```
#
# ============================================================================
# --- 1. INSTALL LIBRARIES ---
#
# ============================================================================
!apt-get install -y ffmpeg > /dev/null 2>&1
!pip install pydub tqdm -q

#
# ============================================================================
# --- 2. IMPORT LIBRARIES ---
#
# ============================================================================
import numpy as np
from scipy.io.wavfile import write, read
import matplotlib.pyplot as plt
from tqdm import tqdm
from IPython.display import Audio, display
from google.colab import files
from pydub import AudioSegment
import io

#
# ============================================================================
# --- 3. CONFIGURATION PARAMETERS ---
#
# ============================================================================
# Filenames
FILENAME_ORIGINAL = "original_audio.wav"
FILENAME_RECEIVED = "received_audio.wav"

# Signal processing parameters
CARRIER_FREQ = 4000
BIT_RATE = 800
# --- MEMORY OPTIMIZATION ---
# Using 4x the carrier frequency is a safe margin that uses less RAM.
RF_SAMPLE_RATE = 4 * CARRIER_FREQ
AMPLITUDE = 1.0
SNR_DB = 25

# --- MEMORY OPTIMIZATION ---
# We will resample all input audio to this rate. 16kHz is standard for voice.
TARGET_SAMPLE_RATE = 16000

#
# ============================================================================
# --- 4. HELPER FUNCTIONS (Unchanged) ---
# All helper functions from the previous version remain the same.
#
# ============================================================================

def upload_audio_colab():
print("□ Please upload a WAV audio file from your device...")
```

```
uploaded = files.upload()
if not uploaded:
print("\n No file was selected.")
return None
filename = list(uploaded.keys())[0]
print(f"\n File '{filename}' uploaded successfully.")
return filename

def bits_to_audio(bit_stream, output_file, sample_rate):
print(f" Converting bit stream back to audio at {sample_rate} Hz...")
extra_bits = len(bit_stream) % 16
if extra_bits != 0: bit_stream = bit_stream[:-extra_bits]
audio_samples = []
for i in range(0, len(bit_stream), 16):
chunk = bit_stream[i:i+16]
unsigned_val = int(chunk, 2)
if unsigned_val >= 32768:
signed_val = unsigned_val - 65536
else:
signed_val = unsigned_val
audio_samples.append(signed_val)
recovered_audio = np.array(audio_samples, dtype=np.int16)
write(output_file, sample_rate, recovered_audio)
print(f" Recovered audio saved as '{output_file}'")
return recovered_audio

def audio_to_bits(audio_file):
print(" Converting audio to bit stream...")
sample_rate, audio_data = read(audio_file)
if audio_data.ndim > 1: audio_data = audio_data.mean(axis=1)
bit_stream = ''.join([format(sample.view(np.uint16), '016b') for sample in
audio_data.astype(np.int16)])
print(f" Conversion complete. Detected sample rate: {sample_rate} Hz")
return bit_stream, audio_data.astype(np.int16), sample_rate

def modulate_bpsk(bit_stream):
print(" Modulating bit stream using BPSK...")
samples_per_bit = int(RF_SAMPLE_RATE / BIT_RATE)
t = np.linspace(0, len(bit_stream) / BIT_RATE, len(bit_stream) * samples_per_bit, endpoint=False)
carrier = AMPLITUDE * np.cos(2 * np.pi * CARRIER_FREQ * t)
modulating_signal = np.repeat([1 if bit == '0' else -1 for bit in bit_stream], samples_per_bit)
modulated_signal = carrier * modulating_signal
print(" Modulation complete.")
return modulated_signal, modulating_signal

def simulate_channel(signal):
print(f" Simulating noisy channel with SNR = {SNR_DB} dB...")
signal_power = np.mean(signal**2)
snr_linear = 10**(SNR_DB / 10.0)
noise_power = signal_power / snr_linear
noise = np.random.normal(0, np.sqrt(noise_power), len(signal))
received_signal = signal + noise
print(" Channel simulation complete.")
return received_signal
```

```
def demodulate_bpsk(received_signal):
print("□ Demodulating received signal...")
