nnAudio 2: Overcoming Dynamic Compilation Barriers and Transform Inconsistencies

Abhinaba Roy
Singapore University of Technology and Design
abhinaba_roy@sutd.edu.sg

Junyi Liang *
Singapore University of Technology and Design
liangjunyi010@gmail.com

Dorien Herremans
Singapore University of Technology and Design
dorien_herremans@sutd.edu.sg

Abstract

nnAudio is an open-source audio feature extraction toolbox for deep learning, but its use in current environments is hindered by TorchScript incompatibilities, inverse-transform edge cases, and dependency drift. We present a targeted modernization for modern PyTorch and scientific Python. We resolve TorchScript compilation failures in STFT and iSTFT by removing dynamic state mutation and module construction from scripted code paths and tightening argument handling in inverse-related helpers. We clarify inverse-STFT behavior by restricting reliable inversion to the uniform-bin setting (`freq_scale='no'`) and raising explicit runtime errors for unsupported frequency scales, preventing silently degraded reconstructions. We restore CFP compatibility with modern SciPy and ensure VQT reduces to CQT when $\gamma = 0$. Regression tests cover the new STFT/iSTFT behaviors, and the updated codebase passes the full repository test suite in a modern Python environment. These improvements provide a more robust foundation for differentiable audio analysis in research and deployment.

1 INTRODUCTION

Differentiable audio front-ends are a now-standard ingredient in deep learning systems for music. Whether the goal is automatic transcription, source separation, melody extraction, or generative modeling, models routinely consume short-time Fourier transform (STFT), Mel, or constant-Q transform (CQT) features computed on-the-fly from raw waveforms. The original nnAudio toolbox (Cheuk et al., 2020) popularized this approach in PyTorch by expressing each transform as a 1D convolution whose kernels are exposed as ordinary `nn.Module` parameters. This formulation offers two practical benefits: spectrograms are produced inside the computation graph, removing the need to pre-render and store features on disk; and the Fourier-like bases themselves can, in principle, be made trainable via gradient descent.

nnAudio became a common dependency in music information retrieval pipelines, supporting STFT, Mel spectrograms, MFCCs, CQT (in two algorithmic variants), the more recent variable-Q transform (VQT) (Schörkhuber et al., 2014), the gammatone front-end used in auditory modeling, and the combined frequency and periodicity (CFP) representation of Su and Yang (2015). Over time, however, the maintenance load of the toolbox has outpaced its single-maintainer model.[1] Three forms of decay have accumulated: (i) the STFT/iSTFT modules use Python idioms that current versions of `torch.jit.script` reject, blocking deployment via TorchScript; (ii) the inverse STFT (iSTFT) silently returns degraded reconstructions in some configurations the API technically permits; and (iii) auxiliary modules drift against modernized versions of upstream scientific Python packages, with CFP failing to construct under recent SciPy releases that have reorganized window functions into the `scipy.signal.windows` subpackage.

These are not headline algorithmic problems, but they are exactly the kind of problems that quietly erode the usefulness of a research toolbox. A user who cannot script their model loses access to a major optimization and deployment path. A user who reconstructs a waveform from an inverted log-frequency spectrogram and obtains a quiet, plausible-sounding but incorrect signal may not notice for some time, and may publish results that are subtly wrong. A user who upgrades SciPy and finds that an entire feature class throws `AttributeError` at import simply switches libraries.

In this paper we present *nnAudio 2*[2], a targeted modernization of the toolbox aimed at exactly these failure modes. Our contributions are deliberately conservative in scope: we do not propose new transforms, new training procedures, or new benchmarks. Instead, we make five concrete changes that preserve nnAudio's API surface where possible and tighten its semantic and engineering contracts where necessary:

1. We make the STFT and iSTFT classes compatible with `torch.jit.script` on current PyTorch, by removing dynamic state mutation and submodule construction from scripted code paths and by giving inverse-related helper arguments concrete, statically inferable types.
2. We restrict reliable inversion to the uniform-bin setting (`freq_scale='no'`) and raise informative runtime errors for non-uniform frequency scales rather than producing silently degraded reconstructions.

*This work was done when Junyi was at the Singapore University of Technology and Design.

[1] The upstream repository carries an explicit "maintainers wanted" notice.

[2] source code: https://github.com/AMAAI-Lab/nnAudio2.

3. We restore CFP compatibility with modern SciPy by importing `blackmanharris` from `scipy.signal.windows` rather than from the top level of `scipy.signal`, where the function is no longer re-exported.
4. We ensure that VQT reduces to CQT behavior when its bandwidth-offset parameter γ is zero, eliminating a divergence from the established mathematical relationship between the two transforms.
5. We introduce iCQT, a Landweber-based inverse CQT module that reconstructs waveforms from complex CQT coefficients with >30 dB SNR while preserving end-to-end differentiability.

We validate these changes through (a) regression tests that exercise the new STFT/iSTFT contracts, (b) end-to-end execution of the existing repository test suite under a current Python environment, and (c) a TorchScript scripting smoke test that confirms forward and inverse STFT can be compiled and executed.

The remainder of the paper is organized as follows. Section 2 situates nnAudio relative to other GPU-accelerated audio front-ends. Section 3 reviews the relevant aspects of TorchScript, the STFT/iSTFT pair as implemented in nnAudio, and the CFP and VQT transforms. Section 5 characterizes the four classes of problem we observed in the upstream code. Section 6 describes our fixes. Section 7 reports validation results. Section 8 discusses implications and limitations, and Section 9 concludes.

