Final Project Report

Voice Bridge-Peer to Peer Real Time Speech Translation (HackAZ)

Tanishk Singh, Kanit Mann

University of Arizona

INFO 511 SP25 002: Foundations of Data Science

Dr. Angela Cruz

May 12th, 2025

Project Question and Purpose

Artificial Intelligence (AI) advancements have led to breakthroughs in various domains of science. Through our project, we wanted to learn as well as explore the possibilities and challenges of building a low resource utilizing real time speech translation system using state of the art AI models across various languages thus experimenting with the efficiency of AI in the field of linguistic speech.

Being international students, we often felt the need to connect with people in their own language. The purpose of the project arises from its highly practical use case of helping people foster connection without the language or economic barrier of using high end expensive devices.

Methods and Analysis

System Architecture and Model Selection

While selecting our models, the goal was to prioritize responsiveness while maintaining adequate quality for effective communication, keeping in mind the limited resources we had for running these:

Speech Capture: Audio is captured via a web-based interface using the device's microphone.

- **Automatic Speech Recognition (ASR):** We chose Open AI's **Whisper** to convert spoken language into text due to its multilingual capabilities (OpenAI, 2022). For our implementation, we used the "tiny" model variant during development to speed up iteration. (See Appendix C, Figure C1 for more information)
- **Machine Translation (MT):** Employs Meta's NLLB-200 model in a streaming translation setup to handle short audio chunks while preserving context. We selected NLLB-200 (No Language Left Behind) because it supports 200+ languages. We used the **600M parameter** version to balance quality, speed and resource management. (Abdelrahman et al., 2022)

- **Text-to-Speech (TTS):** Converts translated text back to speech using Meta's MMS-TTS model. We used this model as it provides natural-sounding speech in multiple languages. (Meta AI, 2022a)
- **Real-Time Communication:** A Flask server with Socket.IO handles peer-to-peer connections (Socket.IO, 2023), ensuring data flows seamlessly between users. (See Appendix A Figure A1 for Visual Representation of Pipeline)

Techniques/Methods:

- Our system's core innovation lies in its streaming architecture that processes audio in small chunks with strategic overlaps. As shown in our implementation (see Appendix C Figure C2), we use a **multi-threaded approach** with dedicated worker threads for each component.
- Incremental translation: Instead of waiting for complete sentences, our "StreamingTranslator "processes text as it arrives. (See Appendix C Figure C3 for implementation)
- The system supports translation between 20+ languages through a **flexible mapping system** (language_utils.py) that handles conversions between different model-specific language codes.
- Audio Data Handling: Audio is processed in-memory and not persistently stored, reducing **privacy risks**. (See Appendix C Figure C4 for implementation)
- Signal Processing: The audio handling includes mathematical operations for signal processing.
- Voice Activity Detection: **Statistical** thresholds are used for speech detection. (See Appendix C Figure C5 for implementation)

Interfaces:

- Python (Flask, Socket.IO) is used to create the real-time server. Our UI and diagnostic tool are built with Gradio (Gradio Team, 2023) and audio processing via the SoundDevice library (SoundDevice Team, 2023) ensures each module works as expected. (See Appendix C Figure C6 for implementation)

Results

Through systematic testing and optimization, we were able to answer our research questions by creating a system that achieves near real-time translation while maintaining adequate speech quality.

- **System Performance**: The implemented translation pipeline achieved an average end-to-end latency of under 3 seconds across most language pairs, measured from the time speech is captured to when translated audio begins playing. (See Appendix B Figure B1). The system was able to process approximately 70-80% speech input accurately, with primary challenges occurring in noisy environments or with heavily accented speech.
- **Latency Optimisation**: Our overlapping chunk processing approach proved highly effective in reducing perceived latency. By processing 1-second audio chunks with 0.25-second overlaps, we maintained context while enabling faster translation. (See Appendix B Figure B2).

