NLP ASSIGNMENT -3

Link for our work on google drive->
Folder:
https://drive.google.com/drive/folders/1XROuhNCJWAZlDPsbaTFP2F16PwEBmxnW?usp=sharing
Task-2:
https://drive.google.com/drive/folders/1sCfheX8GI4R1EsCH3tOul06Ixw9RO_km?usp=sharing
Task-3:
https://drive.google.com/drive/folders/1_jdBsPLia46CXF3UgHRev91Ul0esmmR2?usp=sharing

Part 1

Data loading and Preprocessing.

1. Loading Data

- Data is loaded from three text files: `shakespear_train.txt`, `shakespear_dev.txt`(for validation)
- Each line is stripped of whitespace and stored as a list of sentences.

2. Tokenization

- The corpus is split into tokens (words).
- Words appearing less than twice (`min_freq=2`) are filtered out to reduce noise and vocabulary size.
- Special tokens used:
    - `<PAD>` – Padding token for sequences shorter than `MAX_LEN`.
    - `<START>` – Marks the beginning of each sentence.
    - `<STOP>` – Marks the end of each sentence.
    - `<UNK>` – Used for unknown or rare words.

3. Vocabulary Creation

- A vocabulary is built from the filtered tokens, prepended with special tokens.
- Two mappings are created:

    - `tokenizer` – maps tokens to integer IDs.
    - `tokenizer_inv` – maps integer IDs back to tokens.

4. Sequence Preparation

- Sentences are padded or truncated to a fixed `MAX_LEN` of 256 tokens.
- Each sentence is encoded as a sequence of token IDs using the tokenizer.
- **Dataset Split**: The data was split into training and validation sets before batching.

Model Architecture and Hyperparameters

The model implemented is a simplified **GPT-style Transformer** designed for autoregressive language modeling.

Model Components

- **Embedding Layer**: Transforms token IDs into dense vectors.
- **Positional Encoding**: Adds sinusoidal position information to embeddings.
- **Multi-Head Self-Attention:**
    - Uses multiple attention heads to capture diverse relationships.
    - Includes query, key, value projections, and scaled dot-product attention.
- **Feedforward Network**:
    - Consists of two fully connected layers with a GELU activation in between.
- **Residual Connections** and **Layer Normalization** are used for stable training.
- **Transformer Blocks**: Stacked layers with multi-head self-attention and feedforward sub-layers.
- **Final Linear Layer**: Maps output to vocabulary size for next-token prediction.

Hyperparameter used

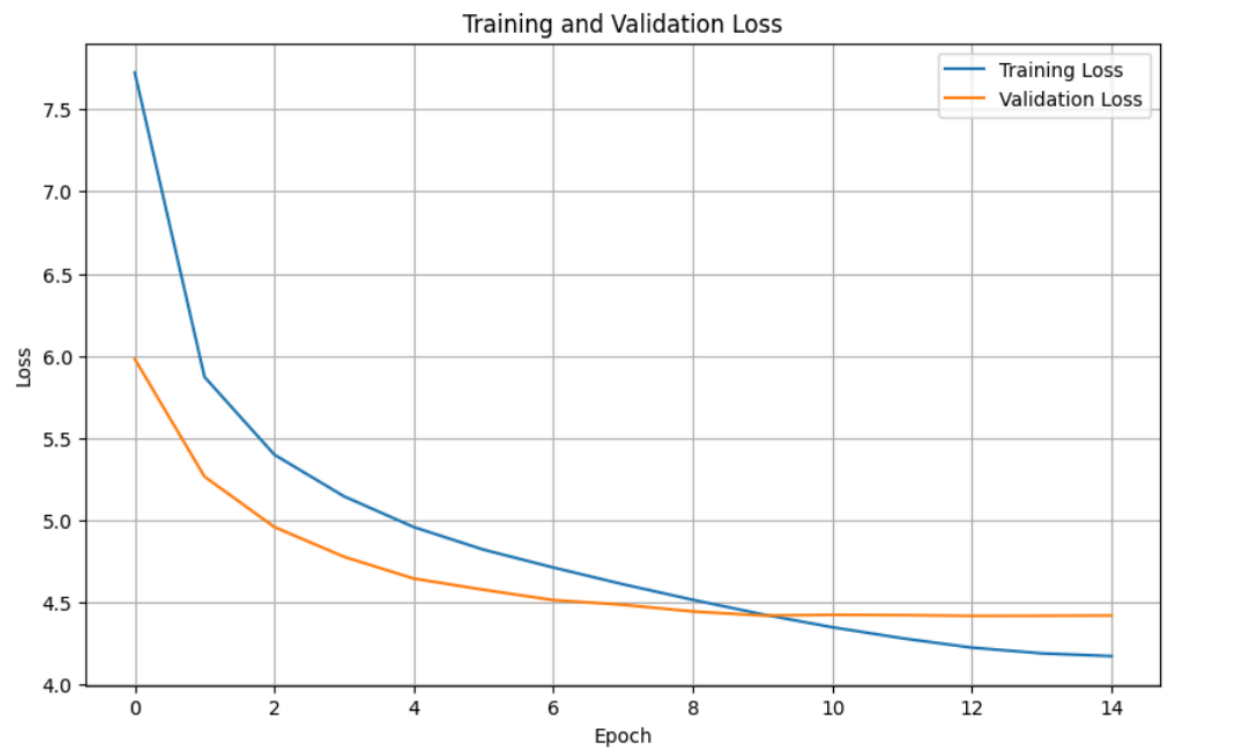
Parameter	Value
d_model	256
n_heads	4
n_layers	4
dropout	0.2
vocab_size	Variable depends upon corpus
Batch_size	32
max_seq_length	256
Learning rate	5e-4