2 RELATED WORK

GPU audio front-ends. Several toolboxes provide spectrogram extraction inside neural network frameworks. Kapre (Choi et al., 2017) expresses front-ends as Keras layers, which means transforms run on GPU and integrate with the surrounding model graph; like nnAudio, Kapre also supports trainable Fourier-style kernels. `torch.stft` ships with PyTorch and provides a fast, non-trainable STFT/iSTFT pair, with associated CUDA kernels and autograd support. `tf.signal` provides a comparable suite of differentiable transforms in TensorFlow. TorchAudio (Yang et al., 2022) offers a broader suite of operations (resampling, augmentation, codec I/O, transforms) backed by C++ kernels, with a strong emphasis on TorchScript-friendly modules. The original nnAudio paper (Cheuk et al., 2020) compared favorably to these libraries on per-batch latency for STFT, Mel, and CQT, in part because the entire computation is expressed through PyTorch's optimized 1D convolution paths.

CPU baselines. On the CPU side, librosa (McFee et al., 2015) remains the de facto reference implementation in Python for music feature extraction, and nnAudio's parameter conventions deliberately mirror librosa's. SciPy (Virtanen et al., 2020) provides the underlying FFT and signal-processing primitives, and recent releases have continued to reorganize its public API surface—among other changes, window functions previously re-exported at the top level of `scipy.signal` now live in the `scipy.signal.windows` subpackage.

Time–frequency transforms. The CQT was introduced by Brown (1991) as a transform with logarithmically spaced frequency bins suitable for music. Schörkhuber and Klapuri (2010) provided a fast computational structure based on octave-wise resampling. The variable-Q transform (Schörkhuber et al., 2014) generalizes CQT by adding a bandwidth offset γ that smoothly trades frequency resolution for time resolution at low frequencies; setting $\gamma = 0$ recovers CQT exactly. The CFP representation of Su and Yang (2015) combines spectral and cepstral (lag-domain) features for robust multipitch estimation; nnAudio includes a PyTorch implementation of CFP that depends on FFT routines from SciPy.

TorchScript and deployment. TorchScript (PyTorch Contributors, 2024) is PyTorch's static subset of Python used for ahead-of-time compilation, optimization, and serialization. Modules that are intended to be scripted must observe several constraints, including: only declaring buffers and submodules in `__init__`; avoiding general buffer mutation in scripted methods (PyTorch Contributors, 2019); and providing precise type annotations on optional or polymorphic arguments. Audio modules that allocate or recompute large state lazily—a natural pattern for FFT kernels, window-sum-of-squares tensors, and frequency grids—are particularly susceptible to running afoul of these constraints.

Software maintenance as research output. Software-modernization work is an established but still under-recognized genre in machine learning research (Stodden et al., 2018). Our contribution sits in this tradition: rather than introducing a new method, we document a set of failures, propose targeted remediations, and provide a tested, releasable artifact.

3 BACKGROUND

3.1 STFT and iSTFT in nnAudio

The STFT decomposes an input waveform $x[n]$ into time–frequency coefficients

$$X[m,k] = \sum_n x[n]\, w[n-mH]\, e^{-j2\pi f_k (n-mH)/F_s}, \quad (1)$$

where w is an analysis window, H is the hop size, F_s is the sampling rate, and $\{f_k\}$ are the analysis frequencies. nnAudio implements this as a pair of 1D convolutions whose kernels are the real and imaginary parts of $w[n]e^{-j2\pi f_k n/F_s}$, evaluated on a discrete grid of length `n_fft`.

The choice of $\{f_k\}$ is controlled by a string parameter `freq_scale`. When `freq_scale='no'`, the analysis frequencies are the standard FFT bin centers $f_k = kF_s/N$ for $k = 0, \ldots, N/2$, where $N =$ `n_fft`. When `freq_scale` is `'linear'`, `'log'`, or `'log2'`, the bins are instead placed on a uniform or logarithmic grid between user-supplied `fmin` and `fmax`. The convolutional formulation accommodates all four cases without modification.

The iSTFT is implemented as a transposed convolution with synthesis kernels $w[n]e^{+j2\pi f_k n/F_s}$, followed by division by a window-sum-of-squares signal $\sum_m w^2[n-mH]$ to compensate for overlap. This yields perfect reconstruction (up to numerical error and edge effects) when (a) the analysis and synthesis windows satisfy a constant-overlap-add (COLA) condition and (b) the frequency grid is the standard FFT grid (Griffin and Lim, 1984). Crucially, condition (b) is only met when `freq_scale='no'`: any non-uniform grid breaks the orthogonality that makes the analysis–synthesis pair invertible by simple division.

