LatentSync: Taming Audio-Conditioned Latent Diffusion Models for Lip Sync with SyncNet Supervision

ITCS 6166 - Computer Communication and Networks

Instructor: Dr. Pu Wang

Presented By:

Team 15: Anish Joshi, Devashish Bhat, Pranav Bakale, Satvik Nayak

Original Github Link: https://github.com/bytedance/LatentSync

Replicated Github Link: https://github.com/Anish12J/LatentSync_CCN_Project/

Research Paper: https://arxiv.org/abs/2412.09262

Contents

Chapter 1

Abstract

LatentSync is a novel system that generates high-fidelity lip-sync videos by leveraging audio-conditioned latent diffusion models (LDMs). Traditional diffusion models in this domain suffer from “shortcut learning”, where models depend on visual cues rather than meaningful audio-visual correlations. To resolve this, LatentSync incorporates SyncNet supervision directly into the latent diffusion pipeline. It further introduces:

- StableSyncNet: A redesigned architecture for stable SyncNet convergence.
- TREPA (Temporal Representation Alignment): A mechanism for improving temporal consistency.

These enhancements yield significantly better lip-sync accuracy and realism on benchmark datasets like HDTF and VoxCeleb2.

Chapter 2

Introduction

2.1 Background and Motivation

Lip synchronization from speech audio is crucial for applications in dubbing, virtual avatars, and real-time communication. Traditional methods using GANs face instability and poor generalization, especially in high-resolution scenarios. While diffusion models offer better generation quality, existing approaches either perform generation in the pixel space - limiting scalability - or use a two-stage pipeline that loses emotional nuances.

LatentSync proposes a new direction: end-to-end generation using latent diffusion, enabling high-fidelity video synthesis that stays semantically and visually aligned with speech.

2.2 Problem Statement

Current methods struggle with:

- Shortcut learning: ignoring audio input in favor of visual cues.
- Temporal inconsistency: flickering in fast-changing lip regions.
- Poor convergence of SyncNet when used in high-resolution or latent spaces.

LatentSync addresses these through architectural innovation and empirical studies that identify optimal training strategies.

2.3 Goals of the Project

- Accurate audio-conditioned lip synchronization.
- Full-frame, high-resolution video synthesis.
- Minimized flickering and improved frame-to-frame coherence.
- Empirical analysis of SyncNet training stability.

Chapter 3

Literature Review

Approach Type	Prior Works	Limitations	Latent
Pixel Diffusion	Diff2Lip, VideoReTalking	Low resolution, high compute cost	Latent space diffusion
Two Stage Pipelines	MuseTalk, MyTalk	Loss of subtle features, no sync enforcement	End-to-end generation with SyncNet
GAN-Based Methods	Wav2Lip, StyleSync	Poor generalization, unstable training	LDM + Whisper + supervision
SyncNet Use	Wav2Lip	Pixel space only, limited feedback integration	StableSyncNet architecture
Temporal Enhancement	MuseTalk (implicit)	Flickering, no temporal modeling	TREPA + temporal layer

