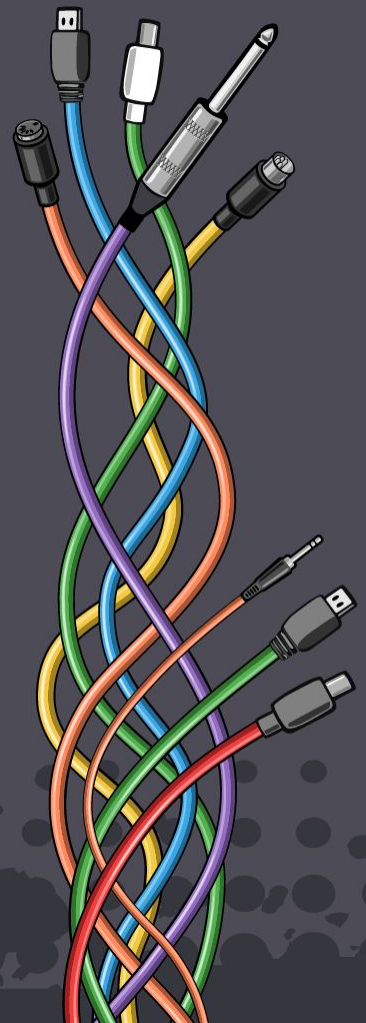
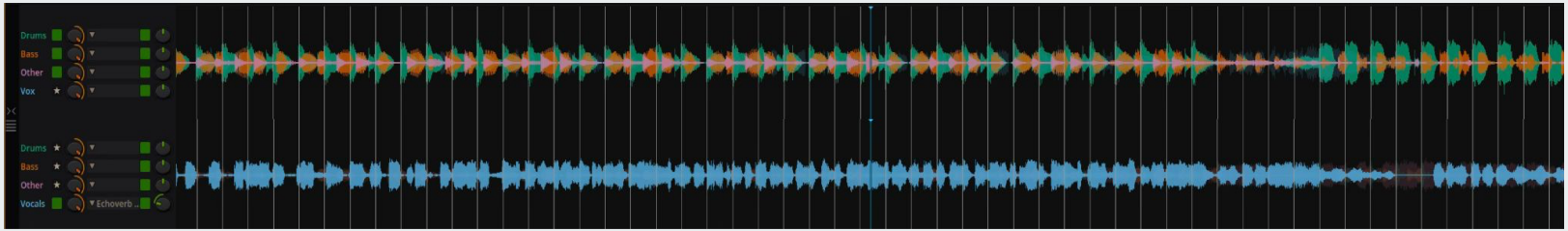


