

Project Documentation: Context-Aware Conversation Support with Augmented Reality and Generative AI

Team Members:

- Sabrina Alvarado
- Keren Rivera
- Jonathan Martinez
- Peace Passos
- Ana Morales

Mentor: Dr. Chen Chen, FIU KFSCIS, Assistant Professor

Instructor: Seyedmasoud Sadjadi

Project Overview

This project develops a mixed-reality conversation support system that provides real-time translation, concept clarification, and AR visual overlays on the Meta Quest 2 headset. The system integrates Speech-To-Text (STT), Gemini translation/explanation, Text-To-Speech (TTS), and Unity-based AR overlays, with a FastAPI backend that coordinates all modules.

Our goal is to reduce conversational barriers caused by language differences and uneven background knowledge, enabling smoother, more inclusive communication in multilingual and cross-disciplinary contexts.

Executive Summary

Our project addresses a common barrier in multilingual and cross-disciplinary communication: participants often struggle with unfamiliar terminology, differences in language proficiency, or gaps in background knowledge. These challenges lead to hesitation, misunderstandings, or reduced engagement.

We proposed an AI-driven mixed-reality system that supports conversation in real time through speech recognition, English and Spanish translation context-aware explanations, and augmented-reality overlays rendered in the Meta Quest 2's Passthrough environment. Using the Gemini API, the system provides both translation and concept clarification, while AR visuals help reinforce understanding.

Initial results demonstrate smooth end-to-end processing, clear translations, readable AR overlays, and real-time feedback. This system shows promise for multilingual education, collaborative STEM work, and accessibility use cases.

Problem & Requirements (O1)

Problem Statement

Effective communication requires shared language and shared background knowledge. In multilingual or cross-disciplinary environments, participants may not understand key terms, domain-specific concepts, slang, cultural references, or technical vocabulary. These gaps create:

- Misunderstandings
- Awkward pauses
- Lower participation

We aim to build a system that supports real-time, natural conversation without requiring users to pause and manually translate or look up definitions.

Context & Stakeholders

- University students
- Multilingual teams
- Educators and classroom environments
- Cross-disciplinary collaboration groups
- Visitors, tourists, or individuals with limited English/Spanish proficiency

Functional Requirements

- Speech-To-Text transcription
- Real-time translation (English <-> Spanish initially)
- Real-time concept clarification (definitions, summaries)
- AR text overlay in Passthrough
- Bidirectional communication between Quest 2 <-> FastAPI backend

Non-Functional Requirements

- Low latency
- High transcription accuracy
- Readable AR placement and typography

- Secure API communication

Constraints

- On-device limitations of Quest 2
- Network dependency for Gemini API

Success Criteria

- Stable end-to-end pipeline
- Accurate translation and clarification
- Smooth AR rendering with minimal delay

Prioritized Backlog

1. STT <-> Backend pipeline
2. Translation module (Gemini)
3. Concept clarification module
4. TTS output
5. AR overlay rendering
6. Language detection
7. UI/UX refinements
8. Error handling & fallback behavior
9. Data formatting
10. Logging and performance metrics

System Overview & Architecture (O2)

System Summary

The system combines Speech-To-Text (STT), Gemini-powered translation and concept clarification, Text-To-Speech (TTS), and AR overlays in Unity to provide real-time, context-aware conversational on the Meta Quest 2.

The pipeline:

1. User speaks -> STT converts speech to text
2. Backend receives text and detects the language
3. Gemini API generates either (a) translation or (b) simplified explanation
4. Backend returns processed text
5. Meta Quest renders AR overlay + plays TTS audio

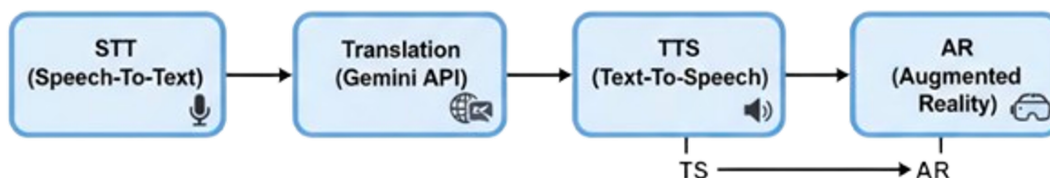
Architecture Diagram

- **Meta Quest 2 Headset**
 - Unity APP
 - Passthrough AR
 - HTTPS client (send/receive JSON)
- **FastAPI Backend**
 - STT endpoint
 - Translation + Explanation endpoint (Gemini)
 - TTS endpoint
 - Routing + orchestration
- **Gemini API**
 - Translation model
 - Context clarification (definitions, summaries)
- **Audio + AR Output**
 - TTS audio playback
 - AR text overlays anchored in Passthrough

