Speech Time-Scale Modification With GANs: In-Depth notes with all Prerequisites

Jayin Khanna

August 2025

Abstract

While listening to spoken content, it is often desired to vary the speech rate while preserving the speaker's timbre and pitch. To date, advanced signal processing techniques are used to address this task, but it still remains a challenge to maintain a high speech quality at all time-scales. Inspired by the success of speech generation using Generative Adversarial Networks (GANs), we propose a novel unsupervised learning algorithm for time-scale modification (TSM) of speech, called ScalerGAN. The model is trained using a set of speech utterances, where no time-scales are provided. The ScalerGAN algorithm is composed of a generator that gets as input speech with the desired rate and outputs a time-adjusted speech; a discriminator that works on various spectrum scales; and a decoder that converts the time-adjusted signal back to the original rate to maintain consistency. Using an A/B test and conditional A/B test, human listeners were asked to compare ScalerGAN with other state-of-the-art TSM methods. The results showed that the speech quality of ScalerGAN outperforms all other methods.

Introduction

All the material required is given in the following:

1. Transformation of RVs Notes
2. A mathematical overview of GANs Notes
3. Audio and Speech Processing Notes
4. Code Breakdown Notes (Notion)
5. Paper Reproduced Code
6. ScalerGAN with Lab Dataset Code
7. Pipeline Notes

Objective

Time-Scale Modification (TSM) aims to adjust the speaking rate (speed up or slow down) of a speech signal without altering:

- Speaker identity (timbre, vocal characteristics).
- Pitch (perceived fundamental frequency).
- Naturalness (avoiding robotic or artifact-laden output).

Traditional signal-processing methods (e.g., Phase Vocoder, WSOLA) rely on heuristic overlap-add techniques and struggle with extreme scaling (e.g., 0.5x or 2x). **ScalerGAN** replaces these with a data-driven, GAN-based approach, improving quality and flexibility.

Key Contributions of ScalerGAN

Unsupervised Learning Framework

- No paired data required: Unlike supervised TSM, ScalerGAN does not need examples of the same utterance at multiple rates.
- Trained on single-rate speech: Uses only "natural" speech (rate = 1x) and synthesizes arbitrary rates at inference.

Three-Component Architecture

- **Generator (G):** Modifies spectrograms to the target rate.
- **Multi-Scale Discriminator (D):** Ensures realism across time-frequency resolutions.
- **Cycle-Consistency Decoder:** Reconstructs the original spectrogram to enforce consistency.

Generalization to Arbitrary Rates

Decouples time interpolation (linear resampling) from quality refinement (U-Net), enabling smooth scaling to unseen rates (e.g., 0.3x–1.8x).

Prerequisites for Beginners

To fully grasp ScalerGAN, familiarity with these concepts is helpful:

1. **Spectrograms (Time-Frequency Representations)**
   What: A 2D representation of speech, where the x-axis is time, y-axis is frequency, and color intensity represents energy (e.g., Mel-spectrogram).
   Why: Spectrograms preserve phonetic content (vowels/consonants) and are easier to manipulate than raw waveforms.
   Key Parameters:
   - FFT window size (e.g., 1024 samples).
   - Hop length (e.g., 256 samples for overlap).
   - Mel bins (e.g., 80 for dimensionality reduction).
2. **Generative Adversarial Networks (GANs)**
   - **Generator (G):** Creates synthetic data (e.g., time-scaled spectrograms).
   - **Discriminator (D):** Classifies whether data is real or generated.
   - **Adversarial Training:** G and D compete in a min-max game:
     - G tries to "fool" D with realistic outputs.
     - D learns to distinguish real vs. fake.
   - **Loss Function:** Often LSGAN (Least-Squares GAN) for stability.
3. **U-Net Architecture Section 15.6**
   Purpose: Refines coarse adjustments (e.g., interpolated spectrograms) while preserving fine details.
   Structure:

- Encoder: Downsamples spectrograms (extracts high-level features).
- Bottleneck: Processes features (e.g., 6 residual blocks).
- Decoder: Upsamples + merges with skip connections (recovers details).

Why U-Net?: Skip connections prevent loss of high-frequency content (critical for speech clarity).

4. **Cycle-Consistency Section 15**
   Concept: If you modify $A \to B$, then modifying B back $\to A'$ should match the original A.
   Application in ScalerGAN:
   - Forward pass: $G(\text{spectrogram}, r) \to$ time-scaled spectrogram.
   - Backward pass: $G(\text{time-scaled spectrogram}, 1/r) \to$ reconstructed spectrogram.
   - Loss:
   $$L_R = \|\text{original} - \text{reconstructed}\|_1.$$

2. Problem Formulation: Time-Scale Modification (TSM) of Speech

Input Representation

Speech Signal Processing Pipeline:

$$\text{Raw audio waveform} \to \text{Short-Time Fourier Transform (STFT)} \to \text{Mel-spectrogram}$$

Mel-spectrogram $\bar{x} = (x_1, \ldots, x_N)$:

- A time-frequency representation where each frame $x_i \in \mathbb{R}^d$ (e.g., $d = 80$ Mel bins).
- Captures perceptual features (e.g., pitch, formants) better than raw STFT.

Why spectrograms?

- Manipulating time/frequency directly is easier than waveforms.
- Preserves phonetic content (e.g., vowels vs. consonants).

Desired Rate r**:**

$$r < 1 : \text{Slow down (e.g., } r = 0.7 = 30\% \text{ slower}), \quad r > 1 : \text{Speed up (e.g., } r = 1.5 = 50\% \text{ faster}),$$

$$r = 1 : \text{Original rate (used for training)}.$$

Output Requirements

Time-scaled spectrogram $\bar{y} = (y_1, \ldots, y_M)$:

$$M = \lceil N \cdot r \rceil : \quad \text{Output length scales linearly with } r.$$

Key constraints:

- **Speaker Identity:** Timbre/vocal traits must match input.
- **Pitch Preservation:** Avoid "chipmunk" ($\uparrow$ pitch) or "robot" ($\downarrow$ pitch) effects.
- **Naturalness:** No artifacts (e.g., echoes, phase distortions).