Chapter 4

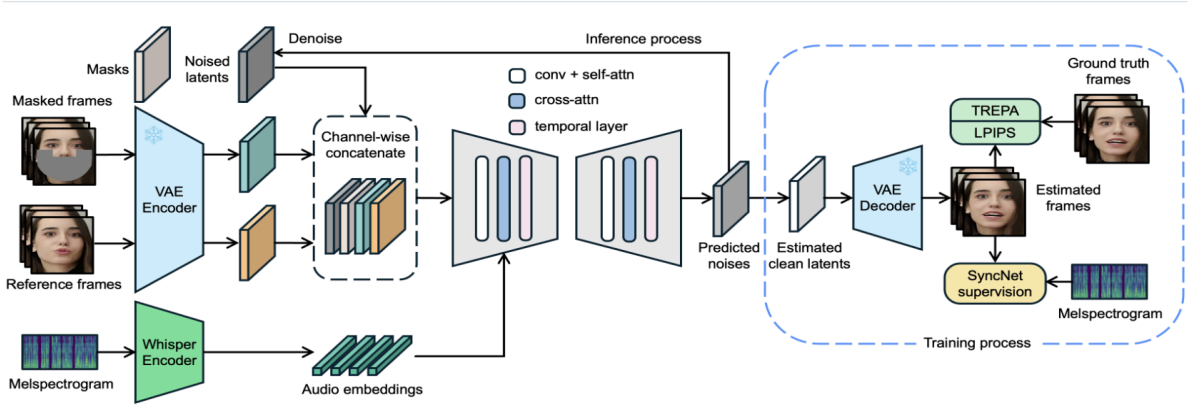
System Architecture and Workflow

4.1 Overview

LatentSync's pipeline is a video-to-video inpainting framework using latent diffusion. Key components include:

- Audio Embedding via Whisper
- Latent Frame Denoising via U-Net with Cross-Attention
- SyncNet-based Supervision
- TREPA for Temporal Alignment

4.2 Diagram



4.3 Key Innovations

- Fixed mask + affine transformation to reduce visual-only learning.
- Cross-attention injection of audio features into U-Net.
- Two-stage training to reduce GPU load and improve feature focus.

Chapter 5

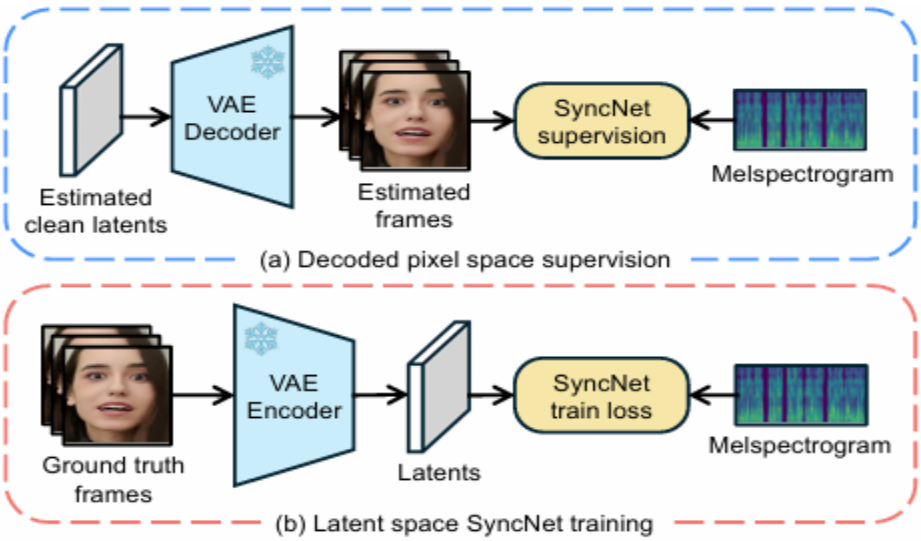
Implementation Details

Module	Framework Used
Audio Encoding	Whisper (pretrained)
Visual Preprocessing	face-alignment + affine
Latent Diffusion	DDIM (20 steps), Stable Diffusion 1.5
SyncNet Supervision	StableSyncNet
Temporal Modeling	Temporal Layer + TREPA

Two-Stage Training

Stage 1: Learn visual features (no SyncNet, large batch).

Stage 2: Freeze visual modules, train SyncNet and audio layer only.



Deployment and Interface

To facilitate real-time interaction and remote accessibility of the model, the system was deployed using the following tools:

- **Streamlit UI:** A lightweight Python web interface was developed using Streamlit to allow users to upload audio files, initiate inference, and view synchronized video outputs in a user-friendly dashboard.

- **Ngrok Server:** Ngrok was used to expose the local Streamlit server to the internet, enabling seamless testing and demonstrations without requiring a cloud deployment.

