SAPIENZA
UNIVERSITÀ DI ROMA

DEPARTMENT OF COMPUTER SCIENCE

Multimodal Sign Language Translator

MULTIMODAL INTERACTION

Professor:
Maria De Marsico

Students:
Federico Barreca, 1736423
Giuseppe Bello, 2053788
Ilaria De Sio, 2064970
Matteo Mortella, 2058262
Daniele Solombrino, 1743111

Academic Year 2023/2024

1 Abstract

Sign Language (SL) users currently lack SL-to-SL translation apps and don't have technology-assisted tools for practicing learned gestures.
In the last few years, **Deep Learning** (DL), a sub-field of Artificial Intelligence (AI), has shown promising results in solving a very broad range of tasks, particularly in language, speech, and visual domains.
With this in mind, and to make **SL-to-SL translation** available for the broadest group of people possible, a deep learning-based solution that follows principles of Multimodal Interactions has been thought of.
Multimodal Sign Language Translator (MMST) is a web app that allows users to input their sentences using speech, text, or SL in diverse available languages and to receive the corresponding translation in one of the aforementioned modalities in the desired language.
This solution represents the **first step** towards the realization of a complete SL-to-SL translator, although current state-of-the-art models are still far from ideal satisfactory performances, especially in Sign-to-* translation scenarios.

2 Report Structure

The paragraphs of this report closely follow the timeline of the design, development, implementation, and test stages of MMST.
Immediately after coming up with the idea of working with sign languages, a first **information gathering** process about the domain was done, as no one in the team had previous knowledge about the domain, this is documented in paragraph 3.
Knowledge gathered in paragraph 3 helped to lay down some high-level, plausible **applications** for MMST and their subsequent **use cases**, which are described in paragraph 4.
Once MMST goals were explicit, some notions were borrowed from the **Human-Computer Interaction** field, and some important decisions were based on accessibility, usability, and user experience. Several aspects seen in class during the lectures were tackled, including Hierarchical Task Analysis and **multimodality**'s type of relations, as reported in paragraph 5.
Paragraph 5 also gave clear perspectives about technical requirements, which led to system architecture design, **software implementation**, and testing, as exposed in paragraph 6.
Finally, a summary concludes the report with an indication of possible **future directions**.

3 Domain knowledge

None of the team members had prior experience with Sign Language, so the natural first step consisted of studying the Sign Language domain. Pretty much any spoken language has its corresponding Signed counterpart, which may not be unique. For example, American Sign Language (ASL) and British Sign Language (BSL) are signed versions of the same spoken language, English.

3.1 Collecting information

MMST started with a broad and generic idea in mind: help Sign Language users. But how to offer help considering the team's unfamiliarity with the domain? With a top-down, need-finding-oriented approach. A **questionnaire** [13] [14], was delivered to potential users and its results served for an analysis. Once studied the **concrete needs** of people directly or indirectly interested in the topic, a glance at **existing solutions** in the wild gave a sense of what is offered and what core elements to offer in an SL-to-SL application.

3.2 Online communities polls

The research journey led to places that include different kinds of users, either directly or indirectly in need of using Sign Languages, like deaf people, mute people, parents, educators, or collaborators of deaf/mute people and people familiar with assisting technologies, disabilities, and learning languages. Questionnaires were brought to the following online communities:

- r/LanguageLearning, r/LearningASL, r/learnASL
- r/AssistiveTechnology/, r/SideProject
- r/mute, r/HardOfHearing, r/Disability_Survey, r/AudiProcDisorder
- r/BSL
- thedeaf.soul Instagram page

3.3 Elements of the poll and results

The questionnaire had the goal of assessing:

1. Users' knowledge or experience with Sign Language-related **apps or services**.
2. **Limits** of current systems.
3. which aspects of current systems could be **improved** or implemented.

68 answers (57 from participants directly in need of Sign Language and 11 from indirect users) reveal that:

- $\sim$ 78% of users would like a Text-to-Sign translation for a **non-Anglophone** language
- $\sim$ 45% of them would like to have the service **integrated** into third-party apps
- $\sim$ 38% of users would like to be able to learn from senior interpreters

3.4 Existing solutions analysis

The most advanced applications, to take inspiration from existing features are the following:

1. Text-to-Sign: HandTalk, SpreadTheSign, Sign2Me.eu
2. Learning (sign languages): ASL Bloom, Melisegno, SignSchool
3. Learning (spoken languages): Duolingo, Babbel

Looking at translation apps, is true that none of them offers direct Sign-to-Sign translation capabilities, which is theoretically doable, as shown in [10].
Sign2Me [21] promises to offer Sign-to-Text and Text-to-Sign capabilities, but they can only be used on their own, independently, rather than together, which would emulate a Sign-to-Sign translation.
As far as learning apps are concerned, every solution offers nice features, like deduction-based learning (ASL Bloom [2]), multi-modal teaching (SignSchool [20]), or challenging activities (Duolingo [4]), but none of them exploit them all. Considering the strength points and drawbacks of such applications and their interfaces, the design includes an intuitive interface that supports different input and output modalities.

4 App Introduction

4.1 High-level view

MMST (Multimodal Sign Language Translator) is a **multimodal**, **polyglot**, and bidirectional web app capable of performing sign language translations, designed for a **wide and diverse range of users**, including individuals who are deaf or hard of hearing, sign language interpreters, educators, and anyone learning or communicating in sign language, as it can be used in various sign language-related scenarios, like everyday interactions and practicings.
The application supports **several languages** like Italian, English, and Spanish spoken languages, together with Italian Sign Language (Lingua dei segni italiani, LIS), American Sign Language (ASL), British Sign Language (BSL) and Spanish Sign Languages (Lengua de Signos Española, LSE). **Multiple input methods** are supported, users can enter sentences through text, speech, or video (SL), through a keyboard, a microphone, or a webcam respectively. Additionally, the app allows uploading pre-recorded audio or video and pasting text directly from the device. Upon receiving input in any chosen modality, MMST processes it and delivers translations in the **preferred output format**: text, audio, or video. This ensures communication across different languages and modes, accommodating a wide array of user needs and use cases. The app's interaction schema is resumed in Figure 1.