epochs	15 (earlystopping)
Gradient clip	1.0
warmup_steps	2000
optimizer	AdamW
scheduler	Cosine decay

After using these parameters and training the model on the train dataset. The validation perplexity of the model comes out to **82.93**. On epoch 13 with **training loss= 4.2246 and validation_loss=4.4180,t**he model was able to learn meaningful patterns from the training set while maintaining good generalization.

```
Epoch 13: 100%|██████████| 308/308 [00:22<00:00, 13.41it/s]
New best model saved with validation loss: 4.4180
Epoch 13:
  Train Loss: 4.2246, Val Loss: 4.4180, Perplexity: 82.93
  Sample: what are you doing in : A And and 'd KING to and for SICINIUS for And That KING on for of And here and SICINIUS for away o away and for at KING 'd and HENRY at KING I
s First for for and What for In and and and and for and or for And First And and for of of and and o 'd And and and 'd for SICINIUS What or First and o and is But and and is Se
cond and and SICINIUS And to And and where And KING and where and
```

TRAINING AND VALIDATION PLOT



- Training Loss: Decreases steadily from ~7.7 to ~4.2 across the 15 epochs.
- Validation Loss: Also decreases from ~6.0 to ~4.4, but flattens out earlier than the training loss.due to early stopping
- The model is learning effectively from the training data.as both the training and validation loss decreases.
- It's able to generalize reasonably well to unseen validation data without overfitting.

Some test sample on and extracted generated sample along with perplexity.

```
sible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details).
In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be a
llowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any us
e case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
  checkpoint = torch.load(model_path, map_location=DEVICE)  # Removed weights_only=True
100%|██████████| 41/41 [00:01<00:00, 32.33it/s]

Final Perplexity on test set: 62.3472

Generated text samples:

Sample 1:
, if there be no remedy for it , but that you will needs buy and sell men and women like beasts , we shall have all the world drink and white bastard . Of and And and And and and And and and
BRUTUS and and and 'd and and for In and and or And A to and And and and And and First and and and or away for With And and What and SICINIUS And and and and First and

Sample 2:
DUKE VINCENTIO : O heavens ! KING KING Second and and and for . for KING KING away A and . and and for for for and And and to KING ; 'd for for and and for on and away and and for of And and
and o

Sample 3:
what stuff is here POMPEY : 'T was never merry world since , of two , the was put down , and the worser by order of law a gown to keep him warm ; and with and too , to signify , that craft ,
being than innocency , stands for the . and and and 'd for and and and for KING and and and And for and First and and And and and and away and and and and and and o on 'd To and And First an
d and and , and That and The and

Sample 4:
ELBOW : Come your way , sir . for and to and ; for for for 'd and and For 'd for What and and for What away KING and for and and for away and 'd with are are for and for for of . and of for
First

Sample 5:
you , good father friar . away and and and away . and And of o are And KING of in With First and That and for And With and at to and and and away and and KING KING RICHARD and And of And and
away and
```

NLP-Part2 Report

Overview

This project focuses on generating normalized claims from social media posts using two transformer-based models: **BART** and **FLAN-T5**. The process includes data cleaning, tokenization, model training, and evaluation.

