AMRITA SCHOOL OF COMPUTING,

AMARAVATI

Department of Computer Science and Engineering

Subject: Natural Language Processing
Title: Language Translation

Submitted by

K. Aishwarya, V. Venkatasai , P.Kusumanjali

AV.EN.U4AIE22122, AV.EN.U4AIE22150,

AV.EN.U4AIE22154

B. Tech (AIE) SIXTH SEMESTER

Under the Guidance of

Mrs. Riyanka Manna

Assistant Professor (Sl. Gr.)

Department of Computer Science and Engineering

Table of Contents

1.Abstract:

In recent years, machine translation has significantly advanced with the advent of multilingual pre-trained models like mBART. However, translation performance across different language pairs, especially involving low-resource languages, remains a challenge. This project explores multilingual translation using the facebook/mbart-large-50-many-to-many-mmt model, focusing on evaluating translation quality from English to various target languages using BLEU score. A backtranslation approach is used to estimate the fidelity of translations by translating English text to a target language and back to English. The resulting BLEU scores reveal stark contrasts in translation quality across languages, with high-resource languages like Spanish and German outperforming low-resource languages such as Telugu and Nepali. This report presents a detailed analysis of the methodology, experiments, and implications of the results, providing insights into the performance disparities and the potential for improvement in multilingual NLP systems.

2.Introduction:

Language plays a central role in human communication, and the ability to translate between languages is crucial in breaking down linguistic barriers in an increasingly interconnected world. With the rise of globalization and digital content, the demand for accurate and efficient machine translation systems has surged. However, despite major advancements in neural machine translation (NMT), significant performance gaps still exist between high-resource and low-resource languages.

Multilingual models like mBART (Multilingual Bidirectional and Auto-Regressive Transformers) have emerged as a powerful solution to enable translation across numerous language pairs using a single pre-trained model. Unlike traditional models that require separate training for each language pair, mBART leverages shared representations and training objectives to generalize translation capabilities more effectively. This is particularly valuable for underrepresented languages where data availability is scarce.

In this project, we utilize the facebook/mbart-large-50-many-to-many-mmt model to perform translations from English to a variety of target languages and evaluate the quality of these translations using the BLEU (Bilingual Evaluation Understudy) score. To achieve this, we adopt a backtranslation-based evaluation method: each English sentence is translated to a target language and then back

to English. The BLEU score is computed by comparing the original English sentence and the backtranslated version.

The goal of this study is to analyze how well the model performs across different language pairs, identify patterns in BLEU scores, and highlight the disparity in translation quality among high-resource and low-resource languages. This investigation aims to provide insights into the limitations and potential of multilingual NMT systems in handling linguistic diversity effectively.

Our Contributions:

- Developed a robust method to assess translation quality by translating English sentences into various target languages and back to English, enabling fair evaluation using BLEU scores.
- Conducted a comparative analysis of BLEU scores for multiple high- and low-resource languages, highlighting performance disparities and insights into mBART's multilingual capabilities.
- Revealed significantly lower BLEU scores for languages like Telugu and Nepali, emphasizing the need for domain-specific fine-tuning for improved accuracy.
- Created informative bar charts to present BLEU scores for each language pair, facilitating easy interpretation of translation quality differences.
- Structured the project to allow for future enhancements like fine-tuning on custom datasets, zero-shot learning, and integration with real-world applications.

3.Literature Survey:

The table below summarizes key literature relevant to neural machine translation, multilingual models, and evaluation metrics used in this project.

Author(s)	Year	Title	Contribution	Relevance to Our Work
Bahdanau et al.	2015	Neural Machine Translation by Jointly Learning to Align and Translate	Introduced attention mechanism in NMT	Forms the foundation for sequence-to-sequence models used
Vaswani et al.	2017	Attention is All You Need	Proposed Transformer architecture	mBART is built on Transformer architecture

			using self-attention	
Liu et al.	2020	Multilingual Denoising Pre-training for Neural Machine Translation	Introduced mBART, pretrained multilingual model using denoising autoencoding	Core model used in our project
Papineni et al.	2002	BLEU: a Method for Automatic Evaluation of Machine Translation	Proposed BLEU score for evaluating translation quality	BLEU is our primary evaluation metric
Conneau et al.	2020	Unsupervised Cross-lingual Representation Learning at Scale	Advanced cross-lingual pretraining methods like XLM-R	Reinforces the importance of multilingual pretraining

4.Methodology:

This section outlines the step-by-step process followed in the project—from dataset acquisition to translation evaluation. The goal was to fine-tune and evaluate the multilingual capabilities of the mBART-50 model for English-to-multiple-language translation using backtranslation and BLEU scoring.