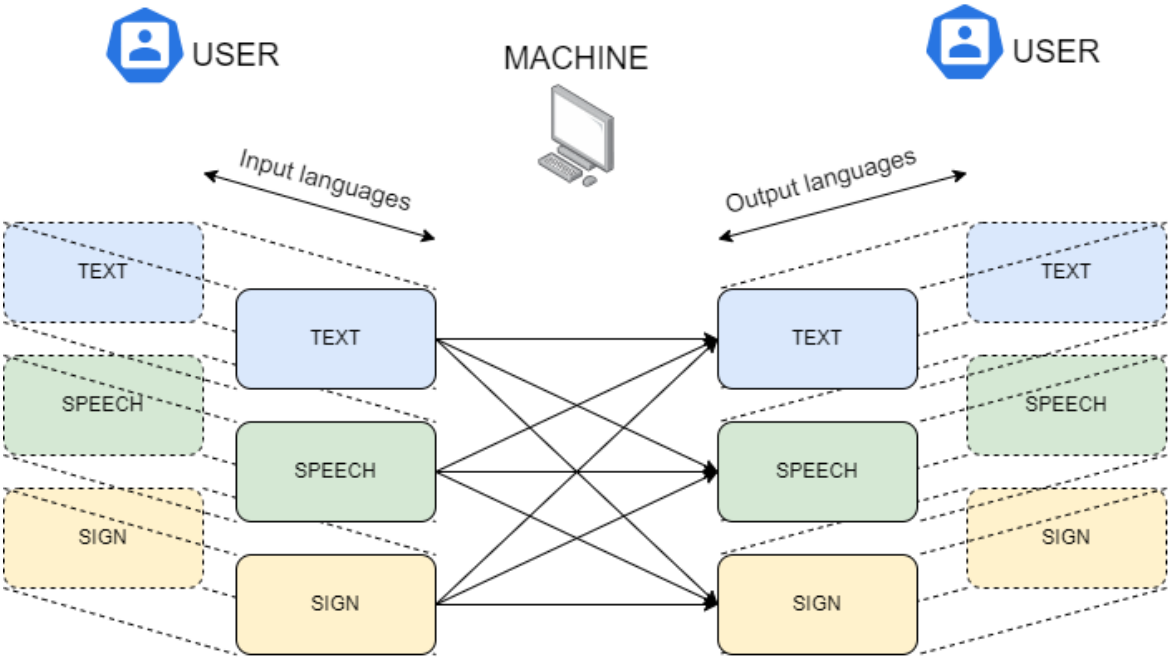

Figure 1: I/O pairs supported in MMST. Full resolution schema available here.

4.2 Applications

MMST has been designed and devised with various sign language-related scenarios in mind, including **everyday** personal and professional **interactions**. Learners can practice new sign languages, improve known ones, or test their level by comparing their signs to those produced by the app.
In everyday interactions, MMST creates a diverse range of communication bridges. It **facilitates conversations** between people who use spoken languages and those who use sign languages but also between individuals who use different sign languages.
One of the main advantages of MMST is the opportunity to **practice** and **learn** when a human teacher is not available.

4.3 Use cases

Some of the previously described practical applications can be associated with the following use cases.

Figure 2: Conceptual representation of a user practicing sign language using feedback from MMST

4.3.1 Use-case 1: Text-to-Sign translation

Use Case Name	Text-to-Sign Translation
Actors	• User: Someone learning a sign language
Precondition	• User wants to translate a text in a spoken language (e.g., English) into a sign language (e.g., British Sign Language). • User can type on a keyboard and understand the source spoken language to translate.
Main Flow	1. Users select a spoken language as the source and a signed language as the destination. 2. User types a sentence or pastes it in the source box. 3. User presses the "Translate" button. 4. MMST generates a video containing the target sign language translated from the text input
Postcondition	• User receives and views a video output containing the sign language translation of the written input. • Input text has been translated into the target sign language.

Table 1: Use Case: Text-to-Sign Translation

4.3.2 Use-case 2: Audio-to-Sign translation

Use Case Name	Audio-to-Sign Translation
Actors	• User: a deaf person who can perceive and understand sign language
Precondition	• The user wants to translate the audio track of a known spoken language (e.g., English) into a sign language (e.g., British Sign Language). • The user has the recording of a sentence in the source language to translate in its device.
Main Flow	1. User selects the spoken language to translate as the source and sign language as the target. 2. User uploads the audio file from its device. 3. User presses the "Translate" button. 4. MMST processes the input internally 5. MMST generates a video of the translated audio track to sign language 6. User can play the generated video
Postcondition	• User receives and views a video output containing the sign language translation of the spoken input. • Input speech has been translated into the selected sign language.

Table 2: Use Case: Audio-to-Sign Translation

4.3.3 Use-case 3: Sign-to-Text translation

Use Case Name	Sign-to-Text Translation
Actors	• User: Someone learning sign language
Precondition	• User wants to translate a sentence in sign language (e.g., American Sign Language, ASL) into a spoken language (e.g., Spanish) that the user can read and understand. • User has a video recording of a sentence in the source sign language to translate in its device.
Main Flow	1. User selects the sign language to translate as the source and spoken language as the target. 2. User uploads a video file containing a sign language sentence from its device. 3. User presses the "Translate" button. 4. MMST processes the input internally 5. MMST places the corresponding text in the dedicated translation area. 6. User can read the translation.
Postcondition	• User receives and views a text output containing the spoken language translation of the signed input. • Input text has been translated into the selected target language.

Table 3: Use Case: Sign-to-Text Translation

4.3.4 Use-case 4: Sign-to-Sign translation

Use Case Name	Sign-to-Text Translation
Actors	• User 1: Someone able to sign a sign language: SL1 • User 2: Someone able to sign a sign language: SL2
Precondition	• User 1 knows SL1, User 2 knows SL2 and SL1 is different from SL2. • User 1 and User 2 want to communicate using only their respective sign language
Main Flow	1. Users select their respective sign languages as source and target languages 2. User 1 records a sentence in SL1 using the webcam. 3. User presses the "Translate" button. 4. MMST processes the input internally 5. MMST produces the video containing the sentence recorded by User1 but expressed now in SL2 6. User 2 understands the message and can swap input and output languages to reply

Postcondition	• User 2 receives and views a video output containing the sign language translation of the signed input. • Input video has been translated into the selected output sign language.

Table 4: Use Case: Sign-to-Text Translation

4.3.5 Use-case 5: Text-to-Text translation

Use Case Name	Text-to-Text Translation
Actors	• User 1: Someone communicating with someone else who speaks another language. • User 2: Someone able to understand the destination spoken language
Precondition	• User 1 wants to translate a spoken language (e.g., English) into another spoken language (e.g., Italian). • User 1 can type or paste the source spoken language to translate using a keyboard. • User 2 can perceive and read the produced written text on the screen

Main Flow	1. Users select the source and the target spoken languages for the translation. 2. User 1 types a sentence or pastes it in the source box. 3. User 1 presses the "Translate" button. 4. MMST processes the input 5. MMST places the translated text in the dedicated translation area. 6. User 2 can read the output and eventually swap source and target languages to reply
Postcondition	• User 2 receives and views the translated text. • Input text has been translated into the selected sign language.

Table 5: Use Case: Text-to-Text Translation

4.4 Activity diagram

All the above MMST applications and use cases can be represented in the following activity diagram which also shows the system processes that are obscure from the user's perspective.

