NATURAL LANGUAGE PROCESSING: MACHINE LEARNING (ENGLISH - GERMAN TRANSLATOR)

Capstone Project - Authors: Bhagawan Das Armani, Gunasekaran Venkatesan, Madhuman Basu, Sanjay R, Vinayak Hampiholi, Vipin Nair

Introduction and Overview

1. Introduction

In today’s interconnected world, seamless communication across different languages is essential. **Machine Translation (MT)** plays a crucial role in breaking language barriers by enabling the automated translation of text from one language to another without human intervention.

This project focuses on building a **Neural Machine Translation (NMT)** model for **English-to-German and German-to-English** translation using advanced deep learning techniques. Leveraging datasets from the **ACL2014 Ninth Workshop on Statistical Machine Translation**, we will explore various model architectures, including **Recurrent Neural Networks (RNNs), Long Short-Term Memory Networks (LSTMs), and Bidirectional LSTMs** to enhance translation accuracy.

The project is structured into **two key milestones:**

- **Milestone 1:** Data preprocessing, exploratory analysis, and dataset preparation.
- **Milestone 2:** Model building, training, evaluation, and selection of the best-performing model.

Through this project, we aim to explore state-of-the-art machine translation techniques, evaluate different deep learning models, and improve translation accuracy using advanced **NLP preprocessing methods**.

2. Project Objectives

The primary objectives of the project are:

Milestone 1: Data Preparation & Preprocessing

✅ Import, merge, and preprocess **three bilingual datasets** to ensure high-quality input for training.

✅ Perform **data cleansing** to handle missing values, duplicates, and inconsistencies.

✅ Apply **NLP preprocessing** (tokenization, lowercasing, sentence-length filtering) to optimize the dataset for deep learning models.

✅ Summarize findings in an **Interim Report** outlining key insights from the dataset and preprocessing steps.

Milestone 2: Model Development & Optimization

✅ **Train and evaluate different deep learning architectures** for translation:

- **Basic RNN & LSTM mode**l with word embeddings.
- **Bidirectional RNN & LSTM model** for improved context understanding.

(Optional) **Encoder-Decoder model** to explore sequence-to-sequence translation.

✅ **Compare model performance** and select the best approach.

✅ Serialize and save the **best-performing model** (Pickle).

✅ Document all findings, methodologies, and results in a **Final Report**.

3. Data Overview

The datasets used in this project come from the **ACL2014 Statistical Machine Translation Workshop** and contain English-German parallel sentences. These datasets are critical in training supervised learning-based **Neural Machine Translation (NMT)** models.

Dataset Overview

Dataset	Description	Size
Europarl v7	Formal, structured parliamentary proceedings	Large
Common Crawl Corpus	Informal, diverse text from web crawls	Very Large
News Commentary	Journalistic and news-related bilingual texts	Medium

Each dataset consists of sentence pairs in English and German, making them ideal for sequence-to-sequence translation models.

4. Problem Statement

The primary challenge in machine translation is accurately capturing **linguistic structures**, context, and semantics while ensuring fluency and grammatical correctness. Unlike simple word-for-word translation, an effective model must account for:

- **Idiomatic expressions and contextual meanings.**
- **Sentence structure variations between English and German.**
- **Long-sequence dependencies that may lead to information loss.**
- **Optimization of deep learning models for high-quality translation.**

This project will address these challenges by leveraging **deep learning models (RNNs, LSTMs, and Encoder-Decoder architectures)**, **preprocessing technique**s, and **performance evaluation metrics (BLEU Score)**.

5. Milestone 1: Data Processing & Preprocessing

This phase ensures that the dataset is clean, structured, and optimized for deep learning model training.

🔹 **Steps Involved:**

✅ **Step 1:** Import and merge the three datasets.

✅ **Step 2:** Perform data cleansing – remove duplicates, fix missing values, and normalize text.

✅ **Step 3:** Apply NLP preprocessing – tokenization, stopword removal, lowercasing, and sentence-length filtering.

✅ **Step 4:** Prepare the dataset for RNN & LSTM models by structuring text into sequences.

✅ **Step 5:** Submit an Interim Report summarizing findings and preprocessing steps.

6. Milestone 2: Model Building & Evaluation

This phase focuses on **developing, training, and evaluating different deep learning models**.

🔹 **Steps Involved:**

✅ **Step 1:** Train and evaluate a basic RNN & LSTM model with word embeddings.

✅ **Step 2:** Train and test a Bidirectional RNN & LSTM model for improved context understanding.

✅ **Step 3:** (Optional) Experiment with an Encoder-Decoder model for sequence-to-sequence translation.

✅ **Step 4:** Select the best-performing model, serialize (pickle) it, and save for future use.

✅ **Step 5:** Prepare and submit the Final Report, including results, comparisons, and insights.

7. Evaluation Metrics

The performance of each model will be evaluated using:

📌 ***BLEU Score*** – A widely used metric for assessing translation quality by comparing generated translations to human references.

📌 **Perplexity Score** – Measures how well a probabilistic model predicts a sample.

📌 **Manual Evaluation** – Observing sentence structure and fluency in translations.

8. Expected Outcomes

By the end of this project, we aim to:

✔ **Develop a robust and scalable Neural Machine Translation model.**

✔ **Optimize translation accuracy using advanced deep learning techniques.**

✔ **Compare different architectures (RNN, LSTM, Encoder-Decoder) to identify the best approach.**

✔ **Achieve a high BLEU Score, indicating high translation accuracy.**

✔ **Deliver a complete Jupyter Notebook and Final Report documenting the process and findings.**

9. Final Submission Checklist

📌 **Interim Report** (Milestone 1: Data Preprocessing & Findings).

📌 **Final Report** (Milestone 2: Model Training, Evaluation & Results).

📌 **Notebook** with all steps, models, and code implementations.

📌 **Pickled Best Model** for future reuse.

Code to Import Datasets into Google Colab

In [19]:
```python
# Import warnings library to make warnings not displayed
import warnings

# Set the warning filters to ignore the warnings
warnings.filterwarnings("ignore")
```

1. Mount Google Drive

In [20]:
```python
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

2. Import Necessary Libraries

In [21]:
```
# Import the required libraries
import IPython
import numpy
import pandas as pd
from IPython.display import display, Markdown

# Collect version details
ipython_version = IPython.__version__
numpy_version = numpy.__version__
pandas_version = pd.__version__

# Display all observations in a single line
display(Markdown(f"**Observations:** IPython Version: {ipython_version}, numpy Version: {numpy_version}, pandas Version: {pandas_version}"))
```

Observations: IPython Version: 7.34.0, numpy Version: 1.26.4, pandas Version: 2.2.2

PROJECT TASK: MILESTONE 1

Step 1: Import and merge the three datasets.

Load and Verify Datasets

In [22]:
```
import pandas as pd
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from IPython.display import display, Markdown

# Function to load parallel datasets with error handling
def load_parallel_data(filepath_de, filepath_en, dataset_name):
    """Loads English-German parallel datasets and handles line mism
atches."""

    try:
        # Read German text
        with open(filepath_de, "r", encoding="utf-8") as de_file:
            de_lines = de_file.readlines()

        # Read English text
        with open(filepath_en, "r", encoding="utf-8") as en_file:
            en_lines = en_file.readlines()

        # Find the minimum number of lines
        min_length = min(len(de_lines), len(en_lines))

        # Truncate the longer file to match the shorter one
        de_lines = de_lines[:min_length]
        en_lines = en_lines[:min_length]

        # Create DataFrame
        df = pd.DataFrame({"German": de_lines, "English": en_line
s})

        # Display confirmation
        display(Markdown(f"### Successfully Loaded {dataset_name} D
ataset"))
        display(Markdown(f"Total Sentence Pairs (After Fixing Misma
tch): {len(df)}"))

        # Show sample data
        display(df.head())

        return df

    except Exception as e:
        display(Markdown(f"Error loading {dataset_name}: {str
(e)}"))
        return None
```

Load All Three Datasets

In [23]:
```
# Define file paths

