Exercise 1

Character-level recurrent sequence-to-sequence model

By: Daniel Mehta

Inports

In [1]:
```
import numpy as np
import os
from pathlib import Path
import random
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torch.nn.utils.rnn import pad_sequence

from tqdm import tqdm
```

Dataset Path Setup

In [2]:
```
# setting path to dataset
data_dir = Path("fra-eng")
data_path = data_dir/"fra.txt"
```

In [3]:
```
if not data_path.exists():
    raise FileNotFoundError(f"Dataset not found at {data_path}")
```

```
print(f"Dataset located at: {data_path}")
```

```
Dataset located at: fra-eng\fra.txt
```

Data Exploration and Cleaning

In [4]:

```
# Reading the file and split into lines
with open(data_path,"r",encoding="utf-8") as f:
    lines = f.read().strip().split("\n")
```

In [5]:

```
print(f"Total sentence pairs in file: {len(lines)}")
print("Sample lines:")
for i in range(5):
    print(lines[i])
```

```
Total sentence pairs in file: 237838
Sample lines:
Go.     Va !    CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #1158250 (Wittydev)
Go.     Marche. CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #8090732 (Micsmithel)
Go.     En route !      CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #8267435 (felix63)
Go.     Bouge ! CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #9022935 (Micsmithel)
Hi.     Salut ! CC-BY 2.0 (France) Attribution: tatoeba.org #538123 (CM) & #509819 (Aiji)
```

In [6]:

```
# Separating into English and French
pairs =[line.split("\t") for line in lines]
english_sentences =[pair[0] for pair in pairs]
french_sentences =[pair[1] for pair in pairs]
```

In [7]:

```
print("\nExample pair:")
print("EN:",english_sentences[0])
print("FR:",french_sentences[0])
```

```
Example pair:
EN: Go.
FR: Va !
```

Configuration

In [8]:
```
# setting up seed
SEED = 5501
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)

if torch.cuda.is_available():
    torch.cuda.manual_seed_all(SEED)
```

In [9]:
```
# Settubg yo start and end tokens
START_TOKEN="\t"
END_TOKEN="\n"
```

In [10]:
```
#  hyperparameters
batch_size = 64 # Batch size
epochs =100 # epochs of training
latent_dim = 256 #Latent dimensionality of the encoding space
num_samples = 10000  # Num of samples

print(f"batch_size={batch_size}, epochs={epochs}, latent_dim={latent_dim}, num_samples={num_samples}")
print(f"Decoder tokens -> start: {repr(START_TOKEN)}, end: {repr(END_TOKEN)}")
```

```
batch_size=64, epochs=100, latent_dim=256, num_samples=10000
Decoder tokens -> start: '\t', end: '\n'
```

In [11]:
```
# Device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("PyTorch version:",torch.__version__)
print("CUDA available:",torch.cuda.is_available())
if torch.cuda.is_available():
    print("GPU:", torch.cuda.get_device_name(0))
```

```
PyTorch version: 2.7.1+cu118
CUDA available: True
GPU: NVIDIA GeForce RTX 4060
```

In [12]:
```
# Building vocabularies

#  sorted unique characters for each language
input_characters = sorted(list(set("".join(english_sentences))))
target_characters = sorted(list(set("".join(french_sentences))))

#mapping dicts
input_char_to_idx ={char: idx for idx, char in enumerate(input_characters)}
input_idx_to_char ={idx: char for char, idx in input_char_to_idx.items()}

target_char_to_idx ={char: idx for idx, char in enumerate(target_characters)}
target_idx_to_char ={idx: char for char, idx in target_char_to_idx.items()}

# vocabulary sizes
input_vocab_size = len(input_characters)
target_vocab_size = len(target_characters)

print(f"Input vocab size: {input_vocab_size}")
print(f"Target vocab size: {target_vocab_size}")
```

```
Input vocab size: 90
Target vocab size: 113
```

In [13]:
```
# Add start/end tokens to french
french_sentences = [START_TOKEN+s+END_TOKEN for s in french_sentences]

# Limit samples
english_sentences = english_sentences[:num_samples]
french_sentences = french_sentences[:num_samples]
```

In [14]:
```
# Getting the sorted unique characters
input_characters = sorted(list(set("".join(english_sentences))))
target_characters = sorted(list(set("".join(french_sentences))))
```

