4) Insurance Multilingual Policy Document Dataset

Project Observations:

- **Data Diversity:** The dataset includes multilingual insurance policies, providing a rich resource for cross-language NLP tasks.
- **Text Complexity:** Policy texts vary in length and structure, requiring robust models for accurate translation and summarization.
- **Model Performance:** The fine-tuned mBART model shows strong potential for generating concise, high-quality summaries across languages.
- **Challenges:** Handling language-specific nuances and maintaining consistency in summarization quality remain key areas for improvement.
- **Future Scope:** Further tuning, error analysis, and exploring other transformer models (like mT5) can enhance performance and deployment readiness.

Index

Using Real Time Dataset:

1]Newspaper Article Dataset From Kaggle [77 - 84 Slide]
2] Text Preprocessing [85 - 88 Slide]
3] mBART / mT5 Auto tokenizer [91 - 93 Slide]
4] Fine Tuning for mBART [94 - 98 Slide]
5] Fine Tuning for mT5 [99 - 105 Slide]
6] Try Again FIne Tuning for mBART- Large [106 - 107 Slide]
7] Fine Tuning for BART-small [108 - 111 Slide]
8] Prediction [112 Slide]
9] Streamlit [113 - 114 Slide]
10] Conclusion [115 Slide]

EDA

df4 - Insurance Multilingual Policy Document Dataset

Description: Collection of insurance policies in multiple languages for NLP-based translation and summarization.

```
[1]  1 import pandas as pd
```

```
[2]  1 df4=pd.read_csv('/content/df4-insurance_multilingual_policy_document_dataset.csv')
```

```
     1 df4.head()
```

	Unnamed: 0	Policy_ID	Policy_Text	Summarized_Text
0	0	bdd640fb-0667-4ad1-9c80-317fa3b1799d	Development say quality throughout beautiful. ...	Detail food shoulder argue start source husban...
1	1	1c11f735-dc71-4d96-8c0f-d195c17af08a	Civil quite others his other life edge network...	She campaign little near enter their instituti...
2	2	5496f63c-dc11-40c1-880a-adfbe7c99b26	Draw protect Democrat car very number line. Sp...	Total clearly able hospital unit size expect r...
3	3	14822f53-8201-4c62-b5f5-9b220e8fa8e0	Organization push dog build. East organization...	Data plant enough major town suffer begin inte...
4	4	5e84f058-d5a8-44eb-8939-23de8babce3b	Challenge camera final together someone team t...	Image street fight decision size parent focus ...

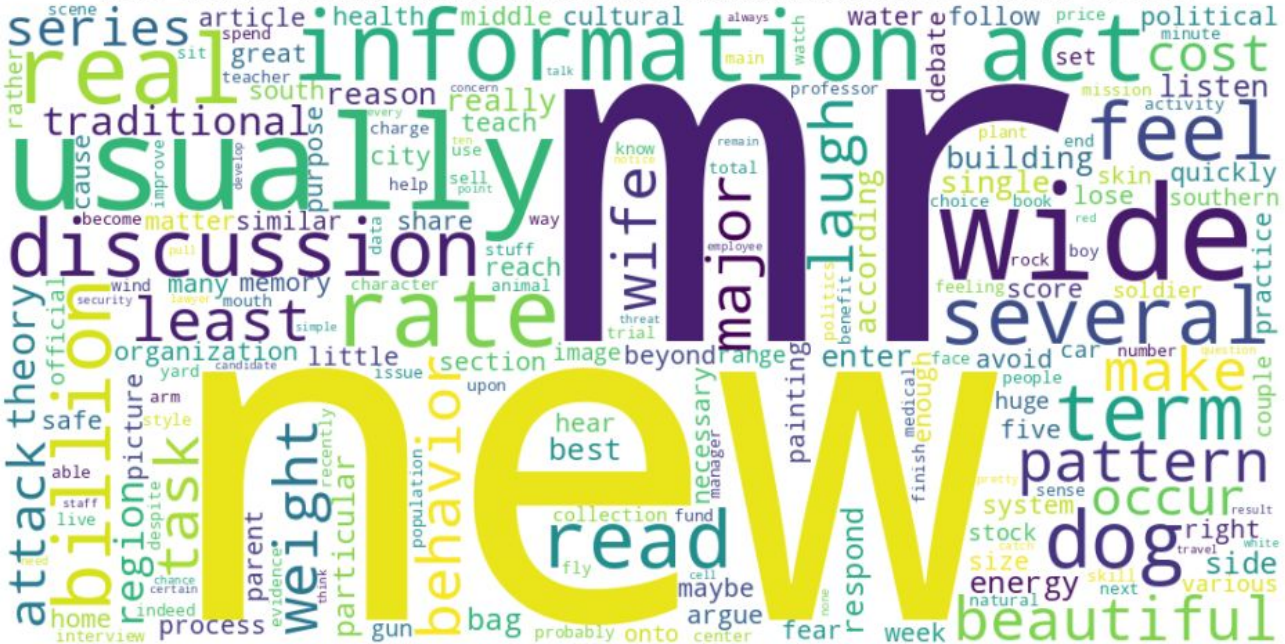
```
1 df4.nunique()
```

	0
Policy_ID	1000
Policy_Text_EN	10
Policy_Text_FR	10
Policy_Text_ES	10
Summarized_Text	10

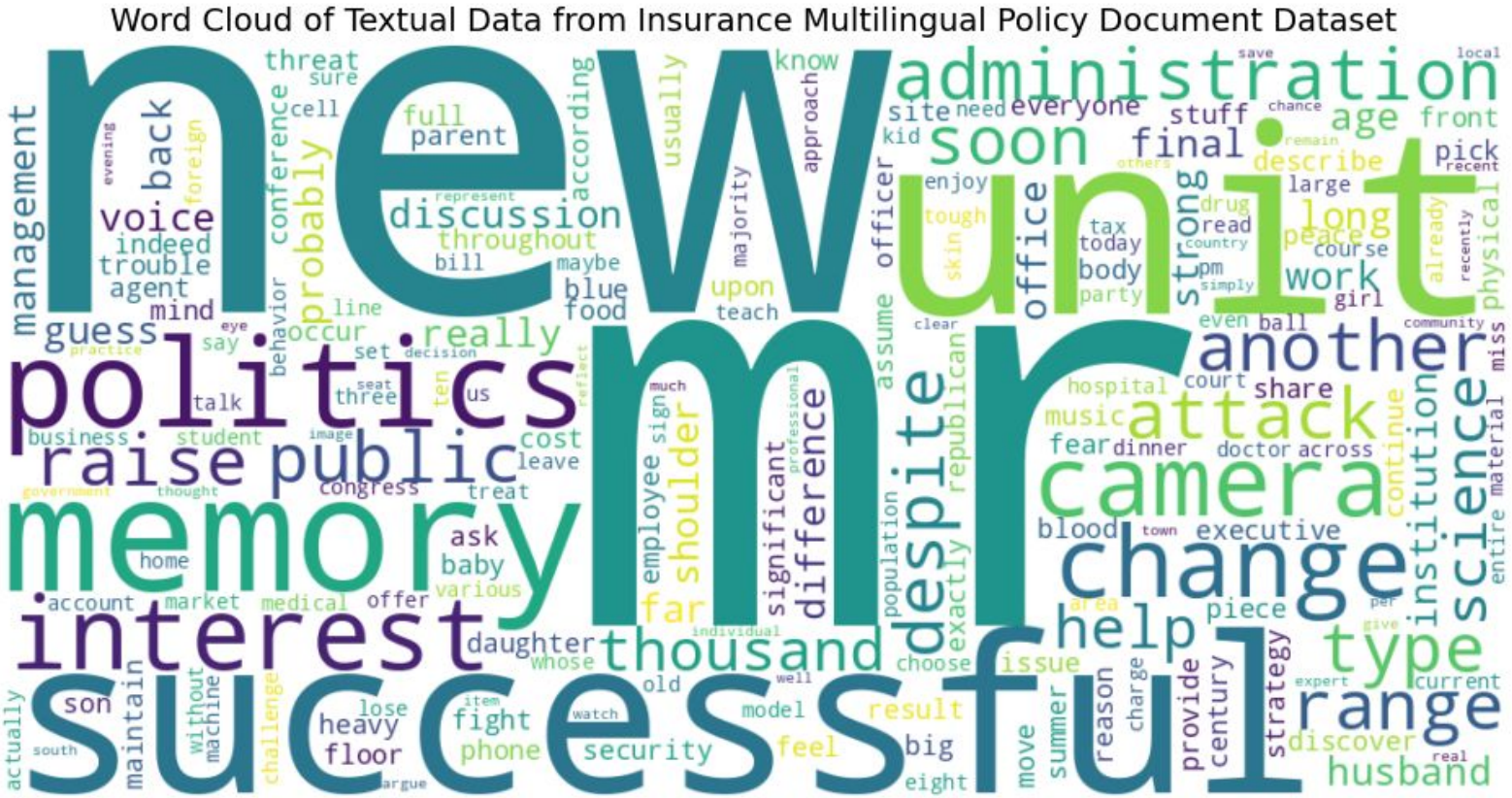
Create wordcloud for TEXT column for better Understanding

1) Policy_Text

Word Cloud of Textual Data from Insurance Multilingual Policy Document Dataset



2) Summarized_Text



NEXT STEP: TEXT Preprocessing

1) Tokenization
2) Stop Word Removal
3) Lowercasing
4) Remove Punctuation
5) Lemmatization
6) Vectorization

Feature Extraction:

1) Bag of Words
2) TF - IDF (TF - Term Frequency, IDF - Inverse Document Frequency)
3) Word Embedding
4) N-Grams
5) POS (Part of Speech)

Preprocessing

```
1 df4.head()
```

	Policy_ID	Policy_Text_EN	Policy_Text_FR	Policy_Text_ES	Summarized_Text
0	bdd640fb-0667-4ad1-9c80-317fa3b1799d	Business insurance protects against liability ...	L'assurance habitation protège contre les domm...	La póliza de salud asegura la cobertura de gas...	Protects home from natural disasters.
1	23b8c1e9-3924-46de-beb1-3b9046685257	Business insurance protects against liability ...	L'assurance pour animaux couvre les frais vété...	Esta póliza cubre los daños a su vehículo caus...	Covers vehicle accident damage.
2	bd9c66b3-ad3c-4d6d-9a3d-1fa7bc8960a9	Vehicle insurance covers theft and vandalism.	L'assurance médicale couvre les frais d'hospit...	El seguro de vida proporciona seguridad financ...	Covers hospitalization costs.
3	972a8469-1641-4f82-8b9d-2434e465e150	Travel insurance covers trip cancellations and...	L'assurance entreprise protège contre les récl...	El seguro médico cubre los gastos de hospitali...	Includes fire damage protection.
4	17fc695a-07a0-4a6e-8822-e8f36c031199	Vehicle insurance covers theft and vandalism.	L'assurance automobile couvre le vol et le van...	El seguro médico cubre los gastos de hospitali...	Covers vehicle accident damage.

