POLITECNICO DI TORINO

MASTER's Degree in Data Science and Engineering

MASTER's Degree Thesis

Cloud-Based Architecture for Italian Sign Language Translation with Fine-Tuned LLMs for LIS-to-Italian Conversion

Supervisors

Prof. Maurizio MORISIO

Company Tutor

Dario SAMMARUGA

Candidate

Tijana ILIEVSKA

APRIL 2025

Abstract

This thesis aims to develop a client-server system for translating Italian Sign Language (LIS) into Italian, focusing on improving communication accessibility for individuals who are deaf or hard of hearing. The work was conducted as part of the research project LIS2S, led by ORBYTA TECH S.r.l., which aims to advance sign language translation technologies. The system integrates machine learning, computer vision, and web technologies to bridge the communication gap between LIS users and the broader Italian-speaking community.

The proposed system captures video input, which is processed to extract motion data. This data is then used to recognize signs, which are converted into LIS glosses (written representations or transcriptions of the signs in sign language) that capture the meaning of the gestures made. These glosses are subsequently translated into written or spoken Italian. The system utilizes Django for web application management, Docker for containerization, and WebSocket for efficient data transmission. Large language models (LLMs) and generative models will be explored to generate synthetic data, supporting the training of models and addressing the challenges in the translation process.

A key challenge addressed in this work is the lack of extensive annotated datasets for training models focused on LIS-to-Italian translation. Additionally, translating LIS glosses into fluent and grammatically correct Italian sentences presents significant complexities. By leveraging machine learning techniques, the system has been trained to map these glosses to the corresponding Italian language structure.

The system's development provides a functional platform for translating LIS to Italian in near real-time, offering a valuable tool for improving communication accessibility. This work contributes to the field of sign language translation by creating an end-to-end solution that enhances the interaction between the deaf and hard-of-hearing community and the Italian-speaking population, furthering inclusivity and bridging language gaps.

Acknowledgements

I would like to thank Professor Maurizio Morisio for his continuous support and guidance throughout the development of this thesis. I am also grateful to the Orbyta Tech team, and especially to my company tutor Dario Sammaruga, for his valuable ideas, technical insights, and advice at every stage of the work.

A heartfelt thank you goes to my parents and my sister, who have always supported me not just in this project, but in every step of my life. They've been a steady presence, always reminding me to keep things in perspective and not lose sight of what really matters. I'm truly grateful for everything they've passed on to me—both in words and by example.

Finally, I want to thank my friends for the motivation, distractions when needed, and the balance they brought. From each of them, I've learned something new, and they made this journey a lot more enjoyable and meaningful.

Table of Contents

List of Tables

List of Figures

Acronyms

LIS

Lingua dei Segni Italiana or Italian Sign Language

LLM

Large Language Model

CNN

Convolutional Neural Network

RNN

Recurrent Neural Network

PCA

Principal Component Analysis

Chapter 1

Introduction

1.1 Motivation

Communication is a fundamental human right, yet individuals who are deaf, non-verbal, or hard of hearing continue to encounter substantial challenges in everyday interactions due to the scarcity of reliable and accessible sign language translation technologies. Italian Sign Language (LIS), like all sign languages, is a rich and complex linguistic system with its own unique grammar, syntax, and visual-spatial structure. This makes direct word-for-word translation from LIS into spoken Italian not only impractical but also prone to significant loss of meaning and nuance.

Despite the rapid advancements in machine translation for spoken and written languages—thanks in large part to the emergence of deep learning and transformer-based models—sign language remains at the periphery of mainstream translation research. One of the central obstacles lies in the multimodal nature of sign language, which involves hand shapes, movements, facial expressions, and spatial references, all of which are difficult to represent in traditional textual formats. Moreover, the scarcity of high-quality, annotated LIS datasets limits the application of data-hungry neural models and hinders the development of generalizable translation systems.

This thesis is motivated by the pressing need to bridge this gap and to empower the deaf community through inclusive technology. It proposes the development of a modular, cloud-based system capable of translating LIS glosses—textual approximations of signs—into coherent Italian sentences. The system leverages the capabilities of large language models (LLMs), which are fine-tuned specifically for this task using both real and synthetically generated training data. By harnessing the flexibility and semantic understanding of LLMs, this approach aims to produce translations that are not only grammatically correct but also contextually appropriate, even when starting from sparse or simplified input. Ultimately, the goal

is to contribute a scalable and adaptable solution to the broader effort of making digital communication more accessible for everyone, regardless of hearing ability.

1.2 Objectives

This thesis aims to design and implement a system for LIS-to-Italian translation, leveraging machine learning and cloud-based technologies. The system follows a client-server architecture, ensuring efficient processing while keeping the client lightweight. The objectives of this project are as follows:

- Develop a scalable cloud-based system for translating LIS glosses into Italian.
- Fine-tune a large language model to improve translation quality.
- Implement an efficient real-time processing pipeline using WebSockets and Redis.
- Ensure the modularity and scalability of the system to facilitate future improvements.

The project is not only intended as a proof of concept but also as a step towards practical applications in improving accessibility for deaf individuals.

1.3 Proposed Methodology and Results

The proposed methodology is divided into three main stages:

1.3.1 System Architecture

The system is built on a client-server model with an architecture designed for asynchronous and concurrent data processing. The client captures video input and extracts frames, which are transmitted in real time to the server via WebSocket. On the server side, each incoming frame is processed asynchronously—keypoints are extracted, compressed using PCA, and stored in Redis streams for efficient, non-blocking access. These steps allow for concurrent handling of data across multiple components. Once sufficient information is gathered, the server performs sign recognition and leverages a fine-tuned Large Language Model (LLM) to generate fluent Italian translations. This architecture enables responsive interaction and lays the foundation for near real-time translation performance.

1.3.2 Dataset Preparation and Model Fine-Tuning

Due to the scarcity of large annotated LIS-to-Italian datasets, an initial dataset of 946 examples was generated and subsequently augmented through techniques such as back-translation and targeted vocabulary expansion. This process expanded the dataset to approximately 2000 examples. Two translation models—mBART and MarianMT—were fine-tuned on this augmented dataset under constrained computational resources, with adjustments to batch sizes and training epochs to fit within available hardware limits.

The performance of the translation models was evaluated using BERTScore, which measures the semantic similarity between the generated translations and the reference sentences. The obtained results indicate that data augmentation leads to a notable improvement in translation quality, with mBART achieving the highest performance.

1.3.3 System Deployment

The system is containerized using Docker and designed to be stored in the cloud, with models dynamically loaded into the server application for real-time translation.

1.4 Challenges and Limitations

Despite advancements in natural language processing, sign language translation presents several challenges that influence both data collection and model development. The main limitations encountered in this project include:

1.4.1 Data-Related Challenges

- The lack of large-scale LIS-Italian datasets makes it difficult to train models with sufficient generalization capabilities.
- Generating synthetic data for sign language is more complex compared to text-based data augmentation techniques.

1.4.2 Model Training Challenges

- Training large language models requires significant computational resources, which were limited in this project.

- Due to hardware constraints, the number of training epochs and batch sizes had to be reduced, potentially affecting model performance.

1.4.3 Real-Time Processing Challenges

- Handling real-time data streams efficiently is computationally demanding, particularly when processing video frames.
- Maintaining stable WebSocket communication and managing temporary storage in Redis requires careful system design.

While these challenges influenced design choices, the system was developed to balance efficiency and performance, ensuring a functional and scalable translation process.

1.5 Contributions

Despite these limitations, this project contributes to the field by:

- Developing an end-to-end system for LIS-to-Italian translation.
- Fine-tuning and evaluating large language models for LIS translation.
- Generating synthetic data and implementing data augmentation strategies to compensate for the limited dataset size.

This work lays the foundation for future improvements in sign language translation technology.

1.6 Structure of the Thesis

This thesis is structured as follows:

- **Chapter 2 – Background and Related Work:** An overview of existing research on sign language translation, machine translation, and large language models.
- **Chapter 3 – System Architecture:** A detailed explanation of the client-server system, data processing pipeline, and model integration.
- **Chapter 4 – Implementation:** A technical discussion of data handling, model fine-tuning, WebSocket communication, and system deployment.

- **Chapter 5 – Experimental Results:** Evaluation of model performance, including translation accuracy and real-time processing efficiency.
- **Chapter 6 – Conclusion and Future Work:** A summary of key findings, limitations, and potential directions for further research.

This structure ensures a clear progression from theoretical foundations to system development and evaluation.

Chapter 2

State of the Art and Related Works

2.1 Sign Language Translation: An Overview

Sign language translation is an active area of research in the field of human-computer interaction, with substantial advancements in languages such as American Sign Language (ASL) and German Sign Language (DGS). These efforts predominantly rely on computer vision and machine learning to map gestures into glosses and subsequently into spoken or written text [1, 2].

Italian Sign Language (LIS), however, remains underrepresented in the literature. One of the main obstacles is the lack of annotated corpora and standardized linguistic resources [3]. LIS poses unique challenges due to its visual-spatial modality, grammar distinct from spoken Italian, and regional variability in signs and syntax. Consequently, real-time LIS-to-Italian translation systems remain limited in both academia and industry.

A standard translation pipeline typically involves two stages: gesture recognition and language generation. The gesture recognition stage uses models like convolutional neural networks (CNNs) and recurrent neural networks (RNNs), while the language generation phase uses either rule-based grammar mappings or neural machine translation (NMT) models to generate output text or speech [4]. Recent approaches also incorporate large language models (LLMs) for gloss-to-text translation tasks [5].

2.2 Current Approaches to Sign Language Translation

2.2.1 Rule-Based Systems

Rule-based approaches are particularly relevant for low-resource sign languages like LIS, where annotated data is scarce. These systems rely on symbolic linguistic rules, handcrafted grammars, and structured pipelines to perform translation. For example, the ALEA editor developed by Hatami et al. [3] facilitates Italian-to-LIS translation by combining dependency parsing (via TULE) and morphological analysis (via TextPro). Semantic disambiguation is achieved using Italian WordNet.

The translation process involves three key stages: (1) linguistic parsing to extract syntax and semantics from the input sentence, (2) transformation rules to adapt spoken Italian into LIS grammar—such as omitting articles and inserting temporal or spatial markers—and (3) lexical selection of the appropriate LIS glosses based on semantic roles. While precise, this approach requires significant linguistic expertise and lacks adaptability to diverse sentence structures.

2.2.2 Neural Machine Translation Approaches

Neural Machine Translation (NMT) has become the dominant approach for high-resource sign languages. Architectures such as sequence-to-sequence models and Transformers have shown success in translating glosses into spoken languages, particularly in ASL and DGS contexts [6, 1]. However, LIS has yet to benefit fully from these advances due to the limited availability of parallel corpora.

One early attempt to integrate deep NLP techniques into LIS translation is presented by Mazzei et al. [7], who proposed a pipeline combining dependency parsing, ontology-based semantic interpretation, and sign generation using Combinatory Categorial Grammar (CCG). Although rule-based at its core, the system represents a bridge between traditional linguistic methods and machine learning.

2.2.3 Hybrid Approaches

Hybrid approaches that combine rule-based systems with statistical or neural models are gaining traction as a practical solution for LIS. These systems aim to compensate for data scarcity while leveraging the pattern recognition strengths of machine learning.

For instance, Moryossef et al. [8] propose an open-source, gloss-based pipeline that includes Swiss Italian Sign Language (LIS-Ticinese) and applies rule-based preprocessing (lemmatization, reordering, and filtering) before feeding sequences

into NMT models. Similarly, Moryossef et al. [9] demonstrate how simplified "near-glosses" can be generated via linguistic rules and then translated into complete glosses using neural networks.

2.3 Gloss-Based Translation Systems

2.3.1 The Role of Glosses in Sign Language Translation

Glosses serve as an intermediate textual representation that simplifies the mapping between sign language gestures and natural language. A gloss typically represents a single sign with an uppercase word in a spoken language, such as HOUSE or GO-TO, without inflections or function words. Glosses are often used in machine translation pipelines to decouple the visual recognition task from the language generation component [10].

For LIS, the use of glosses is especially prevalent due to the lack of extensive parallel LIS-to-Italian corpora. Most existing translation systems therefore adopt a two-step approach: first, recognizing gestures and converting them into glosses, and second, translating glosses into Italian sentences [3, 7].

2.3.2 Near-Gloss Generation

Recent studies have introduced the concept of "near-glosses," simplified textual forms approximating glosses that can be generated from spoken language using rule-based methods. For instance, Moryossef et al. [9] describe a preprocessing pipeline that includes lemmatization, removal of non-content words (e.g., determiners), and sentence reordering to match the typical grammar of sign languages (e.g., SVO to SOV). These near-glosses serve as a bridge between spoken language and gloss format and can then be processed by neural models.

Similarly, Moryossef et al. [8] apply language-specific syntactic transformations to spoken input before translation, ensuring structural alignment with sign language grammar. These methods are valuable for low-resource languages like LIS where direct data-driven learning is not feasible.