Future Research

This project could be further enhanced by building a native mobile application which can be operated offline with downloadable models and wireless communication using Bluetooth. The scope of enhancements is also huge as the model needs to work efficiently in noisy environments and with strong accents where it lacks at present. Research into more efficient model would enable deployment on resource-constrained devices without sacrificing quality.

Conclusion

The project establishes that real-time speech-to-speech translation is achievable with current technologies. We achieved average end-to-end latency under 3 seconds, fast enough to support natural conversation. While our system demonstrates the feasibility of peer-to-peer translation, it also reveals opportunities for further research, particularly in noise filtration and voice preservation. This project builds the foundation for more sophisticated translation systems that preserve the nuances and personal qualities of human conversation across linguistic divides.

References

1. Abdelrahman, A. A., Dutta, C., Li, Z., Andreas, J., & Rush, A. M. (2022). NLLB-200: No language left behind. *arXiv preprint arXiv:2207.04672*. https://doi.org/10.48550/arXiv.2207.04672
2. Gradio Team. (2023). Gradio: *Create UIs for your machine learning model in Python in 3 minutes*. GitHub Repository. https://github.com/gradio-app/gradio
3. SoundDevice Team. (2023). python-sounddevice*: Play and record sound with Python. GitHub Repository*. https://github.com/spatialaudio/python-sounddevice
4. Meta AI. (2022). *MMS-TTS: End-to-end text-to-speech synthesis*. GitHub. https://github.com/facebook/fairseq/tree/main/examples/mms
5. Meta AI. (2022). *NLLB-200: Open-source machine translation*. GitHub. https://github.com/facebookresearch/fairseq/tree/nllb
6. OpenAI. (2022). *Whisper: Robust speech recognition via large-scale weak supervision*. GitHub. https://github.com/openai/whisper
7. Socket.IO. (2023). *Socket.IO: Bidirectional and low-latency communication for every platform*. https://socket.io/
8. Singh, T., & Mann, K. (2025). *hackaz_team_wildhackers*. GitHub. https://github.com/kanitmann01/hackaz_team_wildhackers

Appendix A

Visualisations and Sytem Architecture

The following figures illustrates various stages in system architecture and their use cases.