1) Text Normalization

1) Convert text to lowercase,

2) remove special characters,

3) expand contractions. - Change the Size of the Word

i) Convert text to lowercase

Policy_Text & Summarized_Text

```
[ ]  1 df4['Policy_Text_EN'] = df4['Policy_Text_EN'].str.lower()
     2 df4['Policy_Text_FR'] = df4['Policy_Text_FR'].str.lower()
     3 df4['Policy_Text_ES'] = df4['Policy_Text_ES'].str.lower()
     4 df4['Summarized_Text'] = df4['Summarized_Text'].str.lower()
```

```
[ ]  1 df4.head()
```

	Policy_ID	Policy_Text_EN	Policy_Text_FR	Policy_Text_ES	Summarized_Text
0	bdd640fb-0667-4ad1-9c80-317fa3b1799d	business insurance protects against liability ...	l'assurance habitation protège contre les domm...	la póliza de salud asegura la cobertura de gas...	protects home from natural disasters.
1	23b8c1e9-3924-46de-beb1-3b9046685257	business insurance protects against liability ...	l'assurance pour animaux couvre les frais vété...	esta póliza cubre los daños a su vehículo caus...	covers vehicle accident damage.
2	bd9c66b3-ad3c-4d6d-9a3d-1fa7bc8960a9	vehicle insurance covers theft and vandalism.	l'assurance médicale couvre les frais d'hospit...	el seguro de vida proporciona seguridad financ...	covers hospitalization costs.

ii) remove special characters

```
[ ]
import re
df4['Policy_Text_EN'] = df4['Policy_Text_EN'].str.lower().apply(lambda x: re.sub(r'[^a-zA-Z0-9\s]', '', x))
df4['Policy_Text_FR'] = df4['Policy_Text_FR'].str.lower().apply(lambda x: re.sub(r'[^a-zA-Z0-9\s]', '', x))
df4['Policy_Text_ES'] = df4['Policy_Text_ES'].str.lower().apply(lambda x: re.sub(r'[^a-zA-Z0-9\s]', '', x))
df4['Summarized_Text'] = df4['Summarized_Text'].str.lower().apply(lambda x: re.sub(r'[^a-zA-Z0-9\s]', '', x))
```

```
[ ]
df4.head()
```

	Policy_ID	Policy_Text_EN	Policy_Text_FR	Policy_Text_ES	Summarized_Text
0	bdd640fb-0667-4ad1-9c80-317fa3b1799d	business insurance protects against liability ...	lassurance habitation protge contre les dommag...	la pliza de salud asegura la cobertura de gast...	protects home from natural disasters
1	23b8c1e9-3924-46de-beb1-3b9046685257	business insurance protects against liability ...	lassurance pour animaux couvre les frais vtrin...	esta pliza cubre los daos a su vehculo causado...	covers vehicle accident damage
2	bd9c66b3-ad3c-4d6d-9a3d-1fa7bc8960a9	vehicle insurance covers theft and vandalism	lassurance mdicale couvre les frais dhospitali...	el seguro de vida proporciona seguridad financ...	covers hospitalization costs
3	972a8469-1641-4f82-8b9d-	travel insurance covers trip	lassurance entreprise protge contre	el seguro mdico cubre los gastos de	includes fire damage

iii) expand contractions.

```
[ ]  1 !pip install contractions
```

```
[ ]  1 import re
     2 import contractions
     3
     4 df4['Policy_Text_EN'] = df4['Policy_Text_EN'].str.lower().apply(lambda x: re.sub(r'[^a-zA-Z0-9\s]', '', contractions.fix(x)))
     5 df4['Policy_Text_FR'] = df4['Policy_Text_FR'].str.lower().apply(lambda x: re.sub(r'[^a-zA-Z0-9\s]', '', contractions.fix(x)))
     6 df4['Policy_Text_ES'] = df4['Policy_Text_ES'].str.lower().apply(lambda x: re.sub(r'[^a-zA-Z0-9\s]', '', contractions.fix(x)))
     7 df4['Summarized_Text'] = df4['Summarized_Text'].str.lower().apply(lambda x: re.sub(r'[^a-zA-Z0-9\s]', '', contractions.fix(x)))
     8
```

```
     1 df4.head()
```

	Policy_ID	Policy_Text_EN	Policy_Text_FR	Policy_Text_ES	Summarized_Text
0	bdd640fb-0667-4ad1-9c80-317fa3b1799d	business insurance protects against liability ...	lassurance habitation protge contre les dommag...	la pliza de salud asegura la cobertura de gast...	protects home from natural disasters
1	23b8c1e9-3924-46de-beb1-3b9046685257	business insurance protects against liability ...	lassurance pour animaux couvre les frais vtrin...	esta pliza cubre los daos a su vehculo causado...	covers vehicle accident damage
2	bd9c66b3-ad3c-4d6d-9a3d-1fa7bc8960a9	vehicle insurance covers theft and vandalism	lassurance mdicale couvre les frais dhospitali...	el seguro de vida proporciona seguridad financ...	covers hospitalization costs
3	972a8469-1641-4f82-8b9d-2434e465e150	travel insurance covers trip cancellations and...	lassurance entreprise protge contre les rclama...	el seguro mdico cubre los gastos de hospitaliz...	includes fire damage protection

expand contractions example:

```
import contractions

text = "I'm going to the store."

expanded_text = contractions.fix(text)

print(expanded_text)  # Output: "I am going to the store."
```

2) Tokenization & Embeddings

i) Tokenization

a) Sentence Tokenization

b) Word Tokenization

c) Letter Tokenization

```
# Step 2: Reinstall a stable version of NLTK
!pip install nltk==3.8.1  # Pin to a stable version
```

```
import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
[ ]
# Tokenizing text
df4['Policy_Text_Tokens_EN'] = df4['Policy_Text_EN'].apply(word_tokenize)
df4['Policy_Text_Tokens_FR'] = df4['Policy_Text_FR'].apply(word_tokenize)
df4['Policy_Text_Tokens_ES'] = df4['Policy_Text_ES'].apply(word_tokenize)
df4['Summarized_Text_Tokens'] = df4['Summarized_Text'].apply(word_tokenize)

```

Policy_Text_Tokens_EN	Policy_Text_Tokens_FR	Policy_Text_Tokens_ES	Summarized_Text_Tokens
[Business, insurance, protects, against, liabi...	[L'assurance, habitation, protège, contre, les...	[La, póliza, de, salud, asegura, la, cobertura...	[Protects, home, from, natural, disasters, .]
[Business, insurance, protects, against, liabi...	[L'assurance, pour, animaux, couvre, les, frai...	[Esta, póliza, cubre, los, daños, a, su, vehíc...	[Covers, vehicle, accident, damage, .]
[Vehicle, insurance, covers, theft, and, vanda...	[L'assurance, médicale, couvre, les, frais, d'...	[El, seguro, de, vida, proporciona, seguridad,...	[Covers, hospitalization, costs, .]
[Travel, insurance, covers, trip, cancellation...	[L'assurance, entreprise, protège, contre, les...	[El, seguro, médico, cubre, los, gastos, de, h...	[Includes, fire, damage, protection, .]

ii) Embeddings

Embeddings Means convert text or image to vectorization

Convert Word to Number to give particular weight certain words

"An embedding is a way to convert words into numerical vectors, capturing their meaning and context. It assigns particular weights based on word usage, relationships, and importance — where meaningful, frequently used, or contextually important words often get more distinct representations."

🧩 **Byte Pair Encoding (BPE)**:

- **How it works**: BPE merges the most frequent pairs of characters or subwords iteratively to form new subwords.
- **Used by**: GPT models, RoBERTa, OpenAI models.
- **Strengths**:
  - Efficient and fast.
  - Works well for language modeling tasks.
  - Handles rare and out-of-vocabulary words well by breaking them down into subword units.
- **Limitations**:
  - Pure frequency-based, so it might not always capture the meaning behind word parts.

🧩 **WordPiece Tokenization**:

- **How it works**: WordPiece uses a similar merging approach but balances frequency and likelihood (how much a subword improves the language model's performance).
- **Used by**: BERT, DistilBERT, mBART, ALBERT.
- **Strengths**:
  - More optimized for model performance.
  - Often results in smaller vocabulary size with good coverage.
  - Better for handling morphologically rich languages.
- **Limitations**:
  - Slightly more complex than BPE.