2.3.3 From Glosses to Natural Language

While much work focuses on generating glosses from signs or text, translating glosses into fluent natural language presents its own challenges. Glosses are sparse and omit grammatical elements like articles, tense markers, and inflection. To address this, Fayyazsanavi et al. [5] propose a method using Large Language Models (LLMs) augmented with semantically aware label smoothing. This technique softens the target probabilities during training, allowing the model to assign non-zero

probabilities to semantically similar words, thereby improving translation fluency and coherence. Although their work centers on German Sign Language, the methods are adaptable to LIS and other sign languages.

2.4 Data Challenges and Solutions

2.4.1 Scarcity of Annotated Corpora

A major limitation in LIS translation is the lack of annotated datasets. Unlike ASL, which benefits from resources like RWTH-PHOENIX-Weather, LIS lacks large-scale corpora with aligned video, gloss, and spoken language data [10]. This data scarcity constrains both gesture recognition and natural language generation components and has led to the persistence of rule-based systems in LIS research [3].

2.4.2 Data Augmentation and Synthetic Data Generation

To mitigate the lack of data, several researchers have proposed synthetic data generation using Large Language Models (LLMs). Moryossef and Jiang [11] introduce SignBank+, a multilingual dataset that incorporates synthetic glosses created through LLM prompting. Their method uses predefined linguistic rules (lemmatization, reordering, and omission of function words) and automatic validation mechanisms to ensure quality and consistency.

Other studies have explored the use of LLMs such as GPT-3 to generate glosses or pseudo-parallel datasets for low-resource tasks [12, 13]. Though these studies were not LIS-specific, the same principles apply. Synthetic glosses generated in this way can be used to fine-tune gloss-to-text models or provide training data for gesture-to-gloss systems.

2.4.3 Lack of Standardization in LIS

LIS also suffers from a lack of standardization, with regional variations in vocabulary, grammar, and signing style across Italy. This poses challenges for building models that generalize well across user populations. The issue underscores the need for collaborative data collection and annotation efforts across multiple regions and dialects [10].

2.5 Translation Approaches for LIS

2.5.1 Linguistic Analysis and Transformation

The assisted translation tool described by Hatami et al. [3] illustrates a structured approach to Italian-to-LIS translation that incorporates several key steps:

1. **Linguistic Analysis**: The system uses TULE for dependency parsing and TextPro for lexical and morphological analysis to identify word lemmas, part-of-speech tags, and syntactic roles.

2. **Disambiguation with WordNet**: Ambiguous Italian words are resolved using Italian WordNet, ensuring that the correct LIS gloss is chosen based on context.

3. **Transformation Rules**: The system applies rule-based transformations to adapt Italian grammar to LIS grammar, including word reduction (omitting articles and prepositions), insertion of temporal and spatial markers, and semantic role tagging.

While this system focuses on Italian-to-LIS translation, many of these principles could be adapted for the reverse direction.

2.5.2 Deep Natural Language Processing

Mazzei et al. [7] present a more advanced approach that incorporates deeper natural language processing techniques:

1. **Dependency Parsing**: Analyzes the grammatical structure of Italian sentences to identify relationships between words.

2. **Ontology-Based Semantic Interpretation**: Extracts and interprets sentence meaning using predefined semantic knowledge.

3. **LIS Generation**: Produces a sequence of signs through sentence planning and realization using combinatory categorial grammar (CCG).

This approach represents an early attempt to bridge traditional rule-based methods with more sophisticated semantic analysis.

2.5.3 Large Language Models in Sign Language Translation

The most recent developments in the field point to the potential of Large Language Models (LLMs) in sign language translation. As demonstrated by Fayyazsanavi et al. [14] and Moryossef and Jiang [11], LLMs can be leveraged for various aspects of the translation pipeline, including:

1. Generating synthetic training data to overcome resource limitations
2. Enhancing gloss-to-text translation through semantically aware processing
3. Validating and improving the quality of translations

While these approaches have not yet been fully applied to LIS translation, they represent promising directions for future research.

2.6 Computer-Assisted Translation versus Automatic Translation

As discussed by Hatami et al. [3], sign language translation approaches can be broadly categorized into two types:

1. **Automatic Translation**: Methods like statistical or rule-based translation systems that aim to fully automate the translation process. These approaches often struggle due to the lack of large corpora for sign languages and typically exhibit high error rates.
2. **Computer-Assisted Translation (CAT)**: This approach supports users by partially automating translation tasks, such as suggesting words or phrases, while allowing for manual corrections. The ALEA tool described by Hatami et al. [3] is notably the first to focus specifically on assisted translation for LIS.

Given the current limitations in automatic LIS translation, computer-assisted approaches may offer a more practical path forward in the near term, allowing for human oversight while still benefiting from computational assistance.

2.7 Synthetic Data Generation for Sign Language Translation

A significant challenge in developing sign language translation models, especially for **LIS**, is the **lack of large annotated datasets**. Sign language datasets, such

as those for ASL, are limited, and the unique structure of LIS compounds this issue. As a result, researchers and developers are turning to **synthetic data generation** as a solution to this data scarcity.

Braga et al. [12] provide an overview of methods for generating synthetic datasets in machine learning. Synthetic data generation helps mitigate data limitations by creating realistic, labeled datasets without the need for extensive manual annotation. This is particularly useful in the context of sign language translation, where generating synthetic video data of LIS gestures can help train more robust models. By augmenting real datasets with synthetic data, models can learn to recognize a wider variety of gestures, including those seen in diverse environments and lighting conditions.

Additionally, Braga et al. [12] and Busker et al. [13] discuss how large language models (LLMs) like GPT-3 can generate personalized synthetic data, which can be applied to fields like sign language translation. While focused on question answering, the use of LLMs for generating contextually appropriate data parallels the need for **synthetic gloss generation** in sign language translation. By simulating glosses, synthetic data generation can fill in gaps in real-world datasets, which is critical for training translation systems.

Another relevant study is Busker et al. [13], which demonstrates how large language models can be used for generating synthetic text data. Although the focus is on textual data generation, the approach has potential implications for sign language gloss generation, where **textual glosses** can be generated in large quantities for training purposes, facilitating the development of gloss-to-language translation systems.

In the case of **LIS translation**, using **synthetic data** can also extend to generating both video data and gloss sequences. By utilizing existing computer vision models, synthetic sign language video clips can be created and labeled with glosses, providing a rich source of data for training.

2.8 Fine-Tuning Pre-trained Models for Sign Language Translation

Once sufficient data is collected—whether from real datasets or synthetic ones—the next step involves using pre-trained models for **fine-tuning**. Fine-tuning pre-trained models is particularly beneficial for tasks like **gesture recognition** and **sign language translation**, where the models need to be adapted to recognize the specifics of a new domain (e.g., LIS).

Fine-tuning is a well-established technique in machine learning, where a model pre-trained on a large dataset (such as general image recognition or language tasks) is adapted to a more specific task. In the context of **sign language translation**,

this process allows for quicker training and better performance, especially when dealing with limited labeled data.

Recent research highlights the effectiveness of fine-tuning pre-trained models for **sign language gesture recognition** and **gloss translation**. For example, Camgöz et al. [4] and Orbay and Akarun [6] demonstrate how CNN-based models for gesture recognition and Transformer-based models for translation can be successfully adapted to specific sign languages. Fine-tuning large pre-trained language models (like *BERT* or *GPT*) for gloss-to-text translation has shown promising results, as these models can capture linguistic nuances and better understand the mapping from gloss to spoken language.

Although focused on ASL and DGS, Fayyazsanavi et al. [14] present approaches that are currently being explored for LIS translation, where the **fine-tuned NLP models** are capable of handling the nuances of sign language grammar, which is different from spoken language.

2.9 Evaluation Metrics and Benchmarks

A notable gap in the literature is the lack of standardized evaluation metrics and benchmarks specifically for LIS translation. While general machine translation metrics like BLEU, METEOR, and TER are widely used for other languages, their applicability to sign language translation—and LIS in particular—remains limited. This absence of standard evaluation frameworks makes it difficult to compare different approaches and measure progress in the field.

2.10 Conclusion and Research Gaps

The current state of research on LIS translation reveals several important gaps and opportunities:

1. **Limited Application of Modern NLP Techniques**: While rule-based systems dominate LIS translation research, there remains significant potential for integrating modern neural network methods and large language models.
2. **Data Scarcity**: The absence of large annotated datasets continues to be a major bottleneck, highlighting the need for data augmentation strategies and synthetic data generation.
3. **Lack of End-to-End Systems**: Few, if any, comprehensive end-to-end systems exist for LIS-to-Italian translation, particularly for real-time applications.
4. **Evaluation Standards**: The field would benefit from standardized evaluation metrics and benchmarks specifically designed for LIS translation.

5. **Regional Variations**: Additional research is needed to address the variations in LIS usage across different regions of Italy.

These gaps represent important areas for future research and development, pointing to the need for interdisciplinary collaboration between linguists, computer scientists, and the deaf community to advance the state of LIS translation technology.

Chapter 3

Architecture of the System

3.1 Technical Overview of the components of the LIS-to-Italian Translation Application

The **LIS-to-Italian Translation App** is built using Python with the **Django framework**, leveraging its robustness for handling web-based applications. This app integrates various technologies to provide near real-time, seamless translation of Italian Sign Language (LIS) into spoken or written Italian. The app is structured to operate in a client-server architecture, utilizing WebSocket communication for real-time video stream transmission, deep learning models for gesture recognition, and machine translation techniques to convert recognized glosses into Italian. The following section outlines the key technologies used, the reasoning behind each choice, and the data flow through the system.

The system is structured around several key components, as illustrated in the Figure 3.2.

3.1.1 Client

The client is responsible for video acquisition and initiating the real-time data pipeline. It is implemented as a Django web application, containerized using Docker for consistent deployment and environment management. The client provides an intuitive user-friendly interface for uploading or recording sign language videos directly from the browser or the app.

Once a video is submitted, it is temporarily stored in the client container's file system using Django's file handling system. The uploaded file is then processed locally using OpenCV, which handles frame-by-frame extraction.

Each extracted frame is enriched with metadata—including timestamp, session ID, and run ID—and is compressed to reduce transmission load. The compressed

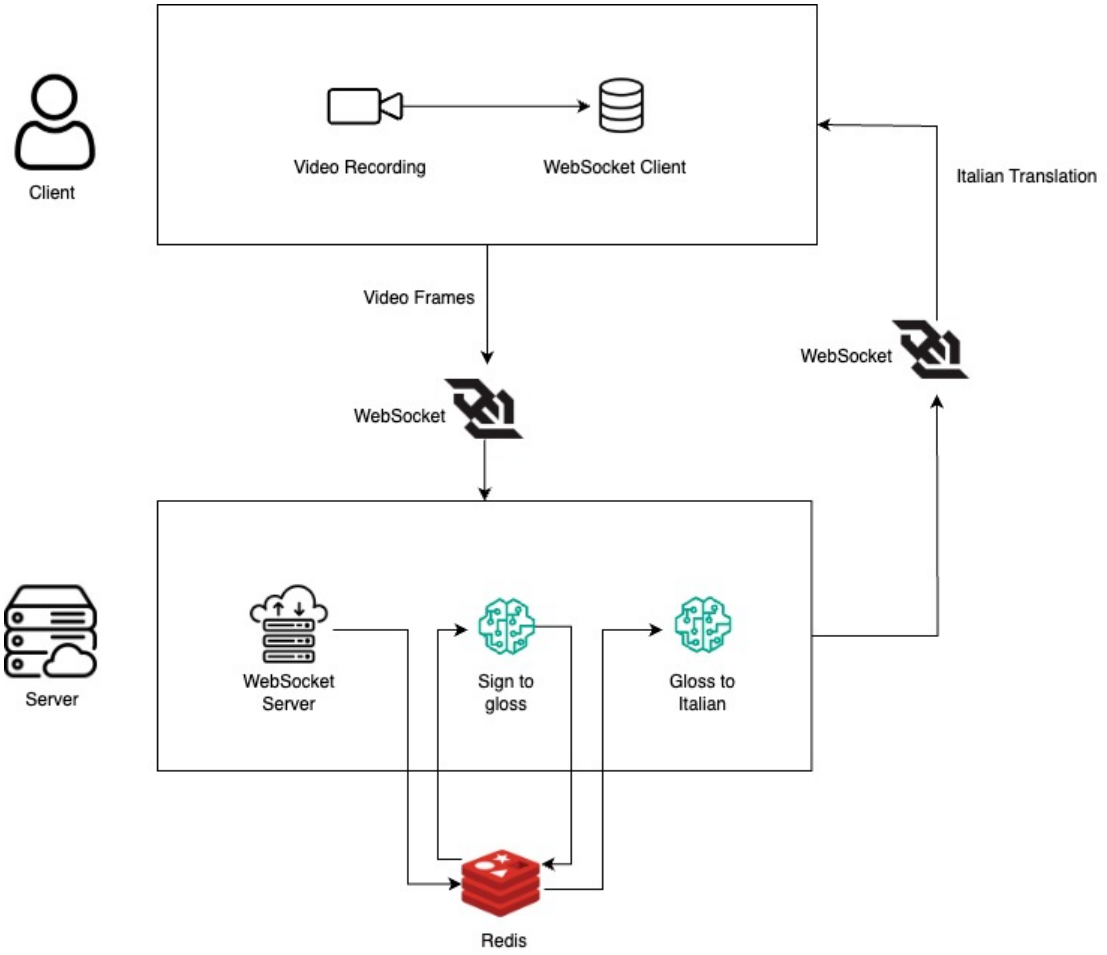

Figure 3.1: System Architecture

data is sent to the server over a persistent WebSocket connection. The WebSocket interface is implemented using Django Channels, enabling asynchronous, low-latency communication between the client and server.