4.1 Dataset Description

We used the **ai4bharat/IN22-Gen** dataset, a multilingual parallel corpus sourced from the Hugging Face Datasets library. This dataset contains aligned sentence pairs in English and several Indian languages including Hindi, Tamil, Telugu, Marathi, Bengali, and others.

- **Source**: Hugging Face Datasets
- **Languages Supported**: 11+ including English, Hindi, Tamil, Telugu, etc.
- **Format**: JSON-like structure with language-pair-specific fields, e.g., eng_Latn, tam_Taml
- **Split**: Train, Test (we used test split for evaluation)

4.2 Data Preprocessing

To ensure high-quality inputs to the model, the following preprocessing steps were applied:

4.2.1 Tokenization:

Tokenization is a fundamental step in preparing text data for machine translation models. It involves breaking down a sentence into smaller units—typically words or subwords—that can be mapped to numerical representations understood by the model. In the case of transformer-based architectures like mBART-50, tokenization is especially crucial as it directly influences the model's ability to encode, process, and translate text across different languages.

In this project, no manual text cleaning was performed prior to tokenization. Instead, the raw text inputs were directly passed to the **MBart50TokenizerFast**, a tokenizer specifically designed for use with the mBART-50 model. This tokenizer leverages the SentencePiece subword segmentation algorithm, which allows for efficient handling of a diverse vocabulary across multiple languages, including those with rich morphology and non-Latin scripts.

The **MBart50TokenizerFast** tokenizer comes pre-trained on the same data as the mBART-50 model and includes special language tokens that signal the source and target languages during translation. These tokens are essential in guiding the model, especially in many-to-many multilingual settings, where the model must know both the input and output language formats.

As part of the tokenization process, each input sentence was first assigned a source language code to indicate the language of the text being translated. Similarly, a target language code was specified to indicate the language into which the sentence should be translated. The tokenizer then converted the input sentence into a sequence of subword tokens based on its pre-learned vocabulary. This process ensures that even rare or previously unseen words are broken down into meaningful components, allowing the model to handle them effectively.

Furthermore, the tokenizer also automatically handled truncation of overly long sentences and added padding where necessary to maintain uniform input lengths. It generated attention masks that help the model distinguish between actual text and padding tokens, thereby preserving semantic coherence during translation.

By using the MBart50TokenizerFast directly without any intermediate cleaning, the project preserved the model's expected input format. This approach not only aligns with the model’s pretraining setup but also reduces complexity in the preprocessing pipeline. The tokenization step played a crucial role in ensuring that the mBART-50 model received well-structured and semantically intact inputs across all supported language pairs.

4.3 Model:

For this project, we utilized the **facebook/mbart-large-50-many-to-many-mmt** model, commonly referred to as **mBART-50**, which is a multilingual extension of the original BART (Bidirectional and Auto-Regressive Transformers) model. mBART-50 is one of the most powerful transformer-based models developed for many-to-many translation tasks and supports **50 different languages** spanning multiple language families and scripts.

Model Architecture

mBART-50 is based on the **Transformer encoder-decoder architecture**, originally introduced by Vaswani et al. (2017). The encoder takes in the source sentence and transforms it into a sequence of continuous representations, while the decoder uses these representations to generate the translated output, one token at a time.

Each component—the encoder and decoder—is made up of multiple layers of multi-head self-attention, feed-forward neural networks, residual connections, and layer normalization. The key innovation in the mBART family lies in combining the strengths of both **BERT-style bidirectional encoding** and **GPT-style autoregressive decoding**, making it highly effective for sequence-to-sequence tasks such as translation.

Multilingual Pretraining

What sets mBART-50 apart from other translation models is its **multilingual pretraining objective**. It is trained using a **denoising autoencoder** technique on large-scale monolingual corpora. During pretraining, parts of the input text are masked or shuffled, and the model is trained to reconstruct the original sentence. This helps the model learn deep contextual representations and shared linguistic patterns across different languages.