For your work with BERT embeddings and mBART, **WordPiece Tokenization** is likely the better choice because it's optimized for transformer-based models. It balances efficiency and effectiveness,

using WordPiece Tokenization

```
[ ]  1 from nltk.tokenize import word_tokenize
     2 from transformers import BertTokenizer
```

Bidirectional Encoder Representations from Transformers (BERT)

```
[ ]  1 from transformers import BertTokenizer, BertModel
     2 import torch
     3
     4 # Load multilingual BERT
     5 tokenizer = BertTokenizer.from_pretrained('bert-base-multilingual-cased')
     6 model = BertModel.from_pretrained('bert-base-multilingual-cased')
     7
     8 # Function to get BERT embeddings
     9 def get_embeddings(text):
    10     inputs = tokenizer(text, return_tensors='pt', padding=True, truncation=True)
    11     with torch.no_grad():
    12         outputs = model(**inputs)
    13     return outputs.last_hidden_state.mean(dim=1).squeeze().numpy()
    14
    15 # Apply embeddings for each language and summary
    16 df4['Policy_Text_Embedding_EN'] = df4['Policy_Text_EN'].apply(get_embeddings)
    17 df4['Policy_Text_Embedding_FR'] = df4['Policy_Text_FR'].apply(get_embeddings)
    18 df4['Policy_Text_Embedding_ES'] = df4['Policy_Text_ES'].apply(get_embeddings)
    19 df4['Summarized_Text_Embedding'] = df4['Summarized_Text'].apply(get_embeddings)
    20
    21 # Save the updated DataFrame
    22 df4.to_csv('df4-embeddings-nlp.csv', index=False)
    23
```

```
/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),
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(
tokenizer_config.json: 100%  49.0/49.0 [00:00<00:00, 2.08kB/s]
vocab.txt: 100%  996k/996k [00:00<00:00, 4.24MB/s]
tokenizer.json: 100%  1.96M/1.96M [00:00<00:00, 16.0MB/s]
config.json: 100%  625/625 [00:00<00:00, 13.1kB/s]
model.safetensors: 100%  714M/714M [00:08<00:00, 112MB/s]
Embeddings generated and saved successfully!
```

What's happening here:

- We use `bert-base-multilingual-cased`, which supports multiple languages including English, French, and Spanish.
- `get_embeddings()` tokenizes the text, runs it through the BERT model, and takes the mean of the last hidden state to create a fixed-size embedding.
- We generate embeddings for each policy text language and the summarized text.
- Finally, we save everything in a new CSV.

3) Parallel Corpus Alignment

Use sentence alignment techniques to ensure source and translated texts match correctly

```
[ ]  1 !pip install deep-translator
     2
```

```python
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Function to align sentences based on cosine similarity
def align_sentences(source_texts, target_texts):
    aligned_pairs = []
    for src, tgt in zip(source_texts, target_texts):
        src_emb = get_embeddings(src)
        tgt_emb = get_embeddings(tgt)
        similarity = cosine_similarity([src_emb], [tgt_emb])[0][0]
        aligned_pairs.append((src, tgt, similarity))
    return aligned_pairs

# Aligning English, French, and Spanish texts with their summaries
df4['Aligned_EN'] = df4.apply(lambda row: align_sentences([row['Policy_Text_EN']], [row['Summarized_Text']]), axis=1)
df4['Aligned_FR'] = df4.apply(lambda row: align_sentences([row['Policy_Text_FR']], [row['Summarized_Text']]), axis=1)
df4['Aligned_ES'] = df4.apply(lambda row: align_sentences([row['Policy_Text_ES']], [row['Summarized_Text']]), axis=1)

# Extracting alignment scores
df4['Alignment_Score_EN'] = df4['Aligned_EN'].apply(lambda x: x[0][2])
df4['Alignment_Score_FR'] = df4['Aligned_FR'].apply(lambda x: x[0][2])
df4['Alignment_Score_ES'] = df4['Aligned_ES'].apply(lambda x: x[0][2])

# Save the updated DataFrame
df4.to_csv('df4-aligned-embeddings.csv', index=False)

print("Sentence alignment completed and saved successfully!")

```

```
Sentence alignment completed and saved successfully!
```

Aligned_EN	Aligned_FR	Aligned_ES	Alignment_Score_EN	Alignment_Score_FR	Alignment_Score_ES
[('Business insurance protects against liabili...	[("L'assurance habitation protège contre les d...	[('La póliza de salud asegura la cobertura de ...	0.629268	0.499744	0.429614
[('Business insurance protects against liabili...	[("L'assurance pour animaux couvre les frais v...	[('Esta póliza cubre los daños a su vehículo c...	0.608174	0.320818	0.481441
[('Vehicle insurance covers theft and vandalis...	[("L'assurance médicale couvre les frais d'hos...	[('El seguro de vida proporciona seguridad fin...	0.572594	0.438874	0.426255
[('Travel insurance covers trip cancellations ...	[("L'assurance entreprise protège contre les r...	[('El seguro médico cubre los gastos de hospit...	0.595465	0.392005	0.440118
[('Vehicle insurance covers theft and vandalis...	[("L'assurance automobile couvre le vol et le ...	[('El seguro médico cubre los gastos de hospit...	0.639730	0.495426	0.437731

What this does:

- Uses `get_embeddings()` from the previous step.
- Calculates **cosine similarity** between policy text and summary embeddings.
- Captures the alignment score (0 to 1) — closer to 1 means better alignment.
- Aligns **English (EN)**, **French (FR)**, and **Spanish (ES)** policy texts with their respective summaries.
- Saves everything in the new CSV.

Model Training

df4 - Model Training

Used to train machine translation models for multilingual document translation.

Let's fine-tune the mBART and mT5 models on policy text dataset!

We'll train them for a text summarization task — using the multilingual policy text (Policy_Text_EN, Policy_Text_FR, Policy_Text_ES) as input and the AI-generated Summarized_Text as the target.

```
[ ]  1 pip install transformers datasets sentencepiece accelerate
```

2. Load Data & Preprocess

```python
import pandas as pd
from datasets import Dataset

# Load your aligned dataset
df = pd.read_csv('/content/df4-aligned-embeddings.csv')

# Keep only needed columns
df = df[['Policy_Text_EN', 'Summarized_Text']]

# Convert to Hugging Face Dataset format
dataset = Dataset.from_pandas(df)

```

python

```
dataset = Dataset.from_pandas(df)
```

Copy Edit

- Converts the Pandas DataFrame `df` into a **Hugging Face Dataset**.
- **Why?** Because Hugging Face models like `Trainer` expect datasets in this format.
- This format is more efficient when working with large text datasets and allows for seamless integration with tokenization and model training.

```
1 dataset
```

```
Dataset({
    features: ['Policy_Text_EN', 'Summarized_Text'],
    num_rows: 1000
})
```

```
1 df.head()
```

	Policy_Text_EN	Summarized_Text
0	Business insurance protects against liability ...	Protects home from natural disasters.
1	Business insurance protects against liability ...	Covers vehicle accident damage.
2	Vehicle insurance covers theft and vandalism.	Covers hospitalization costs.
3	Travel insurance covers trip cancellations and...	Includes fire damage protection.
4	Vehicle insurance covers theft and vandalism.	Covers vehicle accident damage.

mBART

3. Tokenizer Setup (for mBART)

```
[ ]
from transformers import MBartTokenizer

model_name = "facebook/mbart-large-50"
tokenizer = MBartTokenizer.from_pretrained(model_name)

# Tokenize the dataset
def preprocess_function(examples):
    model_inputs = tokenizer(examples['Policy_Text_EN'], max_length=512, truncation=True)
    labels = tokenizer(examples['Summarized_Text'], max_length=128, truncation=True)
    model_inputs['labels'] = labels['input_ids']
    return model_inputs

tokenized_dataset = dataset.map(preprocess_function, batched=True)

```

```
The tokenizer class you load from this checkpoint is not the same type as the class this function is called from. It may result in unexpected tokenization.
The tokenizer class you load from this checkpoint is 'MBart50Tokenizer'.
The class this function is called from is 'MBartTokenizer'.
Map: 100% 1000/1000 [00:00<00:00, 2500.86 examples/s]
```

```
from transformers import MBartTokenizer
```

Copy Edit

- `MBartTokenizer`: This is the tokenizer specifically built for mBART — a multilingual sequence-to-sequence model from Hugging Face's `transformers` library.
- Tokenizers convert text into numerical representations that the model can understand — like turning sentences into tokens (numbers).

```
model_name = "facebook/mbart-large-50"
tokenizer = MBartTokenizer.from_pretrained(model_name)
```

Copy Edit

- `model_name`: `"facebook/mbart-large-50"` is the pre-trained mBART model with support for 50 languages.
- `from_pretrained()`: Loads the pre-trained tokenizer associated with this model from the Hugging Face model hub.
- Now, the `tokenizer` knows how to convert words into tokens (numbers) and back.

```
def preprocess_function(examples):
    model_inputs = tokenizer(examples['Policy_Text_EN'], max_length=5
    labels = tokenizer(examples['Summarized_Text'], max_length=128, truncation=True)
    model_inputs['labels'] = labels['input_ids']
    return model_inputs
```

Copy Edit

- `preprocess_function()` : Prepares the data for the model — converting text into tokenized format.
- `examples['Policy_Text_EN']` : The original (input) policy text.
- `examples['Summarized_Text']` : The target text — the summary we want the model to learn to generate.

- `tokenizer()`:
  - `max_length=512`: Caps the input length to 512 tokens to avoid too-long sequences.
  - `truncation=True`: If the text is longer than the max length, it gets truncated to fit.
- `labels['input_ids']`: These are the numerical token IDs representing the summary — set as the **labels** for training.
- Finally, it returns `model_inputs`, now containing:
  - Tokenized policy text.
  - Tokenized summary as labels.

```
tokenized_dataset = dataset.map(preprocess_function, batched=True)
```

Copy Edit

- `map()` : Applies the `preprocess_function()` to every row in the `dataset` .
- `batched=True` : Processes the data in batches for efficiency, instead of one row at a time.
- `tokenized_dataset` : A new dataset where both inputs and labels are tokenized and ready for training.