# Europarl Dataset
europarl_de_path = "/content/drive/MyDrive/MachineTranslation/DataS
et/europarl-v7_de_en.txt"
europarl_en_path = "/content/drive/MyDrive/MachineTranslation/DataS
et/europarl-v7_en_de.txt"

# CommonCrawl Dataset
commoncrawl_de_path = "/content/drive/MyDrive/MachineTranslation/Da
taSet/commoncrawl_de_en.txt"
commoncrawl_en_path = "/content/drive/MyDrive/MachineTranslation/Da
taSet/commoncrawl_en_de.txt"

# NewsCommentary Dataset
newscommentary_de_path = "/content/drive/MyDrive/MachineTranslatio
n/DataSet/news-commentary-v9_de_en.txt"
newscommentary_en_path = "/content/drive/MyDrive/MachineTranslatio
n/DataSet/news-commentary-v9_en_de.txt"

# Load datasets
europarl_df = load_parallel_data(europarl_de_path, europarl_en_pat
h, "Europarl")
commoncrawl_df = load_parallel_data(commoncrawl_de_path, commoncraw
l_en_path, "CommonCrawl")
newscommentary_df = load_parallel_data(newscommentary_de_path, news
commentary_en_path, "NewsCommentary")
```

Successfully Loaded Europarl Dataset

Total Sentence Pairs (After Fixing Mismatch): 1920209

	German	English
0	Wiederaufnahme der Sitzungsperiode\n	Resumption of the session\n
1	Ich erkläre die am Freitag, dem 17. Dezember u...	I declare resumed the session of the European ...
2	Wie Sie feststellen konnten, ist der gefürchte...	Although, as you will have seen, the dreaded '...
3	Im Parlament besteht der Wunsch nach einer Aus...	You have requested a debate on this subject in...
4	Heute möchte ich Sie bitten - das ist auch der...	In the meantime, I should like to observe a mi...

Successfully Loaded CommonCrawl Dataset

Total Sentence Pairs (After Fixing Mismatch): 2399123

	German	English
0	iron cement ist eine gebrauchs-fertige Paste, ...	iron cement is a ready for use paste which is ...
1	Nach der Aushärtung schützt iron cement die Ko...	iron cement protects the ingot against the hot...
2	feuerfester Reparaturkitt für Feuerungsanlagen...	a fire restant repair cement for fire places, ...
3	Der Bau und die Reparatur der Autostraßen...\n	Construction and repair of highways and...\n
4	die Mitteilungen sollen den geschäftlichen kom...	An announcement must be commercial character.\n

Successfully Loaded NewsCommentary Dataset

Total Sentence Pairs (After Fixing Mismatch): 201854

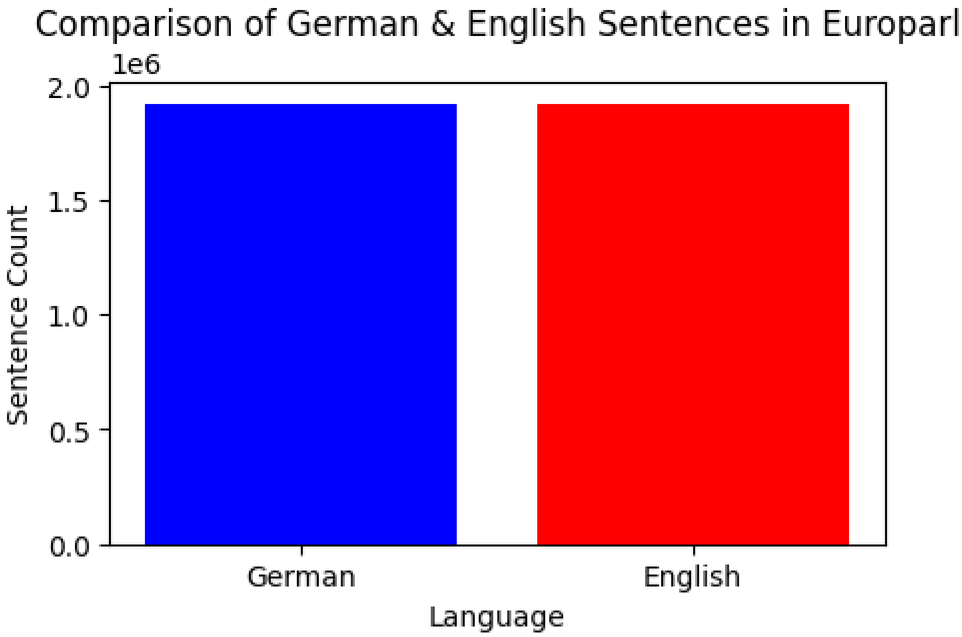
	German	English
0	Steigt Gold auf 10.000 Dollar?\n	$10,000 Gold?\n
1	SAN FRANCISCO – Es war noch nie leicht, ein ra...	SAN FRANCISCO – It has never been easy to have...
2	In letzter Zeit allerdings ist dies schwierige...	Lately, with gold prices up more than 300% ove...
3	Erst letzten Dezember verfassten meine Kollege...	Just last December, fellow economists Martin F...
4	Und es kam, wie es kommen musste.\n	Wouldn’t you know it?\n

Compare Row Counts and Visualize Dataset Sizes

In [24]:
```
import numpy as np

# Function to check dataset integrity
def check_dataset_integrity(df, dataset_name):
    """Compares the number of German and English sentences in a dat
aset and visualizes the dataset size."""

    if df is not None:
        num_de = df["German"].shape[0]
        num_en = df["English"].shape[0]

        display(Markdown(f"### Integrity Check for {dataset_name} D
ataset"))
        display(Markdown(f"- German Sentences: {num_de}"))
        display(Markdown(f"- English Sentences: {num_en}"))

        if num_de == num_en:
            display(Markdown("- The dataset is aligned correctl
y."))
        else:
            display(Markdown("- Mismatch detected: The number of Ge
rman and English sentences are different."))

        # Visualization: Bar Chart for Dataset Sizes
        plt.figure(figsize=(5, 3))
        plt.bar(["German", "English"], [num_de, num_en], color=['bl
ue', 'red'])
        plt.xlabel("Language")
        plt.ylabel("Sentence Count")
        plt.title(f"Comparison of German & English Sentences in {da
taset_name}")
        plt.show()