In [15]:
```
# Map char to index
input_char_to_idx = {char: idx for idx,char in enumerate(input_characters)}
input_idx_to_char = {idx: char for char,idx in input_char_to_idx.items()}
```

```
target_char_to_idx = {char: idx for idx,char in enumerate(target_characters)}
target_idx_to_char = {idx: char for char,idx in target_char_to_idx.items()}
```

In [16]:
```
# Vocabulary sizes
input_vocab_size =len(input_characters)
target_vocab_size =len(target_characters)

print(f"Input vocab size: {input_vocab_size}")
print(f"Target vocab size: {target_vocab_size}")
```

```
Input vocab size: 70
Target vocab size: 91
```

In [17]:
```
# Convertting to index tensors
src_tensors = [torch.tensor([input_char_to_idx[ch] for ch in s],dtype=torch.long)
               for s in english_sentences]
tgt_input_tensors = [torch.tensor([target_char_to_idx[ch] for ch in s[:-1]],dtype=torch.long)
                     for s in french_sentences]
tgt_target_tensors = [torch.tensor([target_char_to_idx[ch] for ch in s[1:]],dtype=torch.long)
                      for s in french_sentences]

# Pad sequences, 0 will be pad index
src_tensors=pad_sequence(src_tensors, batch_first=True,padding_value=0)
tgt_input_tensors=pad_sequence(tgt_input_tensors, batch_first=True,padding_value=0)
tgt_target_tensors=pad_sequence(tgt_target_tensors, batch_first=True,padding_value=0)

# Create Dataloader
dataset = list(zip(src_tensors,tgt_input_tensors,tgt_target_tensors))
train_dataloader =DataLoader(dataset,batch_size=batch_size,shuffle=True)

print(f"Batches in train_dataloader: {len(train_dataloader)}")
```

```
Batches in train_dataloader: 157
```

Building the model

In [18]:
```
embed_dim = 128 # it must be smaller or equal to the latent dim
```

```
class Encoder(nn.Module):
    def __init__(self, input_vocab_size, embed_dim,latent_dim):
        super().__init__()
        self.embedding = nn.Embedding(input_vocab_size, embed_dim)
        self.lstm = nn.LSTM(
            input_size=embed_dim,
            hidden_size=latent_dim,
            num_layers=1,
            batch_first=True
        )


    def forward(self, src_idxs):
        # src_idxs:(batch, src_len)
        embedded =self.embedding(src_idxs) # (batch, src_len, embed_dim)
        outputs,(h,c) =self.lstm(embedded) #outputs not used,keep states
        return h,c

class Decoder(nn.Module):
    def __init__(self, target_vocab_size, embed_dim,latent_dim):
        super().__init__()
        self.embedding =nn.Embedding(target_vocab_size,embed_dim)
        self.lstm = nn.LSTM(
            input_size=embed_dim,
            hidden_size=latent_dim,
            num_layers=1,
            batch_first=True
        )
        self.fc_out =nn.Linear(latent_dim,target_vocab_size)

    def forward(self, tgt_idxs, hidden, cell):
        # tgt_idxs:(batch, tgt_len) with teacher forcing
        embedded = self.embedding(tgt_idxs)# (batch, tgt_len,embed_dim)
        outputs, (h,c) = self.lstm(embedded, (hidden,cell))
        logits = self.fc_out(outputs)#(batch, tgt_len, target_vocab_size)
        return logits, h,c


class Seq2Seq(nn.Module):
    def __init__(self, encoder, decoder):
        super().__init__()
```

```
        self.encoder =encoder
        self.decoder =decoder

    def forward(self, src_idxs, tgt_input_idxs):
        # training forward pass with teacher forcing
        h,c =self.encoder(src_idxs)
        logits,_,_=self.decoder(tgt_input_idxs, h,c)
        return logits


#Instantiate and move to device
encoder = Encoder(input_vocab_size, embed_dim, latent_dim)
decoder = Decoder(target_vocab_size, embed_dim, latent_dim)
model =Seq2Seq(encoder, decoder).to(device)

# Loss and optimizer
PAD_IDX =None
criterion = nn.CrossEntropyLoss(ignore_index=PAD_IDX) if PAD_IDX is not None else nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

print(model)
```

```
Seq2Seq(
  (encoder): Encoder(
    (embedding): Embedding(70, 128)
    (lstm): LSTM(128, 256, batch_first=True)
  )
  (decoder): Decoder(
    (embedding): Embedding(91, 128)
    (lstm): LSTM(128, 256, batch_first=True)
    (fc_out): Linear(in_features=256, out_features=91, bias=True)
  )
)
```

