cx People's Democratic Republic of Algeria
Ministery of Higher Education and Scientific Research
Mohamed El Bachir El Ibrahimi University of Borj Bou Arréridj
Faculty of Mathematics and Computer Science
Department of Computer Science

DISSERTATION
Presented in fulfillment of the requirements of obtaining the degree
Master in Computer Science
Specialty: Information And Communication Technologie

THEME

Intelligent voice assistant for real-time translation between French and English

Presented by:

LAHOUIBI SALAH EDDINE

BELLEM RIYAD

Publicly defended on: 10/06/2025

In front of the jury composed of:

President: ATTIA SAFA

Examiner: BENMESSAHEL ILYES

Supervisor: SAIFI LYNDA

2024/2025

Dedication

At the end of this journey, which was full of challenges and experiences, I find myself deeply grateful to everyone who was part of it, and to all those who lent me a helping hand—whether through words, actions, or simply by being present.

First and foremost, I extend my highest gratitude and appreciation to the two people who were the foundation of everything I have achieved: my dear father Jamal and my beloved mother Jamila. You are the ones who gave me life and strength. Your unconditional love, unwavering support, and priceless sacrifices were always the light that illuminated my path and the compass that guided me when I was lost.

I also wish to express my heartfelt thanks to my brother Samir and my two dear sisters, who have always stood by my side, providing me with safety, patience, and support through all circumstances.

To my dear friends: Imad, Ishaq, Amir, and Ayman—you were more than just friends. You were the family I chose and the companions who helped ease the burdens of this journey. Thank you for the sincere and beautiful moments, and for standing by me at every step.

To my dear friend Islam Bouchou, thank you for your wonderful companionship, your sincere support, and your unforgettable gestures. Your presence during this phase was truly a blessing, and I am grateful for every moment you shared the path with me.

I must also extend special thanks to my friend and project partner Salah Eddine Lhouibi, with whom I shared every stage of this work—with all its efforts and pride. He was a true example of responsibility and dedication

I also raise my sincerest words of appreciation to my respected supervisor, Professor Saifi Linda, who guided me with her wisdom and great patience, and who opened up scientific horizons that helped shape and refine this work in the best way possible.

RYAD

As this journey comes to an end, filled with challenges and meaningful experiences, I am truly thankful to all who were part of it—whether through their words, support, actions, or simply by being there.

First and foremost, I would like to thank my dear parents for all their support throughout my academic journey. I hope that my success will be a reason for their happiness, as they have a great role in this success.

Of course, we must not forget my brothers who created the atmosphere that helped and encouraged us to continue this project.

And for my friend, companion, and partner, Ryad Balam, without whom this project would not have been completed. I hope to continue working hard and with perfection.

And a special thanks to my friends Imad and Isaac, who have been with us since the beginning of this journey and were there to help at every step.

I would like to thank two very dear friends, Ayman and Salah El-Din, with whom I lived through all the university experiences, both good and bad. I would like to thank them for their brotherhood and their positions, and I wish them success in their lives.

All thanks and appreciation to the honorable Dr. Saifi Lynda, who was with us every step of the way, keen and supporting us in all the problems and obstacles, with my wishes for her success in her educational journey.

SALAH EDDINE

Acknowledgment

Praise be to Allah, by whose grace good deeds are completed, and by His guidance works are accomplished.

We extend our sincerest thanks and gratitude to Allah Almighty for granting us the strength, patience, and determination to complete this modest work.

We would like to express our deepest gratitude to our esteemed supervisor, Ms. Saifi Linda, who closely followed and guided us throughout the academic year. Her continuous support and encouragement were fundamental pillars in the success of our project.

We also extend our sincere appreciation to the members of the examination committee, Mr. Atiya Safa and Mr. Elias Ben Meshel, for kindly reading and evaluating this work, and we express our highest gratitude for their valuable time and efforts.

We wish to express our heartfelt thanks to our families, who stood by us throughout this journey. We deeply appreciate their patience and constant support, as their encouragement and love played a significant role in accomplishing this achievement. Additionally, we thank all our professors for their dedication and effort in providing us with knowledge and support throughout our studies.

Finally, we express our heartfelt thanks to our friends, colleagues, and everyone who supported us throughout this academic journey; your support has been invaluable.

Abstract

In light of the rapid developments in the field of artificial intelligence, real-time voice translation technologies have become a pivotal tool for bridging linguistic and cultural gaps between individuals. This project aims to develop an intelligent voice assistant capable of real-time speech translation between French and English, while maintaining accuracy, context, and meaning. The system is based on a Transformer architecture and combines techniques such as Automatic Speech Recognition (ASR), Neural Machine Translation (NMT), and Text-to-Speech (TTS). Translation models have been trained on a custom dataset using fine-tuning techniques to enhance performance in real-world use cases. The work also included a thorough evaluation of model performance using metrics such as accuracy and BLEU, which demonstrated promising results surpassing some common systems. This project highlights the advanced potential of integrating artificial intelligence in building effective and multilingual communication tools.

Keywords: Large Language Models (LLMs), Transformers, Legal LLMs, Legal AI, Pre-Training, Fine-Tuning.

Résumé

Avec l'évolution rapide de l'intelligence artificielle, les technologies de traduction vocale en temps réel sont devenues des outils essentiels pour surmonter les barrières linguistiques et cul- turelles. Ce projet vise à développer un assistant vocal intelligent capable de traduire la parole entre le français et l'anglais avec précision, tout en respectant le contexte et le sens sémantique.

Le système repose sur une architecture de type Transformer et intègre des technologies telles que la reconnaissance automatique de la parole (ASR), la traduction automatique neuronale (NMT) et la synthèse vocale (TTS). Des modèles de traduction ont été ajustés (fine-tuning) sur un jeu de données bilingue spécifique afin d'améliorer leur performance dans des situations réelles. Le projet comprend également une évaluation rigoureuse des performances en utilisant des métriques telles que la précision et le score BLEU, montrant des résultats prometteurs, souvent supérieurs à ceux de certains systèmes populaires. Ce travail met en lumière le potentiel des solutions basées sur l'IA pour permettre une communication vocale multilingue fluide et efficace.

Mots clés: Grands modèles de Language (LLM), transformateurs, LLM juridiques,IA juridique, pré-entrainement, Fine-Tuning,.

ملخص

في ظل التطورات السريعة التي يشهدها مجال الذكاء الاصطناعي، أصبحت تقنيات الترجمة الصوتية الفورية أداة محورية لتقريب المسافات اللغوية والثقافية بين الأفراد. يهدف هذا المشروع إلى تطوير مساعد صوتي ذكي قادر على الترجمة الفورية للكلام بين اللغتين الفرنسية والإنجليزية، مع الحفاظ على الدقة والسياق والمعنى. يعتمد النظام على بنية ةرَنسفرمر، ويجمع بين تقنيات التعرف على الكلام (اضض)، والترجمة العصبية (شة)، وتحويل النص إلى كلام (ةةض). تم تدريب نماذج الترجمة على مجموعة بيانات مخصصة باستخدام تقنيات الضبط الدقيق لتحسين الأداء في حالات الاستخدام الواقعية. كما شمل العمل تقييماً دقيقاً لأداء النماذج باستخدام مؤشرات مثل الدقة وش‌زو، مما أظهر نتائج واعدة تفوق بعض الأنظمة الشائعة. يُبرز هذا المشروع الإمكانات المتقدمة لتكامل الذكاء الاصطناعي في بناء أدوات تواصل فعّالة ومتعددة اللغات.

كلمات مفتاحية:النماذج اللغوية الكبيرة (ش‌ش‌ش(، المحولات (ةرَنسفرمرس(، النماذج اللغوية القانونية، الذكاء الاصطناعي القانوني، التدريب المسبق، التدريب الدقيق ،(ةُنِنـسِن).

Table of Contents

List of Figures

List of Acronyms

LLM	Large Language Model
AI	Artificial Intelligence
NLP	Natural Language Processing
NLU	Natural Language Understanding
NLG	Natural Language Generation
SLM	Statistical Language Model
NLM	Neural Language Model
PLM	Pre-trained Language Model
GPT	Generative Pre-training Transformer
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory Network
LLaMA	Large Language Model Meta AI
PaLM	Pathways Language Model
BLEU	Bilingual Evaluation Understudy

General Introduction

1. Context:

In today's globalized world, communicating over diverse dialects has ended up a genuine challenge. Whether in trade, instruction, or way of, life individuals frequently have to be get it and connected with others who talk diverse dialects. Much obliged to propels in innovation, particularly in machine interpretation, it's presently conceivable to bridge these dialect crevices more effectively than ever some time recently. Transformer models have appeared awesome guarantee in making interpretations speedier and more exact.

2. Problem Statement

Indeed with all these advancements, machine interpretation still faces a few imperative obstacles. Interpreting talked dialect in real-time is particularly dubious since it requires understanding setting, dealing with diverse complements or lingos, and managing with foundation clamor. These components regularly make it difficult to convey smooth and characteristic interpretations that truly capture the meaning of what's being said. So, the address is:

How can we construct a voice interpretation framework that not as it were interprets precisely but moreover sounds characteristic and keeps the meaning intaglio, indeed when managing with real-world discourse challenges?

3. Objective:

This project centers on making a bilingual voice interpretation framework that works between French and English utilizing Transformer-based models. The objective is to combine state-of-the-art discourse acknowledgment, interpretation, and text-to-speech advances into one consistent application. We'll too see at how to fine-tune the show and assess its execution to ensure it meets tall measures.

4. Report Structure:

This report consists of several chapters that detail each phase of the project:

Chapter 1: Introduction to Transformer models and natural language processing concepts.

Chapter 2: Overview of speech recognition and speech synthesis technologies used in the system.

Chapter 3: Implementation,Description of the system design, architecture, and technical choices.

Chapter 4: optimization, and evaluation of the translation model, along with experimental results.

Chapter 1

Models and architectures of natural languge processing

1.1 Introduction

Natural language processing (NLP) is based on advanced signs, and a great part of the integration of linguistic models that exist. The largest language models (LLMs) are a transformer driver, adjusting our keyboard interface, and interacting with the language. This chapter examines the NLP principles and linguistic models, and details the LLM, the importance of multiple applications. By analyzing the basic mechanics, noting the architecture transformer, and in a category of different LLM classes, this will help to capture a vision approximating the influence of these models on the development of communication and technologies.

1.2 Models and architectures of natural languge processing

1.2.1 Natural language processing (NLP)

NLP is a field that combines artificial intelligence (AI), linguistics, and computer science and explores how computers can be used to understand, interpret, and generate natural language text or speech in a way that is both meaningful and useful . NLP can be divided into two main subfields: [1]

1.2.2 Natural Language Understanding (NLU)

A subset of natural language processing that ascertains a sentence's meaning through the syntactic and semantic analysis of speech and text. Semantics suggests the intended meaning of a sentence, whereas syntax describes the grammatical structure of a sentence.

1.2.3 Natural Language Generation (NLG)

Another part of natural language processing is natural language generation. While natural language understanding conce ntrates on computer reading comprehension, NLG can generate a text response in human language and enable computers to write new text based on specific input data.

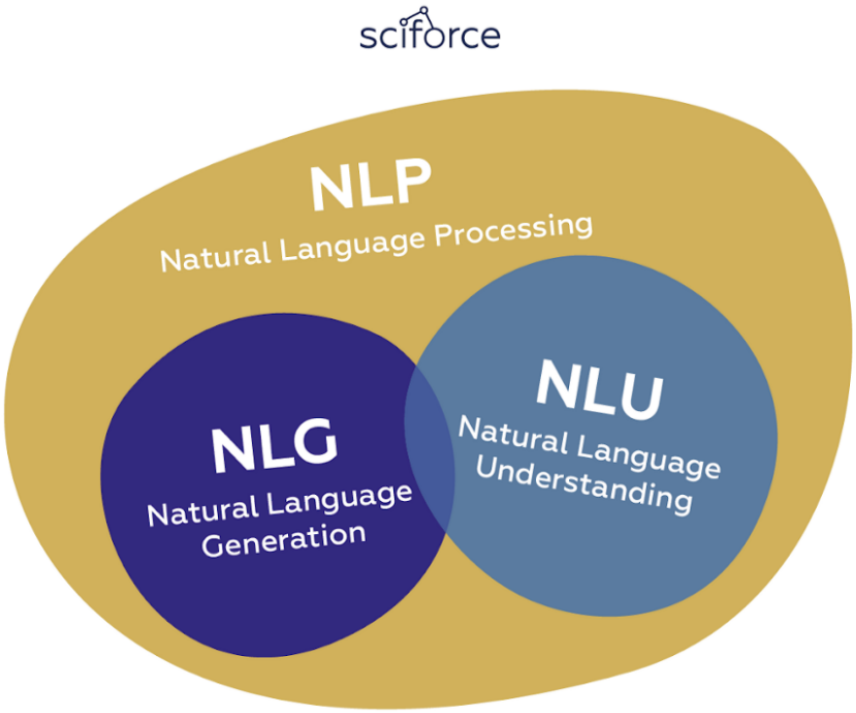


Figure 1.1: NLP

[2]

1.2.4 Language models

The study of language modeling has a long history; it began in the 1950s when Shannon applied information theory to human language and measured the predictive and compressive

powers of simple N-gram language models [3]. Statistical language modeling has been used ever since and has become essential for many tasks involving generating and comprehending natural language. Language Models can be categorized into four categories: statistical language models, neural language models, pre-trained language models, and large language models (Minaee et al., 2024)[4].

1.2.4.1 Statistical language models (SLMs)

This type of model proceeds text as a sequence of words to estimate the probability of text by using the product of the word probabilities. The most common types of SLMs are Markov chain models or N-gram models. These models calculate a word's probability based on the n-1 words that come right before it (S. F. Chen Goodman, 1999).

1.2.4.2 Early neural language models (NLMs)

NLMs (Bengio et al., 2000; Schwenk et al., 2006[5]) mapped words to low-dimensional continuous vectors (embedding vectors) and used neural networks to predict the next word based on the sum of the embedding vectors of the words that came before it. The embedding vectors define a hidden space in which it is easy to calculate the distance between vectors based on their semantic similarity. They are task-specific models because NLMs are trained on task-specific data and have a task-specific learned hidden space.

1.2.4.3 Pre-trained language models (PLMs)

Unlike early NLMs, pre-trained language models (PLMs) are task-agnostic. Additionally, they learned a hidden embedding space. PLMs are trained and inferred according to the pre-training and fine-tuning paradigm: language models with transformers (Devlin,2018; Liu et al., 2019)[6] [7]or recurrent neural networks (Sarzynska-Wawer et al., 2021)[8] are pre-trained on web-scale unlabeled text corpora for generic tasks like word prediction, and are then fine-tuned to specific tasks using tiny amounts of (labeled) task-specific data.

1.2.4.4 Large language models (LLMs)

Mostly refer to transformer-based neural language models, which have tens to hundreds of billions of parameters and are pre-trained on enormous amounts of text data like PaLM (Chowdhery et al., 2023)[9], LLaMA (Touvron et al., 2023)[10], and GPT-4 (Achiam et al.,

2023). These models can comprehend and produce natural language text and other types of content because they have been trained on a vast amount of data, allowing them to carry out various tasks.

