```
!pip install datasets==2.18.0 nltk rouge-score sentence-transformers
torchtext evaluate -q
```

```
  Preparing metadata (setup.py) ...
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 510.5/510.5 kB 42.6 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 99.5 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.1/84.1 kB 8.9 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 170.9/170.9 kB 16.9 MB/s eta
0:00:00
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
gcsfs 2025.3.0 requires fsspec==2025.3.0, but you have fsspec 2024.2.0
which is incompatible.
```

```
# ============================================================
# EduGen — Seq2Seq + Attention + Coverage (fixed)
# ScienceQA Question Generation (No Pretrained Transformer)
# ============================================================

# Install dependencies in Colab if needed:
# !pip install datasets==2.18.0 nltk rouge-score sentence-transformers
torchtext evaluate -q

import os
import random
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from datasets import load_dataset
from nltk.tokenize import word_tokenize
from tqdm import tqdm
from rouge_score import rouge_scorer
from sentence_transformers import SentenceTransformer, util
import evaluate
import nltk
nltk.download('punkt')
nltk.download('punkt_tab')
# -----------------------
# Config (demo / full)
# -----------------------
MODE = "full"   # "demo" or "full"
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
MAX_LEN = 50
BATCH_SIZE = 32 if MODE == "full" else 4
```

```
EPOCHS = 40 if MODE == "full" else 2
LR = 2e-4
DATA_FRAC = 0.1 if MODE == "demo" else 1.0
SEED = 42
random.seed(SEED); np.random.seed(SEED); torch.manual_seed(SEED)
```

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

```
<torch._C.Generator at 0x7cde376a8850>
```

```
# -----------------------
# Load dataset
# -----------------------
print("Loading ScienceQA...")
ds = load_dataset("derek-thomas/ScienceQA")
train_ds, val_ds, test_ds = ds["train"], ds["validation"], ds["test"]

def preprocess_sample(ex):
    # using lecture + hint as context (same as your earlier code)
    context = ex.get("lecture", "") or ""
    hint = ex.get("hint", "") or ""
    input_text = f"{context} {hint}".strip()
    output_text = ex.get("question", "") or ""
    return {"input": input_text, "target": output_text}

train = [preprocess_sample(x) for x in train_ds]
val = [preprocess_sample(x) for x in val_ds]
test = [preprocess_sample(x) for x in test_ds]

if MODE == "demo":
    train = train[:int(len(train) * DATA_FRAC)]
    val = val[:500]
    test = test[:500]

print(f"Train size={len(train)}, Val={len(val)}, Test={len(test)}")

# -----------------------
# Tokenization & Vocabulary (simple word-level)
# -----------------------
from collections import Counter
all_text = [t["input"] + " " + t["target"] for t in train]
tokens = [word.lower() for text in all_text for word in
word_tokenize(text)]
vocab_count = Counter(tokens)

# limit vocab to most common N (adjustable)
VOCAB_SIZE_LIMIT = 8000
```

```
vocab_words = [w for w, _ in
vocab_count.most_common(VOCAB_SIZE_LIMIT)]
vocab = ["<PAD>", "<SOS>", "<EOS>", "<UNK>"] + vocab_words
word2idx = {w: i for i, w in enumerate(vocab)}
idx2word = {i: w for w, i in word2idx.items()}

PAD_IDX, SOS_IDX, EOS_IDX, UNK_IDX = 0, 1, 2, 3

def encode(text, max_len=MAX_LEN):
    toks = []
    for w in word_tokenize(text):
        w_low = w.lower()
        toks.append(word2idx.get(w_low, UNK_IDX))
    toks = [SOS_IDX] + toks[: max_len - 2] + [EOS_IDX]
    if len(toks) < max_len:
        toks += [PAD_IDX] * (max_len - len(toks))
    return toks

def decode(tokens):
    words = []
    for i in tokens:
        if i in (PAD_IDX, SOS_IDX, EOS_IDX):
            continue
        words.append(idx2word.get(i, "<UNK>"))
    return " ".join(words)

# -----------------------
# Dataset class / loaders
# -----------------------
class QGDataset(Dataset):
    def __init__(self, data):
        self.data = data
    def __len__(self): return len(self.data)
    def __getitem__(self, idx):
        src = torch.tensor(encode(self.data[idx]["input"]),
dtype=torch.long)
        tgt = torch.tensor(encode(self.data[idx]["target"]),
dtype=torch.long)
        return src, tgt

train_loader = DataLoader(QGDataset(train), batch_size=BATCH_SIZE,
shuffle=True, drop_last=True)
val_loader = DataLoader(QGDataset(val), batch_size=BATCH_SIZE,
shuffle=False)
```

```
Loading ScienceQA...