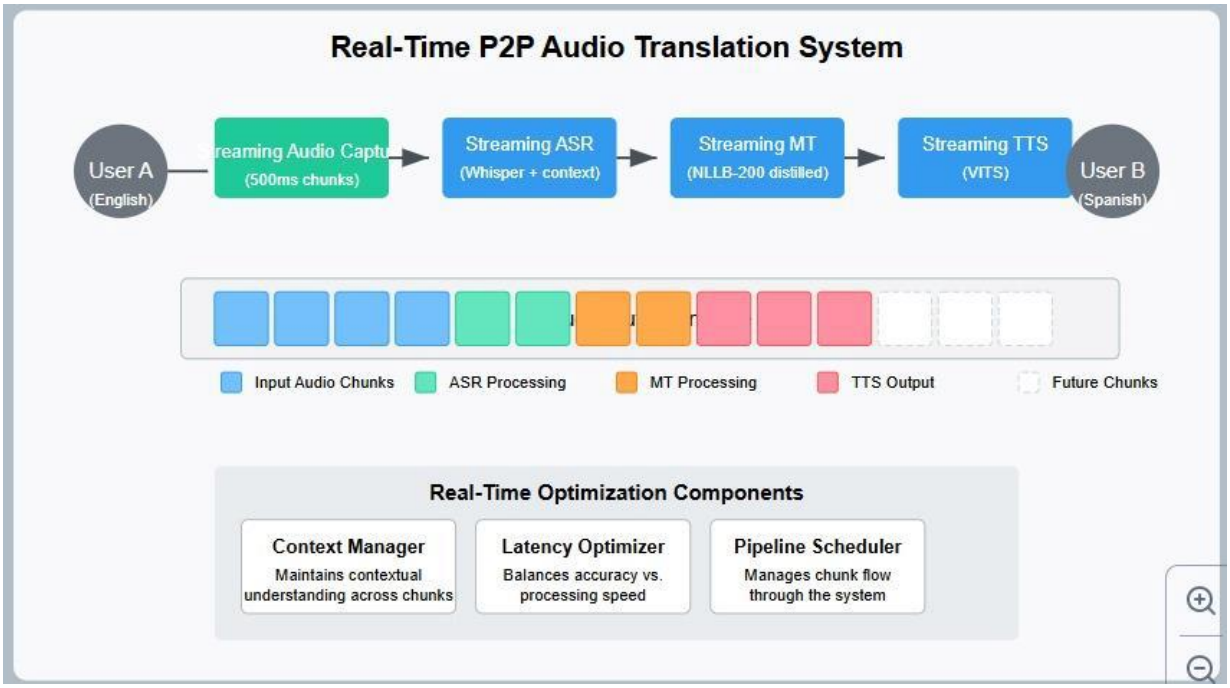

Figure A1. Pipeline and Architecture.

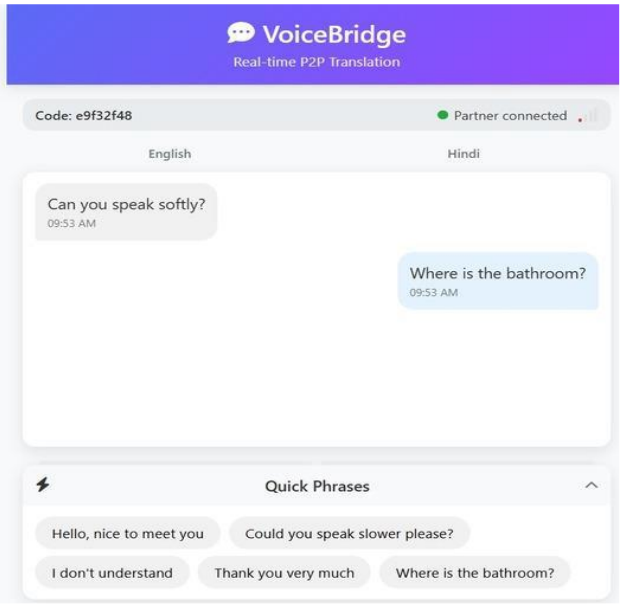

Figure A2. Live translational screen from UI

Appendix B

The follwing figures are related to result statistics

```python
def get_stats(self):
    """
    Get pipeline performance statistics.

    Returns:
        dict: Performance statistics
    """
    with self.lock:
        stats_copy = self.stats.copy()

        # Calculate averages
        if stats_copy["tts_chunks_processed"] > 0:
            stats_copy["avg_latency"] = stats_copy["total_latency"] / stats_copy["tts_chunks_processed"]
        else:
            stats_copy["avg_latency"] = 0

        # Add component-specific stats
        stats_copy["asr_stats"] = self.asr.get_stats()
        stats_copy["mt_stats"] = self.translator.get_stats()
        stats_copy["tts_stats"] = self.tts.get_stats()

        # Add uptime
        if self.start_time:
            stats_copy["uptime"] = time.time() - self.start_time
        else:
            stats_copy["uptime"] = 0

        return stats_copy
```

Figure B1. The get_stats function calculates average latency.