CONVERTING SOURCE SEPARATION MODELS TO ONNX FOR REAL TIME USAGE IN DJ SOFTWARE

ANMOL MISHRA

ADC25 Bristol

Converting Source Separation Models to ONNX for Real-Time Usage in DJ Software



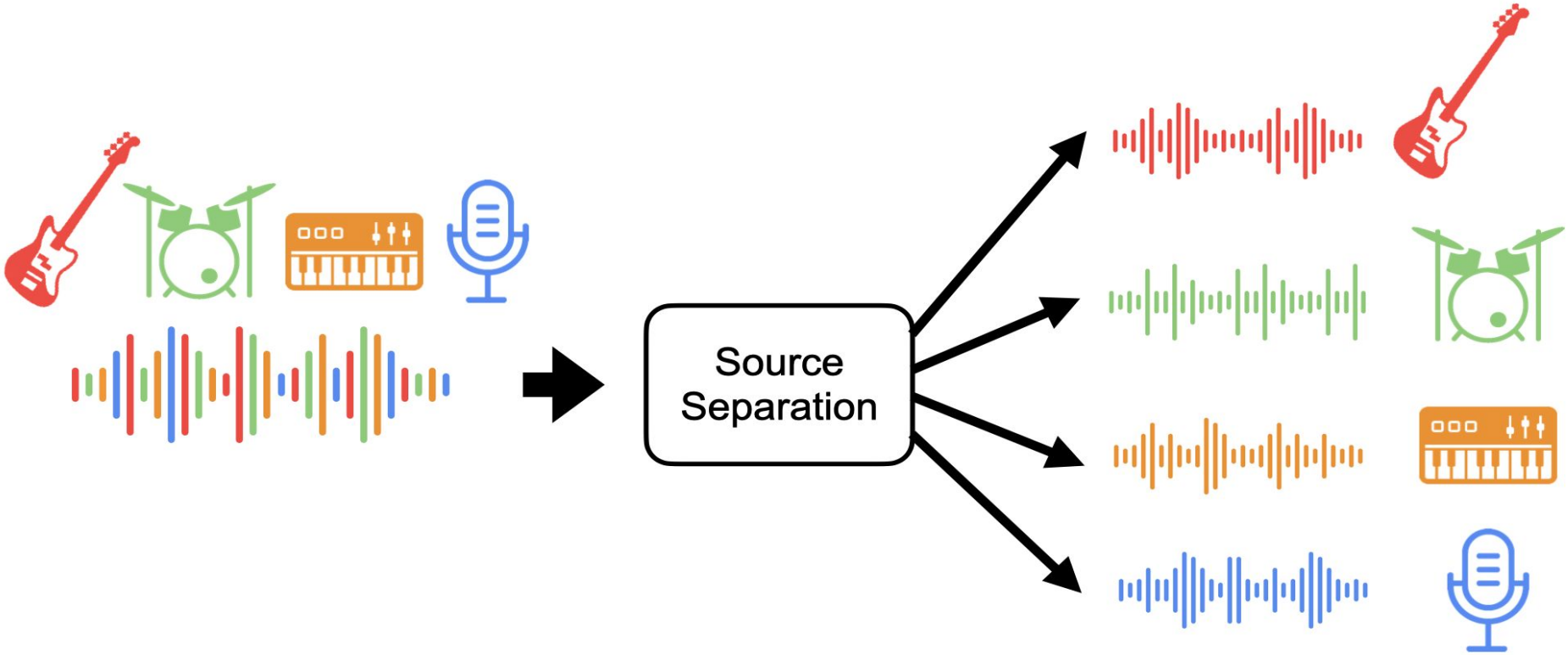
Anmol Mishra

Google Summer of Code

MIXXX

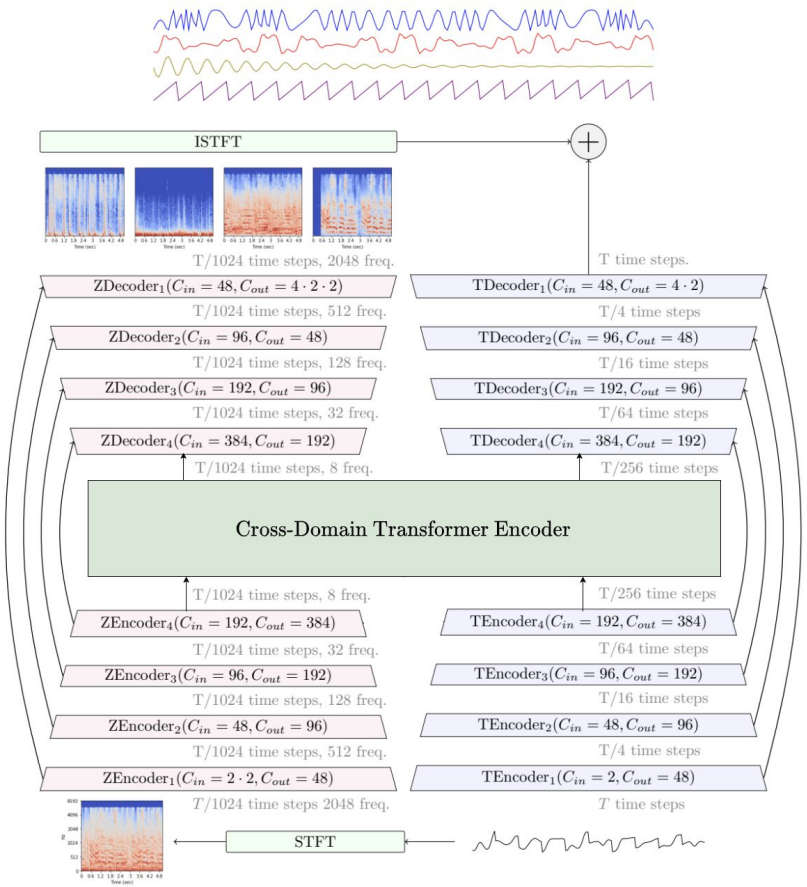
upf. Music Technology Group

From Studio Stems to Real-Time Stems



