Codemix Generation

Lost in Context

Sahasra V. 2022102024
Sreeja P. 2022101081
Shravya P. 2022102077

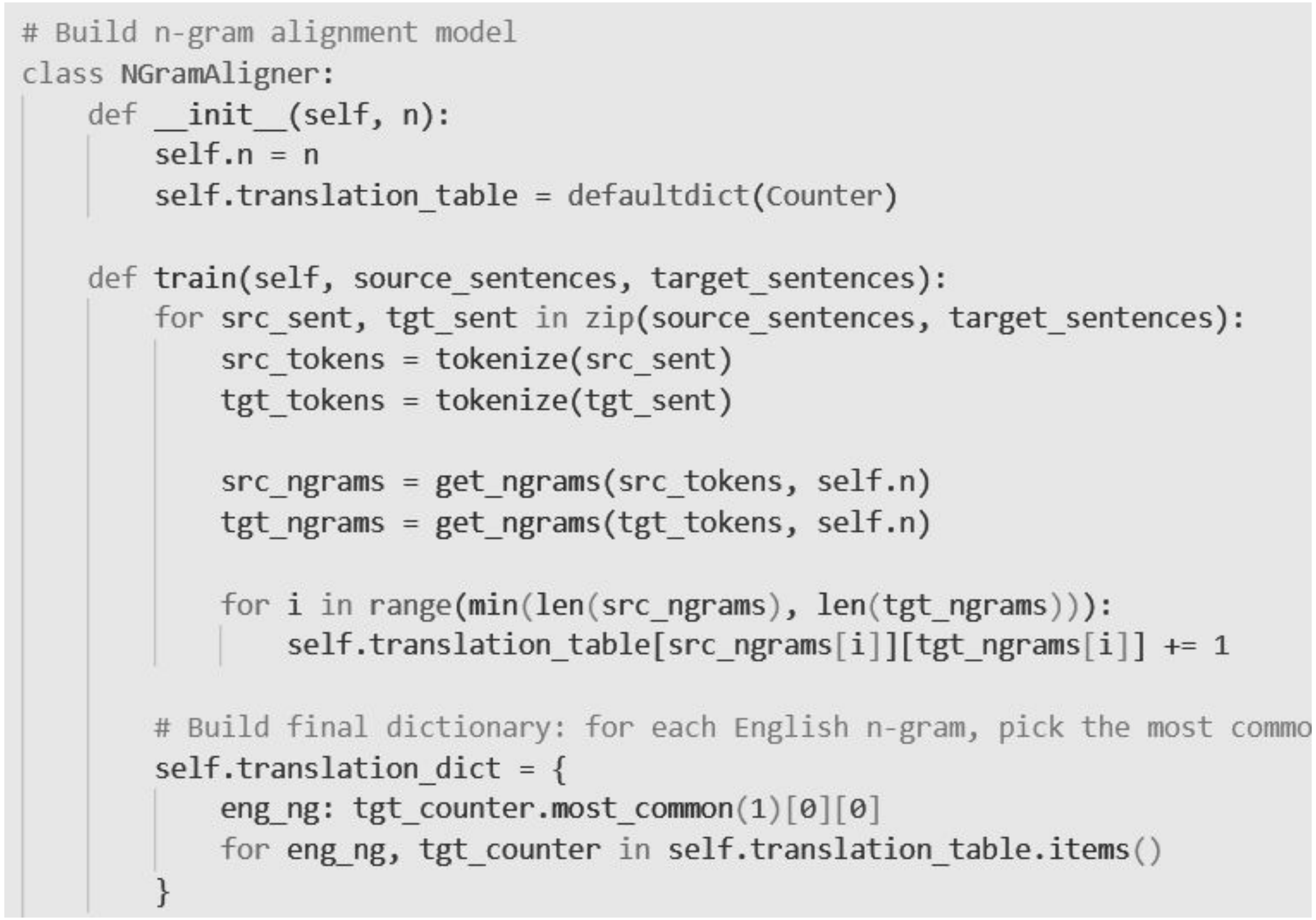
Introduction

Hinglish is a mix of Hindi and English used widely in informal communication. Code-mix generation involves creating such mixed-language text automatically. It requires understanding both languages and their switching patterns. Hinglish generation is challenging due to non-standard spellings and grammar. It is useful in chatbots, translation, and social media analysis.