samples_per_bit = int(RF_SAMPLE_RATE / BIT_RATE)
reference_carrier = np.cos(2 * np.pi * CARRIER_FREQ * np.linspace(0, 1/BIT_RATE,
samples_per_bit, endpoint=False))
recovered_bits = ""
num_bits = len(received_signal) // samples_per_bit
for i in tqdm(range(num_bits), desc="Demodulating"):
segment = received_signal[i*samples_per_bit:(i+1)*samples_per_bit]
correlation = np.sum(segment * reference_carrier)
recovered_bits += '0' if correlation > 0 else '1'
print("□ Demodulation complete.")
return recovered_bits

def visualize(original_audio, bit_stream, modulated_signal, received_signal, recovered_audio,
sample_rate):
# This function is unchanged
print("□ Generating visualizations...")
plt.style.use('seaborn-v0_8-darkgrid')
fig, axs = plt.subplots(5, 1, figsize=(12, 18), constrained_layout=True)
fig.suptitle('Audio Transmission via BPSK Digital Modulation', fontsize=16)
time_audio = np.linspace(0, len(original_audio) / sample_rate, num=len(original_audio))
axs[0].plot(time_audio, original_audio, color='blue'); axs[0].set_title(f'1. Original Audio Waveform
({sample_rate} Hz)')
bits_to_show = 100
axs[1].step(range(bits_to_show), [int(b) for b in bit_stream[:bits_to_show]], where='post',
color='purple'); axs[1].set_title(f'2. Digital Bit Stream (First {bits_to_show} Bits)')
samples_to_show = int(5 * RF_SAMPLE_RATE / BIT_RATE)
time_modulated = np.linspace(0, 5/BIT_RATE, samples_to_show, endpoint=False)
axs[2].plot(time_modulated, modulated_signal[:samples_to_show], color='green');
axs[2].set_title('3. BPSK Modulated Signal (Zoomed In)')
axs[3].plot(time_modulated, received_signal[:samples_to_show], color='orange');
axs[3].set_title(f'4. Received Signal with AWGN (SNR = {SNR_DB} dB)')
time_recovered = np.linspace(0, len(recovered_audio) / sample_rate, num=len(recovered_audio))
axs[4].plot(time_recovered, recovered_audio, color='cyan'); axs[4].set_title('5. Recovered Audio
Waveform')
plt.show()

#
================================================================================
# --- 5. MAIN EXECUTION ---
#
================================================================================
if __name__ == "__main__":
uploaded_filename = upload_audio_colab()

if uploaded_filename:
print(f"□ Loading '{uploaded_filename}'...")
sound = AudioSegment.from_file(uploaded_filename)

# --- MEMORY OPTIMIZATION ---
# Resample the audio to the target sample rate to reduce data size.
print(f" resampling audio to {TARGET_SAMPLE_RATE} Hz...")
```

```
sound = sound.set_frame_rate(TARGET_SAMPLE_RATE)
# Ensure audio is mono for processing
sound = sound.set_channels(1)

# Export the standardized, downsampled audio file.
sound.export(FILENAME_ORIGINAL, format="wav")
print(f"✅ Audio successfully standardized and saved as '{FILENAME_ORIGINAL}'")

print("\n--- 🔊 DIAGNOSTIC: Playing the standardized source audio ---")
display(Audio(FILENAME_ORIGINAL))

# The rest of the simulation proceeds as before, but with less data.
bit_stream, original_audio_data, actual_sample_rate = audio_to_bits(FILENAME_ORIGINAL)
modulated_signal, _ = modulate_bpsk(bit_stream)
received_signal = simulate_channel(modulated_signal)
recovered_bit_stream = demodulate_bpsk(received_signal)
recovered_audio_data = bits_to_audio(recovered_bit_stream, FILENAME_RECEIVED,
actual_sample_rate)

print("\n--- ✅ Process Complete ✅ ---")
errors = sum(1 for a, b in zip(bit_stream, recovered_bit_stream) if a != b)
ber = errors / len(bit_stream) if len(bit_stream) > 0 else 0
print(f"Bit Error Rate (BER): {ber:.6f} ({errors} errors out of {len(bit_stream)} bits)")

visualize(original_audio_data, bit_stream, modulated_signal, received_signal,
recovered_audio_data, actual_sample_rate)
print("\n🔊 Received Audio (after transmission and noise):")
display(Audio(FILENAME_RECEIVED))
```