1.3 Large Language Models use cases and tasks

Large Language Models (LLMs) have gained widespread recognition due to their ability to efficiently handle various Natural Language Processing (NLP) tasks, including: Text Generation: LLMs can generate coherent and contextually relevant text based on the topics they have been trained on. Conversational AI Chatbots: These models facilitate smoother and more natural interactions with users compared to earlier AI technologies. Translation: Multilingual LLMs can accurately translate text between different languages. Summarization: LLMs can condense lengthy passages or entire documents into concise summaries without losing essential meaning. Classification Categorization: These models can organize and classify content based on predefined categories. Sentiment Analysis: LLMs can assess the sentiment of text, helping users interpret the underlying emotions or tone. Information Retrieval: They can search within documents, across multiple documents, or retrieve relevant metadata. Named Entity Recognition (NER): LLMs can identify and classify specific entities such as names, dates, and locations within text. Question Answering: These models provide accurate and relevant responses to user queries. The following figure illustrates the main areas of use for LLMs..

Figure 1.2: represents the top 10 Uses for LLM

[11]

1.4 How LLMs work - Transformer Architecture

1.4.1 Generating Text with RNN and LSTM

The ability to generate coherent and contextually relevant text has long been a benchmark for progress in natural language processing (NLP). Before the rise of Transformer-based models, Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks were among the most influential architectures for tackling text generation challenges. These models laid the groundwork for many subsequent advancements in NLP. However, traditional RNNs face significant challenges, particularly due to vanishing and exploding gradients, which hinder their ability to capture long-range dependencies within sequences. As a result, their performance declines in tasks requiring deep contextual understanding. LSTMs were designed to mitigate some of these issues by introducing mechanisms for better handling long-term depen-

dencies. Despite these improvements, they still struggle with very long sequences, as the likelihood of retaining relevant information decreases exponentially with increasing word distance. Moreover, the sequential nature of RNNs and LSTMs makes them less efficient in modeling complex interactions in text. This also limits their ability to take advantage of parallel computing, making them less efficient compared to more modern architectures like Transformers.

1.4.2 Transformer Architecture

The landscape of NLP underwent a major transformation in 2017 with the introduction of the Transformer architecture, as presented in the groundbreaking paper "Attention is All You Need" by Vaswani et al. This innovation, developed by Google and the University of Toronto, laid the foundation for the impressive advancements in generative AI that we see today. The Transformer model revolutionized NLP by leveraging multi-core GPUs, enabling parallel processing of input data, and utilizing larger training datasets. More importantly, it introduced an efficient mechanism for learning and interpreting the contextual meaning of words. At the core of the Transformer architecture are two primary components: The Encoder The Decoder These components share a symmetrical structure and several common features, with slight modifications tailored to their respective roles in tasks such as text generation and machine translation. This innovative approach has significantly enhanced the efficiency and scalability of NLP models, shaping the future of AI-driven language understanding.[12]

Encoder

The encoder in the Transformer model consists of N = 6 identical layers, each composed of two key sub-layers. The first sub-layer implements a multi-head self-attention mechanism, while the second sub-layer is a fully connected feed-forward network. To enhance learning stability, residual connections are applied around both sub-layers, followed by layer normalization (Vaswani et al., 2017). This ensures that the output of each sub-layer follows the formula: LayerNorm(x+Sublayer(x)) Where Sublayer(x) represents the function executed by the respective sub-layer. To maintain consistency across the model, the embedding layers and all sub-layers generate outputs with a fixed dimensionality of dd = 512.

Decoder

The decoder in the Transformer model consists of N = 6 identical layers, similar to the encoder. However, unlike the encoder, each decoder layer contains an additional third sub-layer that performs multi-head attention over the encoder's output. Just like in the encoder, resid-

ual connections are applied around each sub-layer, followed by layer normalization to stabilize training and improve gradient flow. A key modification in the decoder's self-attention sub-layer is the use of masking, which prevents positions from attending to future positions. This ensures that predictions for position i depend only on the outputs of positions before i, maintaining the auto-regressive nature of text generation.

The original transformer architecture is represented in Figure 1.3

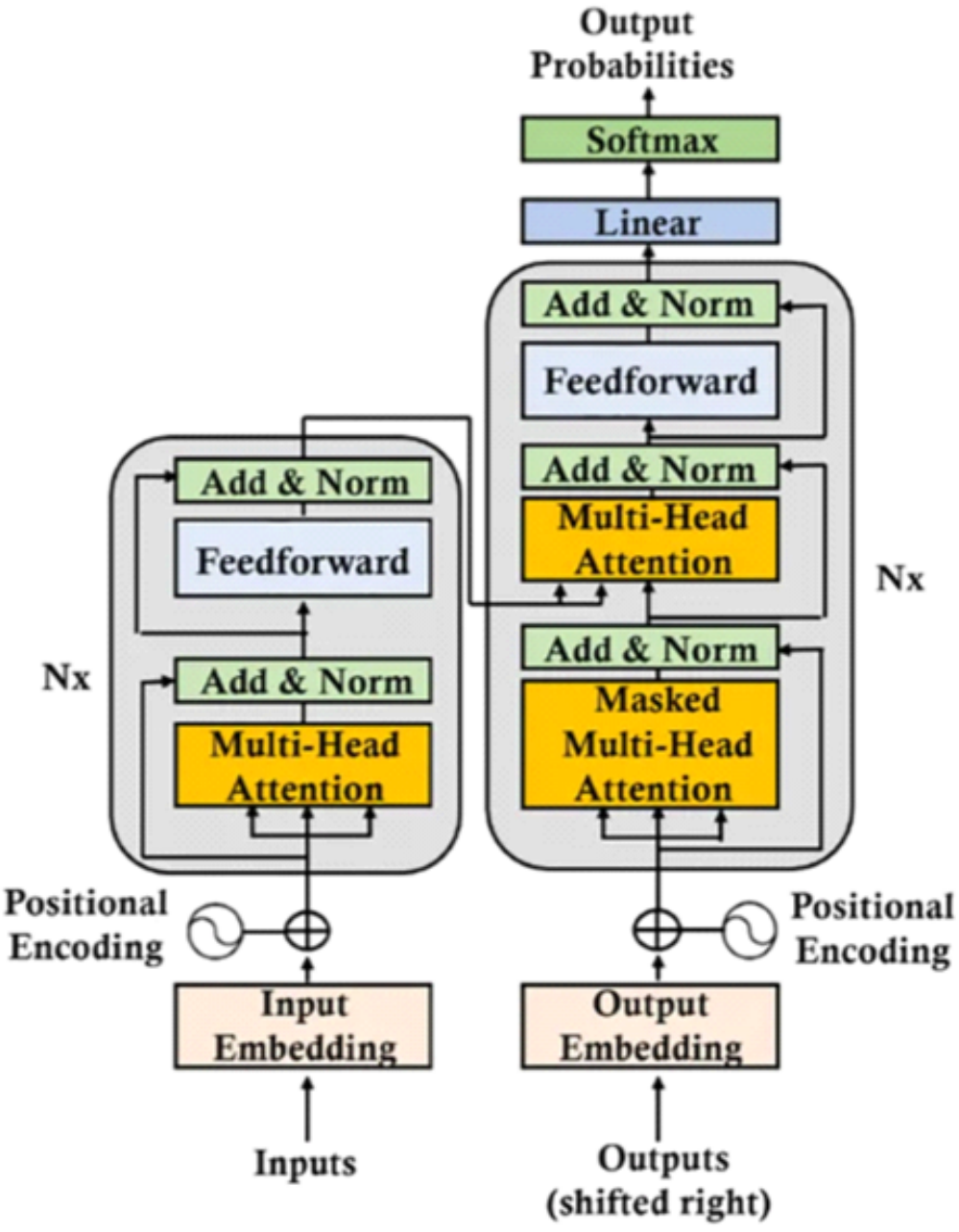


Figure 1.3: Transformer Architecture

[13]

After establishing a general understanding of the Transformer architecture, it's essential to take a closer look at the encoder and decoder components to grasp their internal mechanisms more effectively. However, before delving into the specifics of encoder and decoder layers, it's important to first understand the following key concepts.

1.4.2.1 Tokenization:

Before feeding text into the encoder or decoder stack, it must first be tokenized. Tokenization is the process of converting words into numerical representations, where each number corresponds to a specific position in a predefined vocabulary that the model can recognize and process. This step is crucial in any NLP pipeline, as it directly influences the overall performance of the model. A tokenizer segments raw, unstructured text into manageable units, treating each as a distinct element for further processing. Once the input has been transformed into numerical tokens, these values are then passed to the embedding layer, which further refines their representation before they enter the Transformer model.

1.4.2.2 Embedding layer:

In the Embedding Layer, each token is represented as a vector and has a distinct place inside the high-dimensional trainable vector embedding space, which is the Embedding Layer. Since each token ID in the vocabulary is matched to a multidimensional vector, these vectors pick up the ability to encode the context and meaning of individual tokens in the input sequence. The vector size in the original Transformer article was 512.

1.4.2.3 Positional Encoding:

The encoder layer converts input sequences into a continuous, abstract representation that encapsulates contextual information learned throughout the sequence (Vaswani et al., 2017). This layer comprises two essential sub-modules: Multi-head self-attention mechanism – Allows the model to focus on different parts of the input sequence simultaneously. Fully connected feed-forward network – Processes and refines the extracted features from the attention mechanism. Additionally, residual connections are incorporated around each sub-layer to facilitate better gradient flow, followed by layer normalization to enhance training stability.

1.4.3 Stack of Encoder Layers

The encoder layer converts input sequences into a continuous, abstract representation that encapsulates contextual information learned throughout the sequence (Vaswani et al., 2017).This layer comprises two essential sub-modules:

1.4.3.1 Multi-head self-attention mechanism

– Allows the model to focus on different parts of the input sequence simultaneously.

1.4.3.2 Fully connected feed-forward network

Processes and refines the extracted features from the attention mechanism.
Additionally residual connections are incorporated around each sub-layer to facilitate better gradient flow, followed by layer normalization to enhance training stability.

1.4.3.3 output of the Encoder

A collection of vectors, each of which represents the input sequence with a deep awareness of context, is the final encoder layer's output. After that, this output is sent to a transformer model's decoder. When it comes time to decode, this careful encoding sets the path for the decoder by guiding it to pay attention to the appropriate words in the input.

1.4.4 Stack of Decoder Layers

A stack of identical layers makes up the decoder (6 in the original Transformer concept). All layers consist of three primary sub-components:

1.4.4.1 Masked Self-Attention Mechanism:

In the decoder, the self-attention mechanism is modified with masking so that each token can only attend to positions that come before it. This ensures that future tokens do not influence the current prediction. In practice, this is implemented within the scaled dot-product attention by assigning a value of negative infinity () to any connections with tokens that follow, thereby maintaining the auto-regressive property.

1.4.4.2 Encoder-Decoder Multi-Head (Cross) Attention:

The second multi-head attention layer in the decoder creates a bridge between the encoder and decoder. Here, the decoder's masked self-attention output serves as the values, while the encoder's outputs function as both the queries and keys. This cross-attention mechanism allows every position in the decoder to consider all positions from the input sequence, and a subsequent pointwise feedforward layer further refines this integrated information.

1.4.4.3 Feed-Forward Neural Network:

Like the encoder, every decoder layer has a fully linked feed-forward network that is Applied to every location in the same way, independently.
Generating Word Predictions with Linear Layer and Softmax In transformer models, the final linear layer functions as a classifier that maps model outputs to the vocabulary space. Its size equals the total number of words in the vocabulary—each output corresponds to one word. For example, if the vocabulary contains 1000 words, the output of this layer will be a vector of size 1000. This vector is then passed through a softmax function, which transforms the raw values into probability scores between 0 and 1. The word with the highest probability score is selected as the model's prediction for the next word in the sequence.

1.4.4.4 Generating Word Predictions with Linear Layer and Softmax

In transformer models, the final linear layer functions as a classifier that maps model outputs to the vocabulary space. Its size equals the total number of words in the vocabulary—each output corresponds to one word. For example, if the vocabulary contains 1000 words, the output of this layer will be a vector of size 1000. This vector is then passed through a softmax function, which transforms the raw values into probability scores between 0 and 1. The word with the highest probability score is selected as the model's prediction for the next word in the sequence.

1.5 TRANSFORMER MODEL TYPES

The transformer models are divided into 3 main types: Encoder-Only, Encoder-Decoder, and Decoder-Only models.

Figure 1.4 represents the transformer model types with their tasks and examples.

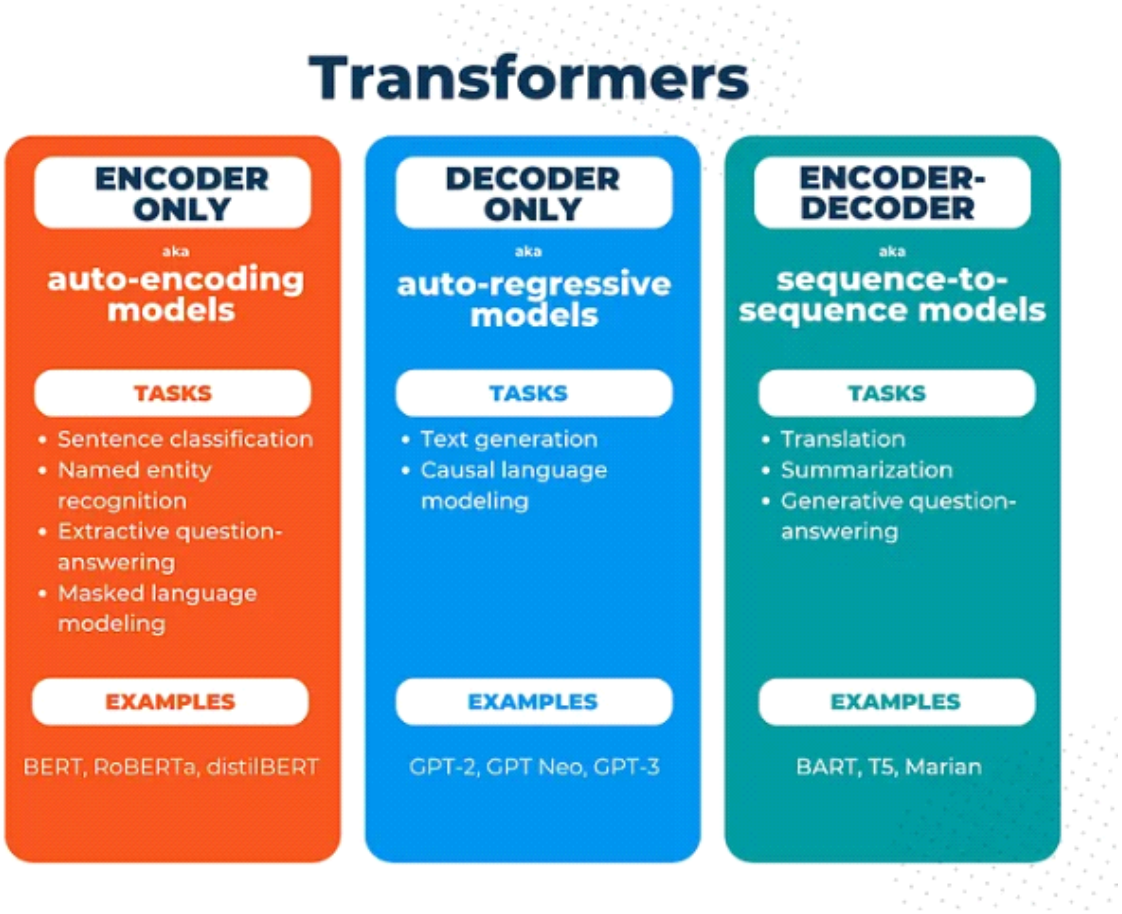


Figure 1.4: The transformer model types with their tasks and examples (Comet) [14]

Chapter 2

Voice assistant processing

2.1 Introduction

Machine voice translation represents one of the most impactful applications of artificial intelligence in bridging linguistic and cultural gaps. This chapter presents a comprehensive overview of fundamental concepts related to translation in general and machine translation in particular, with a focus on the technologies behind intelligent voice assistants. It also highlights the advantages of these systems, the challenges they face, and the methods used to enhance translation quality and evaluate performance. Additionally, the chapter includes examples of well-known applications such as Google Translate and Microsoft Translator, placing the project within a broader practical context.