1. Preprocessing

To ensure cleaner and more consistent input, we apply a few key preprocessing steps:

- **Text Expansion**: We expand common abbreviations and contractions (like "gov." to "governor", or "he'll" to "he will") using a custom dictionary.

- **Text Cleaning**: We remove URLs, special characters (except English and Hindi letters), normalize whitespace, and lowercase all English words. This helps improve tokenization and model performance.

- **Final Preprocessing**: The **`preprocess_text()`** function wraps both steps above and is applied to each post and claim in the dataset.

2. Tokenization & Dataset Preparation

We use Hugging Face's tokenizers for each model:

- **BART** uses the post and claim text as-is.
- **T5** prepends a task-specific prompt ("normalize:") to each input post.

Both models tokenize text to a fixed length (512 for inputs, 128 for outputs), using padding and truncation. The processed data is converted into Hugging Face **`DatasetDict`** objects for easy batching and training.

3. Model and Training Details

We use two pretrained encoder-decoder models:

- **`facebook/bart-base`**: A 6-layer model fine-tuned for generation tasks.
- **`google/flan-t5-large`**: A larger instruction-tuned model with 24 layers.

Key training settings(for both models):

- **Epochs**: 5
- **Batch Size**: 2
- **Learning Rate**: 2e-5
- **Optimizer**: AdamW
- **Loss**: Cross-entropy (with padding ignored)

BART (Bidirectional and Auto-Regressive Transformers)

BART is a transformer-based encoder-decoder model developed by Facebook AI. It combines the strengths of two major types of transformers: BERT (which is great at understanding language) and GPT (which is good at generating it). The encoder reads and processes the input text, while the decoder generates the output, one token at a time.

What makes BART effective is that it was pretrained using a denoising autoencoding objective — this means the model was trained to reconstruct original text from corrupted input. As a

result, BART learns both how to understand messy or noisy inputs and how to generate fluent, meaningful text. This makes it well-suited for tasks like summarization, paraphrasing, and in our case, claim normalization.

T5 (Text-To-Text Transfer Transformer)

T5, or Text-To-Text Transfer Transformer, is another encoder-decoder model developed by Google Research. What sets T5 apart is its simple yet powerful approach to NLP: everything is cast as a text-to-text task. Whether it's translation, summarization, or classification — the model always takes in text and generates text as output.

In this project, we use **FLAN-T5**, a version of T5 that has been instruction-tuned on a wide variety of tasks. This means the model has been trained to follow natural language prompts more effectively. T5's architecture and training style make it particularly good at tasks where language understanding and generation both matter, like converting informal posts into normalized claims.

Metrics and plots for BART:

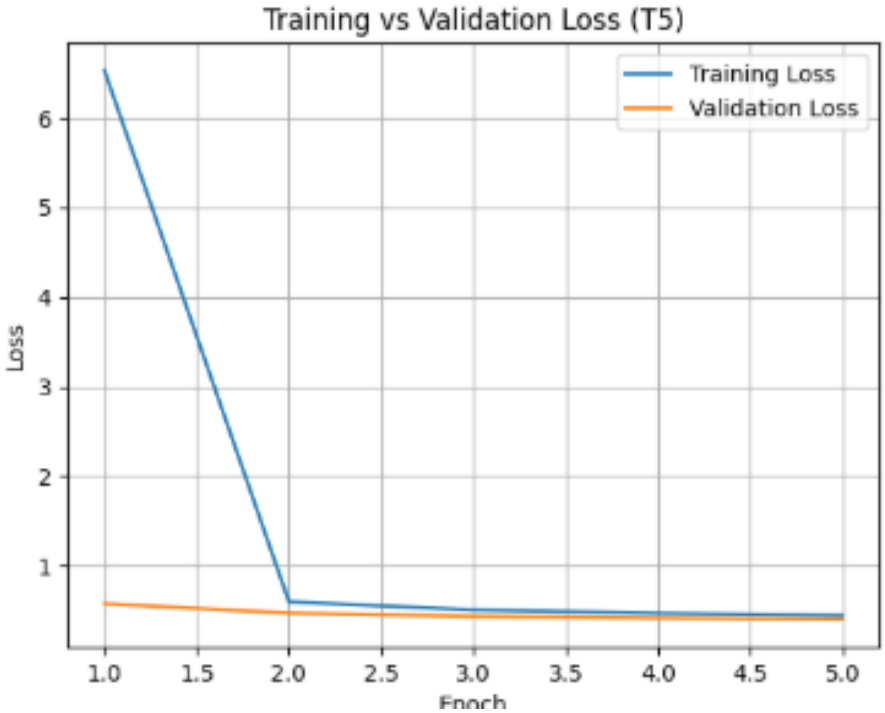
```
This is for BART training

Epoch 1/5
Losses->Train: 3.0979, Val: 2.5585

Epoch 2/5
Losses->Train: 2.5092, Val: 2.4120

Epoch 3/5
Losses->Train: 2.1899, Val: 2.4037

Epoch 4/5
Losses->Train: 1.8938, Val: 2.4058

Epoch 5/5
Losses->Train: 1.6577, Val: 2.3828
```

```
Metrics score->
ROUGE: {'rouge1': 0.33927986341833827, 'rouge2': 0.21499669651082462, 'rougeL': 0.31792184156383896, 'rougeLsum': 0.31580756142137156}
BLEU: 0.1986
BERTScore F1: 0.8783
bart Model saved.
```