Core Challenge: Unsupervised Learning

- **No paired data:** Unlike supervised TSM, there are no examples of $(\bar{x}, \bar{y})$ pairs (same utterance at different rates).
- **ScalerGAN's solution:**
  - Train only on single-rate speech ($r = 1$).
  - At inference, generalize to any $r \in [0.3, 1.8]$.

Mathematical Formulation

- **Generator (G):**

$$G : (\bar{x}, r) \to \bar{y}, \quad \text{where } \bar{y} \text{ is a spectrogram with rate } r.$$

- Must be differentiable to enable backpropagation.
- **Objective:** Minimize a loss combining:
  - **Adversarial loss:** $D(\bar{y})$ should classify $\bar{y}$ as "real."
  - **Reconstruction loss:** $G(G(\bar{x}, r), 1/r) \approx \bar{x}$ (cycle-consistency).

Why Traditional Methods Fail

- **Signal Processing (e.g., Phase Vocoder):**
  - Assumes quasi-stationarity (short-term consistency), but speech is dynamic.
  - Artifacts: Phase incoherence, "phasiness" (metallic sounds).
- **Supervised Learning:**
  - Requires parallel datasets (e.g., the same sentence spoken at 0.5x, 1x, 1.5x), which are impractical to collect.

Key Insight

Decouple time-scaling into two steps:

1. Coarse adjustment: Linear interpolation of $\bar{x}$ to length M.
2. Refinement: U-Net removes interpolation artifacts while preserving speaker/pitch.

Visualization: Problem Setup

$$\begin{array}{ll} \text{Original Speech } (r = 1): & [x_1, x_2, \ldots, x_N] \\ & \downarrow G(\bar{x}, r = 0.5) \\ \text{Slowed-down } (r = 0.5): & [y_1, y_2, \ldots, y_M] \quad (M \approx N/2) \\ & \downarrow G(\bar{y}, r = 2) \\ \text{Reconstructed}: & [x'_1, x'_2, \ldots, x'_N] \approx [x_1, \ldots, x_N] \end{array}$$

1.1 High-level architecture (Overview + Dataflow)

Prerequisites

STFT/Mel-spectrogram Section 9, high-level GAN flow.

Dataflow:

1. **Input:** waveform → Mel-spectrogram $\bar{x}$ (FFT size 1024, hop 256, 80 Mel bins).
2. **Generator** $G(\bar{x}, r)$:
   - Apply interpolation operator T_r to change time axis to target length $M \approx N \cdot r$.
   - Pass through U-Net to refine interpolated spectrogram $\bar{y}$.
3. **Discriminator** D: multi-scale sub-discriminators $\{D_s\}$ operate at different resolutions, output patch probabilities, upscaled and combined.
4. **Decoder/Cycle:** $\bar{y} = G(\bar{x}, r)$ is passed with inverse rate r^{-1} to reconstruct $\bar{x}' = G(\bar{y}, r^{-1})$; L1 loss enforces content preservation.
5. **Inference:** Convert $\bar{y}$ to waveform via HiFi-GAN vocoder.

Why this layout? T_r decouples geometric scaling from content refinement, letting U-Net focus on realism rather than raw resampling.

1.2 Generator G — Detailed (Operator + U-Net)

Prerequisites

U-Net, residual blocks, bilinear interpolation, spectrogram interpretation. Section 9

Operator T_r**:** Interpolates time axis by factor r to target M frames via bilinear interpolation — differentiable and backprop-friendly. Produces coarse resampled spectrogram; U-Net refines details.

U-Net block: Encoder–decoder with skip connections, bottleneck of 6 residual blocks. Preserves high-resolution structure while adjusting long-range dependencies.

Example: Input: 256 frames, $r = 1.3$ (slow down → ∼333 frames). $T_r(\bar{x})$ gives smoothed output; U-Net restores harmonics, cleans artifacts.

1.3 Multi-scale Discriminator D

Prerequisites

PatchGAN, receptive fields, multi-resolution ideas Section 15

Design: $D = \sum_s \alpha_s \, T_{s \times s}(D_s(T_{s^{-1} \times s^{-1}}(\bar{x})))$ where α_s are learnable weights.

- 5 sub-discriminators, scale factors 1.2^n $(n = 0..4)$.
- Each: 4 Conv layers, kernels (3,3,3,1), strides (1,2,1,1), LeakyReLU(0.2), BatchNorm, spectral norm.
- Outputs patch matrices for local realism.

Why multi-scale? Captures both global structure and fine details.

1.4 Decoder (Cycle / Reconstruction)

Prerequisites

Cycle-consistency, L1 loss, inverse operations. .

After generating $\bar{y} = G(\bar{x}, r)$, reconstruct $\bar{x}' = G(\bar{y}, r^{-1})$.

$$L_R(G) = \|G(G(\bar{x}, r), r^{-1}) - \bar{x}\|_1$$

L1 reduces mode collapse, preserves content.

1.5 Losses and Training Objective

Prerequisites

LSGAN basics, minimax GAN objective. .

Least-squares adversarial loss:

$$L_{LS}(G, D) = \mathbb{E}_{\bar{x}\sim p(\bar{x})}[(D(\bar{x}) - J)^2] + \mathbb{E}_{\bar{x}\sim p(\bar{x})}[D(G(\bar{x}, r))^2]$$

where J is all-ones matrix.

Full objective:

$$\min_G \max_D \; L_{LS}(G, D) + \lambda L_R(G), \quad \lambda = 0.1$$

1.6 Training Recipe & Hyperparameters

Prerequisites

Mini-batch training, Adam optimizer.

- Datasets: LJSpeech (train), DR-VCTK (test)
- Mel: FFT=1024, hop=256, 80 bins
- Segments: 256 frames (train), full length (test)
- G: U-Net, 6 residual blocks
- D: 5 sub-Ds, scales 1.2^n $(n = 0..4)$
- Adam: LR=5e-5, $\beta_1 = 0.5$, $\beta_2 = 0.999$
- Batch=24, Epochs=500, $\lambda = 0.1$
- r sampled from $[0.3, 1.8]$ (curriculum from 1.0)

1.7 Inference Pipeline

Prerequisites

Spectrogram extraction, vocoder usage.