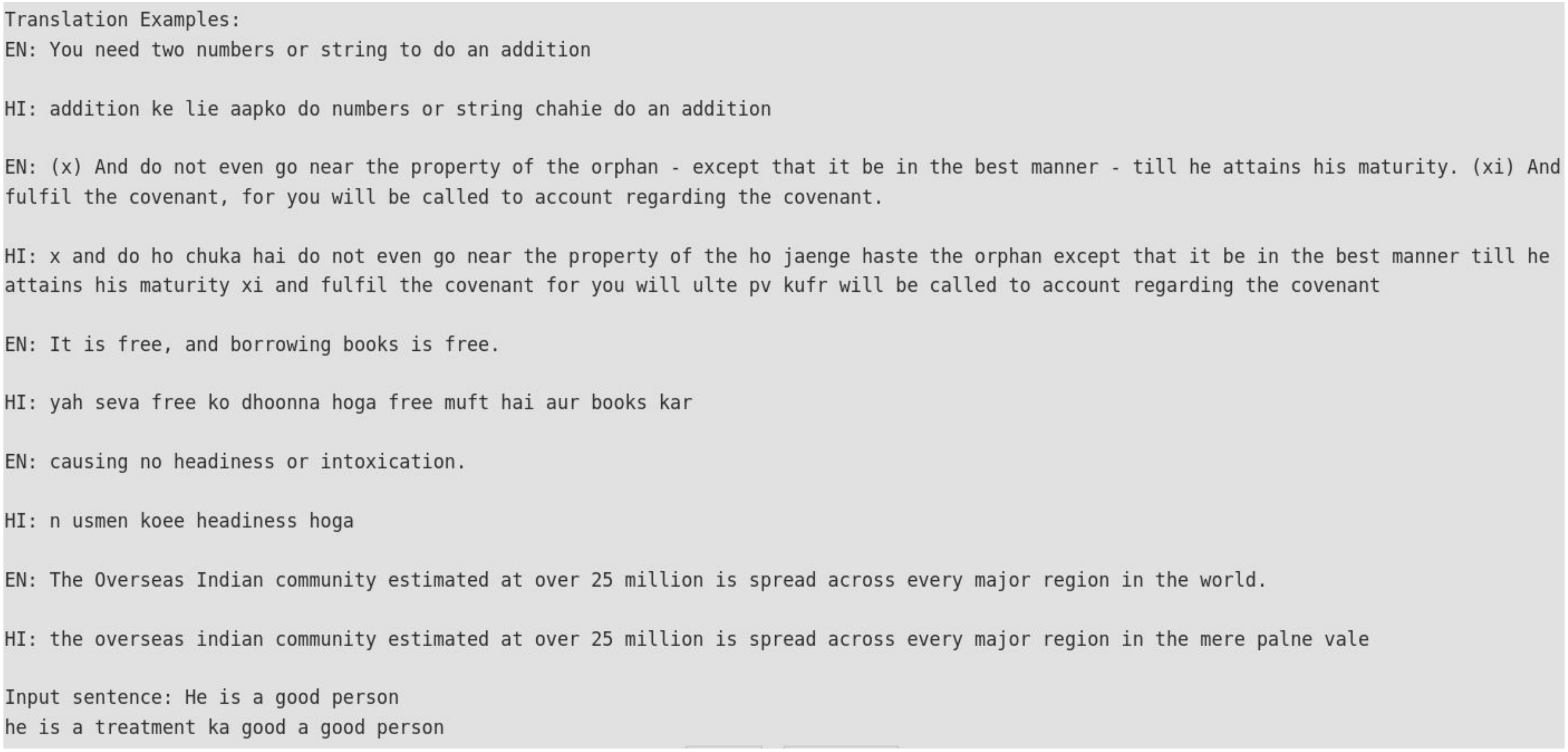
In this project, we have tried translations using several models - both baseline and transformer ones. We compare these models by testing them on user inputs and evaluation metrics to determine the best model.

N-gram model

- Implemented a trigram (n=3) statistical machine translation model for English-to-Hinglish translation.
- Uses statistical co-occurrence instead of neural networks to learn word/phrase alignments from a synthetic parallel corpus.
- Builds a translation dictionary from frequently co-occurring English-Hinglish n-gram pairs.
- At inference, English trigrams are mapped to their most probable Hinglish equivalents, with overlapping parts merged for fluency.



```python
# Build n-gram alignment model
class NGramAligner:
    def __init__(self, n):
        self.n = n
        self.translation_table = defaultdict(Counter)

    def train(self, source_sentences, target_sentences):
        for src_sent, tgt_sent in zip(source_sentences, target_sentences):
            src_tokens = tokenize(src_sent)
            tgt_tokens = tokenize(tgt_sent)

            src_ngrams = get_ngrams(src_tokens, self.n)
            tgt_ngrams = get_ngrams(tgt_tokens, self.n)

            for i in range(min(len(src_ngrams), len(tgt_ngrams))):
                self.translation_table[src_ngrams[i]][tgt_ngrams[i]] += 1

        # Build final dictionary: for each English n-gram, pick the most commo
        self.translation_dict = {
            eng_ng: tgt_counter.most_common(1)[0][0]
            for eng_ng, tgt_counter in self.translation_table.items()
        }
```



```
Translation Examples:
EN: You need two numbers or string to do an addition

HI: addition ke lie aapko do numbers or string chahie do an addition

EN: (x) And do not even go near the property of the orphan - except that it be in the best manner - till he attains his maturity. (xi) And
fulfil the covenant, for you will be called to account regarding the covenant.

HI: x and do ho chuka hai do not even go near the property of the ho jaenge haste the orphan except that it be in the best manner till he
attains his maturity xi and fulfil the covenant for you will ulte pv kufr will be called to account regarding the covenant

EN: It is free, and borrowing books is free.

HI: yah seva free ko dhoonna hoga free muft hai aur books kar

EN: causing no headiness or intoxication.

HI: n usmen koee headiness hoga

EN: The Overseas Indian community estimated at over 25 million is spread across every major region in the world.

HI: the overseas indian community estimated at over 25 million is spread across every major region in the mere palne vale

Input sentence: He is a good person
he is a treatment ka good a good person
```

N-gram model

- The approach is rule-based, lightweight, and interpretable, offering a baseline to complex models like transformers.
- The core hyperparameter is n (n-gram size):
  - n = 3 balances context capture and data sparsity.
  - Lower n (1–2): more general, less expressive translations.
  - Higher n (4–5): more context but risk of sparsity and overfitting.

LSTM model

Architecture: Sequence-to-Sequence (Seq2Seq) model with attention mechanism for better focus on relevant input tokens.

