dYIN and dSWIPE: Differentiable Variants of Classical Fundamental Frequency Estimators

Sebastian Strahl and Meinard Müller, *Fellow, IEEE*

***Abstract*—Fundamental frequency (F0) estimation is a critical task in audio, speech, and music processing applications, such as speech analysis and melody extraction. F0 estimation algorithms generally fall into two paradigms: classical signal processing-based methods and neural network-based approaches. Classical methods, like YIN and SWIPE, rely on explicit signal models, offering interpretability and computational efficiency, but their non-differentiable components hinder integration into deep learning pipelines. Neural network-based methods, such as CREPE, are fully differentiable and flexible but often lack interpretability and require substantial computational resources. In this paper, we propose differentiable variants of two classical algorithms, dYIN and dSWIPE, which combine the strengths of both paradigms. These variants enable gradient-based optimization while preserving the efficiency and interpretability of the original methods. Through several case studies, we demonstrate their potential: First, we use gradient descent to reverse-engineer audio signals, showing that dYIN and dSWIPE produce smoother gradients compared to CREPE. Second, we design a two-stage vocal melody extraction pipeline that integrates music source separation with a differentiable F0 estimator, providing an interpretable intermediate representation. Finally, we optimize dSWIPE's spectral templates for timbre-specific F0 estimation on violin recordings, demonstrating its enhanced adaptability over SWIPE. These case studies highlight that dYIN and dSWIPE successfully combine the flexibility of neural network-based methods with the interpretability and efficiency of classical algorithms, making them valuable tools for building end-to-end trainable and transparent systems.**

***Index Terms*—Differentiable algorithms, fundamental frequency estimation, music processing.**

I. Introduction

THE accurate estimation of fundamental frequency (F0) is a critical task in speech and music processing, commonly required in applications such as melody transcription, harmonic analysis, and speech synthesis. Defined as the lowest frequency of a periodic waveform, the F0 typically corresponds to the perceived pitch. Although there are cases where F0 and pitch differ [1], the terms are often used interchangeably.

Received 24 January 2025; revised 5 May 2025; accepted 8 June 2025. Date of publication 19 June 2025; date of current version 27 June 2025. This work was supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Grant 500643750 (MU 2686/15-1). The associate editor coordinating the review of this article and approving it for publication was Prof. Alberto Bernardini. *(Corresponding author: Sebastian Strahl.)*

The authors are with the International Audio Laboratories Erlangen, a joint institution of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) and Fraunhofer Institute for Integrated Circuits IIS, 91058 Erlangen, Germany (e-mail: sebastian.strahl@audiolabs-erlangen.de; meinard.mueller@audiolabs-erlangen.de).

Digital Object Identifier 10.1109/TASLPRO.2025.3581119

Algorithms for F0 estimation generally belong to one of two main paradigms: classical signal processing [2], [3], [4], [5], [6], [7] or methods based on neural networks (NN) [8], [9], [10], [11], [12], [13], [14], [15], [16], [17]. Prominent examples of classical algorithms include YIN [4], which uses the autocorrelation method, and pYIN [7], a probabilistic extension of YIN that incorporates a hidden Markov model to determine the most likely pitch sequence. Another example is SWIPE [5], which relies on spectral comparison with F0-specific templates based on the sawtooth waveform. Among NN-based methods, CREPE [8] is widely known. It uses a convolutional architecture to process raw audio inputs and directly generates probabilities for a set of F0 candidates, though it requires annotated data for supervised training. Self-supervised approaches like SPICE [11] and PESTO [15] reduce the need for labeled data by using artificially pitch-shifted versions of the input and training models to predict these relative pitches rather than absolute pitches.

The classical algorithms are built on explicit signal models, offering interpretability, predictable behavior, and computational efficiency, which makes them ideal for applications with limited resources. On the other hand, NNs are often computationally expensive and behave like black-box models, making it difficult for users to predict the model's output when presented with inputs dissimilar to the training data. Despite this, NNs are widely favored for F0 estimation due to their superior performance. Also, they integrate easily into deep learning pipelines. For instance, CREPE [8] has been used to create perceptual loss functions, e.g., for training an autoencoder that includes an F0 estimate as part of its latent representation [18] or in the context of singing voice conversion [19]. In contrast, classical signal-processing algorithms often include non-differentiable components, limiting their compatibility with modern deep learning frameworks.

Only little work has been done to combine the two paradigms. For example, the hybrid approach FusedF0 [16] fuses signal processing and NN-based features extracted from raw waveforms, achieving the performance of NN-based methods while retaining the noise robustness of signal processing approaches. However, FusedF0 is incompatible with deep learning pipelines due to its computationally expensive extraction of signal processing features [16] and because these features are not differentiable w.r.t. the time-domain input.

With this work, we aim to further bridge the gap between the two paradigms by transforming the classical F0 estimators YIN and SWIPE into differentiable counterparts, dYIN and dSWIPE. This approach combines the strengths of both paradigms. The

differentiable F0 estimators preserve the interpretability and efficiency of the original methods while enabling gradient-based optimization and seamless end-to-end training of deep learning pipelines. To facilitate reproducibility and further research, we provide our PyTorch [20] implementations of dYIN and dSWIPE on GitHub.[1]

This article is structured as follows. In Section II, we introduce differentiable variants of the classical F0 estimation algorithms YIN and SWIPE. We detail the process of transforming them into dYIN and dSWIPE, making them compatible with gradient-based optimization while preserving their interpretability and efficiency. Section III presents three case studies that highlight the potential of these differentiable signal processing-based algorithms. First, we use gradient descent to reverse-engineer audio signals, generating user-specified F0 trajectories. This demonstrates that the gradients backpropagated through dYIN and dSWIPE are smoother and less noisy compared to those from CREPE. Second, we design a two-stage melody extraction pipeline that combines a music source separation model with a differentiable monophonic F0 estimator, offering an interpretable intermediate representation in the form of a separated melody line. Third, we optimize the spectral templates of dSWIPE for timbre-specific F0 estimation on violin recordings, demonstrating its enhanced adaptability compared to SWIPE. Finally, we conclude the article in Section IV.

II. Method

In this section, we introduce our differentiable F0 estimators, dYIN and dSWIPE, which build upon the classical YIN and SWIPE algorithms. We first describe the modifications that make these methods differentiable and compatible with integration into deep learning pipelines. Next, we illustrate the model outputs and present different options for loss computation. Finally, we summarize strategies for F0 selection from the probabilistic model outputs and provide a quantitative performance comparison with the original YIN and SWIPE algorithms.

A. dYIN

The YIN algorithm [4] operates in the time domain and is based on the autocorrelation method. Conceptually, it aims to find a time lag for which the signal's autocorrelation function exhibits a peak. We now briefly outline how the framewise version of the original algorithm works and then describe our modifications to obtain the differentiable version we refer to as *dYIN*.

Let x be an audio signal with samples $x(n) \in \mathbb{R}$ for time indices $n \in \mathbb{Z}$. To perform F0 estimation, the signal is divided into M overlapping frames x_m using a window of length $N \in \mathbb{N}$ and a hop length of $H \in \mathbb{N}$ samples. For each frame, indexed by $m \in [0 : M-1] := \{0, 1, \ldots, M-1\}$, YIN computes [4] the *cumulative mean normalized difference function* (CMNDF) d'_m defined by

$$d_m(\tau) = \sum_{n=0}^{N-1} \left[x_m(n) - x_m(n+\tau)\right]^2, \tag{1}$$

$$d'_m(\tau) = \begin{cases} 1, & \text{if } \tau = 0, \\ d_m(\tau) \Big/ \left[(1/\tau) \sum_{j=1}^{\tau} d_m(j)\right] & \text{else}, \end{cases} \tag{2}$$

where x_m is considered to be zero outside the interval $[0 : N-1]$. The CMNDF is computed for time lags $\tau \in [0 : \tau_{\max}]$ with $\tau_{\max} = \lceil F_\mathrm{s}/F_{0,\min} \rceil$, where F_s is the sampling frequency and $F_{0,\min}$ is a hyperparameter defining the lowest detectable F0 (both in Hz). For all frames and time lags, we assemble the values of the CMNDF as the entries of a matrix $\mathbf{D}^{(\mathrm{t})} \in \mathbb{R}^{(\tau_{\max}+1)\times M}$, which is illustrated in Fig. 1. YIN aims to find the smallest time lag $\hat{\tau}_m$ for which d'_m exhibits a local minimum, corresponding to the period of x_m. The F0 for that frame is then estimated as $\hat{y}_m = F_\mathrm{s}/\hat{\tau}_m$. To select the time lag $\hat{\tau}_m$, the original YIN algorithm [4] includes the following steps:

1) YIN employs an absolute threshold and selects the smallest time lag for which d'_m exhibits a local minimum with a value below that threshold, reducing subharmonic errors compared to choosing the global minimum of d'_m.
2) YIN applies parabolic interpolation around the selected minimum and then uses the parabola's minimum as estimate $\hat{\tau}_m$ to better deal with cases where the period is not an integer multiple of the sampling period.
3) YIN refines the initial F0 estimate by incorporating information from neighboring frames. Specifically, it uses the initial F0 estimates from adjacent frames to narrow the search range for τ in the current frame. The refined estimate is then obtained from a second pass of the algorithm using the restricted search range.