AI Models like Demucs can separate stems even when originals aren't available

Demucs: Hybrid Transformer for Music Separation



SPECTROGRAM BRANCH

Waveform ->
STFT -> (Problem)
Features ->
ISTFT -> (Problem)
Separated Stems

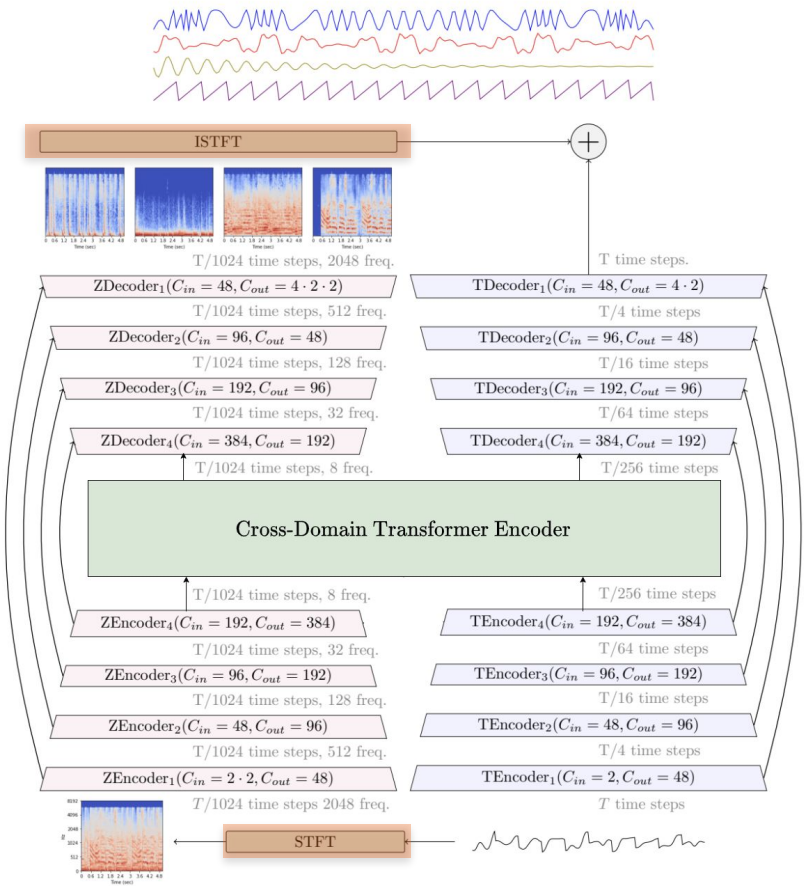
AUDIO BRANCH

Waveform ->
Features ->
Separated Stems

From Research to Real-Time: What Breaks?