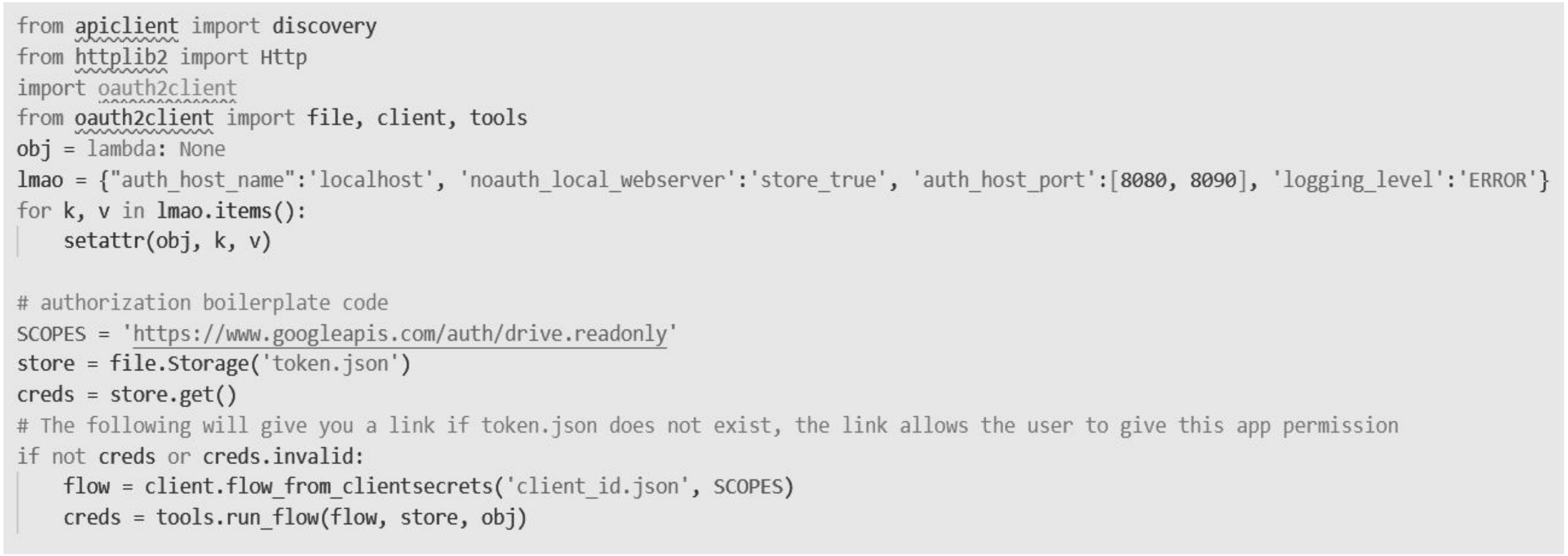
Encoder: Two stacked LSTM layers, each with 300 hidden units, using GloVe embeddings (300-dim) for input representation.

Decoder: Separate embedding layer and single LSTM layer (300 units), generating output tokens sequentially.

Attention: Helps the decoder focus on relevant input parts at each timestep to improve context awareness.

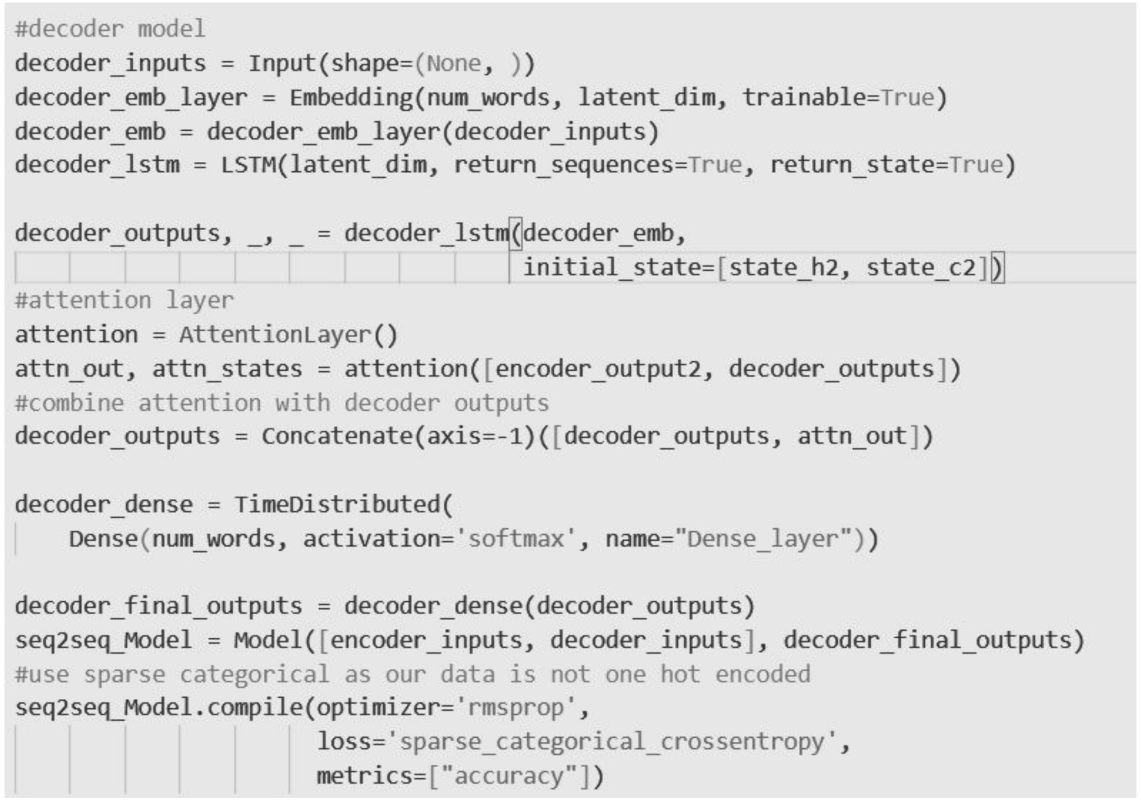
Output Layer: Decoder outputs combined with attention context and passed through a time-distributed dense layer with softmax to predict token probabilities.