```

```
/usr/local/lib/python3.12/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 Colab and restart your session.
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(

{"model_id":"0e893e98436c4261b3c1ce0677413b62","version_major":2,"vers
ion_minor":0}

Downloading data: 100%|████████| 377M/377M [00:01<00:00, 301MB/s]
Downloading data: 100%|████████| 126M/126M [00:00<00:00, 198MB/s]
Downloading data: 100%|████████| 122M/122M [00:00<00:00, 200MB/s]

{"model_id":"8e7f051283ea4741a37ec3b4ce67505c","version_major":2,"vers
ion_minor":0}

{"model_id":"588707de300d4dcb9c29d814c1b547e2","version_major":2,"vers
ion_minor":0}

{"model_id":"16b06f9d20004a8cad04819e58e96515","version_major":2,"vers
ion_minor":0}

Train size=12726, Val=4241, Test=4241


# ------------------------
# Model: Encoder (bi-LSTM) + Attention + Decoder (LSTM) with Coverage
# Fixes made:
# - Encoder uses hidden_size = DEC_HID // 2 so concatenation produces
DEC_HID
# - Attention expects encoder_out dim = DEC_HID
# - Decoder LSTM input_size computed as EMB_DIM + DEC_HID +
DEC_HID_coverage_vector (we map coverage to DEC_HID)
# - Coverage vector shapes are squeezed/handled
# ------------------------

class Encoder(nn.Module):
    def __init__(self, vocab_size, embed_dim, dec_hidden_dim):
        """
        We create a bidirectional LSTM whose hidden_size per direction
= dec_hidden_dim // 2
        so that after concatenation the encoder outputs have size =
dec_hidden_dim (matching decoder).
        """
        super().__init__()
        self.embed = nn.Embedding(vocab_size, embed_dim,
padding_idx=PAD_IDX)
        # hidden size per direction
        enc_hidden = dec_hidden_dim // 2
```

```
        self.lstm = nn.LSTM(embed_dim, enc_hidden, batch_first=True,
bidirectional=True)
        self.enc_hidden_size = enc_hidden
        self.output_dim = enc_hidden * 2  # equals dec_hidden_dim

    def forward(self, src):
        # src: (B, T)
        emb = self.embed(src)                       # (B, T, E)
        outputs, (h, c) = self.lstm(emb)            # outputs: (B, T,
2*enc_hidden)
        # cat final forward/back states to match decoder hidden dim
        # h: (num_layers*2, B, enc_hidden) -> take last two
(forward/back) for top layer
        # build decoder initial hidden/state of shape (1, B,
dec_hidden_dim)
        h_cat = torch.cat((h[-2], h[-1]), dim=1).unsqueeze(0)  # (1,
B, 2*enc_hidden)
        c_cat = torch.cat((c[-2], c[-1]), dim=1).unsqueeze(0)
        return outputs, (h_cat, c_cat)  # outputs
(B,T,dec_hidden_dim), hidden (1,B,dec_hidden_dim)

class Attention(nn.Module):
    def __init__(self, dec_hidden_dim):
        super().__init__()
        # encoder_out dim == dec_hidden_dim
        self.W_enc = nn.Linear(dec_hidden_dim, dec_hidden_dim,
bias=False)
        self.W_dec = nn.Linear(dec_hidden_dim, dec_hidden_dim,
bias=False)
        self.v = nn.Linear(dec_hidden_dim, 1, bias=False)

    def forward(self, encoder_out, dec_hidden):
        # encoder_out: (B, T, dec_hidden_dim)
        # dec_hidden: (1, B, dec_hidden_dim)
        dec_hidden = dec_hidden.permute(1,0,2)          # (B, 1,
dec_hidden_dim)
        score = torch.tanh(self.W_enc(encoder_out) +
self.W_dec(dec_hidden))  # (B, T, dec_hidden_dim)
        attn_unnorm = self.v(score).squeeze(2)          # (B, T)
        attn = torch.softmax(attn_unnorm, dim=1)        # (B, T)
        context = torch.bmm(attn.unsqueeze(1), encoder_out).squeeze(1)
# (B, dec_hidden_dim)
        return context, attn  # attn (B,T), context (B,dec_hidden_dim)

class Decoder(nn.Module):
    def __init__(self, vocab_size, embed_dim, dec_hidden_dim):
        super().__init__()
        self.embed = nn.Embedding(vocab_size, embed_dim,