1. Waveform $\rightarrow$ Mel-spectrogram
2. Choose rate r
3. Interpolate $T_r(\bar{x})$
4. $\bar{y} = G(T_r(\bar{x}), r)$
5. Vocoder $\rightarrow$ waveform

Appendix

A. Bilinear operator: Setup

We have a coarse 2×2 spectrogram patch with pixel values at integer coordinates:

$$\begin{aligned} Q_{11} &= 10 \quad \text{at } (x_1, y_1) = (0, 0), \\ Q_{21} &= 20 \quad \text{at } (x_2, y_1) = (1, 0), \\ Q_{12} &= 30 \quad \text{at } (x_1, y_2) = (0, 1), \\ Q_{22} &= 40 \quad \text{at } (x_2, y_2) = (1, 1). \end{aligned}$$

We want to upsample to a 4×4 grid with normalized coordinates $x', y' \in \{0, \frac{1}{3}, \frac{2}{3}, 1\}$.

Bilinear interpolation formula (two-step)

For general x_1, x_2, y_1, y_2 the horizontal blends are

$$\begin{aligned} R_1(x') &= Q_{11}\frac{x_2 - x'}{x_2 - x_1} + Q_{21}\frac{x' - x_1}{x_2 - x_1}, \\ R_2(x') &= Q_{12}\frac{x_2 - x'}{x_2 - x_1} + Q_{22}\frac{x' - x_1}{x_2 - x_1}. \end{aligned}$$

The vertical blend is

$$P(x', y') = R_1(x')\frac{y_2 - y'}{y_2 - y_1} + R_2(x')\frac{y' - y_1}{y_2 - y_1}.$$

Since $x_1 = 0, x_2 = 1, y_1 = 0, y_2 = 1$ these simplify to

$$\begin{aligned} R_1(x') &= Q_{11}(1 - x') + Q_{21}x', \\ R_2(x') &= Q_{12}(1 - x') + Q_{22}x', \\ P(x', y') &= R_1(x')(1 - y') + R_2(x')y'. \end{aligned}$$

Numeric computations

We compute $P(x', y')$ for each grid point. For readability we show intermediate values $R_1(x')$ and $R_2(x')$ where useful.

Row $y' = 0$

$$\begin{aligned} (x' = 0, y' = 0): &\quad R_1(0) = Q_{11} = 10, \quad P = 10. \\ (x' = \tfrac{1}{3}, y' = 0): &\quad R_1\big(\tfrac{1}{3}\big) = 10\big(1 - \tfrac{1}{3}\big) + 20\big(\tfrac{1}{3}\big) = 10 \cdot \tfrac{2}{3} + 20 \cdot \tfrac{1}{3} = \tfrac{20}{3} \approx 13.3333, \\ &\quad P = R_1(\tfrac{1}{3}) = 13.3333. \\ (x' = \tfrac{2}{3}, y' = 0): &\quad R_1\big(\tfrac{2}{3}\big) = 10\big(1 - \tfrac{2}{3}\big) + 20\big(\tfrac{2}{3}\big) = 10 \cdot \tfrac{1}{3} + 20 \cdot \tfrac{2}{3} = \tfrac{50}{3} \approx 16.6667, \\ &\quad P = 16.6667. \\ (x' = 1, y' = 0): &\quad R_1(1) = Q_{21} = 20, \quad P = 20. \end{aligned}$$

Top row: [10, 13.3333, 16.6667, 20].

Row $y' = \frac{1}{3}$

We compute $R_2(x')$ values too:

$$\begin{aligned}
R_2(0) &= Q_{12} = 30,\\
R_2\big(\tfrac{1}{3}\big) &= 30\big(1-\tfrac{1}{3}\big) + 40\big(\tfrac{1}{3}\big) = 30\cdot\tfrac{2}{3} + 40\cdot\tfrac{1}{3} = \tfrac{100}{3} \approx 33.3333,\\
R_2\big(\tfrac{2}{3}\big) &= 30\big(1-\tfrac{2}{3}\big) + 40\big(\tfrac{2}{3}\big) = 30\cdot\tfrac{1}{3} + 40\cdot\tfrac{2}{3} = \tfrac{110}{3} \approx 36.6667,\\
R_2(1) &= Q_{22} = 40.
\end{aligned}$$

Now vertical mixes with $y' = \frac{1}{3}$ (weights $1 - y' = \frac{2}{3},\ y' = \frac{1}{3}$):

$$\begin{aligned}
(x' = 0): &\quad P = 10\cdot\tfrac{2}{3} + 30\cdot\tfrac{1}{3} = \tfrac{50}{3} \approx 16.6667,\\
(x' = \tfrac{1}{3}): &\quad P = 13.3333\cdot\tfrac{2}{3} + 33.3333\cdot\tfrac{1}{3} = 20.0,\\
(x' = \tfrac{2}{3}): &\quad P = 16.6667\cdot\tfrac{2}{3} + 36.6667\cdot\tfrac{1}{3} = 23.3333,\\
(x' = 1): &\quad P = 20\cdot\tfrac{2}{3} + 40\cdot\tfrac{1}{3} = 26.6667.
\end{aligned}$$

Second row: [16.6667, 20.0, 23.3333, 26.6667].

Row $y' = \frac{2}{3}$

Mix with weights $1 - y' = \frac{1}{3},\ y' = \frac{2}{3}$:

$$\begin{aligned}
(x' = 0): &\quad P = 10\cdot\tfrac{1}{3} + 30\cdot\tfrac{2}{3} = \tfrac{70}{3} \approx 23.3333,\\
(x' = \tfrac{1}{3}): &\quad P = 13.3333\cdot\tfrac{1}{3} + 33.3333\cdot\tfrac{2}{3} = 26.6667,\\
(x' = \tfrac{2}{3}): &\quad P = 16.6667\cdot\tfrac{1}{3} + 36.6667\cdot\tfrac{2}{3} = 30.0,\\
(x' = 1): &\quad P = 20\cdot\tfrac{1}{3} + 40\cdot\tfrac{2}{3} = 33.3333.
\end{aligned}$$

Third row: [23.3333, 26.6667, 30.0, 33.3333].