Metrics and plots for T5::

```
Epoch 1/5
Losses -->Train: 6.5277, Val: 0.5797

Epoch 2/5
Losses -->Train: 0.6006, Val: 0.4742

Epoch 3/5
Losses -->Train: 0.5100, Val: 0.4388

Epoch 4/5
Losses -->Train: 0.4722, Val: 0.4176

Epoch 5/5
Losses -->Train: 0.4480, Val: 0.4070
```

```
Evaluation Metrics-->
ROUGE: {'rouge1': 0.25086252657816405, 'rouge2': 0.15908467351056593, 'rougeL': 0.2316540421233204, 'rougeLsum': 0.2314983753378036}
BLEU: 0.0781
BERTScore F1: 0.8526
T5 model saved.
```



Comparative Analysis of BART and T5 Model Performance

The performance comparison between the BART and T5 models reveals some important insights based on both the training behavior and the evaluation metrics.

From the training logs and loss graphs, it's evident that the **T5 model converged faster and more smoothly** than BART. The T5 model's training loss dropped drastically from 6.52 to 0.448 over five epochs, with the validation loss showing consistent improvement and ending at 0.407. In contrast, BART started with a training loss of 3.89 and ended at 1.657, while its validation loss plateaued around 2.38 after initial improvements. This suggests that T5 generalized better to unseen data in fewer epochs.

However, in terms of evaluation metrics, **BART outperformed T5 on BLEU (0.1986 vs 0.0781)** and slightly on **BERTScore (0.8783 vs 0.8526)**. ROUGE scores were relatively close between the two, with BART having a slight edge in ROUGE-1 and ROUGE-L, while T5 did marginally better on ROUGE-2.

Resource Constraints and Model Selection

A significant factor influencing the choice of models was **GPU memory limitation on Kaggle**. While experimenting, I attempted to use **T5-Large and BART-Large**, but both models consistently maxed out the available GPU memory, making it infeasible to run them within the current resource setup. Due to these constraints, I proceeded with the base versions of both BART and T5 for a fair and practical comparison.

Why FLAN-T5 Is a Better Choice Than Base T5

While this experiment used the base T5 model, it's worth noting that **FLAN-T5 outperforms vanilla T5** due to instruction tuning. FLAN-T5 is trained on a diverse set of instructions and is more robust in zero-shot and few-shot settings. This results in better alignment with downstream tasks without needing extensive task-specific fine-tuning.

NLP-Part3 Report

Saved Checkpoints after each epoch link to drive:
NLP_Assignment_3_files

Preprocessing Steps

The preprocessing pipeline for the Multimodal Sarcasm Explanation (MuSE) task involved careful data preparation from the provided MORE+ dataset. The steps are outlined clearly below:

1. Data Loading
The dataset consists of social media posts, each containing:

- Textual caption
- Corresponding image
- Sarcasm explanation (target explanation)
- Sarcasm target phrase

The data was provided via three main file types:

- Tab-separated .tsv files for train, validation, and test splits (train_df.tsv, val_df.tsv, test_df.tsv).
- Pickle files containing:
  - Image descriptions (D_train.pkl, D_val.pkl, D_test.pkl).
  - Detected objects (O_train.pkl, O_val.pkl, O_test.pkl).

