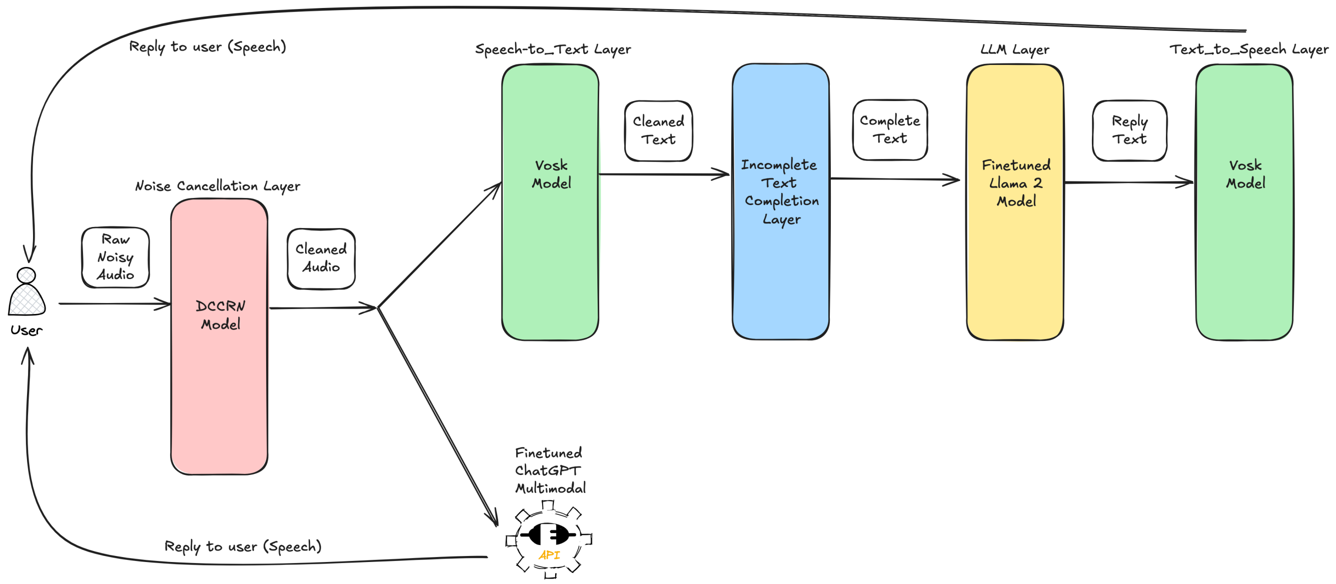
Architecture Diagram




```
# Logging to Hugging Face
from huggingface_hub import login

login()
```

Install Libraries

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Install Libraries

# For noise cancellation with ConvTasNet
!pip install asteroid

# For free speech-to-text
!pip install vosk

#For audio processing
!pip install torchaudio

# For audio speech-to-text processing
!pip install git+https://github.com/openai/whisper.git

# Chatgpt API for text processing
!pip install openai
```

Show hidden output

```
!wget https://alphacephei.com/vosk/models/vosk-model-en-us-0.22.zip
```

```
--2025-04-12 07:26:04--  https://alphacephei.com/vosk/models/vosk-model-en-us-0.22.zip
Resolving alphacephei.com (alphacephei.com)... 188.40.21.16, 2a01:4f8:13a:279f::2
Connecting to alphacephei.com (alphacephei.com)|188.40.21.16|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1913365522 (1.8G) [application/zip]
Saving to: 'vosk-model-en-us-0.22.zip'

vosk-model-en-us-0. 100%[===================>]   1.78G  25.0MB/s    in 90s

2025-04-12 07:27:34 (20.3 MB/s) - 'vosk-model-en-us-0.22.zip' saved [1913365522/1913365522]
```