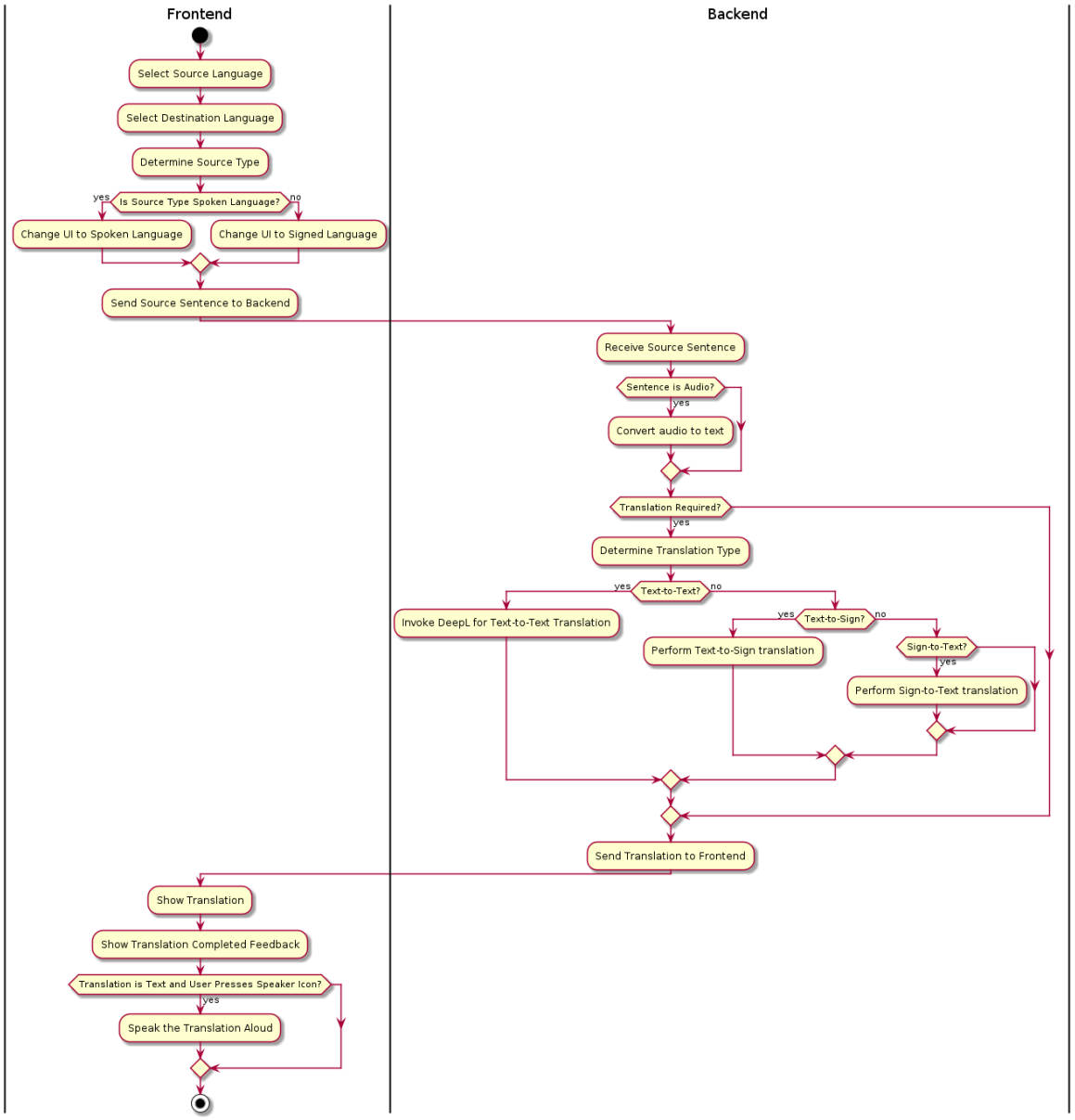

Figure 3: MMST activity diagram. Full resolution schema available here.

5 Theoretical modeling and design

5.1 Human-Computer Interaction

The Human-computer interaction (HCI) field studies the interaction between humans and computers and tries to design interfaces that **facilitate** these **exchanges**.
Figure 4 shows a schematic representation of all Human-Computer Interaction-related topics of MMST.

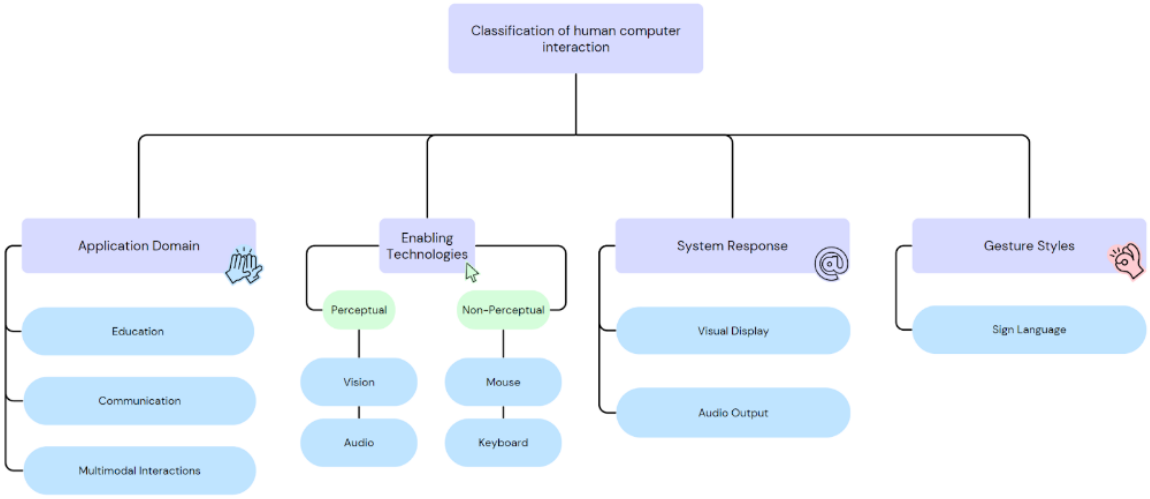

Figure 4: MMST Human-Computer Interaction-related topics. Full resolution schema available here.

H-C interfaces can be described, classified, and evaluated through the following aspects: usability, accessibility, and user experience (UX). The application's design aims to improve on such aspects.

5.1.1 Usability

Usability refers to how easy and efficient it is for users to accomplish their intentions. The first aspect to consider is **learnability**, which expresses how much effort is required for new users to use the interface; The project's interface is presented as a web page with self-explanatory buttons and a brief description of the steps to perform for getting the translation; A new user that is familiar with web browsers logics should be able to learn in **less than a couple of minutes** how to use the interface. The time required for a translation depends on the length of the input and the internet connection: some translations are performed with external tools and a **stable connection** must be ensured; for average length and internet speed, all the translations are produced in a matter of seconds.

5.1.2 Accessibility

It ensures that systems and interfaces can be used by **people with a wide range of abilities and disabilities**. The application, available in Italian, English, and Spanish languages, is designed for people without sense impairments, Speech-impaired people, Deaf or hard-of-hearing people to translate a sentence from their principal communication method to another desired one. The interface's design follows the indications of the **Web Content Accessibility Guidelines (WCAG)** published by the Web Accessibility Initiative, specifically, the **WCAG 2.2** version; The offered content, according to the four principles of accessibility is: perceivable, operable, understandable, and robust. The minimum **conformance level A** for each applicable content is reached. Particular attention is paid to text-background contrasts, that respect the 1.4.3 **Contrast requirement of AA level** so that the application content can be read by people with moderately low vision or impaired contrast perception, without the use of contrast-enhancing assistive technology. For checking the contrast WebAIM tool has been used.