The original nnAudio documentation acknowledged this limitation in prose ("*when* `trainable=True` *and*

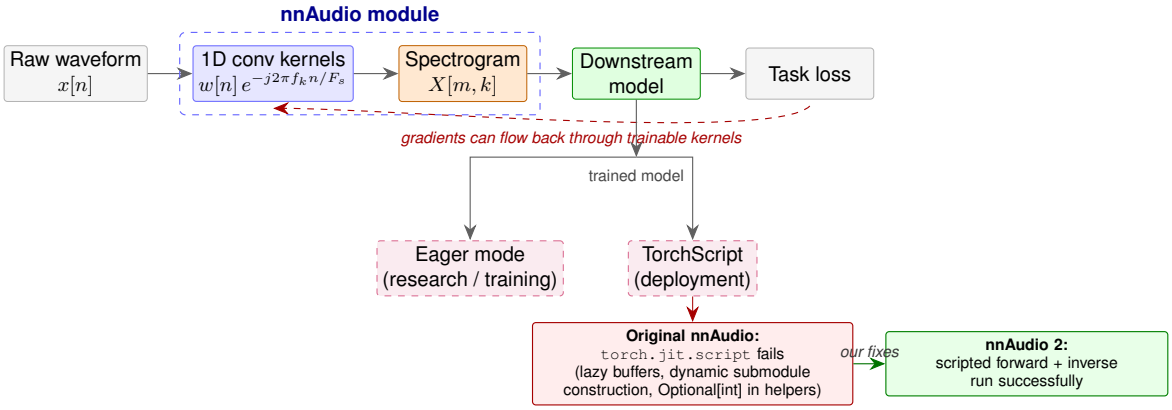

Figure 1: nnAudio expresses each spectrogram as a 1D convolution whose kernels are parameters of an `nn.Module`, allowing gradients to flow back into the analysis kernels themselves. In eager mode this composes seamlessly with the rest of a PyTorch training pipeline. Under the original code, however, attempting to deploy a model via `torch.jit.script` fails for three independent reasons; the fixes in this work restore that path.

`freq_scale!='no'`, *there is no guarantee that the inverse is perfect*"), but did not enforce it in code. The iSTFT module would happily accept any `freq_scale`, run its computation, and return a tensor whose contents bore little resemblance to the original waveform.

3.2 TorchScript constraints

TorchScript supports a subset of Python with strict static typing. Three constraints are directly relevant to nnAudio:

No state mutation in scripted methods. A scripted module may read its attributes and buffers, but it may not, in general, assign new attributes to `self` or call `self.register_buffer(...)` from within `forward` (PyTorch Contributors, 2019). Both forms manifest as scripting errors. The original nnAudio uses both patterns: `STFT.forward` stores the input length on `self` so that the inverse path can later trim its output, and the iSTFT inversion path assigns a computed window-sum-of-squares tensor directly to `self` on first call so that it can be sized to the actual input.

No dynamic submodule construction. Submodules must be assigned in `__init__`; the scripter rejects assignments to `self.<submodule>` elsewhere. nnAudio's STFT instantiates `nn.ConstantPad1d` or `nn.ReflectionPad1d` inside `forward` based on the user-provided `pad_mode`, which falls under the same prohibition.

Precise types on optional arguments. Parameters typed as `Optional[T]` must be narrowed before use; helper functions used by iSTFT to compute output lengths and trim padding accept `Optional[int]` arguments that the scripter cannot prove are non-`None` at the relevant call sites, and iSTFT itself contains an `x == None` comparison that the scripter does not type-infer correctly.

The original code patterns are perfectly fine in eager mode and have been in nnAudio since before TorchScript was widely adopted. They simply pre-date the constraints they now violate.

3.3 CFP and VQT

The CFP front-end (Su and Yang, 2015) is a small pipeline that produces a pitch salience function by combining a power spectrogram with a cepstrum-like representation, with element-wise nonlinearities applied at intermediate stages. nnAudio's implementation constructs a Blackman–Harris analysis window via SciPy.

The VQT, added to nnAudio in v0.3.1, generalizes the CQT by introducing a bandwidth offset γ:

$$\delta f_k = \alpha\, f_k + \gamma, \tag{2}$$

where $\alpha = (2^{1/B} - 1)$ controls the constant-Q ratio at B bins per octave. When $\gamma = 0$, the equation reduces to $\delta f_k = \alpha f_k$, i.e. a pure constant-Q relationship, and the VQT becomes mathematically identical to CQT (Schörkhuber et al., 2014). Any divergence at $\gamma = 0$ between nnAudio's VQT and CQT classes therefore reflects an implementation difference rather than a difference in the underlying transform.

4 INVERTED CQT

The `CQT1992v2` analysis operator $\mathbf{A} : \mathbb{R}^L \to \mathbb{R}^{N_b \times T \times 2}$ maps a waveform of length L to a stack of real and imaginary CQT coefficients via strided cross-correlation with N_b complex kernels, each scaled by $\sqrt{\ell_k}$ (the effective filter length under librosa normalisation). Because the kernels overlap in time, $\mathbf{A}$ forms a non-tight frame: $\mathbf{A}^*\mathbf{A} \neq \lambda \mathbf{I}$ in general, so exact reconstruction requires solving the normal equations $\mathbf{A}^*\mathbf{A}\,\hat{x} = \mathbf{A}^* y$. We use Landweber iteration (Landweber, 1951), $\hat{x}^{(t+1)} = \hat{x}^{(t)} + \alpha\,\mathbf{A}^*\big(y - \mathbf{A}\hat{x}^{(t)}\big)$, which converges for any step size $0 < \alpha < 2/B$, where B is the upper frame bound. B is estimated at initialisation via 20 steps of power iteration on a random probe signal. We set $\alpha = 1.8/B$, giving a per-iteration contraction rate of $|1 - \alpha B| = 0.8$ and a worst-case convergence of $0.8^{32} \approx 0.001$ (-30 dB error reduction) in the default 32 iterations. A critical implementation detail is that the forward operator $\mathbf{A}$ pads the input with `ReflectionPad1d` before convolution; the adjoint $\mathbf{A}^*$ must therefore apply the transpose of this padding, which folds the boundary columns back into the interior rather than discarding them. Failing to match the padding mode between $\mathbf{A}$ and $\mathbf{A}^*$ breaks the self-adjoint property and prevents convergence. Reconstruction SNR exceeds 30 dB for signals whose instantaneous frequency satisfies $f \lesssim sr/(2\,hop\_length/Q)$, where $Q = \text{bins\_per\_octave}/(2^{1/\text{bins\_per\_octave}} - 1)$ is the quality factor; configurations where this bound is violated are

