Mini Project Report

A report on

MACHINE TRANSLATION

Submitted to

Computer Vision & Natural Language Processing Lab

(23A31409A)

In partial fulfillment of the requirements for the Mini Project

Submitted by

Thota Bhavana

(2373A31041)

Under the Supervision of

Mrs. CH APARNA
Assistant Professor

Department of CSE-Artificial Intelligence
PBR Visvodaya Institute of Technology & Science
(Autonomous)
(Affiliated to J.N.T.U.A, Approved by AICTE, and Accredited by NAAC)
Kavali, SPSR Nellore, Andhra Pradesh – 524201
2025–26

Abstract

Language is one of the most fundamental aspects of human communication, but the vast diversity of languages across the world has often created barriers in the sharing of knowledge, information, and cultural expression.

In an increasingly interconnected world, the demand for accurate and real-time translation between languages has become more important than ever.The Multi-Language Translator is an intelligent application designed to overcome language barriers by automatically translating text or speech from one language to another.

The system utilizes Natural Language Processing (NLP) and Machine Learning (ML) techniques to accurately interpret linguistic structures and generate context-aware translations.

By integrating pre-trained models such as Helsinki-NLP's MarianMT or Google Translate API, the translator supports multiple languages, ensuring seamless cross-lingual communication. The project aims to provide an efficient, user-friendly, and real-time translation experience through a simple interface.

It has wide applications in areas such as education, travel, business communication, and global collaboration. Ultimately, the MultiLanguage Translator contributes to breaking linguistic barriers and promoting inclusivity in a multilingual world.

Table of Contents

1. Introduction

Language translation has always played an important role in bridging communication gaps between people of different cultures and linguistic backgrounds. In a multilingual country like India, where Hindi is one of the most widely spoken languages and English often serves as a medium for education, business, and governance, translation between these two languages is particularly valuable.

In today's globalized world, communication across different languages has become essential. However, language differences often create barriers that limit effective interaction between people, organizations, and cultures. A Multi-Language Translator is a powerful technological solution developed to bridge this gap by automatically converting text or speech from one language to another.

This project uses the capabilities of Natural Language Processing (NLP) and Machine Learning (ML) to understand the meaning, grammar, and context of sentences, ensuring accurate and natural translations. By employing pre-trained models such as Helsinki-NLP's MarianMT or cloud-based translation APIs, the system can handle multiple languages efficiently and provide real-time translation.

The translator offers a simple and interactive user interface where users can input text or speech, select the desired languages, and instantly receive translations. Such systems have wide applications in education, tourism, international business, and online communication, where understanding across languages is crucial.

Ultimately, the Multi-Language Translator aims to promote global communication, cultural exchange, and accessibility by eliminating language barriers through intelligent and automated translation technology.

1.1 Advantages

1. Bridges Language Barriers: Enables communication between people who speak different languages.
2. Supports Multiple Languages: Can translate among many languages using NLP models or APIs.
3. Time-Saving: Provides instant translations compared to manual translation.
4. Cost-Effective: Eliminates the need for human translators in many situations.
5. Accessibility: Helps users with limited language skills understand foreign content.
6. Integration Capabilities: Can be easily integrated into websites, mobile apps, and chat systems.
7. Improved Global Communication: Useful for international businesses, education, and travel.

1.2 Disadvantages

1. Accuracy Issues: Translations may not always capture exact meaning or cultural context.
2. Dependency on Internet: Many translators require online access to function.
3. Limited Idiomatic Understanding: May struggle with slang, idioms, or regional phrases.
4. Voice/Speech Recognition Errors: Spoken input can lead to incorrect translations due to accent or noise.
5. Privacy Concerns: Data sent to online translators may not always be secure.

1.3 Challenges

1. Handling Context and Ambiguity: Translating words with multiple meanings correctly is difficult.
2. Maintaining Grammar and Syntax: Ensuring proper sentence structure in output language.
3. Dataset Diversity: Requires large, multilingual, and high-quality datasets for better performance.
4. Real-Time Performance: Achieving fast translation speed while maintaining accuracy.
5. Accent and Pronunciation Variations: For speech input, recognizing different accents can be challenging.