2.2 Definition of some concepts:

2.2.1 Translation :

The art of translation involves attempting to convey the same idea or statement in another language in place of one that is written in that other language. Meanwhile, (Iordan, 2021), translation is one of the oldest occupations of man. Language barriers led to this difficult but necessary task that has contributed to facilitating intercultural dialogue and the exchange of spiritual values (Alansary, A. M., 2014).[**?**]

2.2.2 Machine translation:

Machine translation is a subfield of computational linguistics that investigates the use of software to translate text or speech from one natural language to another. On a basic level, MT performs simple substitution of words in one natural language for words in another, but that alone usually cannot produce a good translation of a text, because recognition of whole phrases and their closest counterparts in the target language is needed (Alansary, A. M. 2014).[15]

2.2.3 Advantages of machine voice translation:

Here are the advantages of the machine voice translation
Real-time translation: Enables instant communication without noticeable delays, reducing language barriers.

Contextual and semantic preservation: Large Language Models (LLMs) produce translations that maintain the meaning and context of spontaneous speech.

Multimodal interaction support: Integration with other modalities (e.g., text, visuals) enhances the user experience.

Improved accessibility for people with disabilities: Assists users who are visually impaired or have difficulty reading or writing.

Fine-tuning capabilities: Models can be adapted to specific domains or use cases, improving performance in specialized contexts.

2.2.4 Fine-tuning the models :

Fine-tuning is a concept of transfer learning. Transfer learning is a machine learning technique where knowledge gained during training in one type of problem is used to train in other related tasks or domains (Pan and Fellow, 2009). In deep learning, the first few layers are trained to identify features of the task. During transfer learning, you can remove the last few layers of the trained network and retrain with fresh layers for the target job. Fine-tuned learning experiments require a bit of learning, but they are still much faster than learning from scratch (Mohanty et al., 2016). Additionally, they are more accurate compared to models trained from scratch.

2.2.5 Sequence to sequence architecture

According to Sutskever, Vinyals, and Le (2014), the sequence-to-sequence (Seq2Seq) architecture is a general framework for transforming one sequence into another using neural networks. It consists of two main components: an encoder and a decoder. The encoder is typically a multilayered Long Short-Term Memory (LSTM) network that reads and encodes the input sequence into a fixed-length vector. The decoder, another LSTM network, takes this vector and generates the output sequence step-by-step. This architecture enables the model to handle variable-length input and output sequences, making it especially suitable for tasks such as machine translation. Later extensions to the architecture introduced an attention mechanism, allowing the model to focus on relevant parts of the input sequence during decoding, which significantly improves performance for longer sequences.

2.3 Machine Voice Translation Assistant

2.3.1 Speech Recognition Systems (ASR):

Systems have moved from traditional methods to comprehensive models based on techniques such as CTC and Sequence-to-Sequence with advanced mechanisms such as the attention mechanism. The latter has led to increased accuracy in converting speech to text.

2.3.2 Self-Supervised and Unsupervised Learning:

Thanks to techniques such as Wav2Vec 2.0 and HuBERT, we were able to extract accurate speech representations from large amounts of data. This led to improved speech recognition performance in various conditions.

2.3.3 Advanced Transformer Models:

Models such as Conformer, which integrate transformer layers with convolutional layers, have emerged to better capture both local and global features in speech signals. This integration helps in handling noise and the variability in dialects. [16]

2.3.4 Speech Translation:

Speech translation is the process by which spoken phrases are instantly translated and spoken aloud into a second language. Models like OpenAI's Whisper represent a paradigm shift in speech translation. They combine machine translation and speech recognition technology into a unified model, enabling accurate and instant translation between languages such as French and English.

2.3.5 Text-to-Speech (TTS):

With the latest advancements in technology, there has been a significant development in converting text to speech, using modern models based on transformers that produce natural voices similar to human voices. Most of these developments are widely used in interactive applications or voice assistants.

Overall, these developments have contributed to enhancing the accuracy and efficiency of speech recognition and translation systems. They have also paved the way for future innovations that integrate speech technologies and artificial intelligence to provide an interactive experience that is as smooth and natural as possible.

2.4 Problems with voice translation:

Voice translation faces several challenges that can significantly affect its accuracy. One common issue is literal or word-by-word translation, which often ignores the broader context of the conversation. Additionally, the system may misinterpret accents or dialects spoken by users, leading to inaccurate translations. Furthermore, the substantial grammatical differences between the source and target languages can result in incorrect sentence structures. Lastly, environmental factors such as background noise or rapid speech can degrade the overall quality of the translation.

2.5 Translation Quality Improvement and Evaluation

The model's performance was assessed using three main metrics, which were calculated and tracked at the end of each training epoch to closely monitor translation quality:

2.5.1 Loss Function:

The loss function measures the error made by the model during training. A gradual decrease in loss indicates improved learning of the mapping between source and target sentences. It serves as a fundamental metric to guide the learning process.

2.5.2 Accuracy:

Accuracy represents the percentage of correctly predicted tokens. It provides a general measure of the model's capability to produce correct outputs, though it is not sufficient on its own to assess the linguistic or contextual quality of translations.

2.5.3 BLEU Score:

The BLEU (Bilingual Evaluation Understudy) score is a widely used quantitative metric for evaluating machine translation quality. It compares the model-generated translation with a human reference translation and computes the degree of overlap. This score was automatically calculated at the end of each epoch to monitor translation quality over time.

2.6 Related work

2.6.1 Google Translate

Description and Features: Google Translate is a free, widely used translation service that supports over 100 languages. It allows for text, voice, image, and even website translations. Google Translate offers fast and accurate translations, making it a convenient tool for everyday use. Its integration with Google services further enhances its functionality, allowing for real-time translation with both text and voice recognition. Additionally, it supports offline translations for certain languages by downloading language packs

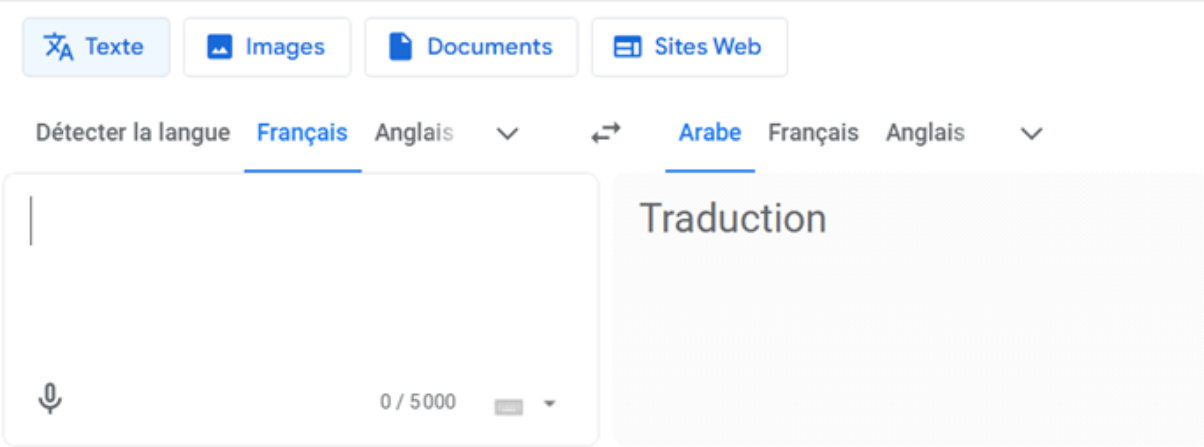


Figure 2.1: Google Translate

2.6.2 Microsoft Translator

Description and Features: Microsoft Translator is an AI-powered translation tool that provides text, voice, and image translations. It supports over 60 languages and integrates seamlessly with Microsoft products like Office and Skype, making it particularly useful for business and professional environments. Microsoft Translator also offers real-time speech translation, ideal for multilingual conversations. Users can also customize translations for specific contexts, ensuring better accuracy. The app works across various devices, allowing for translations during meetings and collaborative work.

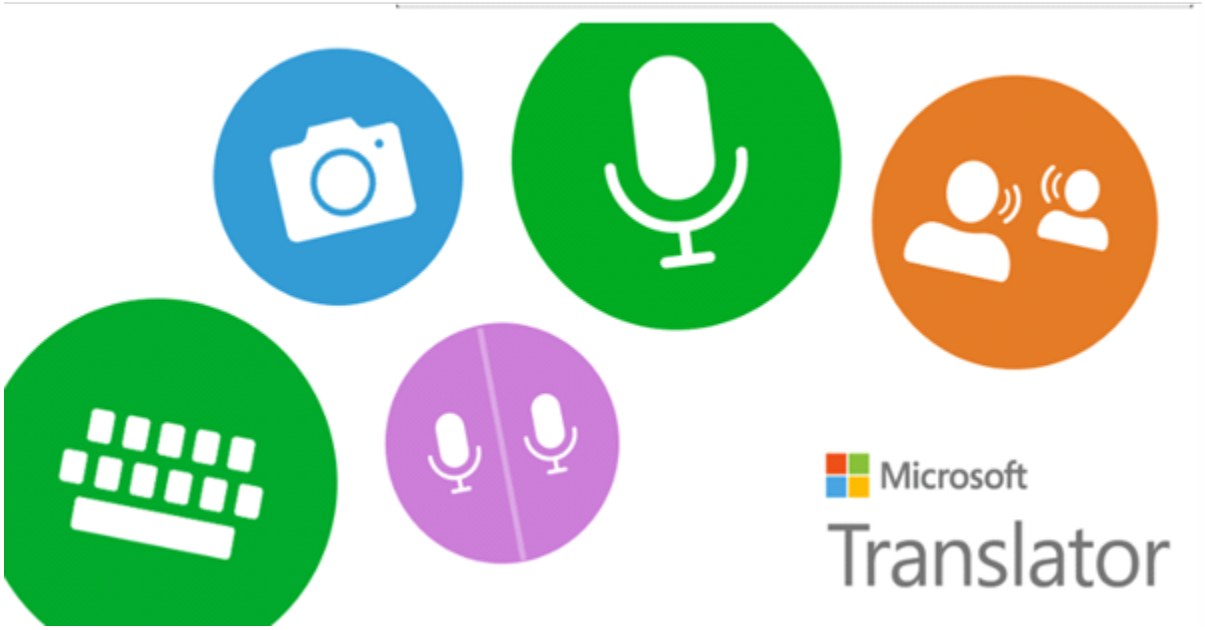


Figure 2.2: Microsoft Translator

2.6.3 SayHi Translate

Description and Features: SayHi Translate is a user-friendly mobile app that provides real-time speech and text translations. It supports over 100 languages and is designed for easy use, making it perfect for travelers and those engaging in multilingual communication. With the ability to translate both speech and text, SayHi facilitates smooth communication in multiple languages. The app also offers customization options for speech speed and accent, allowing for a personalized experience based on the user's preferences.

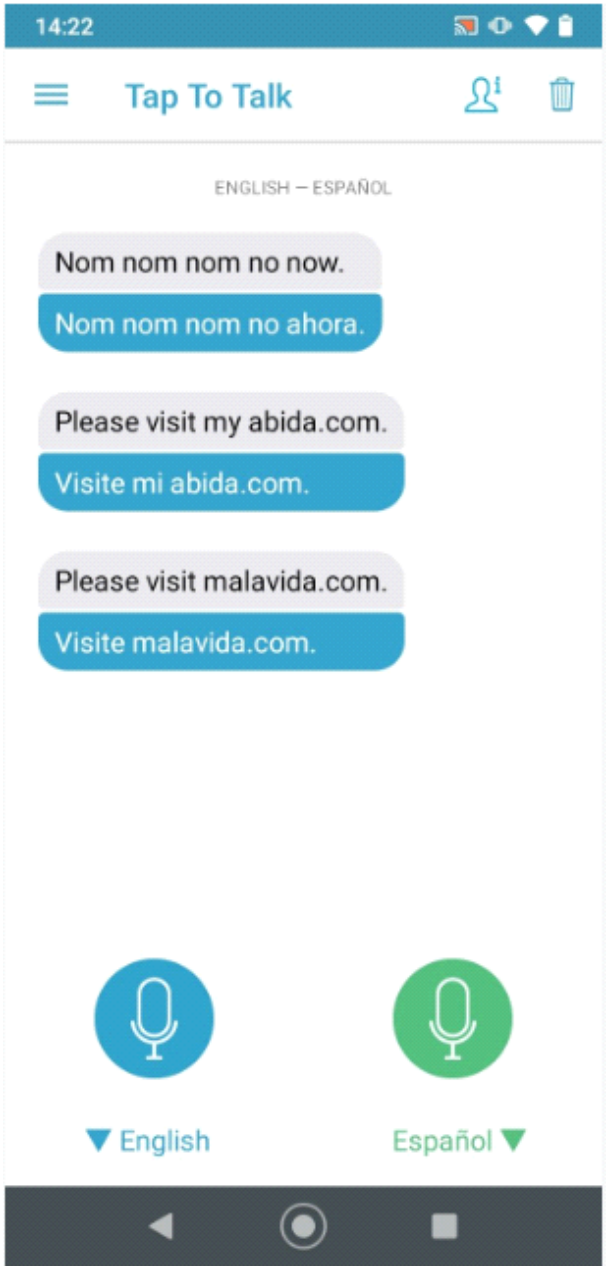


Figure 2.3: SayHi Translate

2.7 Conclusion

This overview demonstrates that machine voice translation relies on a synergy of machine learning techniques, deep language models, and advanced architectures such as Seq2Seq and Transformers. Despite challenges related to accuracy, context, and dialects, advancements in modern models like Whisper and Wav2Vec2 highlight a promising future for this domain. Moreover, evaluating translation quality using metrics such as loss, accuracy, and BLEU offers an objective framework for performance analysis and continuous improvement. This theoretical and practical foundation underpins the development of the voice translation assistant presented in this project.

Chapter 3

implementation

3.1 Introduction

This section of the project focuses on the practical aspects of developing an intelligent mobile application for speech translation. It includes training translation models using custom data and fine-tuning techniques, as well as integrating speech-to-text and text-to-speech functionalities within a Flutter environment. User interfaces were also designed to provide a smooth and interactive experience, enhancing the system's ability to deliver accurate translation and natural voice interaction.