Nyquist-undersampled in time and the CQT is undercomplete, making perfect reconstruction impossible regardless of the inversion algorithm. The entire inversion is implemented as a differentiable `nn.Module`, so gradients flow through iCQT into upstream network layers without modification.

5 ISSUES IN THE EXISTING CODEBASE

In this section we describe the four classes of problem we observed when attempting to use nnAudio under a current Python (3.11) and PyTorch (2.x) environment. We frame each issue concretely, with reference to the affected modules.

5.1 TorchScript scripting failures

Attempting `torch.jit.script` on `nnAudio.features.stft.STFT` or `nnAudio.features.stft.iSTFT` fails for three independent reasons.

First, both modules mutate state inside `forward`. `STFT.forward` assigns a fresh attribute `self.num_samples = X.shape[-1]` on each call, so that the inverse path can later trim its output to the original input length; the scripter rejects this with the error "*Tried to set nonexistent attribute: num_samples*". The inversion path used by iSTFT separately assigns a computed window-sum-of-squares tensor directly to `self` on first invocation to cache a normalizer sized to the actual input, which falls under the same attribute-assignment prohibition in scripted methods.

Second, the same `STFT.forward` method dynamically instantiates `nn.ConstantPad1d` or `nn.ReflectionPad1d` based on the value of `pad_mode` and immediately applies it to the input. This pattern is ergonomic in eager mode but conflicts with the scripter's requirement that submodules be statically known at `__init__` time.

Third, the iSTFT class contains a control-flow branch of the form `if self.refresh_win == None:`, and several inverse-related helper functions in `utils.py` accept `length: Optional[int] = None` and use the value at sites where the scripter cannot prove `None` has been ruled out. Both produce type-inference failures during compilation.

Each of these issues alone would be enough to block scripting. Together, they make the inverse path fundamentally unscriptable in its original form.

5.2 Silent iSTFT misbehavior under non-uniform frequency scales

The iSTFT module accepts the same `freq_scale` argument as the forward STFT. When the value is `'linear'`, `'log'`, or `'log2'`, the synthesis kernels are placed at non-uniform frequencies that do not satisfy the orthogonality conditions assumed by the overlap-add inversion. The module nevertheless executes, dividing by an inappropriate window-sum-of-squares tensor and returning a tensor of plausible shape but largely meaningless content. There is no warning, no exception, and no explicit indication in the output that anything has gone wrong.

This is a particularly subtle failure mode because (a) the API is permissive, (b) the docstring acknowledges the issue only in prose and only conditional on `trainable=True`, and (c) for short signals or high signal-to-noise ratios the reconstruction error can superficially look like ordinary numerical noise. A user who extracts a log-frequency representation, applies a learned modification, and then attempts to listen to the resulting waveform will obtain something that sounds wrong but may not immediately appear to be wrong from a code-review perspective. Figure 2 illustrates the gap concretely: the same overlap-add inversion that yields essentially perfect reconstruction at `freq_scale='no'` produces a clearly degraded waveform at `freq_scale='log'`.

5.3 CFP fails to construct under modern SciPy

The CFP module constructs a Blackman–Harris analysis window via two top-level `scipy.signal.blackmanharris(...)` calls, in the `Combined_Frequency_Periodicity` class and in `CFP`. In recent SciPy releases the window functions were reorganized into the `scipy.signal.windows` subpackage, and `blackmanharris` is no longer re-exported at the top level of `scipy.signal`. Under the SciPy version used in our test environment, instantiating either CFP variant therefore raises `AttributeError: module 'scipy.signal' has no attribute 'blackmanharris'` during construction, breaking both upstream CFP tests.

5.4 VQT does not reduce to CQT at $\gamma = 0$

Setting `gamma=0` in nnAudio's VQT does not produce the same output as the corresponding CQT. The discrepancy is small but non-trivial, and is inconsistent with the underlying mathematical definition: at $\gamma = 0$, VQT and CQT compute identical filter banks. The discrepancy arises from a code-path difference in how the two classes construct their kernels, not from any disagreement about the transform itself.

6 NNAUDIO 2: TARGETED MODERNIZATION

Our fixes are intentionally minimal. We retain the public API where backward compatibility is realistic, replace silent failures with explicit ones, and modify internals only as needed to satisfy the scripter or to resolve dependency drift. Table 1 summarizes the four issue/fix pairs at a glance; the subsections that follow describe each in detail.