Session tracking is handled through Django's middleware to ensure frames and their metadata remain properly grouped. This combination of Django, OpenCV, Channels, and Docker ensures that the client is modular, efficient, and capable of real-time video streaming in a scalable microservice environment.

3.1.2 Server

The server is structured as a modular Django project composed of multiple dedicated apps, each responsible for a specific part of the sign language translation pipeline. It is fully containerized using Docker, ensuring consistency, reproducibility, and

ease of deployment across environments.

The server consists of three key applications:

- **server:** The main app responsible for handling incoming WebSocket connections, storing data, and coordinating the video processing pipeline.
- **inference:** A dedicated app for sign recognition. It loads and manages pre-trained models, applies inference on PCA-reduced keypoints, and returns the predicted glosses.
- **llm:** This app manages the final translation of gloss sequences into fluent Italian sentences using Large Language Models (LLMs).

Django Channels powers the asynchronous WebSocket layer, enabling the server to handle incoming frame streams in real time without blocking the main application thread. Each incoming frame is received by an asynchronous consumer, allowing the system to process multiple streams concurrently while maintaining responsiveness.

Heavy processing tasks, such as keypoint-based sign inference and gloss-to-sentence translation, are offloaded to Celery workers. These tasks are queued and executed asynchronously, ensuring that computationally intensive operations do not interfere with the reception of new frames. Celery is backed by Redis, which acts as both the task message broker and an intermediate data buffer.

Redis Streams are used throughout the pipeline to decouple each stage of processing—from keypoint extraction, to PCA dimensionality reduction, to inference, and finally to sentence translation. Each stage reads from its respective Redis stream, processes the data, and writes the output to the next stream in the sequence. This stream-based communication enables fault tolerance, parallelism, and modularity, as each stage operates independently and can scale horizontally without affecting the others.

This architecture allows the server to maintain high throughput while managing asynchronous tasks in parallel, making it well-suited for real-time gesture processing applications.

3.1.3 Sign Recognition Model

The system employs a pre-trained sign recognition model developed as part of the LIS2S project, described in *Introducing Deep Learning with Data Augmentation and Corpus Construction for LIS* [15]. This model is designed to recognize Italian Sign Language (LIS) gestures from sequences of extracted keypoints and map them to corresponding glosses, which are simplified textual representations of signs.

From an architectural perspective, the model is integrated into the backend as a modular inference component. The model processes the input and outputs the predicted gloss.

The model architecture is based on a convolutional neural network (ConvNet) and it is trained to classify sequences of keypoints extracted from sign language gestures. This makes it well-suited for recognizing dynamic hand and body movements typical of sign language communication.

The prediction logic is encapsulated within the `inference` app of the Django project. By abstracting the model as a self-contained module, the system supports dynamic weight loading, model versioning, and flexible integration with the real-time processing pipeline.

3.1.4 LLM Model for Translation of Gloss Sequences to Italian

To translate recognized gloss sequences into fluent Italian sentences, the system incorporates a Large Language Model (LLM) as a dedicated component within the backend architecture. The LLM operates as a separate service, receiving a list of glosses as input and returning a grammatically coherent natural language sentence. This transformation from gloss to Italian is crucial for enabling understandable, user-facing outputs.

From an architectural standpoint, the LLM module is encapsulated within the `llm` app of the Django project. Translation tasks are offloaded to Celery workers, ensuring that LLM inference runs asynchronously and does not block other components of the pipeline. The model used is a transformer-based architecture pre-trained on multilingual translation tasks, fine-tuned to interpret LIS gloss patterns and produce high-quality Italian output.

Further details about the LLM selection, training, and fine-tuning process are provided in Chapter 5.

3.2 Data Flow Explanation

The data flow in the system follows a structured process, enabling the user's video input to be processed and translated into a meaningful output in the form of an Italian sentence representing the sign language gestures. The system is divided into two main containers: the client container, responsible for extracting and sending frames, and the server container, where the actual processing and model inference take place. Below is a detailed breakdown of the data flow:

1. **User Uploads Video:** The process begins when the user uploads or records a video of sign language gestures with their own device. This video file is saved and ready for further processing.

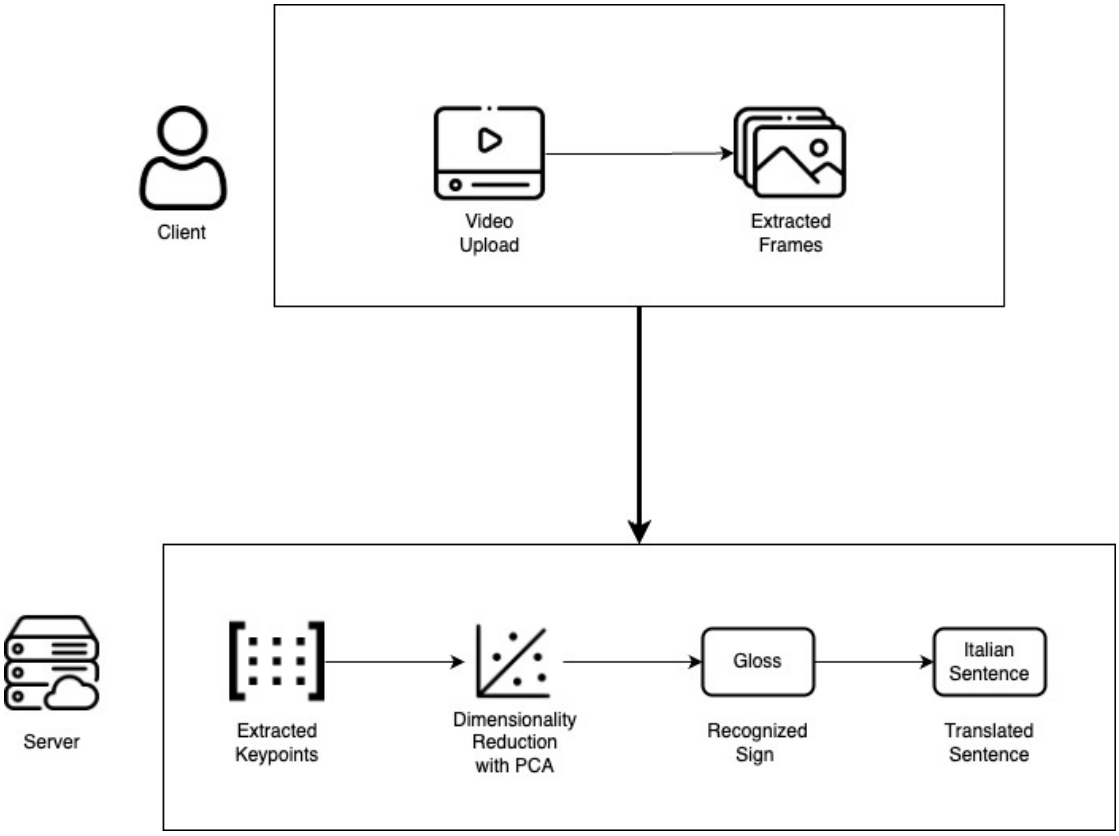

Figure 3.2: Data Flow in the system

2. **Frame Extraction and Transmission from the Client:** Once the user uploads a video, the client container initiates a real-time data flow that begins with temporarily storing the file. The video is then processed frame by frame using OpenCV. For each frame, the system generates a set of metadata, including a timestamp indicating the exact time of processing, a session ID to uniquely identify the user's session, and a run ID to track the specific video upload instance.

   This metadata is crucial to maintain the chronological integrity of the data and to allow the server to correctly associate each frame with its corresponding session and run. After the metadata is generated, it is combined with the image data of the frame, which is serialized and compressed to reduce the transmission load. The frames, originally stored as three-dimensional arrays (matrices) representing RGB pixel values, are converted to JSON-compatible structures and compressed before transmission.

   The WebSocket stream serves as a continuous data channel, transmitting

frames sequentially as they are extracted. A retry mechanism ensures reliability in case of temporary network disruptions. This approach allows for efficient streaming of video data with minimal latency, enabling the server to begin processing frames even before the entire video upload is complete.

3. **Data Storage in Redis Streams:** To ensure smooth and efficient data flow between components, compressed frames and inference results are temporarily stored in Redis Streams as binary data. Redis Streams provide a lightweight and high-throughput buffering mechanism that enables sequential retrieval of data in real time. This approach not only decouples the processing steps but also ensures that each module in the pipeline can asynchronously consume the data it requires without introducing bottlenecks.

   Redis Streams are used consistently throughout the entire pipeline—for example, to pass extracted keypoints to the PCA module, or to provide reduced data to the inference engine. Each step reads from and writes to a dedicated Redis stream, enabling modular communication, fault tolerance, and easy integration of new processing components.

4. **Keypoint Extraction in the Server Container:** After receiving the frames, the server extracts crucial keypoints from the video using **MediaPipe**, a cross-platform framework developed by Google for building multimodal applied ML pipelines. MediaPipe is often used in conjunction with OpenCV for efficient real-time video processing. It utilizes machine learning models to detect and track landmarks on various parts of the body, including the hands, arms, and upper body.

   In this system, MediaPipe is leveraged to identify spatial coordinates of 2D keypoints on each frame. These keypoints are extracted from regions of interest that are particularly relevant for sign language recognition—primarily the hands, arms, and upper body. The extracted keypoints form the basis for further gesture classification and translation tasks downstream.

   For the sign recognition model input requirements, we extract 3D landmarks (x, y, z) for the following regions:

   - **Pose:** 33 landmarks (only the first 25 corresponding to the upper body are used)
   - **Face:** All 468 landmarks
   - **Hands:** 21 landmarks for each hand (left and right)

   Each of these landmarks provides a 3D coordinate, resulting in a flattened 1D array for each region.

   This keypoints extraction process is illustrated in Figure 3.3.

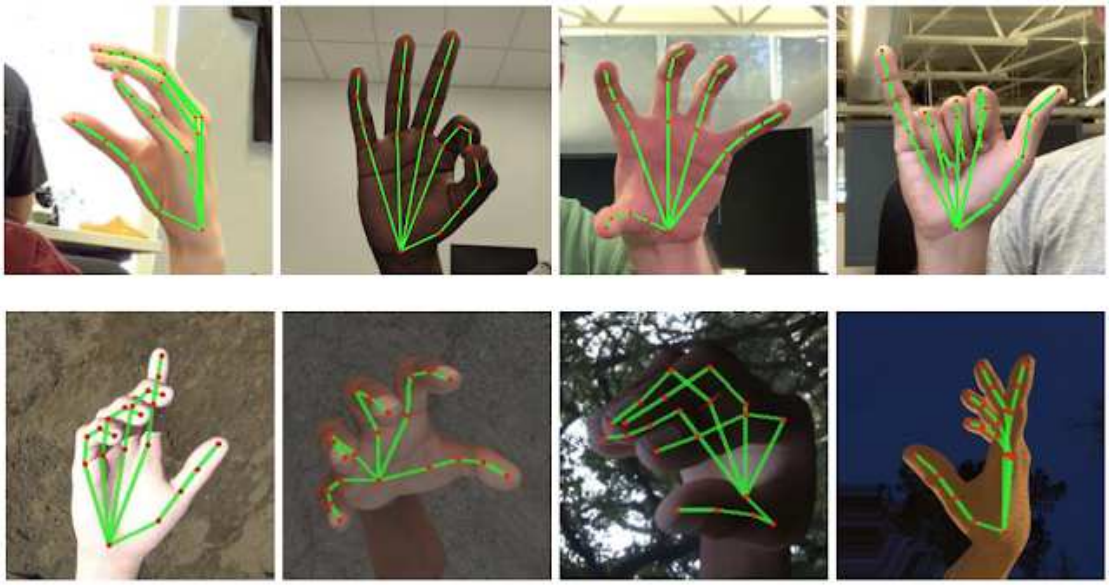

Figure 3.3: Detected keypoints on hand gestures

5. **Saving Keypoints to the Database:** The extracted keypoints—both in their raw form and after dimensionality reduction—are stored in the database to enable persistent access, retrospective analysis, and traceability across different stages of the pipeline. Each frame and its associated data are saved using the `Frame` model, which includes the following fields:

   - `timestamp` (`DateTimeField`): Records the exact time the frame was processed.
   - `session_id` (`CharField`): Uniquely identifies the user session.
   - `run_id` (`CharField`): Groups frames from the same video upload or session run.
   - `frame` (`BinaryField`): Stores the compressed raw frame data.
   - `keypoints` (`BinaryField`): Contains the extracted raw keypoints in binary format.
   - `reduced_data` (`JSONField`): Optionally stores the PCA-reduced keypoint array in JSON format.

   This structure ensures that all relevant information associated with a given frame—including its source, original content, extracted features, and reduced representation—is preserved and accessible for downstream tasks such as model inference, auditing, or visualization.