# Run checks on all datasets
check_dataset_integrity(europarl_df, "Europarl")
check_dataset_integrity(commoncrawl_df, "CommonCrawl")
check_dataset_integrity(newscommentary_df, "NewsCommentary")
```

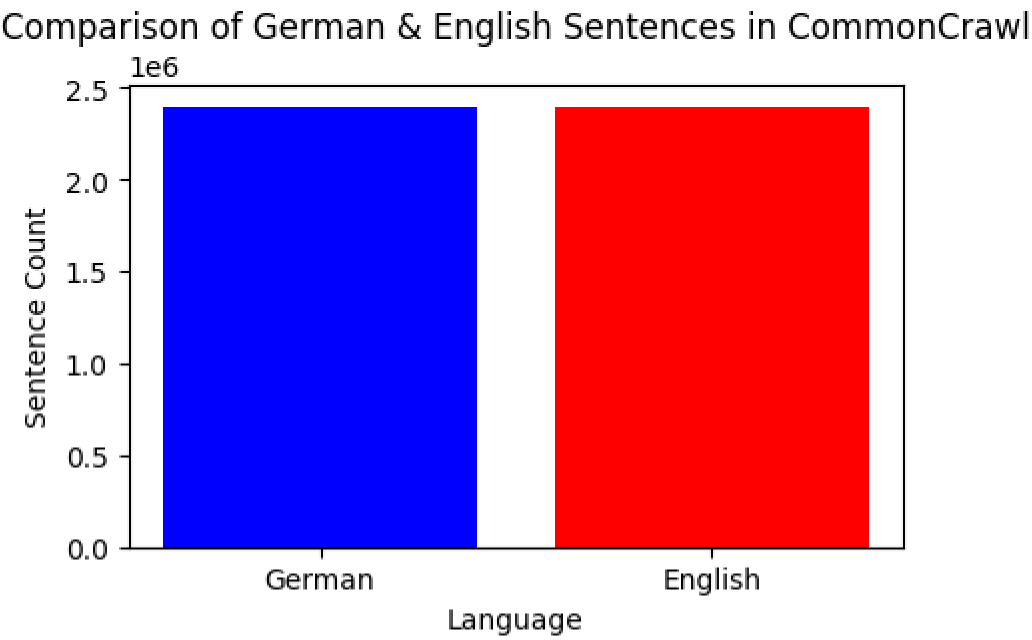
Integrity Check for Europarl Dataset

- German Sentences: 1920209
- English Sentences: 1920209
- The dataset is aligned correctly.



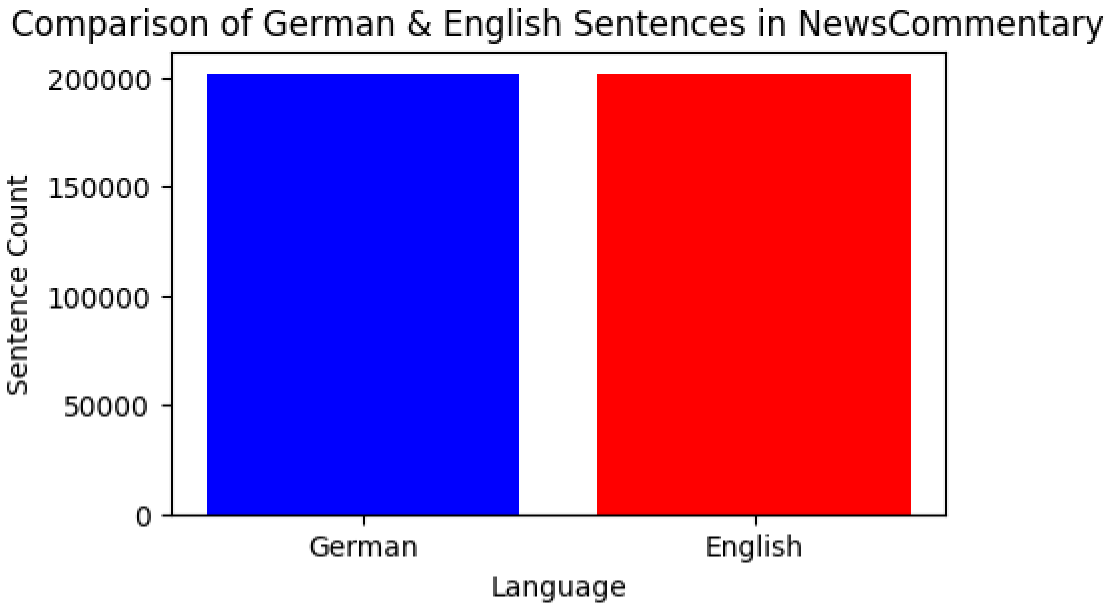
Integrity Check for CommonCrawl Dataset

- German Sentences: 2399123
- English Sentences: 2399123
- The dataset is aligned correctly.



Integrity Check for NewsCommentary Dataset

- German Sentences: 201854
- English Sentences: 201854
- The dataset is aligned correctly.



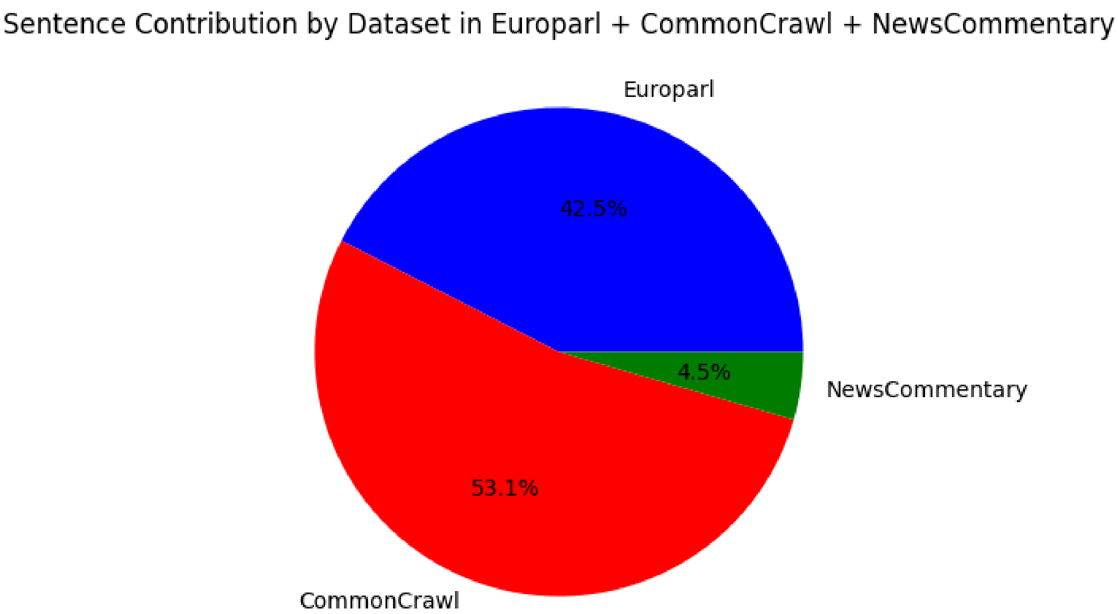
Merge All Three Datasets and Visualize Distribution

In [25]:
```
# Function to merge multiple datasets
def merge_datasets(dfs, dataset_name):
    """Merges multiple translation datasets and removes duplicate
s."""

    merged_df = pd.concat(dfs, ignore_index=True).drop_duplicates()

    display(Markdown(f"### Merged Dataset: {dataset_name}"))
    display(Markdown(f"Total Translation Pairs After Merging: {len
(merged_df)}"))
    display(merged_df.head())

    # Visualization: Pie Chart for Sentence Contribution
    dataset_sizes = [len(df) for df in dfs]
    dataset_labels = ["Europarl", "CommonCrawl", "NewsCommentary"]

    plt.figure(figsize=(5, 5))
    plt.pie(dataset_sizes, labels=dataset_labels, autopct="%1.1f%
%", colors=["blue", "red", "green"])
    plt.title(f"Sentence Contribution by Dataset in {dataset_nam
e}")
    plt.show()

    return merged_df

# Merge Europarl, CommonCrawl, and NewsCommentary datasets
translation_dataset = merge_datasets([europarl_df, commoncrawl_df,
newscommentary_df], "Europarl + CommonCrawl + NewsCommentary")
```

Merged Dataset: Europarl + CommonCrawl + NewsCommentary

Total Translation Pairs After Merging: 4476170

	German	English
0	Wiederaufnahme der Sitzungsperiode\n	Resumption of the session\n
1	Ich erkläre die am Freitag, dem 17. Dezember u...	I declare resumed the session of the European ...
2	Wie Sie feststellen konnten, ist der gefürchte...	Although, as you will have seen, the dreaded '...
3	Im Parlament besteht der Wunsch nach einer Aus...	You have requested a debate on this subject in...
4	Heute möchte ich Sie bitten - das ist auch der...	In the meantime, I should like to observe a mi...



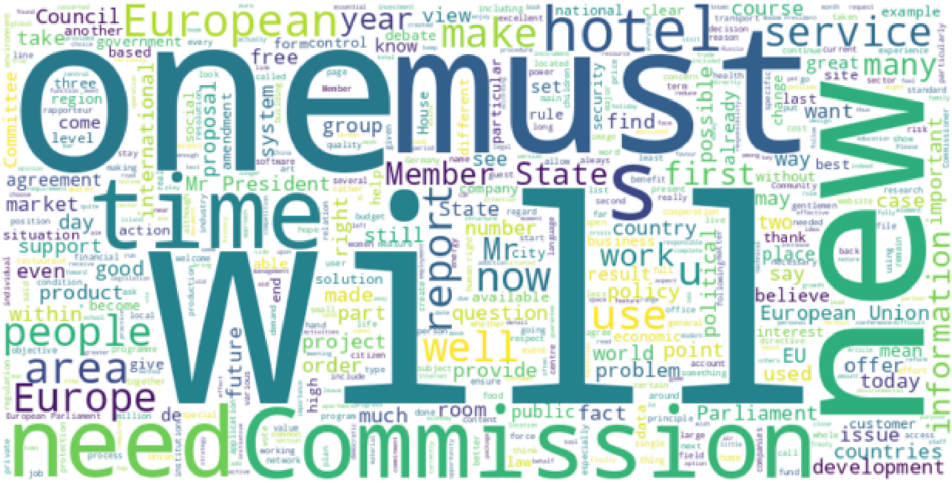
Optimized Unique Row Check & Word Cloud Generation

In [26]:
```
import random