2. Textual Data Preparation
Each textual input sequence for the model was constructed explicitly by concatenating:

- Original textual caption (from text column).
- Corresponding image description (from D_*.pkl file).
- Detected object labels (from O_*.pkl file).
- Sarcasm target phrase, separated by a special token (</s>).

Example of input concatenation clearly demonstrated:
[caption] + [image descriptions] + [detected objects] + </s> + [sarcasm target]

3. Tokenization
Used BART-base tokenizer from Hugging Face (facebook/bart-base).
Tokenized sequences using:

- max_length = 256
- Padding strategy: 'max_length'
- Truncation strategy: enabled (True)

4. Image Processing
Used Vision Transformer (ViT) preprocessor from Hugging Face (google/vit-base-patch16-224).
Images resized and standardized according to ViT model requirements:

- Size: 224 × 224
- Channels: RGB
- Normalization: standard ImageNet normalization.

5. Handling Missing Data
Explicitly managed missing or mismatched entries in the pickle files by replacing missing descriptions and detected objects with empty placeholders to ensure smooth model training without interruptions.

Model Components:

The implemented model closely follows the TURBO architecture without the usage of external Knowledge Graphs (KG) or Graph Convolutional Networks (GCN), as explicitly required.

1. Encoder: Textual Modality
BART-base Encoder (facebook/bart-base):
Encodes concatenated textual input sequences into semantic embeddings (768-dimensional embeddings).

2. Encoder: Visual Modality
Vision Transformer (ViT-base, patch16, 224px):

- Extracts visual features from images into 768-dimensional embeddings.
- Image embedding obtained by averaging ViT's patch embeddings (mean-pooling).

3. Shared Fusion Module (Multimodal Fusion)

- Implements self-attention separately for text and image embeddings.
- Computes explicit cross-modal interactions between text and image.
- Employs gated fusion mechanisms with learnable parameters to effectively balance modality-specific information.
- Final fused embedding dimension: 768.

Shared Fusion learnable parameters clearly defined:
α1, α2, β1, β2: initialized to 0.25, fully trainable.

4. Decoder (Explanation Generation)

BART-base Decoder:

- Takes fused multimodal embeddings from the Shared Fusion module.
- Generates the textual sarcasm explanations autoregressively.

5. Prediction Head
Linear projection (lm_head) mapping decoder outputs to vocabulary logits for generating textual explanations.

Hyperparameter	**Value**
Optimizer	AdamW
Learning Rate	1e-4
Batch Size	8
Number of Epochs	5
Max Sequence Length	256 tokens
Image Size	224 × 224
Attention Heads	8 (in Shared Fusion)
Embedding Dimension	768

Metrics and Examples after each epoch:

Final step results are in bold, which contains train loss, validation loss, rouge(R1, R2, RL), Bleu(bleu1, bleu2, bleu3 and bleu4), Meteor, and BertScore.

