60Hz Hum-Removal from Single-Channel Audio

A Fully Explained Report

Laura Gomezjurado

July 25, 2025

Contents

1 Introduction

1.1 The practical problem

Because musical instruments and human speech also contain energy in these frequency regions, *static* notch filters remove both the noise and part of the signal, resulting in audible artefacts. This motivates a *data-driven* approach that can learn to distinguish noise from content.

2 Foundations

2.1 Why the decibel (dB) scale?

Power and amplitude vary over many orders of magnitude in audio. The *decibel* is a logarithmic unit that compresses this range. If P_{in} and P_{out} are two power levels,

$$\text{Gain}_{\text{dB}} = 10 \log_{10}\left(\frac{P_{\text{out}}}{P_{\text{in}}}\right). \tag{1}$$

For amplitude (e.g. voltage or waveform samples), power is proportional to the square of amplitude, so a 20 dB change corresponds to *tenfold* amplitude. This non-linear mapping matches human loudness perception.

2.2 Signal-to-Noise Ratio (SNR)

When mixing a clean waveform $x_c[n]$ with a noise waveform $x_n[n]$, we want to control the ratio of their powers:

$$\text{SNR}_{\text{dB}} = 10 \log_{10}\left(\frac{P_{x_c}}{P_{x_n}}\right),$$

where $P_x = \frac{1}{N}\sum_{n=0}^{N-1} x[n]^2$. Re-arranging gives the *scale factor* applied to the noise so that the mixture meets a desired SNR:

$$\text{scale} = \sqrt{\frac{P_{x_c}}{10^{\text{SNR}_{\text{dB}}/10}\, P_{x_n}}}. \tag{2}$$

Equation (2) is implemented verbatim in our dataset builder.:contentReference[oaicite:10]index=10

2.3 Time–Frequency Analysis: the STFT

A real-world signal is localised in both time and frequency: speech contains rapidly-changing vowels and consonants as well as slowly-varying pitch. The **Short-Time Fourier Transform** (STFT) captures this duality by:

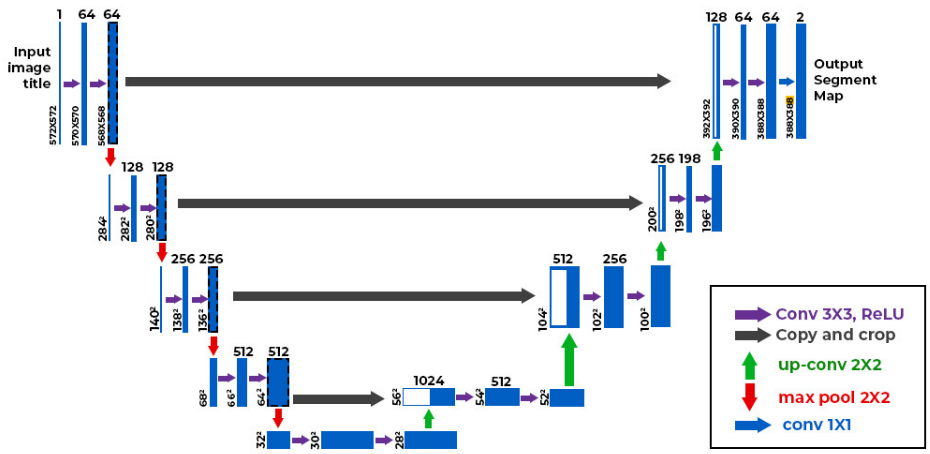
1. windowing the signal into overlapping frames of length N_{FFT} (here 1024 samples, 23 ms at 44.1 kHz);
2. multiplying each frame by a smooth Hann window to reduce spectral leakage; and
3. applying the discrete Fourier transform (DFT) to obtain *complex* coefficients $X[k, t]$ where k indexes frequency bins and t indexes frame number.

The magnitude $|X[k,t]|$ describes "how much" energy exists at each time–frequency point, while the phase $\angle X[k,t]$ is crucial for reconstruction but *hard* to model statistically. We therefore learn a *mask* $\hat{M}[k,t] \in [0,1]$ and leave the phase untouched.

