hsu, T. Gerkmann, A. Richard, and Y.-C. Wu, "FlowDec: A flow-based full-band general audio codec with high perceptual quality," in *Int. Conf. on Learning Repres. (ICLR)*, 2025.

[5] J.-M. Lemercier, J. Richter, S. Welker, and T. Gerkmann, "StoRM: A diffusion-based stochastic regeneration model for speech enhancement and dereverberation," *IEEE Trans. on Audio, Speech, and Lang. Proc. (TASLP)*, vol. 31, pp. 2724–2737, 2023.

[6] T. Peer, S. Welker, and T. Gerkmann, "DiffPhase: Generative diffusion-based STFT phase retrieval," in *IEEE Int. Conf. on Acoustics, Speech and Signal Proc. (ICASSP)*. IEEE, 2023.

[7] S. Liang, D. Markovic, I. D. Gebru, S. Krenn, T. Keebler, J. Sandakly, F. Yu, S. Hassel, C. Xu, and A. Richard, "BinauralFlow: A causal and streamable approach for high-quality binaural speech synthesis with flow matching models," in *Int. Conf. on Machine Learning (ICML)*, 2025.

[8] B. Lay, R. Makarov, and T. Gerkmann, "Diffusion buffer: Online diffusion-based speech enhancement with sub-second latency," *Interspeech*, 2025.

[9] T.-A. Hsieh and S. Braun, "Towards real-time generative speech restoration with flow-matching," *arXiv preprint arXiv:2510.16997*, 2025.

[10] S. Lu, H. Huang, J. Yao, K. Wang, Q. Hong, and L. Li, "A Two-Stage Hierarchical Deep Filtering Framework for Real-Time Speech Enhancement," in *Interspeech*, 2025.

[11] Y. Lipman, R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le, "Flow matching for generative modeling," in *Int. Conf. on Learning Repres. (ICLR)*, 2023.

[12] S. Welker, T. Peer, and T. Gerkmann, "Real-time streaming Mel vocoding with generative flow matching," *arXiv preprint arXiv:2509.15085*, 2025.

[13] S. Welker, M. Hillemann, B. Lay, and T. Gerkmann, "Real-time diffusion demo for speech enhancement with 48ms latency," in *Demo Papers at the ITG Conference on Speech Communication*, 2025.

[14] Y.-J. Lu, Z.-Q. Wang, S. Watanabe, A. Richard, C. Yu, and Y. Tsao, "Conditional diffusion probabilistic model for speech enhancement," in *IEEE Int. Conf. on Acoustics, Speech and Signal Proc. (ICASSP)*. IEEE, 2022.

[15] S. Welker, J. Richter, and T. Gerkmann, "Speech enhancement with score-based generative models in the complex STFT domain," in *Interspeech*, 2022.

[16] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-based generative modeling through stochastic differential equations," in *Int. Conf. on Learning Repres. (ICLR)*, 2021.

[17] A.-A. Pooladian, H. Ben-Hamu, C. Domingo-Enrich, B. Amos, Y. Lipman, and R. T. Q. Chen, "Multisample flow matching: Straightening flows with minibatch couplings," in *Int. Conf. on Machine Learning (ICML)*, 2023.

[18] S. U. Wood and J. Rouat, "Unsupervised low latency speech enhancement with RT-GCC-NMF," *IEEE J. Sel. Top. Signal Proc. (JSTSP)*, vol. 13, no. 2, pp. 332–346, 2019.

[19] Z.-Q. Wang, G. Wichern, S. Watanabe, and J. Le Roux, "STFT-domain neural speech enhancement with very low algorithmic latency," *IEEE Trans. on Audio, Speech, and Lang. Proc. (TASLP)*, vol. 31, pp. 397–410, 2023.

[20] E. Hairer, S. P. Norsett, and G. Wanner, *Solving ordinary differential equations I*, 2nd ed., ser. Springer Series in Computational Mathematics. Springer, 1993.

[21] J. C. Butcher, "On Runge-Kutta processes of high order," *Journal of the Australian Mathematical Society*, vol. 4, no. 2, p. 179–194, 1964.