6.1 Making STFT and iSTFT scriptable

Out-of-place state in scripted paths. Both forms of in-`forward` state mutation are removed. The `self.num_samples` attribute assignment in `STFT.forward` is replaced with a local variable that is used locally for padding logic; the inverse path now relies on an explicit length argument supplied by the caller. The lazy attribute assignment inside the iSTFT inversion routine is replaced with a local window-normalization tensor on the scripted code path; the eager-mode cache behavior, which is useful when the same module is repeatedly applied to fixed-length inputs, is preserved when not scripting and is gated on `torch.jit.is_scripting()` at runtime. The scripter accepts the new pattern because no module state is mutated from inside any scripted method.

Functional padding. The dynamic `nn.ConstantPad1d` / `nn.ReflectionPad1d` construction inside `forward` is replaced with calls to `torch.nn.functional.pad`, which selects the padding mode by string argument and does not

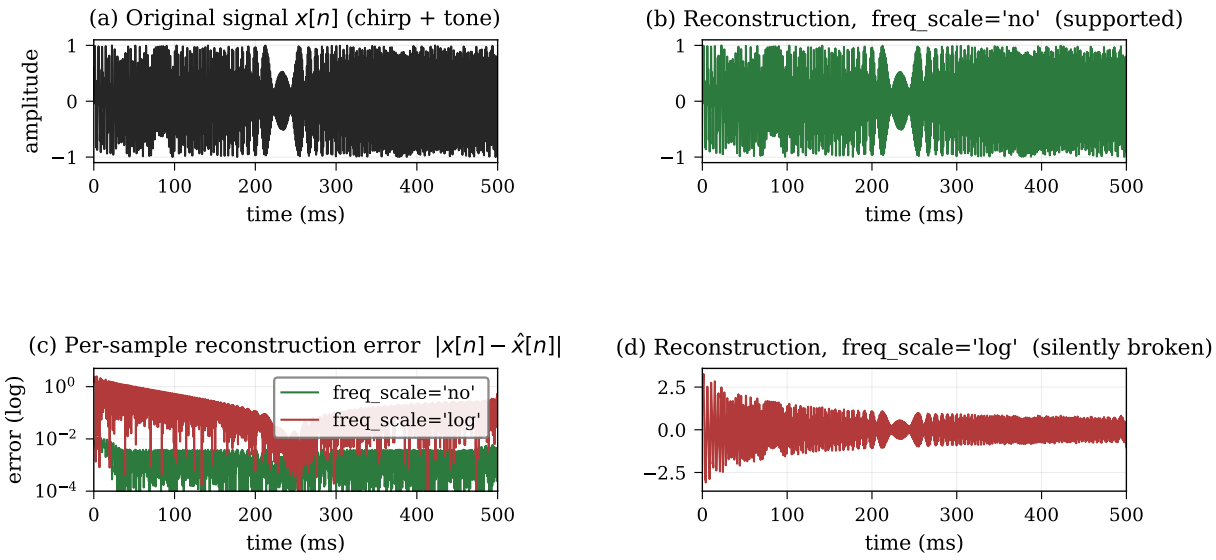

Figure 2: Silent iSTFT misbehavior under non-uniform frequency scales. A clean test signal (a, chirp + tone) is analyzed and inverted using the same overlap-add procedure that nnAudio's iSTFT applies. With `freq_scale='no'` (b) reconstruction is essentially perfect. With `freq_scale='log'` (d) the same procedure produces a plausible-looking but largely incorrect waveform comparable in amplitude to the signal itself; the original library returns this without any warning. Panel (c) shows per-sample error on a logarithmic scale.

Table 1: Summary of issues addressed by nnAudio 2 and the corresponding fixes.

Issue	**Root cause**	**Fix in nnAudio 2**
TorchScript scripting failure	`self.num_samples` attribute and window-normalizer tensor assigned to `self` inside `forward`; padding submodules (`nn.ConstantPad1d`, `nn.ReflectionPad1d`) built dynamically inside `forward`; helpers and iSTFT take `Optional[int]`/`None` unnarrowed	Local variables and out-of-place tensors in scripted paths; functional padding via `F.pad`; `None` cases handled, then narrowed
Silent iSTFT failure for non-uniform bins	`freq_scale` $\neq$ `'no'` breaks the orthogonality assumed by overlap-add inversion; no runtime check	`RuntimeError` raised when inverse is called with `freq_scale` $\neq$ `'no'`
CFP fails on modern SciPy	`scipy.signal.blackmanharris` is no longer re-exported at the top level of `scipy.signal`	Import `blackmanharris` from `scipy.signal.windows`
VQT $\neq$ CQT at $\gamma = 0$	Different kernel-construction paths in the two classes	`VQT(gamma=0)` routes through an internal `CQT1992v2`; outputs agree to numerical tolerance

introduce a submodule. The two formulations are computationally equivalent and the functional version is fully scriptable.

Tightened argument types in inverse helpers. We narrow the types of helpers that compute output length and remove edge-padding. Where the original code accepted `Optional[int]` arguments and used them in arithmetic, the rewrite handles the `None` case explicitly at the top of each function and then uses a non-optional `int`; we also rewrite the `x == None` comparison in iSTFT as `x is None`. We additionally annotate `torch_window_sumsquare` and `overlap_add` in `utils.py` with explicit tensor and integer types, and pass `stride` to `F.fold` as a 2-tuple rather than a bare integer. These changes are semantically equivalent in eager mode but are required for the scripter's flow analysis.