Train model

In [19]:
```
for epoch in range(1, epochs+ 1):
    model.train()
```

```
    total_loss =0

    for src, tgt_input, tgt_target in tqdm(train_dataloader,desc=f"Epoch {epoch}/{epochs}"):
        src = src.to(device)
        tgt_input =tgt_input.to(device)
        tgt_target =tgt_target.to(device)

        optimizer.zero_grad()

        # Forward pass
        output_logits = model(src, tgt_input) #(batch, tgt_len, vocab_size)

        # Reshape for loss, mergeing batch & time dims
        output_logits = output_logits.reshape(-1,target_vocab_size)
        tgt_target = tgt_target.reshape(-1)

        # Compute loss
        loss = criterion(output_logits,tgt_target)
        loss.backward()

        optimizer.step()

        total_loss+=loss.item()

    avg_loss = total_loss/len(train_dataloader)
    print(f"Epoch {epoch} | Loss: {avg_loss:.4f}")

# Save model
torch.save(model.state_dict(),"seq2seq_model.pth")
print("Model saved to seq2seq_model.pth")
```

```
Epoch 1/100: 100%|██████████| 157/157 [00:00<00:00, 171.16it/s]
Epoch 1 | Loss: 0.8973
Epoch 2/100: 100%|██████████| 157/157 [00:00<00:00, 192.48it/s]
Epoch 2 | Loss: 0.5402
Epoch 3/100: 100%|██████████| 157/157 [00:00<00:00, 200.36it/s]
Epoch 3 | Loss: 0.4538
Epoch 4/100: 100%|██████████| 157/157 [00:00<00:00, 201.46it/s]
Epoch 4 | Loss: 0.4037
```

```
Epoch 5/100: 100%|█████████████████████████████████| 157/157 [00:00<00:00, 199.63it/s]
Epoch 5 | Loss: 0.3691
Epoch 6/100: 100%|█████████████████████████████████| 157/157 [00:00<00:00, 200.37it/s]
Epoch 6 | Loss: 0.3401
Epoch 7/100: 100%|█████████████████████████████████| 157/157 [00:00<00:00, 200.77it/s]
Epoch 7 | Loss: 0.3165
Epoch 8/100: 100%|█████████████████████████████████| 157/157 [00:00<00:00, 200.50it/s]
Epoch 8 | Loss: 0.2959
Epoch 9/100: 100%|█████████████████████████████████| 157/157 [00:00<00:00, 201.91it/s]
Epoch 9 | Loss: 0.2775
Epoch 10/100: 100%|████████████████████████████████| 157/157 [00:00<00:00, 198.52it/s]
Epoch 10 | Loss: 0.2605
Epoch 11/100: 100%|████████████████████████████████| 157/157 [00:00<00:00, 199.30it/s]
Epoch 11 | Loss: 0.2456
Epoch 12/100: 100%|████████████████████████████████| 157/157 [00:00<00:00, 200.74it/s]
Epoch 12 | Loss: 0.2317
Epoch 13/100: 100%|████████████████████████████████| 157/157 [00:00<00:00, 202.20it/s]
Epoch 13 | Loss: 0.2192
Epoch 14/100: 100%|████████████████████████████████| 157/157 [00:00<00:00, 201.54it/s]
Epoch 14 | Loss: 0.2064
Epoch 15/100: 100%|████████████████████████████████| 157/157 [00:00<00:00, 197.49it/s]
Epoch 15 | Loss: 0.1951
Epoch 16/100: 100%|████████████████████████████████| 157/157 [00:00<00:00, 203.60it/s]
Epoch 16 | Loss: 0.1853
Epoch 17/100: 100%|████████████████████████████████| 157/157 [00:00<00:00, 202.53it/s]
Epoch 17 | Loss: 0.1747
Epoch 18/100: 100%|████████████████████████████████| 157/157 [00:00<00:00, 201.45it/s]
Epoch 18 | Loss: 0.1649