Row $y' = 1$

$$\begin{aligned}
(x' = 0): &\quad P = Q_{12} = 30,\\
(x' = \tfrac{1}{3}): &\quad P = 33.3333,\\
(x' = \tfrac{2}{3}): &\quad P = 36.6667,\\
(x' = 1): &\quad P = Q_{22} = 40.
\end{aligned}$$

Bottom row: [30, 33.3333, 36.6667, 40].

Final 4×4 upsampled grid (rounded to 4 decimals)

$$P = \begin{bmatrix}
10 & 13.3333 & 16.6667 & 20\\
16.6667 & 20.0 & 23.3333 & 26.6667\\
23.3333 & 26.6667 & 30.0 & 33.3333\\
30 & 33.3333 & 36.6667 & 40
\end{bmatrix}.$$

Interpretation (for MSD)

Bilinear interpolation is a separable, linear weighted average of the four neighboring samples. It preserves coarse gradients while smoothing (blurring) higher-frequency or sharply localized detail. When upsampling a discriminator's coarse output (e.g., a low-resolution feature or gradient map) with bilinear interpolation, the result injects locally averaged, smooth corrections into the generator. This encourages long-range coherence but suppresses sharp, high-frequency adjustments.

B. Mel Spectrograms and MFCCs

B.1 Spectrograms

A **spectrogram** is a visual tool that shows how the frequency content of a signal evolves over time. In speech processing, it bridges the gap between time-domain data (waveforms) and the frequency-domain representation we need for analysis. Think of it as a *musical score* for speech:

- The **horizontal axis** is time (when sounds occur).
- The **vertical axis** is frequency (which pitches or formants are present).
- The **color intensity** represents energy or amplitude at each time–frequency point.

Why Not Just Use Waveforms?

A speech waveform shows amplitude vs. time, which is fine for loudness and duration but hides crucial frequency information. Speech sounds are **non-stationary** — vowels, consonants, and transient sounds all have different frequency content that changes within milliseconds.

Without frequency information, many linguistic and acoustic cues vanish.

How a Spectrogram is Made

The most common method is the **Short-Time Fourier Transform (STFT)**:

1. **Framing:** Break the audio into short overlapping segments (e.g., 25 ms with 10 ms overlap). Short windows assume the signal is locally stationary.
2. **Windowing:** Apply a window function (Hamming, Hann, etc.) to reduce spectral leakage.
3. **Fourier Transform:** Convert each frame from the time domain to the frequency domain.
4. **Magnitude Spectrum:** Take the magnitude (or power) of the complex result.
5. **Stacking:** Place each magnitude spectrum side-by-side in time order.
6. **Color Mapping:** Map magnitude to color or brightness to form the image.

Why Spectrograms are Essential in Speech Processing

- **Time–Frequency Trade-off:** Captures when certain frequencies occur — essential for decoding speech transitions (e.g., "b" vs. "d" sounds).
- **Feature Basis:** Many engineered features (MFCCs, filterbanks, mel-spectrograms) are computed from spectrograms.
- **Noise Analysis:** Easier to visualize noise sources, echoes, and distortions.
- **Pattern Recognition:** Different phonemes and words form distinct spectral "fingerprints" that models can learn.

- **Human Perception Parallel:** Mimics aspects of the cochlea's frequency analysis, making it perceptually relevant.

Types of Spectrograms in Speech Work

- **Linear-Frequency Spectrogram:** Frequencies spaced evenly; good for general analysis.
- **Log-Frequency Spectrogram:** More detail at low frequencies (where speech formants lie).
- **Mel-Spectrogram:** Frequencies spaced on the mel scale to match human hearing sensitivity.
- **Power vs. Amplitude Spectrogram:** Choice depends on whether energy or magnitude is of interest.

B.1 Mel-Spectrograms

Motivation & Concept

Human hearing is not linear in frequency:

- We can distinguish low-frequency differences more precisely than high-frequency ones.
- Example: Difference between 500 Hz and 600 Hz is more noticeable than between 7500 Hz and 7600 Hz.

Mel spectrograms mimic human auditory perception by:

1. Converting a signal into a time–frequency representation via the **Short-Time Fourier Transform (STFT)**.
2. Mapping the frequency axis from linear (Hz) to the **Mel scale** (approximately logarithmic).
3. Applying a **filterbank** to emphasize perceptually important frequency bands.

6.1 Digital Audio

A continuous-time audio signal $x(t)$ is sampled at a rate f_s samples/second:

$$x[n] = x\left(\frac{n}{f_s}\right), \quad n = 0, 1, \ldots, N_{\text{total}} - 1$$

Example:

$$f_s = 8000 \text{ Hz}$$

Duration = 1 s
Total samples $N_{\text{total}} = 8000$

6.2 Why Sampling Rate Matters

Nyquist-Shannon: to capture all information up to frequency $f_{\max}$, need

$$f_s \geq 2f_{\max}.$$

For speech: $f_s = 8$ kHz is enough to capture up to 4 kHz, which covers most intelligibility.

Framing

Speech/audio is quasi-stationary over short intervals (∼20–40 ms). We process it in frames to:

- Capture local frequency content.
- Allow tracking of changes over time.

Formulation:

$$T_f \text{ seconds } \rightarrow \ L = T_f \cdot f_s \text{ samples.}$$

$$T_s \text{ seconds } \rightarrow \ S = T_s \cdot f_s \text{ samples between consecutive frames.}$$

Example:

$$f_s = 8000 \text{ Hz}$$

$T_f = 25$ ms $\Rightarrow L = 0.025 \times 8000 = 200$ samples $T_s = 10$ ms $\Rightarrow S = 0.01 \times 8000 = 80$ samples
Number of frames for $N_{\text{total}} = 8000$ samples:

$$n_{\text{frames}} = 1 + \frac{N_{\text{total}} - L}{S} = 1 + \frac{8000 - 200}{80} = 1 + 97 = 98$$

Necessity: Without framing, Fourier Transform would mix frequency changes across the whole signal — bad for non-stationary signals like speech.

