Project Report: Multilingual Translator

Architecture, Setup, and Usage Guide

Technical Documentation

Contents

Chapter 1

Project Overview & Architecture

1.1 Introduction

In today's interconnected world, language barriers remain a significant challenge. The **Multilingual Translator & Summariser** is an AI-powered application designed to bridge this gap. It allows users to input text in *any* language, automatically translates it into English, and then provides a concise summary of the content.

1.2 Problem & Solution

- **The Problem:** Information Overload. Valuable documents (news, research, user feedback) are often available only in local languages and are too long to digest quickly.
- **The Solution:** A single-click pipeline.
  1. **Translate:** Convert local language 'X' to English using `NLLB-200`.
  2. **Summarize:** Condense the English text using `BART-Large`.
  3. **Present:** Display both outputs in a clean, reactive Web UI.

1.3 System Design

The system follows a modular architecture, separating the core AI logic from the user interface.

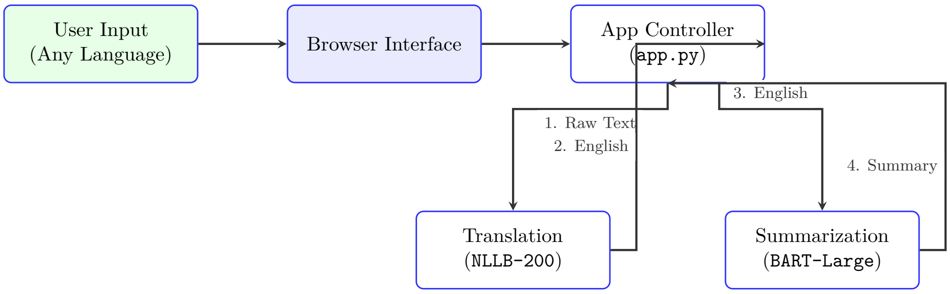

Figure 1.1: Data Flow Architecture

1.4 Model Decisions

Translation Model: We selected ‘facebook/nllb-200-distilled-600M‘.

- **Pros:** Supports 200+ languages, high accuracy, reasonable size (1.2GB).
- **Why Distilled?** The full model is 54GB, which is impossible to run on consumer hardware. The distilled version offers 90% of the performance at 2% of the size.

Summarization Model: We selected ‘facebook/bart-large-cnn‘.

- **Pros:** State-of-the-art for abstractive summarization on news articles.
- **Why CNN?** It was fine-tuned on the CNN/DailyMail dataset, making it excellent for understanding journalistic and factual content.

Chapter 2

Installation & Setup

This guide assumes you are running on Windows, but the commands are similar for Mac/Linux. We will cover the installation of Python, the creation of a virtual environment, and the installation of the machine learning dependencies.

2.1 Prerequisites

Before diving in, ensure you have the following installed on your system:

1. **Python 3.8+**: Hugging Face libraries require a modern Python version. Verification: `python --version`.
2. **Git**: Standard for version control, useful for cloning the repository.
3. **Internet Connection**: To download model weights. The models (`NLLB-200` and `BART-Large`) are approx 2-3 GB combined.
4. **Hardware:** A GPU (NVIDIA GTX 1650 or better) is recommended for fast inference. If you use a CPU, expect delays of 10-15 seconds per request.

2.2 Step-by-Step Installation

2.2.1 1. Clone the Repository

Open your terminal (PowerShell or Command Prompt) and navigate to your workspace.

```
# Clone the project code
git clone https://github.com/your-repo/multilingual-translator.git

# Enter the directory
```

```
cd multilingual-translator
```

2.2.2 2. Create a Virtual Environment

It is ”best practice” to isolate dependencies so they don’t conflict with other projects.

```
# Create the environment named 'venv'
python -m venv venv

# Activate it (Windows)
# Your prompt should change to start with (venv)
venv\Scripts\activate

# Activate it (Linux/Mac)
source venv/bin/activate
```

2.2.3 3. Install Dependencies

We rely on `torch`, `transformers`, and `gradio`.

```
# Upgrade pip first to avoid errors
pip install --upgrade pip

# Install dependencies from the file
pip install -r requirements.txt
```

2.3 GPU Configuration (Optional but Recommended)

Standard `pip install torch` usually installs the CPU version. To enable GPU support:

1. Check your CUDA version (open NVIDIA Control Panel or run `nvidia-smi`).
2. Go to https://pytorch.org/get-started/locally/.
3. Copy the command for your version. For example (CUDA 11.8):

```
pip install torch torchvision torchaudio --index-url
    https://download.pytorch.org/whl/cu118
```

2.4 Directory Structure Verification

After installation, your project folder should look like this:

```
multilingual-translator/
|-- venv/                # Virtual Environment (do not touch)
|-- src/
|    |-- __init__.py
|    |-- translation.py   # Translation Logic
|    |-- summarization.py # Summarization Logic
|-- app.py               # Main Entry Point
|-- requirements.txt     # Dependency List
|-- README.md            # Quick Start Guide
```

Warning

Do not commit the `venv/` folder to Git! It is specific to your machine and very large. Use a `.gitignore` file to exclude it.