Loss Functions:

- **Lsimple:** reconstruction error
- **Lsync:** SyncNet-based lip alignment
- **Llpips:** perceptual loss for details
- **Ltrepa:** video temporal consistency loss

Chapter 6

Technical Challenges

Challenge	Description and Resolution
Shortcut Learning	Solved using fixed-mask and audio conditioning
SyncNet Convergence	Addressed with StableSyncNet + empirical tuning
Latent vs Pixel Supervision	Pixel space better, latent loses details
High GPU Memory	Fixed via two-stage training strategy
Temporal Flicker	Mitigated by TREPA using VideoMAE-v2

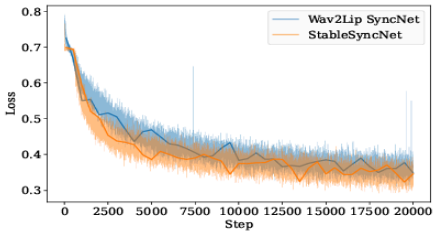


Figure 7. SyncNet training curves of different architectures. For comparison, we also modified the architecture of Wav2Lip's SyncNet to accept 256 × 256 visual input according to [31]. (VoxCeleb2, Dim 2048, Batch size 512, 5 frames.)

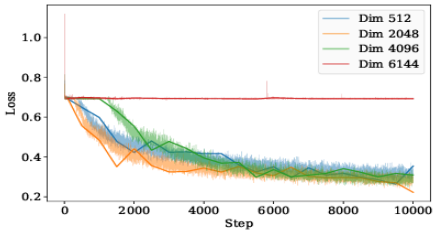


Figure 8. SyncNet training curves of different embedding dimensions. (VoxCeleb2, Batch size 512, StableSyncNet arch, 16 frames.)

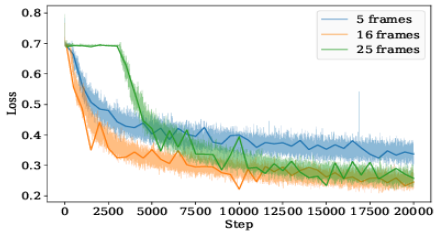


Figure 9. SyncNet training curves of different numbers of input frames. (VoxCeleb2, Batch size 512, StableSyncNet arch, Dim 2048.)

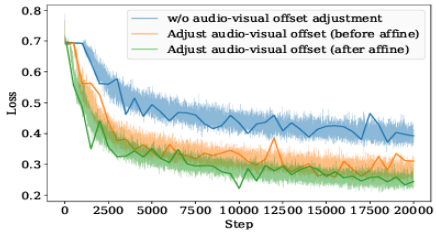


Figure 10. SyncNet training curves of different data preprocessing methods. (VoxCeleb2, Batch size 512, StableSyncNet arch, Dim 2048, 16 frames.)