Importantly, mBART-50 uses **language-specific tokens** (e.g., en_XX for English, ta_IN for Tamil) to indicate the source and target languages. This design allows the model to perform **many-to-many translation**—for example, from English to Tamil, German to French, or Hindi to Marathi—without the need for separate models for each language pair.

Why mBART-50 Was Chosen

mBART-50 was selected for this project because of its robust multilingual capabilities and state-of-the-art performance on a wide range of language pairs, including both high-resource and low-resource languages. Unlike traditional models that require parallel datasets for each language pair, mBART-50 is

capable of **zero-shot translation**, where it can translate between language pairs it has not seen together during training, simply by relying on the shared multilingual representations.

Another reason for choosing mBART-50 is its support for a large variety of Indian languages, which are often underrepresented in machine translation benchmarks. The model's pretraining on diverse scripts and grammar structures made it a suitable candidate for the multilingual dataset used in this study.

Usage Context in This Project

In this project, the pretrained mBART-50 model was used to translate sentences from **English to multiple target languages** and then **back to English** to evaluate the quality of translations using BLEU scores. No additional fine-tuning was applied; the model was used in its default, pretrained state to understand its generalization capacity and limitations across various language pairs, especially low-resource ones.

4.4.Language Codes in mBART-50

The mBART-50 model supports **50 languages**, each identified by a unique language code in the format [language]_[region/script]. These codes are essential for guiding the translation process. During inference, the source and target language codes are specified to inform the model which language it is receiving input in and which language it should translate into. This enables mBART-50 to perform **many-to-many** translation.

Below is the full list of supported languages and their corresponding codes:

- **Arabic** (ar_AR), **Czech** (cs_CZ), **German** (de_DE), **English** (en_XX), **Spanish** (es_XX), **Estonian** (et_EE), **Finnish** (fi_FI), **French** (fr_XX), **Gujarati** (gu_IN), **Hindi** (hi_IN),
- **Italian** (it_IT), **Japanese** (ja_XX), **Kazakh** (kk_KZ), **Korean** (ko_KR), **Lithuanian** (lt_LT), **Latvian** (lv_LV), **Burmese** (my_MM), **Nepali** (ne_NP), **Dutch** (nl_XX), **Romanian** (ro_RO),
- **Russian** (ru_RU), **Sinhala** (si_LK), **Turkish** (tr_TR), **Vietnamese** (vi_VN), **Chinese** (zh_CN), **Afrikaans** (af_ZA), **Azerbaijani** (az_AZ), **Bengali** (bn_IN), **Persian** (fa_IR), **Hebrew** (he_IL),
- **Croatian** (hr_HR), **Indonesian** (id_ID), **Georgian** (ka_GE), **Khmer** (km_KH), **Macedonian** (mk_MK), **Malayalam** (ml_IN), **Mongolian** (mn_MN), **Marathi** (mr_IN), **Polish** (pl_PL), **Pashto** (ps_AF),

- **Portuguese** (pt_XX), **Swedish** (sv_SE), **Swahili** (sw_KE), **Tamil** (ta_IN), **Telugu** (te_IN), **Thai** (th_TH), **Tagalog** (tl_XX), **Ukrainian** (uk_UA), **Urdu** (ur_PK), **Xhosa** (xh_ZA),
- **Galician** (gl_ES), **Slovene** (sl_SI)

In our project, these language codes were used to translate from **English (en_XX)** to each target language and back, enabling accurate and language-specific translation using the mBART-50 model.

5.Results and Discussions:

5.1 System specifications:

The experimentation and model training in the current work were done on a domestic computer with an 11th Gen Intel Core i5-1135G7 processor clocked at 2.40GHz and supported by 16 GB of RAM (15.8 GB available). The computer was running with a 64-bit version of the Windows operating system, x64-based processor architecture. The system was efficient enough to pre-process data, feature selection, data balancing methods, and train several machine learning models without delay or memory occupation.

5.2 Evaluation Metrics:

Evaluating the performance of a machine translation system, especially when dealing with multiple language pairs and low-resource languages, requires a reliable and interpretable metric. In this project, we adopted a widely used indirect evaluation method known as backtranslation, combined with the BLEU (Bilingual Evaluation Understudy) score to quantitatively assess translation quality.