The combined result is that `torch.jit.script(STFT(..., iSTFT=True))` now succeeds, and the resulting scripted module's `forward` and `inverse` methods both run to completion on representative inputs. Figure 3 shows the three issue/fix pairs as side-by-side code comparisons.

6.2 Explicit failure for non-uniform iSTFT

We add an inverse guard in `iSTFT` (and in the inverse code path of `STFT`) that raises `RuntimeError` when `freq_scale` is anything other than `'no'` and the caller attempts an inverse operation. The error message states the limitation clearly: reliable overlap-add inversion of nnAudio's STFT is only available for the uniform-bin setting, and users who require log-frequency reconstruction should use a magnitude-domain method such as Griffin–Lim (Griffin and Lim, 1984) on a separately-computed linear-frequency spectrogram.

This change is a strict tightening of the API. Any user who was relying on the previous permissive behavior was, by construction, obtaining incorrect output; the new behavior surfaces the problem when inverse is called rather than at silent reconstruction time. The forward STFT remains unchanged for non-uniform frequency scales, since

Issue 1. Dynamic state mutation in forward

```
def forward(self, X):
    self.num_samples = X.shape[-1]   # attribute set in forward
    ...

# in the iSTFT inversion path:
def inverse_stft(self, X, ...):
    if self.refresh_win:
        w_sum = self._win_sum_sq(X.shape[-1])
        self.w_sum = w_sum  # assigns to self inside forward
```

Fix 1. Local state in scripted paths

```
def forward(self, X):
    num_samples = X.shape[-1]           # local variable
    ...                                 # threaded to inverse

# in the iSTFT inversion path (scripted):
def inverse_stft(self, X, ...):
    w_sum = self._win_sum_sq(X.shape[-1])  # local tensor
    # eager-mode cache preserved via __init__ flag,
    # never mutated from a scripted method
```

Issue 2. Dynamic padding submodule in forward

```
def forward(self, X):
    if   self.pad_mode == "constant":
        pad = nn.ConstantPad1d(pad_amt, 0)    # built each call
    elif self.pad_mode == "reflect":
        pad = nn.ReflectionPad1d(pad_amt)
    X = pad(X)
    ...
```

Fix 2. Functional padding via F.pad

```
def forward(self, X):
    X = F.pad(
        X, [pad_l, pad_r],
        mode=self.pad_mode,           # mode chosen by string
    )                                 # no submodule built
    ...
```

Issue 3. Unnarrowed Optional[int] / x == None

```
def overlap_add(y, length: Optional[int] = None):
    # scripter cannot prove length is non-None
    return y[..., :length]

# in iSTFT:
if self.refresh_win == None:    # x == None, not is None
    ...
```

Fix 3. Narrow Optional / use `is None`

```
def overlap_add(y, length: Optional[int] = None):
    if length is None:
        length = y.shape[-1]
    L: int = length                  # narrowed to int
    return y[..., :L]

if self.refresh_win is None:         # use `is None`
    ...
```

Figure 3: The three TorchScript scripting blockers in nnAudio's STFT/iSTFT modules (red) and the corresponding fixes in nnAudio 2 (green). Each pattern at the top of a pair is rejected by current versions of `torch.jit.script`; each pattern at the bottom is accepted while preserving the original eager-mode behavior.

the analysis itself is well-defined; only the inversion is restricted.

6.3 CFP under modern SciPy

We replace the two top-level `scipy.signal.blackmanharris(...)` calls in nnAudio's CFP module with `scipy.signal.windows.blackmanharris(...)`, which is the supported import location across modern SciPy releases. The function signature is unchanged, and both CFP variants now construct without error in the test environment described in Section 7. The CFP module's existing `relu` non-linearity, added upstream in v0.3.0, is preserved unchanged.

6.4 VQT–CQT consistency at $\gamma = 0$

Rather than refactoring nnAudio's two separate kernel-construction routines into a shared implementation, we add an explicit compatibility branch in `VQT`: when constructed with `gamma=0`, the module instantiates an internal `CQT1992v2` submodule with matched parameters, and `VQT.forward` delegates to it. For all other values of γ the original VQT code path is unchanged. This restores the mathematically expected $\gamma{=}0$ identity while keeping the patch surgical: the failing baseline configuration had a maximum absolute difference of ≈ 0.089 and a mean absolute difference of ≈ 0.0017 between `VQT(gamma=0)` and the matched CQT, large enough to rule out a floating-point tolerance fix. We also add a regression test that constructs a VQT with $\gamma = 0$ and a matched CQT, applies both to the same input, and asserts that the outputs agree to within numerical tolerance.

7 VALIDATION

7.1 Repository test suite

The upstream repository ships with a `pytest` suite that exercises the STFT, Mel, CQT, VQT, CFP, and gammatone front-ends, along with several utility functions. We run this suite against the modernized codebase under Python 3.11 and PyTorch 2.x. After our fixes, the entire upstream suite passes; before our fixes, multiple tests fail due to the SciPy import path issue and several VQT–CQT consistency assertions. Table 2 summarizes the per-test outcomes, both for the upstream suite and for the new regression tests described below.