padding_idx=PAD_IDX)
        self.attn = Attention(dec_hidden_dim)
```

```
        # coverage projection: map scalar coverage -> dec_hidden_dim
vector via linear
        self.coverage_proj = nn.Linear(1, dec_hidden_dim)
        # LSTM input size: embed_dim + context_dim + coverage_vec_dim
        lstm_input_size = embed_dim + dec_hidden_dim + dec_hidden_dim
        self.lstm = nn.LSTM(lstm_input_size, dec_hidden_dim,
batch_first=True)
        self.out = nn.Linear(dec_hidden_dim, vocab_size)

    def forward_step(self, input_tok, hidden, cell, encoder_out,
prev_coverage):
        # input_tok: (B,) token ids
        emb = self.embed(input_tok).unsqueeze(1)  # (B,1,embed_dim)
        context, attn = self.attn(encoder_out, hidden)  # context
(B,dec_h), attn (B,T)
        # compute coverage: prev_coverage (B,T) -> sum over positions
-> scalar per sample OR vector
        # We'll compute coverage per position for later coverage loss;
here we create a single aggregated scalar
        # coverage_scalar = torch.sum(prev_coverage, dim=1,
keepdim=True)  # (B,1)
        # But better: we map the aggregated coverage scalar to
dec_hidden_dim
        coverage_scalar = (prev_coverage.sum(dim=1, keepdim=True))  #
(B,1)
        cov_vec =
torch.tanh(self.coverage_proj(coverage_scalar)).unsqueeze(1)  #
(B,1,dec_hidden_dim)
        lstm_input = torch.cat([emb, context.unsqueeze(1), cov_vec],
dim=2)  # (B,1,embed + dec + dec)
        output, (hidden, cell) = self.lstm(lstm_input, (hidden, cell))
# output (B,1,dec_h)
        logits = self.out(output.squeeze(1))  # (B, vocab)
        return logits, hidden, cell, attn

    def forward(self, tgt, hidden, cell, encoder_out, prev_coverage):
        # single-step wrapper to be compatible with seq loop
        return self.forward_step(tgt, hidden, cell, encoder_out,
prev_coverage)

class Seq2Seq(nn.Module):
    def __init__(self, encoder, decoder, device):
        super().__init__()
        self.enc = encoder
        self.dec = decoder
        self.device = device

    def forward(self, src, tgt, teacher_forcing=0.5):
        # src: (B, S), tgt: (B, T)
        batch_size, tgt_len = tgt.shape
```

```
        vocab_size = self.dec.out.out_features if hasattr(self.dec,
"out") else self.dec.out.out_features
        vocab_size = self.dec.out.out_features
        outputs = torch.zeros(batch_size, tgt_len, vocab_size,
device=self.device)

        encoder_out, (hidden, cell) = self.enc(src)  # encoder_out (B,
S, dec_hidden)
        # initialize coverage: zeros per position
        coverage = torch.zeros(batch_size, encoder_out.size(1),
device=self.device)  # (B, S)
        input_tok = tgt[:, 0]  # SOS

        for t in range(1, tgt_len):
            logits, hidden, cell, attn =
self.dec.forward_step(input_tok, hidden, cell, encoder_out, coverage)
            outputs[:, t, :] = logits
            # update coverage: add current attention
            coverage = coverage + attn
            teacher_force = random.random() < teacher_forcing
            top1 = logits.argmax(1)
            input_tok = tgt[:, t] if teacher_force else top1

        return outputs, coverage  # return coverage so coverage loss
can be computed outside if desired

# ----------------------
# Initialize model (ensure dims align)
# ----------------------
DEC_HID = 256     # decoder hidden dimension (final)
EMB_DIM = 200
enc = Encoder(len(vocab), EMB_DIM, DEC_HID)
dec = Decoder(len(vocab), EMB_DIM, DEC_HID)
model = Seq2Seq(enc, dec, DEVICE).to(DEVICE)

# Loss / optimizer
criterion = nn.CrossEntropyLoss(ignore_index=PAD_IDX)
optimizer = optim.Adam(model.parameters(), lr=LR)