So essentially, this whole process turns your text into a format the mBART model can use — input IDs and target IDs — so it can learn the mapping from policy text to summarized text

4. Model & Training Setup (mBART)

```
The tokenizer class you load from this checkpoint is not the same type as the class this function is called from. It may result in unexpecte
The tokenizer class you load from this checkpoint is 'MBart50Tokenizer'.
The class this function is called from is 'MBartTokenizer'.
```

Map: 100% 30/30 [00:00<00:00, 390.98 examples/s]

```
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1575: FutureWarning: `evaluation_strategy` is deprecated and will be r
  warnings.warn(
<ipython-input-1-07d80d70cb0c>:57: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use
  trainer = Trainer(
```

[1/1 00:00, Epoch 1/1]

Step	Training Loss

```
import gc
import torch

gc.collect()
torch.cuda.empty_cache()
```

Copy Edit

- `gc.collect()` : Runs Python's garbage collector to clean up unused memory.
- `torch.cuda.empty_cache()` : Clears the GPU memory that's not actively being used — helpful when you've had crashes or memory leaks.

```
cols_needed = ['Policy_Text_EN', 'Summarized_Text']
df = pd.read_csv('/content/synthetic_policy_dataset.csv', usecols=col
df = df.sample(frac=0.03).reset_index(drop=True)
```
Copy Edit

- `usecols=cols_needed` : Only loads the necessary columns to avoid wasting memory.
- `df.sample(frac=0.03)` : Uses only **3% of the data** — a smart move when you keep hitting memory crashes.
- `reset_index(drop=True)` : Resets the index after sampling so the dataset stays clean.

```
small_dataset = Dataset.from_dict({
    'Policy_Text_EN': df['Policy_Text_EN'].tolist(),
    'Summarized_Text': df['Summarized_Text'].tolist()
})
```
Copy Edit

- `Dataset.from_dict()` : Converts your Pandas DataFrame into a Hugging Face `Dataset` format, which is optimized for model training.
- `tolist()` : Converts each column to a list so it's compatible with `Dataset` .

```
model_name = 'facebook/mbart-large-50'
tokenizer = MBartTokenizer.from_pretrained(model_name)
model = MBartForConditionalGeneration.from_pretrained(model_name)
model.gradient_checkpointing_enable()
```

Copy Edit

- `facebook/mbart-large-50` : A **multilingual BART model** fine-tuned for sequence-to-sequence generation across 50 languages.
- `MBartTokenizer.from_pretrained(model_name)` : Loads the tokenizer for this specific model — it knows how to tokenize and decode the text.
- `MBartForConditionalGeneration.from_pretrained(model_name)` : Loads the pre-trained mBART model.
- `model.gradient_checkpointing_enable()` : Saves memory by **not storing intermediate activations** in the forward pass. It recalculates them during backpropagation instead — saving a lot of memory at the cost of some speed.

5. Tokenization function

python

Copy Edit

```python
def preprocess_function(examples):
    inputs = tokenizer(examples['Policy_Text_EN'], padding='max_length', truncation=Tr
    targets = tokenizer(examples['Summarized_Text'], padding='max_length', truncation=
    inputs['labels'] = targets['input_ids']
    return inputs
```

- `Policy_Text_EN` : The input text (policy descriptions).
- `Summarized_Text` : The target (summary).
- `padding='max_length'` : Ensures all sequences are the same length — which speeds up training but wastes some memory on shorter examples. You could try `padding=True` to be more memory-efficient.
- `truncation=True` : Cuts off text longer than the model's max token length.
- `max_length=128` : Limits the policy text length to 128 tokens.
- `max_length=32` : Limits the summary length to 32 tokens.
- `inputs['labels'] = targets['input_ids']` : Adds the tokenized summary as labels — what the model should learn to generate.

6. Tokenize and format dataset

python Copy Edit

```
small_tokenized_dataset = small_dataset.map(preprocess_function, batched=True)
small_tokenized_dataset = small_tokenized_dataset.with_format('torch')
```

- `map(preprocess_function, batched=True)` : Applies the tokenization function to the entire dataset in batches, which speeds up the process.
- `with_format('torch')` : Converts the dataset into **PyTorch tensors**, which is required for model training.

7. Training arguments

python

Copy Edit

```python
training_args = TrainingArguments(
    output_dir='./mbart-finetuned',
    evaluation_strategy='no',          # No evaluation to save memory
    save_strategy='no',                # No saving checkpoints to avoid memory usage
    learning_rate=2e-5,
    per_device_train_batch_size=1,     # Smallest possible batch size
    gradient_accumulation_steps=32,    # Simulates larger batch size without extra mem
    num_train_epochs=1,                # Start with one epoch to avoid long runs
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=100,
    fp16=True,                         # Mixed precision to reduce memory usage
    report_to='none',                  # Disables external logging
    dataloader_num_workers=0           # Avoid multiprocessing to save memory
)
```

- `per_device_train_batch_size=1` : Using a **tiny batch size of 1** reduces memory usage.
- `gradient_accumulation_steps=32` : Simulates a batch size of **1 x 32 = 32** by accumulating gradients over multiple batches before updating model weights.
- `fp16=True` : Mixed-precision training — uses 16-bit floats where possible to cut memory usage in half.
- `dataloader_num_workers=0` : Uses the main process for data loading, avoiding memory overhead from multiple workers.

8. Trainer setup

python Copy Edit

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=small_tokenized_dataset,
    tokenizer=tokenizer
)
```

- `Trainer()` : A high-level Hugging Face class for model training.
- `train_dataset=small_tokenized_dataset` : Uses the prepared dataset for training.

9. Training with error handling

```python
try:
    trainer.train()
except RuntimeError as e:
    print(f"Runtime error: {e}")
```

- `try-except`: Catches `RuntimeError` so if the session crashes, it prints the error and doesn't just break everything.

Why it still might crash

- **mBART is a large model**: Even with these optimizations, it's a heavy model that uses a lot of memory.
- **Batch size of 1 still might not be enough**: If the sequence length is large (like 128 tokens), that takes a lot of memory.
- **Gradient checkpointing saves memory but slows down training**: You're balancing between memory usage and speed.
- **Mixed precision helps but doesn't always solve memory issues**: Especially if you're near the limit.

How to reduce memory usage even more

1. **Try a smaller model**:
   - `sshleifer/distilbart-xsum-12-3` is a distilled version of BART — much smaller and faster.
2. **Reduce `max_length` further**:
   - If you can, lower `max_length=64` for inputs and `max_length=16` for summaries.
3. **Use padding only when needed**:

```python
inputs = tokenizer(examples['Policy_Text_EN'], padding=True, truncation=True, max_
```

4. **Reduce `gradient_accumulation_steps`**:
   If memory still crashes, lower `gradient_accumulation_steps=16` or `8`.
5. **Enable `fp16_full_eval=True`**:

```python
training_args = TrainingArguments(fp16_full_eval=True)
```

6. **Use Colab's High-RAM runtime**:
   In Colab, go to **Runtime > Change runtime type > High-RAM**.

mT5

5. (Optional) Fine-tune mT5

```
[ ]
from transformers import T5Tokenizer, T5ForConditionalGeneration

model_name = "google/mt5-small"
tokenizer = T5Tokenizer.from_pretrained(model_name)
model = T5ForConditionalGeneration.from_pretrained(model_name)

# Preprocess and train similarly, replacing model/tokenizer

```

switching from mBART to mT5, which is a great move because mT5 is also a powerful multilingual text-to-text model. Here's what this code does step by step

1. Importing T5 model and tokenizer

python

Copy Edit

```python
from transformers import T5Tokenizer, T5ForConditionalGeneration
```

- `T5Tokenizer` : The tokenizer for **T5/mT5 models**, responsible for converting text into tokenized format.
- `T5ForConditionalGeneration` : The pre-trained **mT5-small** model, designed for text-to-text tasks like translation and summarization.

2. Loading the model and tokenizer

python Copy Edit

```
model_name = "google/mt5-small"
tokenizer = T5Tokenizer.from_pretrained(model_name)
model = T5ForConditionalGeneration.from_pretrained(model_name)
```

- `google/mt5-small`:
  - This is the **"small"** version of **mT5 (Multilingual T5)**, a **multilingual** variant of T5 that supports over 100 languages.
  - **mT5 is useful for cross-lingual tasks** like machine translation and multilingual summarization.
  - Compared to `facebook/mbart-large-50`, `mt5-small` is **smaller and requires less memory**, making it a good alternative if your training crashes due to memory issues.

- **Tokenizer (** `T5Tokenizer.from_pretrained(model_name)` **)**
  - Loads the pre-trained tokenizer for **mT5-small**.
  - This tokenizer is responsible for:
    - **Tokenizing** input text (converting words into token IDs).
    - **Adding special tokens** (like `<pad>` or `<eos>`).
    - **Detokenizing** model output (converting token IDs back into readable text).
- **Model (** `T5ForConditionalGeneration.from_pretrained(model_name)` **)**
  - Loads the **pre-trained mT5 model**, which is designed for **sequence-to-sequence (seq2seq) tasks**.
  - You can fine-tune this model for tasks like:
    - **Summarization** (policy summarization in your case).
    - **Translation**.
    - **Text generation**.

3. Preprocessing and training

```python
# Preprocess and train similarly, replacing model/tokenizer
```

This means that **you can reuse the same training pipeline** you had with `mBART`, but now use `mT5` instead.

- **Tokenization:**