LSTM model - GloVE embeddings



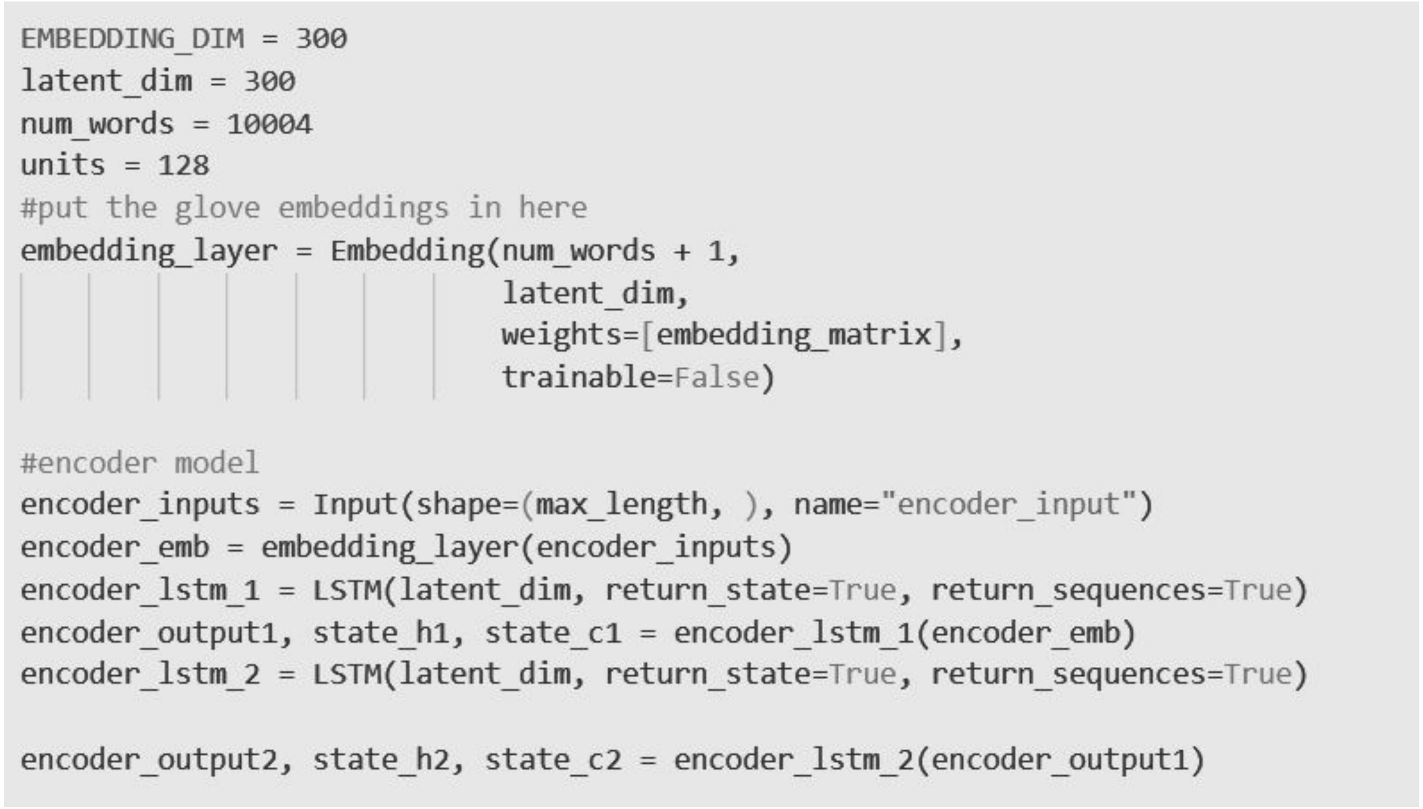
```python
from apiclient import discovery
from httplib2 import Http
import oauth2client
from oauth2client import file, client, tools
obj = lambda: None
lmao = {"auth_host_name":'localhost', 'noauth_local_webserver':'store_true', 'auth_host_port':[8080, 8090], 'logging_level':'ERROR'}
for k, v in lmao.items():
    setattr(obj, k, v)

# authorization boilerplate code
SCOPES = 'https://www.googleapis.com/auth/drive.readonly'
store = file.Storage('token.json')
creds = store.get()
# The following will give you a link if token.json does not exist, the link allows the user to give this app permission
if not creds or creds.invalid:
    flow = client.flow_from_clientsecrets('client_id.json', SCOPES)
    creds = tools.run_flow(flow, store, obj)
```

LSTM model - Decoder



```
#decoder model
decoder_inputs = Input(shape=(None, ))
decoder_emb_layer = Embedding(num_words, latent_dim, trainable=True)
decoder_emb = decoder_emb_layer(decoder_inputs)
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)

decoder_outputs, _, _ = decoder_lstm(decoder_emb,
                                     initial_state=[state_h2, state_c2])
#attention layer
attention = AttentionLayer()
attn_out, attn_states = attention([encoder_output2, decoder_outputs])
#combine attention with decoder outputs
decoder_outputs = Concatenate(axis=-1)([decoder_outputs, attn_out])

decoder_dense = TimeDistributed(
    Dense(num_words, activation='softmax', name="Dense_layer"))

decoder_final_outputs = decoder_dense(decoder_outputs)
seq2seq_Model = Model([encoder_inputs, decoder_inputs], decoder_final_outputs)
#use sparse categorical as our data is not one hot encoded
seq2seq_Model.compile(optimizer='rmsprop',
                      loss='sparse_categorical_crossentropy',
                      metrics=["accuracy"])
```

LSTM model - Encoder



```
EMBEDDING_DIM = 300
latent_dim = 300
num_words = 10004
units = 128
#put the glove embeddings in here
embedding_layer = Embedding(num_words + 1,
                            latent_dim,
                            weights=[embedding_matrix],
                            trainable=False)

#encoder model
encoder_inputs = Input(shape=(max_length, ), name="encoder_input")
encoder_emb = embedding_layer(encoder_inputs)
encoder_lstm_1 = LSTM(latent_dim, return_state=True, return_sequences=True)
encoder_output1, state_h1, state_c1 = encoder_lstm_1(encoder_emb)
encoder_lstm_2 = LSTM(latent_dim, return_state=True, return_sequences=True)

encoder_output2, state_h2, state_c2 = encoder_lstm_2(encoder_output1)
```

LSTM model

Hyperparameters:

- `latent_dim = 300`
- `batch_size = 64`
- `optimizer = RMSProp`
- `loss = sparse_categorical_crossentropy`
- Pre-trained GloVe embeddings (300-dim) used for word initialization

Results and Issues:

- Faced repetitive translations due to softmax bias towards frequent tokens.
- Large vocabulary size caused the model to overuse high-frequency, low-information words, reducing translation quality.

