

Automatic Translator Based on Transformers

Ainhoa Lucia Pérez González

Jose Alejandro Lopez Garcés

1. description of the work.

2. Introduction

2.1 Objective

2.2 Scope

2.3 Methodology

I. Theoretical Part

II. Practical part

Tools and Dependencies

Virtual Environment Setup

Main Code (`app.py`)

Code Explanation

Tests Performed

Limitations

Future Improvements

Web Interface Integration

5. Summary

6. Bibliography

1. Abstract

This project presents the development of a web tool for translating text between multiple languages using neural machine translation models based on Transformers from the Hugging Face library. A Python backend with Flask is implemented that uses specific models for each language pair, and a simple frontend enables user interaction. The problem of unsupported direct language pairs (such as French to German) is addressed by implementing a pivot step through English. Dependencies, tests, limitations, and possible future improvements are analyzed.

2. Introduction

2.1 Objective

The main objective is to create a functional automatic translator that allows users to translate text between different languages using pretrained Transformer models, with an extensible system able to handle unsupported language pairs.

2.2 Scope

The project covers plain text translation among common languages (e.g., English, Spanish, French, German) using Hugging Face models. The system includes a REST API and a basic web interface to facilitate use. Document translation, audio support, or pronunciation features are not included, yet.

2.3 Methodology

The Hugging Face transformers library is used to load neural machine translation models based on the Transformer architecture. The backend is implemented in Python with Flask to receive text and return translations. The frontend is simple HTML/JavaScript that calls the backend via AJAX. For unsupported language pairs (such as French to German), an intermediate step translates first to English and then to the target language, chaining translations.

I. Theoretical Part

Transformers are deep learning models based on attention mechanisms that have revolutionized machine translation. Unlike traditional sequence-to-sequence models, they process full input sequences and capture long-range dependencies effectively.

Hugging Face provides pretrained Helsinki-NLP (opus-mt) models supporting multilingual translation for dozens of language pairs. These models are easily accessed through the transformers API, which includes a translation pipeline.

II. Practical Part

Tools and Dependencies

Python 3.8+: main backend language.

Flask: microframework to build REST API.

transformers: Hugging Face library to load translation models.

torch: backend for models (PyTorch).

Installation command:

```
pip install flask transformers torch
```

Virtual Environment Setup

To maintain project dependencies isolated and avoid conflicts with other Python projects or system packages, a virtual environment was created and used for this project. This ensures a reproducible and clean development environment.

The steps followed to create and activate the virtual environment are:

```
# Create a virtual environment named 'venv' in the project directory
python3 -m venv venv
```

```
# Activate the virtual environment
source venv/bin/activate
```

Using a virtual environment avoids installing packages globally on the system and makes it easy to manage the specific versions required by the project.

To deactivate the environment when finished, simply run:

```
deactivate
```

Main Code (app.py)

```
from flask import Flask, request, jsonify
from transformers import pipeline

app = Flask(__name__)

# Dictionary with directly supported language pairs by Helsinki-NLP models
SUPPORTED_MODELS = {
    ("en", "es"): "Helsinki-NLP/opus-mt-en-es",
    ("es", "en"): "Helsinki-NLP/opus-mt-es-en",
    ("en", "fr"): "Helsinki-NLP/opus-mt-en-fr",
    ("fr", "en"): "Helsinki-NLP/opus-mt-fr-en",
    ("en", "de"): "Helsinki-NLP/opus-mt-en-de",
    ("de", "en"): "Helsinki-NLP/opus-mt-de-en",
    # Add more pairs as needed
}

# Cache to store loaded models and avoid reloading
model_cache = {}

def load_translator(src_lang, dest_lang):
    """
    Load or retrieve from cache the translation pipeline for given language pair.
    Returns None if no model is available.
    """
    key = (src_lang, dest_lang)
    if key in model_cache:
        return model_cache[key]

    model_name = SUPPORTED_MODELS.get(key)
    if not model_name:
        return None
    translator = pipeline("translation", model=model_name)
    model_cache[key] = translator
    return translator

def translate_text(text, src_lang, dest_lang):
    """
    Translate text from src_lang to dest_lang.
    If direct translation model is not available, use English as pivot:
    src_lang -> English -> dest_lang.
    """
    # Try direct translation first
    translator = load_translator(src_lang, dest_lang)
    if translator:
        result = translator(text, max_length=512)
        return result[0]['translation_text']

    # If no direct model, try pivot through English
    if src_lang != "en" and dest_lang != "en":
        to_english = load_translator(src_lang, "en")
        if not to_english:
            raise ValueError(f"No model for {src_lang} to English")
        intermediate = to_english(text, max_length=512)[0]['translation_text']

        from_english = load_translator("en", dest_lang)
```

```

        if not from_english:
            raise ValueError(f"No model for English to {dest_lang}")
        final_translation = from_english(intermediate, max_length=512)[0]
    ['translation_text']
    return final_translation

    raise ValueError(f"Translation from {src_lang} to {dest_lang} not supported")

@app.route('/translate', methods=['POST'])
def translate_api():
    text = request.form.get('text')
    src_lang = request.form.get('src_lang', 'en')
    dest_lang = request.form.get('dest_lang', 'es')
    if not text:
        return jsonify({"error": "No text provided"}), 400

    try:
        translation = translate_text(text, src_lang, dest_lang)
        return jsonify({"translation": translation})
    except Exception as e:
        return jsonify({"error": str(e)}), 500

if __name__ == "__main__":
    app.run(debug=True)

```

Code Explanation

The Flask application exposes a POST endpoint /translate which receives:

text: The text to translate

src_lang: Source language code (default 'en')

dest_lang: Target language code (default 'es')

The core translation functionality relies on the Hugging Face transformers library and its pipeline abstraction.