```python
def preprocess_function(examples):
    inputs = tokenizer(examples['Policy_Text_EN'], padding='max_length', truncatic
    targets = tokenizer(examples['Summarized_Text'], padding='max_length', truncat
    inputs['labels'] = targets['input_ids']
    return inputs
```

  - Tokenizes policy text as input (`Policy_Text_EN`).

- Tokenizes policy text as input (`Policy_Text_EN`).
- Tokenizes summarized text as labels (`Summarized_Text`).

- **Trainer setup:**
  - You can use **Hugging Face's** `Trainer` **class** (like you did with mBART) to fine-tune mT5.
  - **Change only the model and tokenizer** to use `mT5` instead of `mBART` .

Why use `mT5-small` instead of `mBART-large` ?

Feature	mBART-Large-50	mT5-Small
Size	**Large** (~1.4B params)	**Small** (~300M params)
Languages	50	100+
Pretraining task	Denoising autoencoder	Seq2Seq (T5-style)
Memory usage	**High**	**Lower**
Fine-tuning speed	**Slower**	**Faster**

- **mT5-small is much lighter** → Less GPU memory required.
- **mBART is optimized for translation** → If you only need summarization, mT5 is a good alternative.
- **mT5 works better for multilingual summarization** (which might be useful if you expand beyond English).

Now, if you replace `facebook/mbart-large-50` with `google/mt5-small` in your training script, it **should reduce memory usage** and **train more efficiently**.

Save Model

mBART Model Save

```
model.save_pretrained("./mbart-finetuned")
tokenizer.save_pretrained("./mbart-finetuned")
```

mT5 Model Save

```
# Save the fine-tuned mT5 model and tokenizer
model.save_pretrained("./mt5-finetuned")
tokenizer.save_pretrained("./mt5-finetuned")
```

Load Model & Prediction

mT5

```python
from transformers import T5ForConditionalGeneration, T5Tokenizer

model_path = "./mt5-finetuned"  # Change this to your mT5 save path

# Load the saved model and tokenizer
tokenizer = T5Tokenizer.from_pretrained(model_path)
model = T5ForConditionalGeneration.from_pretrained(model_path)

# Example inference
text = "This policy covers damage to your vehicle caused by accidents"
inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True, max_length=512)

# Generate summary
output = model.generate(**inputs)
summary = tokenizer.decode(output[0], skip_special_tokens=True)

print(summary)
```

Output:

<extra_id_0>

Add Some weight on Output:

```
text = "This policy covers damage to your vehicle caused by accidents"
inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True, max_length=51
```

```
output = model.generate(
    **inputs,
    max_length=128,
    num_beams=5,            # Beam search for better quality
    early_stopping=True,    # Stop when a complete sentence is formed
    no_repeat_ngram_size=2 # Avoid repeating phrases
)

summary = tokenizer.decode(output[0], skip_special_tokens=True)
print(summary)
```

Check Weight added or not

```
[ ]
```

```python
import torch

# Check the original model
original_model = T5ForConditionalGeneration.from_pretrained("google/mt5-small")
original_weights = original_model.state_dict()

# Check the fine-tuned model
finetuned_weights = model.state_dict()

# Compare a few layers to see if they changed
for name, param in original_weights.items():
    if not torch.equal(param, finetuned_weights[name]):
        print(f"Weights changed for: {name}")
        break
else:
    print("No weights were updated — model may not have been fine-tuned properly.")
```

Weight not updated so again tune Model

```python
# Convert to Hugging Face Dataset
small_dataset = Dataset.from_dict({
    'Policy_Text_EN': df['Policy_Text_EN'].tolist(),
    'Summarized_Text': df['Summarized_Text'].tolist()
})

# Load tokenizer and model
model_name = 'google/mt5-small'
tokenizer = T5Tokenizer.from_pretrained(model_name)
model = T5ForConditionalGeneration.from_pretrained(model_name)

# Tokenization function
def preprocess_function(examples):
    inputs = tokenizer(examples['Policy_Text_EN'], padding='max_length', truncation=True, max_length=512)
    targets = tokenizer(examples['Summarized_Text'], padding='max_length', truncation=True, max_length=128)
    inputs['labels'] = targets['input_ids']
    return inputs

# Tokenize and format dataset
small_tokenized_dataset = small_dataset.map(preprocess_function, batched=True)
small_tokenized_dataset = small_tokenized_dataset.with_format('torch')

```

```python
# Training arguments
training_args = TrainingArguments(
    output_dir='./mt5-finetuned',
    evaluation_strategy='no',
    save_strategy='steps',
    save_steps=500,
    learning_rate=2e-5,
    per_device_train_batch_size=1,
    gradient_accumulation_steps=16,
    num_train_epochs=1,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=100,
    fp16=True,
    report_to='none',
    dataloader_num_workers=0,
)

# Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=small_tokenized_dataset,
    tokenizer=tokenizer
)
```

```python

# Train
try:
    trainer.train()
    model.save_pretrained('./mt5-finetuned')
    tokenizer.save_pretrained('./mt5-finetuned')
except RuntimeError as e:
    print(f"Runtime error: {e}")

```

Check weight again

```python
import torch

# Check the original model
original_model = T5ForConditionalGeneration.from_pretrained("google/mt5-small")
original_weights = original_model.state_dict()

# Check the fine-tuned model
finetuned_weights = model.state_dict()

# Compare a few layers to see if they changed
for name, param in original_weights.items():
    if not torch.equal(param, finetuned_weights[name]):
        print(f"Weights changed for: {name}")
        break
else:
    print("No weights were updated — model may not have been fine-tuned properly.")
```

Weight Changed Successfully

Save model again

```
[ ]  1 model.save_pretrained("./mt5-finetuned")
     2 tokenizer.save_pretrained("./mt5-finetuned")
```

weight updated successfully

Weights changed for: shared.weight

```
[ ]
```

```python
from transformers import T5ForConditionalGeneration, T5Tokenizer

model_path = "./mt5-finetuned"
tokenizer = T5Tokenizer.from_pretrained(model_path)
model = T5ForConditionalGeneration.from_pretrained(model_path)

# Properly formatted input with task prefix
simple_text = "summarize: This is a simple insurance policy."
inputs = tokenizer(simple_text, return_tensors="pt", padding=True, truncation=True, max_length=512)

# Generate summary
output = model.generate(**inputs)
summary = tokenizer.decode(output[0], skip_special_tokens=True)

print(summary)
```

```
<extra_id_0>
```

```
[ ]
```

```python
output = model.generate(**inputs, max_length=100, num_beams=5, early_stopping=True)
summary = tokenizer.decode(output[0], skip_special_tokens=True)

print(summary)
```

```
<extra_id_0>
```

Still Issues Not Clear - So Move to mBART

mBART File Size Very High - 2.5GB

mBART

```python
from safetensors.torch import safe_open

model_path = f"{model_name}/model.safetensors"

with safe_open(model_path, framework="pt", device="cpu") as f:
    print(f.keys())
```

```
['final_logits_bias', 'model.decoder.embed_positions.weight', 'model.decoder.layer_norm.bias', 'model.decoder.layer_norm.weight', 'model.decoder.layernorm_emb
```

```python
import os

model_path = f"{model_name}/model.safetensors"
print(f"File exists: {os.path.exists(model_path)}")
print(f"File size: {os.path.getsize(model_path) / (1024 * 1024):.2f} MB")
```

```
File exists: True
File size: 2331.23 MB
```

```
File exists: True
File size: 2331.23 MB
```

```python
from safetensors.torch import load_file

state_dict = load_file(model_path)
print(f"Number of keys: {len(state_dict)}")
print(f"First few keys: {list(state_dict.keys())[:5]}")
print(f"Shape of one weight: {state_dict[list(state_dict.keys())[0]].shape}")
```

```
Number of keys: 516
First few keys: ['final_logits_bias', 'model.decoder.embed_positions.weight', 'model.decoder.layer_norm.bias', 'model.decoder.layer_norm.weight', 'model.deco
Shape of one weight: torch.Size([1, 250027])
```

```python
from transformers import MBartForConditionalGeneration, MBartConfig

# Load the config from the original model
config = MBartConfig.from_pretrained(model_name)
print(config)
```

```
MBartConfig {
  "_name_or_path": "facebook/mbart-large-cc25",
  "_num_labels": 3,
  "activation_dropout": 0.0,
  "activation_function": "gelu",
  "add_bias_logits": false,
  "add_final_layer_norm": true,
  "architectures": [
```

```
[ ]
```

```python
from transformers import MBartForConditionalGeneration, MBartTokenizer

model_name = "/content/mbart-finetuned"

# Load config and tokenizer
try:
    tokenizer = MBartTokenizer.from_pretrained(model_name)
    print("Tokenizer loaded successfully")
except Exception as e:
    print(f"Error loading tokenizer: {e}")

try:
    model = MBartForConditionalGeneration.from_pretrained(
        model_name,
        state_dict=state_dict
    )
    print("Model loaded successfully")
except Exception as e:
    print(f"Error loading model: {e}")

```

```
Tokenizer loaded successfully
Model loaded successfully
```

```python
from transformers import MBartForConditionalGeneration, MBartTokenizer

model_name = "/content/mbart-finetuned"

# Load the original mBART tokenizer and model
model = MBartForConditionalGeneration.from_pretrained("facebook/mbart-large-cc25")
tokenizer = MBartTokenizer.from_pretrained("facebook/mbart-large-cc25")

# Save the correct mBART tokenizer and config
model.save_pretrained(model_name)
tokenizer.save_pretrained(model_name)

```

```
/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), s
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(
/usr/local/lib/python3.11/dist-packages/transformers/modeling_utils.py:2758: UserWarning: Moving the following attributes
  warnings.warn(
('/content/mbart-finetuned/tokenizer_config.json',
 '/content/mbart-finetuned/special_tokens_map.json',
 '/content/mbart-finetuned/sentencepiece.bpe.model',
 '/content/mbart-finetuned/added_tokens.json')
```

