P2P Real-Time Audio Translation: HackArizona Project Document

1. Project Overview

1.1 Project Summary

This project aims to develop a real-time peer-to-peer audio translation system that enables seamless communication between speakers of different languages without requiring a common language. The system will leverage speech recognition, machine translation, and speech synthesis technologies to create a complete speech-to-speech pipeline.

1.2 Core Objectives

- Create a functional prototype that can translate speech between at least 2 languages in near real-time
- Achieve reasonable translation accuracy and naturalness of speech
- Develop a simple user interface to demonstrate the functionality
- Prepare a compelling demonstration for the hackathon judges

1.3 Technical Approach Overview

The solution will utilize a modular architecture consisting of three main components: 1. **Automatic Speech Recognition (ASR)**: Converting spoken language to text 2. **Machine Translation (MT)**: Translating text from source to target language 3. **Text-to-Speech Synthesis (TTS)**: Converting translated text to spoken audio

2. Pre-Hackathon Preparation

2.1 Technical Prerequisites

Category	Requirement	Priority	Notes
Hardware	2 laptops with microphones	Essential	Preferably with GPUs
	Headphones/speakers	Essential	For audio output
Software	Python 3.9+	Essential	Ensure compatible with all libraries
	PyTorch	Essential	Pre-installed and tested

Category	Requirement	Priority	Notes
	Required libraries	Essential	See detailed list in section 2.2
	Code editor	Essential	VSCode recommended
Development	GitHub repository	High	Set up with initial structure
	Virtual environment setup	High	Ensure dependency isolation
Knowledge	Basic ASR/MT/TTS concepts	Medium	Review documentation
	API usage for chosen models	High	Practice with examples

2.2 Library Requirements

Before the hackathon, ensure these libraries are installed and properly functioning: - `torch` and `torchaudio` - `transformers` (Hugging Face) - `sounddevice` or `pyaudio` (for audio I/O) - `SpeechRecognition` - `pydub` (for audio processing) - `gradio` (for creating demo interface) - `flask` (if creating a web application)

2.3 Model Selection & Preparation

Pre-download and test these models:

Component	Recommended Model	Alternative	Preparation Task
ASR	Whisper-small	Wav2Vec 2.0	Download model, run basic inference test
MT	M2M-100 (small)	NLLB-200 (distilled)	Download model, verify translation quality
TTS	FastSpeech 2	VITS	Test with sample texts, verify output quality

2.4 Dataset Preparation

Your milestone document mentions Common Voice and CoVoST datasets. Before the hackathon:

1. Download sample subsets of each dataset (not entire datasets)
2. Prepare 10-20 test audio samples for each target language
3. Verify dataset format is compatible with your model inputs
4. Create a small validation set for quick testing during development

2.5 Pre-Hackathon Development Tasks

Complete these tasks before the hackathon to maximize productive time:

1. Create skeleton code for each module (ASR, MT, TTS)
2. Set up data preprocessing functions
3. Implement basic audio capture functionality
4. Create a simple pipeline test that chains a single sample through all components
5. Set up a basic UI framework (command line or simple web interface)
6. Establish logging mechanisms for debugging
7. Design the basic architecture with agreed interfaces between components

3. Hackathon Execution Plan

3.1 Task Division Strategy

Team Member	Primary Responsibilities	Secondary Responsibilities
Kanit	ASR component, Audio I/O	Integration, Testing
Tanishk	MT component, TTS component	User Interface, Presentation

3.2 Detailed Timeline

Day 1 (First 12 Hours)

Time	Task	Owner	Deliverable
0-1h	Setup environment, verify all components	Both	Working development environment
1-3h	Implement ASR component	Kanit	Working speech recognition module
1-3h	Implement MT component	Tanishk	Working translation module
3-5h	Implement TTS component	Tanishk	Working speech synthesis module

Time	Task	Owner	Deliverable
3-5h	Implement audio I/O	Kanit	Audio capture and playback functionality
5-6h	Break & regroup	Both	Status update, issue identification
6-9h	Integrate all components into pipeline	Both	End-to-end pipeline prototype
9-11h	Basic error handling and robustness	Both	Improved pipeline stability
11-12h	Initial performance optimization	Both	Improved response time

Day 2 (Next 12 Hours)

Time	Task	Owner	Deliverable
12-15h	Develop simple user interface	Tanishk	Functional UI
12-15h	Implement pipeline optimizations	Kanit	Improved performance
15-17h	Testing and bug fixing	Both	Stable prototype
17-18h	Break & regroup	Both	Status update, final plans
18-21h	Implement language selection, UX improvements	Tanishk	Enhanced user experience
18-21h	Performance tuning, latency reduction	Kanit	Optimized performance
21-23h	Comprehensive testing	Both	Verified functionality
23-24h	Preparation for demo	Both	Demo script and materials