Windowing

Each frame is multiplied by a window function $g[n]$ (e.g., Hamming):

$$x_m[n] = x[n + mS] \cdot g[n], \quad 0 \leq n < L$$

Why?

- Minimizes spectral leakage in FFT.
- Smooths edges of frame.

Example — Hamming window:

$$g[n] = 0.54 - 0.46 \cos\left(\frac{2\pi n}{L-1}\right)$$

For $L = 5$:

$$g = [0.08,\ 0.54,\ 1.0,\ 0.54,\ 0.08]$$

Short-Time Fourier Transform (STFT)

Basic idea (in plain steps)

1. Choose a window length L_w (e.g., 20–40 ms for speech).
2. Slide the window over the signal with hop size R (often 25–75% overlap).
3. Multiply each frame by a window function $w[n]$ (Hann, Hamming, rectangular, ...).
4. Compute an N-point DFT (FFT) of each windowed frame (often $N \geq L_w$, zero-pad if needed).
5. Collect all DFTs as columns (or rows) → the STFT matrix.

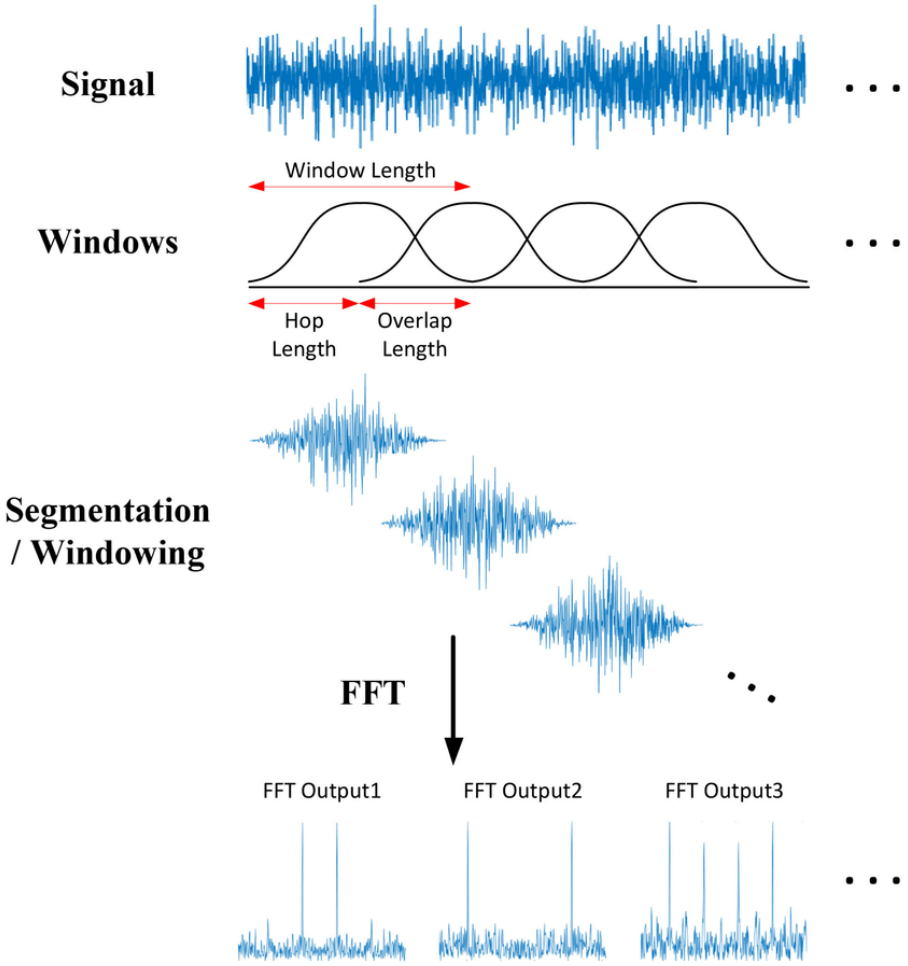

Figure 1: STFT

For each frame m:

$$X_m[k] = \sum_{n=0}^{L-1} x_m[n]\, e^{-j\frac{2\pi kn}{L}}, \quad k = 0, 1, \ldots, K-1$$

K = FFT size (often next power of $2 \geq L$, e.g., 256). Gives complex spectrum.

Fully worked numerical example (tiny toy signal)

This is short on purpose so you can follow every number.

Toy signal (8 samples):

$$x[n] = [1,\ 0,\ -1,\ 0,\ 1,\ 0,\ -1,\ 0]$$

(Think of it as a toy periodic pattern.)

Choose STFT parameters:

- Sampling rate: $f_s = 8$ Hz (toy — just for mapping bins to Hz)
- Window length: $L_w = 4$ samples
- Hop size: $R = 2$ samples (50% overlap)
- DFT length: $N = 4$
- Window: $w[n]$ = rectangular (all ones)

Number of frames:

$$\text{frames} = \frac{8-4}{2} + 1 = 3$$

So we have $m = 0, 1, 2$.

Extract frames (frame m starts at sample mR):

$$\begin{aligned} &\text{Frame 0:} \quad [1,\ 0,\ -1,\ 0] \\ &\text{Frame 1:} \quad [-1,\ 0,\ 1,\ 0] \\ &\text{Frame 2:} \quad [1,\ 0,\ -1,\ 0] \end{aligned}$$

Window multiply with rectangular $\Rightarrow$ unchanged.