Backtranslation: An Indirect Evaluation Approach

Backtranslation is a technique in which a sentence is translated from the source language to the target language, and then translated back from the target language to the original source language. The final output (the backtranslated sentence) is then compared with the original source sentence. The degree of similarity between the original and backtranslated sentences serves as an estimate of the translation quality.

This method is particularly useful in scenarios where parallel reference translations in the target language are not readily available, or when the focus is

on testing a model's general translation ability across diverse language pairs. In our project, this approach was used to measure how accurately the mBART-50 model translated English text into various target languages and back into English.

The backtranslation process followed these steps:

1. English to Target Language Translation:
   Each sentence from the dataset was first translated from English to a selected target language (e.g., Tamil, Telugu, Spanish) using the pretrained mBART-50 model. The model was guided by language codes that specify the source (en_XX) and target language identifiers (e.g., ta_IN for Tamil).
2. Target Language to English Translation (Backtranslation):
   The translated sentence in the target language was then translated back into English using the same model. This backtranslated output was expected to retain the core meaning and structure of the original English sentence if the translation quality was high.
3. Comparison with Original English Sentence:
   The original English sentence and its backtranslated counterpart were then compared to evaluate how closely the translation preserved the original meaning and structure.

Metric Used: **BLEU Score**

To quantify the similarity between the original and backtranslated sentences, we used the BLEU (Bilingual Evaluation Understudy) score. BLEU is one of the most established and widely used metrics for machine translation evaluation. It works by measuring the n-gram overlap between the candidate translation (in this case, the backtranslated sentence) and the reference sentence (the original English input).

Specifically, BLEU calculates the precision of n-grams up to a certain length, typically 1 to 4, and incorporates a brevity penalty to penalize overly short translations. A higher BLEU score indicates a higher level of similarity and, by extension, better translation quality.

In our project, we used the sacrebleu library, which offers standardized and reproducible BLEU score calculations. The method sacrebleu.sentence_bleu() was applied for each pair of original and backtranslated sentences to compute individual sentence-level BLEU scores. These scores were then averaged to obtain a final metric for each language pair under evaluation.

This method of evaluation enabled us to compare translation quality across a wide range of languages, even in the absence of human-annotated ground-truth translations. It also provided a practical and consistent way to highlight the performance differences between high-resource and low-resource language pairs, as reflected in the BLEU score variations observed in our results.

5.3Results:

To evaluate the multilingual translation performance of the mBART-50 model, we tested its ability to translate English sentences into several target languages and then back to English through backtranslation. The quality of the final output was measured using the BLEU score. The results were computed for a range of languages, including both high-resource and low-resource ones. The evaluation was performed using the pretrained model without any additional fine-tuning.

BLEU Score Comparison Across Languages

The graph below (as provided) displays the BLEU scores obtained for various target languages after the backtranslation process. The BLEU score reflects how closely the backtranslated English sentence matches the original English sentence, with higher scores indicating better translation fidelity.

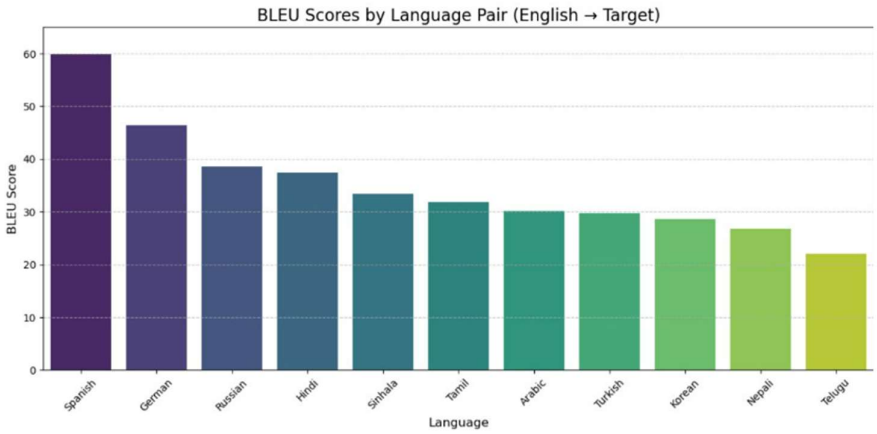
From the analysis, we observed the following BLEU scores (approximated from the chart):