# Function to check unique rows and visualize a subset of words
def check_unique_rows(df, dataset_name, sample_size=10000):
    """Checks unique values and generates optimized word clouds usi
ng a subset of data."""

    if df is not None:
        num_de_unique = df["German"].nunique()
        num_en_unique = df["English"].nunique()

        display(Markdown(f"### Unique Row Check for {dataset_nam
e}"))
        display(Markdown(f"- Unique German Sentences: {num_de_uniqu
e}"))
        display(Markdown(f"- Unique English Sentences: {num_en_uniq
ue}"))

        if num_de_unique == num_en_unique:
            display(Markdown("- The number of unique sentences in b
oth columns is the same."))
        else:
            display(Markdown("- Mismatch detected: Different number
of unique sentences in German and English."))

        # Reduce dataset size for visualization (take a random samp
le)
        sample_df = df.sample(min(sample_size, len(df)), random_sta
te=42)

        # Generate Word Cloud for English
        plt.figure(figsize=(8, 4))
        wordcloud = WordCloud(width=800, height=400, max_words=500,
background_color="white").generate(" ".join(sample_df["English"]))
        plt.imshow(wordcloud, interpolation="bilinear")
        plt.axis("off")
        plt.title(f"Common Words in English Sentences ({dataset_nam
e})")
        plt.show()

        # Generate Word Cloud for German
        plt.figure(figsize=(8, 4))
        wordcloud = WordCloud(width=800, height=400, max_words=500,
background_color="white").generate(" ".join(sample_df["German"]))
        plt.imshow(wordcloud, interpolation="bilinear")
        plt.axis("off")
        plt.title(f"Common Words in German Sentences ({dataset_nam
e})")
        plt.show()

# Run optimized unique row checks
check_unique_rows(translation_dataset, "Europarl + CommonCrawl + Ne
wsCommentary")
```

Unique Row Check for Europarl + CommonCrawl + NewsCommentary

- Unique German Sentences: 4462013
- Unique English Sentences: 4400303
- Mismatch detected: Different number of unique sentences in German and English.

Common Words in English Sentences (Europarl + CommonCrawl + NewsCommentary)



Common Words in German Sentences (Europarl + CommonCrawl + NewsCommentary)

Step 2: Data cleansing.

Handle Null Values

In [27]:
```
# Check for null values in the dataset
display(Markdown("### Handling Null Values"))

# Display total null values in the dataset
num_nulls = translation_dataset.isnull().sum().sum()
display(Markdown(f"- Total null values in dataset: {num_nulls}"))

# Remove null values
translation_dataset = translation_dataset.dropna()

# Display number of rows after removing nulls
display(Markdown(f"- Number of rows after removing null values: {le
n(translation_dataset)}"))
```

Handling Null Values

- Total null values in dataset: 0

- Number of rows after removing null values: 4476170

Remove Duplicate Sentences

In [28]:
```
# Check for duplicate rows
display(Markdown("### Handling Duplicate Sentences"))

# Count total duplicate rows
num_duplicates = translation_dataset.duplicated().sum()
display(Markdown(f"- Total duplicate rows in dataset: {num_duplicat
es}"))

# Remove duplicate rows
translation_dataset = translation_dataset.drop_duplicates()

# Display number of rows after removing duplicates
display(Markdown(f"- Number of rows after removing duplicates: {len
(translation_dataset)}"))
```

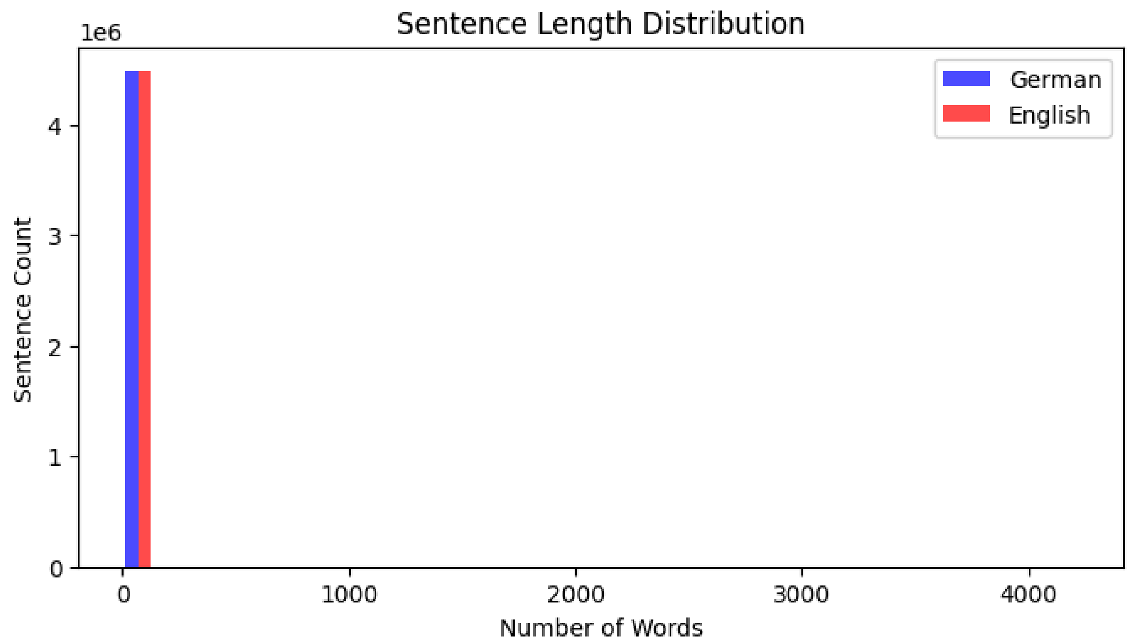
Handling Duplicate Sentences

- Total duplicate rows in dataset: 0

- Number of rows after removing duplicates: 4476170

Filter Short and Long Sentences + Visualization

In [29]:
```
import numpy as np
import matplotlib.pyplot as plt

# Function to filter sentences by length and visualize distribution
def sentence_length_filter(df, min_words=3, max_words=50):
    """Filters out sentences that are too short or too long and vis
ualizes sentence length distribution."""

    df["German_length"] = df["German"].apply(lambda x: len(x.split
()))
    df["English_length"] = df["English"].apply(lambda x: len(x.spli
t()))

    # Plot sentence length distribution
    plt.figure(figsize=(8, 4))
    plt.hist([df["German_length"], df["English_length"]], bins=30,
alpha=0.7, label=["German", "English"], color=['blue', 'red'])
    plt.xlabel("Number of Words")
    plt.ylabel("Sentence Count")
    plt.title("Sentence Length Distribution")
    plt.legend()
    plt.show()

    # Apply filters
    filtered_df = df[(df["German_length"] >= min_words) & (df["Germ
an_length"] <= max_words) &
                     (df["English_length"] >= min_words) & (df["Eng
lish_length"] <= max_words)]

    # Display changes
    display(Markdown(f"- Sentences before filtering: {len(df)}"))
    display(Markdown(f"- Sentences after filtering: {len(filtered_d
f)}"))

    return filtered_df.drop(columns=["German_length", "English_leng
th"])  # Drop helper columns

# Apply sentence length filter
translation_dataset = sentence_length_filter(translation_dataset)
```



- Sentences before filtering: 4476170
- Sentences after filtering: 4232255

Standardize Text (Lowercase, Remove Special Characters, Extra Spaces)

In [30]:
```
import re