Key Hugging Face parts:

Pipeline API:

```
translator = pipeline("translation", model=model_name)
```

The pipeline function creates a ready-to-use inference pipeline for a specific task — in this case, “translation”.

model=model_name loads the pretrained model identified by model_name from Hugging Face’s model hub (or cache).

This pipeline encapsulates all necessary steps internally: tokenization, encoding, model forward pass, decoding, and detokenization.

Model Selection and Caching:

```
def load_translator(src_lang, dest_lang):
    key = (src_lang, dest_lang)
    if key in model_cache:
        return model_cache[key]

    model_name = SUPPORTED_MODELS.get(key)
    if not model_name:
        return None

    translator = pipeline("translation", model=model_name)
    model_cache[key] = translator
    return translator
```

The dictionary `SUPPORTED_MODELS` maps language pairs to Helsinki-NLP pretrained model names (e.g., “Helsinki-NLP/opus-mt-en-es”).

`load_translator` loads the pipeline once per language pair and caches it in `model_cache` to avoid expensive reloads on subsequent calls, significantly improving performance.

If no model exists for the exact language pair, it returns `None`.

Translation Logic:

```
def translate_text(text, src_lang, dest_lang):
    translator = load_translator(src_lang, dest_lang)
    if translator:
        result = translator(text, max_length=512)
        return result[0]['translation_text']

    if src_lang != "en" and dest_lang != "en":
        to_english = load_translator(src_lang, "en")
        intermediate = to_english(text, max_length=512)[0]
        ['translation_text']

        from_english = load_translator("en", dest_lang)
        final_translation = from_english(intermediate,
max_length=512)[0]['translation_text']
        return final_translation
```

```
raise ValueError(...)
```

First, it attempts a direct translation using a model for (src_lang → dest_lang).

If the direct model is unavailable, but both languages are different from English, it implements a pivot translation:

- Translate from source language to English.
- Then translate from English to the target language.

This solves the problem of missing direct models between many language pairs but at the cost of increased latency and potential minor quality loss.

max_length=512 limits the maximum token length to prevent exceeding model capacity.

Flask API Endpoint:

```
@app.route('/translate', methods=['POST'])
def translate_api():
    ...
    translation = translate_text(text, src_lang, dest_lang)
    return jsonify({"translation": translation})
```

Exposes a POST endpoint /translate that expects form data with keys text, src_lang, and dest_lang.

Calls the translation function and returns the translated text in JSON format.

Handles errors and returns appropriate HTTP status codes and messages.

Summary of Hugging Face usage:

The Hugging Face pipeline API abstracts all the complex steps of tokenizing input text, running it through the Transformer model, and decoding the output tokens back into human-readable text.

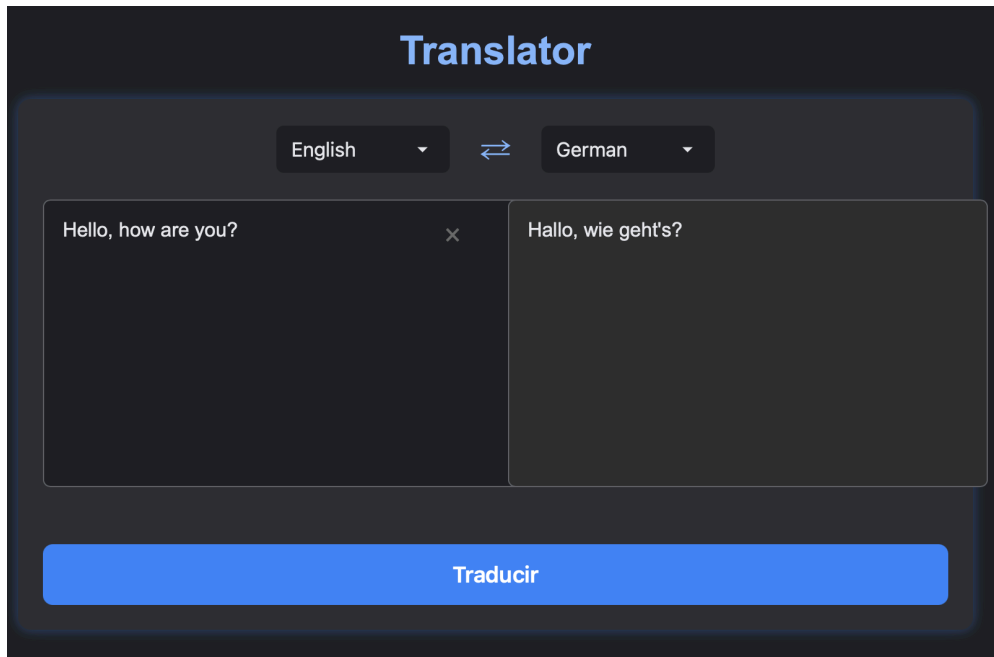
By selecting different pretrained models from the Helsinki-NLP repository, the system supports various language pairs without retraining.