5.1.3 User experience (UX)

The application presents an intuitive user interface with a few buttons, drop-down menus, an input text box, and **step-by-step instructions** on how to use the application. All the possible input errors are handled. After a translation is done it is possible to perform another one with the same modality or with a different one, to bring the input box to the default text status a reset button is available, this is mandatory for uploading a new video after a previously uploaded one. The application is offered on a web page to allow access with a common browser on diverse operating systems. A **good overall responsiveness** characterizes the application, the only time bottleneck could be caused by long translation or connection problems.

5.2 Hierarchical Task Analysis (HTA)

To design a simple and concise app's interface a hierarchical task analysis (HTA) has been conducted. HTA permits breaking down **goals** into single **tasks** that can be divided into sub-tasks on their own; This process shows how users can interact with the system and gives an intuition of the steps required to reach their goals. This creates an opportunity to improve or solve the interface's fallacies. The analysis is summarized by the schema in Figure 5 which shows the tasks to perform a translation and the **plans** to achieve it.

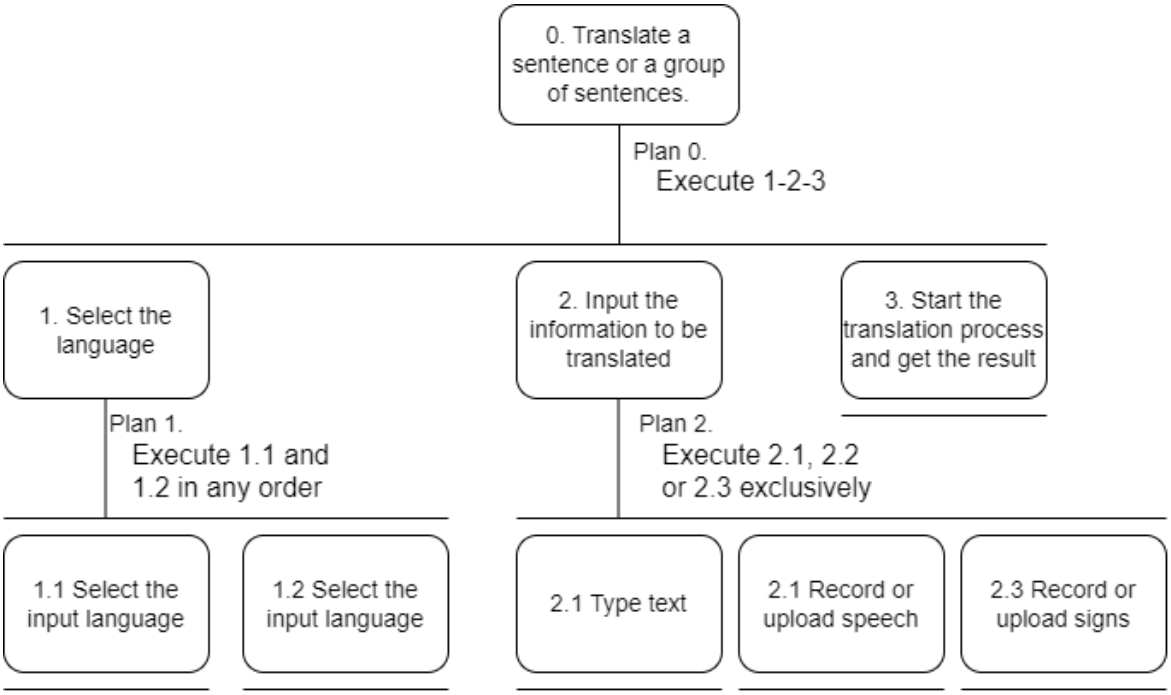

Figure 5: HTA schema. Full resolution schema available here.

5.3 Multimodality: Type of relations

MMST is a multimodal system as it employs speech, vision, and hearing to support a vast diverse pool of users. MMST supports "alternative" and "equivalent" types of multimodal relations, as defined by Bernsen and Dybkjær (2009) in [15] and Martin (1997) in [12]. These relations allow users to input and output the same semantic content through different interchangeable modalities, such as spoken language, written text, or sign language video. This flexibility aligns with the principle of **equivalence**, where different forms serve as **alternative** means to convey the same information.
Additionally, MMST also utilizes "fission", as defined by Foster (2002) [5], to distribute an abstract message across multiple output channels, providing spoken and visual feedback indicators, to signal that a translation has been completed.
However, the app does not currently support relations such as complementarity, addition, redundancy, elaboration, stand-in, substitution, conflict, specialization, and fusion. The primary reason for this is the focus on maintaining **semantic consistency** across modalities; since the same information is handled across all available formats, there is no need for complex relationships that differentiate or expand upon the content. For example, complementarity is not employed to combine different pieces of information from various sources, nor is redundancy used to reinforce messages through multiple simultaneous channels.

6 Practical modeling and design

6.1 Architecture structure

MMST is composed of a **frontend**, a **backend**, and an **API orchestration system**. The front end handles every UI aspect, including language selection, input modality selection, input sentence data gathering, and translation display.
The backend gets the data from the front end, applies the required preprocessing, calls translation APIs, and returns the result to the front end.