```
!unzip vosk-model-en-us-0.22.zip
```

```
Archive:  vosk-model-en-us-0.22.zip
   creating: vosk-model-en-us-0.22/
   creating: vosk-model-en-us-0.22/am/
  inflating: vosk-model-en-us-0.22/am/final.mdl
  inflating: vosk-model-en-us-0.22/am/tree
   creating: vosk-model-en-us-0.22/ivector/
  inflating: vosk-model-en-us-0.22/ivector/final.dubm
  inflating: vosk-model-en-us-0.22/ivector/final.ie
  inflating: vosk-model-en-us-0.22/ivector/final.mat
  inflating: vosk-model-en-us-0.22/ivector/splice.conf
  inflating: vosk-model-en-us-0.22/ivector/global_cmvn.stats
  inflating: vosk-model-en-us-0.22/ivector/online_cmvn.conf
  inflating: vosk-model-en-us-0.22/README
   creating: vosk-model-en-us-0.22/conf/
  inflating: vosk-model-en-us-0.22/conf/mfcc.conf
  inflating: vosk-model-en-us-0.22/conf/model.conf
   creating: vosk-model-en-us-0.22/graph/
  inflating: vosk-model-en-us-0.22/graph/disambig_tid.int
   creating: vosk-model-en-us-0.22/graph/phones/
 extracting: vosk-model-en-us-0.22/graph/phones/optional_silence.int
 extracting: vosk-model-en-us-0.22/graph/phones/optional_silence.csl
  inflating: vosk-model-en-us-0.22/graph/phones/align_lexicon.int
  inflating: vosk-model-en-us-0.22/graph/phones/silence.csl
  inflating: vosk-model-en-us-0.22/graph/phones/align_lexicon.txt
 extracting: vosk-model-en-us-0.22/graph/phones/optional_silence.txt
  inflating: vosk-model-en-us-0.22/graph/phones/disambig.txt
  inflating: vosk-model-en-us-0.22/graph/phones/word_boundary.int
  inflating: vosk-model-en-us-0.22/graph/phones/disambig.int
  inflating: vosk-model-en-us-0.22/graph/phones/word_boundary.txt
  inflating: vosk-model-en-us-0.22/graph/HCLG.fst
 extracting: vosk-model-en-us-0.22/graph/num_pdfs
  inflating: vosk-model-en-us-0.22/graph/phones.txt
  inflating: vosk-model-en-us-0.22/graph/words.txt
   creating: vosk-model-en-us-0.22/rnnlm/
  inflating: vosk-model-en-us-0.22/rnnlm/word_feats.txt
  inflating: vosk-model-en-us-0.22/rnnlm/special_symbol_opts.conf
  inflating: vosk-model-en-us-0.22/rnnlm/special_symbol_opts.txt
  inflating: vosk-model-en-us-0.22/rnnlm/feat_embedding.final.mat
  inflating: vosk-model-en-us-0.22/rnnlm/final.raw
   creating: vosk-model-en-us-0.22/rescore/
  inflating: vosk-model-en-us-0.22/rescore/G.carpa
  inflating: vosk-model-en-us-0.22/rescore/G.fst
  inflating: vosk-model-en-us-0.22/conf/ivector.conf
```

```
!pip install pysoundfile
```

```
Collecting pysoundfile
  Downloading PySoundFile-0.9.0.post1-py2.py3-none-any.whl.metadata (9.4 kB)
Requirement already satisfied: cffi>=0.6 in /usr/local/lib/python3.11/dist-packages (from pysoundfile) (1.17.1)
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-packages (from cffi>=0.6->pysoundfile) (2.22)
Downloading PySoundFile-0.9.0.post1-py2.py3-none-any.whl (24 kB)
Installing collected packages: pysoundfile
Successfully installed pysoundfile-0.9.0.post1
```

@article{reddy2019scalable, title={A Scalable Noisy Speech Dataset and Online Subjective Test Framework}, author={Reddy, Chandan KA and Beyrami, Ebrahim and Pool, Jamie and Cutler, Ross and Srinivasan, Sriram and Gehrke, Johannes}, journal={Proc. Interspeech 2019}, pages={1816--1820}, year={2019} }