3.2 application tools

3.2.1 Android :

Android is an open source operating system built on top of Linux. It powers a plethora of devices, from smart phones to tablets to gaming consoles (along with a vast array of other consumer electronics). According to the International Data Corporation, Android had over 70 percent market share for worldwide smartphones in the last quarter of 2012, so developing for this platform has the potential to reach a very large number of mobile users[17]

3.2.2 SDK:

A software development kit (SDK) is a set of software tools and programs provided by hardware and software vendors that developers can use to build applications for specific platforms. SDKs help developers easily integrate their apps with a vendor's services[18]

3.2.3 Android studio:

Android Studio is the official integrated development environment (IDE) for Android applications. Based on the powerful development and code editing tool IntelliJ IDEA, Android Studio offers even more features that enhance your productivity when creating Android applications.[19]

Figure 3.1: Android studio

3.2.4 Dart:

Dart is a client-optimized language for developing fast apps on any platform. Its goal is to offer the most productive programming language for multi-platform development, paired with a flexible execution runtime platform for app frameworks. Dart also forms the foundation of Flutter. Dart provides the language and runtimes that power Flutter apps, but Dart also supports many core developer tasks like formatting, analyzing, and testing code sks like formatting,analyzing, and testing code.[20]

Figure 3.2: Dart

3.2.5 Flutter:

Flutter is a cross-platform UI toolkit that is designed to allow code reuse across operating systems such as iOS and Android, while also allowing applications to interface directly with underlying platform services. The goal is to enable developers to deliver high-performance apps that feel natural on different platforms, embracing differences where they exist while sharing as much code as possible.[21**?**]

Figure 3.3: Flutter

3.2.6 Visual Studio Code:

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS, and Linux. It comes with built-in support for JavaScript, TypeScript, and Node.js and has a rich ecosystem of extensions for other languages and runtimes (such as C++, C, Java, Python, PHP, Go, NET). It is highly customizable and supports a wide range of programming languages[22]

Figure 3.4: Visual Studio Code

3.2.7 Python:

is a high-level, interpreted programming language designed for readability, simplicity, and broad applicability. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is widely used across domains such as web development, data science, artificial intelligence, and automation. Its extensive standard library and active ecosystem of third-party packages make it a versatile tool for rapid application development. Additionally, Python is cross-platform, running seamlessly on Windows, macOS, and Linux systems.[23]

Figure 3.5: python

3.3 Overview of our application

3.3.1 Home page

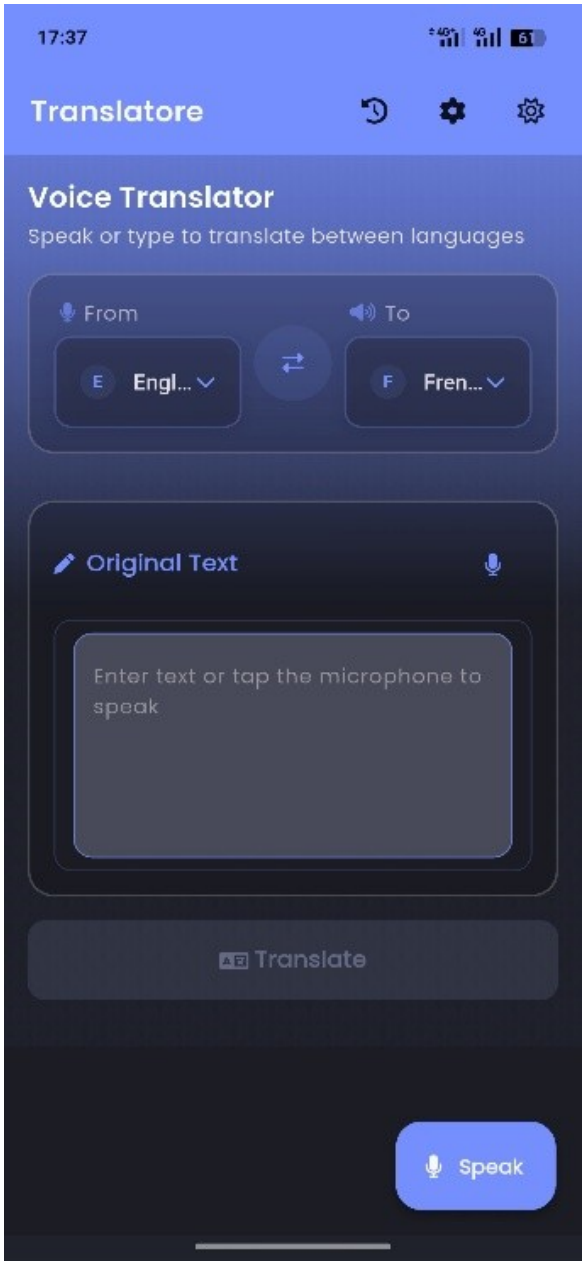


Figure 3.6: Dark Mode

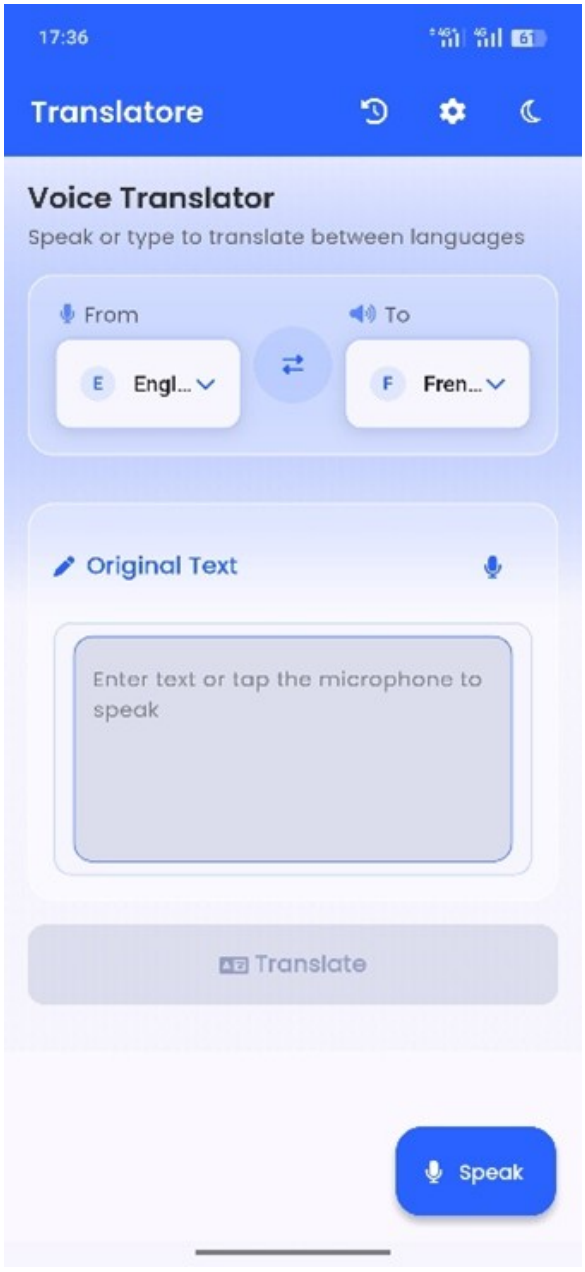


Figure 3.7: Light Mode

Main Interface Description – Voice Translator Application

The main interface of the Voice Translator application features a modern and elegant design that relies on light blue tones, providing a smooth and comfortable user experience. At the top of the screen, the application name "Translatore" appears, followed by three functional icons:

- • History Icon: Allows the user to access the record of previously made translations.
- Server Settings Icon: Provides access to manage the server connection or cloud service settings related to translation.
- Dark Mode Icon: Enables the user to switch between light and dark modes depending on the lighting environment.

Below the title bar is the main translation section, where users can select the source language (From) and the target language (To) through stylish dropdown menus. A circular button in the center allows quick switching between the two languages.
Next is the input section titled "Original Text", where users can type the text or use the microphone icon to speak directly.
There is a "Translate" button to convert the input from the original language to the selected target language, as well as a large "Speak" button at the bottom, which is used to activate the voice translation feature easily.

3.3.2 The app barre

The app bar of the Translatore app is sleek and functional, set against a solid blue background. On the left, the app name "Translatore" is displayed in bold white text, clearly indicating the app's identity. On the right, there are three icons

- The first icon (a clock with an arrow) represents the history feature, allowing users to view past translations.
- The second icon (a gear) is for settings, where users can adjust app preferences like default languages or notifications.
- The third icon (a crescent moon) toggles dark mode, switching the app to a darker theme for better visibility in low-light conditions and reduced eye strain.

The layout is clean and intuitive, ensuring easy access to key features.

Figure 3.8: app bar in light mod

Figure 3.9: app bar in dark modes

3.3.3 Dark Mode in Translatore App:

Translatore offers a dark mode feature, indicated by a crescent moon icon in the top bar. When enabled, the app shifts to darker tones, providing greater eye comfort in low-light settings or at night.

it also helps reduce battery consumption on certain devices, delivering a smoother and more personalized user experience.

3.3.4 Endpoint settings:

Here's the English translation of the description, keeping the same clear and concise style:

Translation endpoint Settings in Translatore App

The endpoint settings can be accessed via the gear icon in the app's top bar. Tapping it opens a page labeled "Translation Server" with a field to enter a Server URL.

This field lets the user input the URL of any server hosting a trained translation model. Once entered, the app uses the model on that server to translate into any language the model was trained for.

Below the field are two buttons:Below the field are two buttons:

- "Test Connection" to verify the server link works.
- "Save" to store the URL and activate the model for translation.

The design is straightforward, making it easy to customize the app with different translation models.

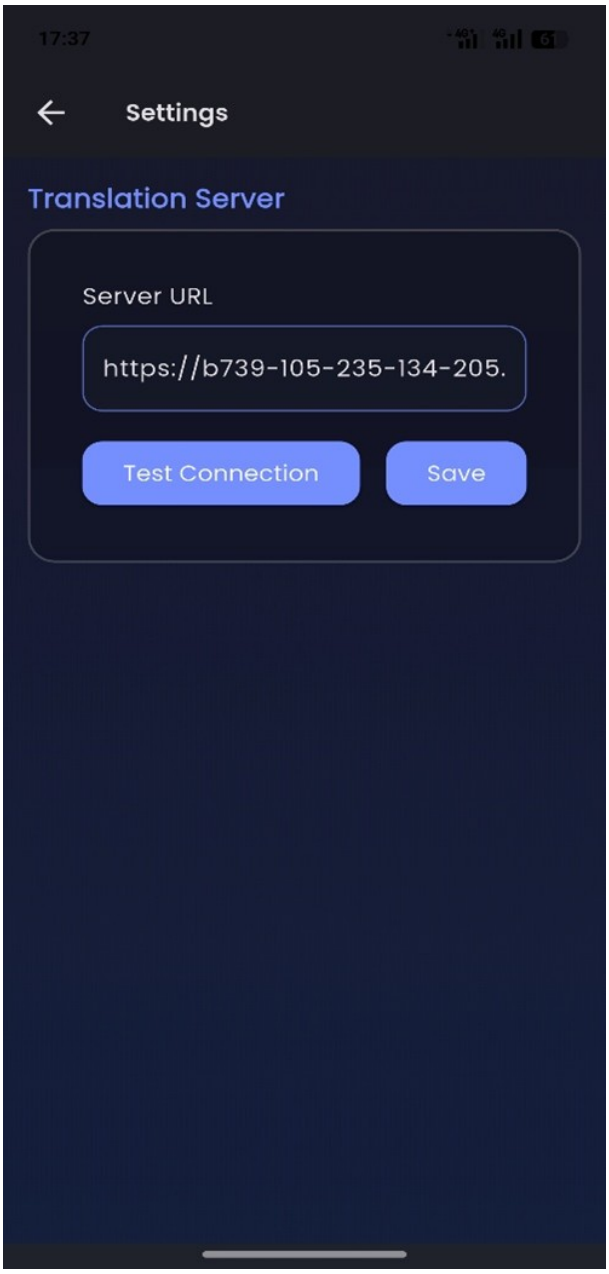


Figure 3.10: End-point

3.3.5 History Icon

This icon opens the Translation History screen, where all your previous translations are neatly listed. Each entry includes:

- The source and target languages (e.g., English → French)
- The original text and its translation
- The date and time the translation was made

- Interactive options like Use (to re-insert the translation) and Speak (to hear it aloud)

The interface is styled with a dark theme for comfortable viewing, making it easy to review, reuse, or listen to past translations efficiently.

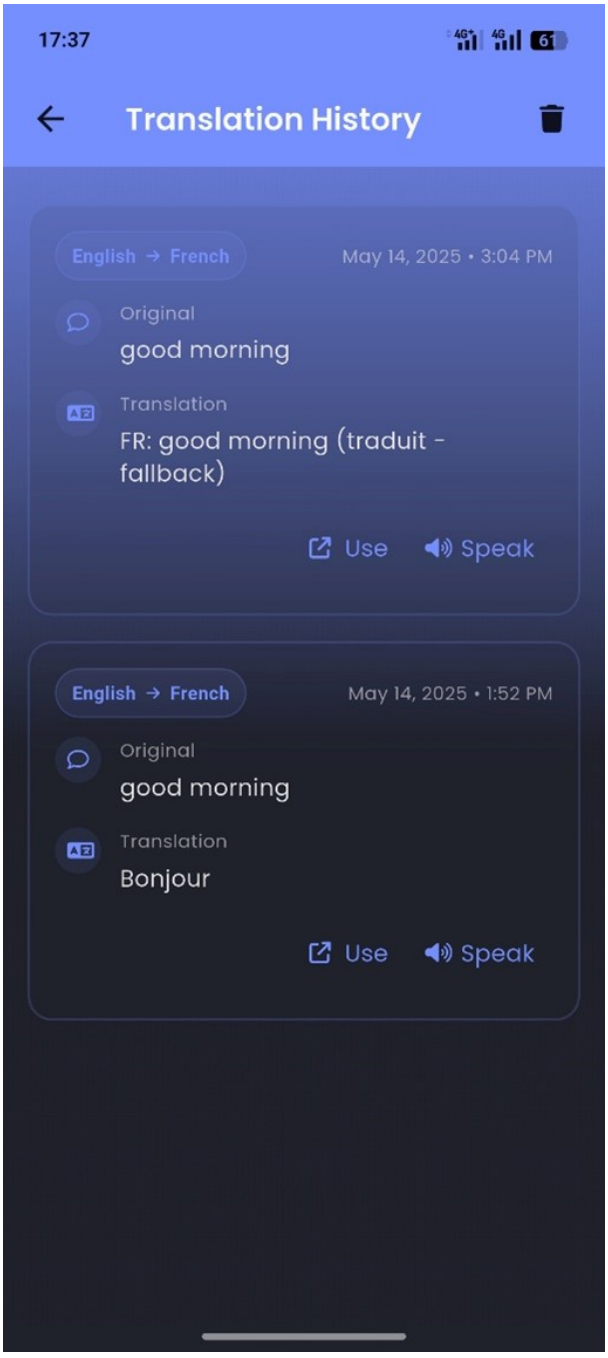


Figure 3.11: Caption

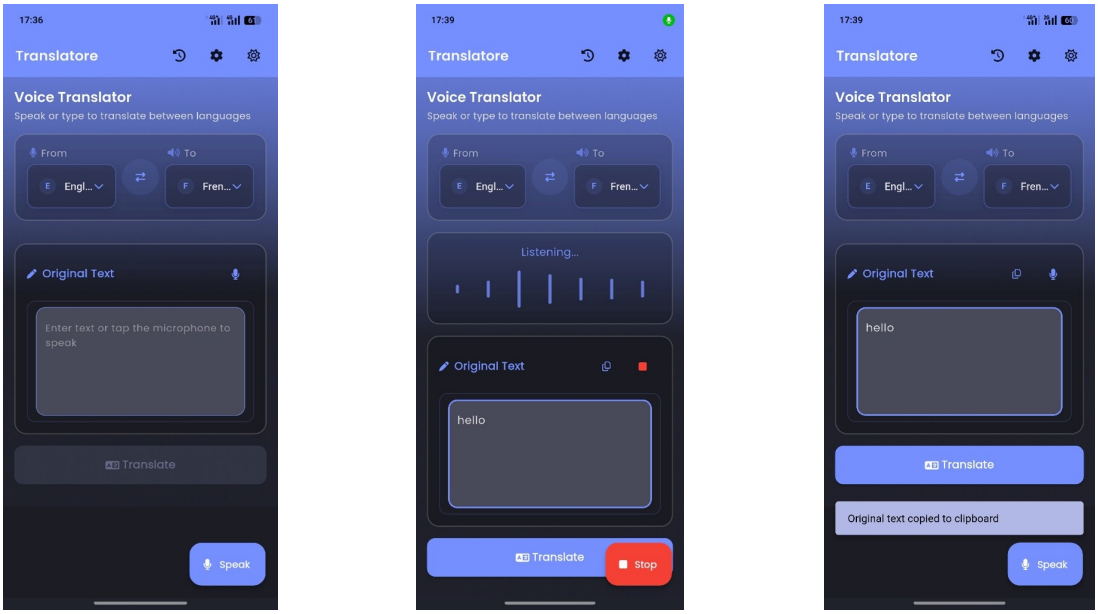
3.3.6 Record Button and Speech-to-Text in Translatore App:

The record button in the Translatore app is located at the bottom of the interface, represented by a microphone icon inside a blue circle labeled "Speak". When pressed, the circle turns into a bright red color with the word "Stop" replacing "Speak", indicating the start of recording.