Since YIN only returns a scalar F0 estimate per frame, it implicitly formulates F0 estimation as a regression problem. For gradient-based optimization, however, it can be advantageous to reformulate regression problems as classification problems—a strategy that proved to be effective for other tasks [21], [22]. For the design of dYIN, we therefore reformulate F0 estimation as a classification problem, where the model outputs for each frame x_m a probability vector $\hat{\mathbf{y}}_m \in [0,1]^K$ with $K \in \mathbb{N}$. The vector's elements $\hat{\mathbf{y}}_m(k)$ are interpreted as probabilities associated with K predefined F0 classes. The F0 classes represent small, non-overlapping frequency ranges and are referred to by their center frequencies. We denote the vector of all center frequencies by $\mathbf{c}^{(\mathrm{f})} \in \mathbb{R}^K$. Following deep learning-based approaches to F0 estimation [8], [12], [13], [14] and motivated by the human perception of pitch [23], we define the F0 classes on a logarithmic frequency axis with center frequencies

$$\mathbf{c}^{(\mathrm{f})}(k) = F_{0,\min} \cdot 2^{k \cdot R/1200} \tag{3}$$

for all $k \in [0 : K-1]$, where R is the distance between the center frequencies of adjacent F0 classes, measured in cents. In our experiments, we choose $F_{0,\min} = 55.0\,\mathrm{Hz}$, $K = 720$, and $R = 10$, resulting in the highest detectable F0 class $\mathbf{c}^{(\mathrm{f})}(719) = 3499.73\,\mathrm{Hz}$. To obtain the probabilities $\hat{\mathbf{y}}_m$, we use the CMNDF d'_m as a starting point. Note that d'_m is differentiable w.r.t. all samples $x_m(n)$. Essentially, we convert the time lag axis of the CMNDF into a frequency axis. To this end, we first convert the center frequencies of all F0 classes into time lags $\mathbf{c}^{(\mathrm{t})} \in \mathbb{R}^K$ via

$$\mathbf{c}^{(\mathrm{t})}(k) = F_\mathrm{s}/\mathbf{c}^{(\mathrm{f})}(k). \tag{4}$$

[1] https://github.com/groupmm/df0

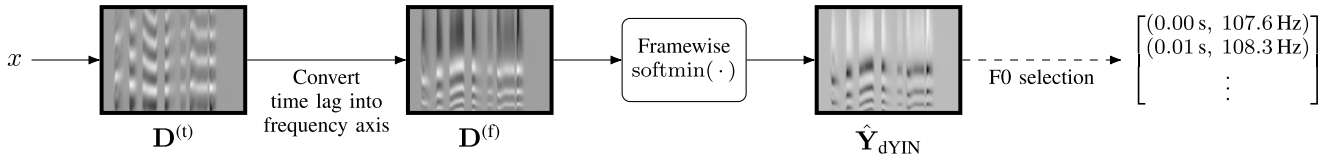

Fig. 1. Visualization of the dYIN algorithm. The cumulative mean normalized difference function (CMNDF) is computed for each frame, with the vertical axis of $\mathbf{D}^{(\mathrm{t})}$ representing time lag and the horizontal axis representing time. The time lag axis is then converted into a frequency axis ($\mathbf{D}^{(\mathrm{f})}$), and framewise application of the softmin function converts the values into probabilities, resulting in an F0 salience matrix $\hat{\mathbf{Y}}_{\mathrm{dYIN}}$. If needed, F0 selection approaches can process the probabilistic output into scalar F0 estimates.

We cannot directly evaluate the CMNDF at these time lags because the CMNDF is only defined for integer time lags, but $\mathbf{c}^{(\mathrm{t})}$ also contains non-integer time lags. Therefore, for each of the F0 classes, parabolic interpolation is used to derive a value from d'_m at the respective non-integer time lag. This step is similar to the parabolic interpolation applied in the original YIN algorithm, except that for dYIN we evaluate the CMNDF at fixed time lags rather than finding the minimum of an interpolating parabola. We denote the vector of interpolated CMNDF values by $\mathbf{d}_m^{(\mathrm{f})} \in \mathbb{R}^K$, and we gather these vectors for all frames in a matrix $\mathbf{D}^{(\mathrm{f})} \in \mathbb{R}^{K\times M}$. For an illustration of $\mathbf{D}^{(\mathrm{f})}$, we refer to Fig. 1. To obtain a probability vector $\hat{\mathbf{y}}_m$, we apply the softmin function

$$\hat{\mathbf{y}}_m = \operatorname{softmin}(\mathbf{d}_m^{(\mathrm{f})}) = \frac{\exp(-\mathbf{d}_m^{(\mathrm{f})})}{\sum_{k=0}^{K-1} \exp(-\mathbf{d}_m^{(\mathrm{f})}(k))}, \tag{5}$$

which ensures that $\sum_{k=0}^{K-1} \hat{\mathbf{y}}_m(k) = 1$. Stacking the vectors $\hat{\mathbf{y}}_m$ of all frames, we obtain a matrix $\hat{\mathbf{Y}}_{\mathrm{dYIN}} \in [0,1]^{K\times M}$, which describes F0 salience over time, see Fig. 1.

The original YIN algorithm includes two steps which we do not incorporate into dYIN: absolute thresholding and estimate refinement using a restricted search range. We omit these steps to attain good gradient flow for dYIN. Both of these steps could be incorporated by a masking operation that is applied after the softmin function, setting to zero the probabilities for F0 classes where the CMNDF exceeds the threshold or which are outside the restricted search range. However, during backpropagation, this would zero out the gradients w.r.t. the probabilities of the masked F0 classes. Therefore, masking would cause an information loss, limiting the effectiveness of gradient-based optimization.

However, we must distinguish between training and evaluation mode. Gradients are only required during training, and we can thus change the behavior of the algorithm during evaluation. In fact, we could apply YIN in its original form during evaluation and only use dYIN for training or if the F0 salience representation $\hat{\mathbf{Y}}_{\mathrm{dYIN}}$ is the desired output.

B. dSWIPE

The SWIPE algorithm [5] (Sawtooth Waveform Inspired Pitch Estimator) operates in the frequency domain and identifies the F0 by comparing the input signal to templates derived from the spectrum of a sawtooth waveform. For each frame, the algorithm compares the input signal to the F0-specific spectral templates and selects the F0 corresponding to the template with the highest similarity. In the following, we summarize how SWIPE[2] works and outline the modifications needed to develop its differentiable variant, *dSWIPE*.

For an audio signal x, SWIPE computes a set $\mathcal{X}$ of $J \in \mathbb{N}$ magnitude spectrograms using varying window sizes to capture the signal's spectral content at multiple time–frequency resolutions. Each spectrogram uses a hop size equal to half the window size, minimizing computational cost and redundancy [5], but leading to different temporal resolutions.

To allow for uniform handling, all spectrograms are resampled to a common time and frequency axis. For simplicity, we use linear interpolation to perform the resampling. The time axis is specified by a frame rate F_{s}/H, where H is the target hop length in samples. The frequency axis is constructed by uniformly sampling the *equivalent rectangular bandwidth* (ERB) scale at $L \in \mathbb{N}$ positions. The ERB scale, derived from psychoacoustics, approximates the bandwidths of auditory filters in human hearing [24], providing a perceptually meaningful representation of frequencies. Resampling spectrograms to the ERB scale thus allows the algorithm to implicitly weigh frequency bands according to their perceptual importance [5], improving accuracy and interpretability in pitch-related tasks. We summarize all resampled spectrograms in a tensor $\mathbf{X}_{\mathrm{ERB}} \in \mathbb{R}^{J\times L\times M}$. Both the set $\mathcal{X}$ and the tensor $\mathbf{X}_{\mathrm{ERB}}$ are illustrated in Fig. 2.

Furthermore, the magnitude of each spectrogram is compressed using the square-root function, which enhances the relative contribution of higher harmonics with lower amplitudes, making them more prominent in the analysis [5]. This step ensures that subtle harmonic components, which might otherwise be dominated by stronger frequencies, receive greater emphasis during the comparison process.

In the next step, the resulting ERB-based spectrogram representations are compared with the spectral templates. The set of F0 classes is chosen in the same way as for dYIN, and each class is represented by a spectral template. In SWIPE, spectral templates are designed to match the harmonic structure of signals from the respective F0 classes. Each template includes positive cosine lobes at the fundamental frequency and its harmonics, with negative lobes placed around them. A harmonically decaying envelope emphasizes lower harmonics to align with energy distributions of typical signals and to reduce subharmonic errors [5]. The description of the templates is not yet complete, but what we mentioned so far is visualized in Fig. 3(a) for the F0 class 1000 Hz.

[2]In this article, the term SWIPE refers to the algorithm known as SWIPE' in [5].