```
# Clone the MS-SNSD dataset repository

!git clone https://github.com/microsoft/MS-SNSD.git
```

```
Cloning into 'MS-SNSD'...
remote: Enumerating objects: 29924, done.
remote: Total 29924 (delta 0), reused 0 (delta 0), pack-reused 29924 (from 1)
Receiving objects: 100% (29924/29924), 3.93 GiB | 47.80 MiB/s, done.
Resolving deltas: 100% (81/81), done.
Updating files: 100% (24399/24399), done.
```

```
import numpy as np
import pandas as pd
import IPython.display as ipd
import torch
import torchaudio
import asteroid.models as ast
import soundfile as sf
from scipy.io.wavfile import write
```

Noise Cancellation Models

DCCRN

We're not training now, just evaluating or inferring, so use the full network and fixed batch statistics.

During training, layers like Dropout and BatchNorm behave differently than during evaluation:

- Dropout randomly disables parts of the network to prevent overfitting.
- BatchNorm uses statistics from the batch rather than accumulated training stats.

```
# Check for GPU and set the device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

if device.type == 'cuda':
    print(f"GPU Name: {torch.cuda.get_device_name(0)}")
else:
    print("GPU not available, using CPU. Processing might be slow.")
    print("Tip: Go to Runtime > Change runtime type and select GPU as Hardware accelerator for faster results.")
```

```
Using device: cpu
GPU not available, using CPU. Processing might be slow.
Tip: Go to Runtime > Change runtime type and select GPU as Hardware accelerator for faster results.
```

```
from asteroid.models import DCCRNet
```

```
# Load the pre-trained DCCRNet model
# This will download the model weights the first time you run it
model_id = "JorisCos/DCCRNet_Libri1Mix_enhsingle_16k"
print(f"Loading model: {model_id}...")
try:
    model_DCCRNet = DCCRNet.from_pretrained(model_id).to(device)
    model_DCCRNet.eval()
    print(f"Model {model_id} loaded successfully.")
    # Get the model's expected sample rate (usually 16kHz for this one)
    try:
        target_sr = model_DCCRNet.sample_rate
    except AttributeError:
        print("Model doesn't directly expose sample_rate, assuming 16000 Hz based on model ID.")
        target_sr = 16000
    print(f"Model expects sample rate: {target_sr} Hz")

except Exception as e:
    print(f"Error loading model {model_id}: {e}")
    print("Please check the model ID and your internet connection.")
    raise e
```

```
Loading model: JorisCos/DCCRNet_Libri1Mix_enhsingle_16k...
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens),
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
pytorch_model.bin: 100%                                  16.4M/16.4M [00:00<00:00, 68.7MB/s]
Model JorisCos/DCCRNet_Libri1Mix_enhsingle_16k loaded successfully.
Model expects sample rate: 16000.0 Hz
```

Preparing datasets

```
'''
1. Clean Speech
2. Noisy Speech
3. Metrics
4. for reference text clean speech

noisy speech -> models -> output "CLEANED" speech -> evaluate on metrics + reference text clean speech

'''
```

```
'\n1. Clean Speech\n2. Noisy Speech\n3. Metrics\n4. for reference text clean speech\n\nnoisy speech -> models -> output
"CLEANED" speech -> evaluate on metrics + reference text clean speech\n\n'
```

```
# Load the clean_test audio file
cleantest0_waveform, sample_rate = torchaudio.load("/content/MS-SNSD/clean_test/clnsp0.wav")

# Play the audio
ipd.Audio(cleantest0_waveform.numpy(), rate=sample_rate)
```

0:00 / 0:11

```
# Load the noise_test audio file
noisetest0_waveform, sample_rate_noisetest0_waveform = torchaudio.load("/content/MS-SNSD/noise_test/AirConditioner_1.wav")

# Play the audio
print(sample_rate_noisetest0_waveform)
ipd.Audio(noisetest0_waveform.numpy(), rate=sample_rate)
```

```
16000
```

0:00 / 0:23

```
# Functions from MS-SNSD github
import sys
sys.path.append('/content/MS-SNSD')
from audiolib import snr_mixer


import os
import torchaudio

def get_audio_files(directory):
    """
    Reads audio files from the given directory and returns a list of dicts
    containing the waveform, sample rate, and filename.