# ----------------------
# Coverage loss helper
# ----------------------
def coverage_loss_calc(attn_history, epsilon=1e-8):
    # attn_history: list of attn vectors per time step (each (B, S)) -
> stack to (B, T-1, S)
    # For simplicity in this training loop we will accumulate coverage
externally; but here's a utility
    # Not used in this simplified training (we compute coverage via
returned coverage)
    attn_stack = torch.stack(attn_history, dim=1)  # (B, T-1, S)
```

```
    cov = torch.cumsum(attn_stack, dim=1)
    min_vals = torch.min(attn_stack, cov - attn_stack + epsilon)
    return torch.sum(min_vals)

# ----------------------
# Training / Evaluation loops
# ----------------------
def train_epoch(model, loader):
    model.train()
    total_loss = 0.0
    for src, tgt in tqdm(loader, desc="Train"):
        src = src.to(DEVICE); tgt = tgt.to(DEVICE)
        optimizer.zero_grad()
        outputs, coverage = model(src, tgt, teacher_forcing=0.5)  # 
outputs (B, T, V)
        # compute CE ignoring first token
        logits = outputs[:, 1:, :].reshape(-1, outputs.size(-1))
        targets = tgt[:, 1:].reshape(-1)
        loss_ce = criterion(logits, targets)
        # compute simple coverage loss penalty: encourage attention 
mass not to repeat too much
        # coverage: (B, S) final accumulated attention; penalize large 
values
        # scaled coverage loss = mean over batch of 
sum(square(coverage))
        cov_loss = torch.mean(torch.sum(coverage * coverage, dim=1))
        # weight coverage (tune lambda)
        LAMBDA_COV = 1e-2
        loss = loss_ce + LAMBDA_COV * cov_loss
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 2.0)
        optimizer.step()
        total_loss += loss.item()
    return total_loss / len(loader)

def evaluate_model(model, loader):
    model.eval()
    losses = []
    with torch.no_grad():
        for src, tgt in tqdm(loader, desc="Val"):
            src = src.to(DEVICE); tgt = tgt.to(DEVICE)
            outputs, _ = model(src, tgt, teacher_forcing=0.0)
            logits = outputs[:, 1:, :].reshape(-1, outputs.size(-1))
            targets = tgt[:, 1:].reshape(-1)
            loss_ce = criterion(logits, targets)
            losses.append(loss_ce.item())
    return float(np.mean(losses))


# ----------------------
```

```python
# Run training
# ----------------------
for ep in range(1, EPOCHS + 1):
    tr_loss = train_epoch(model, train_loader)
    val_loss = evaluate_model(model, val_loader)
    print(f"Epoch {ep}/{EPOCHS} | Train Loss={tr_loss:.4f} | Val
Loss={val_loss:.4f}")

# Save weights
os.makedirs("models", exist_ok=True)
torch.save(model.state_dict(), "models/seq2seq_attn_cov.pt")
print("Model saved to models/seq2seq_attn_cov.pt")

# ----------------------
# Generation (greedy) helper
# ----------------------
def generate(model, text, max_len=MAX_LEN):
    model.eval()
    src = torch.tensor([encode(text)], dtype=torch.long).to(DEVICE)
    encoder_out, (hidden, cell) = model.enc(src)
    coverage = torch.zeros(1, encoder_out.size(1), device=DEVICE)
    input_tok = torch.tensor([SOS_IDX], device=DEVICE)
    generated = []
    with torch.no_grad():
        for _ in range(max_len):
            logits, hidden, cell, attn =
model.dec.forward_step(input_tok, hidden, cell, encoder_out, coverage)
            coverage = coverage + attn  # update coverage
            pred = logits.argmax(1)   # greedy
            w = pred.item()
            if w == EOS_IDX:
                break
            generated.append(idx2word.get(w, "<UNK>"))
            input_tok = pred
    return " ".join(generated)
```

```
Train: 100%|██████████| 397/397 [01:07<00:00,  5.91it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.03it/s]