Epoch 19/100: 100%|████████████████████████████████| 157/157 [00:00<00:00, 197.54it/s]
Epoch 19 | Loss: 0.1561
Epoch 20/100: 100%|████████████████████████████████| 157/157 [00:00<00:00, 199.77it/s]
Epoch 20 | Loss: 0.1478
Epoch 21/100: 100%|████████████████████████████████| 157/157 [00:00<00:00, 204.82it/s]
Epoch 21 | Loss: 0.1400
Epoch 22/100: 100%|████████████████████████████████| 157/157 [00:00<00:00, 202.06it/s]
```

```
Epoch 22 | Loss: 0.1318
Epoch 23/100: 100%|██████████████████████████████████████████████████| 157/157 [00:00<00:00, 202.82it/s]
Epoch 23 | Loss: 0.1250
Epoch 24/100: 100%|██████████████████████████████████████████████████| 157/157 [00:00<00:00, 202.97it/s]
Epoch 24 | Loss: 0.1192
Epoch 25/100: 100%|██████████████████████████████████████████████████| 157/157 [00:00<00:00, 202.64it/s]
Epoch 25 | Loss: 0.1128
Epoch 26/100: 100%|██████████████████████████████████████████████████| 157/157 [00:00<00:00, 203.94it/s]
Epoch 26 | Loss: 0.1071
Epoch 27/100: 100%|██████████████████████████████████████████████████| 157/157 [00:00<00:00, 203.97it/s]
Epoch 27 | Loss: 0.1019
Epoch 28/100: 100%|██████████████████████████████████████████████████| 157/157 [00:00<00:00, 204.06it/s]
Epoch 28 | Loss: 0.0966
Epoch 29/100: 100%|██████████████████████████████████████████████████| 157/157 [00:00<00:00, 201.92it/s]
Epoch 29 | Loss: 0.0913
Epoch 30/100: 100%|██████████████████████████████████████████████████| 157/157 [00:00<00:00, 201.92it/s]
Epoch 30 | Loss: 0.0870
Epoch 31/100: 100%|██████████████████████████████████████████████████| 157/157 [00:00<00:00, 201.47it/s]
Epoch 31 | Loss: 0.0830
Epoch 32/100: 100%|██████████████████████████████████████████████████| 157/157 [00:00<00:00, 198.11it/s]
Epoch 32 | Loss: 0.0795
Epoch 33/100: 100%|██████████████████████████████████████████████████| 157/157 [00:00<00:00, 203.06it/s]
Epoch 33 | Loss: 0.0754
Epoch 34/100: 100%|██████████████████████████████████████████████████| 157/157 [00:00<00:00, 201.48it/s]
Epoch 34 | Loss: 0.0726
Epoch 35/100: 100%|██████████████████████████████████████████████████| 157/157 [00:00<00:00, 201.09it/s]
Epoch 35 | Loss: 0.0700
Epoch 36/100: 100%|██████████████████████████████████████████████████| 157/157 [00:00<00:00, 200.20it/s]
Epoch 36 | Loss: 0.0665
Epoch 37/100: 100%|██████████████████████████████████████████████████| 157/157 [00:00<00:00, 201.53it/s]
Epoch 37 | Loss: 0.0640
Epoch 38/100: 100%|██████████████████████████████████████████████████| 157/157 [00:00<00:00, 201.99it/s]
Epoch 38 | Loss: 0.0615
Epoch 39/100: 100%|██████████████████████████████████████████████████| 157/157 [00:00<00:00, 200.85it/s]
Epoch 39 | Loss: 0.0590
```

```
Epoch 40/100: 100%|██████████████████████████████████| 157/157 [00:00<00:00, 204.06it/s]
Epoch 40 | Loss: 0.0575
Epoch 41/100: 100%|██████████████████████████████████| 157/157 [00:00<00:00, 202.53it/s]
Epoch 41 | Loss: 0.0555
Epoch 42/100: 100%|██████████████████████████████████| 157/157 [00:00<00:00, 202.70it/s]