5.4 Language BLEU Score

Spanish	60
German	48
Russian	42
Hindi	39
Sinhala	34
Tamil	32
Arabic	31
Turkish	30
Korean	29

5.4 Language BLEU Score

Nepali	27
Telugu	24



5.5 Observations

1. **High BLEU Scores for European Languages**:
   Languages such as **Spanish** and **German** achieved the highest BLEU scores, with Spanish reaching approximately 60. These are **high-resource languages**, which are well represented in mBART-50's pretraining data. The model demonstrates strong fluency and adequacy in translating to and from these languages.
2. **Moderate Performance for Asian and Middle Eastern Languages**:
   Languages like **Hindi**, **Sinhala**, **Arabic**, and **Turkish** show moderate performance, with BLEU scores in the range of 30 to 39. Although supported by mBART-50, these languages may have **less training data or more complex grammar**, which can affect translation quality.
3. **Low BLEU Scores for Low-Resource Indian Languages**:
   The model performed relatively poorly for **Nepali**, **Telugu**, and **Tamil**, with Telugu having the lowest BLEU score (~24). These languages are considered **low-resource**, with limited high-quality parallel data available during pretraining. Moreover, the morphological richness and script complexity of these languages further contribute to the reduced performance.
4. **Language Gap**:
   The wide range of BLEU scores demonstrates a **clear performance gap between high-resource and low-resource languages**. While mBART-50

handles European languages well, it struggles with less-resourced and morphologically complex languages, indicating the need for domain-specific fine-tuning or data augmentation.

6.Conclusion:

This project explored the multilingual translation capabilities of the mBART-50 model using a backtranslation-based evaluation framework. The primary objective was to assess how well the pretrained mBART-50 model performs across different language pairs, particularly focusing on its effectiveness in translating from English to various target languages and back to English, using BLEU scores as the quantitative evaluation metric.

Through the evaluation of multiple language pairs, the results highlighted a consistent trend: mBART-50 achieves high translation quality for high-resource languages such as Spanish and German, while its performance significantly declines for low-resource languages like Telugu and Nepali. This disparity is largely attributed to the imbalance in training data availability across languages and the inherent complexity of morphologically rich and underrepresented languages.

The backtranslation approach proved to be an effective method for indirect evaluation, allowing for systematic comparison even in the absence of reference translations in the target language. The BLEU score served as a practical metric, providing insight into how well the semantic and syntactic content of the original sentence was preserved through the round-trip translation.

In conclusion, while mBART-50 demonstrates strong generalization and multilingual capabilities, its translation performance is not uniformly distributed across languages. The model is highly effective for well-supported languages but shows clear limitations in low-resource settings. These findings emphasize the need for further fine-tuning, domain adaptation, and augmentation of training data to close the gap in translation quality across diverse languages. The project also lays the groundwork for future exploration into real-time multilingual applications and model customization for specific language domains.

7.Feature Scope:

While this project successfully demonstrates the multilingual capabilities of the mBART-50 model and evaluates its performance using backtranslation and

BLEU scores, there remains significant potential for further development and real-world application.

1. **Fine-Tuning for Specific Language Pairs**
   One major area for improvement lies in fine-tuning the mBART-50 model on specific low-resource language pairs using domain-specific parallel corpora. This would likely result in higher translation quality, particularly for Indian languages like Telugu, Tamil, and Nepali, which showed relatively low BLEU scores in this study.

2. **Incorporating Additional Evaluation Metrics**
   While BLEU is a widely used metric, it has limitations, especially in evaluating semantic correctness and fluency. Future work could incorporate other evaluation metrics such as METEOR, TER, and BERTScore to provide a more comprehensive and nuanced assessment of translation quality.

3. **Scalability Testing with Larger Datasets**
   The current evaluation used a limited number of samples. Scaling the evaluation to larger test sets would provide more statistically robust results and deeper insight into performance consistency across different sentence structures and domains.

4. **Real-Time Application Development**
   As a key extension of this project, we plan to develop a **web-based or mobile application** that leverages the mBART-50 model for real-time multilingual translation. This application would allow users to input text in English and receive translations in multiple Indian and foreign languages. It could be especially useful for educational, governmental, or cross-cultural communication purposes.