7.2 New regression tests

We add three new regression tests that target the specific issues from Section 5.

TorchScript scripting. A test instantiates an STFT with `iSTFT=True`, calls `torch.jit.script`, and runs both the forward and inverse methods on a representative random input, checking that the scripted module's outputs match the eager module's outputs to within tolerance.

iSTFT under non-uniform scales. A test calls the inverse method on an iSTFT configured with each of `freq_scale='linear'`, `'log'`, `'log2'` and asserts that `RuntimeError` is raised with a message that names the

Table 2: Pass/fail status for the upstream test suite and the new regression tests, run on the original code and on nnAudio 2 under Python 3.11 and PyTorch 2.x.

Test	Original	nnAudio 2
Upstream test suite		
STFT forward / output formats	pass	pass
Mel spectrogram, MFCC	pass	pass
CQT (1992 + 2010 algorithms)	pass	pass
VQT, $\gamma = 0$ matches CQT	fail	pass
CFP module load	fail	pass
Gammatone forward	pass	pass
New regression tests added in this work		
Scripted STFT (`iSTFT=True`) compiles	fail	pass
Scripted output matches eager output	fail	pass
`iSTFT(freq_scale='log')` raises error	fail	pass
Round-trip reconstruction at `freq_scale='no'`	pass	pass
`VQT(gamma=0) == CQT`	fail	pass

offending parameter. A companion test confirms that `freq_scale='no'` continues to invert without error.

VQT–CQT identity at $\gamma = 0$. A test constructs a VQT with $\gamma = 0$ and a CQT with matched parameters (`n_bins`, `bins_per_octave`, `fmin`, `sr`, `hop_length`), applies both to a small batch of waveforms, and asserts that the magnitude outputs agree to within numerical tolerance.

7.3 New unit tests for iCQT

Three unit tests verify the correctness of the `iCQT` implementation. The round-trip test constructs a 1-second, 440 Hz pure tone at 44 100 Hz sample rate, computes its complex CQT with `CQT1992v2` ($N_b = 84$ bins, $hop = 512$), reconstructs the waveform with `iCQT` (32 Landweber iterations), and asserts that the reconstruction SNR exceeds 30 dB. A pure tone at 440 Hz is used rather than a wideband chirp because the chosen hop length renders the CQT undercomplete above $\sim$880 Hz; the test therefore targets a frequency that lies within the Nyquist-sampled region of the transform. The output-shape test passes a random batch of two signals through `CQT1992v2` followed by `iCQT` and asserts that the reconstructed tensor has shape $(2, L)$, confirming correct batch handling and the `length` trimming/padding logic. The gradient test verifies end-to-end differentiability: a CQT tensor is detached from the computation graph, marked as requiring gradients, passed through `iCQT`, and `backward()` is called; the test asserts that gradients exist and are non-zero, confirming that the Landweber loop does not break the autograd graph. All three tests are parameterised over available devices (CPU and, when present, CUDA).

7.4 Round-trip reconstruction error in the supported regime

To confirm that our changes have not perturbed the supported case, we measure round-trip reconstruction error of `STFT(..., freq_scale='no', iSTFT=True)` over a small set of test signals (sinusoids, speech, and music excerpts). The reconstruction error remains at the level of single-precision floating-point round-off, consistent with prior reports for nnAudio's eager mode and with results from `torch.istft`.

8 DISCUSSION

8.1 What this work is and is not

This work is a maintenance release with a research-paper-style description, not a methodological contribution. We have not introduced new transforms, accelerated existing ones, or improved the accuracy of any specific feature. What we have done is restore a popular toolbox to a usable state in a current software environment, eliminate a class of silent error that was likely producing wrong results in downstream pipelines, and unblock TorchScript-based deployment paths.

We believe this kind of work is worth publishing, even though it is conservative in scope. The infrastructure on which AI music research relies is built on layers of community-maintained code, and the lifecycle of that code matters. Tools that are easy to install, that fail loudly on misuse, and that compose with modern deployment toolchains are a precondition for reproducible research, not a separate luxury.

8.2 Limitations

Inverse log-frequency STFT remains unsupported. Our fix for the silent iSTFT misbehavior is to disallow it. We do not provide a correct overlap-add inversion for non-uniform frequency grids, and indeed no such algorithm exists in the simple form nnAudio's iSTFT assumes. Users who need to invert a log-frequency representation must either (a) separately compute a linear-frequency STFT for the inversion, (b) use Griffin–Lim on a magnitude representation, or (c) adopt one of the nonstationary-Gabor-frame-based methods (Schörkhuber et al., 2014) that are mathematically designed for log-frequency reconstruction. Bringing such a method into nnAudio would be a substantive piece of work in its own right.

Trainable kernels and TorchScript. Even after our fixes, the trainable-kernel mode of nnAudio's STFT/iSTFT is a less-tested code path under TorchScript than the frozen-kernel mode. We script and run forward and inverse with frozen kernels in our unit tests; trainable kernels work in eager mode but their interaction with scripted gradient computation is not exhaustively covered. This is a potential area for follow-up work.