	PyTorch (Research)	C++ Audio Plugin (Production)
Runtime	Python	Native C++
Model format	.pt	.onnx
Complex Tensors	✅	❌
Real-time	❌	✅

Unsupported Ops: Complex Tensors & FFTs

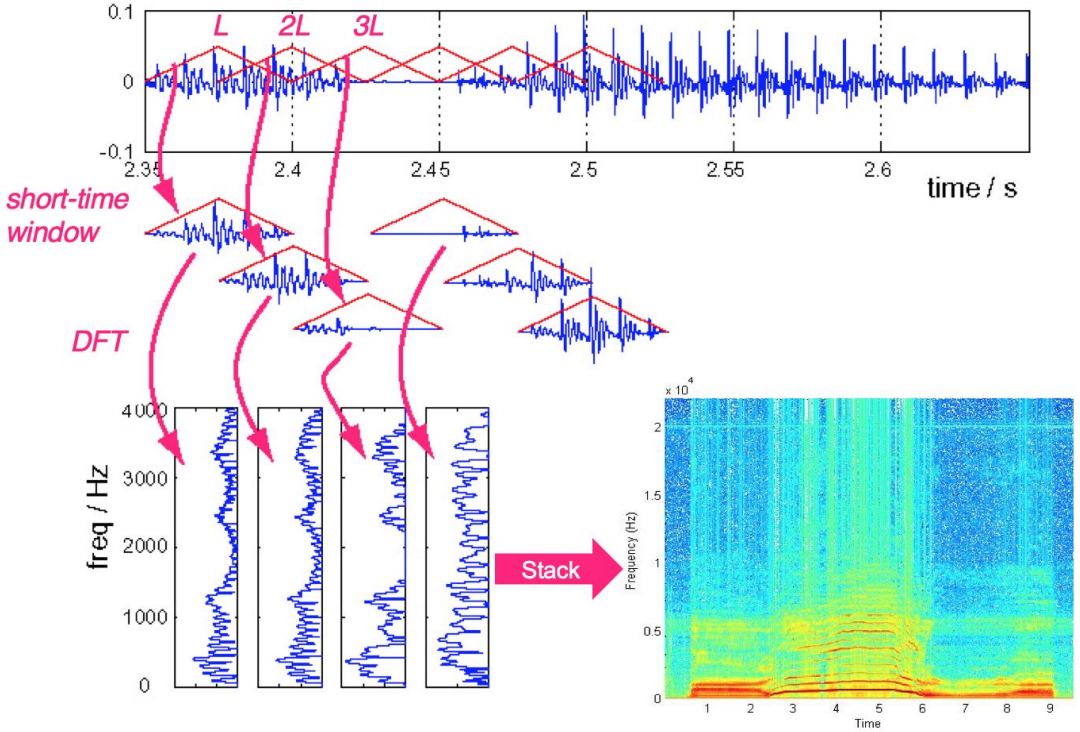


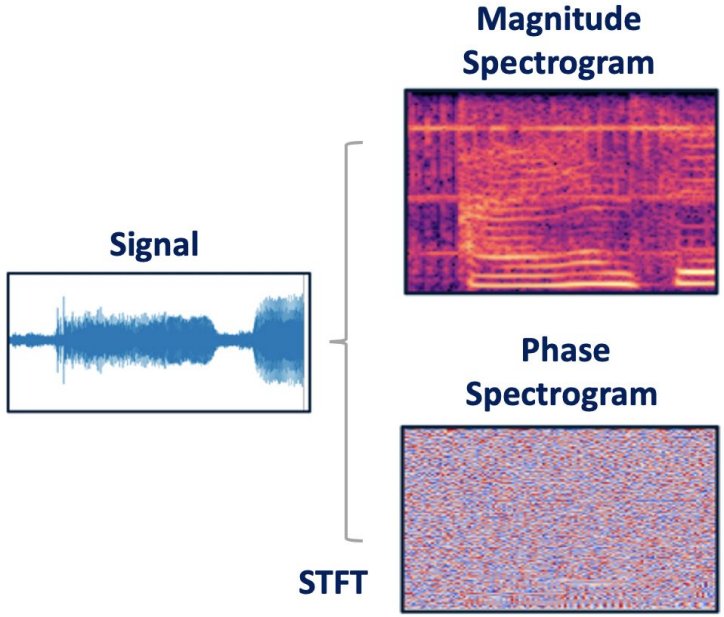

```python
# Export the core model to ONNX
try:
    torch.onnx.export(
        core_model,
        dummy_input,
        onnx_file_path,
        export_params=True,
        opset_version=17,
        do_constant_folding=True,
        input_names=['input'],
        output_names=['output'],
    )
    print(f"Model successfully converted to ONNX format at {onnx_file_path}")
except Exception as e:
    print("Error during ONNX export:", e)