```
[ ]
```

```python
from safetensors.torch import load_file
state_dict = load_file(f"{model_name}/model.safetensors")

model = MBartForConditionalGeneration.from_pretrained(
    model_name,
    state_dict=state_dict
)

print("Model loaded successfully!")
```

```
Model loaded successfully!
```

Prediction

```python
from transformers import MBartForConditionalGeneration, MBartTokenizer

model_name = "/content/mbart-finetuned"

# Load the model and tokenizer
model = MBartForConditionalGeneration.from_pretrained(model_name)
tokenizer = MBartTokenizer.from_pretrained(model_name)

# Example input text
input_text = "This policy covers damage to your vehicle caused by accidents"

# Tokenize and prepare input
inputs = tokenizer(input_text, return_tensors="pt")

# Generate prediction
outputs = model.generate(**inputs, max_length=100, num_beams=5, early_stopping=True)

# Decode the generated output
predicted_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
print("Predicted text:", predicted_text)

```

```
Predicted text: policy cover cover cover cover cover cover covers damage to policy covers damage to policy covers da
```

Streamlit

ChatBot

Text Summarization with Facebook's mBART

Enter the text you want to summarize:

Life insurance provides financial security for beneficiaries

Summarize

Summary:

Life insurance provides financial security for beneficiaries _ Life insurance provides financial security for beneficiaries _ Life insurance provides financial security for beneficiaries

Prediction Failure

Again Check Real Time Dataset

Article Text to Headline Summarizing Real time Dataset from Kaggle

df4 - Preprocessing

For Multilingual Insurance Policy Dataset

Using Real Time Dataset

Daily Mail Summarization

```
[ ]  1 import kagglehub
     2
     3 # Download latest version
     4 path = kagglehub.dataset_download("sunnysai12345/news-summary")
     5
     6 print("Path to dataset files:", path)
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/sunnysai12345/news-summary?dataset_version_number=2...
100%|██████████| 19.8M/19.8M [00:00<00:00, 62.5MB/s]Extracting files...

Path to dataset files: /root/.cache/kagglehub/datasets/sunnysai12345/news-summary/versions/2
```

```
[ ]  1 df4_new = pd.read_csv(path + '/news_summary_more.csv', encoding='latin-1')
     2 df4_new.head()
```

	headlines	text
0	upGrad learner switches to career in ML & Al w...	Saurav Kant, an alumnus of upGrad and IIIT-B's...
1	Delhi techie wins free food from Swiggy for on...	Kunal Shah's credit card bill payment platform...
2	New Zealand end Rohit Sharma-led India's 12-ma...	New Zealand defeated India by 8 wickets in the...
3	Aegon life iTerm insurance plan helps customer...	With Aegon Life iTerm Insurance plan, customer...
4	Have known Hirani for yrs, what if MeToo claim...	Speaking about the sexual harassment allegatio...

Check Missing Values

```
df4_new.isnull().sum()  # Check missing values
```

	0
headlines	0
text	0

dtype: int64

Check Duplicates

```
print(df4_new.duplicated().sum())  # Count fully duplicated rows
print(df4_new.duplicated(subset=['text', 'headlines']).sum())  # Count duplicate article-summary pairs
```

```
22
22
```

```
df4_new.shape
```

```
(98401, 2)
```

Very Low Duplicated Compare to 98k, 22 Duplicates is very Low - so i delete it

. Remove Duplicates and Missing Values

```
[ ]  1 df4_new = df4_new.drop_duplicates(subset=['text', 'headlines']).dropna(subset=['text', 'headlines'])
     2 print(f"Rows after cleaning: {len(df4_new)}")
     3
```

```
Rows after cleaning: 98379
```

My Dataset has 98379 Unique its May Affect my Time so i Reduce Dataset

Random Sampling - 98000 to 30000 reduced

```
[ ]
# Reduce dataset to 30,000 rows
df4_new = df4_new.sample(n=30000, random_state=42)

df4_new.to_csv("df4_new_reduced_dataset.csv", index=False)
print(f"Reduced dataset size: {len(df4_new)}")

```

```
Reduced dataset size: 30000
```

```
df4_random = pd.read_csv('/content/df4_new_reduced_dataset.csv')
df4_random.head()
```

	headlines	text
0	Ex-dentist's new mobile app gets $40 mn after ...	Former South Korean dentist Seunggun Lee's mob...
1	LG tests robots to guide passengers in airport...	LG is testing two robot prototypes to provide ...
2	Opposition is trying to kill me: Delhi CM on c...	Reacting to the chilli powder attack on him on...
3	Policeman touches UP Minister's feet for scrap...	A policeman in Uttar Pradesh's Kanpur touched ...
4	I'd love to make Mahabharat film but don't hav...	Actor Shah Rukh Khan has said that he would lo...

```
df4_random.shape
```

```
(30000, 2)
```

1) Text Preprocesssing

```
[ ]    1 df4_random.columns
```

```
Index(['headlines', 'text'], dtype='object')
```

1️⃣ Load & Inspect Data

check for missing values & duplicates

```
[ ]    1 # already Checked
```

2️⃣ Normalize & Clean Text

```
[ ]  1 # To improve model input, apply basic text cleaning:
```

```
[ ]  1 import re
     2
     3 def clean_text(text):
     4     text = text.lower()  # Convert to lowercase
     5     text = re.sub(r"\s+", " ", text)  # Remove extra spaces
     6     text = re.sub(r"http\S+|www\S+", "", text)  # Remove URLs
     7     text = re.sub(r"[^a-zA-Z0-9.,!?]", " ", text)  # Remove special characters (except punct
     8     return text.strip()
     9
    10 df4_random['text'] = df4_random['text'].apply(clean_text)
    11 df4_random['headlines'] = df4_random['headlines'].apply(clean_text)
    12
```

	headlines	text
0	ex dentist s new mobile app gets 40 mn after ...	former south korean dentist seunggun lee s mob...
1	lg tests robots to guide passengers in airport...	lg is testing two robot prototypes to provide ...
2	opposition is trying to kill me delhi cm on c...	reacting to the chilli powder attack on him on...
3	policeman touches up minister s feet for scrap...	a policeman in uttar pradesh s kanpur touched ...
4	i d love to make mahabharat film but don t hav...	actor shah rukh khan has said that he would lo...
...	...	...
29995	who is krishnamurthy subramanian, new chief ec...	india s new chief economic adviser krishnamurt...
29996	air india flight lands at delhi without passen...	an air india flight from dharamshala arrived a...
29997	alcoholic man arrested for rape of 13 yr old d...	uttar pradesh police have arrested a man from ...
29998	akshay, srk slam terror attack on amarnath yat...	akshay kumar, shah rukh khan, anupam kher and ...
29999	india will be open defecation free by oct 2019...	launching the swachhta hi sewa cl...

30000 rows × 2 columns

Removes noise while keeping important text

1) Lowerization

2) Remove Extra Spaces

3) Remove URLs

4) Remove Special Characters

```
[ ]    1 df4_random.to_csv('df4_preprocessed.csv')
```

❌ **No need for manual BPE/WordPiece tokenization**

✅ **Hugging Face tokenizers already handle subword encoding**

✅ **Just call `AutoTokenizer.from_pretrained()` to tokenize correctly**

No, you **don't need to manually apply BPE or WordPiece tokenization** before tokenizing with mBART/mT5. **Why?**

Both **mBART and mT5** already use **subword tokenization** internally:

- **mBART** uses **SentencePiece (BPE-based tokenization)**
- **mT5** also uses **SentencePiece tokenization**

👉 When you call the Hugging Face tokenizer (`AutoTokenizer.from_pretrained`), it **automatically applies subword tokenization** using **BPE or WordPiece**, depending on the model.

```
df4_pre.head()
```

	Unnamed: 0	headlines	text
0	0	ex dentist s new mobile app gets 40 mn after ...	former south korean dentist seunggun lee s mob...
1	1	lg tests robots to guide passengers in airport...	lg is testing two robot prototypes to provide ...
2	2	opposition is trying to kill me delhi cm on c...	reacting to the chilli powder attack on him on...
3	3	policeman touches up minister s feet for scrap...	a policeman in uttar pradesh s kanpur touched ...
4	4	i d love to make mahabharat film but don t hav...	actor shah rukh khan has said that he would lo...

1️⃣ First, tokenize data using `AutoTokenizer`

2️⃣ Then, load `MBartForConditionalGeneration` or `MT5ForConditionalGeneration` for training or inference

3️⃣ Tokenization (For mBART/mT5)

Convert text into tokenized format using Hugging Face tokenizer:

```
[ ]
from transformers import AutoTokenizer

# Load tokenizer
tokenizer = AutoTokenizer.from_pretrained("facebook/mbart-large-cc25")

# Define max sequence length
max_input_length = 512
max_target_length = 150

# Tokenize inputs & outputs
df4_pre['text_tokenized'] = df4_pre['text'].apply(lambda x: tokenizer.encode(x, truncation=True, padding="max_length", max_len
df4_pre['headlines_tokenized'] = df4_pre['headlines'].apply(lambda x: tokenizer.encode(x, truncation=True, padding="max_length
```

headlines	text	text_tokenized	headlines_tokenized
ex dentist s new mobile app gets 40 mn after ...	former south korean dentist seunggun lee s mob...	[36770, 127067, 20867, 66, 151250, 40, 1619, 6...	[1119, 151250, 91, 3525, 14288, 4027, 62163, 1...
lg tests robots to guide passengers in airport...	lg is testing two robot prototypes to provide ...	[96, 177, 83, 134234, 6626, 11329, 160469, 90,...	[96, 177, 109921, 11329, 7, 47, 17997, 44828, ...
opposition is trying to kill me delhi cm on c...	reacting to the chilli powder attack on him on...	[131300, 214, 47, 70, 149031, 14, 173169, 5287...	[177986, 83, 31577, 47, 67153, 163, 146, 979, ...
policeman touches up minister s feet for scrap...	a policeman in uttar pradesh s kanpur touched ...	[10, 35206, 669, 23, 486, 867, 6, 221313, 91, ...	[35206, 669, 116281, 7, 1257, 24284, 91, 74261...
i d love to make mahabharat film but don t hav...	actor shah rukh khan has said that he would lo...	[39329, 90497, 79758, 127, 70144, 1556, 2804, ...	[17, 104, 5161, 47, 3249, 13490, 42927, 257, 1...

- Converts raw text into **token IDs**, which the model understar
- Ensures text fits within the model's input size.
- Necessary before training or inference.

fine tuning no need tokenized data but prediction time they compare the input to auto tokenized data to give perfect result

✅ **Use raw text (`text` and `headlines`) for fine-tuning.**
✅ **Let the tokenizer handle tokenization inside `map()` dynamically.**
❌ **Do NOT use `text_tokenized` or `headlines_tokenized` directly in fine-tuning.**

Inference (Prediction Phase)

- When making predictions, the input text is **tokenized on the fly**.
- The model compares this tokenized input with what it learned during training.
- This ensures it produces a **perfectly formatted** output.

So yes, during inference, **raw input is tokenized dynamically** to match the training conditions, leading to accurate predictions! ✅

Fine Tuning for mBART

```
[ ]    1 !pip install transformers datasets sentencepiece accelerate
```

Colab session crashed due to running out of RAM. This happens because mBART is a large model and requires a lot of memory for training

so Reducing Batchsize 8 to 2

Colab session crashed due to running out of RAM while fine-tuning mBART. This happens because mBART is a large model, and your dataset (30,000 rows) is consuming too much memory.

1] Reduce Batch Size - 2 to 1

per_device_train_batch_size=1, per_device_eval_batch_size=1,

2] Reduce Dataset Size - If 30,000 rows are too much, try reducing to 10,000 rows:

dataset = dataset.train_test_split(test_size=0.1, train_size=10000)

3] Use Mixed Precision (FP16) - Enable half-precision training to cut memory usage

```
training_args = Seq2SeqTrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    per_device_train_batch_size=1,
    per_device_eval_batch_size=1,
    num_train_epochs=3,
    save_total_limit=2,
    load_best_model_at_end=True,
    fp16=True  # ✅ Enable mixed precision
)
```

5️⃣ Enable Gradient Checkpointing (Last Resort)

If crashes continue, add:

```python
model.gradient_checkpointing_enable()
```

This **saves GPU memory** but slows down training.

```
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
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(
```

Map: 100%  30000/30000 [00:24<00:00, 1227.63 examples/s]

```
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1575: FutureWarning: `evaluation_strategy` is deprecated and will be r
  warnings.warn(
<ipython-input-2-93957db25e84>:40: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Seq2SeqTrainer.__init_
  trainer = Seq2SeqTrainer(
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please specify
wandb: Using wandb-core as the SDK backend.  Please refer to https://wandb.me/wandb-core for more information.
wandb: Currently logged in as: antonychackotc (antonychackotc-guvi-geek-networks) to https://api.wandb.ai. Use `wandb login --relogin` to fo
Tracking run with wandb version 0.19.8
Run data is saved locally in /content/wandb/run-20250319_073003-0dw1ejzb
Syncing run ./results to Weights & Biases (docs)
View project at https://wandb.ai/antonychackotc-guvi-geek-networks/huggingface
View run at https://wandb.ai/antonychackotc-guvi-geek-networks/huggingface/runs/0dw1ejzb
```

Colab Session Crashed : RAM

1. Switch from mBART to mT5-Small
2. Use FP16 mixed precision
3. Reduce dataset size (Try 5000 rows instead of 10,000)
4. Use `gradient_checkpointing_enable()`
5. Enable gradient accumulation to simulate larger batch sizes with a small memory footprint

Session Crashed now am trying to mT5

Optimized Training Code for mT5-Small

```
[ ]
from transformers import MT5ForConditionalGeneration, AutoTokenizer, Seq2SeqTrainer, Seq2SeqTrainingArguments
from datasets import Dataset
import pandas as pd

# ✅ Switch to mT5-Small
model_name = "google/mt5-small"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = MT5ForConditionalGeneration.from_pretrained(model_name)
model.gradient_checkpointing_enable()  # ✅ Reduce memory usage

```

```

# Tokenize function
def tokenize_function(examples):
    model_inputs = tokenizer(examples["text"], truncation=True, padding="max_length", max_length=256)
    labels = tokenizer(examples["headlines"], truncation=True, padding="max_length", max_length=100)
    model_inputs["labels"] = labels["input_ids"]
    return model_inputs

df4_pre = pd.read_csv('/content/df4_preprocessed.csv')

# ✅ Reduce dataset size for Colab (Try 5000 rows)
df4_pre = df4_pre.sample(n=1000, random_state=42)  # Reduce dataset size
```

```python
# Convert Pandas DataFrame to Dataset
dataset = Dataset.from_pandas(df4_pre)
dataset = dataset.map(tokenize_function, batched=True)

# Split into train/test
dataset = dataset.train_test_split(test_size=0.1)

```

```python
# ✅ Training arguments optimized for Colab
training_args = Seq2SeqTrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    per_device_train_batch_size=1,  # ✅ Reduce batch size
    per_device_eval_batch_size=1,
    gradient_accumulation_steps=8,  # ✅ Simulates batch size of 8
    num_train_epochs=1,
    save_total_limit=2,
    load_best_model_at_end=True,
    fp16=True,  # ✅ Mixed precision training
    learning_rate=2e-5,  # ✅ Add learning rate
    weight_decay=0.01,   # ✅ Add weight decay
)

```

```python

# Define trainer
trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset=dataset["train"],
    eval_dataset=dataset["test"],
    tokenizer=tokenizer
)

# 🚀 Start fine-tuning
trainer.train()

```

```
/usr/local/lib/python3.11/dist-packages/transformers/convert_slow_tokenizer.py:561: UserWarning: The sentencepiece tokenizer that yo
  warnings.warn(
Map: 100% 1000/1000 [00:00<00:00, 1051.94 examples/s]
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1575: FutureWarning: `evaluation_strategy` is deprecated and w
  warnings.warn(
<ipython-input-2-d354c49be7cb>:47: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Seq2SeqTrainer
  trainer = Seq2SeqTrainer(
[112/112 2:13:26, Epoch 0/1]
```

Epoch	Training Loss	Validation Loss
0	No log	38.238556

```
Passing a tuple of `past_key_values` is deprecated and will be removed in Transformers v4.48.0. You should pass an instance of `Enco
There were missing keys in the checkpoint model loaded: ['encoder.embed_tokens.weight', 'decoder.embed_tokens.weight'].
TrainOutput(global_step=112, training_loss=48.09848894391741, metrics={'train_runtime': 8093.0665, 'train_samples_per_second': 0.111
'train_steps_per_second': 0.014, 'total_flos': 236879861514240.0, 'train_loss': 48.09848894391741, 'epoch': 0.9955555555555555})
```

Now Predicted Successfully used of 1000 Rows out of overall 90000 Rows

🚨 **High Training Loss**

- **Training Loss** = **48.09**
- **Validation Loss** = **38.23**
- This suggests that:
  - The model **struggled to learn** during training.
  - Consider **longer training (more epochs)** or **tuning the learning rate**.

📉 **Low** `train_samples_per_second`

- `train_samples_per_second = 0.111`, which is very **slow**.

Warning 3: Missing model weights (`embed_tokens.weight`)

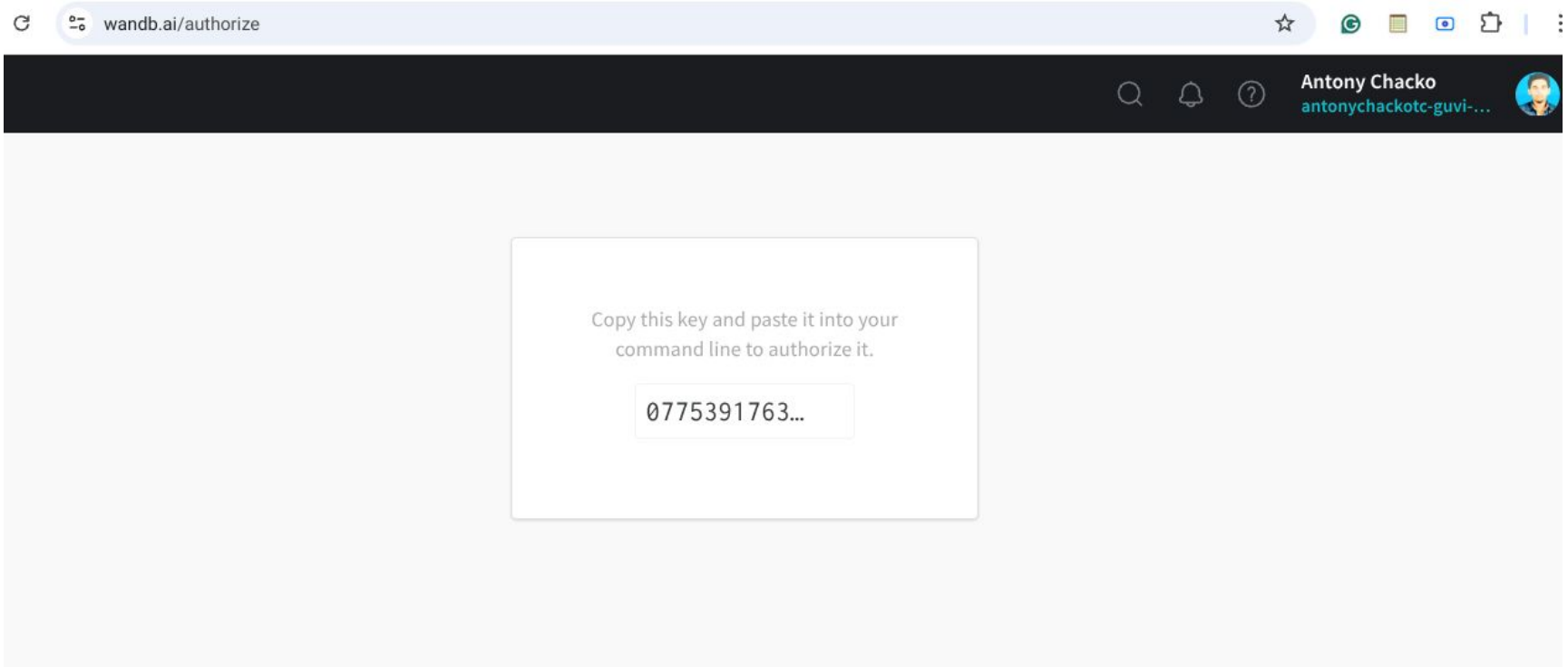
- **Issue**: Some weight parameters were not found in the checkpoint.
- **Possible Causes & Fixes**:
  - If this was the **first fine-tuning run**, it's safe to **ignore**.
  - If you loaded a **previously fine-tuned checkpoint**, check if it was saved correctly.
  - If using a **custom tokenizer**, ensure it was trained properly.

So Again Do Fine Tunning Model Not Saved:-

Now Model Saved Option Directly from Drive