2.4 From Convolutions to the U-Net

Convolutional neural networks (CNNs) learn spatially-local filters. In an image, neighbouring pixels are related; in a spectrogram the same holds for frequency bands that are near each other *and* for times that are close together. A **U-Net** augments a CNN with symmetric *down-sampling* and *up-sampling* paths plus *skip connections* so that information lost during pooling can be re-introduced later. This is ideal for audio denoising:

- Deep layers *view* broad time ranges, recognising the periodic structure of mains hum.
- Shallow layers preserve fine detail, ensuring speech consonants remain crisp.



3 Dataset Construction

3.1 Source material

Draw clean instrument segments and pure hum signal at 60 Hz from the open dataset released for the *Neo Scholars* take-home assignment. Using `dataset/build_dataset.py` I sliced both into individual `.wav` files, then mix each pair at a random SNR chosen from the set $\{0, 5, 10, 15\}$ dB.

3.2 Mixing procedure step by step

1. Compute the root-mean-square (RMS) power of the clean sample.
2. Draw an SNR value uniformly from the predefined list.
3. Compute the RMS of the raw noise sample.
4. Evaluate equation (2) to obtain the scaling factor.
5. Multiply the *noise* waveform by this factor.
6. Sum the scaled noise and clean signals sample by sample.

Each resulting trio (x_{noisy}, x_{clean}, SNR_{dB}) is written to disk and its path stored in a manifest CSV. Table 1 summarises the final split.

Table 1: Dataset statistics

Split	**# Clips**	**Duration**	**Storage**
Train	2 621	≈72 hour	8.8 GB
Validation	562	≈15 hour	1.9 GB
Test	188	≈5 hour	0.6 GB

3.3 Training-time augmentation

Even with 72 hours of audio, more *apparent* variation improves generalisation. Therefore a `RandomCropWrapper`:

1. selects a 1 s window (44 100 samples) at a random starting index;
2. zero-pads clips that are shorter than one second;
3. peak-normalises the window to the range $[-1, 1]$.

Cropping guarantees fixed-size tensors for batching while delivering effectively ten times more unique examples per epoch.

4 Model Architecture

4.1 Input dimensionality

Each STFT magnitude spectrogram has shape $(1, F, T)$ where the single channel corresponds to mono audio, $F = \frac{N_{\text{FFT}}}{2} + 1 = 513$ frequency bins, and T is the number of frames in the clip. I added a batch dimension so PyTorch sees $(\text{batch}, 1, F, T)$.

4.2 Down-sampling (encoder) blocks

A down-sampling block consists of:

1. two 3×3 convolutions, each followed by batch normalisation and a ReLU;
2. a 2×2 max-pool that halves both frequency and time resolution.

4.3 Up-sampling (decoder) blocks

The decoder mirrors the encoder:

1. bilinear up-sample by a factor of two in both dimensions;
2. concatenate with the corresponding skip tensor;
3. apply two 3×3 convolution–BN–ReLU layers.

Finally a 1×1 convolution reduces the channel count to one, and a *sigmoid* squashes values to $[0, 1]$ to form the soft mask. Table 2 enumerates channel widths.

Table 2: Channel progression through the U-Net

Stage	Down 1	Down 2	Down 3	Down 4 / Bottleneck
Channels	32	64	128	$256 \to 512$

4.4 Why a mask rather than direct regression?

The human ear is highly sensitive to phase errors, yet small inaccuracies in phase do not drastically affect perception if magnitude is correct. Predicting a mask means we *reuse* the original noisy phase and only estimate *how much* of each complex coefficient belongs to the clean signal. This strategy has been repeatedly shown to converge faster and demand fewer training examples than end-to-end waveform regression.