    Args:
        directory: Path to the directory containing audio files.

    Returns:
        A list of dicts, each with waveform, sample rate, and filename.
    """
    audio_data = []
    for filename in os.listdir(directory):
        if filename.endswith(('.wav', '.mp3')):  # Add other formats if needed
            filepath = os.path.join(directory, filename)
            try:
                waveform, sample_rate = torchaudio.load(filepath)
                audio_data.append({
                    "filename": filename,
                    "noisetest_waveform": waveform,
                    "sample_rate_noisetest_waveform": sample_rate
                })
            except Exception as e:
                print(f"Error loading file {filename}: {e}")
    return audio_data



noise_directory = "/content/MS-SNSD/noise_test"
noise_audio_files = get_audio_files(noise_directory)



# Keep audio_files as a list
if noise_audio_files:

    first_audio_info = noise_audio_files[1]
    print(noise_audio_files[0]['filename'])
    waveform = first_audio_info['noisetest_waveform']
    sample_rate = first_audio_info['sample_rate_noisetest_waveform']

    ipd.display(ipd.Audio(waveform.numpy(), rate=sample_rate))
else:
    print("No audio files found in the specified directory.")
```

```
Neighbor_5.wav
```

0:03 / 0:30

Start coding or generate with AI.

AirConditioner_Noise Mix

```
cleantest0_waveform = np.squeeze(cleantest0_waveform)
noisetest0_waveform = np.squeeze(noisetest0_waveform)

# Match lengths
if len(noisetest0_waveform) < len(cleantest0_waveform):
    repeat_factor = int(np.ceil(len(cleantest0_waveform) / len(noisetest0_waveform)))
    noisetest0_waveform = np.tile(noisetest0_waveform, repeat_factor)[:len(cleantest0_waveform)]
else:
    noisetest0_waveform = noisetest0_waveform[:len(cleantest0_waveform)]

# 10 dB SNR
clean0_out, noise0_out, noisy0 = snr_mixer(cleantest0_waveform, noisetest0_waveform, snr=10)

# Play audio
print("Noisy Audio Clip")
ipd.display(ipd.Audio(noisy0, rate=sample_rate))
print("Clean Audio Clip")
ipd.display(ipd.Audio(clean0_out, rate=sample_rate))
```

```
Noisy Audio Clip
```

0:00 / 0:11

```
Clean Audio Clip
```

0:00 / 0:11

```
print(type(noisy0))
```

```
<class 'torch.Tensor'>
```

```
noisy0 = noisy0.squeeze().cpu().numpy().astype('float32')
```

```
write("/content/drive/MyDrive/noisy_audio_clips/noisy0.wav", sample_rate, noisy0)
```

Speech-To-Text Function Model (Whisper)

```
import whisper

model_Whisper = whisper.load_model("turbo")
```

```
100%|██████████████████████████████████████| 1.51G/1.51G [00:40<00:00, 39.6MiB/s]
```

```
# load audio and pad/trim it to fit 30 seconds
audio = whisper.load_audio("/content/MS-SNSD/clean_test/clnsp0.wav")
audio = whisper.pad_or_trim(audio)


# make log-Mel spectrogram and move to the same device as the model
mel = whisper.log_mel_spectrogram(audio, n_mels=model_Whisper.dims.n_mels).to(model_Whisper.device)


# detect the spoken language
_, probs = model_Whisper.detect_language(mel)
print(f"Detected language: {max(probs, key=probs.get)}")
```

```
Detected language: en
```

Run for 5mins - 6mins (11 secs audio clip) - For CPU

Run for 2.5mins (11 secs audio clip) - For v2-8 TPU

```
# decode the audio
options = whisper.DecodingOptions()
result = whisper.decode(model_Whisper, mel, options)
```

```
# Set your Google Drive path
save_path = "/content/drive/MyDrive/data_noise/cleantext0_transcription.txt"

# Write the transcription to the file
with open(save_path, "w", encoding="utf-8") as f:
    f.write(f"Detected language:{max(probs, key=probs.get)}\n")
    f.write(result.text)