[22] A. Défossez, G. Synnaeve, and Y. Adi, "Real time speech enhancement in the waveform domain," in *Interspeech*, 2020.

[23] Y. R. Pei, R. Shrivastava, and F. Sidharth, "Optimized Real-time Speech Enhancement with Deep SSMs on Raw Audio," in *Interspeech*, 2025.

[24] E. Dmitrieva and M. Kaledin, "HiFi-Stream: Streaming speech enhancement with generative adversarial networks," *IEEE Signal Proc. Lett. (SPL)*, vol. 32, pp. 3595–3599, 2025.

[25] J. Richter, S. Welker, J.-M. Lemercier, B. Lay, T. Peer, and T. Gerkmann, "Causal diffusion models for generalized speech enhancement," *IEEE Open J. Signal Proc.*, 2024.

[26] L. Hedegaard and A. Iosifidis, "Continual inference: a library for efficient online inference with deep neural networks in PyTorch," in *ECCV Workshops*, 2022.

[27] B. Lay, R. Makarov, S. Welker, M. Hillemann, and T. Gerkmann, "Diffusion buffer for online generative speech enhancement," *arXiv preprint arXiv:2510.18744*, 2025.

[28] Y. Wu and K. He, "Group normalization," in *Eur. Conf. Comput. Vis.*, 2018.

[29] S. Chang, H. Park, J. Cho, H. Park, S. Yun, and K. Hwang, "Subspectral normalization for neural audio data processing," in *IEEE Int. Conf. on Acoustics, Speech and Signal Proc. (ICASSP)*. IEEE, 2021.

[30] 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 *IEEE Int. Conf. on Acoustics, Speech and Signal Proc. (ICASSP)*, 2020.

[31] T. Saeki, S. Maiti, S. Takamichi, S. Watanabe, and H. Saruwatari, "SpeechBERTScore: Reference-aware automatic evaluation of speech generation leveraging NLP evaluation metrics," in *Interspeech*, 2024.

[32] D. de Oliveira, T. Peer, J. Rochdi, and T. Gerkmann, "Are these even words? quantifying the gibberishness of generative speech models," *arXiv preprint arXiv:2510.21317*, 2025.

[33] A. Ralston, "Runge-kutta methods with minimum error bounds," *Mathematics of computation*, vol. 16, no. 80, pp. 431–437, 1962.

[34] J. Guo, Y. Li, W. Lin, Y. Chen, and J. Li, "Network decoupling: From regular to depthwise separable convolutions," in *British Machine Vision Conference*, 2018.

[35] J. Richter, Y.-C. Wu, S. Krenn, S. Welker, B. Lay, S. Watanabe, A. Richard, and T. Gerkmann, "EARS: An anechoic fullband speech dataset benchmarked for speech enhancement and dereverberation," in *Interspeech*, 2024.

[36] R. Kumar, P. Seetharaman, A. Luebs, I. Kumar, and K. Kumar, "High-fidelity audio compression with improved RVQGAN," in *Advances in Neural Inf. Proc. Systems (NeurIPS)*, 2023.

[37] T. Peer, S. Welker, J. Kolhoff, and T. Gerkmann, "A flexible online framework for projection-based STFT phase retrieval," in *IEEE Int. Conf. on Acoustics, Speech and Signal Proc. (ICASSP)*. IEEE, 2024.

[38] J. Kong, J. Kim, and J. Bae, "HiFi-GAN: Generative adversarial networks for efficient and high fidelity speech synthesis," *Advances in Neural Inf. Proc. Systems (NeurIPS)*, 2020.

[39] M. Ravanelli, T. Parcollet, A. Moumen, S. de Langen, C. Subakan, P. Plantinga, Y. Wang, P. Mousavi, L. D. Libera, A. Ploujnikov *et al.*, "Open-source conversational AI with SpeechBrain 1.0," *J. of Machine Learning Research*, vol. 25, no. 333, 2024.

[40] N. Vyas, D. Morwani, R. Zhao, I. Shapira, D. Brandfonbrener, L. Janson, and S. M. Kakade, "SOAP: Improving and stabilizing shampoo using adam for language modeling," in *Int. Conf. on Learning Repres. (ICLR)*, 2025.