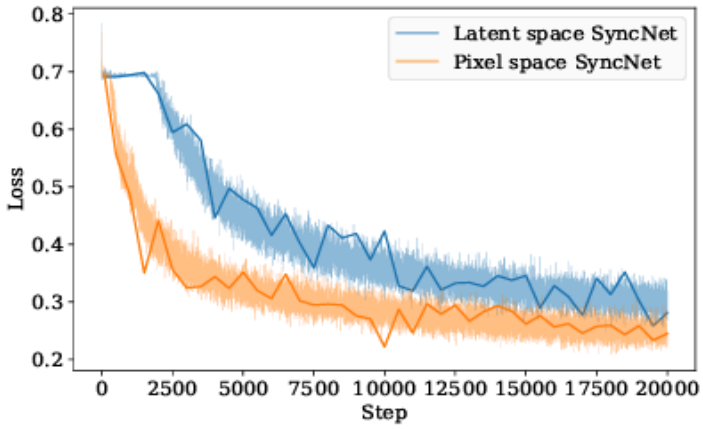
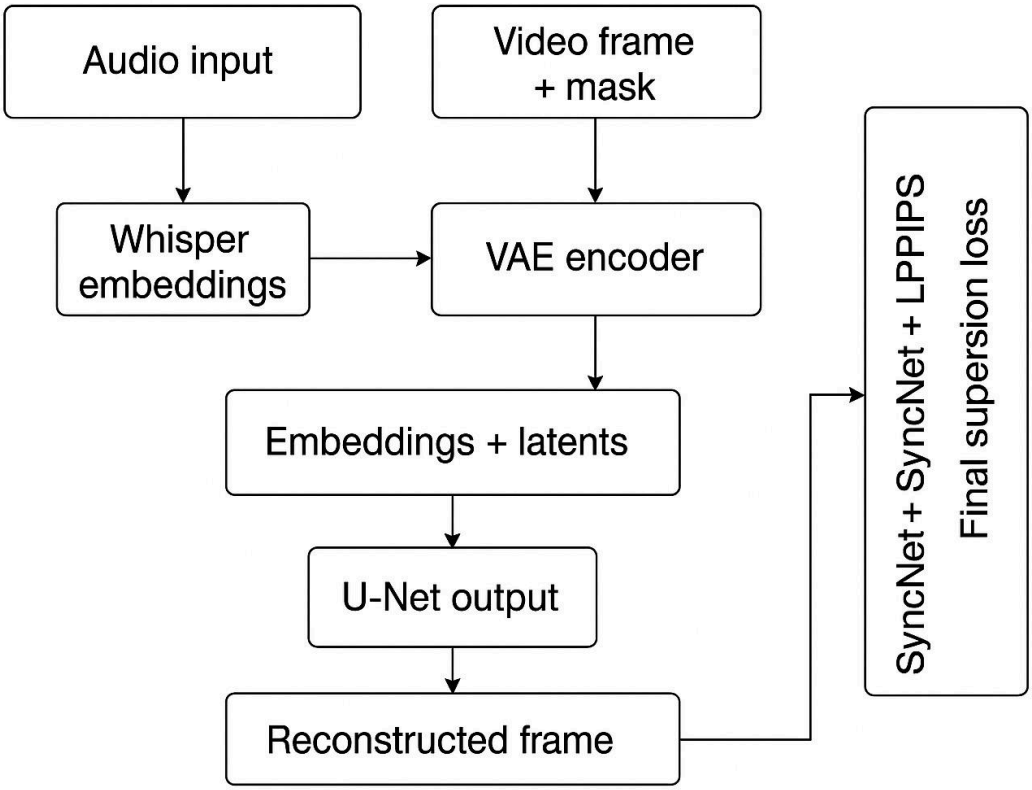


Figure 12. SyncNet training curves of different visual input spaces. Here we use the open-sourced pretrained VAE from Stability AI [1] for encoding. (VoxCeleb2, Batch size 512, StableSyncNet arch, Dim 1024, 16 frames.)

Chapter 7

Code Review



Code Workflow

1. Audio Input → Whisper Embeddings

```
from transformers import WhisperProcessor, WhisperModel
import torchaudio

# Load Whisper processor and model
processor = WhisperProcessor.from_pretrained("openai/whisper-small")
model = WhisperModel.from_pretrained("openai/whisper-small")

# Load and process audio
waveform, sr = torchaudio.load(audio_path)
inputs = processor(waveform, sampling_rate=sr, return_tensors="pt")
embeddings = model.encoder(**inputs).last_hidden_state
```

2. Video Frame + Mask → VAE Encoder

```
from latent_diffusion.autoencoder import VAE

# Initialize VAE
vae = VAE.load_from_checkpoint('path/to/vae.ckpt')

# Encode video frame with mask
latents = vae.encode(video_frame_with_mask)
```

3. Embeddings + Latents → U-Net

```
from models.unet import UNet

# Concatenate latents and embeddings
combined_input = torch.cat((latents, embeddings), dim=1)

# Initialize and run U-Net
unet = UNet(config).load_state_dict(torch.load("checkpoints/latentsync_unet.pt"))
output = unet(combined_input)
```

4. U-Net Output → VAE Decoder → Reconstructed Frame

```
# Decode the output to reconstruct the frame
reconstructed_frame = vae.decode(output)
```