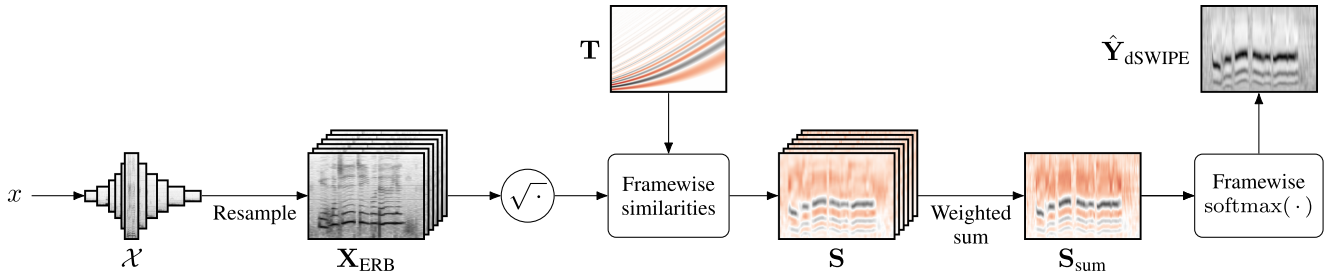

Fig. 2. Visualization of the dSWIPE algorithm. Spectral representations are extracted from the audio signal ($\mathcal{X}$) and aligned on a common time and frequency axis ($\mathbf{X}_{\text{ERB}}$). These are then compared with predefined spectral templates for different F0 classes ($\mathbf{T}$) to compute similarity scores ($\mathbf{S}$). The similarity scores are aggregated ($\mathbf{S}_{\text{sum}}$) and normalized to produce an F0 salience matrix ($\hat{\mathbf{Y}}_{\text{dSWIPE}}$). Red is used to visualize negative values.

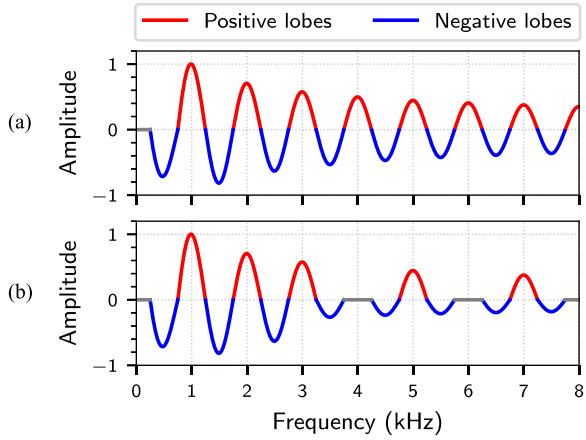

Fig. 3. Spectral templates for the F0 class 1000 Hz. (a) Template including all harmonics. (b) SWIPE template including the first and all prime harmonics.

To further improve accuracy, SWIPE uses only the first and prime harmonics (e.g., $2\times$, $3\times$, $5\times$, $7\times$ F0) instead of all harmonics, see Fig. 3(b). This approach considerably reduces the overlap between the spectrum of a given F0 and the templates corresponding to its subharmonics (e.g., $1/2$, $1/3$, $1/4$ F0) because for each template at most one of the signal's harmonics is captured by a positive lobe. Therefore, excluding non-prime harmonics reduces subharmonic errors [5]. The input signal's spectrum is compared to these templates, and the F0 corresponding to the best-matching template is selected as the F0 estimate.

Formally, let $\mathbf{T} \in \mathbb{R}^{L\times K}$ be a matrix containing all the K templates of length L as its columns, see Fig. 2 for an illustration. Since the templates are designed for comparison with ERB-based spectrograms, they are generated directly within the ERB scale. Using $\mathbf{T}$ and the spectrograms $\mathbf{X}_{\text{ERB}}$, we compute an F0 similarity tensor $\mathbf{S} \in \mathbb{R}^{J\times K\times M}$. Each entry $\mathbf{S}(j,k,m)$ represents the similarity between the mth frame of x and the spectral template corresponding to the fundamental frequency $\mathbf{c}^{(\text{f})}(k)$, computed using the jth window size. For all $j \in [0 : J-1]$, $k \in [0 : K-1]$, and $m \in [0 : M-1]$, this similarity is computed by

$$\mathbf{S}(j,k,m) = \frac{\langle\, \mathbf{T}(:,k),\ \sqrt{\mathbf{X}_{\text{ERB}}(j,:,m)}\,\rangle}{\|\max(\mathbf{T}(:,k),\,0)\| \cdot \left\|\sqrt{\mathbf{X}_{\text{ERB}}(j,:,m)}\right\|}. \quad (6)$$

This metric, a modified version of cosine similarity, normalizes the templates using the ℓ_2-norm of their positive components. Normalizing with only the positive components ensures that the similarity values approach one if the spectrum of a frame closely matches the positive part of a template [5]. This property is useful if the highest similarity is later used as a confidence metric, helping to determine whether a frame is pitched or unpitched. The colon is used to address all values in a dimension, and both the max operator and the square root are applied element-wise.

We have not yet discussed the selection of window sizes for computing the spectrograms in $\mathcal{X}$. For each F0 class, an optimal window size can be calculated that maximizes the overlap between the main lobes of the spectrum at the harmonic frequencies and the positive lobes of the template [5]. However, instead of calculating a separate spectrogram for each F0 class, Camacho and Harris [5] propose computing spectrograms only for window sizes that are powers of 2 and linearly combining the resulting similarities from these spectrograms [5]. This is achieved by computing a weighted sum of the tensor $\mathbf{S}$ along the window size dimension, with weights specific to each F0 class, producing an aggregated matrix $\mathbf{S}_{\text{sum}}$. For details on how the combining weights are calculated, we refer to (19)–(21) in [5].

Finally, following the approach used in YIN, framewise parabolic interpolation is applied around the maximum along the frequency axis of $\mathbf{S}_{\text{sum}}$. The peak position of the fitted parabola is then taken as the F0 estimate, allowing for more precise estimates if the true F0 is falling between the F0 class grid [5].

We now describe the modifications we make to obtain the differentiable dSWIPE. First of all, we again reformulate F0 estimation as a classification problem for the same reasons mentioned in Section II-A. For SWIPE, this change is very simple because with $\mathbf{S}_{\text{sum}}$ the algorithm already computes similarities for the predefined set of F0 classes. To convert these similarities into probabilities, we apply for each frame the softmax function across the frequency axis, resulting in an F0 salience matrix $\hat{\mathbf{Y}}_{\text{dSWIPE}} \in [0,1]^{K\times M}$. Furthermore, we modify the calculation of the similarity metric so that it is differentiable w.r.t. both the ERB-based spectrograms and the templates. Instead of normalizing the templates using the ℓ_2-norm of only their positive components, we normalize them using the standard ℓ_2-norm,

$$\mathbf{S}(j,k,m) = \frac{\langle\, \mathbf{T}(:,k),\ \sqrt{\mathbf{X}_{\text{ERB}}(j,:,m)}\,\rangle}{\|\mathbf{T}(:,k)\| \cdot \left\|\sqrt{\mathbf{X}_{\text{ERB}}(j,:,m)}\right\|}. \quad (7)$$

As a result of this modification, the similarities no longer reach values as close to one as they previously did. However, this has little impact on the method's F0 estimation performance

as observed in Section II-F. Since the similarities are now differentiable w.r.t. the template matrix $\mathbf{T}$, we can treat that matrix as trainable rather than fixed, enabling it to be learned using gradient descent.[3] There are several ways to implement learnable templates. We consider the following options:

1) *Fully trainable template matrix (*`F`*):* Each entry in $\mathbf{T}$ is treated as an independent parameter, providing maximum flexibility but requiring substantial training data for each F0 class to achieve robust generalization.
2) *Prototype-based templates (*`P`*):* A single prototype is learned, with the columns of $\mathbf{T}$ dynamically generated in each iteration by stretching this prototype accordingly. Linear interpolation is used to evaluate the prototype at the required positions. Although this method offers less flexibility than the first approach, it can achieve generalization across F0 classes through parameter sharing.
3) *Learned components (*`C`*):* Specific elements of the original template generation process, such as the cosine lobe or the envelope decay parameter, can be learned. This enables us to use gradient descent to find a data-optimal parameter setting, eliminating the need for manual tuning as done in [5].

In the following, we will refer to these options by using the abbreviations written in parenthesis, e.g., `dSWIPE-P` for the method that generates templates from a prototype.

C. Example

In Fig. 4, we present an example to illustrate and compare the F0 salience representations produced by dYIN and dSWIPE using an excerpt from the MIR-1K dataset [25], which contains only singing voice. For context, the audio's magnitude spectrogram, CREPE's output [8], and the dataset's reference annotations are also included. For all our experiments, we use a PyTorch version of CREPE.[4] To ensure consistency with the other methods, CREPE's final sigmoid activation was replaced with a softmax function.

During voiced intervals, the spectrogram displays strong amplitudes at overtones, while neither dYIN nor dSWIPE produce peaks at frequencies above the annotated F0. However, both methods assign noticeable probabilities to subharmonics, reflecting some vulnerability to subharmonic errors. dYIN generates similar probabilities for the F0 and its subharmonics, whereas dSWIPE assigns reduced probabilities to subharmonics, highlighting improved robustness. In comparison, CREPE's output reveals low-level peaks both above and below the F0, especially near octave shifts.