T5 model

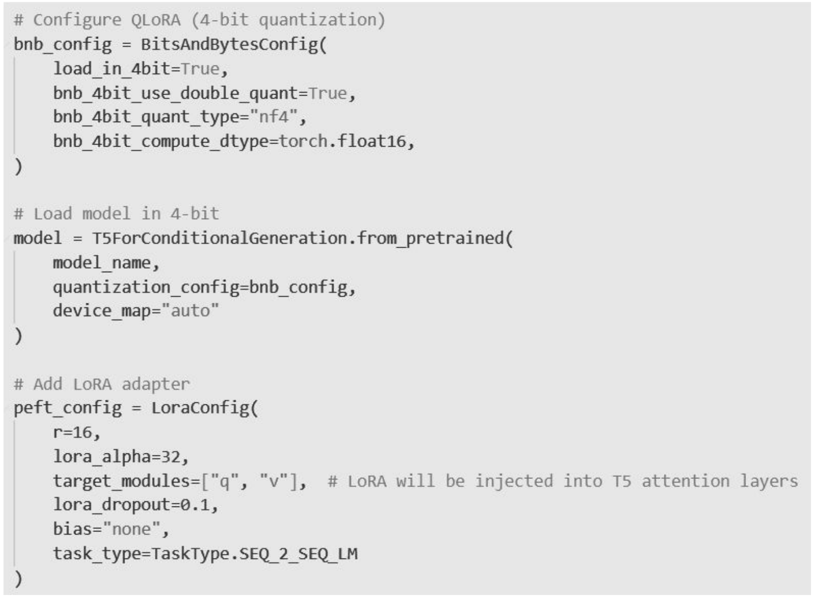
- Model: Used T5-small, a lightweight variant of T5 that treats all NLP tasks in a text-to-text format.
- Fine-tuning method: Employed QLoRA, a parameter-efficient technique combining 4-bit quantization and Low-Rank Adaptation (LoRA).
- Memory efficiency: Used 4-bit quantization via the BitsAndBytes library to reduce GPU memory usage without compromising performance.
- LoRA integration: Inserted LoRA layers into the query and value projections of T5's attention modules.
- Training strategy: Only LoRA adapters were trained, keeping the base model frozen, enabling faster and efficient fine-tuning with fewer parameters.

T5 model

Model: t5-small
Batch size: 32
Training epochs: 3
Learning rate: 1e-5
Max input/output length: 128 tokens each
Quantization: 4-bit using BitsAndBytes
Adapter type: LoRA with

    rank (r) = 16
    scaling factor (α) = 32
    dropout = 0.1

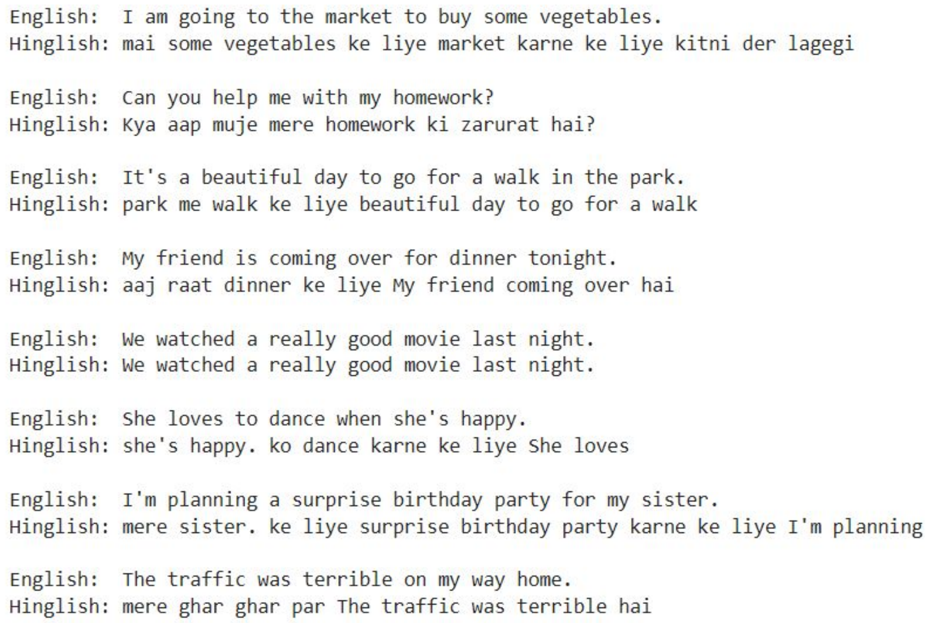
Optimizer: AdamW (configured via Hugging Face Trainer)



```
# Configure QLoRA (4-bit quantization)
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16,
)

# Load model in 4-bit
model = T5ForConditionalGeneration.from_pretrained(
    model_name,
    quantization_config=bnb_config,
    device_map="auto"
)

# Add LoRA adapter
peft_config = LoraConfig(
    r=16,
    lora_alpha=32,
    target_modules=["q", "v"],  # LoRA will be injected into T5 attention layers
    lora_dropout=0.1,
    bias="none",
    task_type=TaskType.SEQ_2_SEQ_LM
)
```