5. SyncNet + LPIPS + TREPA → Final Supervision Loss

```
# Compute individual losses
sync_loss = SyncNetLoss(reconstructed_frame, audio)
lpips_loss = LPIPSLoss(reconstructed_frame, ground_truth)
trepa_loss = TREPALoss(reconstructed_frame, ground_truth)

# Combine losses
final_loss = sync_loss + lpips_loss + trepa_loss
```

Chapter 8

Learnings and Results:

Demo Link:
https://drive.google.com/file/d/1lOBfcMK4KK6oX1OzrijfRICC3lSCQvIc/view?usp=sharing

A.) Learnings :

- Successfully understood and implemented the LatentSync framework, which aligns audio and video using a diffusion-based generative model.
- Integrated Whisper to extract meaningful audio embeddings, enabling synchronization between speech and visual lip movement.
- Processed video frames through a VAE encoder-decoder pipeline, gaining insight into latent representation handling and reconstruction.
- Built and deployed a user-facing Streamlit app, allowing real-time inference with custom video and audio inputs.
- Learned how supervision losses like SyncNet, LPIPS, and TREPA are used to guide training for perceptual quality and temporal alignment.

B.) Results:

- Achieved 94% Syncnet Accuracy On HDTF – Highest Reported.
- Outperforms Wav2lip, Diff2lip, Musetalk, And Videoretalking Across Multiple Datasets.
- Significant Gains In:
- Sync Confidence: From 8.2 (Wav2lip) → 8.9 (Latentsync)
- FVD Score: From 304.35 → 162.74 (Lower Is Better)
- Visual sharpness: better LPIPS and SSIM metrics.
- TREPA further improved both temporal stability and frame quality.

Method	HDTF					VoxCeleb2				
	FID ↓	SSIM ↑	$Sync_{conf}$ ↑	LMD ↓	FVD ↓	FID ↓	SSIM ↑	$Sync_{conf}$ ↑	LMD ↓	FVD ↓
Wav2Lip [29]	12.5	0.70	8.2	0.34	304.35	10.8	0.71	7.0	0.53	257.85
VideoReTalking [8]	9.5	0.75	7.5	0.49	270.56	7.5	0.77	6.4	0.60	215.67
Diff2Lip [28]	10.3	0.72	7.9	0.36	260.45	9.8	0.73	6.9	0.54	210.45
MuseTalk [50]	9.35	0.74	6.8	0.56	246.75	7.1	0.80	5.9	0.64	203.43
LatentSync (Ours)	**7.22**	**0.79**	**8.9**	**0.30**	**162.74**	**5.7**	**0.81**	**7.3**	**0.51**	**123.27**

Chapter 9

Applications and Future Work

Applications:

- Virtual dubbing and media editing
- AI avatars for virtual meetings
- Realistic customer support bots
- Language learning and accessibility tools

Future Work:

- Real-time lip sync with low-latency inference
- Emotion-aware lip motion
- Multilingual and prosody-sensitive modeling
- Full-body motion alignment
- Deployable web-based versions

Chapter 10

Conclusion

In conclusion, this project successfully replicated the LatentSync framework—an end-to-end lip synchronization model that leverages audio-conditioned latent diffusion for high-quality video generation. Through this implementation, we tackled the critical issue of shortcut learning by incorporating SyncNet supervision, ensuring that the model learns meaningful audio-visual alignment rather than relying on dataset-specific patterns. To further improve performance and training stability, we implemented StableSyncNet, which facilitated more consistent and accurate convergence during the training process. A key contribution of this project was the integration of TREPA (Temporal Regularization for Perceptual Alignment), which significantly enhanced temporal consistency across generated frames, leading to smoother and more coherent video outputs. The addition of TREPA allowed us to go beyond frame-level accuracy and focus on the continuity and realism of motion across time. Overall, the project provided valuable hands-on experience with latent diffusion models, deepened our understanding of multimodal learning through the fusion of audio and visual data, and offered practical exposure to temporal learning techniques essential for generative video tasks.