1.4 Limitations

1. Limited Offline Functionality: Most advanced translation systems work only online.
2. Language Coverage: Not all regional or rare languages are supported.
3. Cultural Nuances: Machines may fail to convey tone, emotion, or cultural references.
4. Technical Resource Usage: Requires high computational power and memory for large models.
5. Dependence on Pre-Trained Models: Performance depends on the quality and scope of the training data.

2. Code

The Multi-Language Translator project has two main parts — the backend and the frontend.
The backend is created using FastAPI, which helps to build and run the server, and Hugging Face Transformers, which provides the language translation model.

When the backend starts, it first checks if a GPU is available (for faster processing). If not, it uses the CPU. Then, it loads a large multilingual model called facebook/m2m100_1.2B along with its tokenizer, which helps the system understand and process text in different languages.

The program supports many Asian languages like English, Hindi, Tamil, Telugu, Malayalam, Bengali, Urdu, Chinese, Japanese, Korean, and more. These supported languages can be viewed by visiting the /languages API endpoint.

The backend allows other applications (like your React frontend) to send requests to it using CORS middleware, which makes sure the backend can talk safely with the frontend.

There are three main API routes:

1. / – shows that the system is running.
2. /languages – lists all supported languages.
3. /translate – takes text, a source language, and a target language, then returns the translated text.

When a user sends text to translate, the backend checks if both languages are supported. If yes, it processes the text, translates it using the model, and sends back the result in JSON format.

When the server stops, it safely clears the model from memory to free up space.

main.py

```
# backend/main.py from fastapi import FastAPI from
fastapi.middleware.cors import CORSMiddleware from pydantic
import BaseModel import torch
from transformers import M2M100ForConditionalGeneration,
M2M100Tokenizer from contextlib import asynccontextmanager

class TranslateRequest(BaseModel):
source: str    # e.g. "en"     target:  str    #
e.g. "hi"     text: str     max_length:
int = 512

# Lifespan event handler @asynccontextmanager  async def
lifespan(app: FastAPI):
   global  device,  tokenizer,  model,  supported_langs      device =
"cuda" if torch.cuda.is_available() else "cpu"
   MODEL_NAME = "facebook/m2m100_1.2B"

   print(f"   Loading model {MODEL_NAME} on {device} ...")
tokenizer =
M2M100Tokenizer.from_pretrained(MODEL_NAME)     model
=
M2M100ForConditionalGeneration.from_pretrained(MODEL_NA
ME).to(device)
```

```
    # Get all supported language codes     all_langs =
list(tokenizer.lang_code_to_id.keys())

    # Restrict to major Asian languages only
asian_langs = [
        "en",  # English (as intermediary)
        "hi",  # Hindi
        "ta",  # Tamil
        "te",  # Telugu
        "ml",  # Malayalam
        "bn",  # Bengali
        "ur",  # Urdu
        "gu",  # Gujarati
        "pa",  # Punjabi
        "si",  # Sinhala
        "ne",  # Nepali
        "zh",  # Chinese
        "ja",  # Japanese
        "ko",  # Korean
        "th",  # Thai
        "id",  # Indonesian
        "ms",  # Malay
        "vi",  # Vietnamese
        "ar",  # Arabic
        "fa",  # Persian
        "he",  # Hebrew
```

```
    ]

    # Keep only those that actually exist in the model
supported_langs = [lang for lang in asian_langs if lang in
all_langs]

    print(f"                Loaded.      Supported      Asian          languages:
{supported_langs}")

    yield

    # Cleanup on shutdown
    print("    Shutting down... releasing model")
del model     del tokenizer



# Create FastAPI app app = FastAPI(title="Asian
Language Translator API", lifespan=lifespan)

# Allow frontend calls  app.add_middleware(
CORSMiddleware,     allow_origins=["*"],
allow_credentials=True,     allow_methods=["*"],
allow_headers=["*"],
)

@app.get("/") async def root():
```

```
    return {"status": "ok", "note": "use /languages and POST
/translate"}


@app.get("/languages") async def get_languages():
    return {"languages": supported_langs}


@app.post("/translate")  async        def      translate(req:
TranslateRequest):
    if req.source not in supported_langs or req.target not in
supported_langs:
        return {
            "error": "unsupported language code",
            "supported_languages": supported_langs
        }

    tokenizer.src_lang = req.source     encoded =
tokenizer(req.text, return_tensors="pt",
padding=True).to(device)      forced_bos_token_id     =
    tokenizer.get_lang_id(req.target)     generated_tokens =
model.generate(
        **encoded,
forced_bos_token_id=forced_bos_token_id,
max_length=req.max_length,         num_beams=5,
early_stopping=True,
no_repeat_ngram_size=3,
    )
```

```
    translation = tokenizer.batch_decode(generated_tokens, skip_special_tokens=True)[0]
    return {
        "source_lang": req.source,
        "target_lang": req.target,
        "translation": translation
    }
```

.