D. Loss Computation

Both dYIN and dSWIPE approach F0 estimation as a classification task, producing framewise F0 probabilities as outputs. This makes it straightforward to integrate them into deep learning pipelines, where an F0 classification loss is calculated and gradients can be backpropagated through the algorithms. Framewise F0 annotations y_m in Hz can be converted into target vectors $\mathbf{y}_m \in [0,1]^K$ in different ways. The simplest option is to convert them into one-hot vectors, where the F0 class closest to the annotated F0 gets assigned all probability mass. For all frames with $y_m > 0$, these vectors are defined by

$$\mathbf{y}_m(k) = \begin{cases} 1, & \text{if } k = \lfloor \log_2(y_m / F_{0,\min}) \cdot \frac{1200}{R} \rceil, \\ 0, & \text{else}, \end{cases} \tag{8}$$

where $\lfloor \cdot \rceil$ denotes the rounding operation. Alternatively, Gaussian blurring can be applied across the frequency axis, providing smoother targets and reducing the penalty for minor deviations from the reference [26]. While [26] applies Gaussian blurring after rounding to the nearest F0 class, we instead apply it directly to the targets, bypassing the intermediate rounding step as in [8].

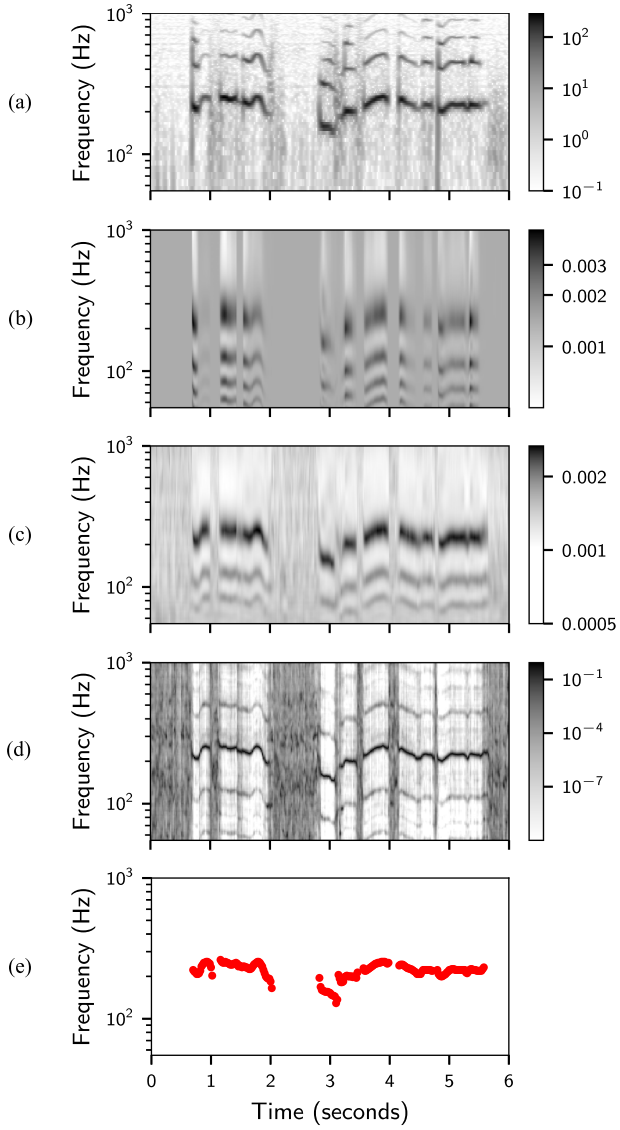

Fig. 4. Comparison of F0 salience representations for a singing voice excerpt from the MIR-1K [25] dataset. (a) Spectrogram of the original audio signal. F0 salience representation based on (b) dYIN, (c) dSWIPE, and (d) CREPE. (e) Reference F0 annotations from the MIR-1 K dataset.

[3] Although gradient-based optimization is also feasible with the similarity metric in (6), using a subgradient as for the rectified linear unit (ReLU), the lack of proper normalization may result in learned templates with disproportionately large negative values.

[4] [Online]. Available: https://github.com/gudgud96/torchcrepeV2

This approach ensures more accurate target representations. The entries of the Gaussian-blurred target vector are then calculated as

$$\mathbf{y}_m(k) = Z_m \cdot \exp\left(\frac{-(k \cdot R - 1200 \cdot \log_2(y_m/F_{0,\min}))^2}{2\sigma^2}\right), \tag{9}$$

where $\sigma > 0$ is the Gaussian's standard deviation in cents, and the normalizing factors $Z_m \in \mathbb{R}$ are chosen so that $\sum_{k=0}^{K-1} \mathbf{y}_m(k) = 1$ for all frames m. As loss function, we choose categorical cross entropy,

$$\mathcal{L}(\hat{\mathbf{y}}_m, \mathbf{y}_m) = -\sum_{k=0}^{K-1} \mathbf{y}_m(k) \cdot \log(\hat{\mathbf{y}}_m(k)), \tag{10}$$

which is the standard choice for classification problems.

E. F0 Selection Approaches

As discussed in Section II-A, the main purpose of dYIN and dSWIPE is to enable gradient backpropagation during training. For evaluation, where differentiability is not required, the original algorithms YIN and SWIPE could be used instead. Nonetheless, we provide a brief overview of common methods for deriving scalar F0 estimates from salience representations for evaluation purposes. We only describe framewise selection approaches and do not consider post-processing methods such as Viterbi decoding [7].

1) *Argmax:* This simplest option selects for each frame the F0 class with the highest salience.
2) *Parabolic interpolation:* This option applies parabolic interpolation to the salience representation, fitting for each frame a parabola around the F0 class with highest probability. The abscissa of the parabola's peak is then taken as F0 estimate, following the procedure used in YIN [4] and SWIPE [5].
3) *Local weighted average:* This method first identifies for each frame the F0 class with the highest probability. Then, the final F0 estimate is computed as a weighted average of neighboring frequencies within a small window, using their predicted probabilities as weights. For further details, see [12].

In the following, we will quantitatively compare the performance of both dYIN and dSWIPE using these F0 selection approaches to the original algorithms YIN and SWIPE.

F. Quantitative Comparison

To evaluate the performance of our algorithms, we use the MIR-1 K dataset [25], which comprises 1000 short clips of karaoke-style recordings. Each clip includes separate audio tracks for the monophonic singing voice and the accompaniment, along with manually created framewise F0 annotations for the singing voice. As a performance metric, we use `mir_eval`'s [27] raw pitch accuracy (RPA), which measures the percentage of frames where the predicted F0 deviates by less than 50 cents from the annotated F0. Additionally, we employ raw chroma accuracy (RCA), which extends RPA by ignoring octave errors. Throughout this article, RPA and RCA are given in percentages.

TABLE I
PERFORMANCE COMPARISON ON THE VOCAL TRACKS OF MIR-1 K

	(d)YIN		(d)SWIPE	
	RPA	RCA	RPA	RCA
F0 Selection Approach				
Argmax	95.22	95.66	95.89	96.48
Parabolic interpolation	95.05	95.49	95.93	96.53
Local weighted average	95.26	95.70	95.89	96.49
Original (`libf0`)	95.20	96.63	96.35	96.89

TABLE II
RPA UNDER NOISY CONDITIONS, EVALUATED ON MIR-1 K

	clean	20 dB	10 dB	0 dB
dYIN (F0 Selection: Argmax)	95.22	93.88	83.59	42.11
YIN (`libf0`, threshold 0.1)	95.20	91.61	62.84	13.59
YIN (`libf0`, threshold 0.3)	92.92	91.70	81.40	32.09
YIN (`libf0`, threshold 0.5)	87.31	86.48	80.28	45.70
dSWIPE (F0 Selection: Argmax)	95.89	93.35	77.68	34.67
SWIPE (`libf0`)	96.35	93.88	78.84	35.52

First, we use the monophonic singing voice as input to the algorithms. The results are shown in Table I, where we compare the performance of both dYIN and dSWIPE in combination with the F0 selection approaches from Section II-E. For reference, we also report the performance of the original algorithms YIN and SWIPE, using the implementations from the `libf0` [28] library. For YIN, we set an absolute threshold of 0.1. We can observe that the performance is almost identical for all selection approaches, e.g., for dYIN with an RPA of 95.22 using "Argmax", 95.05 using "Parabolic interpolation", and 95.26 using "Local weighted average". The same holds for dSWIPE. Furthermore, we observe that dYIN and dSWIPE achieve performance levels comparable to their original counterparts. For instance, dSWIPE combined with "Argmax" achieves an RCA of 96.48, only slightly below the 96.89 achieved by SWIPE, demonstrating that the modifications introduced have minimal impact on the performance. For the rest of this article, we use the "Argmax" F0 selection approach, as our primary focus is to evaluate the underlying F0 estimators themselves.