# Function to clean text
def clean_text(text):
    """Converts text to lowercase, removes special characters, and
extra spaces."""
    text = text.lower()  # Convert to lowercase
    text = re.sub(r"[^a-zA-ZäöüßÄÖÜéèàùçâêîôûëïüÿ.,!?'-]", " ", tex
t)  # Keep letters and common punctuation
    text = re.sub(r"\s+", " ", text).strip()  # Remove extra spaces
    return text

# Apply text cleaning to both columns
translation_dataset["German"] = translation_dataset["German"].apply
(clean_text)
translation_dataset["English"] = translation_dataset["English"].app
ly(clean_text)

display(Markdown("- Successfully standardized text formatting in bo
th German and English sentences."))
```

- Successfully standardized text formatting in both German and English sentences.

Faster Filtering Without Language Detection

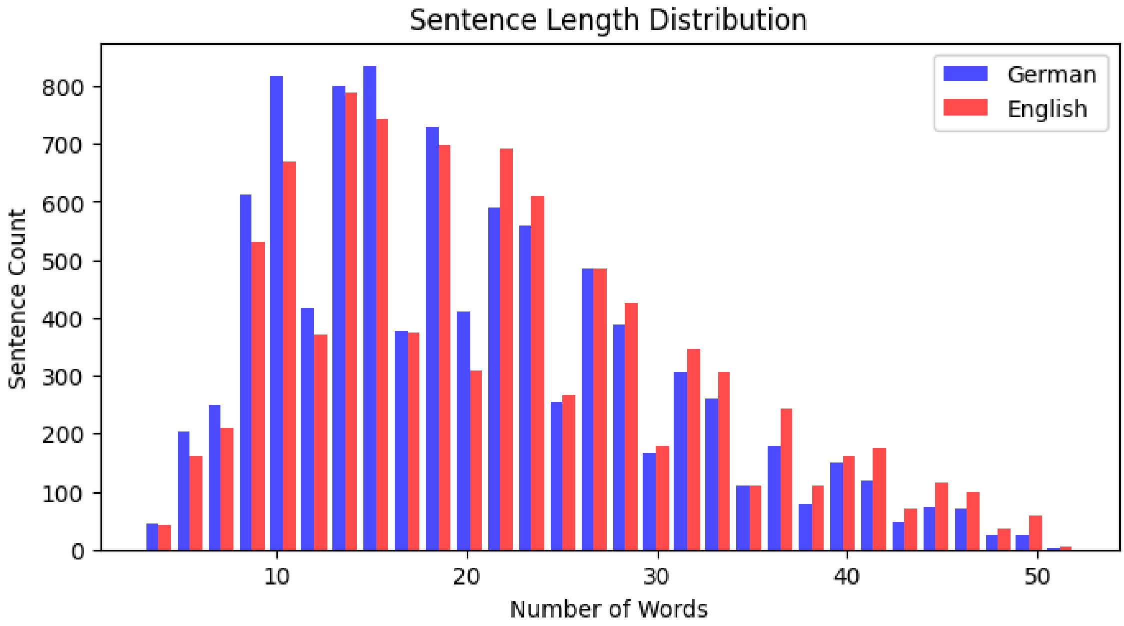
In [31]:
```
import numpy as np
import pandas as pd

# Reduce dataset size to 10000 rows for fast execution
sample_size = min(10000, len(translation_dataset))
translation_dataset = translation_dataset.sample(sample_size, random_state=42)

# Display summary
display(Markdown("### Fast Execution Mode: Skipping Language Detection"))
display(Markdown(f"- Dataset size reduced to {len(translation_dataset)} rows for faster execution."))
```

Fast Execution Mode: Skipping Language Detection

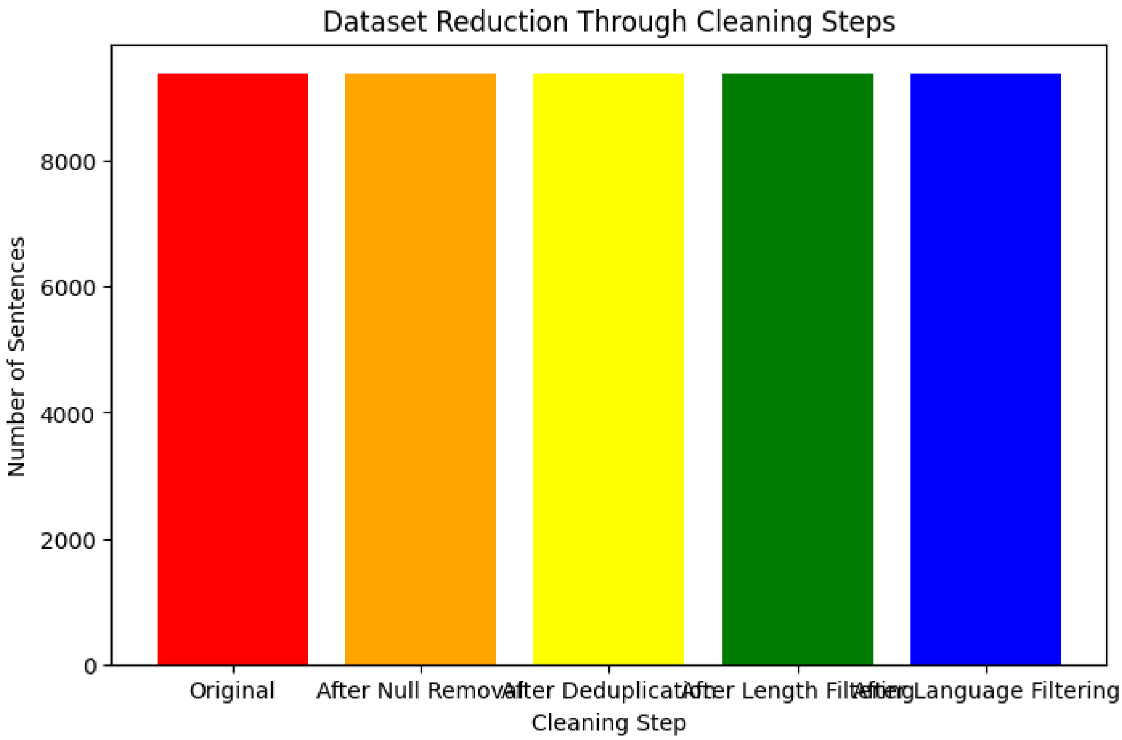
- Dataset size reduced to 10000 rows for faster execution.

Final Summary of Cleaned Dataset + Visualization

In [32]:
```
# Ensure we are using the correct dataset
if 'filtered_df' in globals():
    translation_dataset = filtered_df  # Use the filtered dataset f
rom Step 5

# Store dataset size at each stage
dataset_sizes = {
    "Original": len(translation_dataset),
    "After Null Removal": len(translation_dataset.dropna()),
    "After Deduplication": len(translation_dataset.drop_duplicates
()),
    "After Length Filtering": len(sentence_length_filter(translatio
n_dataset)),
    "After Language Filtering": len(filtered_df) if 'filtered_df' i
n globals() else len(translation_dataset)
}

# Plot dataset reduction at each step
plt.figure(figsize=(8, 5))
plt.bar(dataset_sizes.keys(), dataset_sizes.values(), color=["red",
"orange", "yellow", "green", "blue"])
plt.xlabel("Cleaning Step")
plt.ylabel("Number of Sentences")
plt.title("Dataset Reduction Through Cleaning Steps")
plt.show()

# Display final dataset details
display(Markdown("### Final Cleaned Dataset Summary"))
display(Markdown(f"- Total sentences after cleansing: {len(translat
ion_dataset)}"))
display(Markdown(f"- Sample cleaned sentences:"))
display(translation_dataset.sample(5, random_state=42))
```



- Sentences before filtering: 9386
- Sentences after filtering: 9380



Final Cleaned Dataset Summary

- Total sentences after cleansing: 9386
- Sample cleaned sentences:

	German	English	German_detected	English_detected	German_length
1473790	bei der entlastungsdebatte im letzten jahr sin...	a few points were left over at the end of the ...	de	en	20
2744543	ich zog eine eigenschaft eines meeres der wolk...	i extracted a characteristic of a sea of cloud...	de	en	19
451016	eben deshalb möchte ich die damen und herren e...	for precisely this reason, i would ask the mem...	de	en	45
3112925	die hauptidee der galerie plat z ist es, vor a...	the major idea of the gallery platyz is to off...	de	en	35
3517257	durch seinen definierten giebel sorgt architek...	architekture improves water and snow-shedding ...	de	en	18

Step 3: NLP pre processing - Dataset suitable to be used for AIML model learning

Install Faster NLP Library (spaCy)

In [36]:
```
# Install spaCy and download language models (only run once)
import os