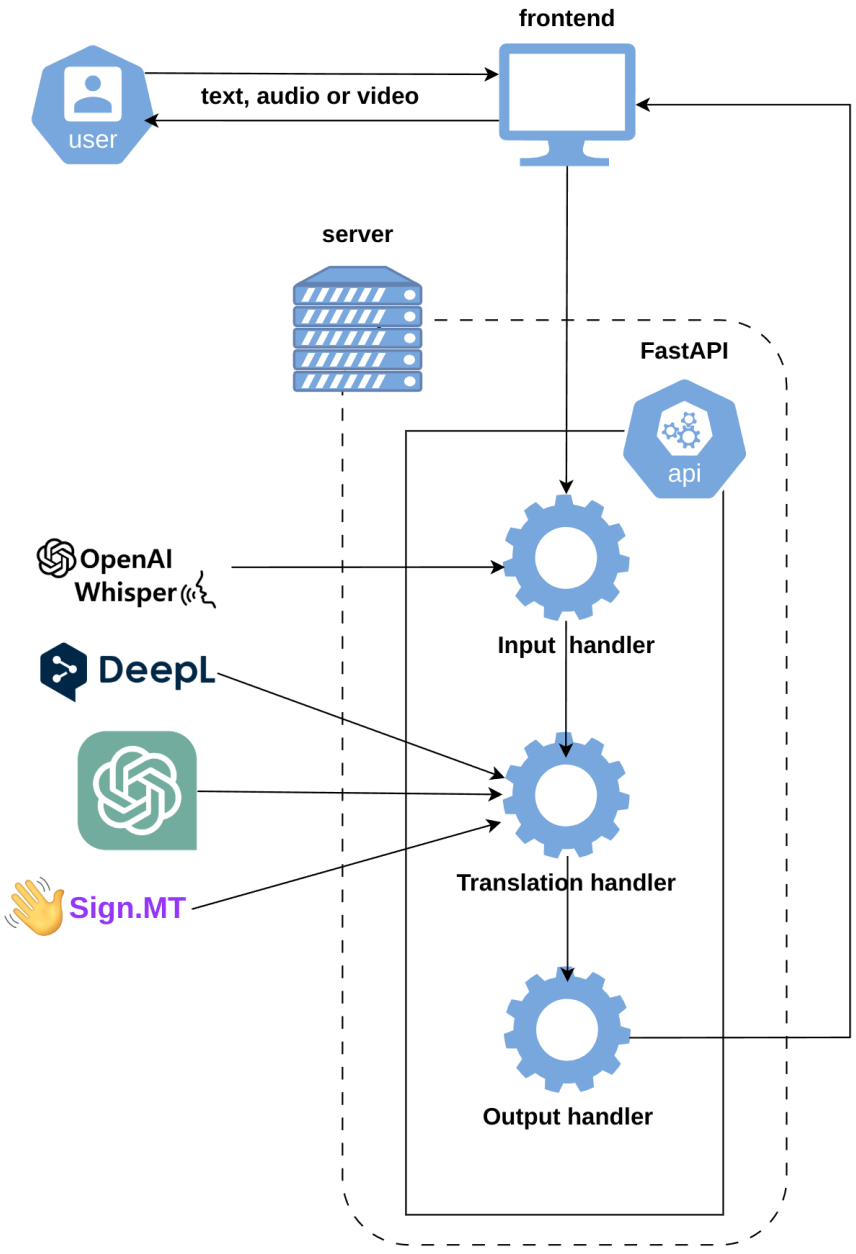

Figure 6: MMST architecture. Full resolution schema available here

6.2 Frontend

MMST frontend is written in **React JS**, a highly popular JavaScript library for building user interfaces and it provides an intuitive interface that allows users to easily interact with the web app.
The straightforward layout ensures easy navigation for users across all ranges of expertise with minimal effort.
The following snapshot is the web app interface of the application.

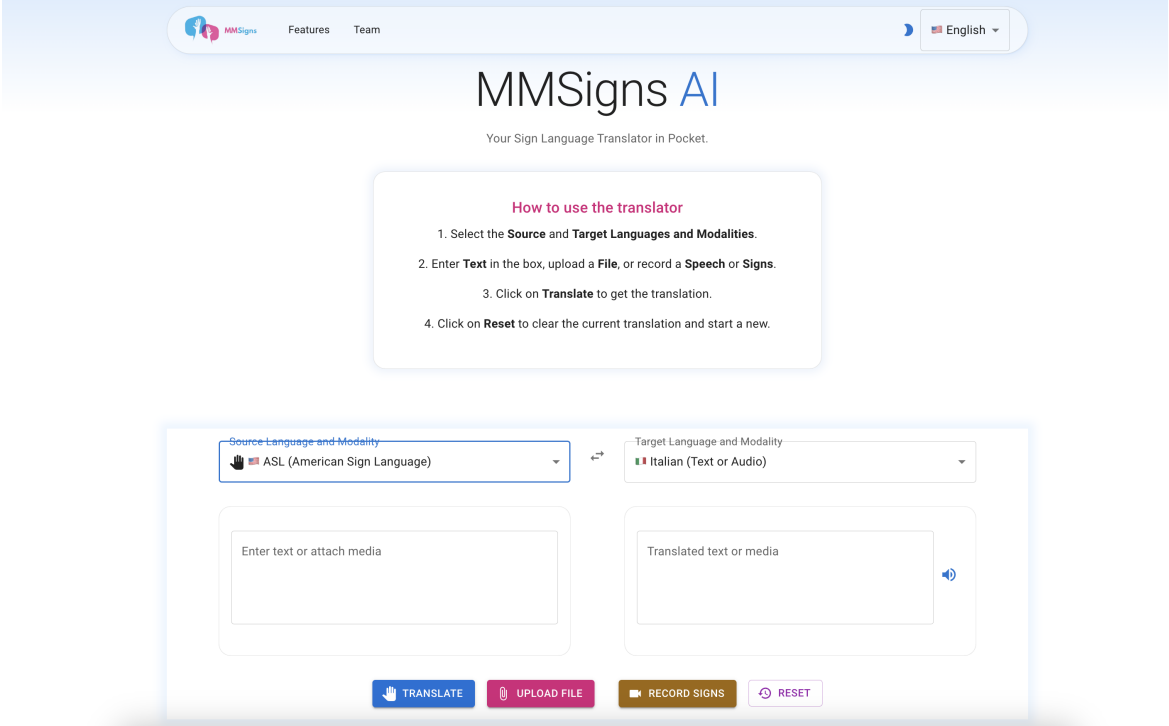

Figure 7: MMST web app interface. Full resolution schema available here.

The main components of the interface are explained:

- **Header**: Shows application logo, name ("MMSigns AI"), and navigation links such as "Features" and "Team". It also features a mode toggle for switching between light and dark themes. Next to it, there is a drop menu for selecting the website language between Italian, English, and Spanish; This function was implemented through the *i18next* packet.

- **Language selection**: allows users to choose the source and target languages for the translation.

- **Input and output area**: areas where the user can input the source sentence and the system will place the resulting translation.
  Both areas are designed to be multimodal, as they support textual, audio, and visual inputs.

In the case of textual inputs, the user can type or paste the source sentence in the box.
In the case of audio and video inputs, the system allows the user to record them on the spot or to upload some pre-existing files.

- **Action Buttons**:
  - **Translate Button**: Initiates the translation process for the provided input.
  - **Upload {Audio,Video}**: Allows users to upload an audio or a video file from the device storage. The system automatically sets the button to "Audio" or "Video" according to which modality is more appropriate for the selected source language. E.g. when inputting sign language, it does not make sense to upload an audio file, but it does for the upload of a video clip. Similarly, when selecting a spoken language, inputting a video file is illegal, while sending an audio file makes sense.
  - **Record {Audio,Video} Button**: Enables users to record an audio or video clip directly through their device. Follows the same dynamicity principle of the Upload button.
  - **Reset Button**: Clears all input and output fields, this action is mandatory when uploading subsequent videos in the app.
  - **Speaker Button**: When an output text is available this button allows to synthesize it to speech through the *react-speech-kit*. This corresponds to the Text-to-Speech transformation, it is the only transformation that occurs solely in the frontend.

Each of the above key components is implemented in its own ReactJS component:

- **App Component**: The main component that encapsulates all other components. It manages the state and logic for rendering the UI and handles the communication between different parts of the app.
- **Header Component**: Manages the display of the app's logo, navigation links, and theme toggle. It provides a consistent navigation experience across the application.
- **Language Selector Component**: Provides the dropdowns for selecting the source and target languages. It is designed to handle the dynamic list of supported languages and trigger appropriate actions when the user makes a selection.
- **Input Area Component**: Handles user inputs, including text and media uploads. This component also manages the state of user-provided inputs and validates them before sending them to the backend for processing.

- **Output Area Component**: Displays the translated output. It updates dynamically based on the results returned from the backend services.
- **Action Buttons Component**: Contains the buttons for various user actions such as translating, uploading media, recording video and audio, and resetting. Each button is linked to corresponding handlers that perform the necessary actions.