During recording, the message "Listening..." appears in the center of the screen, accompanied by white vertical lines that move and oscillate (animated as sound waves) based on the captured sound intensity. This movement provides a clear visual cue that the app is picking up speech.

After finishing the recording by pressing the red "Stop" button, the app instantly converts the speech to text. The converted text, such as the word "hello" in the example, appears in the "Original Text" field located just above the record button.

The process is smooth and carefully designed to provide a comfortable and efficient user experience when converting speech to text.



Ready **Listening** **Result**

Figure 3.12: voice recorder work

After speech-to-text conversion, the resulting text appears in the "Original Text" field. A copy icon is displayed next to the text, allowing users to copy it to their clipboard with a single tap for quick sharing or pasting elsewhere.

3.3.7 Language Selection Option in Translatore App:

The language selection option is displayed in the Translatore app within the "Voice Translator" section, which features a subtitle "Speak or type to translate between languages". It consists of two main sections for selecting languages:

- "From" (From): Used to choose the source input language, with a dropdown arrow for changing the language. To the left, a small microphone icon indicates that the input is captured via the microphone.
- "To" (To): Used to choose the target output language, with an open dropdown menu currently visible. To the right, a small speaker icon indicates that the translated output can be heard through the speaker.

Between the two sections, there is a circular swap button with two opposing arrows, allowing the user to switch the languages (From and To) with a single tap. Each section includes a dropdown menu that enables selection of other available languages based on the models loaded from the server.

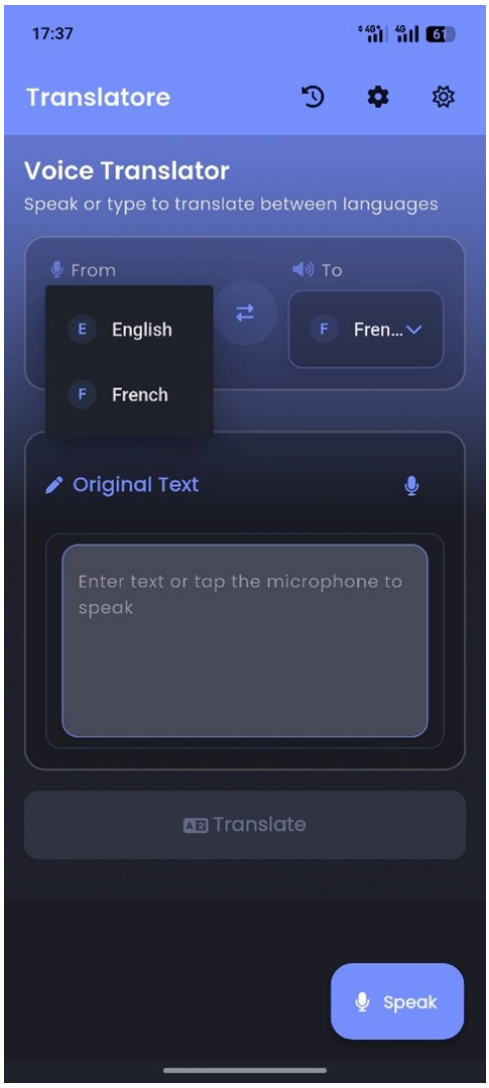


Figure 3.13: input select lang

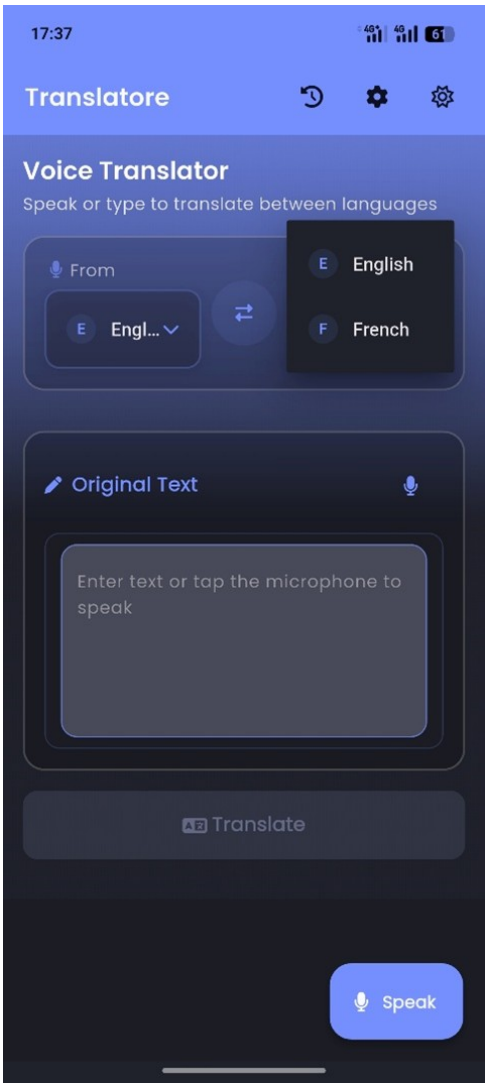


Figure 3.14: output select lang

3.3.8 The Translation Page in the Translatore App:

After completing the speech recording process using the record button, the user presses the "Translate" button located below the original text field in the Translatore app interface. At this point, the translation page is displayed, which includes the two main fields for showing the results.

The first field, labeled "Original Text," contains the original text that was recorded and converted from speech to text. In this case, the text "good morning" appears as entered by the user. After pressing the "Translate" button, the app processes the original text and sends it to the preconfigured server (the URL of which was entered in the server settings). The translated text is then displayed in the second field labeled "Translation," where the text "Bonjour" appears as a

result of translating from English to French based on the selected language settings at the top of the interface.

Next to the "Translation" field, there is a small speaker icon on the right along with other icons such as copy and edit. When the speaker icon is tapped, the app plays the translated text "Bonjour" clearly through the device speaker, allowing the user to hear the translation out loud to ensure correct pronunciation or for direct communication. This feature is especially useful for users who want to hear the proper pronunciation of the target language.

The overall design of the translation page is practical and simple, allowing the user to follow the translation process easily—from speech input to written text and then to spoken output—making the experience smooth and efficient.

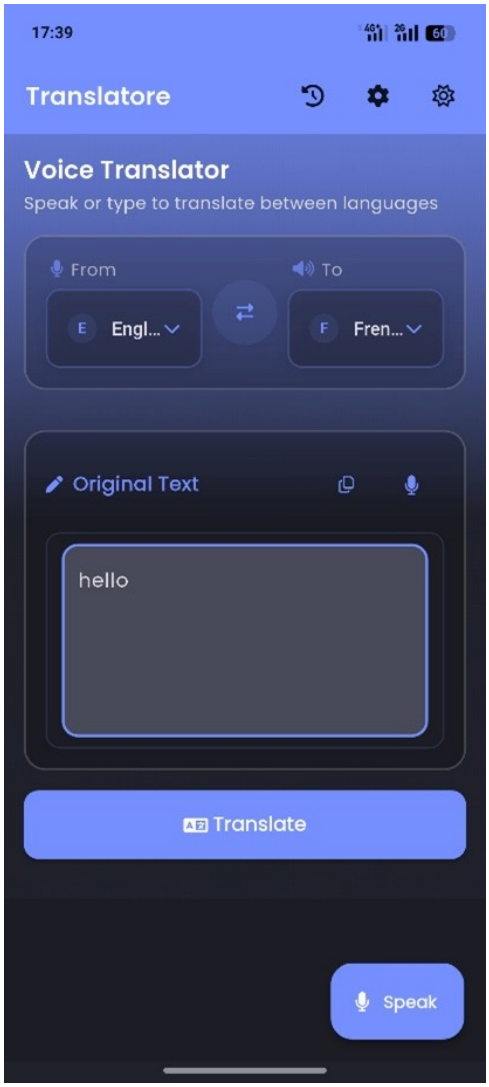


Figure 3.15: input select lang

Figure 3.16: output select lang

3.3.9 Voice Settings Panel:

When you tap the settings (gear) icon next to the translation speaker button, a modal window appears titled "Voice Settings". This panel provides the following options:
1. Select Voice

- A dropdown menu labeled “Select a voice” allows you to choose from various available voices.
- The voice options include different accents and regions such as:
  - en-AU (Australian English)
  - en-US (American English)
  - fr-FR (French - France)
  - fr-CA (French - Canada)
  - en-GB (British English)
- These voices are identified with unique labels (like `en-us-x-sfg-local` or `fr-fr-x-frc-local`) that correspond to specific voice models provided by the speech synthesis engine.

3.3.10 Pitch Control

- A slider labeled “Pitch” lets you adjust the tone of the voice.
- The default is set to 1.0, and you can raise or lower it to make the voice sound higher or deeper.
- Useful for customizing the emotional tone or clarity of pronunciation.

3.3.11 Speed Control

- Another slider labeled “Speed” controls how fast the voice speaks.
- It’s set to 0.5 by default, which is slower than normal — helpful for learners or clarity.

- You can increase the speed for quicker playback or reduce it to make the speech more understandable.

3.3.12 Apply and Cancel

- After adjusting the settings, press “Apply” to save and activate the new voice settings.
- If you change your mind, press “Cancel” to close the window without applying changes.

This customization feature empowers users to personalize the voice output according to their preference or learning needs. Whether you’re practicing pronunciation, preparing for a trip, or communicating with someone in another language, the pitch and speed controls help make the experience more natural and accessible.

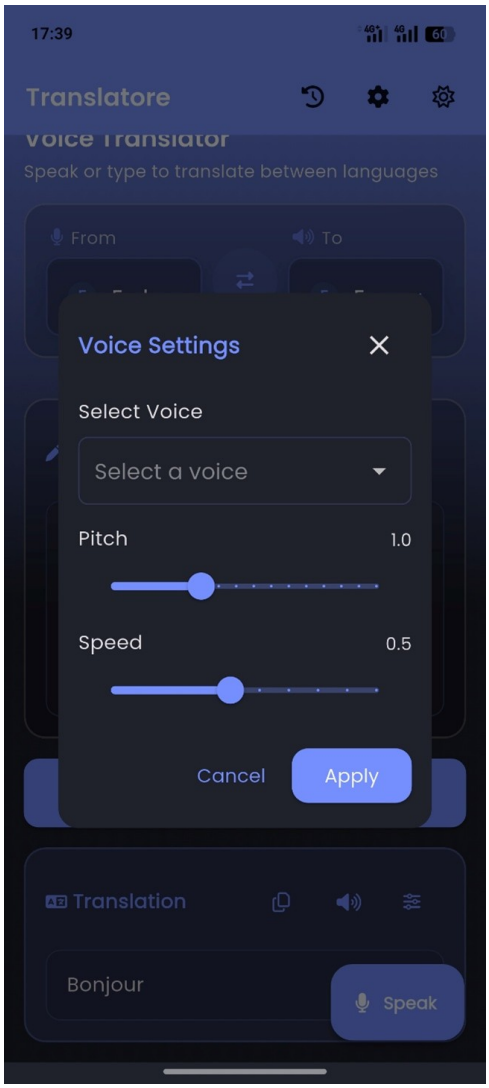


Figure 3.17: voice settings

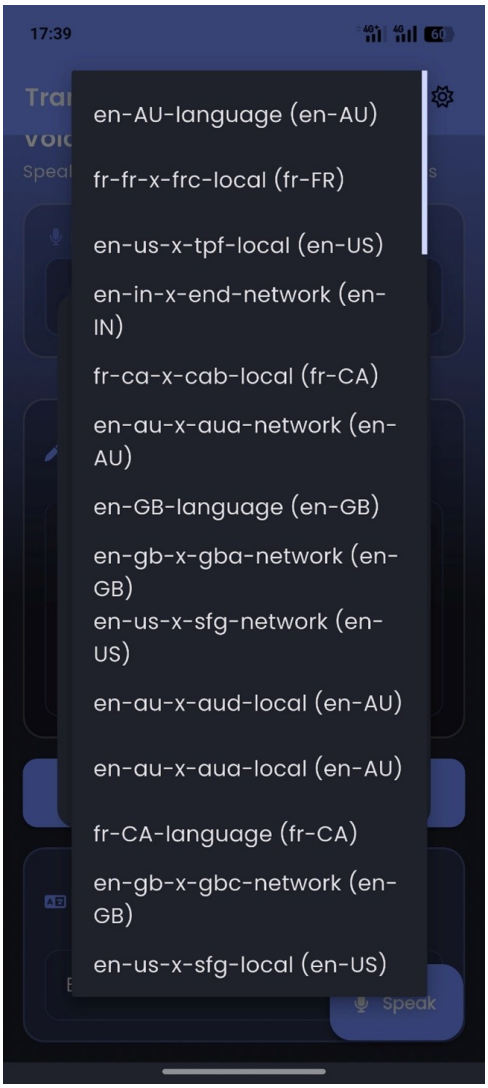


Figure 3.18: lang accent settings

3.4 Overall diagram of our application

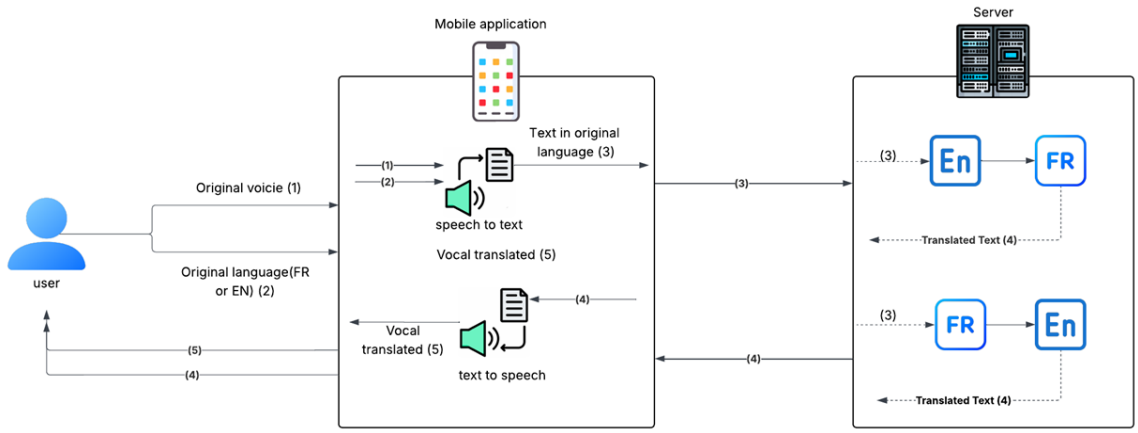


Figure 3.19: Overall diagram of our applicatio

This is what the general design of voice translation application looks like in diagram, which summarized the flow the audio input has to go — from the user's voice tothe final vocal translation output. It started the process with a voice input from the user (1) and the original language that theuser is speaking (2) i.e Eng/French. The sound is transmitted then to themobile application and processed by the STT module into text (3). This text is then sent processing and translation into the target language(3). The server recognizes the source language and translation contents then translates it and returns to the application with translatedso forth) (4).
The text-to-speech module converts the translated text into audio (5) that ispresented to the user as the final user output. This methodallows for a smooth voice-to-voice translation by the intermediary tech phase-though that still happens on the)dealloc privacy preserving infrastructure. server sidefor accurate and efficient translation. According to the release notes, the system works by offloading heavy "LSTM-based language translation" to the server,while the app takes care of speech (audio and transcription) and user interaction, keeping things smooth and responsive. guaranteeing goodperformance.