T5 model

Unlike the baseline models, this one was trained on findnitai/english-to-hinglish which is much larger dataset than HinGE.



```
English:  I am going to the market to buy some vegetables.
Hinglish: mai some vegetables ke liye market karne ke liye kitni der lagegi

English:  Can you help me with my homework?
Hinglish: Kya aap muje mere homework ki zarurat hai?

English:  It's a beautiful day to go for a walk in the park.
Hinglish: park me walk ke liye beautiful day to go for a walk

English:  My friend is coming over for dinner tonight.
Hinglish: aaj raat dinner ke liye My friend coming over hai

English:  We watched a really good movie last night.
Hinglish: We watched a really good movie last night.

English:  She loves to dance when she's happy.
Hinglish: she's happy. ko dance karne ke liye She loves

English:  I'm planning a surprise birthday party for my sister.
Hinglish: mere sister. ke liye surprise birthday party karne ke liye I'm planning

English:  The traffic was terrible on my way home.
Hinglish: mere ghar ghar par The traffic was terrible hai
```

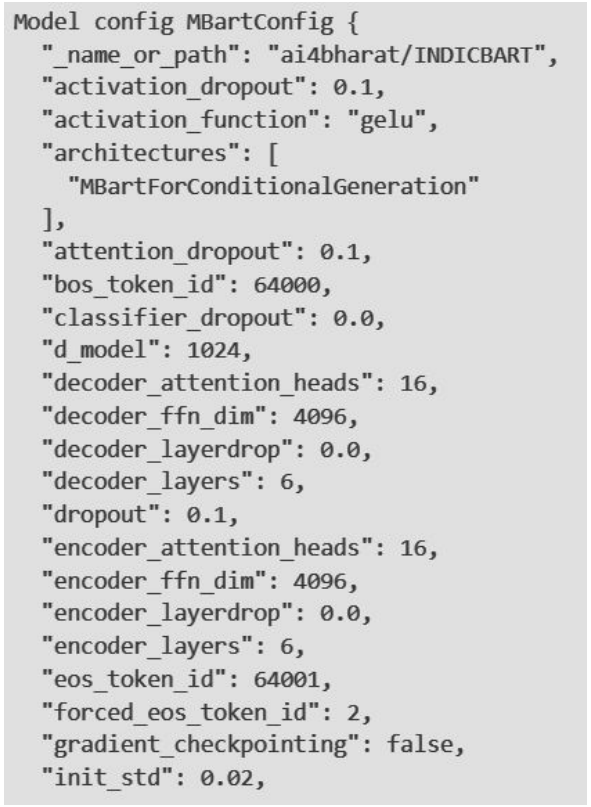
IndicBART model

Model: Fine-tuned `IndicBART` (`ai4bharat/IndicBART`), based on the MBART architecture.
Architecture: Encoder-decoder transformer, ideal for sequence-to-sequence tasks like translation.
Pretraining: Already trained on multiple Indian languages, enabling better transfer for Hinglish generation.
Training observation: Loss consistently decreased, indicating effective learning and good generalization to unseen data.

IndicBART model

Hyperparameters:

- model: indic-bart
- batch_size: 16
- num_train_epochs: 5
- learning_rate: 3e-5
- max_input_length: 128
- max_output_length: 128
- optimizer: AdamW (via Hugging Face Trainer)
- max_steps: 80000

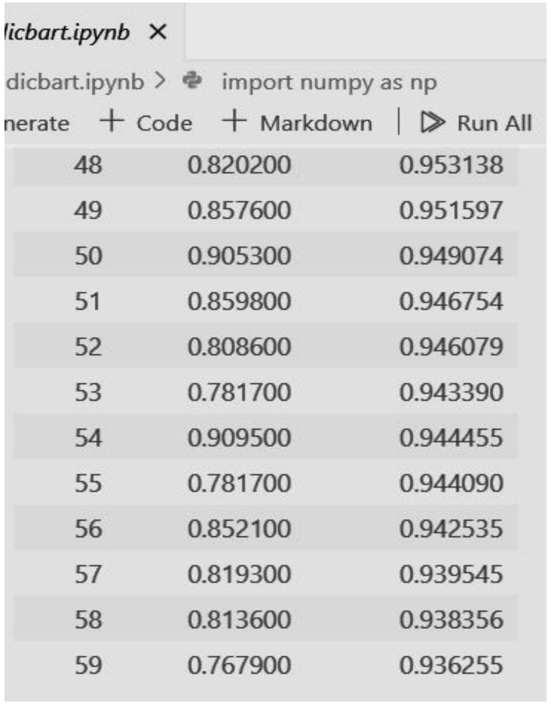


```
Model config MBartConfig {
  "_name_or_path": "ai4bharat/INDICBART",
  "activation_dropout": 0.1,
  "activation_function": "gelu",
  "architectures": [
    "MBartForConditionalGeneration"
  ],
  "attention_dropout": 0.1,
  "bos_token_id": 64000,
  "classifier_dropout": 0.0,
  "d_model": 1024,
  "decoder_attention_heads": 16,
  "decoder_ffn_dim": 4096,
  "decoder_layerdrop": 0.0,
  "decoder_layers": 6,
  "dropout": 0.1,
  "encoder_attention_heads": 16,
  "encoder_ffn_dim": 4096,
  "encoder_layerdrop": 0.0,
  "encoder_layers": 6,
  "eos_token_id": 64001,
  "forced_eos_token_id": 2,
  "gradient_checkpointing": false,
  "init_std": 0.02,
```

IndicBART model

On training with the HinGE dataset the loss values stabilize at 0.9 and don't decrease even with additional epochs.