[41] F. Schaipp, "Optimization benchmark for diffusion models on dynamical systems," in *EurIPS Workshop on Principles of Generative Modeling*, 2025.

[42] H. Schröter, T. Rosenkranz, A. N. Escalante-B., and A. Maier, "DeepFilterNet: Perceptually motivated real-time speech enhancement," in *Interspeech*, 2023.

[43] A. Rix, J. Beerends, M. Hollier, and A. Hekstra, "Perceptual evaluation of speech quality (PESQ)-a new method for speech quality assessment of telephone networks and codecs," in *IEEE Int. Conf. on Acoustics, Speech and Signal Proc. (ICASSP)*, 2001.

[44] J. Jensen and C. H. Taal, "An algorithm for predicting the intelligibility of speech masked by modulated noise maskers," *IEEE Trans. on Audio, Speech, and Lang. Proc. (TASLP)*, vol. 24, no. 11, pp. 2009–2022, 2016.

[45] J. Le Roux, S. Wisdom, H. Erdogan, and J. R. Hershey, "SDR - half-baked or well done?" in *IEEE Int. Conf. on Acoustics, Speech and Signal Proc. (ICASSP)*, 2019.

[46] H. Liu, W. Choi, X. Liu, Q. Kong, Q. Tian, and D. Wang, "Neural vocoder is all you need for speech super-resolution," in *Interspeech*, 2022.

[47] S. Kriman, S. Beliaev, B. Ginsburg, J. Huang, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, and Y. Zhang, "QuartzNet: Deep automatic speech recognition with 1D time-channel separable convolutions," in *IEEE Int. Conf. on Acoustics, Speech and Signal Proc. (ICASSP)*, 2020.

[48] O. Kuchaiev, J. Li, H. Nguyen, O. Hrinchuk, R. Leary, B. Ginsburg, S. Kriman, S. Beliaev, V. Lavrukhin, J. Cook *et al.*, "NeMo: a toolkit for building AI applications using neural modules," *arXiv preprint arXiv:1909.09577*, 2019.

[49] G. Mittag, B. Naderi, A. Chehadi, and S. Möller, "NISQA: A deep CNN-self-attention model for multidimensional speech quality prediction with crowdsourced datasets," in *Interspeech*, 2021.

[50] P. Andreev, A. Alanov, O. Ivanov, and D. Vetrov, "HiFi++: A unified framework for bandwidth extension and speech enhancement," in *IEEE Int. Conf. on Acoustics, Speech and Signal Proc. (ICASSP)*, 2023.

[51] B. Stahl and H. Gamper, "Distillation and pruning for scalable self-supervised representation-based speech quality assessment," in *IEEE Int. Conf. on Acoustics, Speech and Signal Proc. (ICASSP)*, 2025.

[52] ITU-R Rec. BS.1534-3, "Method for the subjective assessment of intermediate quality level of audio systems," *Int. Telecom. Union (ITU)*, 2014.

[53] C. Valentini-Botinhao, X. Wang, S. Takaki, and J. Yamagishi, "Investigating RNN-based speech enhancement methods for noise-robust text-to-speech," in *9th ISCA Workshop on Speech Synthesis Workshop (SSW 9)*, 2016.

[54] V. Elser, "Phase retrieval by iterated projections," *J. Opt. Soc. Am. A*, vol. 20, no. 1, p. 40, 2003.

[55] G. T. Beauregard, X. Zhu, and L. Wyse, "An efficient algorithm for real-time spectrogram inversion," in *Int. Conf. on Digital Audio Effects*, 2005.

[56] H. Zen, V. Dang, R. Clark, Y. Zhang, R. J. Weiss, Y. Jia, Z. Chen, and Y. Wu, "LibriTTS: A corpus derived from LibriSpeech for text-to-speech," in *Interspeech*, 2019.