# print the recognized text
print(result.text)
```

```
She seemed irritated. You've got no business up here. You took me by surprise. There would still be plenty of moments of
```

```
clean0_text = result.text


print(clean0_text)
```

```
She seemed irritated. You've got no business up here. You took me by surprise. There would still be plenty of moments of
```

Speech-To-Text Function Model (Vosk)

```
from vosk import Model, KaldiRecognizer
import os
import wave
import json


model_Vosk = Model('vosk-model-en-us-0.22')


# wf = wave.open("/content/MS-SNSD/clean_test/clnsp0.wav", "rb")
```

Run Models to Clean Noise

```
def preprocessing_noisy_audio_clip_DCCRNet(noisy0):
  print(f"\nUsing existing 'noisy0' variable.")

  # Assume noisy0 is your variable (currently a NumPy array)
  # Make sure you have the sample rate it was created with
  original_sr = 16000

  # --- Check if noisy0 is NumPy and Convert to PyTorch Tensor ---
  if isinstance(noisy0, np.ndarray):
      print("Converting NumPy array 'noisy0' to PyTorch tensor...")
      # Convert numpy array to PyTorch tensor. Use .float() as models usually expect float32
      noisy_tensor = torch.from_numpy(noisy0).float()
  elif torch.is_tensor(noisy0):
      print("'noisy0' is already a PyTorch tensor.")
      noisy_tensor = noisy0 # Use it directly if it's already a tensor
  else:
      # Handle other potential types or raise an error
      raise TypeError(f"Variable 'noisy0' is not a NumPy array or PyTorch tensor, but type: {type(noisy0)}")

  # --- Move tensor to device ---
  # Now use the *tensor* version ('noisy_tensor')
  noisy_waveform = noisy_tensor.to(device)
  print(f"Tensor moved to device: {noisy_waveform.device}")
  print(f"Original SR: {original_sr} Hz, Tensor Shape: {noisy_waveform.shape}")

  # --- Display Original Audio ---
  # ipd.Audio works best with NumPy arrays or lists on CPU
  print("\nOriginal Noisy Audio:")
  try:
      # Use the original numpy array if available and valid
      if isinstance(noisy0, np.ndarray):
          display_data = noisy0
      else: # Otherwise convert the tensor back to numpy on CPU
          display_data = noisy_waveform.cpu().numpy()
      ipd.display(ipd.Audio(data=display_data, rate=original_sr))
  except Exception as e:
      print(f"Could not play audio: {e}")


  # --- Preprocessing (Continues using the PyTorch tensor 'noisy_waveform') ---
  print("Starting preprocessing...")
  # 1. Resample if necessary
```

```
    if original_sr != target_sr:
        print(f"Resampling from {original_sr} Hz to {target_sr} Hz...")
        resampler = torchaudio.transforms.Resample(orig_freq=original_sr, new_freq=target_sr).to(device)
        noisy_waveform = resampler(noisy_waveform)
        print(f"Resampled shape: {noisy_waveform.shape}")

    # 2. Convert to Mono if necessary (DCCRNet expects mono)
    if noisy_waveform.dim() > 1 and noisy_waveform.shape[0] > 1: # Check if tensor has channel dim and > 1 channel
        print(f"Converting to mono...")
        noisy_waveform = torch.mean(noisy_waveform, dim=0, keepdim=True) # Average channels
        print(f"Mono shape: {noisy_waveform.shape}")

    # 3. Ensure correct shape for the model (batch, time)
    if noisy_waveform.dim() == 2 and noisy_waveform.shape[0] == 1: # If shape (1, time)
      noisy_waveform = noisy_waveform.squeeze(0) # Shape: (time)
    elif noisy_waveform.dim() > 1: # Handle unexpected extra dims if necessary
        print("Warning: Tensor has more dimensions than expected after mono conversion. Taking first element.")
        noisy_waveform = noisy_waveform[0] # Or adapt as needed

    # Add batch dimension -> (1, time)
    noisy_waveform = noisy_waveform.unsqueeze(0)
    print(f"Final input shape for model: {noisy_waveform.shape}")
    return noisy_waveform