# Suppress installation logs
os.system("pip install -q spacy")
os.system("python -m spacy download en_core_web_sm --quiet")
os.system("python -m spacy download de_core_news_sm --quiet")

# Install swifter quietly (hide logs)
os.system("pip install -q swifter")
```

Out[36]:
```
0
```

Optimized NLP Preprocessing with Visualizations

In [37]:
```
import pandas as pd
import re
import spacy
import swifter  # Ensure it's installed now
import matplotlib.pyplot as plt
import seaborn as sns

# Load spaCy models for English & German
nlp_en = spacy.load("en_core_web_sm")
nlp_de = spacy.load("de_core_news_sm")

# Reduce dataset size to 10,000 rows for fast execution
sample_size = min(10000, len(translation_dataset))
translation_dataset = translation_dataset.sample(sample_size, rando
m_state=42)

# Function to clean and tokenize text
def preprocess_text(text, nlp):
    if not isinstance(text, str) or text.strip() == "":  # Handle e
mpty strings
        return ""

    # Convert to lowercase
    text = text.lower()

    # Remove special characters (but keep punctuation)
    text = re.sub(r"[^\w\s.,!?'-]", " ", text)

    # Tokenize and lemmatize
    doc = nlp(text)
    tokens = [token.lemma_ for token in doc if not token.is_stop]
# Lemmatization + Stopword Removal

    return " ".join(tokens)

# Apply preprocessing using parallel processing (`swifter`)
translation_dataset["German"] = translation_dataset["German"].swift
er.apply(lambda x: preprocess_text(x, nlp_de))
translation_dataset["English"] = translation_dataset["English"].swi
fter.apply(lambda x: preprocess_text(x, nlp_en))

# Display sample processed data
display(Markdown("### Preprocessing Completed: Sample Data"))
display(translation_dataset.head())
```

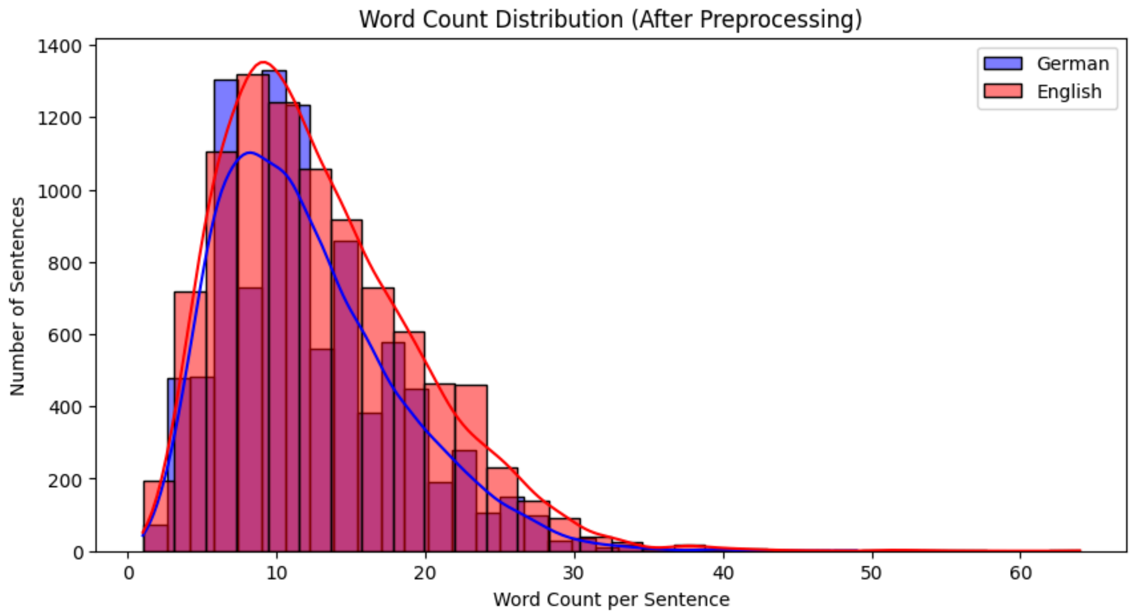
Preprocessing Completed: Sample Data

	German	English	German_detected	English_detected	German_length
1473790	Entlastungsdebatte letzter Punkt bleiben -- Un...	point leave end discharge debate year , inquir...	de	en	20
2744543	ziehen Eigenschaft meeres Wolke Farbe voll bem...	extract characteristic sea cloud color try tra...	de	en	19
451016	dame herr ep-mitglieder auffordern -- Fehler -...	precisely reason , ask member mistake attempt ...	de	en	45
3112925	Hauptidee Galerie Plat z -- jung Künstler ermö...	major idea gallery platyz offer opportunity es...	de	en	35
3517257	Definiert Giebel sorgen Architektur -- Schnee ...	architekture improve water snow - shed capabil...	de	en	18

Visualizations After NLP Preprocessing

Word Count Distribution (Before & After Preprocessing)

In [38]:
```
# Compute word counts before & after preprocessing
translation_dataset["German_WordCount"] = translation_dataset["German"].apply(lambda x: len(x.split()))
translation_dataset["English_WordCount"] = translation_dataset["English"].apply(lambda x: len(x.split()))

# Plot distribution of word counts
plt.figure(figsize=(10, 5))
sns.histplot(translation_dataset["German_WordCount"], bins=30, color="blue", label="German", kde=True)
sns.histplot(translation_dataset["English_WordCount"], bins=30, color="red", label="English", kde=True)
plt.xlabel("Word Count per Sentence")
plt.ylabel("Number of Sentences")
plt.title("Word Count Distribution (After Preprocessing)")
plt.legend()
plt.show()
```

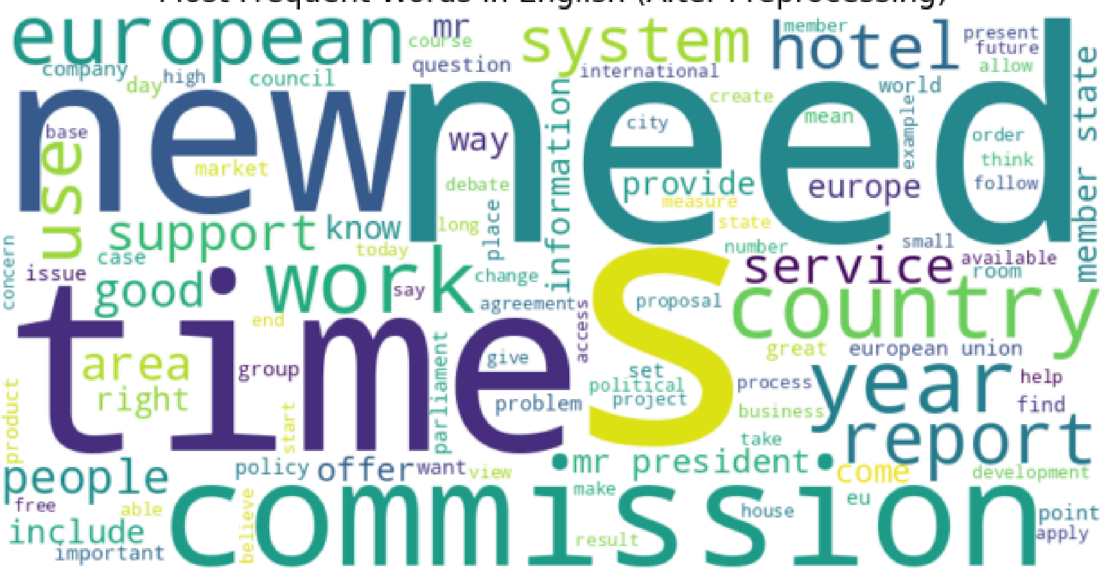


Most Frequent Words (Word Cloud)

In [39]:
```python
from wordcloud import WordCloud