Step: 49, Tag: train_loss, Value: 0.5665047764778137
Step: 49, Tag: epoch, Value: 0.0
Step: 99, Tag: train_loss, Value: 0.33550897240638733
Step: 99, Tag: epoch, Value: 0.0
Step: 149, Tag: train_loss, Value: 0.26080650091171265
Step: 149, Tag: epoch, Value: 0.0
Step: 199, Tag: train_loss, Value: 0.1972956508398056
Step: 199, Tag: epoch, Value: 0.0
Step: 249, Tag: train_loss, Value: 0.14948351681232452
Step: 249, Tag: epoch, Value: 0.0
Step: 299, Tag: train_loss, Value: 0.16805507242679596
Step: 299, Tag: epoch, Value: 0.0
Step: 349, Tag: train_loss, Value: 0.09759218245744705
Step: 349, Tag: epoch, Value: 0.0
Step: 372, Tag: val_loss, Value: 0.15321733057498932
Step: 372, Tag: val_rouge1, Value: 0.7073759436607361
Step: 372, Tag: val_rouge2, Value: 0.5588889122009277
Step: 372, Tag: val_rougeL, Value: 0.706817090511322
Step: 372, Tag: val_bleu1, Value: 0.7393844723701477
Step: 372, Tag: val_bleu2, Value: 0.6493244767189026
Step: 372, Tag: val_bleu3, Value: 0.5784980058670044
Step: 372, Tag: val_bleu4, Value: 0.5186132192611694
Step: 372, Tag: val_meteor, Value: 0.7405498027801514
Step: 372, Tag: val_bertscore, Value: 0.9388163685798645
Step: 372, Tag: epoch, Value: 0.0
Step: 399, Tag: train_loss, Value: 0.09759923815727234
Step: 399, Tag: epoch, Value: 1.0
Step: 449, Tag: train_loss, Value: 0.11802801489830017
Step: 449, Tag: epoch, Value: 1.0
Step: 499, Tag: train_loss, Value: 0.08998755365610123
Step: 499, Tag: epoch, Value: 1.0
Step: 549, Tag: train_loss, Value: 0.12633782625198364
Step: 549, Tag: epoch, Value: 1.0
Step: 599, Tag: train_loss, Value: 0.06764454394578934
Step: 599, Tag: epoch, Value: 1.0
Step: 649, Tag: train_loss, Value: 0.0655255988240242
Step: 649, Tag: epoch, Value: 1.0
Step: 699, Tag: train_loss, Value: 0.08621694147586823
Step: 699, Tag: epoch, Value: 1.0
Step: 745, Tag: val_loss, Value: 0.10350071638822556

Step: 745, Tag: val_rouge1, Value: 0.8159939646720886
Step: 745, Tag: val_rouge2, Value: 0.7169985175132751
Step: 745, Tag: val_rougeL, Value: 0.8156417012214661
Step: 745, Tag: val_bleu1, Value: 0.8381211161613464
Step: 745, Tag: val_bleu2, Value: 0.7829239368438721
Step: 745, Tag: val_bleu3, Value: 0.7355190515518188
Step: 745, Tag: val_bleu4, Value: 0.6931716203689575
Step: 745, Tag: val_meteor, Value: 0.843808650970459
Step: 745, Tag: val_bertscore, Value: 0.9614045023918152
Step: 745, Tag: epoch, Value: 1.0
Step: 749, Tag: train_loss, Value: 0.026484452188014984
Step: 749, Tag: epoch, Value: 2.0
Step: 799, Tag: train_loss, Value: 0.03884346783161163
Step: 799, Tag: epoch, Value: 2.0
Step: 849, Tag: train_loss, Value: 0.03356918320059776
Step: 849, Tag: epoch, Value: 2.0
Step: 899, Tag: train_loss, Value: 0.0321369469165802
Step: 899, Tag: epoch, Value: 2.0
Step: 949, Tag: train_loss, Value: 0.03833429142832756
Step: 949, Tag: epoch, Value: 2.0
Step: 999, Tag: train_loss, Value: 0.03346368670463562
Step: 999, Tag: epoch, Value: 2.0
Step: 1049, Tag: train_loss, Value: 0.01613662950694561
Step: 1049, Tag: epoch, Value: 2.0
Step: 1099, Tag: train_loss, Value: 0.026745811104774475
Step: 1099, Tag: epoch, Value: 2.0
Step: 1118, Tag: val_loss, Value: 0.08694779872894287
Step: 1118, Tag: val_rouge1, Value: 0.8749780058860779
Step: 1118, Tag: val_rouge2, Value: 0.7982417941093445
Step: 1118, Tag: val_rougeL, Value: 0.8746809363365173
Step: 1118, Tag: val_bleu1, Value: 0.8726016879081726
Step: 1118, Tag: val_bleu2, Value: 0.8299521207809448
Step: 1118, Tag: val_bleu3, Value: 0.7916625738143921
Step: 1118, Tag: val_bleu4, Value: 0.7558540105819702
Step: 1118, Tag: val_meteor, Value: 0.8913568258285522
Step: 1118, Tag: val_bertscore, Value: 0.9722815155982971
Step: 1118, Tag: epoch, Value: 2.0
Step: 1149, Tag: train_loss, Value: 0.01400039903819561
Step: 1149, Tag: epoch, Value: 3.0
Step: 1199, Tag: train_loss, Value: 0.018302638083696365
Step: 1199, Tag: epoch, Value: 3.0
Step: 1249, Tag: train_loss, Value: 0.016554657369852066
Step: 1249, Tag: epoch, Value: 3.0
Step: 1299, Tag: train_loss, Value: 0.01590472087264061