The frontend of MMST interacts with the backend through well-defined API endpoints, one for each of the supported input-output pairs: {audio,text,sign}-to-{audio,text,sign}, as per Figure 1

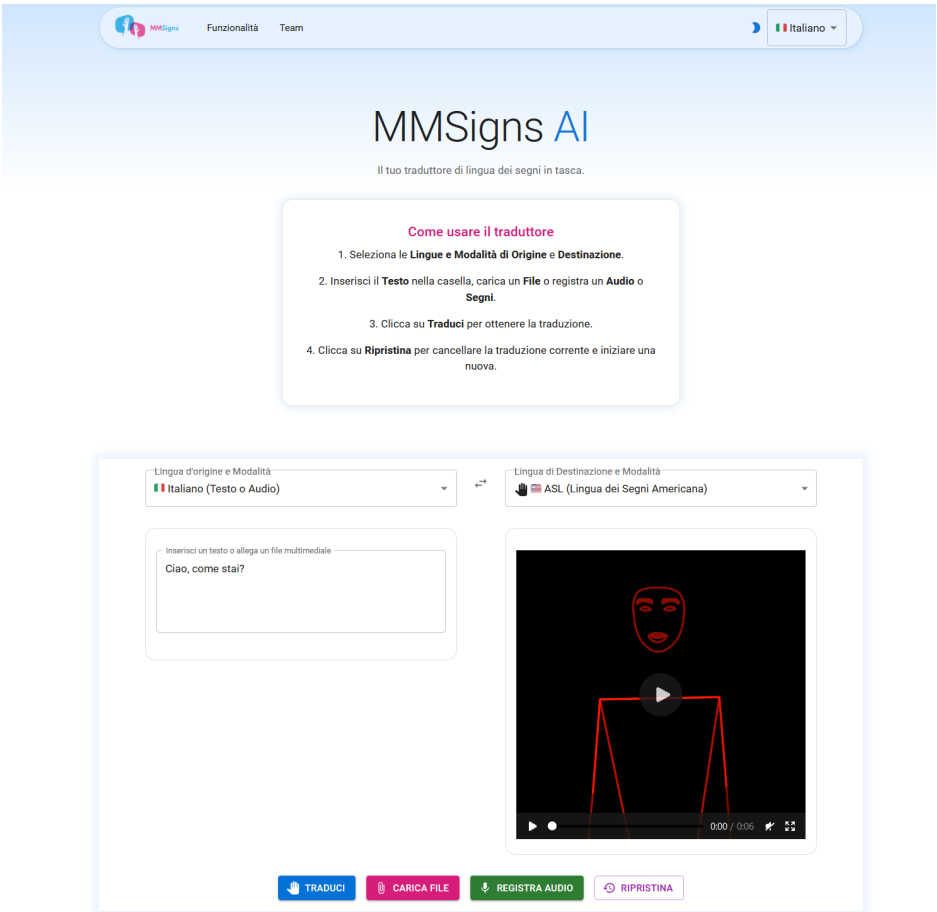

Figure 8: MMST web app interface, text-to-sign translation completed. Full resolution image available here.

```
90    const handleTranslate = async () => {
122     setIsLoading(true);
123     if (sourceText) {
124       if (isTargetSignLanguage) {
125         ({ url, requestBody, headers } = createTextToSignRequest(
126           sourceText,
127           sourceLanguage,
128           targetLanguage
129         ));
130       } else {
131         ({ url, requestBody, headers } = createTextToTextRequest(
132           sourceText,
133           sourceLanguage,
134           targetLanguage
135         ));
136       }
137     } else if (audioSrc) {
138       if (!audioSrc) {
139         newErrors.audioSrc = <Trans i18nKey="error2"> </Trans>;
140         setErrors(newErrors);
141         setIsLoading(false);
142         return;
143       }
144       ({ url, requestBody, headers } = await createAudioRequest(
145         audioSrc,
146         sourceLanguage,
147         targetLanguage
148       ));
149     } else if (recordedVideoUrl || file) {
150       if (!recordedVideoUrl && !file) {
151         newErrors.videoSrc = <Trans i18nKey="error3"> </Trans>;
152         setErrors(newErrors);
153         setIsLoading(false);
154         return;
155       }
156       if (isSourceSignLanguage && isTargetSignLanguage) {
157         ({ url, requestBody, headers } = await createSignToSignRequest(
158           recordedVideoUrl || file,
159           sourceLanguage,
160           targetLanguage
161         ));
162       } else if (isTargetSignLanguage) {
163         ({ url, requestBody, headers } = await createVideoRequest(
164           recordedVideoUrl || file,
165           sourceLanguage,
166           targetLanguage
167         ));
168       } else {
169         ({ url, requestBody, headers } = await createVideoRequest(
170           recordedVideoUrl || file,
171           sourceLanguage,
172           targetLanguage
173         ));
174       }
```

Figure 9: Frontend code to handle each of the supported input source type. Full resolution image available here

```
const createTextToSignRequest = (text, src, trg) => ({
  url: "http://localhost:8000/translate/text_to_sign",
  requestBody: { text, src, trg },
  headers: { "Content-Type": "application/json" },
});

const createTextToTextRequest = (text, src, trg) => ({
  url: "http://localhost:8000/translate/text_to_text",
  requestBody: { text, src, trg },
  headers: { "Content-Type": "application/json" },
});

const createAudioRequest = async (audioSrc, src, trg) => {
  const response = await fetch(audioSrc);
  const audioBlob = await response.blob();
  const arrayBuffer = await audioBlob.arrayBuffer();
  const base64Audio = btoa(
    String.fromCharCode(...new Uint8Array(arrayBuffer))
  );

  const requestBody = {
    base64Audio: base64Audio,
    src: src,
    trg: trg,
  };
  return {
    url: "http://localhost:8000/translate/audio_to_text",
    requestBody: JSON.stringify(requestBody),
    headers: { "Content-Type": "application/json" },
  };
};
```

Figure 10: Frontend code to create HTTP requests for each of the supported input source type. Full resolution image available here

```
const createVideoRequest = async (videoSrc, src, trg) => {
  if (file) videoSrc = URL.createObjectURL(file);

  const response = await fetch(videoSrc);
  const videoBlob = await response.blob();

  const toBase64 = (videoBlob) =>
    new Promise((resolve, reject) => {
      const reader = new FileReader();
      reader.readAsDataURL(videoBlob);
      reader.onload = () => resolve(reader.result.split(",")[1]);
      reader.onerror = (error) => reject(error);
    });

  const base64Video = await toBase64(videoBlob);
  console.log(base64Video);
  return {
    url: "http://localhost:8000/translate/sign_to_text",
    requestBody: JSON.stringify({ base64Video, src, trg }),
    headers: { "Content-Type": "application/json" },
  };
};

const createSignToSignRequest = async (videoSrc, src, trg) => {
  const response = await fetch(videoSrc);
  const videoBlob = await response.blob();

  const toBase64 = (videoBlob) =>
    new Promise((resolve, reject) => {
      const reader = new FileReader();
      reader.readAsDataURL(videoBlob);
      reader.onload = () => resolve(reader.result.split(",")[1]);
      reader.onerror = (error) => reject(error);
    });

  const base64Video = await toBase64(videoBlob);
  console.log(base64Video);
  return {
    url: "http://localhost:8000/translate/sign_to_sign",
    requestBody: { base64Video, src, trg },
    headers: { "Content-Type": "application/json" },
  };
};
```