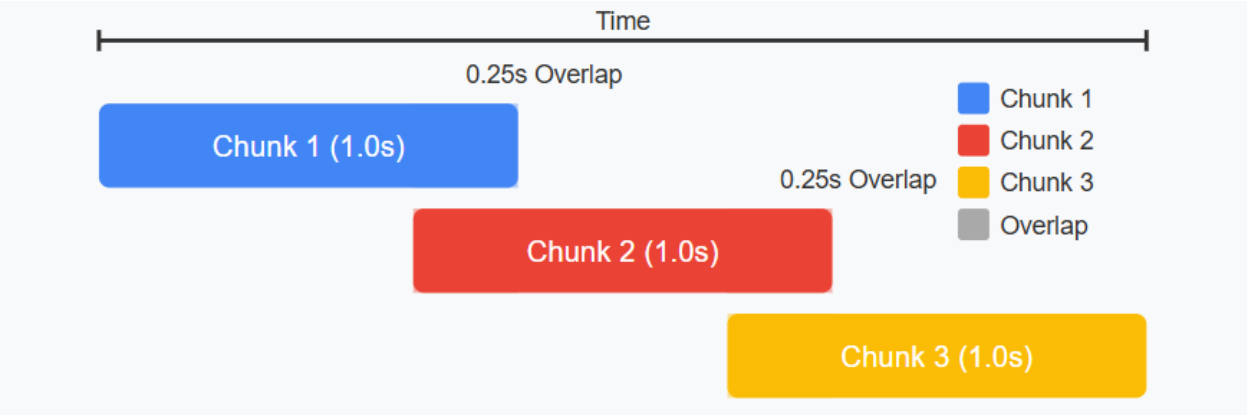

Figure B2. Overlapping chunks approach for maintaining coninuity.

Appendix C

The following figures represents code snippets from our project to better understand the implementation.

```
# Load model and processor
print(f"Loading Whisper model '{model_name}' on {self.device}...")
self.processor = WhisperProcessor.from_pretrained(model_name)
self.model = WhisperForConditionalGeneration.from_pretrained(model_name).to(self.device)

# Configure for better streaming performance
if self.device.type == 'cuda':
    self.model = self.model.half()  # Half-precision for faster processing
print("Whisper model loaded successfully!")
```

Figure C1. Loading whisper model on “Cuda” device if available.

```
# Start background workers
self.asr_worker = Thread(
    target=self._asr_worker_thread,
    daemon=True,
    name="ASR-Worker"
)
self.asr_worker.start()

self.mt_worker = Thread(
    target=self._mt_worker_thread,
    daemon=True,
    name="MT-Worker"
)
self.mt_worker.start()

self.tts_worker = Thread(
    target=self._tts_worker_thread,
    daemon=True,
    name="TTS-Worker"
)
self.tts_worker.start()
```

Figure C2. Using threads to handle different componenents for parallal processing.

```
# Update source buffer
complete_source = self.source_buffer + " " + text_chunk if self.source_buffer else text_chunk
complete_source = complete_source.strip()
self.source_buffer = complete_source

# Skip translation if chunk is too small (unless forced)
words = complete_source.split()
is_end_of_sentence = any(p in text_chunk for p in ['.', '!', '?', '。', '！', '？'])
if len(words) < self.min_chunk_size and not force and not is_end_of_sentence:
    return {
        'text': '',
        'full_text': self.translated_buffer,
        'processing_time': 0
    }
```

Figure C3. Maintaining buffer and prcessing chunks rather than waiting for whole sentence.

```python
def process_queue(self):
    """
    Process audio chunks from the queue and call the user callback.
    """
    while self.running:
        try:
            # Get chunk from queue with reduced timeout for faster response
            chunk = self.audio_queue.get(timeout=0.2)

            # Normalize audio level for consistent processing
            if np.max(np.abs(chunk)) > 0.0:
                normalized_chunk = chunk / np.max(np.abs(chunk)) * 0.9
            else:
                normalized_chunk = chunk

            # Call user callback with the audio chunk
            self.callback(normalized_chunk, self.sample_rate)

            # Mark task as done
            self.audio_queue.task_done()

        except queue.Empty:
            # Queue timeout, continue loop
            continue
        except Exception as e:
            self.last_error = str(e)
            print(f"Error processing audio chunk: {e}")
```

Figure C4. Chunks are processed in memory and passed directly to the callback without being permanently stored.

```python
# Basic Voice Activity Detection
active_frames = np.sum(np.abs(current_data) > self.vad_threshold)
active_ratio = active_frames / len(current_data)

# More detailed audio diagnostics
self.total_chunks_captured += 1
if active_ratio > self.min_active_ratio:
    self.chunks_with_activity += 1
```