6. **Dimensionality Reduction:** To reduce the dimensionality of the extracted keypoints and prepare the data for efficient model input, we followed the methodology proposed in the LIS2S project [15]. The reduction process was conducted in three stages:

   - **Face Keypoint Selection:** Given that the face alone contributes 468 keypoints—significantly more than the hands or upper body—there was a risk of over-representing facial information in the learning process. To mitigate this imbalance, the suggested custom selection function was implemented to reduce the face keypoints from 468 to 128. These retained points were chosen based on their relevance to facial contours, such as the outline, eyes, eyebrows, and mouth.
   - **Dimensional Reduction from 3D to 2D:** As suggested by the MediaPipe documentation and reinforced by the LIS2S paper, we discarded the z coordinate (depth) from each keypoint due to its reduced accuracy. This transformation reduced each point's representation from 3D to 2D (x, y), helping simplify the input while preserving essential spatial information.
   - **Principal Component Analysis (PCA):** After spatial filtering, PCA was applied to further reduce the dimensionality of the flattened keypoint vector. The transformation retained only the top 4–6 components that captured approximately 95% of the variance in the data. This significantly reduced computational complexity while preserving the most informative aspects of the gesture representation.

   This multi-step approach to dimensionality reduction ensured a compact and balanced input representation, aligning with best practices established in prior work. It also helped enhance the training stability and performance of the downstream sign-to-gloss recognition model.

7. **Model Inference for Sign Recognition:** After the PCA-reduced keypoint data is collected and grouped, it is passed to the sign recognition model for inference. The input to the model is a two-dimensional NumPy array representing the temporal sequence of reduced keypoint vectors for a given time window. This input is reshaped and expanded along a new axis to match the expected input shape of the model.

   The model outputs a prediction vector containing probability scores for each possible sign class. If the highest probability exceeds a confidence threshold (e.g., 0.8), the corresponding sign label is retrieved from a predefined list of class names. This label, referred to as the recognized phrase, represents the predicted LIS gloss.

8. **Saving Recognized Signs to the Database:** Once a sign is recognized, it, along with its gloss and metadata, is saved in the database. This ensures that the system can track all recognized signs and their relationships to translated phrases.

9. **LLM Model for Sentence Translation:** After collecting a sequence of recognized signs (glosses), the system passes this list as input to a Large Language Model (LLM) to generate a fluent Italian sentence. The input to the LLM is a sequence of LIS glosses in textual form, structured as a list or string (e.g., `["IO", "SCUOLA", "ANDARE"]`), and the output is a grammatically correct and semantically coherent sentence in natural Italian (e.g., *"Io vado a scuola"*).

   Several pre-trained and fine-tuned LLMs were evaluated for this task. The model that consistently provided the most accurate and contextually appropriate translations was selected and integrated into the final pipeline.

   The LLM step transforms discrete sign-level predictions into complete spoken-language sentences, enabling natural interaction and interpretation from LIS to Italian.

10. **Saving Translated Phrases to the Database:** Once the Large Language Model (LLM) generates the final Italian sentence, the output is saved in the database using the `Sentence` model. This step ensures that each translation is persistently stored and can be retrieved later for user interaction, evaluation, or auditing.

    The data is saved with the following structure:

    - `batch_id` (`CharField`): A unique identifier that corresponds to the timestamp of the first frame in the group of recognized signs. This serves as a reference point for aligning gloss sequences with their original video context.
    - `gloss` (`JSONField`): The sequence of recognized signs (glosses) that were used as input to the LLM.
    - `italiano` (`JSONField`): The resulting Italian sentence produced by the LLM based on the provided glosses.

    This structure guarantees traceability from raw frames to the final translated sentence, preserving both the intermediate and final stages of the linguistic transformation process.

11. **Output:** Finally, the translated sentence is returned to the user as the final output, completing the process of translating sign language gestures into a meaningful Italian sentence.

In summary, the flow of data in this system involves several stages, from the initial upload of the video to the final translation. The entire pipeline is designed to be efficient, near real-time, and capable of accurately translating sign language gestures into text in Italian. The use of Redis streams ensures smooth and efficient data flow, while saving keypoints, recognized signs, and translations in the database ensures that the system can track and retrieve valuable data for future processing.

3.3 Cloud Deployment Design

The application was developed using a containerized architecture to enable efficient deployment and scalability on cloud platforms such as Microsoft Azure. Each major component of the system was isolated in its own Docker container, ensuring modularity, ease of maintenance, and consistent runtime environments across development and production settings.

3.3.1 Deployment Strategy on Azure

The system architecture is designed to integrate seamlessly with Azure's cloud services. In a standard deployment setup, containers would be uploaded to **Azure Container Registry (ACR)** and orchestrated using **Azure Kubernetes Service (AKS)** or similar infrastructure.

Supporting services in the deployment include:

- **Azure Blob Storage** for storing video data, keypoints, and predictions.
- **Azure Redis Cache** for managing temporary results and streaming buffers.
- **Azure Database for PostgreSQL** for persistent storage of metadata, sessions, and user inputs.

3.3.2 Scalability and Maintainability

This architecture enables efficient scaling of individual components based on workload (e.g., deploying multiple translation or inference containers under load). It also facilitates maintainability by isolating services, allowing them to be updated or replaced independently as the project evolves.

3.4 Technologies and Techniques Used

3.4.1 Django Framework (Web backend and server-side logic)

The application is built using the Django framework [16], a high-level web framework for Python that promotes rapid development and clean, pragmatic design. Django provides a robust and well-documented foundation for building complex server-side applications, making it particularly suitable for this project, which involves real-time data ingestion, task orchestration, and interaction with machine learning components.

Django's built-in support for model-view-template (MVT) architecture allows for a clear separation of concerns between the user interface, server-side logic, and database interactions. This modularity is essential for managing the multiple stages of the video processing and translation pipeline, such as session tracking, metadata management, and database logging of predictions and translations.

Furthermore, Django offers native support for middleware, authentication, and session management. In this project, these features are leveraged to ensure that each user's uploaded video and its associated frames, predictions, and translations are consistently and securely tracked through unique session IDs and run IDs.

In addition to its synchronous request-handling capabilities, Django is extended using Django Channels to support asynchronous workflows such as WebSocket communication. This makes it a versatile foundation that not only handles traditional HTTP requests but also powers real-time streaming of video frames from the client.

Overall, Django provides a scalable and maintainable core for the backend, with seamless integration into modern Python-based data processing pipelines and compatibility with containerized deployment environments.

3.4.2 Django Channels (Real-time asynchronous communication)

Django Channels [17] is a powerful extension of the Django framework that enables asynchronous communication capabilities, such as WebSocket support, which are not natively available in traditional Django. In this project, Django Channels plays a crucial role in supporting the real-time flow of data between the client and server, particularly for transmitting video frames as they are extracted.

By leveraging Channels, the application is able to handle multiple WebSocket connections concurrently without blocking other server operations. This is achieved through the use of asynchronous consumers that run in an event loop, allowing the system to receive and process incoming frames in real time while other tasks—such as inference or database operations—continue to run independently.

Channels integrates seamlessly with Django's existing request-handling model, enabling the use of familiar routing, middleware, and authentication mechanisms in the context of WebSocket communication. Within this project, each video frame is received by a WebSocket consumer in the `server` app, decoded, and then forwarded to the next stage of the pipeline, such as keypoint extraction or Redis stream storage.

Django Channels is also compatible with Django's session management system, allowing each WebSocket connection to carry metadata such as session and run IDs. This ensures that each set of frames can be correctly grouped and processed in downstream stages.

In summary, Django Channels provides the essential infrastructure for handling real-time, low-latency communication in a scalable and maintainable way, making it an indispensable component of the application's architecture.

3.4.3 WebSocket (Real-Time Communication)

WebSocket [18] is a communication protocol that enables full-duplex, bidirectional communication channels over a single TCP connection. Unlike traditional HTTP, which follows a request-response pattern, WebSocket allows the server and client to send messages to each other at any time without waiting for a response. This makes it particularly well-suited for real-time applications such as video streaming and live data transmission.

In this project, WebSocket is used as the backbone for real-time communication between the client and the server. Once the user uploads or records a video, the client application extracts frames and transmits them incrementally to the server through an open WebSocket connection. This streaming approach minimizes latency, as the server can begin processing frames as soon as they are received, rather than waiting for the entire video file to upload.

The use of WebSocket is essential for maintaining a smooth and interactive user experience. It ensures that gesture recognition and translation can proceed in near real-time, providing quick feedback and minimizing waiting time. Moreover, because WebSocket maintains a persistent connection, it allows the system to handle metadata and control signals alongside frame data, improving the coordination between pipeline components.

The WebSocket connection is handled using Django Channels, which provides asynchronous consumers to receive, process, and route each incoming frame. This architecture supports multiple concurrent WebSocket streams, enabling the system to scale and serve multiple users simultaneously without blocking.

Overall, WebSocket plays a critical role in enabling the real-time, low-latency performance of the system, serving as the communication layer between the video capture front end and the server-side processing pipeline.

3.4.4 Celery (Asynchronous Task Queue)

Celery [19] is an asynchronous task queue used to offload and manage long-running or computationally intensive operations in the system. In this project, Celery plays a central role in handling backend processes such as sign recognition model inference and gloss-to-text translation using Large Language Models (LLMs).

By integrating Celery with Django, the application is able to delegate tasks to background workers that execute independently of the main WebSocket or HTTP request-response cycle. This ensures that the server remains responsive while managing real-time video streaming and avoids blocking operations such as model predictions or data storage.

Each major step in the processing pipeline—such as PCA reduction, model inference, and LLM translation—is executed as a Celery task. These tasks are defined as discrete Python functions that can be queued and processed asynchronously. Celery workers retrieve the tasks from a Redis message broker, execute them in parallel, and return results to be stored or forwarded to the next stage of the pipeline.

This design enables scalability, as the number of Celery workers can be increased based on processing load. It also improves modularity and fault tolerance by isolating heavy computation from the main application thread.

3.4.5 Celery Beat (Periodic Task Scheduler)

Celery Beat [20] is used in conjunction with Celery to schedule periodic or time-based tasks. While Celery manages asynchronous execution, Celery Beat acts as a scheduler that enqueues specific tasks at defined intervals or timestamps.

In this project, Celery Beat is used to periodically check whether a sufficient number of frames have been processed and stored in Redis to trigger the sign recognition task. It acts as a scheduler that monitors the accumulation of PCA-reduced keypoint data within Redis streams and, once a threshold is met, initiates the corresponding Celery task for gesture inference. This mechanism ensures that the recognition process starts only when a complete and meaningful sequence of data is available, enabling structured batch processing without requiring manual intervention or real-time polling.

Beat tasks are defined using Django's scheduling configuration and are executed automatically at predefined intervals. This mechanism ensures that background routines continue to run reliably without manual intervention, supporting the overall robustness and automation of the pipeline.

3.4.6 Poetry (Dependency and Environment Management)

Poetry [21] is used in this project as the primary tool for dependency management and packaging. It provides a modern and reliable approach to managing Python projects by simplifying the handling of libraries, virtual environments, and project metadata. Unlike traditional tools such as `pip` and `requirements.txt`, Poetry ensures a more consistent and reproducible development environment by locking dependencies with precision across platforms and stages of deployment.

In this application, both Django projects, `server` and `client` are managed as a separate Poetry project. This modular setup allows each service to have its own isolated environment, including only the specific dependencies it requires (e.g., machine learning libraries for inference, NLP tools for translation). This structure contributes to better maintainability, reduced risk of dependency conflicts, and smaller, more efficient Docker images.

Poetry also facilitates the publishing and packaging of reusable modules, which is especially useful for the long-term scalability and distribution of the application. It handles virtual environments automatically and ensures that all collaborators or deployment environments can recreate the same dependency setup using a single command (`poetry install`).

By incorporating Poetry into the project's tooling, the development workflow becomes more predictable, the deployment pipeline is more stable, and the application remains easy to scale and maintain as it evolves.

3.4.7 Docker (Containerization)

Docker [22] is an open-source platform that enables applications to be developed, deployed, and run inside containers—lightweight, portable environments that bundle together the application code along with its dependencies, such as runtime, libraries, and system tools. These containers ensure consistency across various systems and stages of development. Unlike virtual machines, containers run on the same operating system kernel as the host, making them more efficient and less resource-intensive.

Docker enables developers to build and distribute applications with predictable behavior, reducing compatibility issues between development and production environments. It also improves scalability by allowing services to be deployed, stopped, or replicated independently.

In this project, Docker is used to containerize the entire backend system, including the Django server, Celery workers, the Redis instance, and the LLM component. Each service runs in its own isolated container, enabling modular development and simplifying orchestration. This setup ensures that all dependencies are properly managed and that the system can be deployed seamlessly across different machines

or platforms without environment-related issues.

3.4.8 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) [23] is a widely used dimensionality reduction technique in machine learning and data preprocessing. It reduces the dimensionality of a dataset by finding the principal components—uncorrelated axes that retain the most significant variation present in the original data. By projecting data onto these components, PCA reduces noise and redundancy while preserving the most significant patterns.

