**API INTEGRATION**

Integration points between voice and machine learning (ML) modules typically involve multiple technology components interacting via APIs, streaming data pipelines, and neural network inference engines. Integration Points Between Voice and ML Modules

**1.Voice Data Capture and Preprocessing** Microphones or specialized sensors (e.g., wearable skin-attached acoustic sensors like SAAS) capture raw voice signals, including vibrations and acoustic data even in noisy environments

**2. Speech Recognition (ASR) Module** Converts audio signals into text using deep learning models such as residual neural networks or transformer-based architectures

**3.Natural Language Processing (NLP) and Intent Recognition** Processes the recognized text to extract user intent, entities, or commands using intent classification models or generative AI models like GPT-4

**4.Text-to-Speech (TTS) Module** Converts text responses or system messages back into audio output using neural voice synthesis, supporting multiple languages and custom voices

**5.Real-Time Interaction and Feedback Loop** Streaming APIs: Real-time voice interaction systems use WebSocket or streaming APIs to send audio input and receive text and audio output with minimal latency

```
# SPEECH TO TEXT CONVERSION

def process_voice_input(audio_stream):
    stt_request = {
        'audio': audio_stream,  # binary audio data
        'language': 'en-US',    # language code
        'sample_rate': 16000    # Hz
    }

    response = stt_api.post('/recognize', data=stt_request)
    return response.text        # extracted transcript


# ML MODEL API PSEUDOCODE

def analyze_text(text):
    ml_request = {
        'text': text,
        'models': ['intent', 'sentiment', 'ner'],  # Which ML models to run
        'format': 'json'
    }
    response = ml_api.post('/predict', data=ml_request)
    return {
        'intent': response.intent,          # Detected user intent
        'sentiment': response.sentiment_score,  # Sentiment analysis
        'entities': response.entities       # Named entities
    }


# TEXT TO SPEECH

def generate_speech(response_text):
    tts_request = {
        'text': response_text,
```

```
        'voice': 'en-US-Wavenet-D',  # Voice profile
        'speed': 1.0,                # Playback speed
        'pitch': 0.0                 # Pitch adjustment
    }
    response = tts_api.post('/synthesize', data=tts_request
    return response.audio  # Binary audio (e.g., MP3, WAV)
```

**PSEUDOCODE**

START

1. Listen to user voice and record audio audio = record_voice()
2. Clean the audio to remove noise clean_audio = clean_noise(audio)
3. Convert the clean audio to text using speech recognition text = speech_to_text(clean_audio)
4. Understand what the user wants (intent) intent = understand_intent(text)
5. Decide what to do based on the intent IF intent is recognized THEN

```
response = create_response(intent)
```

ELSE

```
response = "Sorry, I did not understand."
```

6. Convert the response text back to speech speech = text_to_speech(response)
7. Play the speech audio to the user play_audio(speech)

END