```

PROBLEMS 1 OUTPUT TERMINAL ··· zsh - mixxx-demucs

```
Error during ONNX export: STFT does not currently support complex types  [Caused by th
e value '636 defined in (%636 : Float(*, *, strides=[351232, 1], requires_grad=0, devi
ce=cpu) = onnx::Reshape[allowzero=0](%626, %635), scope: demucs.htdemucs.HTDemucs:: #
```

Understanding the Short Time Fourier Transform





STFT shows how frequency content evolves with time - both in magnitude and phase

Replacing Complex Tensors with Real Tensors

```python
def spectro(x, n_fft=512, hop_length=None, pad=0):
    *other, length = x.shape
    x = x.reshape(-1, length)
    z = th.stft(x,
                n_fft * (1 + pad),
                hop_length or n_fft // 4,
                window=th.hann_window(n_fft).to(x),
                win_length=n_fft,
                normalized=True,
                center=True,
                return_complex=True,
                pad_mode='reflect')
    _, freqs, frame = z.shape
    return z.view(*other, freqs, frame)
```

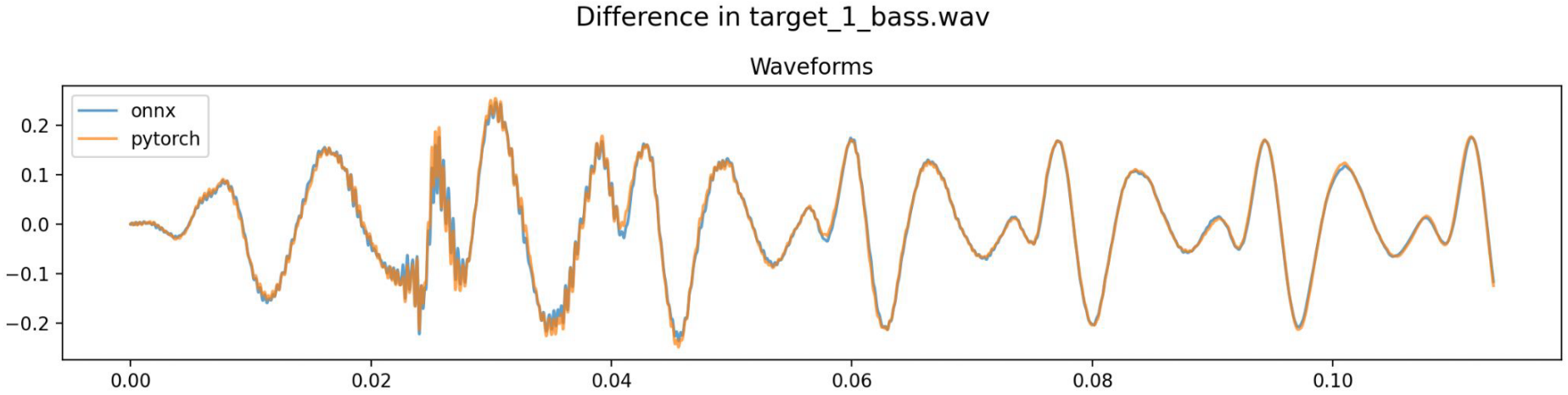
```python
def spectro(x, n_fft=512, hop_length=None, pad=0, onnx_exportable=False):
    *other, length = x.shape
    x = x.reshape(-1, length)
    is_mps_xpu = x.device.type in ["mps", "xpu"]
    if is_mps_xpu:
        x = x.cpu()

    if onnx_exportable:
        z = demucs_stft(
            x.view(-1, 1, length)
        )  # z will return 1 more dimension - z.size(-1) will be 2
        _, freqs, frame, dim = z.shape
        assert dim == 2, "STFT should return complex numbers"
        return z.view(*other, freqs, frame, dim)
```