Caching loaded models optimizes performance by avoiding repeated model loading.

The pivot translation technique leverages English as an intermediary language to expand the coverage of translations beyond directly supported pairs

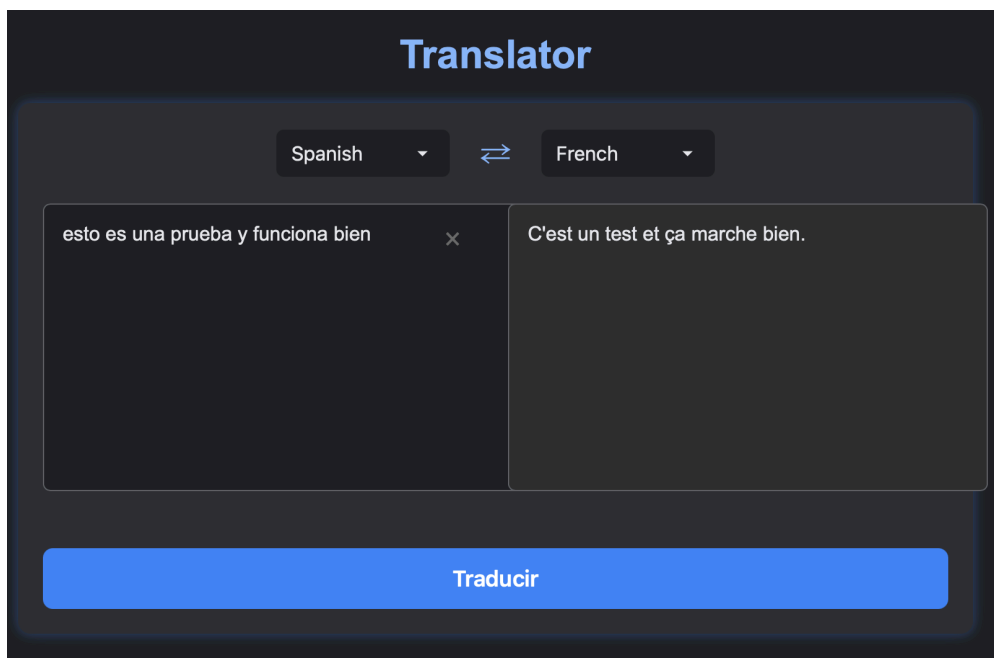
Tests Performed

Direct translation:



English to German : "Hello, how are you?" → "Hallo, wie geht's?"

Pivot translation:



French to German: "esto es una prueba y funciona bien"

French to English: "This is a test and it works well."

English to German: "C'est un test et ça marche bien."

Translations were accurate and coherent within expected limits of automatic translation.

Limitations

Pivot translation increases response time (two model calls).

Models limited to pretrained Helsinki-NLP pairs.

Text length capped to 512 tokens.

No pronunciation or document support.

PyTorch dependency may slow performance on limited hardware.

Future Improvements

Speed optimization:

Model quantization, ONNX export.

Persistent server with loaded models.

Language coverage:

Add more Helsinki-NLP or larger multilingual models.

PDF reader integration:

Extract and translate text from uploaded PDFs.

User interface enhancements:

File upload support, history, saved translations.

Pronunciation support:

Use external APIs for audio or phonetics.

Batch and multi-language translation:

Translate one text to multiple target languages.

Long text handling:

Chunking and merging translated segments.

Web Interface Integration

The project includes a simple frontend allowing users to choose source and target languages, input text, and get translations in real time. Features:

Language selectors with swap button.

Source and target text areas.

Debounced translation to limit API calls.

Clear text functionality.

5. Summary

A functional translation tool using Hugging Face Transformer models was created, supporting multiple language pairs with pivot translation via English when needed. The project covers both the theoretical basis of Transformer-based translation and a practical implementation of an API and web interface. The system is scalable and can be enhanced in many directions to improve speed, functionality, and user experience.

6. Bibliography

Hugging Face Transformers Documentation. <https://huggingface.co/docs/transformers/en/tasks/translation>
Vaswani et al., "Attention is All You Need," 2017.
Wikipedia: Transformer (machine learning model). [https://en.wikipedia.org/wiki/Transformer_\(machine_learning_model\)](https://en.wikipedia.org/wiki/Transformer_(machine_learning_model))
Helsinki-NLP Opus-MT Models. <https://huggingface.co/Helsinki-NLP>
<https://github.com/huggingface/transformers>