Step: 1299, Tag: epoch, Value: 3.0
Step: 1349, Tag: train_loss, Value: 0.014206493273377419
Step: 1349, Tag: epoch, Value: 3.0
Step: 1399, Tag: train_loss, Value: 0.007244852837175131
Step: 1399, Tag: epoch, Value: 3.0
Step: 1449, Tag: train_loss, Value: 0.7524173855781555
Step: 1449, Tag: epoch, Value: 3.0
Step: 1491, Tag: val_loss, Value: 0.09799870103597641
Step: 1491, Tag: val_rouge1, Value: 0.8581203818321228
Step: 1491, Tag: val_rouge2, Value: 0.7791200280189514
Step: 1491, Tag: val_rougeL, Value: 0.8581831455230713
Step: 1491, Tag: val_bleu1, Value: 0.8689537048339844
Step: 1491, Tag: val_bleu2, Value: 0.8258771896362305
Step: 1491, Tag: val_bleu3, Value: 0.7876883745193481
Step: 1491, Tag: val_bleu4, Value: 0.7524678111076355
Step: 1491, Tag: val_meteor, Value: 0.8683880567550659
Step: 1491, Tag: val_bertscore, Value: 0.9683798551559448
Step: 1491, Tag: epoch, Value: 3.0
Step: 1499, Tag: train_loss, Value: 0.026229944080114365
Step: 1499, Tag: epoch, Value: 4.0
Step: 1549, Tag: train_loss, Value: 0.016450444236397743
Step: 1549, Tag: epoch, Value: 4.0
Step: 1599, Tag: train_loss, Value: 0.02610783651471138
Step: 1599, Tag: epoch, Value: 4.0
Step: 1649, Tag: train_loss, Value: 0.015443291515111923
Step: 1649, Tag: epoch, Value: 4.0
Step: 1699, Tag: train_loss, Value: 0.007812345866113901
Step: 1699, Tag: epoch, Value: 4.0
Step: 1749, Tag: train_loss, Value: 0.009937013499438763
Step: 1749, Tag: epoch, Value: 4.0
Step: 1799, Tag: train_loss, Value: 0.00782430823892355
Step: 1799, Tag: epoch, Value: 4.0
Step: 1849, Tag: train_loss, Value: 0.009203124791383743
Step: 1849, Tag: epoch, Value: 4.0
Step: 1864, Tag: val_loss, Value: 0.08208069205284119
Step: 1864, Tag: val_rouge1, Value: 0.8713607788085938
Step: 1864, Tag: val_rouge2, Value: 0.7950979471206665
Step: 1864, Tag: val_rougeL, Value: 0.8713168501853943
Step: 1864, Tag: val_bleu1, Value: 0.8825816512107849
Step: 1864, Tag: val_bleu2, Value: 0.8414592146873474
Step: 1864, Tag: val_bleu3, Value: 0.8044975399971008
Step: 1864, Tag: val_bleu4, Value: 0.7698612809181213
Step: 1864, Tag: val_meteor, Value: 0.8902734518051147
Step: 1864, Tag: val_bertscore, Value: 0.9720677137374878

Step: 1864, Tag: epoch, Value: 4.0

Below are the validation examples after each epoch in ascending order:

.....

```
Sample Generated Sarcasm Explanations:

Example 1:
Generated Explanation: the author is pissed at <user> for not getting
platform inal.
Actual Explanation: the author is pissed at <user> for not getting network
in malad.

Example 2:
Generated Explanation: the worst than waiting for an hour on the t bacon
for a parking to come open in violent, windy chicago.
Actual Explanation: nothing worst than waiting for an hour on the tarmac
for a gate to come open in snowy, windy chicago.


Example 3:
Generated Explanation: theody likes getting one hour of their life stuck
away.
Actual Explanation: nobody likes getting one hour of their life sucked
away.

Example 4:
Generated Explanation: the a min americay baconi salad on monday morning
is not a good way to start the new week.
Actual Explanation: having a salivary gland biopsy on monday morning is
not a good way to start the new week.

Example 5:
Generated Explanation: the author is worried that the weekend is going to
being with a high of, 1 and wind chi not,10.
Actual Explanation: the author is worried that the weekend is going to be
freezing with a high of -1 and windchill probably -30.
```

.....

```
Sample Generated Sarcasm Explanations:
```

Example 1:
Generated Explanation: the author is pissed at <user> for not getting network in betweenad.
Actual Explanation: the author is pissed at <user> for not getting network in malad.