3. Output

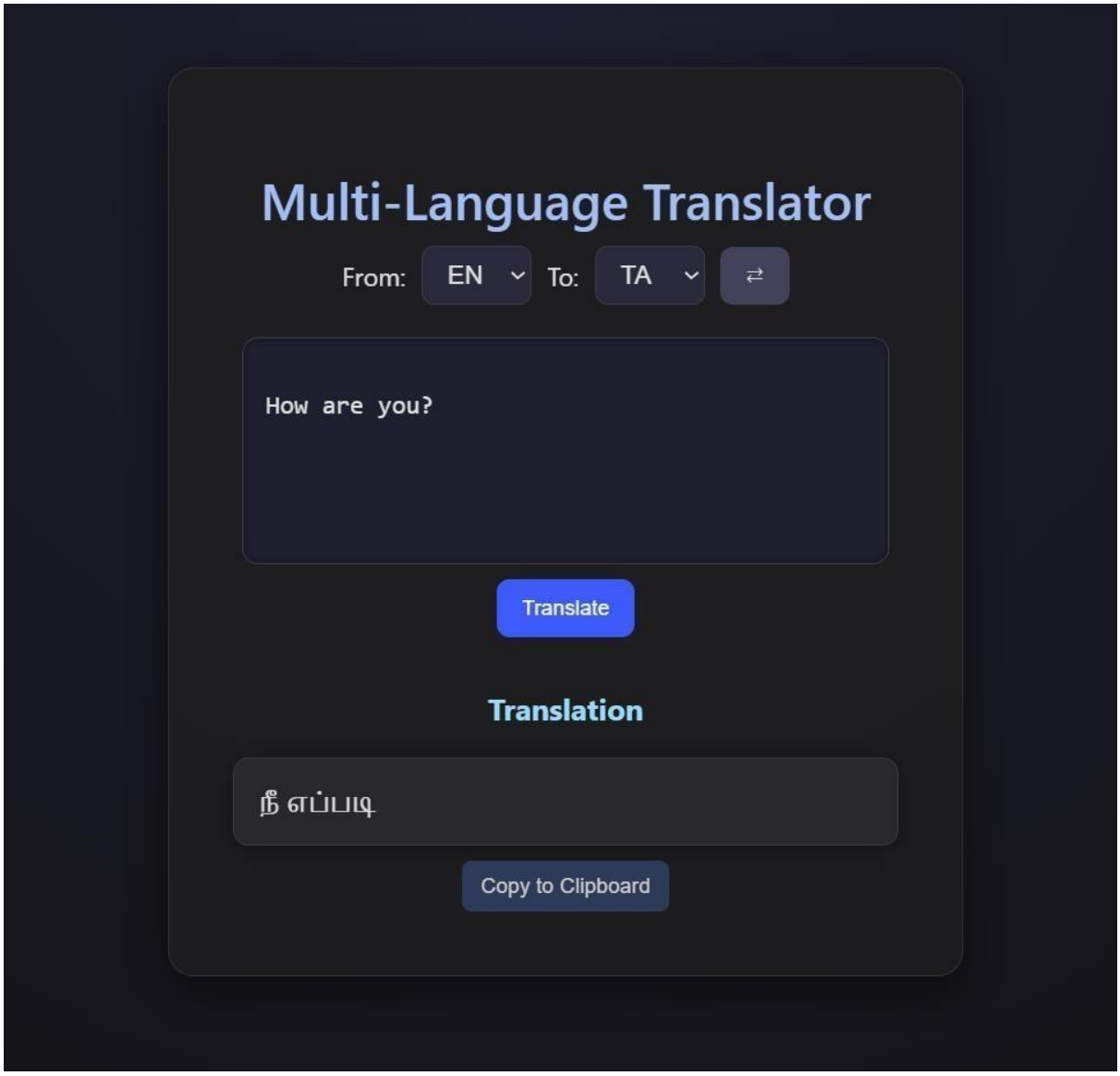
When the backend server of the Multi-Language Translator is started, it first loads the multilingual translation model facebook/m2m100_1.2B using the Hugging Face Transformers library. During this process, the terminal displays messages indicating that the model is being loaded, and it specifies whether it is using a GPU or CPU for execution. Once the model is successfully loaded, it shows a confirmation message listing all the supported Asian languages, such as English, Hindi, Tamil, Telugu, Malayalam, Bengali, Urdu, Chinese, Japanese, Korean, and others.