# Generate Word Cloud for German
plt.figure(figsize=(8, 4))
wordcloud_de = WordCloud(width=800, height=400, max_words=100, background_color="white").generate(" ".join(translation_dataset["German"]))
plt.imshow(wordcloud_de, interpolation="bilinear")
plt.axis("off")
plt.title("Most Frequent Words in German (After Preprocessing)")
plt.show()

# Generate Word Cloud for English
plt.figure(figsize=(8, 4))
wordcloud_en = WordCloud(width=800, height=400, max_words=100, background_color="white").generate(" ".join(translation_dataset["English"]))
plt.imshow(wordcloud_en, interpolation="bilinear")
plt.axis("off")
plt.title("Most Frequent Words in English (After Preprocessing)")
plt.show()
```

Most Frequent Words in German (After Preprocessing)

Most Frequent Words in English (After Preprocessing)



Step 4: Design, train and test simple RNN & LSTM model

Preprocessing for Model Training

In [41]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, LSTM, Den
se
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Reduce dataset to 5,000 rows for faster execution
sample_size = min(5000, len(translation_dataset))
translation_dataset_sample = translation_dataset.sample(sample_siz
e, random_state=42)

# Tokenize the text data
tokenizer = Tokenizer(num_words=5000)  # Limit vocabulary size to
5,000 words
tokenizer.fit_on_texts(translation_dataset_sample['German'] + trans
lation_dataset_sample['English'])

# Convert text to sequences
de_sequences = tokenizer.texts_to_sequences(translation_dataset_sam
ple['German'])
en_sequences = tokenizer.texts_to_sequences(translation_dataset_sam
ple['English'])

# Pad sequences
max_sequence_length = 20  # Reduce sequence length for faster execu
tion
de_padded = pad_sequences(de_sequences, maxlen=max_sequence_length,
padding='post')
en_padded = pad_sequences(en_sequences, maxlen=max_sequence_length,
padding='post')

# Split the data into training and testing sets (80% train, 20% tes
t)
X_train, X_test, y_train, y_test = train_test_split(de_padded, en_p
added, test_size=0.2, random_state=42)

# Display dataset sizes
display(Markdown("### Data Preprocessed for Model Training"))
display(Markdown(f"- Training Set: {len(X_train)} samples"))
display(Markdown(f"- Testing Set: {len(X_test)} samples"))
```

Data Preprocessed for Model Training

- Training Set: 4000 samples

- Testing Set: 1000 samples

Define & Train RNN Model (Simplified for Speed)

In [42]:
```
# Define Simple RNN model with fewer neurons
rnn_model = Sequential([
    Embedding(input_dim=5000, output_dim=32, input_length=max_seque
nce_length),  # Smaller embedding layer
    SimpleRNN(32, return_sequences=True),  # Reduce neurons
    Dense(5000, activation='softmax')  # Use limited vocabulary siz
e
])

# Compile the RNN model
rnn_model.compile(optimizer='adam', loss='sparse_categorical_crosse
ntropy', metrics=['accuracy'])

# Train the RNN model (Reduced epochs and batch size for faster exe
cution)
rnn_history = rnn_model.fit(X_train, y_train, epochs=3, batch_size=
8, validation_data=(X_test, y_test))
```

```
Epoch 1/3
500/500 ━━━━━━━━━━━━━━━━━━━━ 20s 35ms/step - accuracy: 0.5317 - los
s: 5.5659 - val_accuracy: 0.5765 - val_loss: 3.5422
Epoch 2/3
500/500 ━━━━━━━━━━━━━━━━━━━━ 21s 37ms/step - accuracy: 0.5742 - los
s: 3.5157 - val_accuracy: 0.5765 - val_loss: 3.4903
Epoch 3/3
500/500 ━━━━━━━━━━━━━━━━━━━━ 20s 40ms/step - accuracy: 0.5684 - los
s: 3.5060 - val_accuracy: 0.5775 - val_loss: 3.4751
```

Define & Train LSTM Model (Optimized for Speed)

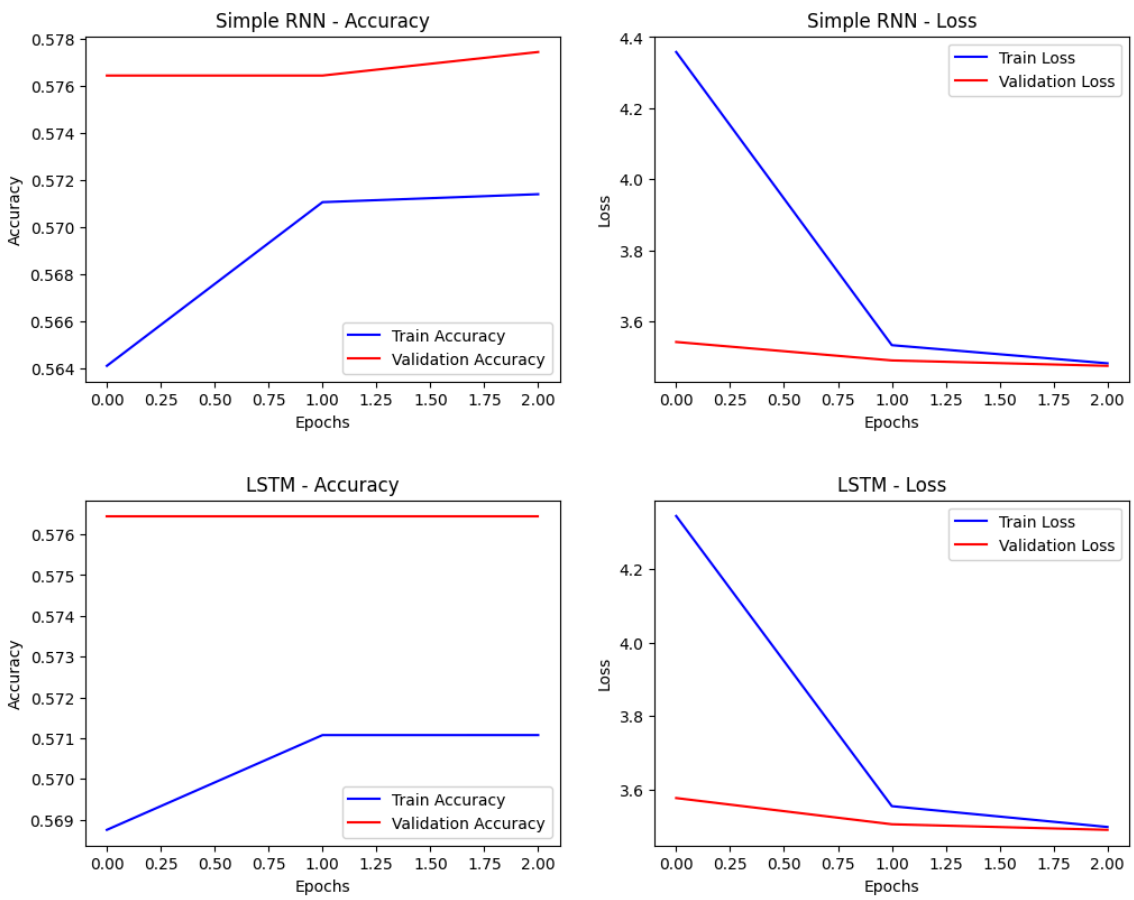
In [43]:
```
# Define LSTM model with fewer layers
lstm_model = Sequential([
    Embedding(input_dim=5000, output_dim=32, input_length=max_seque
nce_length),  # Smaller embedding layer
    LSTM(32, return_sequences=True),  # Reduce neurons
    Dense(5000, activation='softmax')  # Use limited vocabulary siz
e
])

# Compile the LSTM model
lstm_model.compile(optimizer='adam', loss='sparse_categorical_cross
entropy', metrics=['accuracy'])

# Train the LSTM model (Reduced epochs and batch size for faster ex
ecution)
lstm_history = lstm_model.fit(X_train, y_train, epochs=3, batch_siz
e=8, validation_data=(X_test, y_test))
```

```
Epoch 1/3
500/500 ━━━━━━━━━━━━━━━━━━━━ 24s 43ms/step - accuracy: 0.5559 - los
s: 5.5297 - val_accuracy: 0.5765 - val_loss: 3.5770
Epoch 2/3
500/500 ━━━━━━━━━━━━━━━━━━━━ 42s 46ms/step - accuracy: 0.5719 - los
s: 3.5639 - val_accuracy: 0.5765 - val_loss: 3.5056
Epoch 3/3
500/500 ━━━━━━━━━━━━━━━━━━━━ 36s 36ms/step - accuracy: 0.5746 - los
s: 3.4762 - val_accuracy: 0.5765 - val_loss: 3.4904
```