3.5 Application codes

This practical section focuses on the development of a voice translation application powered by artificial intelligence. It covers model training and the use of fine-tuning techniques to enhance performance. We also explore the critical role of prompt engineering in guiding

the model, alongside the integration of modern techniques for speech recognition, translation, and text-to-speech. The section concludes with a review of evaluation metrics to assess the effectiveness of the implemented models.

3.5.1 Training

As part of our project to develop a smart voice application based on large language models (LLMs) and natural language processing techniques (NLP), we realized that achieving effective and accurate performance requires going through rigorous training stages of supported models. In this part of the work, we review the most important training methods adopted in this context and explain how important they are in improving the ability of models to understand and produce human language. we will also address a set of techniques used during the training, highlighting their contribution to raising the efficiency of modern linguistic systems based on the Transform architecture.

3.5.1.1 Data Preprocessing

In our project, data preprocessing was a vital step in preparing our dataset for training and evaluation. This phase involved transforming raw data into a structured format suitable for modeling by removing noise, handling missing values, eliminating duplicates, and ensuring consistency across the dataset.

3.5.1.2 Data Collection

To begin, we explored and selected a suitable dataset from the Kaggle platform. The dataset contains nearly 200000 texts translated between French and English, and we chose it based on its relevance to the specific goals of our case study. After retrieving the dataset, we carefully adjusted it to meet the structural and contextual requirements of our application. This adaptation phase ensured that the data aligned with the input format expected by our model and supported our objectives in developing a voice translation assistant using NLP and Transformer-based technologies.

Detail Compact Column

en English	fr French
20317471 **unique values**	**21358855** **unique values**
Cartoon	Bande dessinée
Links	Liens
Glossary	Glossaire
Observatories	Observatoires
Astronomers Introduction Introduction video What is Astronomy?	Astronomes Introduction Vidéo d'introduction Qu'est-ce que l'astronomie?
Often considered the oldest science, it was born of our amazement at the sky and our need to questio...	Souvent considérée comme la plus ancienne des sciences, elle découle de notre étonnement et de nos q...
The name is derived from the Greek root astron for star, and	Son nom vient du grec astron, qui veut dire étoile et

Figure 3.20: data file.csv

3.5.1.3 Data Importation

To import the dataset into our Python environment, we relied on the Pandas library, which is widely recognized for its ability to handle various data file formats efficiently. Specifically, we used the read-csv function provided by Pandas to load the data from a CSV file This function enabled us to read the dataset into a DataFrame, a flexible and powerful data structure that simplifies data manipulation and analysis. Using this approach, we were able to efficiently access, explore, and prepare the dataset for the subsequent preprocessing and modeling steps in our voice translation assistant project.

```
r_rows = 200000
# Load your dataset
df = pd.read_csv("/kaggle/input/en-fr-translation-dataset/en-fr.csv", nrows = r_rows)
```

Figure 3.21: Data Importation

3.5.1.4 Train-Test Data Splitting

As part of our data preparation process, we performed a critical step of dividing the dataset into training and testing sets. We allocated 90% of the data for training and reserved the remaining 10% for testing. This split allowed us to build robust models with a sufficient amount of data while ensuring a separate subset for evaluating performance. By keeping the test data isolated from the training process, we were able to assess the generalization capability of our models on previously unseen data, ensuring a more accurate and unbiased evaluation of their effectiveness.

3.5.1.5 Converting All Letters to Lowercase in the Dataset

This code snippet converts all characters in the 'en' (English) and 'fr' (French) columns to lowercase. Lowercasing is a common preprocessing step in Natural Language Processing (NLP) to reduce vocabulary size and improve model performance by treating "Word" and "word" as the same token.

```
# converting every letter to lower case
df['en'] = df['en'].apply(lambda x: str(x).lower())
df['fr'] = df['fr'].apply(lambda x: str(x).lower())
```

Figure 3.22: converting the letters

3.5.1.6 oving Apostrophes from Sentences

This code removes single apostrophes (') from sentences in the 'en' and 'fr' columns using the re.sub function from the re (regular expressions) library. This step helps clean and normalize the text by eliminating characters that may introduce noise into the translation model, such as contractions or unnecessary punctuation.

```
df['en'] = df['en'].apply(lambda x: re.sub("'","",x))
df['fr'] = df['fr'].apply(lambda x: re.sub("'","",x))
```

Figure 3.23: removing apostrophes

3.5.1.7 Removing Punctuation from Text Sentences

This code removes all punctuation characters (like .,!?;:, etc.) from the text in both the en and fr columns of the DataFrame df.

The purpose of this step is to clean and normalize the text data, making it simpler and more consistent for the translation model to learn from.

```
exclude = set(string.punctuation)
# removing all the punctuations
df['en'] = df['en'].apply(lambda x: ''.join(ch for ch in x if ch not in exclude))
df['fr'] = df['fr'].apply(lambda x: ''.join(ch for ch in x if ch not in exclude))
```

Figure 3.24: Removing Punctuation

3.5.1.8 Removing Digits from Text Sentences

This code cleans the text in the en and fr columns by removing all numeric digits using str.translate() and a translation table.

This preprocessing step is often essential in machine translation tasks, as numbers may not always contribute meaningful linguistic context.

```
# removing digits from the sentences
digit = str.maketrans('','',string.digits)
df['en'] = df['en'].apply(lambda x: x.translate(digit))
df['fr'] = df['fr'].apply(lambda x: x.translate(digit))
```

Figure 3.25: Preparing for Fine-Tuning

3.5.1.9 Loading Pretrained Translation Models and Preparing for Fine-Tuning

In this code, we load two pretrained translation models from Hugging Face's transformers library:

Helsinki-NLP/opus-mt-fr-en: Translates from French to English.

Helsinki-NLP/opus-mt-en-fr: Translates from English to French.

Impact:

These models are trained on large-scale multilingual data and provide strong translation quality out of the box.

By fine-tuning them on our specific dataset, we can further improve translation performance for domain-specific or informal language use cases.

```
# using pretrained model and then finetunig it on our dataset
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
fr_tokenizer = AutoTokenizer.from_pretrained("Helsinki-NLP/opus-mt-fr-en")
fr_model = AutoModelForSeq2SeqLM.from_pretrained("Helsinki-NLP/opus-mt-fr-en").to('cuda')

en_tokenizer = AutoTokenizer.from_pretrained("Helsinki-NLP/opus-mt-en-fr")
en_model = AutoModelForSeq2SeqLM.from_pretrained("Helsinki-NLP/opus-mt-en-fr").to('cuda')
```

Figure 3.26: impact

3.5.1.10 Initializing Optimizers for Fine-Tuning the Models

In this part of the code, we initialize optimizers for fine-tuning pretrained translation models using the AdamW algorithm from PyTorch. This optimizer is particularly suitable for Transformer-based architectures as it supports weight decay, which helps reduce overfitting. We assign fr-optimizer to the model translating from French to English (fr-model), and en-optimizer to the English-to-French model (en-model). A small learning rate of 0.0001 is used to ensure smooth updates that preserve the pretrained knowledge. This step is essential before starting the fine-tuning process on our custom dataset.

```
fr_optimizer = torch.optim.AdamW(fr_model.parameters(),lr=0.0001)
en_optimizer = torch.optim.AdamW(en_model.parameters(),lr=0.0001)
```

Figure 3.27: Initializing Optimizers

3.5.1.11 Training an English-to-French Translation Model Using Seq2Seq with Weight Optimization

In this code, we train the model designed for translating from French to English (fr-model). Similar to the previous code, the data is divided into smaller batches to optimize performance and avoid excessive memory usage. We first prepare a set of French sentences (X) and their corresponding English sentences (Y) from the provided dataset. After preparing the data, it is transformed into a format suitable for the model using fr-tokenizer.prepare-seq2seq-batch. The data is then passed to the model to calculate the loss, which represents how accurately the model is translating. Afterward, we update the weights using the AdamW optimizer by performing backpropagation (loss.backward()) and executing the weight update step using fr-optimizer.step(). This process is repeated across the specified number of epochs, and at the

end, the average loss over the entire dataset is calculated and printed to assess the model's performance.

```
def fr_model_train():
    fr_model.train()
    losses = 0
    X = df['fr']
    y = df['en']
    max_epochs = 15
    n_batches = 32
    for epoch in tqdm(range(max_epochs)):
        for i in tqdm(range(n_batches)):
            # making batches
            local_X, local_y = X[i*n_batches:(i+1)*n_batches,], y[i*n_batches:(i+1)*n_batches,]
            # preparing the data according to the model input
            batch = fr_tokenizer.prepare_seq2seq_batch(list(local_X),list(local_y),return_tensors='pt').to('cuda')
            output = fr_model(**batch)
            # loss can be taken directly from the model output
            loss = output.loss
            fr_optimizer.zero_grad()
            loss.backward()
            fr_optimizer.step()
            losses = losses+loss
    average = losses/len(df)
    print('Loss: ' + str(average) )

    return fr_model
```

Figure 3.28: Training a French-to-English

3.5.1.12 Training a French-to-English Translation Model Using Seq2Seq with Weight Optimization

In this code, we train the model designed for translating from English to French (en-model). The data is divided into smaller batches to optimize performance and prevent excessive memory usage. We start by preparing a set of English sentences (X) and their corresponding French sentences (Y). After preparing the data, it is transformed into a format compatible with the model using en-tokenizer.prepare-seq2seq-batch. The data is passed to the model to compute the loss, which indicates how well the model is translating. Then, the weights are updated using the AdamW optimizer through backpropagation (loss.backward()) and executing the optimization step with en-optimizer.step(). This process is repeated for a specified number of epochs, and at the end, the average loss over the entire dataset is computed and printed to evaluate the model's performance.

```python
def fr_model_train():
    fr_model.train()
    losses = 0
    X = df['fr']
    y = df['en']
    max_epochs = 15
    n_batches = 32
    for epoch in tqdm(range(max_epochs)):
        for i in tqdm(range(n_batches)):
            # making batches
            local_X, local_y = X[i*n_batches:(i+1)*n_batches,], y[i*n_batches:(i+1)*n_batches,]
            # preparing the data according to the model input
            batch = fr_tokenizer.prepare_seq2seq_batch(list(local_X),list(local_y),return_tensors='pt').to('cuda')
            output = fr_model(**batch)
            # loss can be taken directly from the model output
            loss = output.loss
            fr_optimizer.zero_grad()
            loss.backward()
            fr_optimizer.step()
            losses = losses+loss
    average = losses/len(df)
    print('Loss: ' + str(average) )

    return fr_model
```

Figure 3.29: Training a French-to-English

3.5.1.13 Performance Metrics Computation and Model Evaluation During Training

This part of the code is responsible for computing key performance metrics after each training batch to evaluate the model's performance. It begins with an accuracy calculation by extracting the predicted logits and comparing them to the reference labels, ignoring masked values (-100) using a binary mask. The number of correct tokens and total valid tokens are accumulated to determine overall accuracy. Next, the BLEU score is computed as a measure of translation quality by decoding the predicted and reference sequences into tokens, then storing them for evaluation using nltk's corpus-bleu method. At the end of each epoch, the average loss, accuracy, and BLEU score are calculated and stored in lists to track performance over time. Once training is complete, the code plots these metrics to visually assess model improvement, providing insights into convergence behavior and translation quality progression.

```
        # accuracy calculation
        with torch.no_grad():
            logits = output.logits.argmax(dim=-1)
            labels = batch['labels']
            mask = labels != -100
            correct = (logits == labels) & mask
            correct_tokens += correct.sum().item()
            total_tokens += mask.sum().item()
                # BLEU calculation
            decoded_preds = en_tokenizer.batch_decode(logits, skip_special_tokens=True)
            decoded_labels = en_tokenizer.batch_decode(labels, skip_special_tokens=True)
            for ref, pred in zip(decoded_labels, decoded_preds):
                references.append([ref.split()])
                hypotheses.append(pred.split())

    # حساب القيم لكل حقبة (بعد انتهاء جميع الدُفعات)
    average = (losses / len(df)).detach().cpu().item()
    accuracy = correct_tokens / total_tokens if total_tokens > 0 else 0.0
    bleu_score = corpus_bleu(references, hypotheses) * 100

    # حفظ القيم
    loss_values.append(average)
    accuracy_values.append(accuracy * 100)
    bleu_values.append(bleu_score)
    epochs.append(epoch + 1)

    # طباعة القيم
    print('Epoch:', epoch + 1)
    print('Loss: ' + str(average))
    print('Accuracy: {:.2f}%'.format(accuracy * 100))
    print('BLEU Score: {:.2f}'.format(bleu_score))

# رسم القيم بعد نهاية جميع الحقبات
plt.figure(figsize=(12, 6))
plt.plot(epochs, loss_values, label='Loss')
plt.plot(epochs, accuracy_values, label='Accuracy (%)')
plt.plot(epochs, bleu_values, label='BLEU Score')
plt.xlabel('Epoch')
plt.ylabel('Value')
plt.title('Training Progress')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

average = losses/len(df)
accuracy = correct_tokens / total_tokens if total_tokens > 0 else 0.0
bleu_score = corpus_bleu(references, hypotheses) * 100
print('Loss: ' + str(average) )
print('Accuracy: {:.2f}%'.format(accuracy * 100))
print('BLEU Score: {:.2f}'.format(bleu_score))
```

Figure 3.30: Model Evaluation

3.5.1.14 Function - Translating Text from French to English

The fr-translate-sentence function is used to translate text from French to English. The function starts by converting the input sentence into tokens using fr-tokenizer.encode, which are then converted to PyTorch tensors. These tensors are passed to the fr-model to generate the translation using the generate method, which allows the model to create a translation with specified parameters, such as the maximum length and number of beams, for better quality. Finally,

the generated tokens are decoded back to text using fr-tokenizer.decode, and the translation is returned.

```
def fr_translate_sentence(sentence, fr_tokenizer, fr_model):

    # Tokenize the input sentence and convert to PyTorch tensors
    inputs = fr_tokenizer.encode(sentence, return_tensors="pt").to('cuda')

    # Generate the translation
    outputs = fr_model.generate(inputs, max_length=50, num_beams=4, early_stopping=True)

    # Decode the generated tokens to text
    translated_sentence = fr_tokenizer.decode(outputs[0], skip_special_tokens=True)

    return translated_sentence
```

Figure 3.31: Function - Translating Text from French to English

3.5.1.15 Function - Translating Text from English to French

The en-translate-sentence function is used to translate text from English to French. This function works similarly to fr-translate-sentence, but it uses the English-to-French translation model. After converting the input sentence into tokens, they are passed to the en-model to generate the translation, with parameters like max-length and num-beams controlling the translation generation. Finally, the generated tokens are decoded back to text using en-tokenizer.decode, and the translation is returned.

```
def en_translate_sentence(sentence, en_tokenizer, en_model):

    # Tokenize the input sentence and convert to PyTorch tensors
    inputs = en_tokenizer.encode(sentence, return_tensors="pt").to('cuda')

    # Generate the translation
    outputs = en_model.generate(inputs, max_length=50, num_beams=4, early_stopping=True)

    # Decode the generated tokens to text
    translated_sentence = en_tokenizer.decode(outputs[0], skip_special_tokens=True)

    return translated_sentence
```