Epoch 42 | Loss: 0.0538
Epoch 43/100: 100%|██████████████████████████████████| 157/157 [00:00<00:00, 202.91it/s]
Epoch 43 | Loss: 0.0523
Epoch 44/100: 100%|██████████████████████████████████| 157/157 [00:00<00:00, 201.44it/s]
Epoch 44 | Loss: 0.0508
Epoch 45/100: 100%|██████████████████████████████████| 157/157 [00:00<00:00, 200.56it/s]
Epoch 45 | Loss: 0.0487
Epoch 46/100: 100%|██████████████████████████████████| 157/157 [00:00<00:00, 202.62it/s]
Epoch 46 | Loss: 0.0484
Epoch 47/100: 100%|██████████████████████████████████| 157/157 [00:00<00:00, 204.33it/s]
Epoch 47 | Loss: 0.0471
Epoch 48/100: 100%|██████████████████████████████████| 157/157 [00:00<00:00, 203.42it/s]
Epoch 48 | Loss: 0.0457
Epoch 49/100: 100%|██████████████████████████████████| 157/157 [00:00<00:00, 202.84it/s]
Epoch 49 | Loss: 0.0446
Epoch 50/100: 100%|██████████████████████████████████| 157/157 [00:00<00:00, 203.49it/s]
Epoch 50 | Loss: 0.0438
Epoch 51/100: 100%|██████████████████████████████████| 157/157 [00:00<00:00, 203.82it/s]
Epoch 51 | Loss: 0.0430
Epoch 52/100: 100%|██████████████████████████████████| 157/157 [00:00<00:00, 201.16it/s]
Epoch 52 | Loss: 0.0419
Epoch 53/100: 100%|██████████████████████████████████| 157/157 [00:00<00:00, 203.72it/s]
Epoch 53 | Loss: 0.0421
Epoch 54/100: 100%|██████████████████████████████████| 157/157 [00:00<00:00, 202.35it/s]
Epoch 54 | Loss: 0.0411
Epoch 55/100: 100%|██████████████████████████████████| 157/157 [00:00<00:00, 204.24it/s]
Epoch 55 | Loss: 0.0407
Epoch 56/100: 100%|██████████████████████████████████| 157/157 [00:00<00:00, 204.44it/s]
Epoch 56 | Loss: 0.0402
Epoch 57/100: 100%|██████████████████████████████████| 157/157 [00:00<00:00, 204.12it/s]
```

```
Epoch 57 | Loss: 0.0389
Epoch 58/100: 100%|██████████| 157/157 [00:00<00:00, 198.95it/s]
Epoch 58 | Loss: 0.0387
Epoch 59/100: 100%|██████████| 157/157 [00:00<00:00, 203.28it/s]
Epoch 59 | Loss: 0.0396
Epoch 60/100: 100%|██████████| 157/157 [00:00<00:00, 204.13it/s]
Epoch 60 | Loss: 0.0382
Epoch 61/100: 100%|██████████| 157/157 [00:00<00:00, 205.42it/s]
Epoch 61 | Loss: 0.0375
Epoch 62/100: 100%|██████████| 157/157 [00:00<00:00, 203.83it/s]
Epoch 62 | Loss: 0.0376
Epoch 63/100: 100%|██████████| 157/157 [00:00<00:00, 202.79it/s]
Epoch 63 | Loss: 0.0366
Epoch 64/100: 100%|██████████| 157/157 [00:00<00:00, 202.12it/s]
Epoch 64 | Loss: 0.0365
Epoch 65/100: 100%|██████████| 157/157 [00:00<00:00, 204.10it/s]
Epoch 65 | Loss: 0.0359
Epoch 66/100: 100%|██████████| 157/157 [00:00<00:00, 203.98it/s]
Epoch 66 | Loss: 0.0359
Epoch 67/100: 100%|██████████| 157/157 [00:00<00:00, 204.13it/s]
Epoch 67 | Loss: 0.0352
Epoch 68/100: 100%|██████████| 157/157 [00:00<00:00, 201.00it/s]
Epoch 68 | Loss: 0.0346
Epoch 69/100: 100%|██████████| 157/157 [00:00<00:00, 203.14it/s]
Epoch 69 | Loss: 0.0354
Epoch 70/100: 100%|██████████| 157/157 [00:00<00:00, 202.33it/s]
Epoch 70 | Loss: 0.0356
Epoch 71/100: 100%|██████████| 157/157 [00:00<00:00, 202.08it/s]