In this project, PCA is applied to the extracted keypoints from each video frame to reduce the input dimensionality before gesture recognition. The original keypoints include coordinates from the face, hands, and upper body, resulting in a high-dimensional vector. PCA is used to retain only the most informative components, significantly decreasing computational load and improving model performance. Additionally, this step helps mitigate overfitting by removing less relevant variations in the data.

3.4.9 OpenCV (Computer Vision Library)

OpenCV (Open Source Computer Vision Library) [24] is an open-source toolkit widely used for image and video processing tasks. It provides a comprehensive suite of tools for reading, manipulating, and analyzing visual data in real time. OpenCV is written in C++ with bindings for Python, making it both fast and accessible for integration with machine learning workflows.

In this project, OpenCV is used within the client application to handle video processing. It is responsible for reading uploaded video files and extracting individual frames, which are then serialized, compressed, and sent to the server for further processing. OpenCV ensures efficient frame handling and preprocessing, enabling the system to process video input in real time without introducing latency or bottlenecks.

3.4.10 MediaPipe (Keypoint Extraction)

MediaPipe [25] is an open-source framework developed by Google for building multimodal machine learning pipelines, with a strong focus on real-time perception tasks such as face detection and pose estimation. It includes pre-trained models optimized for running efficiently on both CPUs and GPUs, making it suitable for edge devices and real-time systems.

In this project, MediaPipe is used on the server side to extract 3D keypoints (landmarks) from the video frames. Specifically, it identifies and tracks 33 pose

landmarks, 468 face landmarks, and 21 landmarks for each hand, producing coordinate data in three dimensions (x, y, z). These keypoints serve as the foundation for sign recognition. To prepare the data, the coordinates are flattened into one-dimensional arrays and further reduced using PCA. MediaPipe's robust and lightweight models make it an ideal choice for integrating real-time gesture analysis into the pipeline.

3.4.11 Redis (Real-Time Data Management)

Redis [26] (Remote Dictionary Server) is an open-source, in-memory data structure store used as a database, cache, and message broker. It supports various data types such as strings, hashes, lists, sets, and more advanced structures like streams. Its in-memory design enables extremely fast read and write operations, making it a popular choice for high-performance applications requiring real-time responsiveness.

In this project, Redis serves two primary roles: it functions both as the message broker for Celery and as a temporary data store for intermediate results between pipeline stages. Redis Streams are specifically used to hold compressed frame data, PCA-reduced keypoints, inference outputs, and translation results. Each stage in the pipeline reads from one stream and writes to another, enabling asynchronous and decoupled processing of data in real time.

Redis also plays a key role in enabling fault tolerance and traceability, as each message in the stream is timestamped and can be replayed if necessary. Its integration with both Django and Celery makes Redis a reliable and high-performance backbone for real-time data flow throughout the system.

3.4.12 PostgreSQL (Relational Database)

PostgreSQL [27] is an open-source relational database management system (RDBMS) known for its reliability and adherence to SQL standards. It supports advanced features such as ACID transactions, indexing, full-text search, and JSON storage, making it suitable for both structured and semi-structured data.

In this project, PostgreSQL is used as the primary database for persistent storage of application data. It stores information related to user sessions, video metadata, extracted keypoints, PCA-reduced vectors, recognized glosses, and translated Italian sentences. Custom Django models are used to define database tables, and PostgreSQL handles the transactional consistency required to keep the processing pipeline synchronized and reliable.

Its compatibility with Django's Object-Relational Mapping (ORM) layer allows for seamless integration between Python code and database operations. PostgreSQL's support for complex data types, such as binary fields and JSON fields, is

particularly useful for storing compressed video frames and structured keypoint data within the same relational schema.

3.5 Conclusion

The LIS-to-Italian Translation App was developed as a modular and scalable system, leveraging real-time communication, machine learning, and cloud-friendly technologies to enable efficient sign language translation. From the beginning, the architectural design focused on ensuring that each component could operate independently while still contributing to a seamless end-to-end experience for users. This approach enables both flexibility and extensibility, allowing for individual components to be replaced, improved, or scaled without requiring changes to the entire system.

Key technologies were integrated to support these objectives. Real-time interaction was made possible through the use of WebSocket connections and Django Channels, enabling asynchronous communication between the client and server. This setup ensures that video frames can be transmitted and processed with minimal delay, which is crucial for maintaining a responsive translation experience. Redis was used to handle temporary, in-memory storage of keypoint data and intermediate results, providing the speed and efficiency needed for smooth data handling in a real-time pipeline.

To support maintainability and deployment flexibility, all major system components were developed as Docker containers. This containerized design allows each service—such as the backend server, WebSocket interface, PCA module, gloss prediction model, and translation engine—to be managed and deployed independently. The use of Docker not only simplifies development and testing but also ensures consistency across environments.

The application is designed to be efficiently hosted in the cloud. With minimal configuration changes, the system can be deployed to platforms like Microsoft Azure using Azure Kubernetes Service (AKS). Kubernetes enables automated orchestration, load balancing, and scaling of the containers, making the system adaptable to varying levels of user demand. Supporting services such as Azure Blob Storage, Redis Cache, and PostgreSQL further enhance the system's robustness and data persistence capabilities.

The combination of these technologies results in a highly responsive and modular translation system. The real-time architecture, supported by efficient data streaming and fast inference, enables the translation of sign language videos with minimal latency. Moreover, the modular design ensures that the system is not only functional but also extensible—ready to accommodate future improvements such as enhanced models, larger datasets, or multilingual support.

In summary, the LIS-to-Italian Translation App demonstrates how modern web technologies, containerization, and machine learning can be combined to address the challenge of sign language translation. It lays a solid foundation for future development and scaling, aiming to become a practical and impactful tool for improving communication accessibility for the deaf and hard-of-hearing community in Italy.

Chapter 4

Creating the Dataset

4.1 Data Generation with GPT-4

4.1.1 Data Scarcity in Italian Sign Language (LIS)

Data scarcity is a significant challenge in the development of sign language processing systems, particularly for Italian Sign Language (LIS). Unlike spoken languages, sign languages like LIS do not have large, publicly available annotated datasets. This limited availability of data for training machine learning models impedes the development of automatic systems for sign recognition, translation, and gloss generation. The lack of sufficient data makes it difficult to train accurate models, which ultimately limits their performance in real-world applications aimed at assisting the deaf and hard-of-hearing communities.

4.1.2 Manual Extraction of Grammar Rules

To overcome the data scarcity issue, we manually extracted 104 key grammatical rules from the book *A Grammar of Italian Sign Language (LIS)*, which is a crucial resource for understanding the syntax and structure of LIS. These rules cover various linguistic aspects such as sentence structure, verb usage, and word order. The extraction process allowed us to define clear rules that would help generate meaningful glosses, which are essential for training sign language models.

For example, a common rule in LIS is that the Subject-Verb-Object (SVO) structure is typically followed in declarative sentences. This can be illustrated with the following example:

- Spoken Italian: *Mamma mangia la mela.* (Mom eats the apple.)
- Glossed LIS: *Mamma mela mangaire.* (Mom apple eat.)

In this example, the subject *Mamma* (Mom), the object *mela* (apple), and the verb *mangaire* (eat) are arranged according to the typical SVO structure in

LIS, though the word order may vary slightly due to the unique properties of sign language grammar.

4.1.3 Synthetic Data Generation with GPT-4

After extracting the necessary rules from *A Grammar of Italian Sign Language (LIS)*, we used GPT-4 to generate synthetic data for LIS glosses. This process involved providing GPT-4 with specific prompts designed to generate LIS translations that adhere to the rules we had manually extracted.

For example, we instructed GPT-4 to generate at least 10 examples for each rule, using the examples we had provided as guidance to ensure the outputs were consistent with the manual rules we had extracted. The prompt would be applied iteratively for different rules, ensuring that the generated sentences adhered to the desired structure.

Using this approach, GPT-4 generated multiple variations of glosses such as:

- *Mamma mela mangiare.* (Mom apple eat.)
- *Fratello libro leggere.* (Brother book read.)
- *Sorella bicicletta pedalare.* (Sister bicycle pedal.)

This GPT-based generation strategy enabled us to synthesize approximately 1000 Gloss–Italian sentence pairs, significantly expanding the available training data. The use of GPT for this task is supported by findings from recent work demonstrating the viability of large language models for high-quality synthetic data generation across structured domains [13]. By combining rule-based constraints with generative language models, we were able to preserve grammatical consistency while achieving diversity in content.

4.2 Data Augmentation

Data augmentation plays a vital role in enhancing the performance of the LIS-to-Italian translation model. Given the limited availability of annotated datasets in Italian Sign Language (LIS), we implemented multiple augmentation strategies to improve generalization, increase linguistic diversity, and expose the model to a broader range of sentence structures and vocabulary. The following subsections detail each augmentation technique used in this project.

4.2.1 Domain Data Augmentation

To improve domain-specific coverage, we generated synthetic gloss-translation pairs using prompt-based generation with GPT-4. This allowed us to create examples across a variety of domains such as medicine, work, education, and technology.

The model was instructed to maintain the original meaning while varying grammar and vocabulary to reflect realistic usage in these domains. Each generated sample followed the format presented by the examples in Table 4.1

Gloss	**Italian Sentence**
STAGISTA OSPEDALE LAVORARE ESPERIENZA ACQUISIRE	Lo stagista lavora in ospedale e acquisisce esperienza.
SMART_WORKING COMODO, MA COMUNICAZIONE SFIDA	Lo smart working è comodo, ma la comunicazione è una sfida.
DOTTORE PAZIENTE VISITA ATTENTA FARE	Il dottore effettua una visita attenta al paziente.
STUDENTI PROGETTO GRUPPO INSIEME COMPLETARE	Gli studenti completano insieme il progetto di gruppo.
PROGETTO COMPLETARE PRIMA SCADENZA IMPORTANTE	È importante completare il progetto prima della scadenza.

Table 4.1: Examples of gloss-italian sentence pairs created through domain data augmentation

A total of 500 new examples were generated using this method, including:

- 120 examples in the domain of medicine
- 110 examples in the domain of work
- 135 examples in the domain of education/studies
- 135 examples in the domain of technology and digital communication

This helped improve the model's understanding of specific terminologies and structures associated with each context.

4.2.2 Paraphrasing via Italian-English-Italian Back Translation

Inspired by some augmentation techniques discussed in the paper [14], we implemented a paraphrasing strategy using back translation.

The process involved translating LIS gloss translations (in Italian) into English using a pre-trained machine translation model (Helsinki-NLP/opus-mt-it-en), and then back into Italian using the reverse model (Helsinki-NLP/opus-mt-en-it). This introduced paraphrased variants of the original translations while preserving semantic meaning.

Some representative examples from the gloss-to-Italian translation dataset are reported in Table 4.2, showcasing the semantic alignment between LIS glosses, original Italian translations, and their corresponding English meaning back-translated to Italian again.

Gloss (LIS)	**Italian (Original)**	**English Translation**	**Italian (Paraphrased)**
NONNA TORTA PREPARARE	La nonna prepara una torta.	Grandma makes a cake.	La nonna fa una torta.
TU SPESA FARE DOVERE	Tu devi fare la spesa.	You have to go shopping.	Devi andare a fare shopping.
RAGAZZO ESAME FALLIRE PERCHÉ	Perché il ragazzo ha fallito l'esame?	Why did the boy fail the exam?	Perché il ragazzo ha saltato l'esame?
PRESENTAZIONE PRONTO ESSERE, TEAM SODDISFATTO SENTIRE IX3a	Quando la presentazione è pronta, il team si sente soddisfatto.	When the presentation is ready, the team feels satisfied.	Quando la presentazione è pronta, la squadra si sente soddisfatta.
LORO ESERCIZI FARE DOVERE	Loro devono fare gli esercizi.	They have to do the exercises.	Devono fare gli esercizi.

Table 4.2: Examples of paraphrased Italian sentences obtained via back-translation

Only paraphrased results that differed from the original translation were retained. This method generated over 300 unique paraphrased entries.

4.2.3 Targeted Augmentation Using 147 Key Glosses

To strengthen the model's familiarity with a core vocabulary of 147 frequently used glosses (also used in the sign recognition model), we applied targeted prompt-based augmentation.

The prompts were constructed to:

- Preserve the original gloss meaning

- Include at least one of the 147 target words in each sentence
- Vary grammatical structure and reflect diverse, everyday scenarios

Example Prompt Instruction:

> "Ogni coppia deve includere almeno una volta una delle seguenti parole: casa, scuola, lavoro, medico, telefono..."

This augmentation technique added over 400 high-quality gloss-translation pairs, enhancing the robustness of the model in handling key glosses across different sentence structures.

4.2.4 Final Dataset

The final dataset used for training the models consisted of glosses extracted from Italian Sign Language (LIS) along with their corresponding Italian translations. This augmented dataset was created by generating synthetic data using GPT-4, ensuring a diverse set of glosses and translations. The combination of the original dataset and the synthetically generated augmented data allowed the models to learn a wider range of translation patterns.

All augmentation techniques contributed significantly to dataset expansion. In total, more than 1,200 new gloss-translation pairs were added to the original dataset:

- 400 pairs from domain-specific generation
- 700 pairs from back-translation and paraphrasing
- 166 pairs from targeted augmentation using key glosses

An example of a training data pair included in the augmented dataset is:

- **Gloss:** STUDENTE SCUOLA LIBRO LEGGERE
  Italian: Lo studente legge un libro a scuola.