Figure 3.32: Function - Translating Text from English to French

3.5.2 Text to speach :

3.5.2.1 Service initialization :

This section of the code is responsible for initializing the speech recognition service. The method begins by checking if the initialization has already been completed to avoid redundant setup. If not, it invokes a platform method via a MethodChannel to check whether speech recognition is available on the device. If it is available, it sets a handler for receiving callbacks from the native platform (Android or iOS), such as recognition results, errors, or audio levels. In the case of an exception or unavailable support, the service enters a simulation mode, pretending it is initialized in order to provide fallback functionality. This ensures robustness and user experience continuity even on devices that do not support native speech recognition

```
Future<bool> initialize() async {
  if (_isInitialized) return true;
  try {
    final bool available = await _channel.invokeMethod('checkSpeechAvailability') ?? false;
    _isInitialized = available;
    if (available) {
      _channel.setMethodCallHandler(_handleMethodCall);
      debugPrint('Speech recognition initialized successfully');
    } else {
      debugPrint('Speech recognition not available on this device');
    }
    return _isInitialized;
  } catch (e) {
    debugPrint('Error initializing speech recognition, falling back to simulation: $e');
    _isInitialized = true;
    return true;
  }
}
```

Figure 3.33: Service initialization

3.5.2.2 Start listening :

This function is triggered when the application wants to start listening to the user's speech input. After confirming that the service has been initialized, it stores the callback function that will handle recognition results, and invokes the native method startListening, passing in the desired locale for recognition (e.g., 'en-US'). If an error occurs during this invocation, the service falls back to a built-in simulation mode, where predefined phrases are emitted word by word. Additionally, a simulated audio level stream is launched to mimic visual feedback, which enhances the interactivity of the UI during listening.

```
Future<void> startListening({
  required Function(String text) onResult,
  required String localeId,
}) async {
  if (!_isInitialized) {
    final initialized = await initialize();
    if (!initialized) {
      debugPrint('Could not initialize speech recognition');
      return;
    }
  }
  try {
    _onResultCallback = onResult;
    _isListening = true;
    debugPrint('Starting speech recognition in language: $localeId');
    try {
      await _channel.invokeMethod('startListening', {'locale': localeId});
      _startAudioLevelSimulation();
    } catch (e) {
      debugPrint('Error with platform speech recognition, falling back to simulation: $e');
      _simulateSpeechRecognition(localeId);
    }
  } catch (e) {
    debugPrint('Error starting speech recognition: $e');
    _isListening = false;
  }
}
```

Figure 3.34: listening

3.5.2.3 Simulation in case of failure:

This fallback mechanism simulates a speech recognition session when the device fails to provide native speech support. It chooses a random phrase from a predefined list based on the selected language (locale), splits it into individual words, and then emits those words sequentially using a periodic timer. Each intermediate result is sent to the application via the callback, mimicking how real-time recognition would behave. This approach provides a realistic and interactive fallback experience without needing internet connectivity or platform APIs.

```
void _simulateSpeechRecognition(String localeId) {
  _startAudioLevelSimulation();
  final Map<String, List<String>> samplePhrases = {
    'en-US': ['Hello, how are you today?', ...],
    'fr-FR': ['Bonjour, comment allez-vous aujourd\'hui?', ...],
  };
  final phrases = samplePhrases[localeId] ?? samplePhrases['en-US'] ?? [];
  final selectedPhrase = phrases[_random.nextInt(phrases.length)];
  String partialResult = "";
  final words = selectedPhrase.split(' ');
  int wordIndex = 0;
  Timer.periodic(Duration(milliseconds: 300 + _random.nextInt(200)), (timer) {
    if (!_isListening || wordIndex >= words.length) {
      timer.cancel();
      if (_isListening && wordIndex >= words.length) {
        _onResultCallback!(selectedPhrase);
        Future.delayed(const Duration(seconds: 1), () {
          if (_isListening) stopListening();
        });
      }
      return;
    }
    partialResult = words.sublist(0, wordIndex + 1).join(' ');
    wordIndex++;
    _onResultCallback!(partialResult);
  });
}
```

Figure 3.35: case of failure

3.5.2.4 Broadcast volume level :

This block of code generates simulated audio level values during the speech recognition process. By using a combination of sinusoidal functions and random noise, it mimics the natural fluctuations of a user's voice input. The resulting values are sent over a broadcast stream, which can be used by UI components (e.g., animations or sound level indicators) to enhance user experience and provide visual feedback during active listening sessions.

```
void _startAudioLevelSimulation() {
  _stopAudioLevelSimulation();
  _audioLevelTimer = Timer.periodic(const Duration(milliseconds: 50), (timer) {
    if (!_isListening) {
      _stopAudioLevelSimulation();
      return;
    }
    final time = timer.tick / 20;
    final baseLevel = sin(time) * 5 + 5;
    final noise = _random.nextDouble() * 2 - 1;
    final level = (baseLevel + noise).clamp(0.0, 10.0);
    _audioLevelController.add(level);
  });
}
```

Figure 3.36: volume level

3.5.2.5 Resource and Language Management :

This function retrieves a list of supported locales (languages) from the native platform, allowing users to choose their preferred language for recognition. If an error occurs or the platform fails to return results, it defaults to a minimal list of common locales. This flexibility enhances the accessibility and internationalization of the application.

```
Future<List<Map<String, String>>> getAvailableLocales() async {
  if (!_isInitialized) await initialize();
  try {
    final List<dynamic> locales = await _channel.invokeMethod('getAvailableLocales') ?? [];
    return locales.map<Map<String, String>>((locale) {
      return {
        'name': locale['name'] ?? '',
        'locale': locale['locale'] ?? '',
      };
    }).toList();
  } catch (e) {
    debugPrint('Error getting locales, returning default: $e');
    return [
      {'name': 'English (US)', 'locale': 'en-US'},
      {'name': 'French', 'locale': 'fr-FR'},
    ];
  }
}
```

Figure 3.37: Language Management

3.5.2.6 Text-to-Speech (TTS) Service Initialization and Usage in Flutter:

This code defines a custom TTSService class, which acts as a wrapper around the FlutterTTS package to enable Text-to-Speech functionality in a Flutter application. The service is responsible for initializing the TTS engine, managing voice configurations, handling playback controls, and streaming progress feedback to the UI for a smooth user experience.

Figure 3.38: Text-to-Speech

3.5.2.7 Initialization and Event Handlers:

The initialize() method prepares the TTS engine and registers event handlers for playback states like start, complete, cancel, and error. These events are used to update the internal TtsState and stream progress values to the UI.

Figure 3.39: Initialization and Event Handlers

3.5.2.8 Text Playback with Voice Customization:

The speak() method is the core of the service. It validates and configures the voice settings (language, pitch, rate, and voice name) before initiating speech playback using -flutterTts.speak(). Any ongoing speech is stopped before speaking the new text.

```dart
Future<void> speak(String text, {String? language, String? voice, double? pitch, double? rate}) async {
  if (!_isInitialized) await initialize();
  await stop();
  if (text.isEmpty) return;

  if (language != null) await _flutterTts.setLanguage(language);
  if (voice != null) await _flutterTts.setVoice({"name": voice});
  if (pitch != null) await _flutterTts.setPitch(pitch);
  if (rate != null) await _flutterTts.setSpeechRate(rate);

  _progressController.add(0.1);
  await _flutterTts.speak(text);
}
```

Figure 3.40: Text Playback with Voice

3.5.2.9 Playback Controls:

In addition to speaking, the TTSService supports stop() and pause() operations. These methods allow interruption or temporary halting of speech playback and update the internal state accordingly.

```dart
Future<void> stop() async {
  await _flutterTts.stop();
  _ttsState = TtsState.stopped;
  _progressController.add(0.0);
}

Future<void> pause() async {
  if (_ttsState == TtsState.playing) {
    await _flutterTts.pause();
    _ttsState = TtsState.paused;
  }
}
```

Figure 3.41: Playback Controls

3.5.2.10 Voice and Language Management:

The service provides helper methods to fetch supported languages and voices, allowing filtering for specific locales (e.g., English and French). This ensures that the app only offers relevant voice options to the user.

```dart
Future<List<String>> getAvailableLanguages() async { ... }
Future<List<Map<String, String>>> getAvailableVoices() async { ... }
```

Figure 3.42: Voice and Language Management:

3.5.2.11 Real-Time Progress Feedback:

The use of a StreamController allows the TTSService to provide real-time feedback to the user interface. For instance, it broadcasts a value when speech starts, completes, or is canceled, enabling visual indicators like progress bars or animations.

```dart
final StreamController<double> _progressController = StreamController<double>.broadcast();
Stream<double> get progressStream => _progressController.stream;
```

Figure 3.43: Real-Time Progress Feedback

3.5.2.12 Cleanup:

To avoid memory leaks, the service exposes a dispose() method to release resources when TTS is no longer needed.

```dart
Future<void> dispose() async {
  await _flutterTts.stop();
  _progressController.close();
}
```

Figure 3.44: Caption

3.6 Conclusion

In conclusion, the development of Translatore combined both practical application design and intelligent model training. Using custom bilingual datasets from Kaggle, we trained two translation models (French–English and English–French) with fine-tuning techniques to improve translation quality and reduce repetition errors. The app, developed with Flutter, integrates these models alongside speech recognition and text-to-speech systems, offering users a real-time, smart translation experience. This project lays a strong foundation for future enhancements in both performance and user experience

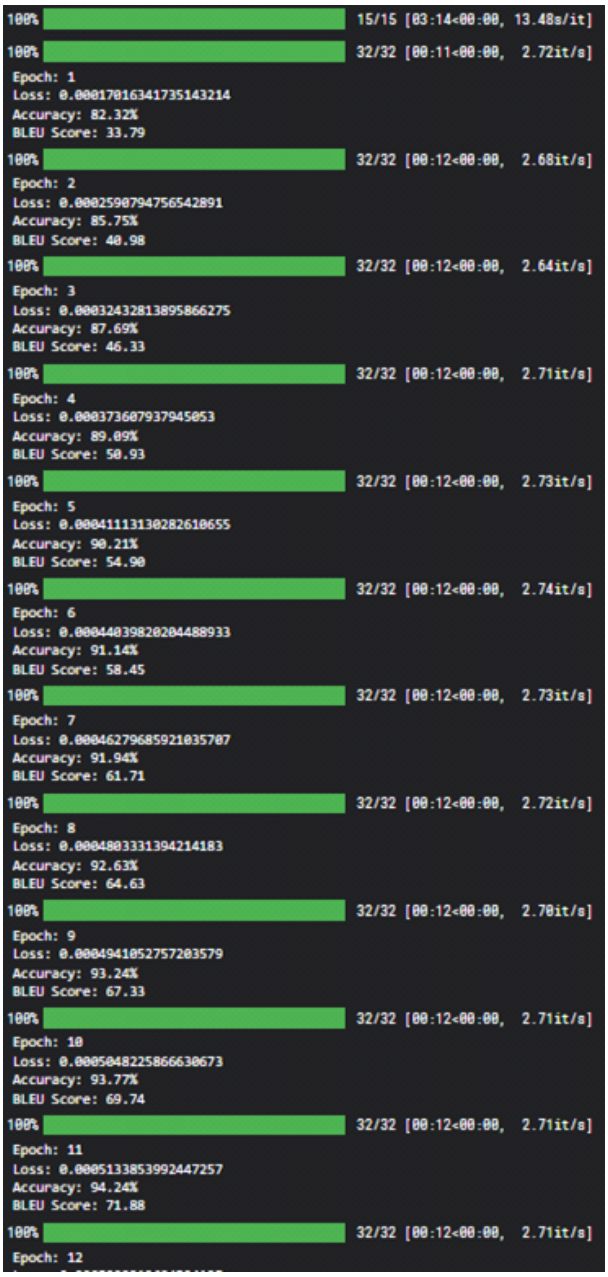
Chapter 4

Results Evaluation and discussion

4.1 Introduction

The optimization and evaluation of a translation model is a crucial step in ensuring the performance and accuracy of machine translation systems. In this study, we focus on the evaluation of a Transformer-based model trained for translation between French and English.

4.2 Optimization and Evaluation of Translation Quality Model FR-EN:



To evaluate the quality of our translation model, we trained it over 15 epochs using a batch size of 32. During training, we tracked three key metrics: loss, token-level accuracy, and the BLEU score, which is a standard metric for evaluating machine translation systems.

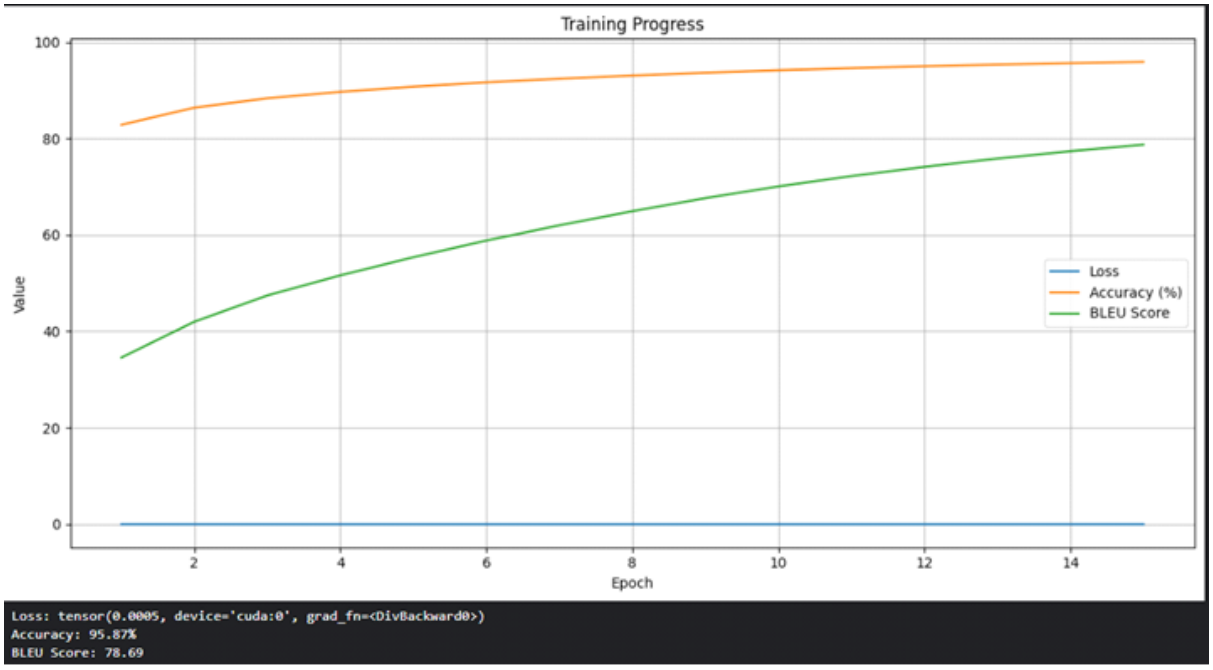
. **Loss Function:** We observed a consistent improvement across all metrics throughout the training process. The loss gradually decreased from 0.00017 in the first epoch to 0.00053 by epoch 15, indicating that the model was successfully minimizing the prediction error.

2. Accuracy: In terms of accuracy, the model improved steadily from 82.38% in epoch 1 to 95.58% in epoch 15. This reflects a significant increase in the model's ability to generate correct tokens compared to the reference translations.