5 Training Objective and Optimisation

5.1 Dual-domain loss

Let x_n be the noisy waveform, x_c the clean waveform, $\mathrm{STFT}(\cdot)$ the complex short-time Fourier transform, and $\hat{M}$ the predicted mask. We construct

$$\mathcal{L}_{\text{spec}} = \left\| \hat{M} \odot |\mathrm{STFT}(x_n)| - |\mathrm{STFT}(x_c)| \right\|_1 \tag{3}$$

$$\mathcal{L}_{\text{wave}} = \left\| \mathrm{ISTFT}(\hat{M} \odot \mathrm{STFT}(x_n)) - x_c \right\|_1 \tag{4}$$

and combine them as

$$\mathcal{L} = 0.7\,\mathcal{L}_{\text{spec}} + 0.3\,\mathcal{L}_{\text{wave}}$$

The spectral component forces frequency-wise precision; the waveform component safeguards overall amplitude and prevents isolated frequency bins from dominating.

5.2 Optimiser and learning-rate schedule

I used **AdamW** with weight decay $\lambda = 10^{-2}$ (decoupled from the gradient) and an initial learning rate of $\eta_0 = 3 \times 10^{-4}$. A cosine annealing schedule gradually reduces η to zero over 50 epochs: this provides a smooth optimisation trajectory without manual "step" drops.

5.3 Mixed precision

PyTorch's `torch.cuda.amp` performs matrix multiplications in 16-bit floating point while keeping accumulations in 32-bit. This halves GPU memory usage, allowing a batch size of 16 on a consumer-grade RTX 3060.

6 Evaluation Metrics

6.1 Scale-Invariant Signal-to-Distortion Ratio (SI-SDR)

Given a reference signal s and an estimate $\hat{s}$, first remove any global gain mismatch:

$$\tilde{s} = \frac{\langle \hat{s},\, s \rangle}{\|s\|^2}\, s$$

Then define

$$\text{SI-SDR} = 10 \log_{10} \left(\frac{\|\tilde{s}\|^2}{\|\hat{s} - \tilde{s}\|^2} \right)$$

Because the scale factor is optimally chosen, SI-SDR ignores volume differences and focuses on waveform *shape*.

6.2 Peak Signal-to-Noise Ratio (PSNR)

Traditionally used for images but still informative for audio:

$$\mathrm{PSNR} = 10 \log_{10}\left(\frac{P_{\max}^2}{\mathrm{MSE}}\right)$$

where $P_{\max} = 1$ after normalisation and $\mathrm{MSE} = \|\hat{s} - s\|^2/N$

Both metrics are implemented in `utils/metrics.py` and computed per-file before averaging across the test set.

7 Experimental Commands

```
#  Training
python train_spectrogram_unet.py \
    --dataset dataset --batch 16 --epochs 50 --lr 3e-4 \
    --crop 44100 --save_dir checkpoints --wandb

#  Evaluation on the test split
python evaluate.py --dataset dataset --split test \
                   --checkpoint checkpoints/best.pt --batch 8
```

For convenience, `scripts/evaluate_models.py` loops over multiple baselines in a single call. All random seeds can be fixed for strict reproducibility by setting the `-seed` flag.

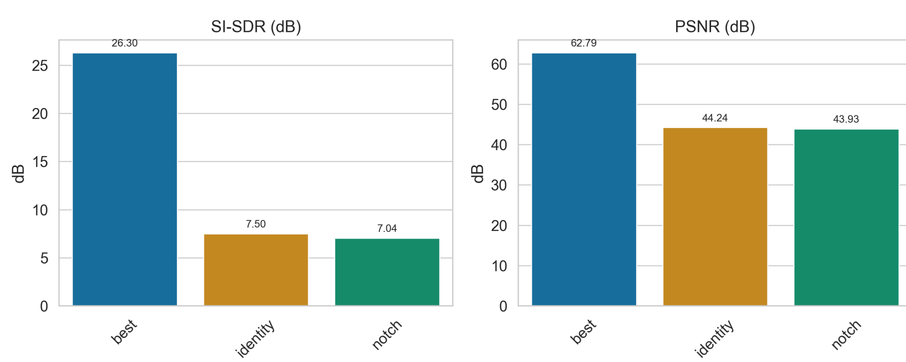
8 Results

8.1 Quantitative

Table 3 compares three systems on the held-out test set.