4.3 Technologies Used in Synthetic Data Generation

This chapter outlines the tools and technologies employed in the generation and augmentation of synthetic data.

- **OpenAI API** [28]: Used to generate synthetic gloss and Italian sentence pairs through GPT models by carefully designed prompting strategies.

- **Transformers Library** [29]: Provided the interface for leveraging large pre-trained transformer models, including models useful for zero-shot and few-shot tasks.
- **Datasets Library** [30]: Facilitated loading and manipulating datasets in Hugging Face format, as well as integrating custom JSON and CSV files.
- **Pandas** [31]: Used for cleaning and reshaping textual data into structured formats suitable for processing and generation.
- **NumPy** [32]: Supported numerical and array operations necessary for pre-processing and data manipulation.
- **JSON and File I/O** [33]: Enabled structured data exchange between different components and persistent storage of synthetic examples.

Chapter 5

Model Fine-Tuning

To train a high-quality translation model for converting Italian Sign Language (LIS) glosses into Italian text, we fine-tuned two state-of-the-art multilingual models: **mBART** and **MarianMT**. This chapter outlines the motivations for choosing these models, their internal architectures, the training methodology, and the performance comparison between training on the original dataset and the augmented dataset.

5.1 Motivation for Choosing mBART and MarianMT

Both mBART and MarianMT are encoder-decoder models designed for machine translation tasks. We chose these models for the following reasons:

- **Multilingual Capability:** LIS is a gloss-based system with simplified grammar and vocabulary. Both mBART and MarianMT can generalize across multilingual contexts, making them suitable for handling simplified LIS structures translated into natural Italian.

- **Pre-trained on Large Corpora:** These models are pre-trained on large multilingual corpora, giving them a strong baseline understanding of syntax and semantics, which reduces the data requirements for fine-tuning.

- **Open-Source and HuggingFace Integration:** Both models are available through HuggingFace's Transformers library, simplifying the integration and fine-tuning pipeline.

5.2 Understanding the Models

5.2.1 mBART (Multilingual BART)

mBART is a sequence-to-sequence denoising autoencoder for pretraining multilingual translation models. It is based on the BART architecture, which combines the benefits of BERT (for understanding) and GPT (for generation).

- **Architecture:** mBART uses a Transformer encoder-decoder structure. During pretraining, it corrupts input text and learns to reconstruct the original sentence.
- **Input Format:** Inputs are tokenized using a sentencepiece model and prefixed with a language token (e.g., ___ita___ for Italian).
- **Advantages:** Because of its denoising pretraining and multilingual focus, mBART is well-suited for tasks like gloss translation where the input structure is often non-standard.

5.2.2 MarianMT

MarianMT is a multilingual translation model trained on OPUS data. It provides language-specific models and supports a wide range of low-resource language pairs.

- **Architecture:** MarianMT is based on a Transformer encoder-decoder framework optimized for speed and translation quality.
- **Language Tokens:** Unlike mBART, MarianMT does not require special language tokens at the beginning of the sequence.
- **Advantages:** It is lightweight and faster to fine-tune, making it an excellent choice for comparison with mBART.

5.3 Fine-Tuning Methodology

The models were fine-tuned using the HuggingFace Trainer API. Training was conducted in two phases: once on the **initial dataset without augmentation**, and once on the **final dataset with all three augmentation strategies**.

5.3.1 Data Preprocessing

- Tokenization was performed using the respective model's tokenizer.

- Glosses were treated as source sentences, and their Italian translations as targets.
- Datasets were split into 90% training and 10% evaluation.

5.3.2 Training Configuration

Due to the limited computational resources available during experimentation (using platforms like Google Colab), the training parameters were selected to minimize memory usage while maintaining experimental rigor. Although these settings are suboptimal for large-scale production training, they allowed us to test different architectures effectively.

mBART Training Arguments:

- Learning Rate: 2e-5
- Train Batch Size: 2
- Eval Batch Size: 1
- Gradient Accumulation Steps: 4
- Number of Epochs: 3
- Weight Decay: 0.01
- Mixed Precision: Enabled (`fp16=True`)

MarianMT Training Arguments:

- Learning Rate: 2e-5
- Train Batch Size: 2
- Eval Batch Size: 1
- Gradient Accumulation Steps: 4
- Number of Epochs: 3
- Weight Decay: 0.01
- Mixed Precision: Enabled (`fp16=True`)
- Reporting: Disabled `report_to="none"`

Despite the resource constraints, these configurations were sufficient to demonstrate trends in model behavior and validate the effectiveness of the augmentations. For improved performance, future training could benefit from larger batch sizes, more epochs, and increased GPU memory.

5.4 Technologies Used in Fine-Tuning and Evaluation

This chapter presents the tools used during the fine-tuning of translation models and the evaluation of their performance.

5.4.1 Model Fine-Tuning

- **Transformers Library** [29]: Enabled fine-tuning of models such as `MarianMTModel`, `MBartForConditionalGeneration`, and `AutoModelForSeq2SeqLM` using `Trainer` and `Seq2SeqTrainer`.
- **MarianMT** [34]: Used as a baseline in the fine-tuning process.
- **mBART** [35]: A multilingual sequence-to-sequence model used for fine-tuning on gloss-to-Italian translation.
- **Torch (PyTorch)** [36]: Served as the underlying framework for model training and GPU-based optimization.

Chapter 6

Results and Comparative Analysis of the Fine-Tuned Models

This section consolidates the results of fine-tuning experiments conducted using the mBART and MarianMT models on both the initial and augmented LIS-to-Italian datasets. The evaluation includes training metrics, semantic similarity scores (BERTScore), and qualitative observations to assess the effectiveness of data augmentation and model selection.

6.1 Evaluation Metrics

Model performance was evaluated using the following:

- **Training Loss:** Indicates how well the model fits the training data. While not used for final evaluation, it helps detect convergence and overfitting.
- **Validation Loss:** Measures the model's generalization during training and is used for model selection.
- **BERTScore:** Captures semantic similarity between generated and reference sentences.
  - F1: Harmonic mean of precision and recall
  - Precision: Semantic similarity of generated text to ground truth
  - Recall: Coverage of reference content in the generated sentence

Why BERTScore? While traditional metrics like BLEU and ROUGE focus on surface-level overlap (e.g., n-gram matches), BERTScore compares contextual embeddings of generated and reference sentences using pre-trained transformer models. This makes it especially suitable for evaluating paraphrased and semantically correct outputs — which are common in LIS-to-Italian translations due to gloss simplifications. Thus, BERTScore better reflects the actual quality and meaning of translations in this context.

6.1.1 Evaluation Metrics

Model performance was evaluated using the following:

- **Validation Loss:** Measures the model's generalization during training.
- **BERTScore:** Captures semantic similarity between generated and reference sentences.
  - F1: Harmonic mean of precision and recall
  - Precision: Semantic similarity of generated text to ground truth
  - Recall: Coverage of reference content in the generated sentence

6.2 Summary of Results

Model	Dataset	Val Loss	BERTScore F1	Precision / Recall
mBART	Initial	0.1191	0.8772	0.8798 / 0.8751
MarianMT	Initial	0.3238	0.7596	0.7699 / 0.7584
mBART	Augmented	**0.1098**	**0.8888**	**0.8923 / 0.8858**
MarianMT	Augmented	0.3469	0.7696	0.7781 / 0.7620

Table 6.1: Comparison of model performance on initial vs. augmented datasets

6.2.1 Observations and Analysis

1. Impact of Augmentation: Augmentation significantly boosted model performance across both architectures. mBART showed improved validation loss and BERTScore metrics after training on the augmented dataset. While MarianMT showed some improvement in BERTScore, its validation loss did not decrease further, suggesting it may have reached its optimal performance on simpler data.

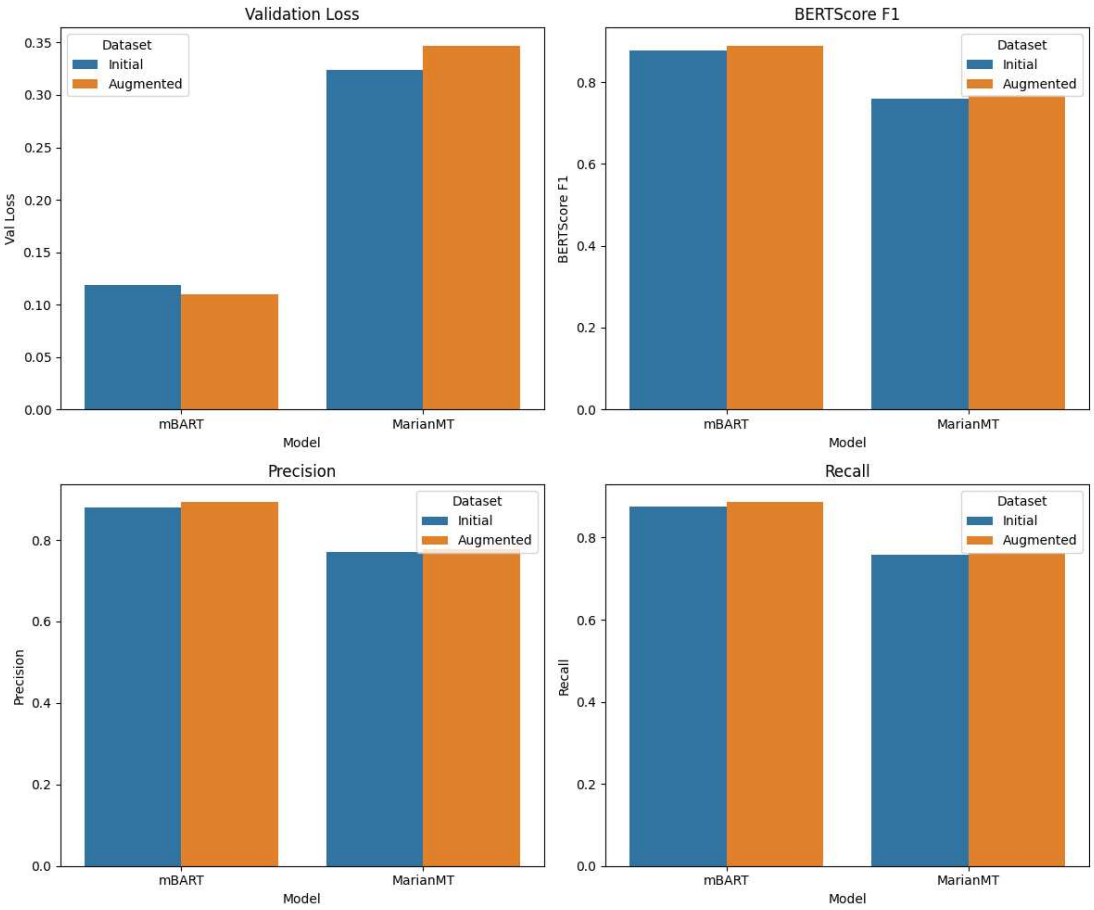

Figure 6.1: Model performance comparison for initial vs. augmented datasets

2. mBART vs. MarianMT:

- **mBART:** Delivered the best results overall with the augmented dataset, achieving the lowest validation loss (0.1098) and highest semantic quality (BERTScore F1 of 0.8888).

- **MarianMT:** Despite performing well on the initial dataset with a low validation loss (0.3238), it did not benefit significantly from data augmentation in terms of validation loss or BERTScore.

3. Training Stability: Training loss for all models steadily decreased, indicating that convergence was achieved. However, the validation results were more informative in assessing generalization.

4. Best Configuration: The most effective configuration was mBART trained on the augmented dataset, which demonstrated strong generalization (low validation loss) and semantic richness (highest BERTScore F1 and Precision).

6.3 Translations with MarianMT Trained on Augmented Data

As shown in Table 6.2, the model produces syntactically or semantically flawed Italian sentences.

Gloss	**Incorrect Italian Translation**
STUDENTE SCUOLA LIBRO LEGGERE	Il studenti ha leggere il scuola.
MEDICO OSPEDALE PAZIENTE CURARE	Il medico hosiedale, curare il paziente.
STUDENTE TESI INVIARE_SMS	Il studenti ha inviato il tesi.
OGGI APERTO SPORTELLO	L'annosto è un programmato.
POMERIGGIO COMMISSIONE LETTERA INVIATO NON	Il pomeriggio, non-envoyed letter.
STELLE CIELO SPLENDERE	Il stelle splende il cielo.
DOMANI NOI CASA NUOVA TRASLOCO	Tumani noi casa è nuova trasloco.
OGNI GIORNO STUDENTE SCUOLA ANDARE	Each day, a student goes to school.
MEDICO OSPEDALE LAVORARE	Hospital Medico è lavori.
MARIA CIBO CUCINARE LUCA MANGIARE	Maria cucinare il cucinare il luca.