Day 3 (Final 12 Hours)

Time	Task	Owner	Deliverable
24-27h	Create presentation materials	Tanishk	Slides, videos, graphics
24-27h	Final bug fixes and optimizations	Kanit	Polished application
27-30h	Rehearse presentation	Both	Finalized presentation
30-33h	Final testing and contingency buffer	Both	Production-ready demo
33-36h	Submission preparation	Both	Complete submission package

3.3 Contingency Planning

Prioritize features in this order: 1. Working ASR + MT (text output only) 2. Complete ASR + MT + TTS pipeline 3. User interface improvements 4. Language selection 5. Performance optimizations

If facing time constraints, focus on demonstrating core translation functionality even with limited language options or simplified UI.

4. Technical Architecture & Implementation

4.1 System Architecture

```
[User A]                                       [User B]



 Audio        ASR             MT              Audio
 Capture    (Whisper)      (M2M-100)      Output





                              TTS
                              (FastSpeech2)
```

4.2 Component Details

4.2.1 ASR Component

- Use Whisper-small for optimal speed/accuracy balance
- Implementation approach:

```
from transformers import WhisperProcessor, WhisperForConditionalGeneration

# Pre-load model during initialization
processor = WhisperProcessor.from_pretrained("openai/whisper-small")
model = WhisperForConditionalGeneration.from_pretrained("openai/whisper-small")

def transcribe_audio(audio_array, sampling_rate=16000):
    # Process audio input
    input_features = processor(audio_array, sampling_rate=sampling_rate, return_tensors

    # Generate transcription
    predicted_ids = model.generate(input_features)
    transcription = processor.batch_decode(predicted_ids, skip_special_tokens=True)[0]

    return transcription
```

4.2.2 MT Component

- Use M2M-100 (small) for multilingual translation
- Implementation approach:

```
from transformers import M2M100ForConditionalGeneration, M2M100Tokenizer

# Pre-load model during initialization
tokenizer = M2M100Tokenizer.from_pretrained("facebook/m2m100_418M")
model = M2M100ForConditionalGeneration.from_pretrained("facebook/m2m100_418M")

def translate_text(text, source_lang, target_lang):
    # Set source language
    tokenizer.src_lang = source_lang

    # Tokenize input text
    encoded_text = tokenizer(text, return_tensors="pt")

    # Generate translation
    generated_tokens = model.generate(
        **encoded_text,
        forced_bos_token_id=tokenizer.get_lang_id(target_lang)
    )

    # Decode translation
    translation = tokenizer.batch_decode(generated_tokens, skip_special_tokens=True)[0]
```

```
        return translation
```

4.2.3 TTS Component

- Use FastSpeech2 for quick synthesis
- Implementation approach:

```
from transformers import SpeechT5Processor, SpeechT5ForTextToSpeech
import torch
import soundfile as sf

# Pre-load model during initialization
processor = SpeechT5Processor.from_pretrained("microsoft/speecht5_tts")
model = SpeechT5ForTextToSpeech.from_pretrained("microsoft/speecht5_tts")

def synthesize_speech(text, output_path=None):
    # Tokenize text
    inputs = processor(text=text, return_tensors="pt")

    # Generate speech
    speech = model.generate_speech(inputs["input_ids"], speaker_embeddings=None)

    if output_path:
        sf.write(output_path, speech.numpy(), samplerate=16000)

    return speech.numpy(), 16000
```

4.2.4 Audio I/O Use PyAudio or SoundDevice for real-time audio capture and playback:

```
import sounddevice as sd
import numpy as np

def record_audio(duration=5, sample_rate=16000):
    print("Recording...")
    audio_data = sd.rec(int(duration * sample_rate), samplerate=sample_rate, channels=1, dty
    sd.wait()
    print("Recording complete")
    return audio_data.flatten()

def play_audio(audio_data, sample_rate=16000):
    sd.play(audio_data, sample_rate)
    sd.wait()
```

4.3 Pipeline Integration

```
def translation_pipeline(audio_input, source_lang, target_lang):
    # Step 1: Speech-to-text
    transcription = transcribe_audio(audio_input)

    # Step 2: Text translation
    translation = translate_text(transcription, source_lang, target_lang)

    # Step 3: Text-to-speech
    audio_output, sample_rate = synthesize_speech(translation)

    return {
        "transcription": transcription,
        "translation": translation,
        "audio_output": audio_output,
        "sample_rate": sample_rate
    }
```