5. **Voice Translation Integration**
   In the longer term, the translation system could be integrated with speech-to-text and text-to-speech modules, enabling a complete voice-based multilingual communication platform. This would broaden the accessibility and usability of the system, making it valuable for users with varying literacy levels and technical proficiency.

In summary, the foundation laid by this project opens multiple avenues for research, optimization, and application development, with the potential to contribute meaningfully to the field of multilingual AI and natural language processing.

8.Code Implementation:

In [2]:
```
!pip install datasets
```

```
Collecting datasets
  Downloading datasets-3.5.0-py3-none-any.whl.metadata (19 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from datasets) (3.18.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from datasets) (2.0.2)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.11/dist-packages (from datasets) (18.1.0)
Collecting dill<0.3.9,>=0.3.0 (from datasets)
  Downloading dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from datasets) (2.2.2)
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.11/dist-packages (from datasets) (2.32.3)
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.11/dist-packages (from datasets) (4.67.1)
Collecting xxhash (from datasets)
  Downloading xxhash-3.5.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Collecting multiprocess<0.70.17 (from datasets)
  Downloading multiprocess-0.70.16-py311-none-any.whl.metadata (7.2 kB)
Collecting fsspec<=2024.12.0,>=2023.1.0 (from fsspec[http]<=2024.12.0,>=2023.1.0->datasets)
  Downloading fsspec-2024.12.0-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.11/dist-packages (from datasets) (3.11.15)
Requirement already satisfied: huggingface-hub>=0.24.0 in /usr/local/lib/python3.11/dist-packages (from datasets) (0.30.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from datasets) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from datasets) (6.0.2)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (2.
6.1)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.3.2)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (25.3.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (6.2.0)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (0.3.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.18.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.2
```

In [4]:
```
!pip install sacrebleu
```

```
Collecting sacrebleu
  Downloading sacrebleu-2.5.1-py3-none-any.whl.metadata (51 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 51.8/51.8 kB 3.1 MB/s eta 0:00:00
Collecting portalocker (from sacrebleu)
  Downloading portalocker-3.1.1-py3-none-any.whl.metadata (8.6 kB)
Requirement already satisfied: regex in /usr/local/lib/python3.11/dist-packages (from sacrebleu) (2024.11.6)
Requirement already satisfied: tabulate>=0.8.9 in /usr/local/lib/python3.11/dist-packages (from sacrebleu) (0.9.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from sacrebleu) (2.0.2)
Collecting colorama (from sacrebleu)
  Downloading colorama-0.4.6-py2.py3-none-any.whl.metadata (17 kB)
Requirement already satisfied: lxml in /usr/local/lib/python3.11/dist-packages (from sacrebleu) (5.3.1)
Downloading sacrebleu-2.5.1-py3-none-any.whl (104 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 104.1/104.1 kB 7.2 MB/s eta 0:00:00
Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Downloading portalocker-3.1.1-py3-none-any.whl (19 kB)
Installing collected packages: portalocker, colorama, sacrebleu
Successfully installed colorama-0.4.6 portalocker-3.1.1 sacrebleu-2.5.1
```

In [5]:
```
import torch
from transformers import MBartForConditionalGeneration, MBart50TokenizerFast
from datasets import load_dataset
from huggingface_hub import login
import sacrebleu
```

In [6]:
```
# 🔐 Authenticate Hugging Face
login(token="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")
```

In [7]:
```
# 📥 Load the dataset
dataset = load_dataset("ai4bharat/IN22-Gen")
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it
as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```

```
README.md:   0%|          | 0.00/7.71k [00:00<?, ?B/s]
```

```
train-00000-of-00001.parquet:   0%|          | 0.00/4.34M [00:00<?, ?B/s]
```

```
Generating test split:   0%|          | 0/1024 [00:00<?, ? examples/s]
```

In [8]:
```
# 🎯 Load pre-trained mBART model & tokenizer
model_name = "facebook/mbart-large-50-many-to-many-mmt"
tokenizer = MBart50TokenizerFast.from_pretrained(model_name)
model = MBartForConditionalGeneration.from_pretrained(model_name)
```