Table 6.2: Examples of LIS glosses with incorrect Italian translations

A possible hypothesis for MarianMT's underperformance in LIS gloss-to-Italian translation lies in its training paradigm and architecture. MarianMT is optimized for clean, grammatically correct sentence pairs in specific language directions, and lacks pretraining on noisy or low-resource inputs like glosses. Since LIS glosses are not full natural language sentences—they omit function words, lack inflection, and often use non-standard word order—MarianMT's encoder-decoder architecture may struggle to derive meaningful semantic representations from them. As a result, MarianMT exhibits poor generalization when fine-tuned on gloss-style inputs, making it less suitable for this specific translation task.

6.4 Translations with mBART Trained on Initial Data

The results from the mBart trained on the inital dataset are respresented in Table 6.3.

Gloss	**Italian Translation (mBART)**
STUDENTE SCUOLA LIBRO LEGGERE	Lo studente legge un libro in scuola.
MEDICO OSPEDALE PAZIENTE CURARE	Il medico ospedaliale cura il paziente.
STUDENTE TESI INVIARE_SMS	Lo studente invita a scrivere i suoi messaggi.
OGGI APERTO SPORTELLO	Oggi è aperto il parco.
POMERIGGIO COMMISSIONE LETTERA INVIATO NON	La commissione non ha invitato la lettera al giorno.
STELLE CIELO SPLENDERE	La stella splende sul cielo.
DOMANI NOI CASA NUOVA TRASLOCO	Domani ci sarà una nuova casa.
OGNI GIORNO STUDENTE SCUOLA ANDARE	Ogni giorno lo studente va a scuola.
MEDICO OSPEDALE LAVORARE	Il medico lavora in ospedale.
MARIA CIBO CUCINARE LUCA MANGIARE	Maria cucina il cibo e Luca mangia.

Table 6.3: Gloss-to-Italian translations using mBART on the initial dataset

Despite mBART's superior performance over MarianMT, the outputs generated using the initial dataset reveal that the model still produces semantically and syntactically flawed translations in several cases. While simpler glosses yield accurate results (e.g., "OGNI GIORNO STUDENTE SCUOLA ANDARE"), more complex or abstract sequences like "POMERIGGIO COMMISSIONE LETTERA INVIATO NON" lead to incorrect word choices and ungrammatical phrases. This suggests that although mBART generalizes better to low-resource inputs, its effectiveness is still limited by the quality and diversity of the training data, emphasizing the need for further domain-specific fine-tuning and augmentation.

6.5 Final Translations with mBART Trained on Augmented Data

The final translations produced by mBART trained on the augmented dataset are are shown in Table 6.4.

Gloss	**Italian Translation**
STUDENTE SCUOLA LIBRO LEGGERE	Lo studente legge un libro a scuola.
MEDICO OSPEDALE PAZIENTE CURARE	Il medico ha curato il paziente in ospedale.
STUDENTE TESI INVIARE_SMS	Lo studente ha inviato un messaggio.
OGGI APERTO SPORTELLO	Oggi è ovvio e sportivo.
POMERIGGIO COMMISSIONE LETTERA INVIATO NON	Nel pomeriggio, non ho mandato la lettera.
STELLE CIELO SPLENDERE	Le stelle splendeno nel cielo.
DOMANI NOI CASA NUOVA TRASLOCO	Domani abbiamo una nuova casa in città.
OGNI GIORNO STUDENTE SCUOLA ANDARE	Tutti i giorni lo studente va a scuola.
MEDICO OSPEDALE LAVORARE	Il medico lavora in ospedale.
MARIA CIBO CUCINARE LUCA MANGIARE	Maria cucina il cibo e Luca mangia.

Table 6.4: Examples of LIS gloss sequences and their Italian translations

These translations highlight the model's ability to capture the essence of the glosses and translate them into coherent Italian sentences. However, some of the translations, such as "Oggi è ovvio e sportivo" (for "OGGI APERTO SPORTELLO"), show some deviation from the expected translation, which could indicate a need for further fine-tuning or improvement in translation quality for certain glosses.

6.6 Technologies Used in Evaluation

6.6.1 Evaluation and Metrics

- **Evaluate Library** [37]: Used to compute translation metrics and track training performance.
- **BERTScore** [38]: Employed for evaluating semantic similarity between predicted translations and references.

6.7 Conclusion

The experiments confirm that data augmentation significantly improves translation performance for LIS glosses. Both mBART and MarianMT show noticeable gains when trained on enriched datasets; however, mBART consistently outperforms MarianMT in this context. This performance gap can be attributed to fundamental differences in their architecture and pretraining. MarianMT is designed for clean, bilingual sentence pairs and lacks robustness to non-standard input formats like glosses, which often omit grammatical markers and follow unique syntactic rules. In contrast, mBART benefits from multilingual denoising pretraining, allowing it to better handle noisy, partial, or unstructured sequences. Its flexibility in multilingual and low-resource scenarios makes it particularly well-suited for semantically sparse inputs such as LIS glosses. Consequently, mBART proves to be the more effective model for LIS-to-Italian translation, especially in the presence of data augmentation.

Chapter 7

End-to-End Testing and Evaluation

7.1 Overview

To evaluate the effectiveness and integration of the entire LIS-to-Italian translation system, we conducted an end-to-end test using a controlled set of video inputs. The purpose of this test was to validate that all stages—from video upload to final natural language output—functioned correctly and in coordination, even under constrained computational resources.

7.2 Test Methodology

Three LIS video clips were selected for testing. Each clip features a single isolated sign gesture and was processed through the complete pipeline in sequence. The pipeline stages included:

- Frame-by-frame extraction and WebSocket streaming
- Keypoint detection via Mediapipe
- PCA-based dimensionality reduction
- Gloss prediction from reduced keypoints
- Final gloss-to-Italian translation using the fine-tuned mBART model

7.3 Test Videos

The three videos used in the evaluation each contained a single LIS sign gesture, allowing for precise tracking of the pipeline's behavior and outputs. Table 7.1 summarizes their characteristics and a representative frame from each video is shown in Figure 7.1.

- **FPS (Frames Per Second)** refers to how many frames are captured per second of video. A higher FPS indicates smoother motion.
- **Resolution** indicates the size of the video frame in pixels (width × height).
- **Frames** is the total number of individual frames in the video.
- **Duration** is the total length of the video in seconds.

Sign Represented	FPS	Resolution	Frames	Duration (s)
Studente	25.0	720x576	104	4.16
Modulo	25.0	720x576	104	4.16
Consegnare	25.0	720x576	107	4.28

Table 7.1: Metadata and signs for tested LIS videos

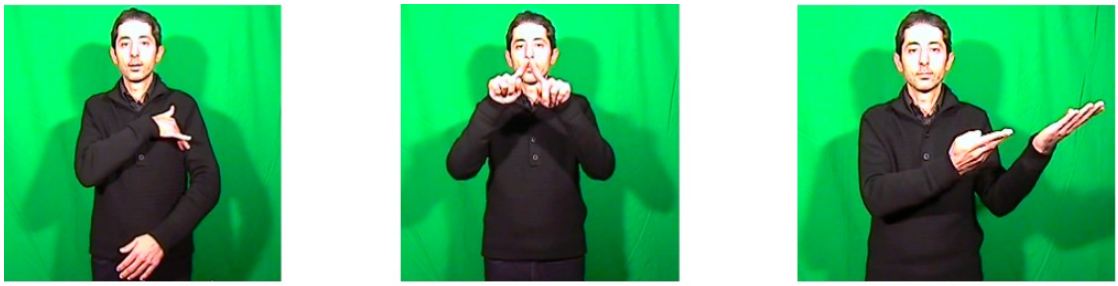

Figure 7.1: Sample input frames from the three test videos

7.4 Results

The end-to-end test validated the complete functionality of the pipeline. Each video was correctly processed, and the corresponding sign was accurately recognized. Specifically, the system successfully detected and identified the signs:

- **STUDENTE**
- **MODULO**
- **CONSEGNARE**

After the recognition phase, the identified glosses were combined and passed through the fine-tuned mBART model to generate a fluent Italian sentence. The final translated output was:

Lo studente consegna il modulo.

This demonstrates the system's ability to not only recognize isolated signs but also assemble them into a semantically coherent and grammatically correct sentence. It confirms that the integration between keypoint-based inference and the language model component functions effectively for structured inputs.

7.5 System Performace and Limitations

The total time required to process all three LIS videos—from initial upload to the generation of the final translated sentence—was approximately **52 seconds**. This end-to-end delay includes all stages of the pipeline executed sequentially: video frame extraction and streaming, keypoint extraction via Mediapipe, PCA-based dimensionality reduction, gloss prediction for each individual sign, and final translation of the combined glosses into natural Italian using a fine-tuned mBART model.

It is important to note that this test was conducted on a local machine with limited computational resources, without GPU acceleration or parallelization. Although the system is designed with asynchronous processing capabilities—allowing different components to run concurrently where possible—the overall processing was still limited by CPU-bound tasks, I/O delays, and the absence of full parallelization or cloud-scale infrastructure. These factors collectively contributed to the observed latency.

Despite the longer processing time, the experiment successfully validates the system's architecture and functional correctness. With adequate cloud-based infrastructure, GPU acceleration, and parallel processing of frames, the total latency could be substantially reduced. Moreover, considering the linguistic structure of LIS—where the full sequence of signs must be completed before translation begins—this delay is not disruptive to user comprehension, and the system can still be considered *near real-time* for short utterances.

7.6 Conclusion

This test demonstrates that the entire pipeline—from video input to Italian sentence output—operates as expected. While further optimization and scalability testing are needed, particularly for longer sequences, the current results confirm that the system can handle end-to-end translation in near real-time.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

This work presented an end-to-end system for translating Italian Sign Language (LIS) glosses into Italian, combining real-time data processing with modern machine learning techniques. The architecture was designed with concurrency and asynchronicity in mind: the system follows a client-server model where video frames are captured on the client side and streamed in real time to the server via WebSocket. On the server, asynchronous components process incoming frames—extracting keypoints using Mediapipe, reducing dimensionality through PCA, and storing intermediate data in Redis streams for efficient, non-blocking access. This modular architecture enables different stages of the pipeline (such as frame reception, preprocessing, inference, and translation) to run concurrently, supporting responsive and scalable real-time performance.

Beyond infrastructure, this study focused on developing machine learning models for LIS-to-Italian translation, addressing a core challenge in the field: the scarcity of annotated datasets. To overcome this limitation, we implemented a synthetic data generation pipeline using GPT-4, producing over 1,200 additional gloss-translation pairs. This expanded dataset was essential in training more robust and generalizable models.

The initial dataset consisted of real-world LIS glosses and their translations, serving as the foundation for training. However, given the limited scope and linguistic coverage of the original data, synthetic augmentation was applied through techniques such as domain-specific generation, back-translation, and controlled rule-based gloss construction. These methods enriched the dataset both in size and linguistic diversity, enabling the models to better handle a broader range of expressions and grammatical structures.

Experimental results showed that the models effectively learned complex translation patterns, especially when trained on augmented data. Although some challenges were observed—particularly with the MarianMT model, which underperformed compared to mBART—the results provided useful insights into the impact of model architecture and data diversity. In particular, mBART demonstrated strong performance gains, with lower validation losses and higher BERTScore F1 scores when trained on the synthetically expanded dataset.

Finally, this thesis highlights the significant potential of combining synthetic data generation with fine-tuned multilingual models to advance the field of sign language translation. While further work is needed to extend real-time capabilities and scale testing, this work lays the groundwork for future systems that are not only linguistically accurate but also architecturally optimized for real-time and cloud-based deployment in low-resource sign languages such as LIS.

8.2 Future Work

While this research has provided important insights, several areas remain for future exploration, which could further enhance the quality and applicability of sign language translation models. The following outlines key directions for future work:

8.2.1 Scalability and Performance Optimization

While the current system demonstrates promising results on shorter sequences, future work will involve large-scale testing to evaluate performance under realistic usage conditions. This includes processing longer video sequences, handling continuous signing, and managing simultaneous requests from multiple users.

To support such demands, the application architecture can be further optimized for scalability. Key areas of focus include:

- **Dynamic Worker Allocation:** Adjusting the number of Celery workers and concurrency settings to better match incoming traffic can improve throughput and reduce latency, particularly during peak usage.

- **Asynchronous Task Queue Management:** Prioritizing tasks (e.g., real-time frames vs. batch uploads) and monitoring task queues can help balance real-time responsiveness with overall system load.

- **Load Balancing and Horizontal Scaling:** Deploying multiple containers for the PCA and inference services behind a load balancer (e.g., using Kubernetes Horizontal Pod Autoscaler) can ensure resilience and efficient resource usage.

- **GPU Acceleration:** Offloading compute-intensive operations such as keypoint extraction and model inference to GPU-enabled nodes can significantly reduce per-request latency.

These enhancements will be critical for transitioning the system from a prototype to a production-ready application capable of real-time LIS translation at scale.

8.2.2 Improving the Dataset Quality

Although we successfully augmented the dataset, the quality of the synthetic data could be further refined. Currently, GPT-4 generates translations based on gloss-to-text pairs, but there are still challenges related to the precision and contextual accuracy of these translations. Future work could focus on using a combination of machine-generated and human-curated data to improve the accuracy and reliability of the dataset. In particular, it would be useful to engage native sign language interpreters in the dataset creation process to ensure the linguistic and cultural context of the translations is accurately captured.