Example 2:
Generated Explanation: it worst than waiting for an hour on the ter for a gate to come open in snowy, windy chicago.
Actual Explanation: nothing worst than waiting for an hour on the tarmac for a gate to come open in snowy, windy chicago.

Example 3:
Generated Explanation: theody likes getting one hour of their life bunny away.
Actual Explanation: nobody likes getting one hour of their life sucked away.

Example 4:
Generated Explanation: this a minianary gardenian exam on monday morning is not a good way to start the new week.
Actual Explanation: having a salivary gland biopsy on monday morning is not a good way to start the new week.

Example 5:
Generated Explanation: the author is worried that the weekend is going to be freezing with a high of -1 and windchill possibly -30.
Actual Explanation: the author is worried that the weekend is going to be freezing with a high of -1 and windchill probably -30.

.....
Sample Generated Sarcasm Explanations:

Example 1:
Generated Explanation: the author is pissed at <user> for not getting network in malad.
Actual Explanation: the author is pissed at <user> for not getting network in malad.

Example 2:
Generated Explanation: people worst than waiting for an hour on the t wheel for a gate to come open in snowy, windy chicago.
Actual Explanation: nothing worst than waiting for an hour on the tarmac for a gate to come open in snowy, windy chicago.

Example 3:
Generated Explanation: nobody likes getting one hour of their life sob away.
Actual Explanation: nobody likes getting one hour of their life sucked away.

Example 4:
Generated Explanation: trump a plant foodary pet i reply on monday morning is not a good way to start the new week.
Actual Explanation: having a salivary gland biopsy on monday morning is not a good way to start the new week.

Example 5:
Generated Explanation: the author is worried that the weekend is going to be freezing with a high of -1 and windchill possibly -30.
Actual Explanation: the author is worried that the weekend is going to be freezing with a high of -1 and windchill probably -30.

.....
Sample Generated Sarcasm Explanations:

Example 1:
Generated Explanation: the author is pissed at <user> for not getting network in malad.
Actual Explanation: the author is pissed at <user> for not getting network in malad.

Example 2:
Generated Explanation: to worst than waiting for an hour on the t sidewalks for a gate to come open in snowy, windy chicago.

Actual Explanation: nothing worst than waiting for an hour on the tarmac for a gate to come open in snowy, windy chicago.

Example 3:
Generated Explanation: thereody likes getting one hour of their life bunny away.
Actual Explanation: nobody likes getting one hour of their life sucked away.

Example 4:
Generated Explanation: when a colary volcanosis on monday morning is not a good way to start the new week.
Actual Explanation: having a salivary gland biopsy on monday morning is not a good way to start the new week.

Example 5:
Generated Explanation: the author is worried that the weekend is going to be freezing with a high of -1 and windchill it, 30.
Actual Explanation: the author is worried that the weekend is going to be freezing with a high of -1 and windchill probably -30.

.....
Sample Generated Sarcasm Explanations:

Example 1:
Generated Explanation: the author is pissed at <user> for not getting network in malad.
Actual Explanation: the author is pissed at <user> for not getting network in malad.

Example 2:
Generated Explanation: when worst than waiting for an hour on the t sidewalks for a gate to come open in snowy, windy chicago.
Actual Explanation: nothing worst than waiting for an hour on the tarmac for a gate to come open in snowy, windy chicago.

Example 3:
Generated Explanation: nobody likes getting one hour of their life bunny away.

```
Actual Explanation: nobody likes getting one hour of their life sucked
away.

Example 4:
Generated Explanation: these a styl avarythuriosis on monday morning is
not a good way to start the new week.
Actual Explanation: having a salivary gland biopsy on monday morning is
not a good way to start the new week.

Example 5:
Generated Explanation: the author is worried that the weekend is going to
be freezing with a high of -1 and windchill possibly -30.
Actual Explanation: the author is worried that the weekend is going to be

freezing with a high of -1 and windchill probably -30.
```

Contribution
Task 1 ayaan hasan
Task 2 ram dabas
Task 3 kartik