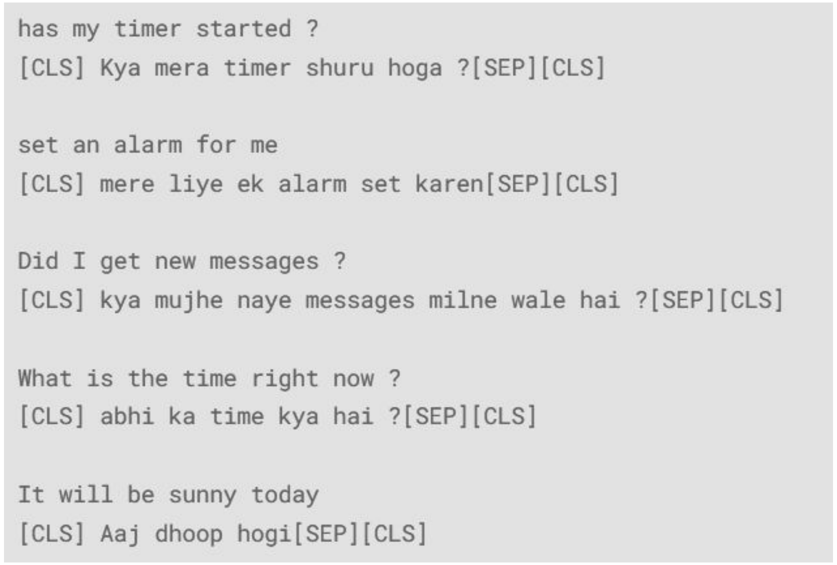
Since it didn't perform well by training on a small dataset we train it using the findnitai/english-to-hinglish dataset from hugging face.


licbart.ipynb ×
dicbart.ipynb > import numpy as np
nerate + Code + Markdown | Run All

48	0.820200	0.953138
49	0.857600	0.951597
50	0.905300	0.949074
51	0.859800	0.946754
52	0.808600	0.946079
53	0.781700	0.943390
54	0.909500	0.944455
55	0.781700	0.944090
56	0.852100	0.942535
57	0.819300	0.939545
58	0.813600	0.938356
59	0.767900	0.936255

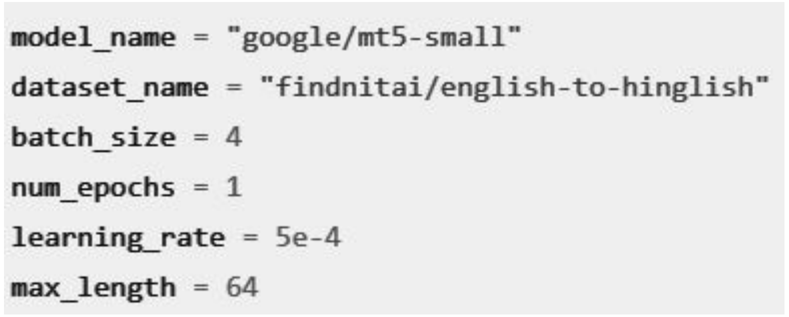
IndicBART model

Training the model on the find-nitai dataset improved its performance but it doesn't generalize well to less common sentences or longer sentences.



```
has my timer started ?
[CLS] Kya mera timer shuru hoga ?[SEP][CLS]

set an alarm for me
[CLS] mere liye ek alarm set karen[SEP][CLS]

Did I get new messages ?
[CLS] kya mujhe naye messages milne wale hai ?[SEP][CLS]

What is the time right now ?
[CLS] abhi ka time kya hai ?[SEP][CLS]

It will be sunny today
[CLS] Aaj dhoop hogi[SEP][CLS]
```

mT5 model



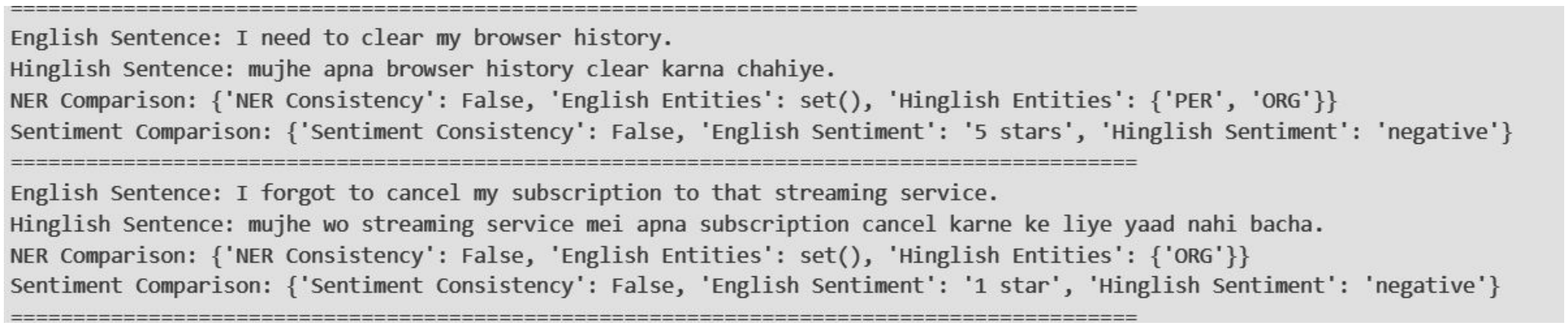
```
model_name = "google/mt5-small"
dataset_name = "findnitai/english-to-hinglish"
batch_size = 4
num_epochs = 1
learning_rate = 5e-4
max_length = 64
```

The mT5 model which is the multilingual equivalent of the T5 model is trained of the findnitai/english-to-hinglish dataset.

The model does well on sentences with n-grams appearing in the training, but for particularly long ones or for totally new sentences, it ends up giving the english input as the output.

Llama model

The Llama-2 model was trained on the findnitai/english-to-hinglish dataset. Amongst the other transformer models it is most consistent in its translations. Though it suffers the same problem of generalizing to completely new sentences, it works better than mT5 and IndicBART.