DFT of a frame (example: frame 0): We compute

$$X_0[k] = \sum_{n=0}^{3} x_0[n] e^{-j2\pi kn/4}$$

with $x_0 = [1,\ 0,\ -1,\ 0]$.

$$\begin{aligned} k = 0: &\quad 1 + 0 + (-1) + 0 = 0 \\ k = 1: &\quad 1 + 0 + 1 + 0 = 2 \\ k = 2: &\quad 1 + 0 + (-1) \cdot 1 + 0 = 0 \\ k = 3: &\quad (\text{symmetry}) = 2 \end{aligned}$$

So:

$$X_0 = [0,\ 2,\ 0,\ 2]$$

DFTs for all frames:

$$\begin{aligned} X_0 &= [0,\ 2,\ 0,\ 2] \\ X_1 &= [0,\ -2,\ 0,\ -2] \\ X_2 &= [0,\ 2,\ 0,\ 2] \end{aligned}$$

STFT matrix (frames $\times$ bins):

$$\begin{bmatrix} 0 & 2 & 0 & 2 \\ 0 & -2 & 0 & -2 \\ 0 & 2 & 0 & 2 \end{bmatrix}$$

(All imaginary parts are zero.)

Magnitude:

$$|X(m,k)| = \begin{bmatrix} 0 & 2 & 0 & 2 \\ 0 & 2 & 0 & 2 \\ 0 & 2 & 0 & 2 \end{bmatrix}$$

9.1 Power Spectrum

We use magnitude squared:

$$P_m[k] = \frac{1}{L} |X_m[k]|^2$$

Example: Suppose $L = 4$, $x_m = [1, 2, 0, 0]$ (already windowed). FFT size $K = 4$:

$$X_m[0] = 1 + 2 + 0 + 0 = 3$$

$$X_m[1] = 1 - 2j + 0 - 0 = 1 - 2j$$

$$X_m[2] = 1 - 2 + 0 + 0 = -1$$

$$X_m[3] = 1 + 2j + 0 + 0 = 1 + 2j$$

Power spectrum:

$$P_m = \frac{1}{4}[9,\ 5,\ 1,\ 5]$$

Shape after processing all frames:

$$P \in \mathbb{R}^{K \times n_{\text{frames}}}$$

For our speech example: $K = 257$ (half of FFT 512 + 1), $n_{\text{frames}} = 98$.

Mel Scale

Humans perceive pitch roughly logarithmically. Mel mapping:

$$m(f) = 1125 \cdot \ln\left(1 + \frac{f}{700}\right)$$

Inverse:

$$f(m) = 700 \cdot \left(e^{m/1125} - 1\right)$$

Mel Filterbank

11.1 Building Filterbank Matrix T

Choose M Mel filters (e.g., $M = 26$). Convert $f_{\min}$ and $f_{\max}$ to Mel. Space $M + 2$ points evenly in Mel scale. Convert back to Hz. Map Hz to nearest FFT bin.

For each filter m:

- Rising slope from f_{m-1} to f_m
- Falling slope from f_m to f_{m+1}

Shape:

$$T \in \mathbb{R}^{M \times K}$$

Sparse: each row nonzero only over a small frequency range.
Example (tiny): Let $K = 5$, $M = 3$:

$$T = \begin{bmatrix} 0.0 & 0.5 & 1.0 & 0.5 & 0.0 \\ 0.0 & 0.0 & 0.5 & 1.0 & 0.5 \\ 0.0 & 0.0 & 0.0 & 0.5 & 1.0 \end{bmatrix}$$

Apply Filterbank

Matrix multiplication:

$$E = T \cdot P$$

T $(M \times K)$ $\times$ P $(K \times n_{\text{frames}}) \rightarrow E$ $(M \times n_{\text{frames}})$ $E_{m,n}$ = energy in Mel filter m at frame n.
Example: If

$$P = [4,\ 9,\ 16,\ 9,\ 4]$$

for one frame ($K = 5$), and T as above:
Filter 1 energy:

$$0 \cdot 4 + 0.5 \cdot 9 + 1.0 \cdot 16 + 0.5 \cdot 9 + 0 \cdot 4 = 4.5 + 16 + 4.5 = 25$$

Filter 2 energy:

$$0 \cdot 4 + 0 \cdot 9 + 0.5 \cdot 16 + 1.0 \cdot 9 + 0.5 \cdot 4 = 8 + 9 + 2 = 19$$

Filter 3 energy:

$$0 \cdot 4 + 0 \cdot 9 + 0 \cdot 16 + 0.5 \cdot 9 + 1.0 \cdot 4 = 4.5 + 4 = 8.5$$

So:

$$E = [25,\ 19,\ 8.5]$$

for this frame.

Log Compression

We take:

$$\log E \in \mathbb{R}^{M \times n_{\text{frames}}}$$

Mimics human loudness perception. Compresses dynamic range.

Output Dimensions for Model

For speech example: $M = 26$, $n_{\text{frames}} = 98 \Rightarrow$ Mel spectrogram shape:

$$(26 \times 98)$$

This is the input to CNN/RNN/Transformer.

Pipeline Summary

Signal: $x[n]$ — shape $(N_{\text{total}},)$ Framing $\rightarrow$ $(L,\ n_{\text{frames}})$ Windowing FFT $\rightarrow$ power spectrum P — $(K,\ n_{\text{frames}})$ Mel Filterbank T — (M, K) Matrix Multiply $E = TP$ — $(M,\ n_{\text{frames}})$ Log — final Mel spectrogram.

