Step 1 — Install Dependencies

```
!pip install sacrebleu -q
print("Dependencies installed")
```

Step 2 — Upload Dataset

```
from google.colab import files
uploaded = files.upload()
DATA_PATH = list(uploaded.keys())[0]
print(f"Uploaded: {DATA_PATH}")
with open(DATA_PATH) as f:
    n = sum(1 for _ in f)
print(f"Total lines: {n:,}")
```

```
# Alternative: load from Google Drive
# from google.colab import drive
# drive.mount('/content/drive')
# DATA_PATH = '/content/drive/MyDrive/spa.txt'
```

Step 3 — Hyperparameters

```
MAX_SAMPLES = 10_000
MIN_FREQ    = 2
MAX_LEN     = 50
EMBED_DIM   = 256
HIDDEN_DIM  = 512
NUM_LAYERS  = 2
DROPOUT     = 0.3
BATCH_SIZE  = 64
EPOCHS      = 20
LR          = 1e-3
CLIP        = 1.0
SEED        = 42

import random, math, time, os, re
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import torch, torch.nn as nn, torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pack_padded_sequence, pad_packed_sequence
from collections import Counter
from typing import List, Tuple, Optional

# random.seed(SEED); np.random.seed(SEED); torch.manual_seed(SEED)
# DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
# print(f"Device : {DEVICE}")
# if DEVICE.type == "cuda":
#     print(f"GPU    : {torch.cuda.get_device_name(0)}")
```

Step 4 — Data Preprocessing

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

def normalize_string(s):
    s = s.lower().strip()
    s = re.sub(r'([.!?\u00bf\u00a1])', r' \1', s)
    s = re.sub(r'[^a-zA-Z.!?\u00bf\u00a1\u00e1\u00e9\u00ed\u00f3\u00fa\u00fc\u00f1\']+',
    return s.strip()

def read_pairs(path, max_samples=10_000, seed=42):
    pairs = []
    with open(path, encoding='utf-8') as f:
        for line in f:
            parts = line.strip().split('\t')
            if len(parts) >= 2:
                s = normalize_string(parts[0]); t = normalize_string(parts[1])
                if s and t: pairs.append((s, t))
    random.seed(seed); random.shuffle(pairs)
    return pairs[:max_samples]

def split_pairs(pairs, tr=0.8, vr=0.1):
    n=len(pairs); nt=int(tr*n); nv=int(vr*n)
    return pairs[:nt], pairs[nt:nt+nv], pairs[nt+nv:]

class Vocabulary:
    def __init__(self, name):
        self.name = name
        self.word2idx = {"<pad>":0,"<sos>":1,"<eos>":2,"<unk>":3}
        self.idx2word = {v:k for k,v in self.word2idx.items()}
        self.freq = Counter()
    def build(self, sents, min_freq=2):
        for s in sents: self.freq.update(s.split())
        for w,c in sorted(self.freq.items()):
            if c>=min_freq and w not in self.word2idx:
                i=len(self.word2idx); self.word2idx[w]=i; self.idx2word[i]=w
    def encode(self, s): return [self.word2idx.get(w,UNK_IDX) for w in s.split()]
    def decode(self, ids, skip=True):
        sp={0,1,2} if skip else set()
        return ' '.join(self.idx2word.get(i,'<unk>') for i in ids if i not in sp)
    def __len__(self): return len(self.word2idx)

class TranslationDataset(Dataset):
    def __init__(self, pairs, sv, tv, ml=50):
        self.data=[]
        for src,tgt in pairs:
            s=[SOS_IDX]+sv.encode(src)+[EOS_IDX]; t=[SOS_IDX]+tv.encode(tgt)+[EOS_IDX]
            if len(s)<=ml and len(t)<=ml: self.data.append((s,t))
    def __len__(self): return len(self.data)
    def __getitem__(self,i): return self.data[i]

def collate_fn(batch):
    ss,ts=zip(*batch)
    sl=torch.tensor([len(s) for s in ss]); tl=torch.tensor([len(t) for t in ts])
    sp=torch.zeros(len(ss),sl.max(),dtype=torch.long)
    tp=torch.zeros(len(ts),tl.max(),dtype=torch.long)
    for i,(s,t) in enumerate(zip(ss,ts)):
        sp[i,:len(s)]=torch.tensor(s); tp[i,:len(t)]=torch.tensor(t)
    sl,order=sl.sort(descending=True)
    return sp[order],tp[order],sl,tl[order]

pairs = read_pairs(DATA_PATH, MAX_SAMPLES)
train_pairs, val_pairs, test_pairs = split_pairs(pairs)
print(f"Train: {len(train_pairs):,} | Val: {len(val_pairs):,} | Test: {len(test_pairs):,

src_vocab=Vocabulary("english"); tgt_vocab=Vocabulary("spanish")
src_vocab.build([p[0] for p in train_pairs], MIN_FREQ)
tgt_vocab.build([p[1] for p in train_pairs], MIN_FREQ)
print(f"Src vocab: {len(src_vocab):,} | Tgt vocab: {len(tgt_vocab):,}")
```