8.2.3 Exploring Other Data Augmentation Techniques

While we focused on domain-specific generation, back-translation, and targeted augmentation, other data augmentation techniques could be explored. For instance, exploring paraphrasing at the sentence level or augmenting data through the use of advanced techniques like style transfer or adversarial training could help improve model robustness. Additionally, incorporating multimodal data (e.g., visual and textual input) could enhance the translation quality by allowing the models to better interpret the meaning behind each gloss in context.

8.2.4 Expanding to Other Sign Languages

While this work focused on Italian Sign Language (LIS), the methodology can be expanded to other sign languages, further improving cross-linguistic machine translation models. Given the wide variety of sign languages, it would be valuable to adapt the techniques used in this thesis to other languages, such as American Sign Language (ASL) or British Sign Language (BSL). Additionally, developing multilingual sign language translation systems that can handle multiple sign languages simultaneously would be an important step toward creating universal translation tools for the deaf and hard-of-hearing community.

8.2.5 Collaboration with the Deaf Community

Future efforts should involve collaboration with the Deaf community to improve both dataset creation and model evaluation. Engaging with Deaf sign language users can provide invaluable insights into the nuances of sign language that may not be captured by automated processes. This collaboration could include feedback loops where the community provides corrections to the model's outputs, enabling more accurate and contextually appropriate translations.

8.3 Final Remarks

This thesis presented the development of a complete end-to-end system for translating Italian Sign Language (LIS) into written Italian, with a strong focus on system architecture and real-time processing. The application was built with a modular and asynchronous design to ensure flexibility, maintainability, and responsiveness. Each component was developed to operate independently, enabling concurrent processing and laying the groundwork for future scalability and deployment in cloud environments.

At the same time, the project addressed one of the major limitations in this field: the lack of annotated LIS datasets. To overcome this, we explored synthetic data generation techniques using large language models such as GPT-4 and applied multiple strategies to expand and diversify the training data. This resulted in improved model performance, especially in terms of handling uncommon or complex gloss patterns.

While the system performed well in small-scale tests, there is room for further optimization—both in terms of infrastructure and translation accuracy. With more computational resources and larger datasets, the pipeline can be extended and tested in more realistic conditions. Still, this work shows that it is possible to build a functioning LIS translation system using accessible tools, and it provides a strong foundation for future development aimed at real-world use.

While the system performed well in small-scale tests, there is room for further optimization—both in terms of infrastructure and translation accuracy. With more computational resources and larger datasets, the pipeline can be extended and tested in more realistic conditions. Still, this work shows that it is possible to build a functioning LIS translation system using accessible tools, and it provides a strong foundation for future development aimed at real-world use. By working closely with the Deaf community, we can pave the way for the development of real-world applications that bridge communication gaps and facilitate more inclusive communication for Deaf individuals.

Bibliography

[1] Necati Cihan Camgöz, Oscar Koller, Simon Hadfield, and Richard Bowden. «Neural sign language translation». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7784–7793 (cit. on pp. 6, 7).

[2] Oscar Koller. «Quantitative survey of the state of the art in sign language recognition». In: *arXiv preprint arXiv:2008.09918* (2020) (cit. on p. 6).

[3] Nadereh Hatami, Paolo Prinetto, and Gabriele Tiotto. «An Editor for Assisted Translation of Italian Sign Language». In: *Proceedings of the International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*. IEEE, 2011, pp. 589–594. DOI: 10.1109/CISIS.2011.106 (cit. on pp. 6–11).

[4] Necati Cihan Camgöz, Ben Saunders, Richard Bowden, and Simon Hadfield. «Sign language transformers: Joint end-to-end sign language recognition and translation». In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 10023–10033 (cit. on pp. 6, 13).

[5] Pooya Fayyazsanavi, Antonios Anastasopoulos, and Jana Košecká. «Gloss2Text: Sign Language Gloss Translation Using LLMs and Semantically Aware Label Smoothing». In: *Findings of the Association for Computational Linguistics: EMNLP 2024*. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 16162–16171. DOI: 10.18653/v1/2024.findings-emnlp.947. URL: https://aclanthology.org/2024.findings-emnlp.947/ (cit. on pp. 6, 8).

[6] Alptekin Orbay and Lale Akarun. «Neural Sign Language Translation by Learning Tokenization». In: *Proceedings of the 15th IEEE International Conference on Automatic Face and Gesture Recognition (FG)*. IEEE, 2020. DOI: 10.1109/FG47880.2020.00002. URL: https://doi.org/10.1109/FG47880.2020.00002 (cit. on pp. 7, 13).

[7] Alessandro Mazzei, Leonardo Lesmo, Cristina Battaglino, Mara Vendrame, and Monica Bucciarelli. «Deep Natural Language Processing for Italian Sign Language Translation». In: *Proceedings of the XIII Conference of the Italian*

*Association for Artificial Intelligence (AI*IA 2013)*. Vol. 8249. Lecture Notes in Computer Science. Springer, 2013, pp. 193–204. DOI: 10.1007/978-3-319-03524-6_17. URL: https://link.springer.com/chapter/10.1007/978-3-319-03524-6_17 (cit. on pp. 7, 8, 10).

[8] Amit Moryossef, Mathias Müller, Anne Göhring, Zifan Jiang, Yoav Goldberg, and Sarah Ebling. «An Open-Source Gloss-Based Baseline for Spoken to Signed Language Translation». In: *Proceedings of the Second International Workshop on Automatic Translation for Signed and Spoken Languages*. Tampere, Finland: European Association for Machine Translation, June 2023, pp. 22–33. URL: https://aclanthology.org/2023.at4ssl-1.3/ (cit. on pp. 7, 8).

[9] Amit Moryossef, Kayo Yin, Graham Neubig, and Yoav Goldberg. «Data Augmentation for Sign Language Gloss Translation». In: *Proceedings of the 1st International Workshop on Automatic Translation for Signed and Spoken Languages (AT4SSL)*. 2021, pp. 1–11. URL: https://aclanthology.org/2021.mtsummit-at4ssl.1/ (cit. on p. 8).

[10] Carlo Geraci, Alessandro Mazzei, and Marco Angster. «Some Issues on Italian to LIS Automatic Translation: The Case of Train Announcements». In: *Proceedings of the First Italian Conference on Computational Linguistics (CLiC-it 2014)*. Torino, Italy, 2014, pp. 138–142. URL: https://www.aclweb.org/anthology/W14-4906/ (cit. on pp. 8, 9).

[11] Amit Moryossef and Zifan Jiang. «SignBank+: Preparing a Multilingual Sign Language Dataset for Machine Translation Using Large Language Models». In: *arXiv preprint arXiv:2309.11566* (2023). URL: https://arxiv.org/abs/2309.11566 (cit. on pp. 9, 11).

[12] Marco Braga, Pranav Kasela, Alessandro Raganato, and Gabriella Pasi. «Synthetic Data Generation with Large Language Models for Personalized Community Question Answering». In: *arXiv preprint arXiv:2410.22182* (2024). URL: https://arxiv.org/abs/2410.22182 (cit. on pp. 9, 12).

[13] Tony Busker, Sunil Choenni, and Mortaza S. Bargh. «Exploiting GPT for Synthetic Data Generation: An Empirical Study». In: *Government Information Quarterly* 42 (2025), p. 101988. DOI: 10.1016/j.giq.2025.101988. URL: https://www.researchgate.net/publication/387664252_Exploiting_GPT_for_synthetic_data_generation_An_empirical_study (cit. on pp. 9, 12, 34).

[14] Pooya Fayyazsanavi, Antonios Anastasopoulos, and Jana Košecká. *Gloss2Text: Sign Language Gloss translation using LLMs and Semantically Aware Label Smoothing*. 2024. arXiv: 2407.01394 [cs.CV]. URL: https://arxiv.org/abs/2407.01394 (cit. on pp. 11, 13, 35).

[15] Manuela Marchisio, Alessandro Mazzei, and Dario Sammaruga. «Introducing Deep Learning with Data Augmentation and Corpus Construction for LIS». In: *Proceedings of the Italian Conference on Computational Linguistics (CLiC-it)*. 2023. URL: https://api.semanticscholar.org/CorpusID:266726316 (visited on 03/24/2025) (cit. on pp. 17, 22).

[16] Django Software Foundation. *Django Web Framework*. Version 4.x, Accessed: 2025-03-25. Django Software Foundation. 2024. URL: https://www.djangoproject.com/ (cit. on p. 25).

[17] Django Software Foundation. *Django Channels Documentation*. Accessed: 2025-03-25. 2024. URL: https://channels.readthedocs.io/ (cit. on p. 25).

[18] IETF. *The WebSocket Protocol*. https://datatracker.ietf.org/doc/html/rfc6455. RFC 6455, Accessed: 2025-03-25. 2011 (cit. on p. 26).

[19] Celery Project. *Celery: Distributed Task Queue*. Accessed: 2025-03-25. 2024. URL: https://docs.celeryq.dev/ (cit. on p. 27).

[20] Celery Project. *Celery Beat: Periodic Task Scheduler*. Accessed: 2025-03-25. 2024. URL: https://docs.celeryq.dev/en/stable/userguide/periodic-tasks.html (cit. on p. 27).

[21] Python Packaging Authority. *Poetry: Python Dependency Management and Packaging Made Easy*. Accessed: 2025-03-25. 2024. URL: https://python-poetry.org/ (cit. on p. 28).

[22] Docker Inc. *Docker: Enterprise Container Platform*. Accessed: 2025-03-25. 2024. URL: https://www.docker.com/ (cit. on p. 28).

[23] scikit-learn developers. *Principal Component Analysis (PCA) — scikit-learn documentation*. Accessed: 2025-03-25. 2024. URL: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html (cit. on p. 29).

[24] OpenCV Team. *OpenCV: Open Source Computer Vision Library*. Accessed: 2025-03-25. 2024. URL: https://opencv.org/ (cit. on p. 29).

[25] Camillo Lugaresi, Jiuqiang Tang, Hartwig Adam Nash, Chuo-Ling McCormick, et al. «MediaPipe: A Framework for Building Perception Pipelines». In: *Proceedings of the 25th ACM SIGMM International Conference on Multimedia*. ACM, 2019, pp. 2276–2279. DOI: 10.1145/3343031.3350539 (cit. on p. 29).

[26] Redis Ltd. *Redis: In-Memory Data Structure Store*. Accessed: 2025-03-25. 2024. URL: https://redis.io/ (cit. on p. 30).

[27] PostgreSQL Global Development Group. *PostgreSQL: The World's Most Advanced Open Source Relational Database*. Accessed: 2025-03-25. 2024. URL: https://www.postgresql.org/ (cit. on p. 30).

[28] OpenAI. *OpenAI API*. Accessed: 2025-03-25. 2024. URL: https://platform.openai.com/docs (cit. on p. 37).

[29] Thomas Wolf et al. «Transformers: State-of-the-Art Natural Language Processing». In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (2020), pp. 38–45. DOI: 10.18653/v1/2020.emnlp-demos.6. URL: https://aclanthology.org/2020.emnlp-demos.6 (cit. on pp. 38, 42).

[30] Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Numa Thain, Suraj Patil, Julien Chaumond, Patrick von Platen, Yacine Jernite, Teven Le Scao, et al. «Datasets: A community library for natural language processing». In: *arXiv preprint arXiv:2109.02846* (2021). URL: https://arxiv.org/abs/2109.02846 (cit. on p. 38).

[31] Jeff Reback, Wes McKinney, et al. «pandas-dev/pandas: Pandas». In: *Zenodo* (2020). DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134 (cit. on p. 38).

[32] Charles R Harris et al. «Array programming with NumPy». In: *Nature* 585.7825 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2 (cit. on p. 38).

[33] Python Software Foundation. *json — JSON encoder and decoder*. Accessed: 2025-03-25. 2024. URL: https://docs.python.org/3/library/json.html (cit. on p. 38).

[34] Marcin Junczys-Dowmunt et al. «Marian: Fast Neural Machine Translation in C++». In: *Proceedings of ACL 2018, System Demonstrations*. Association for Computational Linguistics, 2018, pp. 116–121. URL: https://aclanthology.org/P18-4020 (cit. on p. 42).

[35] Yinhan Liu, Jiatao Li, Yuxian Meng, Hao Zhou, Veselin Stoyanov, and Furu Wei. «Multilingual Denoising Pre-training for Neural Machine Translation». In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*. 2020, pp. 875–885. DOI: 10.18653/v1/2020.acl-main.64. URL: https://aclanthology.org/2020.acl-main.64 (cit. on p. 42).

[36] Adam Paszke et al. «PyTorch: An Imperative Style, High-Performance Deep Learning Library». In: *Advances in Neural Information Processing Systems* 32 (2019). URL: https://papers.nips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf (cit. on p. 42).

[37] Hugging Face. *Evaluate: A library for evaluating NLP models*. Accessed: 2025-03-25. 2022. URL: https://huggingface.co/docs/evaluate (cit. on p. 48).

[38] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. «BERTScore: Evaluating Text Generation with BERT». In: *International Conference on Learning Representations (ICLR)*. 2020. URL: `https://openreview.net/forum?id=SkeHuCVFDr` (cit. on p. 48).