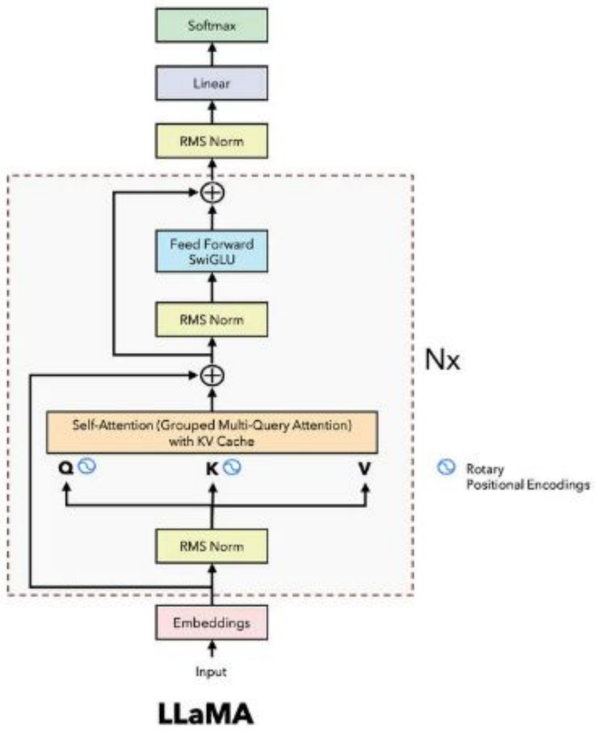


```
================================================================================
English Sentence: I need to clear my browser history.
Hinglish Sentence: mujhe apna browser history clear karna chahiye.
NER Comparison: {'NER Consistency': False, 'English Entities': set(), 'Hinglish Entities': {'PER', 'ORG'}}
Sentiment Comparison: {'Sentiment Consistency': False, 'English Sentiment': '5 stars', 'Hinglish Sentiment': 'negative'}
================================================================================
English Sentence: I forgot to cancel my subscription to that streaming service.
Hinglish Sentence: mujhe wo streaming service mei apna subscription cancel karne ke liye yaad nahi bacha.
NER Comparison: {'NER Consistency': False, 'English Entities': set(), 'Hinglish Entities': {'ORG'}}
Sentiment Comparison: {'Sentiment Consistency': False, 'English Sentiment': '1 star', 'Hinglish Sentiment': 'negative'}
================================================================================
```

Llama model

Hyperparameters used -

- `max_seq_length`: `100`
- `batch_size`: `8`
- `grad_accum_steps`: `2`
- `learning_rate`: `4e-4`
- `log_interval`: `100`
- `bnb_config`, `LoRAConfig`
- `optimizer`: `torch.optim.AdamW`
- `scheduler`: `get_linear_schedule_with_warmup`
- `gradient scaler`: `torch.cuda.amp.GradScaler()`


Softmax
Linear
RMS Norm
Feed Forward SwiGLU
RMS Norm
Nx
Self-Attention (Grouped Multi-Query Attention) with KV Cache
Q
K
V
Rotary Positional Encodings
RMS Norm
Embeddings
Input
LLaMA

Challenges faced and Drawbacks

N-gram Model:

- The model often failed to capture complex sentence structures and long-range dependencies due to its fixed context window (n = 3).
- It relied heavily on surface-level patterns, leading to incorrect or unnatural translations when the test data contained unseen or less frequent trigrams.

LSTM-based Seq2Seq:

- The model initially produced repetitive or generic outputs, largely due to the softmax over a large vocabulary, which favored high-frequency words.
- Despite using attention, it struggled with long sentences and suffered from exposure bias during inference.

Challenges faced and Drawbacks

T5-small with QLoRA:

- Although 4-bit quantization and LoRA adapters made training resource-efficient, the model drastically underperformed in sentence formation and grammar.With many translations just giving specific phrases over and over.
- The large training size from the dataset posed challenges, which were partially addressed using QLoRA.

IndicBART:

- Despite its better alignment with Hindi linguistic structures, computational limitations led to training on a smaller dataset, which affected the model's ability to generalize.
- After training on the larger dataset, the model’s performance improved but it still can’t generalize to longer sentences and less common ones

Challenges faced and Drawbacks

mT5

- Although an improvement from t5 model, it lacked in sentence formation,grammar and for sentences with no similar structures in training data it gave translations devoid of meaning.

Llama

- The model had the most weights, which led to memory constantly running out, and slow training and inference.
- When it translates niche sentences with no similar structures found in training data it resorts to printing the english sentence itself.

Evaluation metrics

GLUECoS

It serves as a set of metrics for a wide range of tasks including both classification and generation. It is used for code switching datasets like Hindi- English, English-Telugu, etc. Here we use NER and Sentiment analysis on the original english input and the translated outputs to get the score.

MIPE

The term MIPE generally encompasses a standard set of evaluation metrics and preprocessing protocols used in Indic NLP generation tasks. Here we use BLEU, chrF, ROUGE-L, SacreBleu and Exact Match scores

Analysis

After evaluating the three transformer models on the GLUECoS and MIPE benchmarks, the following inferences can be drawn:

- LLaMA outperforms both IndicBART and mT5, achieving the best scores on both GLUECoS (classification) and MIPE (generation) tasks.
  This is likely due to its larger parameter count, recent architecture optimizations, and better handling of mixed-lingual contexts.

- IndicBART shows relatively strong MIPE (generation) scores but performs poorly on GLUECoS (classification).
  This could be attributed to its training on Indian languages, aiding generation in Hinglish-like structures, but lacking robustness for classification due to limited pretraining diversity.

Analysis

- mT5 performs similarly to LLaMA on GLUECoS but shows significantly lower MIPE scores.
  This suggests that while mT5 can capture semantic content for classification, it may struggle with generation in low-resource or code-mixed settings, possibly due to tokenization limitations or less fine-grained pretraining on code-mixed corpora.

- Overall Ranking: LLaMA > mT5 > IndicBART. LLaMA is the most robust and balanced across tasks.

Conclusion

For Codemix generation, we started off with the baseline models of Ngram and LSTM.

Then we progressed to mT5 and IndicBART, which gave good translations but weren’t able to generalize well.

Then we moved to the Llama model which gave the best performance of the lot, with better generalization. This inference was made solely by observation.

Then we evaluated the transformer models using GLUECoS and MIPE, from this we could quantitatively say that Llama is the best model.

In conclusion: LLaMA > mT5 > IndicBART > T5 > LSTM > N-gram for English to Hinglish translation