4.4 UI Implementation

Use Gradio for rapid UI development:

```
import gradio as gr

def create_interface():
    with gr.Blocks() as demo:
        gr.Markdown("# Real-Time P2P Audio Translation")

        with gr.Row():
            with gr.Column():
                source_lang = gr.Dropdown(["en", "es", "fr", "de"], label="Source Language"
                audio_input = gr.Audio(source="microphone", type="numpy", label="Input Audi

            with gr.Column():
                target_lang = gr.Dropdown(["en", "es", "fr", "de"], label="Target Language"
                audio_output = gr.Audio(label="Translated Audio")

        with gr.Row():
            transcription = gr.Textbox(label="Transcription")
            translation = gr.Textbox(label="Translation")

        translate_btn = gr.Button("Translate")

        translate_btn.click(
            fn=translation_pipeline,
            inputs=[audio_input, source_lang, target_lang],
            outputs=[transcription, translation, audio_output]
        )

    return demo

if __name__ == "__main__":
    interface = create_interface()
    interface.launch()
```

5. Testing Strategy

5.1 Component Testing

Component	Test Approach	Test Data	Success Criteria
ASR	Test with sample recordings	Common Voice samples	>80% word accuracy
MT	Compare with reference translations	CoVoST paired data	Semantic correctness
TTS	Subjective evaluation	Generated output	Intelligibility
Audio I/O	Record and playback test	Live recording	Clear audio quality

5.2 Integration Testing

Test the complete pipeline with various scenarios: - Short phrases (1-2 seconds) - Medium sentences (3-5 seconds) - Longer statements (5-10 seconds) - Different speaker accents - Background noise conditions - Different language pairs

5.3 Performance Benchmarks

Metric	Target	Measurement Method
End-to-end latency	<3 seconds	Stopwatch timing
ASR accuracy	>80%	Word Error Rate
MT adequacy	Comprehensible	Manual evaluation
TTS quality	Intelligible	Manual evaluation

6. Presentation & Demo

6.1 Demo Script

1. Introduction (1 minute)
    - Problem statement

    - Solution approach
    - Technologies used
2. Live Demo (3 minutes)
    - Show translation between two common language pairs
    - Demonstrate real-time capability
    - Show transcription and translation text
3. Technical Highlights (1 minute)
    - Architecture overview
    - Challenges overcome
    - Future improvements

6.2 Presentation Materials

- Create 3-5 slides covering:
    - Problem and solution
    - Architecture diagram
    - Technical highlights
    - Results and metrics
    - Future work
- Prepare a 1-minute video backup in case of technical issues

6.3 Judging Criteria Alignment

Criterion	Project Strength	Emphasis Point
Technical Difficulty	ML pipeline integration	Complex integration of three ML tasks
Innovation	Real-time P2P approach	Focus on latency optimizations
Completeness	End-to-end solution	Demonstrate full pipeline functionality
Practicality	Real-world use case	Emphasize communication barrier solution
Presentation	Clear demo	Show real-time translation in action

7. Pre-Checks & Quality Assurance

7.1 Pre-Hackathon Checklist

☐ Development environment setup on both laptops
☐ All required libraries installed and tested
☐ Models pre-downloaded and verified
☐ Sample audio files prepared
☐ GitHub repository initialized with README

☐ Basic project structure created
☐ Individual component test scripts ready
☐ Internet connectivity backup plan (mobile hotspot)

7.2 Quality Checkpoints During Hackathon

☐ ASR component functioning independently (Hour 3)
☐ MT component functioning independently (Hour 3)
☐ TTS component functioning independently (Hour 5)
☐ End-to-end pipeline integration (Hour 9)
☐ Basic UI implementation (Hour 15)
☐ Multi-language support verified (Hour 21)
☐ Performance optimization complete (Hour 27)
☐ Final testing complete (Hour 33)

7.3 Common Issues & Mitigations

Potential Issue	Mitigation Strategy
Model download issues	Pre-download all models before hackathon
High latency	Prepare smaller model alternatives, optimize batch size
Memory limitations	Test memory usage beforehand, have quantized versions ready
Audio quality issues	Implement noise filtering, test microphones beforehand
Integration bugs	Build and test skeleton pipeline in advance

8. Resources & References

8.1 Documentation

- Hugging Face Transformers Docs
- PyTorch Audio Documentation
- Gradio Documentation

8.2 Model References

- Whisper Models
- M2M-100 Documentation
- SpeechT5 Models

8.3 Dataset References

- Common Voice Dataset

- CoVoST Dataset

Final Notes

This document provides a comprehensive plan for your hackathon project. Remember that in a 36-hour hackathon, scope management is critical. Focus on getting a working prototype before adding features or optimizations. Have a minimal viable product ready early, then enhance it as time permits.

Good luck with your hackathon!