Epoch 1/40 | Train Loss=5.8638 | Val Loss=4.5926

Train: 100%|██████████| 397/397 [01:05<00:00,  6.03it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.01it/s]

Epoch 2/40 | Train Loss=4.6111 | Val Loss=3.7525

Train: 100%|██████████| 397/397 [01:04<00:00,  6.12it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.24it/s]

Epoch 3/40 | Train Loss=3.8101 | Val Loss=3.1589
```

```
Train: 100%|██████████| 397/397 [01:04<00:00,  6.14it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.15it/s]

Epoch 4/40 | Train Loss=3.3173 | Val Loss=2.8453

Train: 100%|██████████| 397/397 [01:04<00:00,  6.16it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.27it/s]

Epoch 5/40 | Train Loss=3.0145 | Val Loss=2.6645

Train: 100%|██████████| 397/397 [01:04<00:00,  6.15it/s]
Val: 100%|██████████| 133/133 [00:08<00:00, 14.92it/s]

Epoch 6/40 | Train Loss=2.8149 | Val Loss=2.5329

Train: 100%|██████████| 397/397 [01:05<00:00,  6.11it/s]
Val: 100%|██████████| 133/133 [00:08<00:00, 15.51it/s]

Epoch 7/40 | Train Loss=2.6596 | Val Loss=2.4520

Train: 100%|██████████| 397/397 [01:04<00:00,  6.18it/s]
Val: 100%|██████████| 133/133 [00:08<00:00, 15.44it/s]

Epoch 8/40 | Train Loss=2.5286 | Val Loss=2.3952

Train: 100%|██████████| 397/397 [01:04<00:00,  6.19it/s]
Val: 100%|██████████| 133/133 [00:08<00:00, 14.94it/s]

Epoch 9/40 | Train Loss=2.4347 | Val Loss=2.3621

Train: 100%|██████████| 397/397 [01:03<00:00,  6.20it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.77it/s]

Epoch 10/40 | Train Loss=2.3461 | Val Loss=2.3088

Train: 100%|██████████| 397/397 [01:04<00:00,  6.17it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.27it/s]

Epoch 11/40 | Train Loss=2.2743 | Val Loss=2.2814

Train: 100%|██████████| 397/397 [01:04<00:00,  6.20it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.31it/s]

Epoch 12/40 | Train Loss=2.2037 | Val Loss=2.2911

Train: 100%|██████████| 397/397 [01:03<00:00,  6.22it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.32it/s]

Epoch 13/40 | Train Loss=2.1533 | Val Loss=2.2388

Train: 100%|██████████| 397/397 [01:03<00:00,  6.21it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.26it/s]

Epoch 14/40 | Train Loss=2.1038 | Val Loss=2.1997
```

```
Train: 100%|██████████| 397/397 [01:03<00:00,  6.24it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.21it/s]

Epoch 15/40 | Train Loss=2.0703 | Val Loss=2.1993

Train: 100%|██████████| 397/397 [01:03<00:00,  6.23it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.25it/s]

Epoch 16/40 | Train Loss=2.0168 | Val Loss=2.1966

Train: 100%|██████████| 397/397 [01:03<00:00,  6.25it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.48it/s]

Epoch 17/40 | Train Loss=1.9800 | Val Loss=2.2166

Train: 100%|██████████| 397/397 [01:03<00:00,  6.23it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.38it/s]

Epoch 18/40 | Train Loss=1.9534 | Val Loss=2.1814

Train: 100%|██████████| 397/397 [01:03<00:00,  6.26it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.30it/s]

Epoch 19/40 | Train Loss=1.9238 | Val Loss=2.1478

Train: 100%|██████████| 397/397 [01:03<00:00,  6.23it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.56it/s]

Epoch 20/40 | Train Loss=1.8898 | Val Loss=2.2192

Train: 100%|██████████| 397/397 [01:04<00:00,  6.18it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.69it/s]

Epoch 21/40 | Train Loss=1.8718 | Val Loss=2.1634

Train: 100%|██████████| 397/397 [01:05<00:00,  6.10it/s]
Val: 100%|██████████| 133/133 [00:08<00:00, 15.20it/s]

Epoch 22/40 | Train Loss=1.8451 | Val Loss=2.1282

Train: 100%|██████████| 397/397 [01:04<00:00,  6.14it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.46it/s]