```
print(f"Src vocab: {len(src_vocab):,} | Tgt vocab: {len(tgt_vocab):,}")

train_ds=TranslationDataset(train_pairs,src_vocab,tgt_vocab,MAX_LEN)
val_ds  =TranslationDataset(val_pairs,  src_vocab,tgt_vocab,MAX_LEN)
test_ds =TranslationDataset(test_pairs, src_vocab,tgt_vocab,MAX_LEN)
train_loader=DataLoader(train_ds,BATCH_SIZE,shuffle=True, collate_fn=collate_fn)
val_loader  =DataLoader(val_ds,  BATCH_SIZE,shuffle=False,collate_fn=collate_fn)
test_loader =DataLoader(test_ds, BATCH_SIZE,shuffle=False,collate_fn=collate_fn)
print(f"Batches => Train:{len(train_loader)} | Val:{len(val_loader)} | Test:{len(test_lo
```

```
class Encoder(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim, num_layers, dropout):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=PAD_IDX)
        self.lstm = nn.LSTM(embed_dim, hidden_dim, num_layers, batch_first=True,
                            dropout=dropout if num_layers>1 else 0.0)
        self.dropout = nn.Dropout(dropout)

    def forward(self, src, src_lens):
        emb    = self.dropout(self.embedding(src))
        packed = pack_padded_sequence(emb, src_lens.cpu(),
                                      batch_first=True, enforce_sorted=True)
        out_p, (h, c) = self.lstm(packed)
        outputs, _    = pad_packed_sequence(out_p, batch_first=True)
        return outputs, h, c


class BahdanauAttention(nn.Module):
    """
    Reference: Bahdanau et al. (2015) arXiv:1409.0473

    score(h_s, s_t) = V * tanh( W_enc*h_s + W_dec*s_t )

    Three learnable weight matrices. Additive composition allows
    the model to compare encoder and decoder states flexibly.
    Context is injected BEFORE the decoder LSTM step.
    """
    def __init__(self, hidden_dim):
        super().__init__()
        self.W_enc = nn.Linear(hidden_dim, hidden_dim, bias=False)
        self.W_dec = nn.Linear(hidden_dim, hidden_dim, bias=False)
        self.V     = nn.Linear(hidden_dim, 1,          bias=False)

    def forward(self, dec_hidden, enc_outputs, src_mask=None):

        dec_exp  = dec_hidden.unsqueeze(1)
        energy   = torch.tanh(
            self.W_enc(enc_outputs) + self.W_dec(dec_exp)
        )
        scores   = self.V(energy).squeeze(-1)
        if src_mask is not None:
            scores = scores.masked_fill(~src_mask, float('-inf'))
        alpha   = F.softmax(scores, dim=-1)
        context = torch.bmm(alpha.unsqueeze(1), enc_outputs).squeeze(1)
        return context, alpha

class LuongAttention(nn.Module):
    """
    Reference: Luong et al. (2015) arXiv:1508.04025
    'General' variant: score(h_s, s_t) = h_s * W * s_t

    Single weight matrix W. More parameter-efficient than Bahdanau.
    All three Luong variants: dot | general | concat
    We use 'general' as it balances simplicity and expressiveness.
    """
    def __init__(self, hidden_dim):
```

```
            super().__init__()
            self.W = nn.Linear(hidden_dim, hidden_dim, bias=False)

        def forward(self, dec_hidden, enc_outputs, src_mask=None):

            Wh     = self.W(dec_hidden).unsqueeze(2)
            scores = torch.bmm(enc_outputs, Wh).squeeze(2)
            if src_mask is not None:
                scores = scores.masked_fill(~src_mask, float('-inf'))
            alpha   = F.softmax(scores, dim=-1)
            context = torch.bmm(alpha.unsqueeze(1), enc_outputs).squeeze(1)
            return context, alpha


    class AttentionDecoder(nn.Module):
        """
        Works with either BahdanauAttention or LuongAttention.