Figure C5. Setting threshold to detect voice and filter out noise. Here vad_threshold = 0.003

```python
def create_ui():
    """Create the diagnostic UI."""
    with gr.Blocks(title="Translation System Diagnostics") as demo:
        gr.Markdown("# Translation System Component Diagnostics")
        gr.Markdown("Test each component individually to diagnose issues")

        # Console output
        with gr.Row():
            console_log = gr.Textbox(
                label="Console Log",
                value="",
                lines=20,
                max_lines=100,
                interactive=False
            )

        with gr.Tabs():
            # ===== AUDIO CAPTURE TAB =====
            with gr.TabItem("Audio Capture"):
                gr.Markdown("## Audio Capture Testing")
```

Figure C6. Diagnostic tool, gradio is a python framework for building quick UI tools and integrate ML models into them.

Generative AI Tool Acknolwdgement

We acknolwdge that during the span of HackArizona (March 22-23, 2025), we used Generative AI to work on our project. Generative AI proved to be a very helpful tool in building our project's website, establishing testing code and debugging our code base. It also helped us build our pipelines and improve latency by including features like context chunks and half precison. We used Generative AI through the platform provided by AI Core at The University of Arizona : https://chat.aicore.arizona.edu/. Specifically we used Anthropy's Claude AI 3.7 Sonnet Model which is avilable on the AI Core platform.

Below are some of the prompts we used during our hackathon:

- What are the best open-source models for speech recognition, machine translation, and text-to-speech that I could integrate for a hackathon project?
- Write a Python class for streaming ASR using Whisper that can process audio chunks incrementally rather than waiting for complete utterances.
- I need to implement incremental text translation using NLLB-200. Show me how to maintain translation context between chunks for more coherent output.
- How can I create a multi-threaded pipeline in Python that connects ASR, MT, and TTS components with queues for efficient real-time processing?
- Help me design a diagnostic tool with Gradio that can test each component of my translation pipeline individually.
- What techniques can I use to reduce latency in my speech-to-speech translation pipeline? I need to make it as real-time as possible.
- My Whisper ASR component is giving empty transcriptions for short audio chunks. How can I diagnose and fix this issue?

Researcher Bio-Sketch

Here we are introducing the people invoilved in the project, their contributions in the project along with their research interests.

Tanishk Singh : A first-semester graduate student pursuing a Master's degree in Data Science. For this project, Kanit led the development of the Automatic Speech Recognition (ASR) component and the audio I/O system. He implemented the real-time audio capture and processing pipeline, optimized the Whisper ASR integration for streaming capabilities, managed the diagnostic testing framework and optimizing model performance. Tanishk's future research interests involves using NLP for Scientific Exploration, utilisng data science to understand natural phenomenas and using AI to understand systems to make them more resilient and sustainable.

Kanit Mann : A first smester graduate student of Master's in Data Science. In this project, Kanit was responsible for the Machine Translation (MT) and Text-to-Speech (TTS) components, as well as the user interface development. He implemented the streaming translation pipeline using the NLLB-200 model, integrated the MMS-TTS synthesis system, and designed the Gradio-based user interface.. Kanit's also played a crucial role in prompt engineering, structuring the pipleine and debugging across various modules.

Peer Review Response

As part of in-class peer review by Mangalam Srivastav on April 8th, 2025. We worked on improving accuracy of our model by reducing noise filteration. We tried to achieve this by making minor changes like extending context duration (seconds of audio context to keep between chunk), however not much change in accuracy was observed. We are planning to deep dive into these issues and introduce major changes like implementing frequency band-specific processing to better distinguish speech from noise. This can be achieved using SciPy's signal processing library to create and apply a bandpass filter that only allows frequencies between the specified lowcut and highcut values to pass through. This is something we need to learn and research more about and is part of future efforts.