Epoch 71 | Loss: 0.0349
Epoch 72/100: 100%|██████████| 157/157 [00:00<00:00, 202.03it/s]
Epoch 72 | Loss: 0.0344
Epoch 73/100: 100%|██████████| 157/157 [00:00<00:00, 202.99it/s]
Epoch 73 | Loss: 0.0339
Epoch 74/100: 100%|██████████| 157/157 [00:00<00:00, 202.33it/s]
Epoch 74 | Loss: 0.0337
```

```
Epoch 75/100: 100%|██████████| 157/157 [00:00<00:00, 204.23it/s]
Epoch 75 | Loss: 0.0340
Epoch 76/100: 100%|██████████| 157/157 [00:00<00:00, 204.13it/s]
Epoch 76 | Loss: 0.0332
Epoch 77/100: 100%|██████████| 157/157 [00:00<00:00, 205.19it/s]
Epoch 77 | Loss: 0.0338
Epoch 78/100: 100%|██████████| 157/157 [00:00<00:00, 202.82it/s]
Epoch 78 | Loss: 0.0336
Epoch 79/100: 100%|██████████| 157/157 [00:00<00:00, 203.59it/s]
Epoch 79 | Loss: 0.0331
Epoch 80/100: 100%|██████████| 157/157 [00:00<00:00, 202.03it/s]
Epoch 80 | Loss: 0.0330
Epoch 81/100: 100%|██████████| 157/157 [00:00<00:00, 204.54it/s]
Epoch 81 | Loss: 0.0326
Epoch 82/100: 100%|██████████| 157/157 [00:00<00:00, 204.37it/s]
Epoch 82 | Loss: 0.0323
Epoch 83/100: 100%|██████████| 157/157 [00:00<00:00, 203.85it/s]
Epoch 83 | Loss: 0.0324
Epoch 84/100: 100%|██████████| 157/157 [00:00<00:00, 200.44it/s]
Epoch 84 | Loss: 0.0316
Epoch 85/100: 100%|██████████| 157/157 [00:00<00:00, 205.11it/s]
Epoch 85 | Loss: 0.0320
Epoch 86/100: 100%|██████████| 157/157 [00:00<00:00, 205.09it/s]
Epoch 86 | Loss: 0.0323
Epoch 87/100: 100%|██████████| 157/157 [00:00<00:00, 198.72it/s]
Epoch 87 | Loss: 0.0313
Epoch 88/100: 100%|██████████| 157/157 [00:00<00:00, 205.08it/s]
Epoch 88 | Loss: 0.0310
Epoch 89/100: 100%|██████████| 157/157 [00:00<00:00, 204.64it/s]
Epoch 89 | Loss: 0.0310
Epoch 90/100: 100%|██████████| 157/157 [00:00<00:00, 201.84it/s]
Epoch 90 | Loss: 0.0311
Epoch 91/100: 100%|██████████| 157/157 [00:00<00:00, 204.14it/s]
Epoch 91 | Loss: 0.0314
Epoch 92/100: 100%|██████████| 157/157 [00:00<00:00, 203.14it/s]
```

```
Epoch 92 | Loss: 0.0317
Epoch 93/100: 100%|██████████████████████████████████████████| 157/157 [00:00<00:00, 205.30it/s]
Epoch 93 | Loss: 0.0309
Epoch 94/100: 100%|██████████████████████████████████████████| 157/157 [00:00<00:00, 205.09it/s]
Epoch 94 | Loss: 0.0309
Epoch 95/100: 100%|██████████████████████████████████████████| 157/157 [00:00<00:00, 206.21it/s]
Epoch 95 | Loss: 0.0312
Epoch 96/100: 100%|██████████████████████████████████████████| 157/157 [00:00<00:00, 204.84it/s]
Epoch 96 | Loss: 0.0303
Epoch 97/100: 100%|██████████████████████████████████████████| 157/157 [00:00<00:00, 203.41it/s]
Epoch 97 | Loss: 0.0305
Epoch 98/100: 100%|██████████████████████████████████████████| 157/157 [00:00<00:00, 205.95it/s]
Epoch 98 | Loss: 0.0301
Epoch 99/100: 100%|██████████████████████████████████████████| 157/157 [00:00<00:00, 205.43it/s]
Epoch 99 | Loss: 0.0309
Epoch 100/100: 100%|█████████████████████████████████████████| 157/157 [00:00<00:00, 205.92it/s]
Epoch 100 | Loss: 0.0303
Model saved to seq2seq_model.pth
```