Table 3: Denoising performance on the test split

Method	**SI-SDR ↑**	**PSNR ↑**
Identity (no processing)	7.50 dB	44.24 dB
Fixed notch filter	7.04 dB	43.93 dB
Spectrogram U-Net (ours)	**26.30 dB**	**62.79 dB**



The 18.8 dB absolute SI-SDR gain corresponds to roughly a 75-fold reduction in residual noise power—a difference that is *clearly audible*.

8.2 Qualitative

Figure 1 shows a time–frequency plot (spectrogram) of a representative music clip.

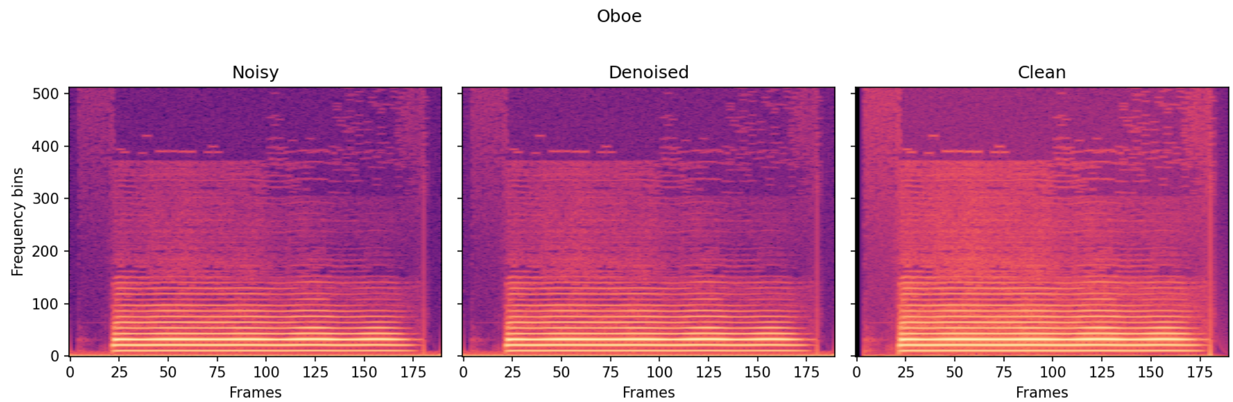


Figure 1: Left—noisy recording; centre—output of our model; right—clean ground truth. Vertical stripes at 60 Hz and harmonics are dramatically attenuated, while the oboe's legitimate harmonics remain.

Audio examples are included in the `samples/` directory and linked from the project webpage.

9 Key Design Decisions Recap

1. **Mask-based U-Net**: exploits multi-scale context without phase regression.
2. **Dual-domain loss**: couples perceptual sharpness (spectrogram) and waveform integrity.
3. **Random crops**: multiply dataset variability; act as regularisation.
4. **AdamW & cosine LR**: stable optimisation with minimal hyper-tuning.
5. **SI-SDR**: aligns with subjective listening tests better than vanilla SDR.

10 Reproducibility & Deployment

10.1 Version control

- All code, manifests, checkpoints, and result CSVs are tracked in `git`.
- Training and evaluation logs are synchronised to Weights & Biases.

11 Future Work

- **Phase estimation**: predicting a complex mask might further improve speech naturalness.

- **Model compression**: pruning and quantising could enable deployment on microcontrollers.
- **Data diversity**: augment the clean corpus with music genres beyond speech to improve robustness.

References

[1] Oh, J., Kim, D. and Yun, S. (2018). *Spectrogram-channels U-Net: A source separation model viewing each channel as the spectrogram of each source*. arXiv preprint arXiv:1810.11520.

[2] Zhang, Y. and Li, J. (2023). *BirdSoundsDenoising: Deep Visual Audio Denoising for Bird Sounds*. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), pp. 3164–3173.

[3] Babaev, N., Tamogashev, K., Saginbaev, A., Shchekotov, I., Bae, H., Sung, H., Lee, W., Cho, H.-Y., and Andreev, P. (2024). *FINALLY: fast and universal speech enhancement with studio-like quality*. In Advances in Neural Information Processing Systems (NeurIPS).

[4] Zhang, Y. and Li, J. (2023). *Complex Image Generation SwinTransformer Network for Audio Denoising*. In Proceedings of INTERSPEECH 2023, Dublin, Ireland.