# Noise Cancellation with DCCRNet


def DCCRNet_noise_cancellation(noisy_waveform0: torch.Tensor) -> torch.Tensor:
    print("\nPerforming noise cancellation on DCCRNet Model...")
    with torch.no_grad():  # Disable gradient calculations for inference
        estimated_clean_waveform = model_DCCRNet(noisy_waveform0)
    print(f"Inference complete. Output shape: {estimated_clean_waveform.shape}")
    return estimated_clean_waveform


def play_cleaned_noisy_audio_toNumpy(estimated_clean_waveform):
    print(type(estimated_clean_waveform))
    print(estimated_clean_waveform.shape)

    # Ensure tensor is on CPU before converting
    audio_np_cleannoisy0 = estimated_clean_waveform.squeeze().detach().cpu().numpy()

    sample_rate = 16000
    print("Cleaned Noisy Audio Clip: noisy0")
    ipd.display(ipd.Audio(audio_np_cleannoisy0, rate=sample_rate))

    return audio_np_cleannoisy0


noisy_waveform0 = preprocessing_noisy_audio_clip_DCCRNet(noisy0)
```

```
Using existing 'noisy0' variable.
Converting NumPy array 'noisy0' to PyTorch tensor...
Tensor moved to device: cpu
Original SR: 16000 Hz, Tensor Shape: torch.Size([176320])

Original Noisy Audio:

      0:01 / 0:11

Starting preprocessing...
Final input shape for model: torch.Size([1, 176320])
```

```
print(type(noisy_waveform0))
```

```
<class 'torch.Tensor'>
```

```
clean_noisy0_tensors = DCCRNet_noise_cancellation(noisy_waveform0)
```

```
Performing noise cancellation on DCCRNet Model...
Inference complete. Output shape: torch.Size([1, 1, 176320])
```

```
print(type(clean_noisy0_tensors))
```

```
<class 'torch.Tensor'>
```

```
clean_noisy0_numpy = play_cleaned_noisy_audio_toNumpy(clean_noisy0_tensors)
```

```
<class 'torch.Tensor'>
torch.Size([1, 1, 176320])
Cleaned Noisy Audio Clip: noisy0
```

0:01 / 0:11

Start coding or generate with AI.

Evaluation on Metrics

Evaluating the effectiveness of noise cancellation (NC)

```
from pystoi import stoi
from scipy.io import wavfile
from pesq import pesq


sample_rate = 16000


# def calculate_mse(clean_signal, processed_signal):
#     return np.mean((clean_signal - processed_signal) ** 2)

def calculate_mse_torch(clean_signal: torch.Tensor, processed_signal: torch.Tensor) -> torch.Tensor:
    return torch.mean((clean_signal - processed_signal) ** 2)


# def calculate_pesq(clean_signal, processed_signal, sample_rate):
#     return pesq(sample_rate, clean_signal, processed_signal, 'wb')
#     # 'wb' = wideband

def calculate_pesq(clean_signal: torch.Tensor, processed_signal: torch.Tensor, sample_rate: int) -> float:
    clean_np = clean_signal.squeeze().detach().cpu().numpy()
    processed_np = processed_signal.squeeze().detach().cpu().numpy()
    return pesq(sample_rate, clean_np, processed_np, 'wb')  # 'wb' = wideband


# def calculate_stoi(clean_signal, processed_signal, sample_rate):
#     return stoi(clean_signal, processed_signal, sample_rate, extended=False)

def calculate_stoi(clean_signal: torch.Tensor, processed_signal: torch.Tensor, sample_rate: int) -> float:
    clean_np = clean_signal.squeeze().detach().cpu().numpy()
    processed_np = processed_signal.squeeze().detach().cpu().numpy()
    return stoi(clean_np, processed_np, sample_rate, extended=False)