Epoch 23/40 | Train Loss=1.8196 | Val Loss=2.1687

Train: 100%|██████████| 397/397 [01:04<00:00,  6.11it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.10it/s]

Epoch 24/40 | Train Loss=1.7912 | Val Loss=2.2079

Train: 100%|██████████| 397/397 [01:04<00:00,  6.19it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.16it/s]

Epoch 25/40 | Train Loss=1.7782 | Val Loss=2.1579
```

```
Train: 100%|██████████| 397/397 [01:04<00:00,  6.20it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.38it/s]

Epoch 26/40 | Train Loss=1.7573 | Val Loss=2.1839

Train: 100%|██████████| 397/397 [01:04<00:00,  6.19it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.19it/s]

Epoch 27/40 | Train Loss=1.7323 | Val Loss=2.1787

Train: 100%|██████████| 397/397 [01:04<00:00,  6.18it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.29it/s]

Epoch 28/40 | Train Loss=1.7208 | Val Loss=2.2147

Train: 100%|██████████| 397/397 [01:03<00:00,  6.22it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.29it/s]

Epoch 29/40 | Train Loss=1.7036 | Val Loss=2.2132

Train: 100%|██████████| 397/397 [01:04<00:00,  6.16it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.69it/s]

Epoch 30/40 | Train Loss=1.6908 | Val Loss=2.2215

Train: 100%|██████████| 397/397 [01:04<00:00,  6.18it/s]
Val: 100%|██████████| 133/133 [00:08<00:00, 15.50it/s]

Epoch 31/40 | Train Loss=1.6706 | Val Loss=2.2214

Train: 100%|██████████| 397/397 [01:04<00:00,  6.19it/s]
Val: 100%|██████████| 133/133 [00:08<00:00, 15.37it/s]

Epoch 32/40 | Train Loss=1.6620 | Val Loss=2.2092

Train: 100%|██████████| 397/397 [01:04<00:00,  6.19it/s]
Val: 100%|██████████| 133/133 [00:08<00:00, 15.43it/s]

Epoch 33/40 | Train Loss=1.6421 | Val Loss=2.2245

Train: 100%|██████████| 397/397 [01:04<00:00,  6.20it/s]
Val: 100%|██████████| 133/133 [00:08<00:00, 15.42it/s]

Epoch 34/40 | Train Loss=1.6198 | Val Loss=2.2287

Train: 100%|██████████| 397/397 [01:04<00:00,  6.19it/s]
Val: 100%|██████████| 133/133 [00:08<00:00, 15.04it/s]

Epoch 35/40 | Train Loss=1.6092 | Val Loss=2.2420

Train: 100%|██████████| 397/397 [01:04<00:00,  6.18it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.58it/s]

Epoch 36/40 | Train Loss=1.6066 | Val Loss=2.2737
```

```
Train: 100%|██████████| 397/397 [01:04<00:00,  6.17it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.04it/s]

Epoch 37/40 | Train Loss=1.5938 | Val Loss=2.2147

Train: 100%|██████████| 397/397 [01:04<00:00,  6.17it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.22it/s]

Epoch 38/40 | Train Loss=1.5789 | Val Loss=2.2663

Train: 100%|██████████| 397/397 [01:04<00:00,  6.17it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.32it/s]

Epoch 39/40 | Train Loss=1.5575 | Val Loss=2.2931

Train: 100%|██████████| 397/397 [01:04<00:00,  6.19it/s]
Val: 100%|██████████| 133/133 [00:09<00:00, 14.33it/s]

Epoch 40/40 | Train Loss=1.5463 | Val Loss=2.2867
Model saved to models/seq2seq_attn_cov.pt

# ------------------------
# Quick sample generations
# ------------------------
print("\n--- Sample generations ---")
print_samples = random.sample(train, min(10, len(train)))  # print 10
samples
save_samples = random.sample(train, min(50, len(train)))   # save 50
samples
pairs = []

# Print 10 sample outputs
for s in print_samples:
    inp = s["input"]
    gen_q = generate(model, inp)
    print("Input (truncated):", inp[:200])
    print("Generated:", gen_q)
    print("-" * 60)

# Collect 50 samples for saving
for s in save_samples:
    inp = s["input"]
    gen_q = generate(model, inp)
    pairs.append((inp[:200], gen_q))


--- Sample generations ---
Input (truncated): Chemical changes and physical changes are two
common ways matter can change.