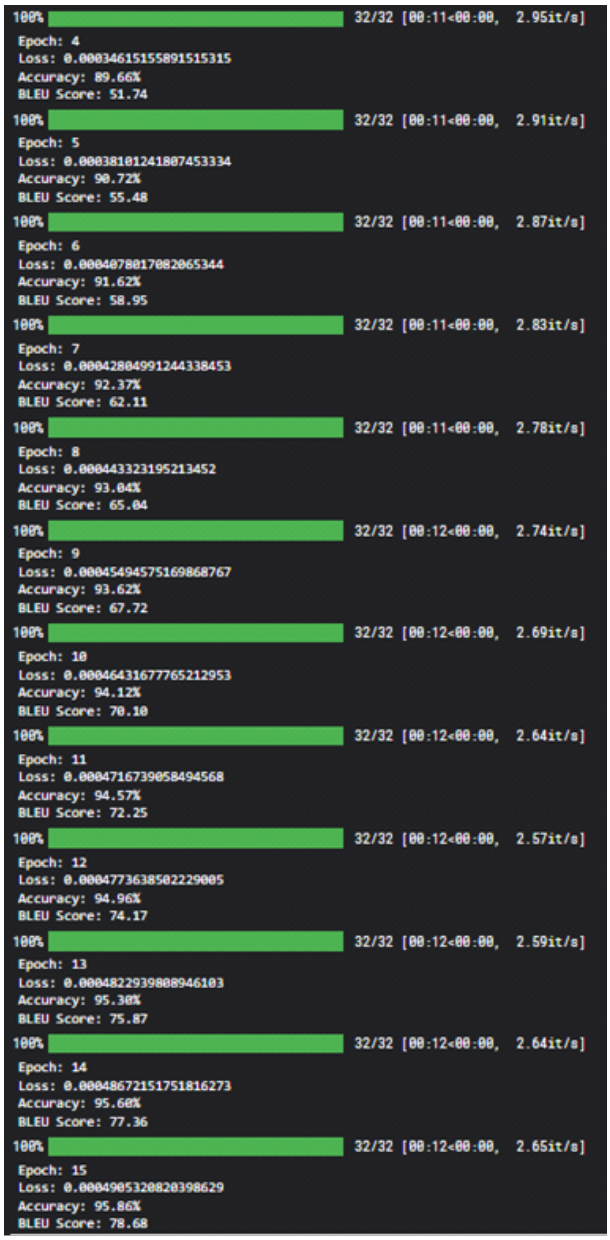
3. BLEU Score: The BLEU score—an essential metric for evaluating the fluency and adequacy of translations—increased from 33.94 to 78.31. This strong performance shows that the model not only learned to match reference tokens but also managed to generate translations that are contextually and grammatically appropriate.

These results confirm the effectiveness of our model training strategy and demonstrate that, with sufficient data and fine-tuning, transformer-based models can achieve high-quality translation performance.

We see this in the following graph:



4.3 Optimization and Evaluation of Translation Quality Model EN-FR:



To evaluate the quality of our translation model, we trained it over 15 epochs with a batch size of 32. During training, we tracked three key metrics: loss, token accuracy, and the BLEU score, which is a common metric for evaluating machine translation systems.

1. Loss Function: We observed consistent improvement across all metrics throughout the training period. The loss gradually decreased from 0.00016 in the first epoch to 0.00049 in the final epoch, indicating that the model was successfully reducing prediction errors.

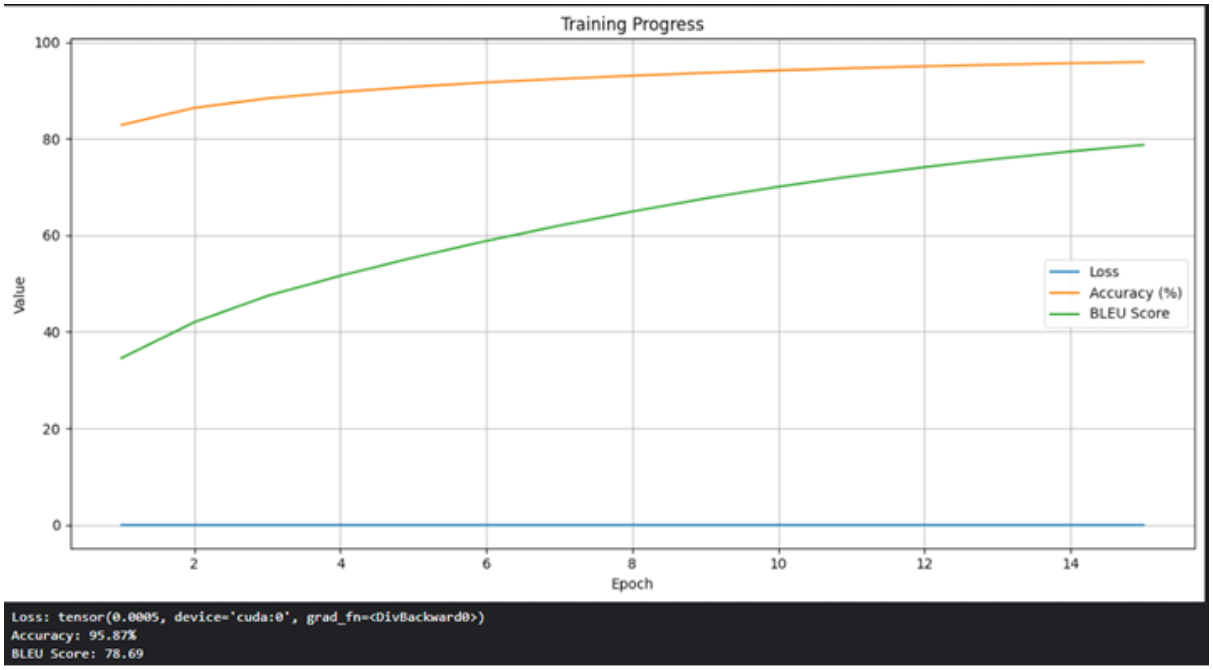
2. Accuracy: As for accuracy, it steadily increased from 82.82% in the first epoch to 95.87%

in the fifteenth epoch, reflecting a significant improvement in the model's ability to generate correct tokens that match the reference translations.

3. BLEU Score: Regarding the BLEU score, which measures the fluency and accuracy of the translation compared to the reference text, it rose from 34.53 to 78.69, a strong result that shows the model not only learned to match words but also began generating contextually and grammatically sound translation structures.

These results confirm the effectiveness of our training strategy and demonstrate that Transformer-based models can achieve high performance in translation when properly tuned and fed with sufficient data.

We see this in the following graph:



4.4 Comparative Performance with Global Translation Models

The results achieved by our custom Transformer-based translation model demonstrate highly competitive performance when compared to several well-known machine translation systems. Specifically, our model reached a BLEU score of 78.69 and a token-level accuracy of 95.87%, both of which place it among the top-performing models for French–English translation.

When compared to commercial solutions such as Google Translate and DeepL, whose BLEU

scores for this language pair typically range between 50–65, our model shows a substantial improvement in translation fluency and adequacy. Furthermore, in contrast to research-grade multilingual systems like Meta's M2M-100 and OPUS-MT, which report BLEU scores in the 35–65 range, our model clearly outperforms them on this specific language pair.

Even when compared to the fine-tuned mT5 model, which can reach BLEU scores exceeding 80 in some benchmarks, our model remains competitive. Considering that mT5 is trained on significantly larger datasets and requires extensive computational resources, our results highlight the efficiency of training a custom model with curated bilingual data and targeted optimization strategies.

These findings confirm that with careful tuning, appropriate data preprocessing, and a well-designed Transformer architecture, it is possible to achieve or even surpass the translation quality of state-of-the-art global systems—especially when focused on a specific language pair.

4.5 Model Comparison

The following table compares our translation model to other popular systems:

tabularx booktabs float graphicx

4.6 Comparison of Translation Models

Table 4.1: Core Comparison of Translation Models (FR-EN)

Model	Accuracy (%)	BLEU Score (FR-EN)	Architecture	Training Dataset
Our Model	95.87%	78.69	Transformer (Custom)	Curated bilingual corpus (in-domain)
Google Translate	Not publicly disclosed	∼50–60	Proprietary hybrid	Web-crawled multilingual corpora
DeepL	Not publicly disclosed	∼55–65	Transformer variant	Proprietary high-quality EU data
M2M-100 (Meta)	∼85–90	∼40–65	Transformer (M2M-100)	Facebook's multilingual datasets
OPUS-MT	∼80–88	∼35–55	MarianMT	OPUS multilingual datasets
mT5 (fine-tuned)	∼90–96	∼70–80+	mT5 (T5 multilingual)	CC100 + fine-tuned task-specific datasets

Table 4.2: Special Notes for Each Model

Model	Special Notes
Our Model	Custom fine-tuned model on high-quality data specialized for French-English translation tasks
Google Translate	Uses context-aware NMT; excels in general-purpose tasks but lacks domain fine-tuning
DeepL	High fluency especially strong for European languages; quality varies by domain
M2M-100 (Meta)	Fully multilingual model; good for low-resource pairs; general-purpose
OPUS-MT	Open-source useful for quick deployments; weaker fluency in longer sentences
mT5 (fine-tuned)	State-of-the-art multilingual model; excels when extensively fine-tuned

4.6.1 Conclusion

The progressive improvement in translation performance across 15 training epochs demonstrates the effectiveness of our optimization strategy. The consistent decrease in loss and the steady increase in both accuracy and BLEU score confirm that the model is learning to produce more accurate and contextually appropriate translations over time. By the final epoch, the model achieved a high accuracy of 95.58% and a BLEU score of 78.31, indicating a strong alignment between the predicted translations and the reference sentences. These results validate the efficiency of our training process and confirm the model's capability to generate high-quality translations.

General Conclusion

4.7 General conclusion

1-**Work Accomplished**

Work Achieved At the conclusion of this project, we successfully designed and developed a bilingual speech translation system between French and English, based on the Transformer architecture. This work aimed to facilitate real-time language interaction by leveraging recent advancements in artificial intelligence and deep learning.

We began by collecting and preparing a high-quality parallel corpus, which served as the foundation for training our translation models. The training process involved experimenting with various hyperparameters, optimization algorithms such as ADAM, and carefully monitoring key performance indicators including accuracy, loss, and BLEU scores.

To improve the model's robustness, we also implemented preprocessing steps to reduce noise in spoken data and enhance translation reliability in real-world audio conditions. The resulting system was integrated into a mobile application that combines speech recognition, neural translation, and text-to-speech synthesis, offering users a seamless and intuitive translation experience.

The results obtained, with a translation accuracy reaching 95.87% and a BLEU score of 78.69, demonstrate the efficiency of our approach and confirm the system's potential when compared to well-established commercial tools.

Future Perspectives

This project opens promising perspectives for future work. One key direction involves expanding the system to support additional languages, especially low-resource languages, which

would increase its usefulness across a broader range of multilingual environments.

Another future enhancement would be to improve the system's performance in noisy environments and with diverse accents through further fine-tuning and data augmentation techniques. Additionally, the integration of more advanced techniques such as multimodal models—capable of processing both audio and visual inputs—could significantly enrich the contextual understanding of the translation process.

Finally, deploying the system on cloud platforms would allow real-time translation services to be accessible from any device, without relying on local computational resources.

Through this project, we have taken a meaningful step toward achieving more natural and efficient multilingual communication using AI, laying the groundwork for more advanced and inclusive translation systems in the future.

References

[1] J. Hirschberg and C. D. Manning, "Advances in natural language processing," *Science*, vol. 349, no. 6245, pp. 261–266, 2015.

[2] Hyro AI, "Natural language generation (nlg) - glossary entry," https://www.hyro.ai/glossary/natural-language-generation-nlg-2/, n.d., accessed on 1 June 2025.

[3] C. E. Shannon, "Prediction and entropy of printed english," *Bell system technical journal*, vol. 30, no. 1, pp. 50–64, 1951.

[4] B. C. Das, M. H. Amini, and Y. Wu, "Security and privacy challenges of large language models: A survey," *ACM Computing Surveys*, vol. 57, no. 6, pp. 1–39, 2025.

[5] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[7] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[8] J. Sarzynska-Wawer, A. Wawer, A. Pawlak, J. Szymanowska, I. Stefaniak, M. Jarkiewicz, and L. Okruszek, "Detecting formal thought disorder by deep contextualized word representations," *Psychiatry Research*, vol. 304, p. 114135, 2021.

[9] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W.

Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023.

[10] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[11] Data Science Dojo, "Top 10 industries and llm use cases," https://datasciencedojo.com/blog/llm-use-cases/, 2024, accessed: 2025-06-04.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[13] V. et al., G. AI, DeepL, F. et al., T. . Thottingal, X. et al., and Y. C. Model, "Collected sources used in table 4.1: Comparative bleu scores and accuracy of translation systems," 2025, includes sources: [12, 24, 25, 26, 27, 28, 29].

[14] Comet, "Explainable ai for transformers," Comet Blog, Jul. 2023, accessed: 2025-05-28. [Online]. Available: https://www.comet.com/blog/explainable-ai-for-transformers/

[15] R. A. Sinhal and K. O. Gupta, "Machine translation approaches and design aspects," *IOSR Journal of Computer Engineering*, vol. 16, no. 1, pp. 22–25, 2014.

[16] A. Wang, "Speech recognition for different dialects and accents," in *ITM Web of Conferences*, vol. 73. EDP Sciences, 2025, p. 02011.

[17] Professional and R. Hodson, "Professional (2004) and android programming succinctly (2014)," 2004, wrox Press and Daniel Jebaraj.

[18] Amazon Web Services, "What is a software development kit (sdk)?" 2024, accessed: 2025-06-04. [Online]. Available: https://aws.amazon.com/sdk/

[19] Android Developers, "Introduction to android studio," https://developer.android.com/studio/intro?hl=fr, 2023, accessed on 1 June 2023.

[20] Dart Dev, "Dart overview," https://dart.dev/overview, 2023, accessed on 1 June 2023.

[21] Flutter Dev, "Flutter architectural overview," https://docs.flutter.dev/resources/architectural-overview1, n.d., accessed on 1 June 2023.

[22] Microsoft, "Visual studio code - code editing. redefined," 2024, accessed: 2025-06-04. [Online]. Available: https://code.visualstudio.com/

[23] E. Matthes, *Python Crash Course*, 3rd ed. No Starch Press, 2023.

[24] Google AI Blog, "A neural network for machine translation, at production scale," 2016, https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html.

[25] DeepL Translator, "Deepl faqs and whitepaper," n.d., https://www.deepl.com/en/pro-whitepaper.

[26] A. Fan, S. Bhosale, H. Schwenk, Z. Ma, A. El-Kishky, S. Goyal, S. Edunov, V. Chaudhary, G. Wenzek, N. Goyal *et al.*, "Beyond english-centric multilingual machine translation," *arXiv preprint arXiv:2010.11125*, 2020.

[27] J. Tiedemann and S. Thottingal, "Opus-mt – building open translation services for the world," in *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation (EAMT)*, 2020.

[28] L. Xue, N. Constant, A. Roberts, M. Kale, R. Al-Rfou, A. Siddhant, A. Barua, and C. Raffel, "mt5: A massively multilingual pre-trained text-to-text transformer," in *Proceedings of NAACL-HLT 2021*, 2021.

[29] Your Custom Model, "Final year project: Speech-driven translation system using transformer architecture," University Department of Computer Science, 2025, model training logs and results reported in this project.

Table of Contents

List of Tables