        Each decode step:
          1. attention(dec_hidden_top, enc_outputs) -> context, alpha
          2. lstm_input = concat(embed(y_{t-1}), context)
          3. logits = Linear( concat(lstm_out, context) )
        """
        def __init__(self, vocab_size, embed_dim, hidden_dim,
                     attention, num_layers, dropout):
            super().__init__()
            self.attention = attention
            self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=PAD_IDX)

            self.lstm = nn.LSTM(embed_dim + hidden_dim, hidden_dim, num_layers,
                                batch_first=True,
                                dropout=dropout if num_layers>1 else 0.0)

            self.fc_out  = nn.Linear(hidden_dim * 2, vocab_size)
            self.dropout = nn.Dropout(dropout)

        def forward(self, token, hidden, cell, enc_outputs, src_mask=None):
            emb      = self.dropout(self.embedding(token.unsqueeze(1)))
            top_h    = hidden[-1]
            ctx, attn = self.attention(top_h, enc_outputs, src_mask)
            lstm_in  = torch.cat([emb, ctx.unsqueeze(1)], dim=2)
            out, (hidden, cell) = self.lstm(lstm_in, (hidden, cell))
            logits   = self.fc_out(
                torch.cat([out.squeeze(1), ctx], dim=1)
            )
            return logits, hidden, cell, attn


    class Seq2SeqAttention(nn.Module):
        def __init__(self, encoder, decoder):
            super().__init__()
            self.encoder = encoder
            self.decoder = decoder

        def forward(self, src, src_lens, tgt, tf_ratio=0.5):
            B, T      = tgt.size()
            V         = self.decoder.fc_out.out_features
            outputs   = torch.zeros(B, T, V,           device=src.device)
            attn_maps = torch.zeros(B, T, src.size(1), device=src.device)

            enc_out, h, c = self.encoder(src, src_lens)
            src_mask = (src != PAD_IDX)
            token    = tgt[:, 0]

            for t in range(1, T):
                logits, h, c, attn = self.decoder(token, h, c, enc_out, src_mask)
                outputs[:, t] = logits
                attn_maps[:, t, :attn.size(1)] = attn
```

```
            attn_maps[:, t, :attn.size(1)] = attn
            token = tgt[:, t] if random.random() < tf_ratio else logits.argmax(-1)

        return outputs, attn_maps

print("All model classes defined")
```

Step 6 — Training Utilities

```
criterion = nn.CrossEntropyLoss(ignore_index=PAD_IDX)

def train_epoch(model, loader, optimizer, tf_ratio):
    model.train(); total = 0.0
    for src, tgt, sl, _ in loader:
        src, tgt, sl = src.to(DEVICE), tgt.to(DEVICE), sl.to(DEVICE)
        optimizer.zero_grad()
        out, _ = model(src, sl, tgt, tf_ratio)
        loss = criterion(out[:,1:].reshape(-1, out.size(-1)),
                         tgt[:,1:].reshape(-1))
        loss.backward()
        nn.utils.clip_grad_norm_(model.parameters(), CLIP)
        optimizer.step()
        total += loss.item()
    return total / len(loader)

@torch.no_grad()
def evaluate(model, loader):
    model.eval(); total = 0.0
    for src, tgt, sl, _ in loader:
        src, tgt, sl = src.to(DEVICE), tgt.to(DEVICE), sl.to(DEVICE)
        out, _ = model(src, sl, tgt, 0.0)
        loss = criterion(out[:,1:].reshape(-1, out.size(-1)),
                         tgt[:,1:].reshape(-1))
        total += loss.item()
    return total / len(loader)

def run_training(model, model_name):
    opt = torch.optim.Adam(model.parameters(), lr=LR)
    sch = torch.optim.lr_scheduler.ReduceLROnPlateau(opt, patience=3, factor=0.5)
    best_val, best_state = float('inf'), None
    tl_h, vl_h = [], []
    n = sum(p.numel() for p in model.parameters() if p.requires_grad)
    print(f"\n{'='*60}")
    print(f"  Training: {model_name}  ({n:,} params)")
    print(f"{'='*60}")
    print(f"{'Epoch':>6} | {'Train':>9} | {'Val':>9} | {'PPL':>7} | {'TF':>5} | Time")
    print("-" * 52)
    for epoch in range(1, EPOCHS + 1):
        tf  = max(0.5, 1.0 - (epoch-1)*(0.5/EPOCHS))
        t0  = time.time()
        tl  = train_epoch(model, train_loader, opt, tf)
        vl  = evaluate(model, val_loader)
        sch.step(vl); tl_h.append(tl); vl_h.append(vl)
        mark = " <-" if vl < best_val else ""
        if vl < best_val:
            best_val   = vl
            best_state = {k: v.cpu().clone() for k,v in model.state_dict().items()}
        print(f"{epoch:>6} | {tl:>9.4f} | {vl:>9.4f} | {math.exp(vl):>7.2f} | "
              f"{tf:>5.2f} | {time.time()-t0:.1f}s{mark}")
    model.load_state_dict({k: v.to(DEVICE) for k,v in best_state.items()})
    print(f"  Best model restored (val_loss={best_val:.4f})")
    return tl_h, vl_h