Figure 11: Frontend code to create HTTP requests for each of the supported input source type. Full resolution image available here

6.3 Backend

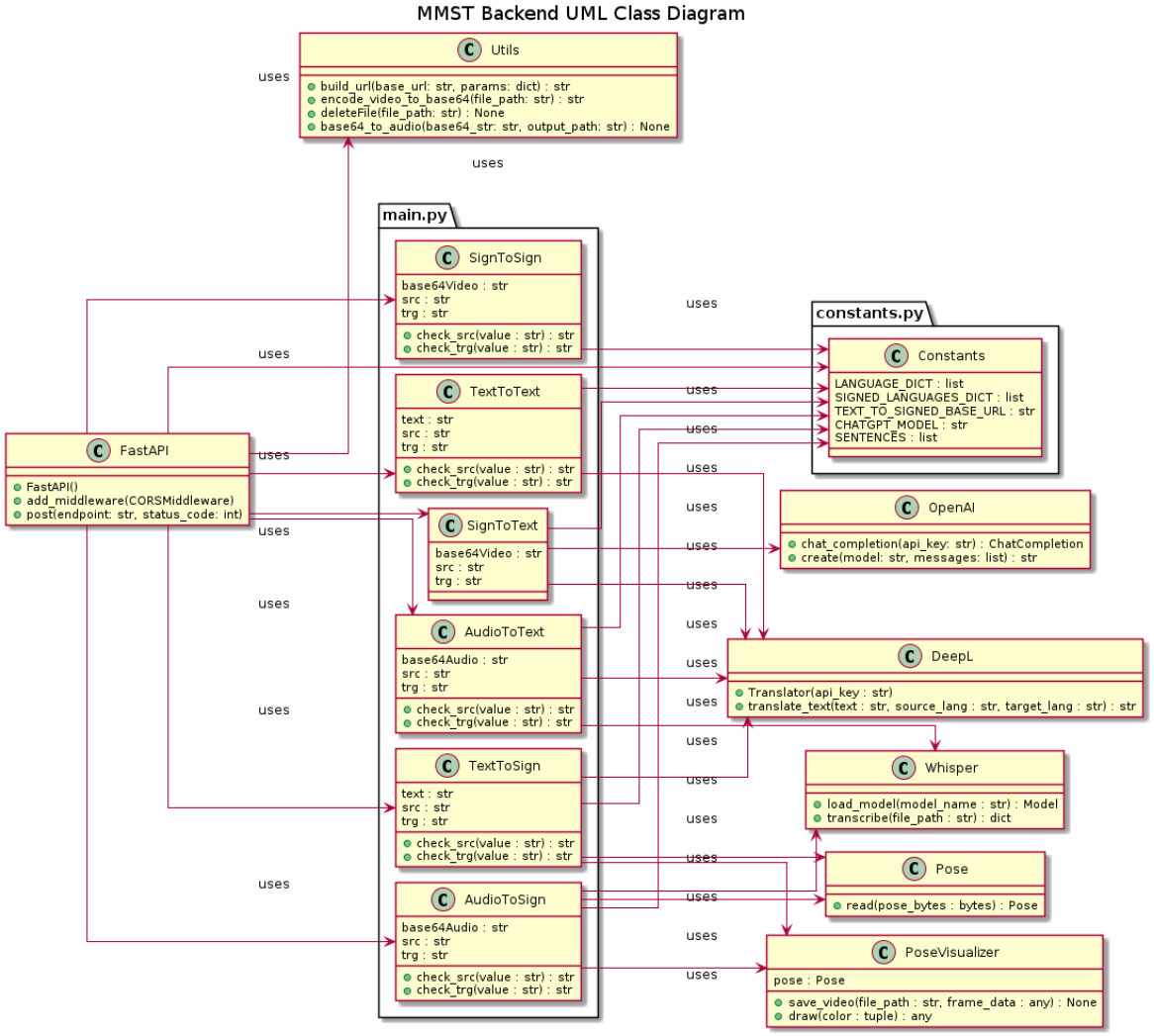

Figure 12: UML schema of MMST backend. Full resolution schema available here

6.3.1 Frameworks

The backend of the **MMST** application employs **FastAPI** with Python bindings, chosen for its versatility and the extensive availability of supporting libraries. FastAPI is the core framework of the MMST backend, as it offers high performance, ease of use, and inherent support for asynchronous processing. HTTP requests are used in the backend-frontend interactions.

The backend has been implemented so as to offer appropriate data handling for text, audio, and video modalities, as well as asynchronous processing, crucial for correct communication with the frontend, and validation. Validation ensures that data can actually be fed to the models and is performed using **Pydantic**, as it allows to check whether incoming requests conform to the expected structure and content.

OpenAI **Whisper** [19] is used to convert spoken language inputs into text. It has been chosen due to its high Speech-to-Text performance across a vast array of

languages, noise robustness, and easy integration with our code. Whisper's advantages come from its Transformer Neural Network [25], which has been trained to minimize a multitask loss, composed by a transcription and a translation objective.
`soundfile` handles audio data, while `io` manages in-memory file operations, when needed.
DeepL is used for Text-to-Text translations, as it offers high accuracy and fluency, across multiple languages.
Sign.mt is used to handle sign language-related translations and since it does not offer Python bindings nor official support, HTTP requests are done via the `requests` library. When the destination language is a sign language, Sign.mt returns the video as a pose object, which is handled and animated using the `pose_format` library and the **ffmpeg** suite.

6.3.2 Utility libraries

dotenv is used to store and retrieve sensitive information, like API keys
os, **datetime**, and **random** provide utility methods to handle files in an Operating System-agnostic way, manage time and randomicity.
base64 and **OpenCV** are used to deal with audio and video files, respectively, to encode audio into base64 representation during the frontend-backend communication and to pre-process sign language input videos before feeding them to GPT-4o.

6.3.3 Methods

Figures below show high-level steps of each **API Endpoints** exposed in the backend. Sign-to-Sign is a **concatenation** of Sign-to-Text and Text-to-Sign, while the speech part of *-to-Audio translations transforms the result from a *-to-Text operation with a speech synthesis applied in the frontend.
For Sign-To-Text (Figure 17), inspired by [11], **meaningful frames** are extracted from the source video, applying a Laplacian operator using methods from the OpenCV library, and then fed to GPT-4o [8], which translates them to the desired spoken destination language.
This solution was favored over other candidate approaches, like [22] and [17], as they would have required deploying Deep Learning models on our machines which would have resulted in much slower computations.

Figure 13: Text-to-Text backend method schema. Full resolution schema available here.

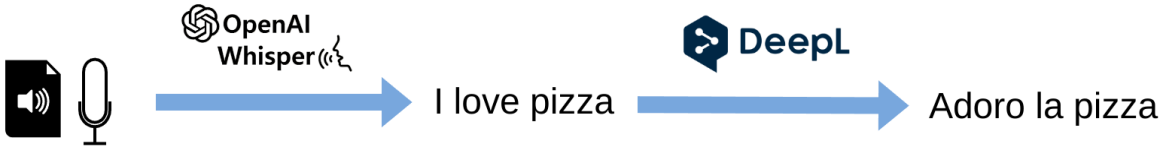

Figure 14: Audio-to-Text backend method schema. Full resolution schema available here.