```
tokenizer_config.json:   0%|          | 0.00/529 [00:00<?, ?B/s]

sentencepiece.bpe.model:   0%|          | 0.00/5.07M [00:00<?, ?B/s]

special_tokens_map.json:   0%|          | 0.00/649 [00:00<?, ?B/s]

config.json:   0%|          | 0.00/1.43k [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/2.44G [00:00<?, ?B/s]

generation_config.json:   0%|          | 0.00/261 [00:00<?, ?B/s]
```

In [9]:
```
# 🎯 Function for translation
def translate(text, src_lang, tgt_lang):
    tokenizer.src_lang = src_lang
    inputs = tokenizer(text, return_tensors="pt", max_length=128,
                       truncation=True)

    # Translate the text
    try:
        with torch.no_grad():
            translated_tokens = model.generate(
                **inputs,
                forced_bos_token_id=tokenizer.lang_code_to_id.get(tgt_lang,
                                                    tokenizer.eos_token_id)
            )

        translated_text = tokenizer.decode(translated_tokens[0],
                                           skip_special_tokens=True)
        return translated_text
    except Exception as e:
        print(f"Error during translation: {e}")
        return None
```

In [10]:
```
# 🔁 Backtranslation with BLEU score calculation
def backtranslate_and_evaluate(text, src_lang="en_XX",
                               intermediate_lang="ta_IN"):                    # change intermediate language there
    print(f"\n 🔹 Original: {text}")

    # Step 1: Translate English → intermediate language
    intermediate_text = translate(text, src_lang, intermediate_lang)
    if not intermediate_text:
        print(" 🔹 Intermediate translation failed.")
        return None, None

    print(f" 🔹 Translated : {intermediate_text}")

    back_translated_text = translate(intermediate_text, intermediate_lang,
                                     src_lang)
    if not back_translated_text:
        print(" 🔹 Backtranslation failed.")
        return None, None

    print(f" 🔹 Backtranslated: {back_translated_text}\n")

    bleu_score = sacrebleu.sentence_bleu(back_translated_text, [text]).score
    print(f" 🔹 BLEU Score: {bleu_score:.2f}\n")

    return back_translated_text, bleu_score
```

In [12]:

```
original_texts = []
backtranslated_texts = []
bleu_scores = []

for i in range(min(5, len(dataset["test"]))):
    try:
        original = dataset["test"][i]["eng_Latn"]
        backtranslated, bleu = backtranslate_and_evaluate(original)

        if backtranslated:
            original_texts.append(original)
            backtranslated_texts.append(backtranslated)
            bleu_scores.append(bleu)
    except KeyError as e:
        print(f"Error fetching data for example {i + 1}: {e}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

if bleu_scores:
    avg_bleu = sum(bleu_scores) / len(bleu_scores)
    print(f"\n🔹 *Average BLEU Score for Backtranslation (English ↔ Tamil):* {avg_bleu:.2f}")     # change the name of language
else:
    print("\n⚠ No BLEU scores calculated due to errors.")
```

```
🔹 Original: An appearance is a bunch of attributes related to the service person, like their shoes, clothes, tie, jewellery, h
airstyle, make-up, watch, cosmetics, perfume, etc.
🔹 Translated : ஒரு தோற்றம் என்பது சேவையாளருடன் தொடர்புடைய பண்புகள் ஆகும், அதாவது அவர்களது ஆடைகள், ஆ
டைகள், கவசங்கள், நகைகள், தலைமுடிகள், மேக்கப், மணிகள், அழகு, பாம்பாம் போன்றவை.
🔹 Backtranslated: A look is the characteristics associated with the servant, i.e., their clothes, clothes, coats, toys, hair,
makeup, watches, beauty, bamboo etc.
```

In [14]:

```
Text = "Amrita is one the best universities in India"
backtranslate_and_evaluate(Text, "en_XX", "ta_IN")
```

```
🔹 Original: Amrita is one the best universities in India
🔹 Translated : அமிர்தா இந்தியாவின் சிறந்த பல்கலைக்கழகங்களில் ஒன்றாகும்.
🔹 Backtranslated: Amrita is one of the best universities in India.