# def calculate_insertion_loss(noise_only_signal, noise_with_anc_signal):
#     noise_power = np.mean(noise_only_signal ** 2)
#     anc_power = np.mean(noise_with_anc_signal ** 2)
#     insertion_loss = 10 * np.log10(noise_power / anc_power)
#     return insertion_loss

def calculate_insertion_loss(noise_only_signal: torch.Tensor, noise_with_anc_signal: torch.Tensor) -> float:
    noise_np = noise_only_signal.squeeze().detach().cpu().numpy()
    anc_np = noise_with_anc_signal.squeeze().detach().cpu().numpy()
    noise_power = np.mean(noise_np ** 2)
    anc_power = np.mean(anc_np ** 2)
    insertion_loss = 10 * np.log10(noise_power / anc_power)
    return insertion_loss
```

```
print(type(clean0_out))
```

```
<class 'torch.Tensor'>
```

```
print(type(clean_noisy0_tensors))
```

```
<class 'torch.Tensor'>
```

```
calculate_mse_torch(clean0_out,clean_noisy0_tensors)
```

```
tensor(0.0022)
```

An MSE of 0.0022 suggests high similarity between "noisy0", clean and processed signal, which usually means effective noise suppression with minimal distortion

```
calculate_pesq(clean0_out,clean_noisy0_tensors, sample_rate)
```

```
2.010915756225586
```

This falls into the "Poor" category.

It suggests there's still noticeable distortion or degradation in perceived audio quality, even though the intelligibility (STOI) is excellent.

```
calculate_stoi(clean0_out,clean_noisy0_tensors, sample_rate)
```

```
np.float64(0.9702082388273956)
```

Suggests excellent speech clarity after noise cancellation.

Nearly indistinguishable from clean speech in terms of understandability.

```
calculate_insertion_loss(clean0_out, clean_noisy0_tensors)
```

```
np.float32(15.085939)
```

An insertion loss of 15.08 dB indicates that your noise cancellation algorithm is highly effective at suppressing background noise.

DCCRNet is great at removing noise and preserving clarity, it may still introduce artifacts (e.g., muffling, robotic sounds), which hurts perceived naturalness.

- Can improve in terms of natural sound quality (as perceived by listeners)

Evaluating Speech-To-Text Effectiveness

```
# Speech to text
# Evaluate on Word Error Rate metric (WER)
```

```
!pip install jiwer
```

```
Collecting jiwer
  Downloading jiwer-3.1.0-py3-none-any.whl.metadata (2.6 kB)
Requirement already satisfied: click>=8.1.8 in /usr/local/lib/python3.11/dist-packages (from jiwer) (8.1.8)
Collecting rapidfuzz>=3.9.7 (from jiwer)
  Downloading rapidfuzz-3.13.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Downloading jiwer-3.1.0-py3-none-any.whl (22 kB)
Downloading rapidfuzz-3.13.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.1/3.1 MB 36.5 MB/s eta 0:00:00
Installing collected packages: rapidfuzz, jiwer
Successfully installed jiwer-3.1.0 rapidfuzz-3.13.0
```

```
from jiwer import wer
```

```
import numpy as np
import torch

def prepare_audio_input_numpy(audio_input, target_dtype=np.float32):
    """
    Validates and transforms an audio tensor or ndarray to 1D float32 NumPy array for Vosk.

    Args:
        audio_input (torch.Tensor or np.ndarray): The audio input.
        target_dtype (np.dtype): Target dtype (default: np.float32).

    Returns:
        np.ndarray: 1D NumPy array suitable for Vosk transcription.
    """
    # Convert PyTorch tensor to NumPy
    if isinstance(audio_input, torch.Tensor):
        audio_input = audio_input.detach().cpu().numpy()

    # Squeeze to remove dimensions of size 1
    audio_array = np.squeeze(audio_input)

    # Ensure 1D shape for mono audio
    if audio_array.ndim != 1:
        raise ValueError(f"Expected 1D mono audio, got shape {audio_array.shape}")

    # Convert dtype
    audio_array = audio_array.astype(target_dtype)
```

```
    return audio_array


clean0_numpy = prepare_audio_input_numpy(clean0_out)


clean_noisy0_numpy = prepare_audio_input_numpy(clean_noisy0_tensors)
```

Transcribe using Vosk