To further assess the robustness of our algorithms, we evaluate their performance under noisy conditions. Specifically, we create mixed inputs by combining the monophonic singing voice tracks from MIR-1 K with their corresponding accompaniment tracks at various signal-to-noise ratios (SNRs), and use these mixtures as input to the algorithms. The predicted F0 values are still evaluated against the annotations of the monophonic singing voice. The results are summarized in Table II. As a reference, we first examine the performance of the original YIN algorithm, using three different values for its absolute threshold parameter. For clean inputs (high SNR), the lowest threshold of 0.1 yields the best performance, with an RPA of 95.20. At lower SNRs, however, higher thresholds are more effective. For instance, at an SNR of 10 dB, a threshold of 0.3 results in the best performance with an RPA of 81.40. In comparison, dYIN

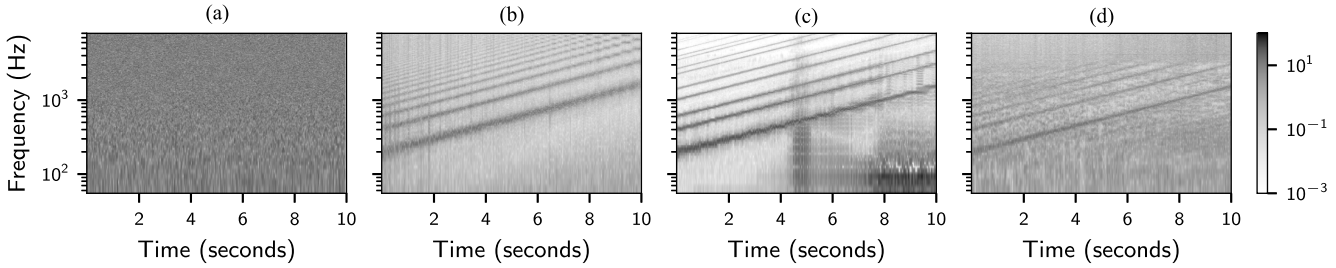

Fig. 5. Results of the reverse engineering experiment using gradient descent with different F0 estimators. (a) Spectrogram of the initial random noise signal, serving as the starting point. Spectrograms of the optimized signals after gradient descent with (b) dYIN, (c) dSWIPE, and (d) CREPE. The results in (b) and (c) show clearer harmonic structures than (d), highlighting the greater interpretability of the differentiable model-based F0 estimators dYIN and dSWIPE.

achieves a performance comparable to the best threshold setting of YIN across all tested SNRs, for example reaching an RPA of 83.59 at an SNR of 10 dB. While this makes dYIN appear even more robust than YIN in this particular setting—achieving comparable performance without threshold tuning—this observation may be specific to the YIN implementation used here. In particular, the way the CMNDF is computed differs across implementations, and subtle variations in this step can substantially influence performance and robustness. A more detailed analysis of these implementation-dependent effects is beyond the scope of this work. Overall, we conclude that dYIN retains the robustness of YIN under noisy conditions, even though it omits absolute thresholding and estimate refinement using a restricted search range. Likewise, we compare the robustness to noise for dSWIPE and SWIPE. For instance, at 10 dB SNR, dSWIPE reaches an RPA of 77.68, compared to 78.84 for SWIPE. Although this moderate difference is consistent across all tested SNRs, the overall performance under noisy conditions can still be considered comparable.

III. Case Studies

In this section, we present three case studies to investigate the behavior of our differentiable signal processing-based F0 estimators, dYIN and dSWIPE, and compare them with the deep learning-based CREPE.

A. Reverse Engineering

The first experiment explores the input signals required by each F0 estimator to produce a user-specified F0 trajectory. To find such signals, we create an initial signal $x^{(\text{init})}$ consisting of random noise, which serves as the starting point for optimization. We do not assume any signal model, but each sample of the signal is treated as an independent parameter to be optimized. Additionally, we define a target F0 trajectory that increases exponentially in frequency from 200 Hz to 1500 Hz over a duration of 10 s, representing the F0 contour of a chirp-like signal. We convert this F0 trajectory into target vectors using Gaussian blurring with a standard deviation of $\sigma = 25$ cents as described in Section II-D.

For each F0 estimator, starting with $x^{(\text{init})}$ as input, we iteratively compute the classification loss between the estimator's predictions and the target vectors and apply gradient descent w.r.t. the samples of the time-domain input signal. We perform 2000 optimization steps each, after which the loss converged for all F0 estimators. This reverse engineering process relies on the differentiability of the F0 estimators and produces optimized input signals tailored to the respective models, which we denote by $x^{(\text{opt})}_{\text{dYIN}}$, $x^{(\text{opt})}_{\text{dSWIPE}}$, and $x^{(\text{opt})}_{\text{CREPE}}$.

The spectrograms of $x^{(\text{init})}$ and the reverse-engineered signals are shown in Fig. 5. The time-domain signals were max-normalized before spectrogram computation to improve visualization and facilitate comparison. The results reveal that $x^{(\text{opt})}_{\text{dYIN}}$ (Fig. 5(b)) and $x^{(\text{opt})}_{\text{dSWIPE}}$ (Fig. 5(c)) exhibit a clear harmonic structure with many overtones. Although dYIN's periodic signal model is theoretically satisfied by any periodic signal, including a single sinusoid, $x^{(\text{opt})}_{\text{dYIN}}$ is not merely a sinusoidal chirp but includes multiple harmonics, being similar to real-world periodic signals.

In contrast, $x^{(\text{opt})}_{\text{dSWIPE}}$ (Fig. 5(c)) mainly consists of the fundamental frequency and its prime harmonics, reflecting the design of dSWIPE's spectral templates. Additionally, frequency components between the harmonics are more strongly suppressed in $x^{(\text{opt})}_{\text{dSWIPE}}$ compared to $x^{(\text{opt})}_{\text{dYIN}}$. This suppression occurs because the spectral templates include negative lobes around the positive ones, which actively penalize the presence of frequencies at the positions of these negative lobes. However, the softmax activation function introduces an additional effect: Increasing the probability of a specific F0 class can be achieved either by increasing the similarity to that class's template or by reducing the similarities to the templates of all other F0 classes. This effect becomes particularly noticeable starting at the 5-second mark in Fig. 5(c), where $x^{(\text{opt})}_{\text{dSWIPE}}$ shows significant low-frequency noise. This noise arises during the optimization process, as reducing the similarities to the templates of low F0 classes—whose negative lobes overlap this frequency range—unintentionally amplifies low-frequency components.

Finally, $x^{(\text{opt})}_{\text{CREPE}}$ (Fig. 5(d)) also shows harmonic structure, but it is less pronounced and more noisy compared to $x^{(\text{opt})}_{\text{dYIN}}$. Further, we cannot see as many overtones, and the harmonic structure does not continue above 4 kHz, which demonstrates the improved interpretability of model-based F0 estimation methods compared to deep learning-based approaches.

B. Vocal Melody Extraction

In our second case study, we address the task of vocal melody extraction, which involves estimating the F0 trajectory of a

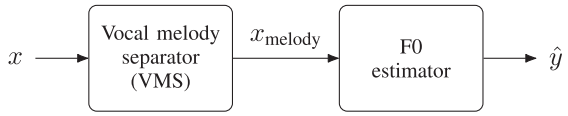

Fig. 6. Block diagram of our modular vocal melody extraction pipeline.

monophonic singing voice within a polyphonic music context [29]. Early signal-processing approaches to this problem are based on salience estimation [30], [31] or source separation [32], [33]. In contrast, most recent approaches apply deep learning [34], [35], [36], [37], [38], either directly predicting the vocal melody from the mixture input [34], [35], [36], [37], or by first training a model for explicit source separation and subsequently fine-tuning it for vocal melody extraction [38].

1) Processing Pipeline: Here we address vocal melody extraction using a modular processing pipeline similar to the early source separation-based approaches, consisting of a vocal melody separator (VMS) and an F0 estimator as shown in Fig. 6. The VMS takes as input the mixture audio signal x and outputs an audio signal x_{melody} which should only contain components related to the vocal melody. Since the accompaniment may also include singing voices that are not considered part of the melody, this task differs slightly from singing voice separation. Having x_{melody} as input, an F0 estimator then estimates the vocal melody $\hat{y}$. Note that we do not require the VMS to output a clean version of the separated melody line, but rather to prepare an optimized input for the subsequent F0 estimator, cf. Section III-A.

As VMS, we use common music source separation models such as Open-Unmix (UMX) [39] or the Mel-RoFormer (MRF) [38]. For end-to-end training, a differentiable F0 estimator is required, where we use dYIN, dSWIPE, or CREPE. We also consider non-trainable pipelines with Melodia [30] as F0 estimator.[5] To describe these pipelines, we adopt the naming convention `VMS–F0`, where bold text indicates components trained within the end-to-end pipeline, and an asterisk marks pretrained components. For example, **`UMX`**`–CREPE`* represents a pipeline where we train UMX from scratch using gradients from CREPE, and CREPE is pretrained for F0 estimation but kept frozen during the training of the pipeline.

2) Datasets and Training Details: Following [35], [37], we select all clips from MIR-1 K [25] and 35 tracks with vocal melody from MedleyDB [40] as training data. MedleyDB is a multitrack dataset designed for various music information retrieval tasks, and it includes tracks with detailed vocal melody annotations. Note that both MIR-1 K and the selected tracks from MedleyDB contain segments where singing voices are part of the accompaniment.

If model training is involved, we use the Adam optimizer [41] with learning rate 1e-6, segments of 5 s length at a sampling rate of 16 kHz, a batch size of 16, and optimize for up to 1500 epochs. In this case study, we use binary F0 targets as described in (8), having achieved better results than Gaussian-blurred targets in preliminary experiments.