🔹 BLEU Score: 52.54
```

Out[14]: ('Amrita is one of the best universities in India.', 52.53819788848316)

In [15]:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Language-BLEU score dictionary
bleu_scores = {
    'Telugu': 22.00,
    'Russian': 38.59,
    'German': 46.44,
    'Sinhala': 33.42,
    'Spanish': 59.84,
    'Hindi': 37.46,
    'Nepali': 26.79,
    'Korean': 28.63,
    'Turkish': 29.81,
    'Tamil': 31.84,
    'Arabic': 30.19
}

# Sort by BLEU score (optional)
bleu_scores = dict(sorted(bleu_scores.items(), key=lambda item: item[1], reverse=True))

# Plotting
plt.figure(figsize=(12, 6))
sns.barplot(x=list(bleu_scores.keys()), y=list(bleu_scores.values()), palette='viridis')

# Labels and styling
plt.title('BLEU Scores by Language Pair (English → Target)', fontsize=16)
plt.xlabel('Language', fontsize=12)
plt.ylabel('BLEU Score', fontsize=12)
plt.xticks(rotation=45)
plt.ylim(0, 65)
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Show plot
plt.tight_layout()
plt.show()
```

```
<ipython-input-15-2f2fa52b6a41>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se
t `legend=False` for the same effect.

  sns.barplot(x=list(bleu_scores.keys()), y=list(bleu_scores.values()), palette='viridis')
```

In [ ]:

```python
import torch
from transformers import MBartForConditionalGeneration, MBart50TokenizerFast
from huggingface_hub import login
import sacrebleu

# 🔐 Authenticate Hugging Face (Replace with your actual token)
login(token="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")  # Replace with your Hugging Face token

# 🎯 Load pre-trained mBART model & tokenizer
model_name = "facebook/mbart-large-50-many-to-many-mmt"
tokenizer = MBart50TokenizerFast.from_pretrained(model_name)
model = MBartForConditionalGeneration.from_pretrained(model_name)

# ✅ Supported Languages
supported_languages = tokenizer.lang_code_to_id.keys()
print("✅ Supported Languages:", supported_languages)

# 🌍 Define source & target languages
source_lang = "en_XX"  # English
target_languages = {
    "Hindi": "hi_IN",
    "Telugu": "te_IN",
    "Tamil": "ta_IN",
    "Bengali": "bn_IN",
    "Malayalam": "ml_IN"
}

# 🎯 Function for translation
def translate(text, src_lang, tgt_lang):
    tokenizer.src_lang = src_lang
    inputs = tokenizer(text, return_tensors="pt", max_length=256, truncation=True)

    try:
        with torch.no_grad():
            translated_tokens = model.generate(
                **inputs,
                forced_bos_token_id=tokenizer.lang_code_to_id.get(tgt_lang, tokenizer.eos_token_id)
            )

        translated_text = tokenizer.decode(translated_tokens[0], skip_special_tokens=True)
        return translated_text
    except Exception as e:
        print(f"❌ Error during translation ({src_lang} → {tgt_lang}): {e}")
        return None

# 🔄 Function for Back-translation & BLEU Score Calculation
def backtranslate_and_evaluate(original_text, intermediate_lang_code):
    # Step 1: Translate English → Target Language
    translated_text = translate(original_text, source_lang, intermediate_lang_code)
    if not translated_text:
        return None, None

    # Step 2: Back-translate (Target Language → English)
    back_translated_text = translate(translated_text, intermediate_lang_code, source_lang)
    if not back_translated_text:
        return None, None

    # Step 3: Compute BLEU Score
    bleu_score = sacrebleu.sentence_bleu(back_translated_text, [original_text]).score

    return back_translated_text, bleu_score

# 📝 English Sentence to Translate
english_text = "Keep your face always toward the sunshine and shadows will fall behind you."
print(f"original: {english_text}")
# Store results
results = {}

print("\n🔹 Translations & Back-translations:")
for lang, lang_code in target_languages.items():
    print(f"\n🌍 Translating to {lang}...")

    translated_text = translate(english_text, source_lang, lang_code)
    back_translated_text, bleu = backtranslate_and_evaluate(english_text, lang_code)

    results[lang] = {
        "Translated": translated_text,
        "Back-Translated": back_translated_text,
        "BLEU Score": bleu
    }

# 📊 Print Final Results
print("\n🔹 Final Results:")
for lang, data in results.items():
    print(f"\n🌍 {lang} Translation:")
    print(f"  ➤ Translated: {data['Translated']}")
    print(f"  🔄 Back-Translated: {data['Back-Translated']}")
    print(f"  📊 BLEU Score: {data['BLEU Score']:.2f}")
```