Simon Welker received a B.Sc. in Computing in Science (2019) and M.Sc. in Bioinformatics (2021) from University of Hamburg, Germany. He is currently a PhD student in the labs of Prof. Timo Gerkmann (Signal Processing, University of Hamburg) and Prof. Henry N. Chapman (Center for Free-Electron Laser Science, DESY, Hamburg), researching machine learning techniques for solving inverse problems that arise in speech processing and X-ray imaging. He received the VDE ITG award 2024.

Bunlong Lay obtained a B.Sc. and M.Sc. in Mathematics in 2015 and 2017 from the University of Bonn, Germany. He subsequently joined the research institute Fraunhofer FKIE in Wachtberg Germany from 2018 until 2021, where he focused on research in the field of radar signal processing. In 2021 he started his Ph.D. at the University of Hamburg. Currently researching Diffusion-based models for Speech Enhancement for real-time applications. He received the VDE ITG award 2024.

Maris Hillemann obtained a B.Sc. in Computer Science in 2024 from the University of Hamburg, Germany. He is currently a master's student in Computer Science at the University of Hamburg and a student assistant in the Signal Processing group of Prof. Timo Gerkmann.

Tal Peer received the B.Sc. degree in General Engineering Science (2016) and the M.Sc. degree in Electrical Engineering (2019) from the Hamburg University of Technology. He is currently pursuing a PhD with the Signal Processing group at the University of Hamburg. His research interests include phase-aware speech enhancement and phase retrieval for speech and audio applications.

Timo Gerkmann (S'08–M'10–SM'15) is a professor for Signal Processing at the Universität Hamburg, Germany. He has previously held positions at Technicolor Research & Innovation in Germany, the University of Oldenburg in Germany, KTH Royal Institute of Technology in Sweden, Ruhr-Universität Bochum in Germany, and Siemens Corporate Research in Princeton, NJ, USA. His main research interests are on statistical signal processing and machine learning for speech and audio applied to communication devices, hearing instruments, audio-visual media, and human-machine interfaces. Timo Gerkmann served as member of the IEEE Signal Processing Society Technical Committee on Audio and Acoustic Signal Processing and is currently a Senior Area Editor of the IEEE/ACM Transactions on Audio, Speech and Language Processing. He received the VDE ITG award 2022.

VIII. SUPPLEMENTARY MATERIAL

In the following, we list additional technical details for our Stream.FM models, which are not necessary for the method itself but are useful to implement real-time capable streaming inference.

A. Functional frame-wise inference API

To implement our frame-wise inference scheme, we implement our model using PyTorch and PyTorch Lightning. we design an application programming interface (API) similar to the one proposed for CINs [26]. The official CINs implementation stores tracked states on each layer, making every layer stateful. This is problematic for our purposes, since we want to use only a single DNN instance v_θ with shared weights, but multiple state collections (one for each diffusion solver step).

Our API hence aims to make the layers themselves stateless, so they do not track their own state. It is based on two functions, `init_state()` and `forward_step(x, state)`, which each stateful streaming layer and the overall model must implement. `init_state()` creates and returns a data structure containing all tracked state variables (e.g., buffers) of each module as well as their respective nested modules. `forward_step(frame, state)` receives a single new `frame` as well as the previous `state`, and returns an output frame along with an updated `state` data structure. Higher-level modules then receive, pass along, and update their own state as well as the state for any nested modules.

We note that, unlike pure functional programming APIs, our API does not necessarily construct a new `state` object for each call, but may instead modify an existing `state` in-place. We made this choice for better compatibility with `torch.compile` and CUDA graphs, but leave further optimization to future work.

B. Minimizing overhead

In initial real-time experiments with our models for real-time streaming inference, we found that the processing time per frame quickly exceeded our 16 ms budget even for low NFE values such as 3. After careful inspection of profiler traces, we found that the processing was dominated by CPU-GPU overhead. To reduce this overhead as much as possible, we first tried using model compilation via `torch.compile` to merge CUDA kernels, which reduced the overhead to some extent but not enough to allow real-time inference. We then modified our code to use CUDA Graphs as implemented in PyTorch. By capturing the entire solver computation including the sequence of N DNN calls in a single CUDA Graph and then replaying this CUDA Graph for every frame, we are able to realize up to 5 DNN calls per frame within the 16 ms time budget.