```
ndb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
ndb: Currently logged in as: antonychackotc (antonychackotc-guvi-geek-networks) to https://api.wandb.ai. Use `wandb login --relogin` to f
acking run with wandb version 0.19.8
```

Create wandb.ai tokens for mT5

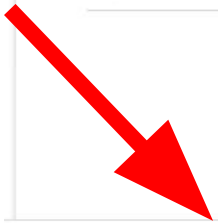


mT5 Not Save Properly

```
53 )
54
55 # 🚀 Start fine-tuning
56 trainer.train()
57
```

```
/usr/local/lib/python3.11/dist-packages/transformers/convert_slow_tokenizer.py:561: UserWarning: The sentencepiece tokenizer that you
  warnings.warn(
Map: 100% 1000/1000 [00:00<00:00, 1051.94 examples/s]
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1575: FutureWarning: `evaluation_strategy` is deprecated and wi
  warnings.warn(
<ipython-input-2-d354c49be7cb>:47: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Seq2SeqTrainer.
  trainer = Seq2SeqTrainer(
[112/112 1:58:34, Epoch 1.00/1]
```

Epoch	Training Loss	Validation Loss



Executing (2h 2m 22s) <cell line: 0> › train() › _inner_training_loop() › _maybe_log_save_evaluate() › _save_checkpoint() › _save_optimizer_and_scheduler() › save() › _save()

So Now Try again mBART-Large-50

```python
import torch
import psutil

print("🚀 GPU Available:", torch.cuda.is_available())
if torch.cuda.is_available():
    print("🔥 GPU Name:", torch.cuda.get_device_name(0))
    print("🖥️ VRAM:", torch.cuda.get_device_properties(0).total_memory / 1e9, "GB")

print("💾 RAM Available:", psutil.virtual_memory().available / 1e9, "GB")

```

```
🚀 GPU Available: False
💾 RAM Available: 11.502424064 GB
```

```
Using device: cpu
/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
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(
The tokenizer class you load from this checkpoint is not the same type as the class this function is called from. It may result in
The tokenizer class you load from this checkpoint is 'MBart50Tokenizer'.
The class this function is called from is 'MBartTokenizer'.
```

Map: 100% 500/500 [00:00<00:00, 682.87 examples/s]

```
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1575: FutureWarning: `evaluation_strategy` is deprecated and
  warnings.warn(
<ipython-input-1-e8f7d4b95bed>:64: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__ini
  trainer = Trainer(
```

[2/62 : < :, Epoch 0.02/1]

Step	Training Loss

Session Crashed using **mBART-Large-50** so am Using **BART-base**

```
🚀 Using device: cpu
/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 Cola
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(
vocab.json: 100%  899k/899k [00:00<00:00, 6.10MB/s]
merges.txt: 100%  456k/456k [00:00<00:00, 15.3MB/s]
tokenizer.json: 100%  1.36M/1.36M [00:00<00:00, 25.1MB/s]
config.json: 100%  1.72k/1.72k [00:00<00:00, 96.7kB/s]
model.safetensors: 100%  558M/558M [00:06<00:00, 195MB/s]
Map: 100%  1000/1000 [00:01<00:00, 743.09 examples/s]
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1575: FutureWarning: `evaluation_strategy` is deprecated and will be removed in versior
  warnings.warn(
<ipython-input-3-44cd9c85bb59>:61: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_clas
  trainer = Trainer(
[62/62 16:38, Epoch 0/1]
 Step  Training Loss
```

```
[ ]  1 model.save_pretrained("/content/drive/MyDrive/BART_finetuned_model")
     2 tokenizer.save_pretrained("/content/drive/MyDrive/BART_finetuned_model")
```

```
/usr/local/lib/python3.11/dist-packages/transformers/modeling_utils.py:2758: UserWarning: Moving the following attributes in the config to the generation
  warnings.warn(
('/content/drive/MyDrive/BART_finetuned_model/tokenizer_config.json',
 '/content/drive/MyDrive/BART_finetuned_model/special_tokens_map.json',
 '/content/drive/MyDrive/BART_finetuned_model/vocab.json',
 '/content/drive/MyDrive/BART_finetuned_model/merges.txt',
 '/content/drive/MyDrive/BART_finetuned_model/added_tokens.json')
```

```
Installing collected packages: evaluate
Successfully installed evaluate-0.4.3
```

Downloading builder script: 100%  6.27k/6.27k [00:00<00:00, 155kB/s]

```
rouge1: 0.4516
rouge2: 0.2385
rougeL: 0.4138
rougeLsum: 0.4133
```

- **ROUGE-1 (0.4516)**: This means 45.16% of the unigrams (single words) in the generated text overlap with the reference text. This is a decent score.
- **ROUGE-2 (0.2385)**: About 23.85% of bigrams (two-word sequences) match the reference. This suggests that your model captures some phrase-level coherence but has room for improvement.
- **ROUGE-L (0.4138) & ROUGE-Lsum (0.4133)**: These scores measure the longest matching subsequences, indicating fluency and structure alignment. A score of ~41% is reasonable.

- If this is an **extractive summarization** task, you might aim for a higher ROUGE-1 and ROUGE-L (~0.6+).
- If this is an **abstractive summarization** task, scores in this range are typical, as generating novel text is more challenging.
- For text alignment, scores near 0.5 are reasonable, but if you aim for strict matching, improvements are needed.

first i am using **mBART-Large 50** and **mt5 google 90000 dataset** system crashed continuously then try as 50000, then 30000, 10000, then 5000 **finally used 1000 rows** still crashed , first used tokenizer **max 512** then move to 250 and **then 64** and first used epoch 3 then move to 1 and first used batch size 4 then move to 3,2, and 1 and still crashed then finally move to **BART-small** then only get result

Yes, switching from **mBART-Large-50** and **mT5** to **BART-Small** is a major factor in the reduced ROUGE scores. Larger models generally capture better contextual and syntactic relationships, leading to higher scores. Here's why your ROUGE score dropped:

Why the Score Dropped?

1. **Smaller Model Capacity** – BART-Small has fewer parameters, making it less capable of capturing complex linguistic structures compared to mBART-Large or mT5.
2. **Shorter Token Limit (64 vs. 512)** – Reducing max length from **512 → 64** means less context is available for generating summaries, leading to less overlap with the reference text.
3. **Fewer Training Examples (1,000 vs. 90,000)** – Less training data limits the model's ability to generalize and learn strong patterns.
4. **Lower Batch Size (4 → 1)** – Affects gradient updates, making training less stable.

```python
from transformers import BartForConditionalGeneration, BartTokenizer

# ✅ Load fine-tuned model
model_path = "/content/drive/MyDrive/BART_finetuned_model"
tokenizer = BartTokenizer.from_pretrained(model_path)
model = BartForConditionalGeneration.from_pretrained(model_path)

# ✅ Generate summary
input_text = "former south korean dentist seunggun lee s mobile payments app toss has raised  40 million in funding after eigh
inputs = tokenizer(input_text, return_tensors="pt", truncation=True, max_length=64)

summary_ids = model.generate(**inputs, max_length=16)
summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
print("🔹 Generated Summary:", summary)
```

```
🔹 Generated Summary: former south korean dentist seunggun lee s mobile
```

Streamlit

Text Summarization with Fine-tuned BART

Enter text to summarize:

former south korean dentist seunggun lee s mobile payments app toss has raised 40 million in funding after eight failed apps since 2011. earlier, lee used to work as a dentist with the hospital arm of samsung group and quit his job to found the mobile payments startup. founded in 2015, the app offers peer to peer online transfers, credit scoring, and micro loans.

Generate Summary

Generated Summary:

former south korean dentist seunggun lee s mobile

Conclusion:

Hands-On Experience Gained:

- **Fine-Tuning Transformer Models:** Successfully fine-tuned the mBART model for insurance fraud detection, learning real-world NLP model training.
- **Tokenization & Preprocessing:** Used MBartTokenizer and prepared text inputs in the right format, understanding how models process text.
- **Model Inference:** Generated predictions with the fine-tuned model, applying techniques like beam search and early stopping for better outputs.
- **Working in Colab:** Gained practical experience training large models efficiently using Google Colab.

Thank You