TABLE III
RESULTS OF THE VOCAL MELODY EXTRACTION EXPERIMENT

	Training GPU memory	ADC2004		MIREX05	
		RPA	RCA	RPA	RCA
`None–dYIN`	✗	50.61	64.96	38.69	58.03
`UMX``–dYIN`	1.0 GB	52.93	67.65	58.99	71.16
`MRF`*`–dYIN`	✗	79.61	82.22	91.16	91.49
`None–dSWIPE`	✗	50.98	59.26	32.39	42.27
`UMX``–dSWIPE`	1.5 GB	71.42	75.46	81.55	82.68
`MRF`*`–dSWIPE`	✗	89.73	89.89	93.68	93.77
`None–CREPE`*	✗	73.68	78.85	81.37	83.46
`UMX``–CREPE`*	24.0 GB	74.16	79.79	83.55	85.70
`MRF`*`–CREPE`*	✗	91.23	91.41	94.24	94.29
`None–Melodia`	✗	71.34	73.14	77.38	78.50
`MRF`*`–Melodia`	✗	78.36	78.66	83.35	84.98
Shao et al. [37]	✗	85.70	85.80	87.60	87.60

For evaluation, we use 12 tracks from ADC2004 and 9 tracks from MIREX05[6] which contain vocal melody with accompaniment. We consider the tracks from ADC2004 as more difficult, as they include opera recordings, synthesized vocal melody, or singing voices in the accompaniment. In contrast, the MIREX05 tracks are pop and jazz recordings, where no additional singing voices are present beyond the vocal melody.

3) Experimental Results: To evaluate the effectiveness of our approach, we compare melody extraction pipelines with different configurations: a VMS pretrained for singing voice separation, a VMS trained from scratch using gradients from the F0 estimator, and a pipeline without VMS. As performance metrics we use RPA and RCA as introduced in Section II-F. The results of our experiment are presented in Table III.

The pipelines without VMS and that use model-based F0 estimators show limited performance. For instance, on ADC2004, `None–dYIN` and `None–dSWIPE` achieve RPAs of only 50.61 and 50.98, respectively, and even worse performance on MIREX05. This is expected since these F0 estimators are designed to receive monophonic inputs, while the test datasets contain polyphonic music. In contrast, `None–CREPE`* performs much better, achieving RPAs of 73.68 and 81.37 on ADC2004 and MIREX05, respectively. This performance is particularly impressive when compared to the approach of Shao et al. [37], which is among the current state of the art in vocal melody extraction, achieving RPAs of 85.70 and 87.60 on these datasets. However, CREPE's training data includes MIR-1 K and MedleyDB, both of which provide not only the monophonic audios corresponding to the melody annotations but also the accompaniment. It remains unclear whether CREPE was trained using the monophonic audios as inputs or the mixtures. In the latter case, CREPE would have effectively been trained for melody extraction, which would explain its good performance in this experiment. The pipeline `None–Melodia` also performs well, achieving RPAs of 71.34 and 77.38 on ADC2004 and MIREX05. This good performance is expected, as Melodia is specifically designed as a melody extraction algorithm.

[5]https://github.com/MTG/essentia

[6]http://labrosa.ee.columbia.edu/projects/melody/

When using a randomly initialized UMX model as VMS and training the pipeline in an end-to-end fashion, the results vary considerably between configurations and test datasets. For instance, while **UMX**–dYIN shows only a slight improvement over None–dYIN on ADC2004 with an RPA of 52.93 compared to 50.61, it demonstrates a substantial performance boost on MIREX05, achieving an RPA of 58.99 compared to 38.69. Still, the performance of **UMX**–dYIN is far behind that of None–CREPE*, indicating that the gradients provided by dYIN are not sufficient to successfully train a model like UMX. The pipeline **UMX**–dSWIPE shows strong improvements over None–dSWIPE with an RPA of 71.42 compared to 50.98 on ADC2004 and an RPA of 81.55 compared to 32.39 on MIREX05. On MIREX05, **UMX**–dSWIPE even reaches the RPA of 81.37 achieved by None–CREPE*, highlighting dSWIPE's potential for being used in end-to-end trainable pipelines. For **UMX**–CREPE*, we observe only small improvements over None–CREPE*, with an RPA of 74.16 compared to 73.68 on ADC2004 and 83.55 compared to 81.37 on MIREX05. This suggests that CREPE may already have learned to focus on the melody line.

In Table III, we also provide the GPU memory requirements for training these pipelines. Both **UMX**–dYIN and **UMX**–dSWIPE are highly resource-efficient, requiring only 1.0 GB and 1.5 GB of memory, respectively, making them suitable for machines with limited GPU capacity. In contrast, training **UMX**–CREPE* demands with 24.0 GB significantly more memory due to the model size of CREPE.

The best results are achieved by pipelines utilizing a pretrained MRF model[7] as VMS, demonstrating this model's outstanding ability to separate singing voice from accompaniment. Having the singing voice estimate of MRF as input, all differentiable F0 estimators produce very good vocal melody estimates for MIREX05, with RPAs of 91.16 for dYIN, 93.68 for dSWIPE, and 94.24 for CREPE. These pipelines thereby outperform direct approaches to vocal melody transcription such as the method by Shao et al. [37]. For ADC2004, the results are less favorable because MRF's singing voice estimates include the background singers, which, however, are not considered as melody.

4) Qualitative Examples: Due to the design of our pipeline with x_{melody} as an interpretable intermediate representation, we can now inspect the properties of that representation in different pipelines. In Fig. 7, we show spectrograms for an audio excerpt from the ADC2004 dataset. In particular, in Fig. 7(b) and (c), we compare the spectrograms of x_{melody} originating from the pipelines **UMX**–dSWIPE and MRF*–dSWIPE, where the MRF model of the latter is trained to perform singing voice separation but was not trained using the gradients from dSWIPE. As mentioned before, the goal of the VMS is not to perform perfect source separation but to transform the mixture signal into a signal from which the vocal melody can be well estimated by the subsequent F0 estimator. This behavior is evident in Fig. 7(b), within the red frame, where we observe characteristics previously seen in dSWIPE's optimized input in Fig. 5(c): Only

[7][Online]. Available: https://github.com/KimberleyJensen/Mel-Band-Roformer-Vocal-Model

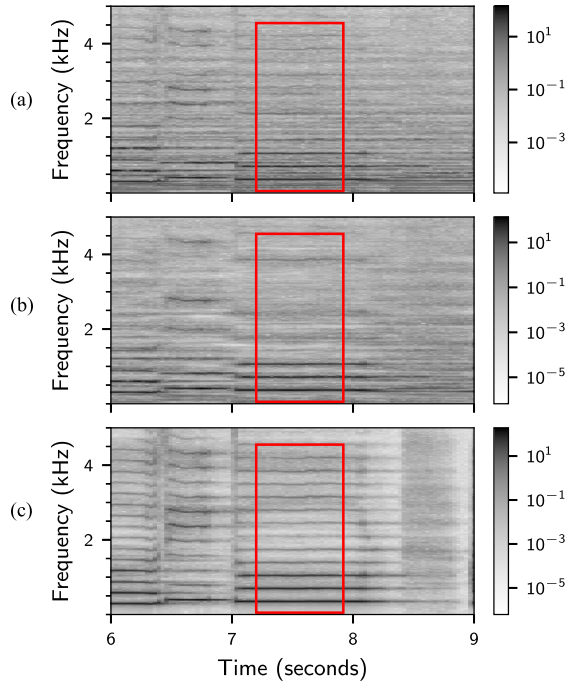

Fig. 7. Comparison of the intermediate representations x_{melody} for different vocal melody extraction pipelines using an audio excerpt from ADC2004 (vocal melody with accompaniment). (a) Spectrogram of the original mixture x. Spectrogram of x_{melody} obtained from the pipelines (b) **UMX**–dSWIPE and (c) MRF*–dSWIPE.

the first and the prime harmonics are visible, maximizing the similarity with one of dSWIPE's spectral templates. Also, the higher harmonics appear as blurred rather than sharp lines, resembling the cosine lobes of the templates. In contrast, the MRF model produces a signal x_{melody} that contains all harmonics without any blurring, see Fig. 7(c).

5) Summary: Although best results are achieved by leveraging a pretrained source separation model, this experiment indicates the potential of backpropagating through the differentiable model-based F0 estimators dYIN and dSWIPE. While the training of a music source separation model such as MRF requires separated stems and is very resource-expensive [38], training a pipeline such as **UMX**–dSWIPE only requires mixed audio and the corresponding melody annotations and is computationally efficient. Furthermore, compared to direct approaches such as [37], **UMX**–dSWIPE benefits from improved interpretability, providing an intermediate source-separation-like representation.

C. Timbre-Specific F0 Estimation

In this final case study, we aim to demonstrate the potential of training dSWIPE as a timbre-specific F0 estimator by optimizing its F0-dependent spectral templates for a specific type of input—in this case, violin recordings. While the authors of [42] consider transcribing the violin pitch from a mixture with piano, we here only consider monophonic violin recordings without any accompaniment. For this purpose, we use the ViolinEtudes dataset [43], which comprises 925 monophonic violin recordings of pedagogical pieces performed by 21 distinct players, along with framewise F0 annotations for each recording.