```
!wget https://alphacephei.com/vosk/models/vosk-model-en-us-0.22.zip
!unzip vosk-model-en-us-0.22.zip
```

Show hidden output

```
import io
from vosk import Model, KaldiRecognizer
import json


def transcribe_Vosk(audio_array: np.ndarray, model_Vosk, sample_rate: int = 16000) -> str:
    # Ensure audio is in float32 [-1, 1]
    if audio_array.dtype != np.float32:
        audio_array = audio_array.astype(np.float32)

    # Convert to bytes in 16-bit PCM WAV format using an in-memory buffer
    buffer = io.BytesIO()
    sf.write(buffer, audio_array, sample_rate, format='WAV', subtype='PCM_16')
    buffer.seek(0)

    # Use Vosk recognizer on the buffer
    rec = KaldiRecognizer(model_Vosk, sample_rate)

    results = []
    while True:
        data = buffer.read(4000)
        if len(data) == 0:
            break
        if rec.AcceptWaveform(data):
            result = json.loads(rec.Result())
            results.append(result.get("text", ""))

    final_result = json.loads(rec.FinalResult())
    results.append(final_result.get("text", ""))

    transcription = " ".join(results)
    return transcription


text_reference = transcribe_Vosk(clean0_numpy, model_Vosk, sample_rate=16000)
print("Transcribed Text:", text_reference)
```

```
Transcribed Text: she seemed irritated you've got no business up here you took me by surprise there would still be plent
```

```
text_hypothesis = transcribe_Vosk(clean_noisy0_numpy, model_Vosk, sample_rate=16000)
print("Transcribed Text:", text_hypothesis)
```

```
Transcribed Text: she seemed irritated you've got no business up here he took me by surprise there would still be plenty
```

```
reference = text_reference
hypothesis = text_hypothesis

score = wer(reference, hypothesis)
print(f"WER: {score:.3f}")
```

```
WER: 0.053
```

Start coding or generate with AI.

Transcribe using Whisper

```
def transcribe_whisper_audio(audio_input, model_Whisper):
    """
    Transcribe audio using OpenAI Whisper.
```

```
    Args:
        audio_input (np.ndarray or torch.Tensor): The raw audio waveform.
        model_Whisper: Loaded Whisper model.

    Returns:
        dict: Dictionary containing language and transcription text.
    """
    import torch
    import numpy as np
    import whisper

    # Convert to NumPy if it's a tensor
    if isinstance(audio_input, torch.Tensor):
        audio_input = audio_input.cpu().numpy()

    # Pad or trim the audio to fit 30 seconds
    audio = whisper.pad_or_trim(audio_input)

    # Create log-Mel spectrogram and move to the model's device
    mel = whisper.log_mel_spectrogram(audio, n_mels=model_Whisper.dims.n_mels).to(model_Whisper.device)

    # Detect language
    _, probs = model_Whisper.detect_language(mel)
    detected_language = max(probs, key=probs.get)

    # Decode the audio
    options = whisper.DecodingOptions()
    result = whisper.decode(model_Whisper, mel, options)

    return {
        "language": detected_language,
        "text": result.text
    }
```

```
# 3mins 21secs
```

```
result = transcribe_whisper_audio(clean_noisy0_numpy, model_Whisper)
print("Language:", result["language"])
print("Transcribed Text:", result["text"])
```

```
Language: en
Transcribed Text: She seemed irritated. You've got no business up here. He took me by surprise. There would still be ple
```

```
refer_result = transcribe_whisper_audio(clean0_numpy, model_Whisper)
print("Language:", refer_result["language"])
print("Transcribed Text:", refer_result["text"])
```

```
Language: en
Transcribed Text: She seemed irritated. You've got no business up here. You took me by surprise. There would still be pl
```

```
noisy_text = result["text"]
reference_text = refer_result["text"]
```

```
# test_a = "this is a test"
# test_b = "this is test"

reference = reference_text
hypothesis = noisy_text

score = wer(reference, hypothesis)
print(f"WER: {score:.3f}")
```

```
WER: 0.026
```

Start coding or generate with AI.