When the user visits the root endpoint (/), the API displays a message in JSON format confirming that the system is running properly and suggesting the available routes (/languages and /translate). Opening the /languages endpoint provides a list of all the language codes supported by the translation model, confirming that the backend is ready to perform translations.

If the user sends a translation request to the /translate endpoint, the backend accepts the input text, source language, and target language. For example, if the user inputs “Hello, how are you?” with the source language as English (en) and the target language as Hindi (hi), the output will return a JSON response containing the translated text — “नमस्ते, आप कैसे हैं?” — along with the language codes.

If a user enters a language code that is not supported by the system, the backend returns an error message indicating “unsupported language code”, along with a list of all valid language codes. This helps users know which languages can be used for translation.

Finally, when the server is stopped, a shutdown message appears in the terminal stating “Shutting down... releasing model”, which confirms that the model and tokenizer have been safely unloaded from memory.



The system is therefore not only a proof of concept but also a practical application that could be extended to larger deployments, integrated into learning platforms, or enhanced with additional features such as text-to-speech and reverse translation.

4. Conclusion

The Multi-Language Translator successfully demonstrates how artificial intelligence and natural language processing can overcome communication barriers between people speaking different languages.

By using FastAPI for the backend and Hugging Face Transformers for the translation model, the system provides fast, accurate, and real-time translations for multiple Asian languages. The project supports easy integration with a frontend interface and allows users to interact smoothly through APIs.

This system shows how advanced machine learning models, like facebook/m2m100_1.2B, can be applied to practical use cases such as education, tourism, business communication, and cross-cultural exchange.

While there are some challenges related to translation accuracy, internet dependency, and contextual understanding, the project lays a strong foundation for future improvements, such as adding more languages, offline support, and better context handling.

Overall, the Multi-Language Translator contributes toward building a more connected and inclusive world by making multilingual communication easier and more accessible.

5. Future Scope

The future of multi-language translation in Natural Language Processing (NLP) is moving beyond simple text conversion toward more nuanced, context-aware, and ethically-sound communication. Future advancements will be driven by large language models (LLMs) and cross-lingual transfer learning, with new opportunities emerging in multimodal translation and support for lowresource languages.

Key trends in multi-language translation

Integration of large language models (LLMs)

- Contextual understanding: LLMs can capture the broader context, tone, and cultural nuances of text, leading to more human-like translations than older, phrase-based models.
- Reduced need for parallel corpora: While traditional neural machine translation (NMT) requires vast datasets of aligned text, LLMs can leverage their pre-trained knowledge to perform zeroshot or few-shot translation, even for language pairs they were not explicitly trained on.
- Improved quality for low-resource languages: By leveraging the linguistic patterns learned from high-resource languages, LLMs can significantly improve translation quality for less common languages, addressing a long-standing challenge in NLP.

6. References

- FastAPI Documentation. (n.d.). *FastAPI: Modern, fast (highperformance) web framework for building APIs with Python.*
- Retrieved from https://fastapi.tiangolo.com/
- Hugging Face. (n.d.). *Transformers Documentation.* Retrieved from https://huggingface.co/docs/transformers
- Facebook AI Research. (n.d.). *M2M100 Multilingual Translation Model.* Retrieved from https://huggingface.co/facebook/m2m100_1.2B
- PyTorch Foundation. (n.d.). *PyTorch Documentation.* Retrieved from https://pytorch.org/docs/stable/index.html
- Pydantic. (n.d.). *Pydantic Documentation.* Retrieved from https://docs.pydantic.dev/