TABLE IV
RESULTS OF TIMBRE-SPECIFIC F0 ESTIMATION ON THE VIOLINETUDES TEST SET

	#parameters	RPA	RCA
YIN	0	63.60	71.58
SWIPE	0	78.28	79.11
CREPE*	22.2 M	81.43	82.37
dSWIPE-F	236 k	67.34	70.53
dSWIPE-P	420	79.40	81.25
dSWIPE-C	51	78.26	79.12

We split the dataset into a training set of 694 recordings and a test set of 231 recordings, ensuring no overlap in performers between the two sets. This performer-based split avoids the "album effect" [44], which can lead to overly optimistic results when tracks with identical recording conditions are included in both training and test sets. This setup enables a more robust evaluation of the model's ability to generalize across varying recording conditions and timbral characteristics.

In Section II-B, we presented three approaches to make dSWIPE trainable. We explore all these options: **dSWIPE-F** has a fully trainable template matrix, **dSWIPE-P** learns a prototype shared by the templates of all F0 classes, and **dSWIPE-C** only learns certain components, in our case a prototype lobe and the harmonic decay factor from which all templates are constructed. All trainable parameters were initialized randomly, except the harmonic decay factor which was initialized with its default value of 0.5. In Table IV, we provide for all options the respective numbers of trainable parameters and the performance metrics when evaluated on the ViolinEtudes test set. Only the methods printed in bold were trained on the ViolinEtudes training set.

Among the untrained baselines, YIN shows limited performance with an RPA of 63.60, while SWIPE achieves considerably better results, reaching an RPA of 78.28. This is only slightly surpassed by CREPE*, which reaches an RPA of 81.43. Among the trained methods, **dSWIPE-F** with individually learned templates does not generalize well, achieving an RPA of 67.34 which suggests overfitting. In contrast, template sharing of **dSWIPE-P** serves as an effective regularization strategy, resulting in an RPA of 79.40, slightly improving over the training-free SWIPE. Sharing a single prototype template introduces a strong inductive bias: While learning each template individually allows maximum flexibility (**dSWIPE-F**), generating all templates from a shared prototype enforces consistent behavior across F0 classes and thus supports generalization across F0 classes (**dSWIPE-P**). Finally, **dSWIPE-C**, which learns only a prototype lobe and the harmonic decay factor, also yields strong results with an RPA of 78.26, essentially performing as good as SWIPE.

While the trained methods perform similarly compared to the original SWIPE (**dSWIPE-P** and **dSWIPE-C**), and in one case less effectively (**dSWIPE-F**), the findings are still noteworthy. Specifically, this experiment demonstrates that, without relying on explicit engineering knowledge, it is possible to learn spectral templates that perform as effectively as the hand-crafted templates of the original SWIPE. Although we are not aware of any cases where the original SWIPE templates fail, we now have a technique to identify suitable templates for such situations, underscoring the potential of data-driven approaches in designing adaptable and task-specific components.

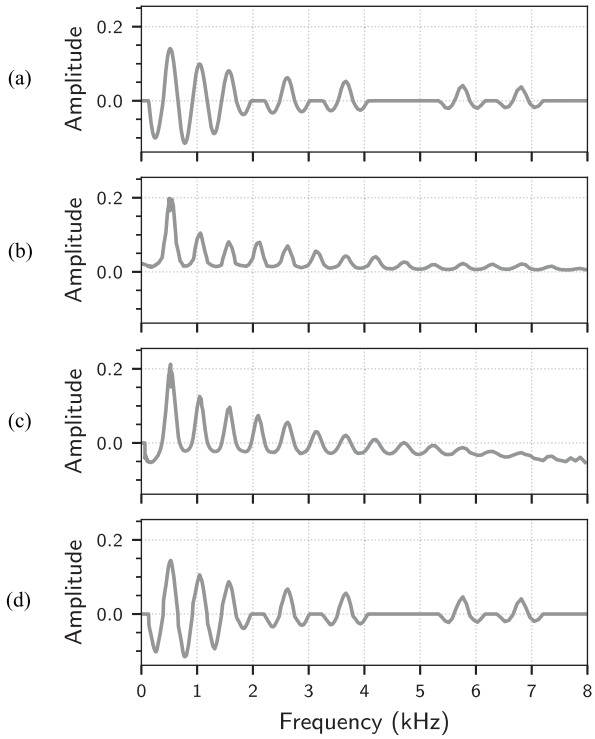

Fig. 8. Normalized spectral templates for the F0 class 523 Hz. (a) Original template of SWIPE. Learned template of (b) **dSWIPE-F**, (c) **dSWIPE-P**, and (d) **dSWIPE-C**.

Finally, we compare the spectral templates of all dSWIPE variations in Fig. 8. To facilitate comparison, we show the ℓ_2-normalized templates. While the original template in (a) contains both positive and negative lobes, the template learned by **dSWIPE-F** in (b) only shows positive lobes, being one potential reason for the limited performance. Additionally, we notice that the template includes all harmonics, rather than just the first and prime harmonics as in the original template, potentially causing more subharmonic errors. In contrast, the shared template of **dSWIPE-P** in (c) also includes small negative lobes, thus being more similar to the original template in (a). However, in (c), we notice a baseline drift, which could be due to the use of the softmax function, helping to reduce the similarity to all incorrect F0 classes. The template of **dSWIPE-C** in (d) is constructed like the original template, but using a learned lobe and an optimized harmonic decay factor. As already suggested by the quantitative results, the templates in (d) and (a) exhibit strong similarity, with the learned lobe closely resembling a cosine lobe and the harmonic decay factor of 0.476 being only slightly lower than the 0.5 seen in (a).

These observations underline the sophisticated design of the original SWIPE's templates. Despite being hand-crafted, they achieve a good performance, proving their effectiveness in capturing the key spectral features for F0 estimation. This balance between hand-crafted and learned approaches provides valuable insights into the strengths of both methodologies.

IV. Conclusion

In this article, we introduced dYIN and dSWIPE, differentiable variants of the classical F0 estimation algorithms YIN and SWIPE. These methods bridge the gap between the paradigms of classical signal processing and deep learning by preserving interpretability and computational efficiency while enabling gradient-based optimization.

Through three case studies, we demonstrated the versatility and potential of these models. In reverse engineering, dYIN and dSWIPE produced smoother and more interpretable gradients compared to NN-based methods like CREPE. For vocal melody extraction, dSWIPE showed strong performance in end-to-end pipelines, being particularly useful for resource-limited scenarios or if pretrained source separation models are unavailable. Additionally, the timbre-specific F0 estimation experiment highlighted that dSWIPE's templates can be optimized for specific tasks, achieving results comparable to hand-crafted templates without extensive manual engineering.

These findings emphasize the broader potential of differentiable models like dYIN and dSWIPE. Beyond traditional F0 estimation, their dual role as signal processors and trainable components opens new possibilities for resource-efficient, interpretable, and adaptable systems. Future work will focus on integrating these methods into larger multi-task pipelines and exploring self-supervised and unsupervised learning contexts, thereby expanding their applicability across diverse audio processing tasks.

Acknowledgment

The authors thank Simon Schwär for proof-reading the manuscript. The International Audio Laboratories Erlangen are a joint institution of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) and Fraunhofer Institute for Integrated Circuits IIS.

References

[1] F. Esqueda, V. Välimäki, and J. Parker, "Barberpole phasing and flanging illusions," in *Proc. Int. Conf. Digit. Audio Effects*, Trondheim, Norway, 2015, pp. 87–94.

[2] P. Boersma, "Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound," *Proc. Inst. Phonetic Sci.*, vol. 17, no. 1193, 1993, pp. 97–110.

[3] D. Talkin, "A robust algorithm for pitch tracking (RAPT)," in *Speech Coding and Synthesis*, W. B. Kleijn and K. K. Paliwal, Eds. Amsterdam, The Netherlands: Elsevier, 1995, ch. 14, pp. 495–518.

[4] A. de Cheveigné and H. Kawahara, "YIN, a fundamental frequency estimator for speech and music," *J. Acoust. Soc. Amer.*, vol. 111, no. 4, pp. 1917–1930, 2002.

[5] A. Camacho and J. G. Harris, "A sawtooth waveform inspired pitch estimator for speech and music," *J. Acoust. Soc. America*, vol. 124, no. 3, pp. 1638–1652, 2008.

[6] L. N. Tan and A. Alwan, "Multi-band summary Correlogram-based pitch detection for noisy speech," *Speech Commun.*, vol. 55, no. 7–8, pp. 841–856, 2013.

[7] M. Mauch and S. Dixon, "pYIN: A fundamental frequency estimator using probabilistic threshold distributions," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, Florence, Italy, 2014, pp. 659–663.

[8] J. W. Kim, J. Salamon, P. Li, and J. P. Bello, "CREPE: A convolutional representation for pitch estimation," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, Calgary, Canada, 2018, pp. 161–165.

[9] L. Ardaillon and A. Roebel, "Fully-convolutional network for pitch estimation of speech signals," in *Proc. Annu. Conf. Int. Speech Commun. Assoc.*, Graz, Austria, 2019, pp. 2005–2009.

[10] J. Engel, R. Swavely, L. H. Hantrakul, A. Roberts, and C. Hawthorne, "Self-supervised pitch detection by inverse audio synthesis," in *Proc. Int. Conf. Mach. Learn., Workshop Self-Supervision Audio Speech*, Vienna, Austria, 2020.

[11] B. Gfeller, C. Frank, D. Roblek, M. Sharifi, M. Tagliasacchi, and M. Velimirovic, "SPICE: Self-supervised pitch estimation," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 28, pp. 1118–1128, 2020.