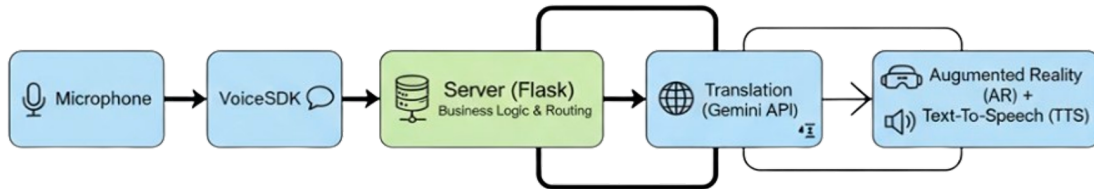
Key Technologies

- Meta Quest 2 (Unity, Meta XR SDK)
- Unity UI Toolkit for overlays
- FastAPI backend
- Gemini 2.0 API (translation + explanation)
- Google Speech-To-Text
- HTTP/HTTPS JSON messaging
- Android Build Pipeline for Quest deployment

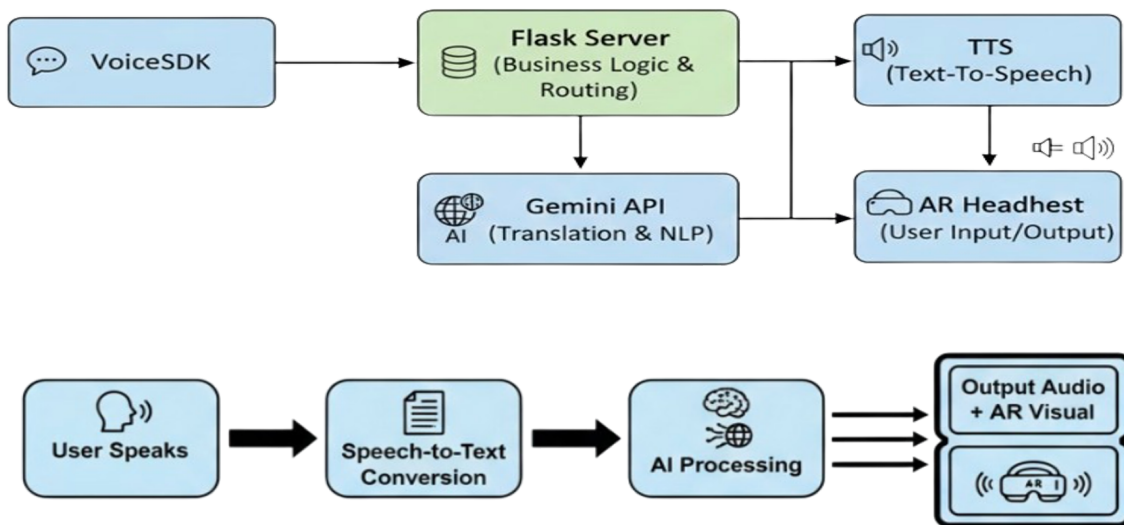
Proposed Solution



Core Capabilities



System Architecture



Discipline-Specific Depth (O6)

Algorithms & Data Structures

- Language Detection: lightweight rule-based + model-assisted method to switch between English and Spanish
- Translation/Explanation: Optimized Gemini prompt templates for fast responses
- AR Text Placement: Unity world-space canvases anchored to head pose for stable overlays

Architecture & Patterns

- Modular Backend: STT, translation, clarification, and TTS run as separate FastAPI endpoints
- Producer Consumer Model: Headset sends text; backend returns processed output
- Async Processing: FastAPI async routes reduce blocking and improve latency

Engineering Evidence

- AR overlay readability
- STT accuracy tests in quiet vs. noisy settings

Implementation Notes (O2)