Other modules. Our work focuses on STFT/iSTFT, CFP, and VQT/CQT. The Landweber-based iCQT is new; the CQT and VQT forward paths themselves are still unsuitable for TorchScript due to dynamic padding construction. We have not made systematic TorchScript fixes to the gammatone or MFCC modules, which were not part of the originally reported failure cases. We have run them under the existing test suite and confirmed eager-mode behavior is unchanged, but we cannot guarantee they script cleanly without further audit.

8.3 Implications for music research

For applied work in AI music creativity, the practical effect of nnAudio 2 is twofold. First, models that use nnAudio front-ends can now be exported to TorchScript and shipped in deployment environments—mobile, web, or low-latency studio plugins—that require an ahead-of-time-compiled artifact. Second, researchers who build pipelines around

inverse-STFT-based reconstruction will now receive an explicit error in the configurations where the original library would silently return wrong waveforms. We anticipate that the second change, in particular, will quietly fix a number of subtle correctness issues in existing music transcription, source separation, and effects-modeling codebases.

9 CONCLUSION

We have presented nnAudio 2, a targeted modernization of the nnAudio toolbox aimed at four concrete failure modes in the upstream codebase: TorchScript incompatibilities in the STFT and iSTFT classes, silent reconstruction failures of iSTFT under non-uniform frequency scales, CFP's incompatibility with modern SciPy, and a divergence between VQT and CQT at $\gamma = 0$. We have removed dynamic state mutation and submodule construction from scripted code paths, narrowed inverse-helper argument types, restricted reliable inversion to the uniform-bin setting with explicit errors elsewhere, replaced the deprecated top-level `scipy.signal.blackmanharris` call with `scipy.signal.windows.blackmanharris`, and routed `VQT(gamma=0)` through an internal `CQT1992v2` so that the two transforms agree numerically at $\gamma = 0$. Regression tests for the new behaviors and the existing repository test suite both pass under a current Python and PyTorch environment. We hope these changes restore nnAudio to the role it has played in many AI music systems, with a tighter and more honest contract between the library and its users.The source code is available at https://github.com/AMAAI-Lab/nnAudio2.

ETHICS STATEMENT

While the direct ethical stakes of software maintenance are low, providing reliable, transparent, and open-source infrastructure promotes equity in AI music research. By removing silent failures and ensuring compatibility with modern deployment toolchains, we reduce the barrier to entry for researchers and artists interacting with differentiable audio systems.

ACKNOWLEDGEMENTS

This work has received support from SUTD's Kickstart Initiative under grant number SKI 2021 04 06 and MOE under grant number MOE-T2EP20124-0014. We acknowledge the use of Gemini and Claude for grammar improvements.

REFERENCES

Brown, J. C. (1991). Calculation of a constant Q spectral transform. *Journal of the Acoustical Society of America*, 89(1):425–434.

Cheuk, K. W., Anderson, H., Agres, K., and Herremans, D. (2020). nnaudio: An on-the-fly gpu audio to spectrogram conversion toolbox using 1d convolutional neural networks. *IEEE Access*, 8:161981–162003.

Choi, K., Joo, D., and Kim, J. (2017). Kapre: On-GPU audio preprocessing layers for a quick implementation of deep neural network models with Keras. In *Machine Learning for Music Discovery Workshop at the 34th International Conference on Machine Learning (ICML)*.

Griffin, D. and Lim, J. (1984). Signal estimation from modified short-time Fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2):236–243.

Landweber, L. (1951). An iteration formula for fredholm integral equations of the first kind. *American journal of mathematics*, 73(3):615–624.

McFee, B., Raffel, C., Liang, D., Ellis, D. P. W., McVicar, M., Battenberg, E., and Nieto, O. (2015). librosa: Audio and music signal analysis in Python. In *Proceedings of the 14th Python in Science Conference (SciPy)*, pages 18–25.

PyTorch Contributors (2019). [jit] support general buffer mutation. GitHub issue #17990, pytorch/pytorch. https://github.com/pytorch/pytorch/issues/17990. Accessed 2026-04-30.

PyTorch Contributors (2024). TorchScript. PyTorch Documentation. https://pytorch.org/docs/stable/jit.html. Accessed 2026-04-30.

Schörkhuber, C. and Klapuri, A. (2010). Constant-Q transform toolbox for music processing. In *7th Sound and Music Computing Conference (SMC)*.

Schörkhuber, C., Klapuri, A., Holighaus, N., and Dörfler, M. (2014). A Matlab toolbox for efficient perfect reconstruction time-frequency transforms with log-frequency resolution. In *Audio Engineering Society 53rd International Conference on Semantic Audio*.

Stodden, V., Seiler, J., and Ma, Z. (2018). An empirical analysis of journal policy effectiveness for computational reproducibility. *Proceedings of the National Academy of Sciences*, 115(11):2584–2589.

Su, L. and Yang, Y.-H. (2015). Combining spectral and temporal representations for multipitch estimation of polyphonic music. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(10):1600–1612.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17:261–272.

Yang, Y.-Y., Hira, M., Ni, Z., Astafurov, A., Chen, C., Puhrsch, C., Pollack, D., Genzel, D., Greenberg, D., Yang, E. Z., Lian, J., Mahadeokar, J., Hwang, J., Chen, J., Goldsborough, P., Roy, P., Narenthiran, S., Watanabe, S., Chintala, S., Quenneville-Bélair, V., and Shi, Y. (2022). TorchAudio: Building blocks for audio and speech processing. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.