C. Multi-Scale Discriminator

1. **Motivation in their setup**
   Speech TSM changes both coarse-scale (global timing) and fine-scale (local harmonic and formant) structure in a spectrogram.
   A single discriminator would miss some issues:
   A fine-scale critic sees harmonic artifacts but can't detect long-range timing changes.
   A coarse-scale critic sees timing issues but might miss tiny distortions.
   So they use several sub-discriminators, each working at a different resolution of the spectrogram.
2. **The $T_{s\times s}$ operator**
   $T_{s\times s}$ = interpolation operator that upsamples along both time and frequency axes by a factor s.
   Example: $T_{2\times 2}$ on an 8×8 spectrogram $\rightarrow$ 16×16 spectrogram (bilinear or bicubic interpolation used).
   $T_{s^{-1}\times s^{-1}}$ = downsampling by factor s along both axes (reduce resolution).
3. **How input is prepared for each sub-discriminator**
   For each scale s:
   Take the original spectrogram $\bar{x}$ (real or generated).
   Downscale it using $T_{s^{-1}\times s^{-1}}$ $\rightarrow$ this shrinks it so each sub-discriminator sees a different resolution.
   Fine-scale discriminator: $s = 1 \rightarrow$ sees original resolution.
   Coarse-scale discriminator: $s > 1 \rightarrow$ sees smaller spectrogram (blurred global structure).
4. **How each sub-discriminator works**
   Architecture: Each D_s is a 4-layer CNN.
   Output: Not a single real/fake score, but a matrix of probabilities — each entry corresponds to a patch in the input spectrogram.
   This is a PatchGAN idea: instead of one scalar, each receptive field patch gets its own real/fake probability.
   Upscaling the output: The probability matrix from each scale is upsampled with $T_{s\times s}$ so that all outputs have the same spatial size (match the original spectrogram resolution).
5. **Combining multiple scales**
   The final discriminator output is a weighted sum over all scales:

$$D(\bar{x}) = \sum_s \alpha_s \, T_{s\times s} \, D_s \left(T_{s^{-1}\times s^{-1}}(\bar{x})\right)$$

   α_s are learnable weights — the network learns how much each scale matters.
   Now $D(\bar{x})$ is a matrix of the same size as the original spectrogram, containing "realness" scores per patch.

6. **Why this is important**
   This lets the discriminator judge realism at multiple resolutions while producing a single output map.
   The generator is trained to make all scales' critics happy at once, meaning:
   Fine details look real.
   Coarse timing/form structure also looks real.
7. **Loss function (LS-GAN form)**
   They use Least Squares GAN loss, but extended to matrix outputs (PatchGAN style):

$$L_{LS}(G, D) = \mathbb{E}_{\bar{x}\sim p_{\text{data}}}\left[(D(\bar{x}) - J)^2\right] + \mathbb{E}_{\bar{x}\sim p_{\text{data}}}\left[D(G(\bar{x}, r))^2\right]$$

   where: J = all-ones matrix (same size as $D(\bar{x})$).
   First term: real spectrograms should produce outputs close to 1 everywhere.
   Second term: generated spectrograms should produce outputs close to 0 everywhere.

   For the generator:

$$L_G = \mathbb{E}_{\bar{x}\sim p_{\text{data}}}\left[(D(G(\bar{x}, r)) - J)^2\right]$$

   This pushes the generator to produce spectrograms whose multi-scale PatchGAN outputs are all close to 1.
8. **Relation to wavelet analysis**
   Downscaling $T_{s^{-1}\times s^{-1}} \approx$ keeping low-frequency (coarse) components, discarding high-frequency details.
   Multiple scales $\approx$ multiple wavelet subbands:
   fine scale $\approx$ high-frequency subband (harmonics, transients)
   coarse scale $\approx$ low-frequency subband (formants, prosody)
   By having discriminators at all scales, they're essentially forcing the generator to match wavelet coefficients across scales.
9. **Fully computed toy example**
   Say original spectrogram $\bar{x}$ is 8×8, and we have two scales:
   $s = 1$ (full resolution)
   $s = 2$ (coarse)
   Step-by-step for real data:
   D_1 gets $\bar{x}$ $(8 \times 8) \rightarrow$ outputs prob matrix P_1 of size 4×4.
   D_2 gets $T_{0.5\times 0.5}(\bar{x})$ $(4 \times 4) \rightarrow$ outputs P_2 size 2×2.
   Upscale:
   $T_{1\times 1}(P_1) = P_1$ (4×4)
   $T_{2\times 2}(P_2) = 4 \times 4$.
   Weighted sum:

$$D(\bar{x}) = \alpha_1 P_1 + \alpha_2 T_{2\times 2}(P_2) \quad (4 \times 4)$$

   Compare $D(\bar{x})$ to all-ones 4×4 matrix $\rightarrow$ compute squared error $\rightarrow$ discriminator loss term for real data.
   For fake data $G(\bar{x}, r)$, same steps but target = all-zeros matrix.

D. Cycle Consistency in ScalerGAN

15.1 1. Problem They Are Addressing

In GAN-based time-scale modification (TSM), the generator G transforms a spectrogram $\bar{x}$ into a new spectrogram $\bar{y} = G(\bar{x}, r)$, where r is the time-scaling factor (e.g., $r > 1$ = stretch, $r < 1$ = compress).

Issue: Without additional constraints, the generator could:

- Learn trivial mappings (e.g., ignoring the input content and producing a generic "good-looking" spectrogram) — a kind of mode collapse.
- Change harmonic structure/formants in unintended ways.
- Remove or add random artifacts.

To fix this, they add a cycle-consistency constraint.

15.2 2. Cycle Consistency in ScalerGAN

Cycle-consistency means:

> If you transform a signal forward and then transform it back, you should recover the original.

Here:

1. Start with real spectrogram $\bar{x}$.
2. Apply time-scaling with factor r: $\bar{y} = G(\bar{x}, r)$.
3. Apply time-scaling with the inverse factor r^{-1} to $\bar{y}$: $\bar{x}' = G(\bar{y}, r^{-1})$.
4. If G is good and doesn't destroy content, then $\bar{x}' \approx \bar{x}$.

15.3 3. L1 Distance as the Cycle Loss

They measure the similarity between $\bar{x}'$ and $\bar{x}$ with the L1 norm:

$$L_R(G) = \|\bar{x}' - \bar{x}\|_1$$

Expanded:

$$L_R(G) = \|G(G(\bar{x}, r), r^{-1}) - \bar{x}\|_1$$

Why L1 instead of L2? L1 is more robust to outliers (L2 penalizes large deviations heavily, causing blur). In image/spectrogram reconstruction, L1 better preserves sharp edges and harmonic structure, avoiding over-smoothing.