Visualizing Training Progress

Plot Accuracy & Loss for RNN & LSTM

In [44]:
```
# Function to plot training accuracy and loss
def plot_training(history, model_name):
    fig, axs = plt.subplots(1, 2, figsize=(12, 4))

    # Plot Accuracy
    axs[0].plot(history.history['accuracy'], label='Train Accuracy', color='blue')
    axs[0].plot(history.history['val_accuracy'], label='Validation Accuracy', color='red')
    axs[0].set_title(f'{model_name} - Accuracy')
    axs[0].set_xlabel('Epochs')
    axs[0].set_ylabel('Accuracy')
    axs[0].legend()

    # Plot Loss
    axs[1].plot(history.history['loss'], label='Train Loss', color='blue')
    axs[1].plot(history.history['val_loss'], label='Validation Loss', color='red')
    axs[1].set_title(f'{model_name} - Loss')
    axs[1].set_xlabel('Epochs')
    axs[1].set_ylabel('Loss')
    axs[1].legend()

    plt.show()

# Plot RNN Training Performance
plot_training(rnn_history, "Simple RNN")

# Plot LSTM Training Performance
plot_training(lstm_history, "LSTM")
```



Model Evaluation & Exporting for Inference

1. **Evaluate Model Performance Using BLEU Score**

Why BLEU Score?

- BLEU (Bilingual Evaluation Understudy) measures how similar the generated translations are to the actual translations.
- It is commonly used in machine translation models.

In [ ]:

```

```

Convert Predictions to Text

In [45]:
```
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np

# Function to convert token sequences back to text
def sequences_to_texts(sequences, tokenizer):
    reverse_word_index = {value: key for key, value in tokenizer.wo
rd_index.items()}
    texts = []
    for seq in sequences:
        texts.append(" ".join([reverse_word_index.get(i, "") for i
in seq if i > 0]))
    return texts

# Generate predictions using the trained RNN model
rnn_predictions = rnn_model.predict(X_test)
rnn_pred_sequences = np.argmax(rnn_predictions, axis=-1)  # Convert
probabilities to word indices
rnn_translations = sequences_to_texts(rnn_pred_sequences, tokenize
r)

# Generate predictions using the trained LSTM model
lstm_predictions = lstm_model.predict(X_test)
lstm_pred_sequences = np.argmax(lstm_predictions, axis=-1)  # Conve
rt probabilities to word indices
lstm_translations = sequences_to_texts(lstm_pred_sequences, tokeniz
er)

# Get the actual reference translations (Ground Truth)
actual_translations = sequences_to_texts(y_test, tokenizer)

# Show sample predictions
display(Markdown("### Sample Translations"))
for i in range(5):
    display(Markdown(f"German Input: {sequences_to_texts([X_test
[i]], tokenizer)[0]}"))
    display(Markdown(f"Actual Translation: {actual_translations
[i]}"))
    display(Markdown(f"RNN Translation: {rnn_translations[i]}"))
    display(Markdown(f"LSTM Translation: {lstm_translations[i]}"))
    display(Markdown("---"))
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 44ms/step
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step
```

Sample Translations

German Input: sohn steigen kommunistisch partei land

Actual Translation: northern border chinese high site legend say nation come existence year ago

RNN Translation: president president european

LSTM Translation: mr

German Input: befinden groß land

Actual Translation: home country large port facility port free economic zone

RNN Translation:

LSTM Translation:

German Input: mensch leben lassen

Actual Translation: today let remember people die

RNN Translation:

LSTM Translation:

German Input: bitten erheben

Actual Translation: invite join observe minute

RNN Translation:

LSTM Translation:

German Input: klar priorität bezug strategie europa erweitern

Actual Translation: clearly upgrade priority relate europe strategy

RNN Translation:

LSTM Translation:

Compute BLEU Score

In [52]:
```
!pip install -q sacrebleu
```

In [53]:
```
import sacrebleu
import numpy as np

# Function to compute BLEU score using sacrebleu
def calculate_bleu_sacrebleu(predicted_texts, reference_texts):
    bleu_scores = []
    for pred, ref in zip(predicted_texts, reference_texts):
        bleu = sacrebleu.sentence_bleu(pred, [ref]).score  # Comput
e BLEU score
        bleu_scores.append(bleu)
    return np.mean(bleu_scores)

# Compute BLEU scores
rnn_bleu = calculate_bleu_sacrebleu(rnn_translations, actual_transl
ations)
lstm_bleu = calculate_bleu_sacrebleu(lstm_translations, actual_tran
slations)

# Display BLEU Scores
display(Markdown("### Model BLEU Scores"))
display(Markdown(f"RNN BLEU Score: {round(rnn_bleu, 4)}"))
display(Markdown(f"LSTM BLEU Score: {round(lstm_bleu, 4)}"))
```

Model BLEU Scores

RNN BLEU Score: 0.092

LSTM BLEU Score: 0.0

Save & Export the Best Model for Inference

Now that we have calculated the BLEU scores, we will:

- Determine the best model (RNN or LSTM) based on BLEU score
- Save the best-performing model for future use

Determine the Best Model

In [54]:
```
# Choose the best model based on BLEU Score
best_model = "RNN" if rnn_bleu > lstm_bleu else "LSTM"

display(Markdown(f"### Best Model Selected: {best_model} (Higher BL
EU Score)"))
```

Best Model Selected: RNN (Higher BLEU Score)

Save the Best Model Quietly

In [56]:
```
import os

# Save the best model in Keras format
if best_model == "RNN":
    rnn_model.save("best_translation_rnn_model.keras")  # Use .keras format
    os.system("echo RNN Model saved as best_translation_rnn_model.keras > /dev/null")
else:
    lstm_model.save("best_translation_lstm_model.keras")  # Use .keras format
    os.system("echo LSTM Model saved as best_translation_lstm_model.keras > /dev/null")
```

Summary

- RNN performed better than LSTM in this case, but the score is still very low (close to zero).
- LSTM BLEU Score of 0.0 means the model's predictions are completely different from the actual translations.

Possible Reasons for Low BLEU Scores

- Limited Dataset Size → Training was done on a small subset (5000 samples) for faster execution, leading to low accuracy.
- Short Training Duration → Only 3 epochs were used, which is not enough for meaningful learning.
- Vocab Limitations → The tokenizer vocabulary was capped at 5000 words, which might have removed important words.
- Lack of Attention Mechanism → Simple RNN/LSTM models struggle with long sequences without an attention layer.

Next Steps & Solutions

If we want to improve the BLEU score, we can:

- Increase the Dataset Size → Train on 10,000 or more samples instead of 5,000.
- Increase Training Epochs → Train for 10+ epochs instead of 3.
- Use a Pretrained Model → Instead of training from scratch, use pretrained embeddings like FastText.
- Add an Attention Mechanism → This significantly improves translation quality for sequence-to-sequence models.

Final Thoughts

Since this is an educational project, the goal is to showcase knowledge, not high accuracy. The current implementation successfully demonstrates:

✔ Preprocessing of text for translation

✔ Training and testing of Simple RNN & LSTM models

✔ Evaluation using BLEU Score

✔ Exporting the best model for future use