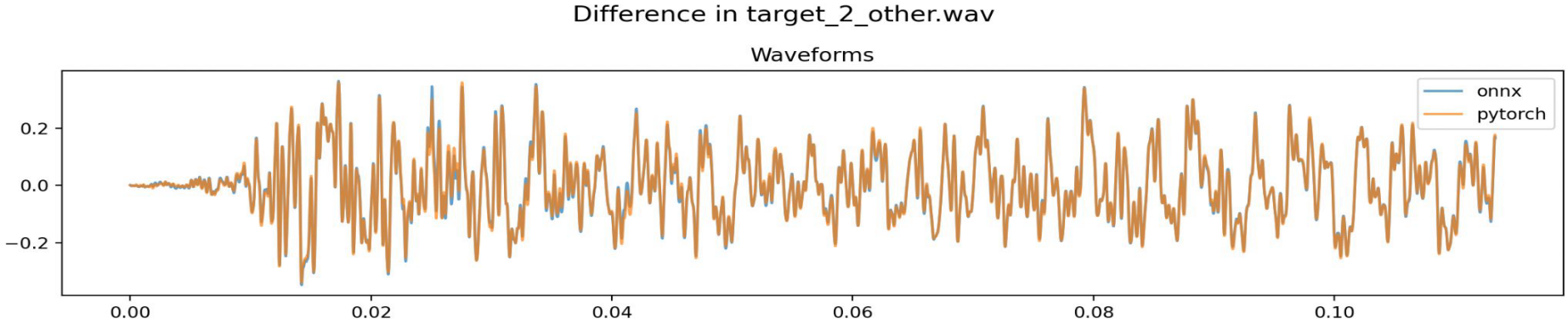
Replacing Complex Tensors with Real Tensors

```python
def ispectro(z, hop_length=None, length=None, pad=0):
    *other, freqs, frames = z.shape
    n_fft = 2 * freqs - 2
    z = z.view(-1, freqs, frames)
    win_length = n_fft // (1 + pad)
    x = th.istft(z,
                 n_fft,
                 hop_length,
                 window=th.hann_window(win_length).to(z.real),
                 win_length=win_length,
                 normalized=True,
                 length=length,
                 center=True)
    _, length = x.shape
    return x.view(*other, length)
```

```python
def ispectro(z, hop_length=None, length=None, pad=0, onnx_exportable=False):
    if onnx_exportable:
        # B, S, -1, Fr, T  (complex)      ------>     # B, S, -1, Fr, T, 2 shape
        *other, freqs, frames, dim = z.shape  # dim is 2
        assert dim == 2, "iSTFT should receive complex numbers"
        n_fft = 2 * freqs - 2
        z = z.view(-1, freqs, frames, dim)
        win_length = n_fft // (1 + pad)
        is_mps_xpu = z.device.type in ["mps", "xpu"]
        if is_mps_xpu:
            z = z.cpu()
        x = demucs_istft(z[..., 0], z[..., 1], length=length)
        _, length = x.shape
        return x.view(*other, length)
```