Flask Server:

```

1  import flask
2  from flask import app
3
4  text = "Hello, Flask!"
5
6
7
8  ✓ def main() -> None:
9      app = flask.Flask(__name__)
10
11     @app.route("/", methods=["POST"])
12     def home():
13         global text
14         text = flask.request.get_data().decode("utf-8")
15         return text
16
17     @app.route("/", methods=["GET"])
18     def index():
19         return text
20
21     app.run(host="0.0.0.0", port=5000)
22
23 if __name__ == "__main__":
24     main()

```

Translations and Images

```

1  from os import getenv, environ
2  import google.generativeai as genai
3  from fastapi import FastAPI
4  from fastapi.responses import JSONResponse, RedirectResponse
5  from googleapiclient.discovery import build
6  from pydantic import BaseModel
7  from google.cloud import translate
8
9  # --- Configure Gemini ---
10 environ["GOOGLE_API_KEY"] = "AIzaSyBrVcLSjKSkGwOWtIUIt1UwGbFaZgpGtM"
11 genai.configure(api_key=getenv("GOOGLE_API_KEY"))
12
13 # Custom search engine ID
14 CX_SEARCH_ENGINE = "8773d847969914505"
15
16 app = FastAPI(title="Bilingual Translation & Food Image API", version="1.0")
17
18 class TranslationRequest(BaseModel):
19     text: str
20     target_language: str
21
22 class GeneralTranslationRequest(BaseModel):
23     text: str
24     source_language_code: str
25     target_language_code: str
26
27 @app.post("/general-translation")
28 ✓ def general_translation(request: GeneralTranslationRequest):
29     translated_text = translate_text(request.text, request.source_language_code, request.target_language_code)
30
31     return {
32         "translated_text": translated_text
33     }
34
35
36
37 @app.post("/translate-with-image")
38 ✓ def translate_with_image(request: TranslationRequest):
39     # 1. Translate the text
40     # Getting the language
41     target_language = request.target_language.title()
42
43     # The gemini model to translate the sentence/input
44     model = genai.GenerativeModel("models/gemini-2.5-flash")
45
46     # Prompt for gemini to get the translation
47     prompt = (
48         f"Translate this text into {target_language}. "
49         f"Text: {request.text}"
50     )
51
52     try:
53         # What gemini gives us based on the prompt
54         response = model.generate_content(prompt)
55
56         # Formatted translation
57         result = getattr(response, "text", None)
58
59         # If something goes wrong with translation, here it will show us what it was able to get
60         if not result:

```

```

60         if not result:
61             return JSONResponse(status_code=500, content={"error": "No translation returned from Gemini.", "raw_response": str(response)})
62
63         # Cleaning up the response
64         translation = result.strip()
65
66         # Should something go wrong, it will show us what the error is
67     except Exception as e:
68         return JSONResponse(status_code=500, content={"error": f"Translation error: {str(e)}"})
69
70     # 2 Getting general description of the text
71     descPrompt = (f"Provide a brief description of the following text in {target_language}."
72                  f"If it includes idioms or culturally specific expressions, explain briefly in two or three sentences. "
73                  f"Keep the translation natural and conversational.\n\n"
74                  f"Text: {request.text}")
75
76
77     try:
78         # The response from gemini based on the description prompt
79         descResponse = model.generate_content(descPrompt)
80         descResult = getattr(descResponse, "text", None)
81         if not descResult:
82             return JSONResponse(status_code=500, content={"error": "No description returned from Gemini.", "raw_response": str(descResponse)})
83
84         # Cleaning up the description response
85         description = descResult.strip()
86
87     except Exception as e:
88         return JSONResponse(status_code=500, content={"error": f"Description error: {str(e)}"})
89