C. Learned coefficients for custom learned low-NFE ODE solvers

In this section, we list the parameters $\{\mathbf{A}, \mathbf{b}, \mathbf{c}\}$ of the learned Runge-Kutta ODE solver schemes, see Section III.E in the main paper, for each of the six tasks investigated in the main paper.

1) Speech enhancement:

$$
\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.458 & 0 & 0 & 0 \\ -0.847 & 1.623 & 0 & 0 \\ 2.029 & -1.707 & 0.528 & 0 \end{bmatrix}
$$
$$
\mathbf{b} = \begin{bmatrix} 0.339, & 0.444, & 0.102, & 0.114 \end{bmatrix}
$$
$$
\mathbf{c} = \begin{bmatrix} 0, & 0.458, & 0.776, & 0.850 \end{bmatrix}
$$
(22)

2) Dereverberation:

$$
\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0.152 & 0 & 0 & 0 & 0 \\ -0.065 & 0.312 & 0 & 0 & 0 \\ 0.088 & 0.296 & 0.152 & 0 & 0 \\ 0.565 & 0.856 & 1.425 & -1.997 & 0 \end{bmatrix}
$$
$$
\mathbf{b} = \begin{bmatrix} 0.079, & 0.223, & 0.423, & 0.184, & 0.091 \end{bmatrix}
$$
$$
\mathbf{c} = \begin{bmatrix} 0, & 0.152, & 0.247, & 0.536, & 0.850 \end{bmatrix}
$$
(23)

3) Codec post-filtering:

$$
\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0.298 & 0 & 0 & 0 & 0 \\ 0.049 & 0.375 & 0 & 0 & 0 \\ -0.245 & 1.030 & -0.219 & 0 & 0 \\ 0.672 & -0.168 & -0.276 & 0.622 & 0 \end{bmatrix}
$$
$$
\mathbf{b} = \begin{bmatrix} 0.089, & 0.211, & 0.307, & 0.100, & 0.292 \end{bmatrix}
$$
$$
\mathbf{c} = \begin{bmatrix} 0, & 0.298, & 0.424, & 0.566, & 0.850 \end{bmatrix}
$$
(24)

4) Bandwidth extension:

$$
\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0.112 & 0 & 0 & 0 & 0 \\ -0.244 & 0.535 & 0 & 0 & 0 \\ -1.093 & 1.840 & -0.217 & 0 & 0 \\ -1.587 & 1.783 & 0.236 & 0.419 & 0 \end{bmatrix}
$$
$$
\mathbf{b} = \begin{bmatrix} 0.085, & 0.211, & 0.262, & 0.097, & 0.344 \end{bmatrix}
$$
$$
\mathbf{c} = \begin{bmatrix} 0, & 0.112, & 0.291, & 0.529, & 0.850 \end{bmatrix}
$$
(25)

5) STFT phase retrieval:

$$
\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0.271 & 0 & 0 & 0 & 0 \\ 0.216 & 0.198 & 0 & 0 & 0 \\ -0.029 & 0.147 & 0.454 & 0 & 0 \\ 0.072 & 0.208 & 0.326 & 0.244 & 0 \end{bmatrix}
$$
$$
\mathbf{b} = \begin{bmatrix} 0.128, & 0.209, & 0.307, & 0.130, & 0.227 \end{bmatrix}
$$
$$
\mathbf{c} = \begin{bmatrix} 0, & 0.271, & 0.413, & 0.572, & 0.850 \end{bmatrix}
$$
(26)

6) Mel vocoding:

$$
\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0.251 & 0 & 0 & 0 & 0 \\ 0.104 & 0.286 & 0 & 0 & 0 \\ -0.005 & 0.200 & 0.379 & 0 & 0 \\ 0.091 & 0.181 & 0.344 & 0.234 & 0 \end{bmatrix}
$$
$$
\mathbf{b} = \begin{bmatrix} 0.134, & 0.208, & 0.307, & 0.122, & 0.229 \end{bmatrix}
$$
$$
\mathbf{c} = \begin{bmatrix} 0, & 0.251, & 0.390, & 0.574, & 0.850 \end{bmatrix}
$$
(27)