Run inference (sampling)

In [20]:

```python
def decode_sequence(model, src_seq, max_target_len=100):
    model.eval()

    # encode the input sequence to get initial hidden and cell state
    with torch.no_grad():
        h,c=model.encoder(src_seq)

        # Start token as the first decoder input
        start_idx=target_char_to_idx[START_TOKEN]
        decoder_input=torch.tensor([[start_idx]],dtype=torch.long,device=device)

        decoded_chars=[]

        for _ in range(max_target_len):
```

```
        # Pass throughthe  decoder for 1 step
        output,h,c =model.decoder(decoder_input,h,c)

        # Output is (batch=1, seq_len=1, vocab_size) to take last step
        output_char_idx=output.argmax(2).item()
        sampled_char=target_idx_to_char[output_char_idx]

        # Stop if end token
        if sampled_char ==END_TOKEN:
            break

        decoded_chars.append(sampled_char)

        # Use predicted char as next input
        decoder_input = torch.tensor([[output_char_idx]],dtype=torch.long,device=device)

    return "".join(decoded_chars)
```

In [21]:

```
def prepare_input_sentence(sentence):
    #Converting string to tensor of indices and add batch dimension
    idxs = [input_char_to_idx[ch] for ch in sentence]
    tensor = torch.tensor(idxs, dtype=torch.long).unsqueeze(0).to(device)
    return tensor
```

In [22]:

```
for _ in range(20):
    idx = random.randint(0, len(english_sentences) - 1)
    test_sentence = english_sentences[idx]
    input_tensor = prepare_input_sentence(test_sentence)
    translation = decode_sequence(model, input_tensor)

    print(f"EN: {test_sentence}")
    print(f"PREDICTED: {translation}")
    print(f"REFERENCE: {french_sentences[idx]}")
    print("-" * 40)
```

```
EN: I admire you.
PREDICTED: Je t'admire.
REFERENCE:      Je vous admire.

----------------------------------------
EN: Many thanks.
PREDICTED: Merci mille fois !
REFERENCE:      Mille mercis !

----------------------------------------
EN: Are you ready?
PREDICTED: Es-tu prêt ?
REFERENCE:      Est-ce que vous êtes prêts ?

----------------------------------------
EN: Shame on you.
PREDICTED: Honte à toi.
REFERENCE:      Honte à toi.

----------------------------------------
EN: They lied.
PREDICTED: Ils se sont en avance.
REFERENCE:      Ils ont menti.

----------------------------------------
EN: We yawned.
PREDICTED: On a eu de la folle.
REFERENCE:      Nous avons bâillé.

----------------------------------------
EN: Relax.
PREDICTED: lenu un piège.
REFERENCE:      Détends-toi !

----------------------------------------
EN: I'm a girl.
PREDICTED: Je suis paresseux.
REFERENCE:      Je suis une fille.

----------------------------------------
EN: Tom is OK now.
```

```
PREDICTED: Thomas est OK maintenant.
REFERENCE:      Thomas est OK maintenant.

----------------------------------------
EN: I'm speaking.
PREDICTED: Je suis en train de payer.
REFERENCE:      Je suis en train de parler.

----------------------------------------
EN: Can he see us?
PREDICTED: Est-il en mesure de nous voir ?
REFERENCE:      Peut-il nous voir ?

----------------------------------------
EN: Duck!
PREDICTED: iese !
REFERENCE:      À terre !

----------------------------------------
EN: I'll attend.
PREDICTED: Je serai présent.
REFERENCE:      Je serai présent.

----------------------------------------
EN: I can't see.
PREDICTED: Je ne sais pas dehors.
REFERENCE:      Je ne vois rien.

----------------------------------------
EN: It's wrong.
PREDICTED: C'est iroité.
REFERENCE:      C'est faux.

----------------------------------------
EN: I needed it.
PREDICTED: J'ai besoin de colle.
REFERENCE:      J'en avais besoin.

----------------------------------------
EN: What is it?
PREDICTED: Qu'est-ce ?
```

```
REFERENCE:      Qu'est-ce que c'est ?

----------------------------------------
EN: Tom's alive.
PREDICTED: Tom est excitant.
REFERENCE:      Tom est en vie.

----------------------------------------
EN: I must object.
PREDICTED: Il me faut émettre une objection.
REFERENCE:      Il me faut émettre une objection.

----------------------------------------
EN: Are you busy?
PREDICTED: Es-tu occupé ?
REFERENCE:      Êtes-vous occupé ?

----------------------------------------
```

In [ ]:

```

```