In a chemical change, the type of matter changes. The types of matter
before and after a chemical change are always differe
Generated: what do these two changes have in common ? a piece of glass
```

```
turning brown a piece of wood
------------------------------------------------------------
Input (truncated):
Generated: what is the capital of the ?
------------------------------------------------------------
Input (truncated): Measurements are written with both a number and a
unit. The unit comes after the number. The unit shows what the number
means.
Mass is a measurement of how much matter something contains.
There are ma
Generated: what is the mass of a full ?
------------------------------------------------------------
Input (truncated): Vertebrates and invertebrates are both groups of
animals.
A vertebrate has a backbone. The backbone is made of many bones in an
animal's back. A vertebrate's backbone helps connect the different
parts
Generated: select the vertebrate .
------------------------------------------------------------
Input (truncated): Fossils are the remains of organisms that lived
long ago. Scientists look at fossils to learn about the traits of
ancient organisms. Often, scientists compare fossils to modern
organisms.
Some ancient
Generated: which statement is supported by these pictures ?
------------------------------------------------------------
Input (truncated): Present tense verbs tell you about something that
is happening now.
Most present-tense verbs are regular. They have no ending, or they end
in -s or -es.
Two verbs are irregular in the present tense, t
Generated: which tense does the sentence use ? the will will the the
in the .
------------------------------------------------------------
Input (truncated): Figures of speech are words or phrases that use
language in a nonliteral or unusual way. They can make writing more
expressive.
A euphemism is a polite or indirect expression that is used to de-
emphas
Generated: which figure of speech is used in this text ? the 's
boutique claims to have a something for everyone , but it is generally
understood that he target market is women of a certain age .
------------------------------------------------------------
Input (truncated): Present tense verbs tell you about something that
is happening now.
Most present-tense verbs are regular. They have no ending, or they end
in -s or -es.
Two verbs are irregular in the present tense, t
Generated: which tense does the sentence use ? the will will the the
```

```
in the .
------------------------------------------------------------
Input (truncated): A force is a push or a pull.
A force can make an object start moving or stop an object that is
moving. A force can also make an object speed up, slow down, or change
direction.
Forces can be different
Generated: the donkeys move the carts at the same speed . which cart
is pulled with a larger force ?
------------------------------------------------------------
Input (truncated): A clause is a group of words that contains both a
subject and a predicate.
An independent clause is a complete thought that can stand alone as a
sentence. A dependent clause (or subordinate clause) is
Generated: which type of sentence is this ? an avid reader , and
attends weekly book club meetings , and he finishes several novels
every month .