Verifying Equivalence

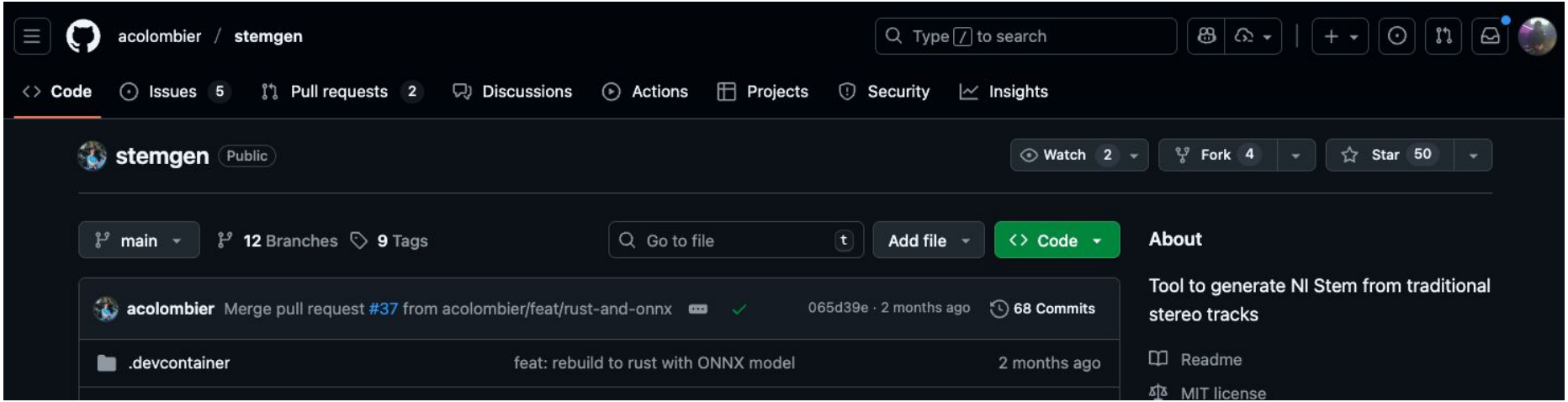


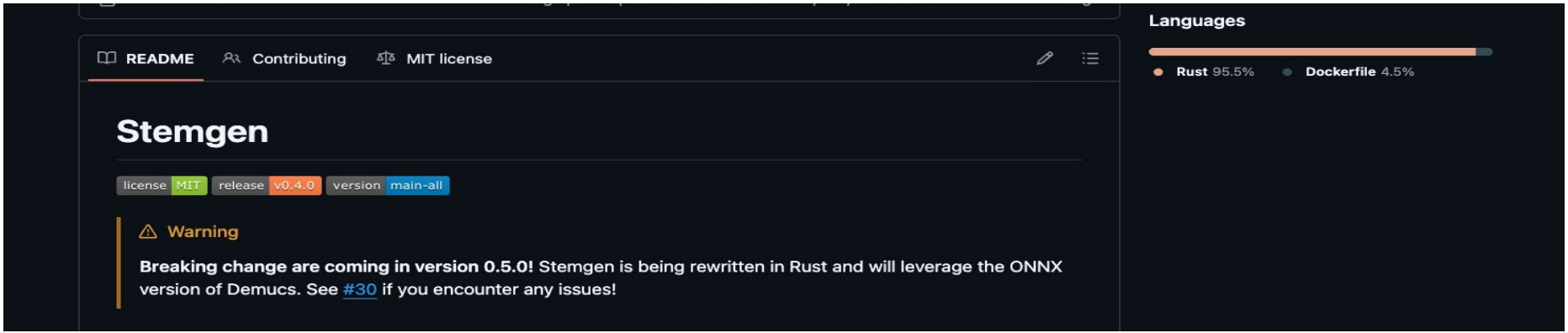


Testing Numerical Fidelity

Stem	PyTorch (in dB)	ONNX with C++ (in dB)
drums	9.46	9.47
bass	7.76	7.77
other	4.69	4.65
vocals	7.84	7.83
Overall	**7.44**	**7.43**

Bringing Real-Time Stems to DJs





Takeaways: Porting ML Models to Audio Apps

1. Don’t trust torch.onnx.export() blindly. Inspect each op.
2. Replace complex ops with real-valued equivalents.
3. Test numerically, not visually.
4. Benchmark end-to-end performance.
5. Think signal processing + software engineering, not just machine learning.

Getting Involved