2.5 Troubleshooting Common Issues

- **Error: ”Module not found”**: Ensure you activated the virtual environment before running the app.

- **Error: ”Torch not compiled with CUDA enabled”**: You installed the CPU version of Torch. Uninstall it (`pip uninstall torch`) and reinstall the GPU version using the command above.

- **Slow Performance**: If the translation takes ¿30 seconds, you are likely running on CPU. This is expected for large models like NLLB-200.

- **Download Fails**: These models are hosted on Hugging Face Hub. If you have a firewall, you might need to use a proxy or check your internet connection.

Chapter 3

Usage Manual

This chapter guides you through running the application and understanding the user interface.

3.1 Running the Application

With your environment active, run the following command in your terminal:

```
python app.py
```

3.1.1 Startup Logs

You should see output similar to the following. Note the model loading times:

```
Initializing AI Services...
Loading Translation Model: facebook/nllb-200-distilled-600M...
(This may take 10-20 seconds on first run to download 1.2GB)
Loading Summarization Model: facebook/bart-large-cnn...
(This may take 10-15 seconds to download 1.6GB)
Services loaded successfully.
Running on local URL:  http://127.0.0.1:7860
```

Open your browser (Chrome/Edge/Firefox) and navigate to `http://127.0.0.1:7860`.

3.2 Interface Overview

The User Interface (UI) is built with Gradio and consists of three main sections:

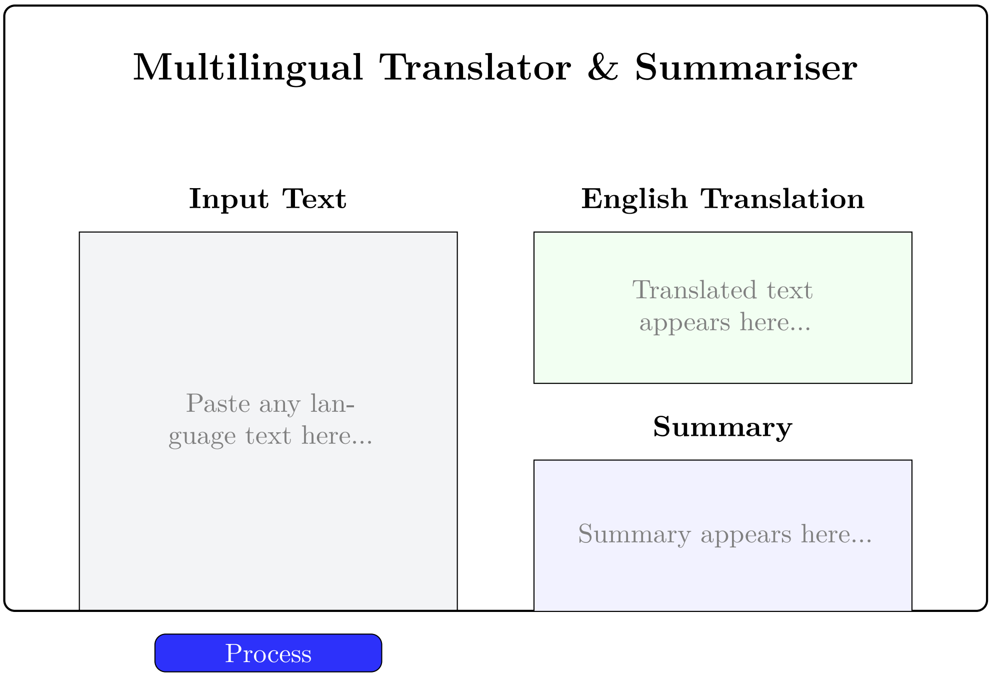
Multilingual Translator & Summariser
Input Text
Paste any language text here...
English Translation
Translated text appears here...
Summary
Summary appears here...
Process

Figure 3.1: UI Layout Mockup

3.3 Step-by-Step Workflow

1. **Input Field:** Copy and paste any text into the large box on the left. The model handles over 200 languages (Spanish, Hindi, French, Japanese, etc.).

2. **Process Button:** Click the "Process" button. The button will show a loading spinner while the GPU processes the request.