```

```

90     ### 3 conversational response suggestions
91     # # maxSuggestions = 4
92     # # convoSuggestionsPrompt = (f"Suggest at least 1 or at most {maxSuggestions} conversational responses in {target_language} based on the followi
93
94     # # try:
95     # #     convoResponse = model.generate_content(convoSuggestionsPrompt)
96     # #     convoResult = getattr(convoResponse, "text", None)
97     # #     if not convoResult:
98     # #         return JSONResponse(status_code=500, content={"error": "No conversational suggestions returned from Gemini.", "raw_response": str(con
99
100     # #     # Cleaning up the conversational suggestions response
101     # #     convoCleanResult = convoResult.split('\n')
102     # #     convoSuggestions = [suggestion.strip() for suggestion in convoCleanResult.getlines() if suggestion.strip()]
103
104     # except Exception as e:
105     #     return JSONResponse(status_code=500, content={"error": f"Conversational suggestions error: {str(e)}"})
106
107     # 4. Get the image from Google search
108     # Number of max images to return
109     maxImgsNum = 3
110
111     # The "custom" search engine aka just a loophole to google search images hence the google api key being here
112     service = build("customsearch", "v1", developerKey="AIzaSyBrVcLSjKSkGwOWtIUIt1UwGbFfaZgpGtM")
113
114     # The result of the custom search bar. I begins by looking at the input. It uses the custom search engine
115     res = service.cse().list(q=request.text, cx=CX_SEARCH_ENGINE, searchType="image", num=maxImgsNum).execute()
116
117     # The link to each image
118     image_urls = [item["link"] for item in res.get("items", [])]
119

```



```

120     # The output we get
121     return {
122         "original_text": request.text,
123         "translation": translation,
124         "description": description,
125         "image_url": image_urls,
126     }
127
128 ✓ def translate_text(text="Hello, world!", source_langauge_code = "en-US", target_language_code = "es", project_id="mimetic-sunset-479220-f9"):
129
130     client = translate.TranslationServiceClient()
131     location = "global"
132     parent = f"projects/{project_id}/locations/{location}"
133     response = client.translate_text(
134         request={
135             "parent": parent,
136             "contents": [text],
137             "mime_type": "text/plain",
138             "source_language_code": source_langauge_code,
139             "target_language_code": target_language_code,
140         }
141     )
142
143     for translation in response.translations:
144         print("Translated text: {}".format(translation.translated_text))
145     return translation.translated_text
146     return None
147
148
149 @app.get("/")
150 def root():
151     return RedirectResponse(url="/docs")
152
153 if __name__ == "__main__":
154     environ["GOOGLE_APPLICATION_CREDENTIALS"] = "mimetic-sunset-479220-f9-868f285745aa.json"
155     import uvicorn
156     uvicorn.run(app, host="0.0.0.0", port=8000)

```

Unity Path on GitHub

- Includes assets, packages, and project settings

<https://github.com/lxD-Lab-FIU/capstone-2-fall-25/tree/66d0c1737ae4c61e5c0bd123d0580906b77e4075/unity>

GenAI Branch

<https://github.com/lxD-Lab-FIU/capstone-2-fall-25/tree/GenAi>

Testing & Evaluation (O3)

Testing Strategy

- Unit tests:
 - API endpoints
 - JSON schema validation
 - Prompt formatting
- Integration tests:
 - End-to-end STT -> backend -> Gemini -> TTS

- AR overlay rendering
- Performance tests:
 - Latency measurement at each top
 - FPS maintenance in AR view
- User Evaluation:
 - Early testers rated clarity and usefulness of AR overlays
 - Translation accuracy validated across sample dialogues

Security, Privacy, Accessibility & Compliance (O5)

Security

- All communication uses HTTPS
- No audio or text is stored
- Backend validates all incoming requests

Privacy

- No user data is retained
- Speech context is never logged
- Gemini API requests avoid persistent identifiers

Accessibility

- AR overlays designed with high contrast

Ethical Considerations

- Careful handling of cultural and colloquial language
- Prioritize inclusive support for multilingual users

Deployment & Operations

Deployment Workflow

1. Unity build -> Android APK
2. Install APK on Meta Quest 2
3. Launch FastAPI backend
4. Configure Quest app with backend URL
5. System ready for speech stream and AR display

Operations

- Backend supports

- Unity log accessible via Quest Developer Hub
- Monitoring includes API latency and backend uptime

Limitations & Future Work

Current Limitations

- Limited language support (English/Spanish only)
- Relies on cloud APIs
- AR overlay layout is simplistic
- Concept explanations may vary with prompt phrasing

Future Work

- More languages and dialects
- Improve AR visualization (icons, diagrams, animations)
- On-device translation for lower latency
- Emotion or intent detection
- User-defined vocabulary or glossary

References

1. Google Gemini API Documentation
2. Meta Quest Developer Documentation
3. Unity XR Toolkit Guide
4. FastAPI Documentation