15.4 4. Preventing Trivial Solutions

Without $L_R(G)$, the generator could ignore the input $\bar{x}$ and produce any "realistic-looking" spectrogram that fools D. Example: Always output an average spectrogram for speech of a given length — D might accept it, but it's useless for real TSM.

The cycle loss forces G to preserve content so that applying G forward and backward returns the original $\bar{x}$. This eliminates degenerate mappings and helps avoid mode collapse — where G maps many inputs to the same output.

15.5 5. Combined Loss

The final training objective is:

$$\min_G \max_D \; L_{LS}(G, D) + \lambda L_R(G)$$

Where:

- $L_{LS}(G, D)$ = adversarial LS-GAN loss (multi-scale PatchGAN).
- $L_R(G)$ = cycle-consistency L1 loss.
- λ = weight controlling importance of cycle loss vs adversarial loss.

Interpretation:

- D tries to maximize LS-GAN loss → better at distinguishing real vs fake.
- G tries to minimize LS-GAN loss → fool D into thinking generated spectrograms are real, while minimizing cycle loss → keep spectral content intact.

15.6 6. Step-by-Step Numeric Example

Let's simulate a tiny 1D "spectrogram" with 4 values.

Original:

$$\bar{x} = [1.0, 2.0, 3.0, 4.0]$$

Suppose $r = 2$ (stretch in time).

First pass:

$$G(\bar{x}, r) \to \bar{y} = [1.0, 1.5, 2.5, 3.5, 4.0, 4.0, 4.0, 4.0]$$

Reverse pass:

$$G(\bar{y}, r^{-1}) \to \bar{x}' = [1.1, 2.0, 3.1, 3.9]$$

Cycle loss:

$$L_R(G) = |1.1 - 1.0| + |2.0 - 2.0| + |3.1 - 3.0| + |3.9 - 4.0| = 0.1 + 0 + 0.1 + 0.1 = 0.3$$

This 0.3 is multiplied by λ and added to G's adversarial loss.

Over training: G must produce $\bar{y}$ that fools D and ensures $\bar{x}' \approx \bar{x}$ — preventing content drift.

E. U-Net Architecture

Table 1: U-Net Structure used in the ScalerGAN paper

Stage	Operation	Output shape (time × freq × channels)
Input	Spectrogram patch (e.g., $64 \times 64 \times 1$)	$64 \times 64 \times 1$
Encoder block 1	Conv + ReLU + Downsample (stride 2)	$32 \times 32 \times 64$
Encoder block 2	Conv + ReLU + Downsample	$16 \times 16 \times 128$
Encoder block 3	Conv + ReLU + Downsample	$8 \times 8 \times 256$
Bottleneck	6 Residual Blocks [**?**]: each with Conv, ReLU, skip connections	$8 \times 8 \times 256$
Decoder block 1	Upsample + Conv + ReLU + concatenate skip connection from Encoder 3	$16 \times 16 \times 256$
Decoder block 2	Upsample + Conv + ReLU + concatenate skip connection from Encoder 2	$32 \times 32 \times 128$
Decoder block 3	Upsample + Conv + ReLU + concatenate skip connection from Encoder 1	$64 \times 64 \times 64$
Output layer	Conv (1×1) + Activation (e.g., Tanh or Linear)	$64 \times 64 \times 1$

We use a typical setting inspired by the paper:

- **Input:** Mel-spectrogram of size 80 frequency bins × 256 time frames × 1 channel.
- **Downsampling factor:** ×2 per encoder stage.
- **Encoder:** 4 stages before bottleneck; channels start at 64 and double each stage.
- **Bottleneck:** 6 residual blocks at smallest resolution (channels fixed).
- **Decoder:** Mirrors encoder with upsampling and skip connections.

Encoder Path

Stage	Operation	Freq × Time × Channels	Notes
Input	—	$80 \times 256 \times 1$	Original interpolated Mel spectrogram
Enc1	Conv (stride 1) → Conv (stride 2)	$40 \times 128 \times 64$	Halved resolution, 64 feature maps
Enc2	Conv → Conv (stride 2)	$20 \times 64 \times 128$	Channels doubled
Enc3	Conv → Conv (stride 2)	$10 \times 32 \times 256$	Larger receptive field
Enc4	Conv → Conv (stride 2)	$5 \times 16 \times 512$	Smallest resolution before bottleneck

Bottleneck (Latent Space)

Stage	Operation	Freq × Time × Channels	Notes
Bottleneck	6 residual blocks	$5 \times 16 \times 512$	High-dimensional latent space: few spatial locations but rich feature depth (512 dims per location). Receptive field covers most of input.

If we flatten this latent tensor:

$$\text{Size} = 5 \times 16 \times 512 = 40,960$$

Compared to the original:

$$80 \times 256 \times 1 = 20,480$$

Thus, the bottleneck maps into a higher-dimensional feature space with reduced spatial resolution.

Decoder Path

Stage	Operation	Freq × Time × Channels	Notes
Dec4	Upsample (×2) → Concat skip from Enc4 → Conv	$10 \times 32 \times 512$	Merge global features with Enc4 details
Dec3	Upsample → Concat skip from Enc3 → Conv	$20 \times 64 \times 256$	Detail resolution increases
Dec2	Upsample → Concat skip from Enc2 → Conv	$40 \times 128 \times 128$	More fine-grained
Dec1	Upsample → Concat skip from Enc1 → Conv	$80 \times 256 \times 64$	Almost full detail restored
Output	Conv (1×1)	$80 \times 256 \times 1$	Final refined spectrogram

Key Takeaways

- **Encoder:** Reduces resolution ($80 \times 256 \rightarrow 5 \times 16$) while increasing channels ($1 \rightarrow 512$), forming a rich latent representation.
- **Bottleneck:** Small spatial size but large channel depth; heavy processing (residual blocks) is efficient here.
- **Decoder:** Gradually upsamples while merging encoder's detail with bottleneck's global corrections.
- **Skip connections:** Preserve fine structure (harmonics, formants) lost in downsampling.