3. **Outputs:**

    - **Top Right (Translation):** The system first translates the input into English. This preserves the full detail of the original text.
    - **Bottom Right (Summary):** The system then reads the English translation and creates a 2-3 sentence abstractive summary.

3.4 Example Scenarios

3.4.1 Scenario 1: News Article (Hindi)

- **Input:** "Bharat ek vishaal desh hai..." (A long paragraph about India's geography).

- **Action:** User clicks Process.
- **Translation:** ”India is a vast country...”
- **Summary:** ”India is geographically diverse with ancient culture.”

3.4.2 Scenario 2: Tech Review (French)

- **Input:** ”Ce nouvel ordinateur portable est incroyablement rapide mais la batterie...”
- **Translation:** ”This new laptop is incredibly fast but the battery...”
- **Summary:** ”The laptop has excellent performance but poor battery life.”

Warning

The first request might be slow (10-20 seconds) as the computer ”warms up” the models (JIT Compilation). Subsequent requests will be much faster.

3.5 Advanced Customization

Want to change the models? You can modify ‘src/translation.py‘:

3.5.1 Using a Smaller Model

If you are on an older laptop, switch to the 600M distilled version or even the 1.3B version if you have more RAM.

```
# In src/translation.py
self.model_name = "facebook/nllb-200-distilled-600M" # Fast
# self.model_name = "facebook/nllb-200-3.3B"          # Very Slow, More Accurate
```

3.5.2 Changing the Summary Style

You can make the summary bullet-pointed by editing ‘src/summarization.py‘:

```
# In src/summarization.py
def summarize(self, text):
    prompt = f"Summarize this in bullet points: {text}"
    # ... rest of code
```

Chapter 4

Code Implementation Details

This chapter provides a deep dive into the implementation. We use object-oriented programming to keep the codebase modular and testable.

4.1 Translation Service

File: `src/translation.py`

The ‘TranslationService‘ class encapsulates the complexity of the NLLB model.

```python
class TranslationService:
    def __init__(self, model_name="facebook/nllb-200-distilled-600M"):
        # We check for GPU availability automatically
        self.device = 0 if torch.cuda.is_available() else -1

        print(f"Loading Translation Model: {model_name}...")
        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
        self.model = AutoModelForSeq2SeqLM.from_pretrained(model_name)

        if self.device == 0:
            self.model = self.model.to("cuda")
```

Listing 4.1: Initializing NLLB

4.1.1 Handling Multilingual Inputs

The core challenge with NLLB is that it requires a ”Forced BOS Token” (Beginning of Sentence) to know which language to translate *into*.

```python
    def translate(self, text, target_lang="eng_Latn"):
        if not text: return ""

        # 1. Tokenize Input
        inputs = self.tokenizer(text, return_tensors="pt", padding=True)
        if self.device == 0:
            inputs = {k: v.to("cuda") for k, v in inputs.items()}

        # 2. Force Target Language Token
        # "eng_Latn" is the NLLB code for English
        forced_bos = self.tokenizer.convert_tokens_to_ids(target_lang)

        # 3. Generate
        with torch.no_grad():
            tokens = self.model.generate(
                **inputs,
                forced_bos_token_id=forced_bos,
                max_length=512
            )

        # 4. Decode
        return self.tokenizer.batch_decode(tokens, skip_special_tokens=True)[0]
```

Listing 4.2: Forcing English Output

4.2 Summarization Service

File: `src/summarization.py`

The summarizer is simpler because we leverage the Hugging Face ‘pipeline‘ abstraction.

```python
class SummarizationService:
    def __init__(self):
        # Pipelines handle tokenization, device placement, and decoding internally
        self.summarizer = pipeline("summarization", model="facebook/bart-large-cnn")

    def summarize(self, text):
        # We set tight constraints on length to ensure concise outputs
        result = self.summarizer(
            text,
            max_length=130,  # Max summary length
            min_length=30,    # Min summary length
            do_sample=False   # Deterministic output
        )
        return result[0]['summary_text']
```

4.3 The Application Controller

File: `app.py`

Finally, we tie everything together.

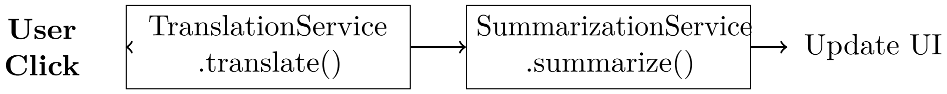
User Click
TranslationService .translate()
SummarizationService .summarize()
Update UI

Figure 4.1: Execution Flow

The `app.py` script initializes these services once (globally) so they don't reload on every request, which would be very slow.