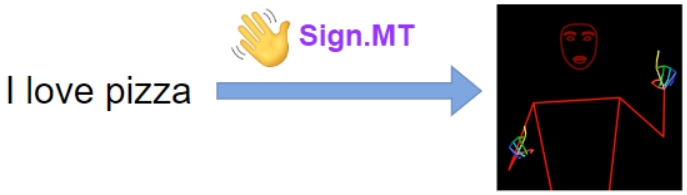

Figure 15: Text-to-Sign backend method schema. Full resolution schema available here.

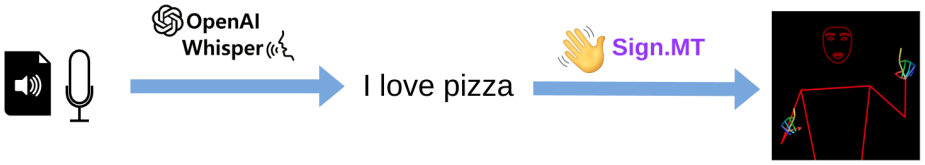

Figure 16: Audio-to-Sign backend method schema. Full resolution schema available here.

Figure 17: Sign-to-Text backend method schema. Full resolution schema available here.

6.4 Code

Our entire codebase, which includes the frontend and the backend, is fully open source and available in the official GitHub repository at this address.

7 Conclusions and Future works

In this report, Multimodal Sign Language Translator (MMST), a **multimodal**, **polyglot**, and **bidirectional** web app has been presented.
The current version of the application orchestrates several services to offer a **unified solution** available on the web through a simple and intuitive interface. As for now, the translations are managed through external APIs, which guarantees **high-quality outputs**, but at the cost of **no control** on external decisions or shut-offs. More work is necessary to be independent of external services, but this requires training custom models or including pre-trained models in the application logic. Another existing pain point is the Sign-to-* translation because it uses GPT-4o services, which during the time of this report require money to process an input and return an output. The application as it is can have a **real impact** on people learning sign languages, for communication it works, but **without real-time** translation functionality, the process could be slow.
Aspects regarding domain knowledge, theoretical and practical design, and software implementation have been discussed throughout the report, and now some possible **future works** and improvements are presented below.

7.1 More Type of Multimodal relations

7.1.1 Complementarity

Complementarity type of multimodal relation could be implemented to address and resolve ambiguities inherent in translation. For example, the Italian word "pesca" may correspond to "fishing" or "peach", in English (and, consequently, in American or British Sign Language).
To **disambiguate** such cases, MMST could show additional contextual information, such as images, audio clips, or descriptive text, in a complementary modality, allowing users to understand the precise meaning intended in the translation.

7.1.2 Redundancy

Redundancy could further improve the comprehensibility of translations. When displaying a sign language video, adding subtitles beneath the video could serve as a supplementary cue, **reinforcing** the message conveyed through signs.
This multimodal reinforcement can be particularly beneficial for learners, who may

learn sign languages faster thanks to the translation being presented in multiple formats.

7.2 General improvements to the application

Expanding the range of **supported** spoken and sign **languages** would be essential to cater to a broader, global user base.
Integrating **Natural Language Understanding** (NLU) capabilities would enhance translation accuracy by allowing the system to grasp context, idioms, and colloquialisms, providing more appropriate translations in ambiguous cases.
Large Language Models (LLMs) such as GPTs [18] [3] [16], LLaMA [24] [23] [9], Gemini [7] [6] may be leverage for this direction.
Additionally, recent research managed to optimize LLMs to run well on mobile devices [1], which would help in porting MMST onto a mobility app.
Adopting **real-time capabilities** would benefit {Audio, Sign}-to-Sign translations, and Augmented Reality (AR) in the form of overlaying sign language translations onto real-world environments may be one possible way of including real-time capabilities into a mobile version of MMST.

References

[1] *A Survey on Model Compression for Large Language Models*. 2023. URL: https://arxiv.org/abs/2308.07633.

[2] *ASL Bloom*. URL: https://www.aslbloom.com/.

[3] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. URL: https://arxiv.org/abs/2005.14165.

[4] *Duolingo*. URL: https://www.duolingo.com/.

[5] Mary Ellen Foster. *State of the art review: Multimodal fission*. 2002. URL: https://www.researchgate.net/publication/220029694_State_of_the_art_review_Multimodal_fission.

[6] *Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context*. 2024. URL: https://arxiv.org/abs/2403.05530.

[7] *Gemini: A Family of Highly Capable Multimodal Models*. 2024. URL: https://arxiv.org/abs/2312.11805.

[8] *GPT-4o*. 2024. URL: https://openai.com/index/hello-gpt-4o/.

[9] *Introducing Meta Llama 3: The most capable openly available LLM to date*. 2024. URL: https://ai.meta.com/blog/meta-llama-3/.

[10] Zifan Jiang et al. *Machine Translation between Spoken Languages and Signed Languages Represented in SignWriting*. 2023. arXiv: 2210.05404 [cs.CL].

[11] *LLMs are Good Sign Language Translators*. 2024. URL: https://openaccess.thecvf.com/content/CVPR2024/papers/Gong_LLMs_are_Good_Sign_Language_Translators_CVPR_2024_paper.pdf.

[12] Jean-Claude Martin. *Towards "intelligent" cooperation between modalities. The example of a system enabling multimodal interaction with a map*. 1997.

[13] *MMST Need Finding Questionnaire - English*. 2023. URL: https://forms.gle/nHjUKv1ZqwVo2hdw9.

[14] *MMST Need Finding Questionnaire - Italian*. 2023. URL: https://forms.gle/CHCxT9FGr7vp9yyY8.

[15] Laila Dybkjær Niels Ole Bernsen. *Multimodal Usability*. 2009. URL: https://link.springer.com/book/10.1007/978-1-84882-553-6.

[16] OpenAI. *GPT-4 Technical Report*. 2023. URL: https://arxiv.org/abs/2303.08774.

[17] *OpenASL: A Large-Scale Open-Domain Sign Language Translation Dataset*. 2022. URL: https://arxiv.org/abs/2205.12870.

[18] Alec Radford et al. *Language Models are Unsupervised Multitask Learners*. 2019. URL: https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[19] Alec Radford et al. *Robust Speech Recognition via Large-Scale Weak Supervision*. 2022. arXiv: 2212.04356 [eess.AS].

[20] *Sign School*. URL: https://www.signschool.com/.

[21] *Sign2Me*. URL: https://play.google.com/store/apps/details?id=eu.sign2me&hl=en_US&pli=1.

[22] *SLTUNET: A Simple Unified Model for Sign Language Translation*. 2023. URL: https://openreview.net/forum?id=EBS4C77p_5S.

[23] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. URL: https://arxiv.org/abs/2307.09288.

[24] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. URL: https://arxiv.org/abs/2302.13971.

[25] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].