[12] S. Singh, R. Wang, and Y. Qiu, "DeepF0: End-to-end fundamental frequency estimation for music and speech signals," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, Toronto, Canada, 2021, pp. 61–65.

[13] W. Wei, P. Li, Y. Yu, and W. Li, "HarmoF0: Logarithmic scale dilated convolution for pitch estimation," in *Proc. IEEE Int. Conf. Multimedia Expo*, Taipei, Taiwan, 2022, pp. 1–6.

[14] W. Chung, D. Kim, S. Chung, and H. Kang, "MF-PAM: Accurate pitch estimation through periodicity analysis and multi-level feature fusion," in *Proc. Annu. Conf. Int. Speech Commun. Assoc.*, Dublin, Ireland, 2023, pp. 4499–4503.

[15] A. Riou, S. Lattner, G. Hadjeres, and G. Peeters, "PESTO: Pitch estimation with self-supervised transposition-equivariant objective," in *Proc. Int. Soc. Music Inf. Retrieval Conf.*, Milano, Italy, 2023, pp. 535–544.

[16] E. Eren, L. N. Tan, and A. Alwan, "FusedF0: Improving DNN-based F0 estimation by fusion of summary-correlograms and raw waveform representations of speech signals," in *Proc. Annu. Conf. Int. Speech Commun. Assoc.*, Dublin, Ireland, 2023, pp. 4523–4527.

[17] X. Li, H. Huang, Y. Hu, L. He, J. Zhang, and Y. Wang, "YOLOPitch: A time-frequency dual-branch YOLO model for pitch estimation," in *Proc. Annu. Conf. Int. Speech Commun. Assoc.*, Kos Island, Greece, 2024, pp. 72–76.

[18] J. Engel, L. Hantrakul, C. Gu, and A. Roberts, "DDSP: Differentiable digital signal processing," in *Proc. Int. Conf. Learn. Representations*, Virtual, 2020, pp. 12010–12028.

[19] A. Polyak, L. Wolf, Y. Adi, and Y. Taigman, "Unsupervised cross-domain singing voice conversion," in *Proc. Annu. Conf. Int. Speech Commun. Assoc.*, Shanghai, China, 2020, pp. 801–805.

[20] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, Vancouver, BC, Canada, 2019, pp. 8024–8035.

[21] L. Stewart, F. R. Bach, Q. Berthet, and J. Vert, "Regression as classification: Influence of task formulation on neural network features," in *Proc. Int. Conf. Artif. Intell. Statist.*, Valencia, Spain, 2023, pp. 11563–11582.

[22] J. Farebrother et al., "Stop regressing: Training value functions via classification for scalable deep RL," in *Proc. Int. Conf. Mach. Learn.*, 2024, pp. 13049–13071.

[23] M. Müller, *Fundamentals of Music Processing–Using Python and Jupyter Notebooks*, 2nd ed. Berlin, Germany: Springer Verlag, 2021.

[24] B. R. Glasberg and B. C. J. Moore, "Derivation of auditory filter shapes from notched-noise data," *Hear. Res.*, vol. 47, no. 1, pp. 103–138, 1990.

[25] C. -L. Hsu and J. -S. R. Jang, "On the improvement of singing voice separation for monaural recordings using the MIR-1K dataset," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 18, no. 2, pp. 310–319, Feb. 2010.

[26] R. M. Bittner, B. McFee, J. Salamon, P. Li, and J. P. Bello, "Deep salience representations for F0 tracking in polyphonic music," in *Proc. Int. Soc. Music Inf. Retrieval Conf.*, Suzhou, China, 2017, pp. 63–70.

[27] C. Raffel et al., "MIR_EVAL: A transparent implementation of common MIR metrics," in *Proc. Int. Soc. Music Inf. Retrieval Conf.*, Taipei, Taiwan, 2014, pp. 367–372.

[28] S. Rosenzweig, S. Schwär, and M. Müller, "libf0: A Python library for fundamental frequency estimation," in *Proc. Demos Late Breaking News Int. Soc. Music Inf. Retrieval Conf.*, Bengaluru, India, 2022.

[29] J. Salamon, E. Gómez, D. P. W. Ellis, and G. Richard, "Melody extraction from polyphonic music signals: Approaches, applications, and challenges," *IEEE Signal Process. Mag.*, vol. 31, no. 2, pp. 118–134, Mar. 2014.

[30] J. Salamon and E. Gómez, "Melody extraction from polyphonic music signals using pitch contour characteristics," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 20, no. 6, pp. 1759–1770, Aug. 2012.

[31] V. Arora and L. Behera, "On-line melody extraction from polyphonic audio using harmonic cluster tracking," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 21, no. 3, pp. 520–530, Mar. 2013.

[32] H. Tachibana, T. Ono, N. Ono, and S. Sagayama, "Melody line estimation in homophonic music audio signals based on temporal-variability of melodic source," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, Dallas, Texas, USA, 2010, pp. 425–428.

[33] J. -L. Durrieu, G. Richard, B. David, and C. Févotte, "Source/filter model for unsupervised main melody extraction from polyphonic audio signals," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 18, no. 3, pp. 564–575, Mar. 2010.

[34] S. Kum, J. Lee, K. L. Kim, T. Kim, and J. Nam, "Pseudo-label transfer from frame-level to note-level in a teacher-student framework for singing transcription from polyphonic music," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, Virtual Singapore, 2022, pp. 796–800.
[35] K. Chen, S. Yu, C. Wang, W. Li, T. Berg-Kirkpatrick, and S. Dubnov, "TONet: Tone-octave network for singing melody extraction from polyphonic music," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, Virtual Singapore, 2022, pp. 621–625.
[36] S. Yu, X. Chen, and W. Li, "Hierarchical graph-based neural network for singing melody extraction," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, Virtual, Singapore, 2022, pp. 626–630.
[37] K. Shao, K. Chen, T. Berg-Kirkpatrick, and S. Dubnov, "Towards improving harmonic sensitivity and prediction stability for singing melody extraction," in *Proc. Int. Soc. Music Inf. Retrieval Conf.*, Milano, Italy, 2023, pp. 657–663.
[38] J. -C. Wang, W. -T. Lu, and J. Chen, "Mel-RoFormer for vocal separation and vocal melody transcription," in *Proc. Int. Soc. Music Inf. Retrieval Conf.*, San Francisco, CA, USA, 2024, pp. 454–461.
[39] F. Stöter, S. Uhlich, A. Liutkus, and Y. Mitsufuji, "Open-Unmix–A reference implementation for music source separation," *J. Open Source Softw.*, vol. 4, no. 41, 2019.
[40] R. M. Bittner, J. Salamon, M. Tierney, M. Mauch, C. Cannam, and J. P. Bello, "MedleyDB: A multitrack dataset for annotation-intensive MIR research," in *Proc. Int. Soc. Music Inf. Retrieval Conf.*, Taipei, Taiwan, 2014, pp. 155–160.
[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, San Diego, California, USA, 2015.
[42] N. C. Tamer, Y. Özer, M. Müller, and X. Serra, "TAPE: An end-to-end timbre-aware pitch estimator," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, Rhodes Island, Greece, 2023, pp. 1–5.
[43] N. C. Tamer, P. Ramoneda, and X. Serra, "Violin etudes: A comprehensive dataset for f0 estimation and performance analysis," in *Proc. Int. Soc. Music Inf. Retrieval Conf.*, Bengaluru, India, 2022, pp. 517–524.
[44] A. Flexer and D. Schnitzer, "Effects of album and artist filters in audio similarity computed for very large music databases," *Comput. Music J.*, vol. 34, no. 3, pp. 20–28, 2010.

Sebastian Strahl received the B.Sc. degree in Electrical Engineering – Electronics – Information Technology and the M.Sc. degree in Advanced Signal Processing and Communications Engineering from Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany, in 2021 and 2023, respectively. He joined the International Audio Laboratories Erlangen, Erlangen, in 2023, where he is currently working toward the Ph.D. degree under the supervision of Prof. Meinard Müller. His research interests include fundamental frequency estimation, music transcription, automatic arrangement, and differentiable algorithms.

Meinard Müller (Fellow, IEEE) received the Diploma in mathematics and the Ph.D. degree in computer science from the University of Bonn, Bonn, Germany, in 1997 and 2001, respectively. After his postdoctoral studies (2001–2003) in Japan and his habilitation (2003–2007) in multimedia retrieval in Bonn, he worked as a Senior Researcher with Saarland University, Saarbrücken, Germany, and the Max-Planck Institut für Informatik, Saarbrücken, (2007–2012). Since 2012, he has been holding a professorship for Semantic Audio Signal Processing with International Audio Laboratories Erlangen, Erlangen, a joint institute of the Friedrich-Alexander-Universität Erlangen-Nürnberg, and the Fraunhofer Institute for Integrated Circuits IIS. His research interests include music processing, music information retrieval, audio signal processing, and motion processing. He was a member of the IEEE Audio and Acoustic Signal Processing Technical Committee (2010–2015), a member of the Senior Editorial Board of the IEEE Signal Processing Magazine (2018–2022), and a member of the Board of Directors, International Society for Music Information Retrieval (2009–2021, being its president in 2020/2021). In 2020, he was elevated to IEEE Fellow for contributions to music signal processing.