------------------------------------------------------------


# -----------------------
# Evaluation metrics: Perplexity (via model CE), BERTScore, BLEU,
ROUGE, Cosine (SBERT), Distinct-n, Entropy
# -----------------------
bleu = evaluate.load("bleu")
rouge_obj = rouge_scorer.RougeScorer(['rouge1','rouge2','rougeL'],
use_stemmer=True)
sbert = SentenceTransformer('all-MiniLM-L6-v2', device=DEVICE)

def ppl_from_loss(loss):
    # approx perplexity from average cross-entropy loss
    return float(np.exp(loss))

def evaluate_generated_set(model, data_list, n_samples=200):
    refs, gens = [], []
    sample_list = data_list if len(data_list) <= n_samples else
random.sample(data_list, n_samples)
    # gather CE/val-loss-like for perplexity we compute with LM (here
we approximate using model CE on generated pairs)
    ce_losses = []
    for d in sample_list:
        inp = d["input"]
        ref = d["target"]
        gen = generate(model, inp)
        refs.append(ref)
        gens.append(gen)
    # BLEU
    bleu_score = bleu.compute(predictions=gens, references=refs)
["bleu"]
    # ROUGE
```

```
    r1 = np.mean([rouge_obj.score(r, g)['rouge1'].fmeasure for r, g in
zip(refs, gens)])
    r2 = np.mean([rouge_obj.score(r, g)['rouge2'].fmeasure for r, g in
zip(refs, gens)])
    rl = np.mean([rouge_obj.score(r, g)['rougeL'].fmeasure for r, g in
zip(refs, gens)])
    # BERT / SBERT cosine similarity
    emb_refs = sbert.encode(refs, convert_to_tensor=True)
    emb_gens = sbert.encode(gens, convert_to_tensor=True)
    cos_sims = util.cos_sim(emb_refs, emb_gens).diagonal()  # per-
sample cosine
    cos_mean = float(torch.mean(cos_sims).cpu().numpy())
    # Distinct-n and entropy
    import nltk
    from collections import Counter
    def distinct_n(texts, n):
        total = 0
        uniq = set()
        for t in texts:
            toks = word_tokenize(t.lower())
            ngrams = list(nltk.ngrams(toks, n)) if len(toks) >= n else
[]
            total += len(ngrams)
            uniq.update(ngrams)
        return (len(uniq) / total) if total > 0 else 0.0
    def token_entropy(texts):
        cnt = Counter()
        for t in texts:
            cnt.update(word_tokenize(t.lower()))
        total = sum(cnt.values())
        if total == 0:
            return 0.0
        p = np.array([v / total for v in cnt.values()])
        return float(-np.sum(p * np.log(p + 1e-12)))
    distinct1 = distinct_n(gens, 1)
    distinct2 = distinct_n(gens, 2)
    ent = token_entropy(gens)
    # Perplexity estimate: approximate by computing CE of generated ->
reference using model? Simpler: use mean token cross-entropy between
ref & gen via evaluate BLEU/LR not exact.
    # We'll compute an approximate perplexity using SBERT similarity
inversion (not perfect). For reliable PPL use an LM — omitted to keep
this self-contained.
    metrics = {
        "BLEU": bleu_score,
        "ROUGE-1": r1, "ROUGE-2": r2, "ROUGE-L": rl,
        "Cosine-SBERT": cos_mean,
        "distinct-1": distinct1, "distinct-2": distinct2,
        "entropy": ent,
```

```
        "samples": len(gens)
    }
    return metrics, list(zip(gens, refs))

print("\nEvaluating on validation set (sampled)...")
val_metrics, pairs = evaluate_generated_set(model, val, n_samples=200)
print("Validation metrics:", val_metrics)
```

```
Evaluating on validation set (sampled)...
Validation metrics: {'BLEU': 0.46780426473036296, 'ROUGE-1':
np.float64(0.7627179583657225), 'ROUGE-2':
np.float64(0.6608251408829565), 'ROUGE-L':
np.float64(0.7570458171802241), 'Cosine-SBERT': 0.694872260093689,
'distinct-1': 0.1606125356125356, 'distinct-2': 0.34700920245398775,
'entropy': 4.904271775357443, 'samples': 200}
```

```
# Save 50 samples to file
os.makedirs("outputs", exist_ok=True)
with open("outputs/generated_val_samples.txt", "w", encoding="utf-8")
as f:
    for inp, gen in pairs:
        f.write("Input (truncated): " + inp + "\n")
        f.write("Generated: " + gen + "\n")
        f.write("-" * 60 + "\n\n")

print("Saved 50 generated samples to
outputs/generated_val_samples.txt")
```

```
Saved 50 generated samples to outputs/generated_val_samples.txt
```