print("Training utilities ready")
```

Step 7 — Train Bahdanau Attention Model

```
enc_b  = Encoder(len(src_vocab), EMBED_DIM, HIDDEN_DIM, NUM_LAYERS, DROPOUT)
attn_b = BahdanauAttention(HIDDEN_DIM)
dec_b  = AttentionDecoder(len(tgt_vocab), EMBED_DIM, HIDDEN_DIM,
                          attn_b, NUM_LAYERS, DROPOUT)
model_b = Seq2SeqAttention(enc_b, dec_b).to(DEVICE)

tl_b, vl_b = run_training(model_b, "Bahdanau Attention")
```

Step 8 — Train Luong Attention Model

```
enc_l  = Encoder(len(src_vocab), EMBED_DIM, HIDDEN_DIM, NUM_LAYERS, DROPOUT)
attn_l = LuongAttention(HIDDEN_DIM)
dec_l  = AttentionDecoder(len(tgt_vocab), EMBED_DIM, HIDDEN_DIM,
                          attn_l, NUM_LAYERS, DROPOUT)
model_l = Seq2SeqAttention(enc_l, dec_l).to(DEVICE)

tl_l, vl_l = run_training(model_l, "Luong Attention")
```

Step 9 — BLEU Evaluation

```
@torch.no_grad()
def translate(model, sentence, max_len=50):
    model.eval()
    toks = [SOS_IDX] + src_vocab.encode(normalize_string(sentence)) + [EOS_IDX]
    src_t = torch.tensor(toks).unsqueeze(0).to(DEVICE)
    sl    = torch.tensor([len(toks)])
    enc_out, h, c = model.encoder(src_t, sl)
    src_mask = (src_t != PAD_IDX)
    token = torch.tensor([SOS_IDX], device=DEVICE)
    result = []; attn_list = []
    for _ in range(max_len):
        logits, h, c, attn = model.decoder(token, h, c, enc_out, src_mask)
        nt = logits.argmax(-1)
        if nt.item() == EOS_IDX: break
        result.append(nt.item())
        attn_list.append(attn.squeeze(0).cpu())
        token = nt
    translation  = tgt_vocab.decode(result)
    attn_matrix  = torch.stack(attn_list, 0) if attn_list else None
    return translation, attn_matrix

def _ngrams(toks, n):
    return Counter(tuple(toks[i:i+n]) for i in range(len(toks)-n+1))

def bleu_simple(hyp, ref):
    h, r = hyp.split(), ref.split()
    if not h: return 0.0
    cc=tc=0
    for n in range(1,5):
        hn,rn=_ngrams(h,n),_ngrams(r,n)
        cc+=sum(min(c,rn.get(g,0)) for g,c in hn.items()); tc+=max(0,len(h)-n+1)
    if tc==0: return 0.0
    return min(1.0,math.exp(1-len(r)/len(h)))*cc/tc

def compute_bleu(model, pairs, max_n=500):
```

```
def compute_bleu(model, pairs, max_p=500):
    hyps,refs=[],[]
    for s,r in pairs[:max_p]:
        h,_ = translate(model, s); hyps.append(h); refs.append(r)
    try:
        import sacrebleu as sb
        return sb.corpus_bleu(hyps,[refs]).score
    except ImportError:
        return 100*sum(bleu_simple(h,r) for h,r in zip(hyps,refs))/len(hyps)

bleu_b = compute_bleu(model_b, test_pairs)
bleu_l = compute_bleu(model_l, test_pairs)
tl_b_  = evaluate(model_b, test_loader)
tl_l_  = evaluate(model_l, test_loader)

print(f"\n{'='*55}")
print(f"  {'Model':<12} | {'Test Loss':>9} | {'PPL':>7} | {'BLEU':>7}")
print(f"  {'-'*48}")
print(f"  {'Bahdanau':<12} | {tl_b_:>9.4f} | {math.exp(tl_b_):>7.2f} | {bleu_b:>7.2f}")
print(f"  {'Luong':<12} | {tl_l_:>9.4f} | {math.exp(tl_l_):>7.2f} | {bleu_l:>7.2f}")
print(f"{'='*55}")
```

Step 10 — Training Curves Comparison

```
fig, axes = plt.subplots(1, 3, figsize=(18, 5))
eps = range(1, EPOCHS+1)

axes[0].plot(eps, tl_b, 'b-o',  ms=3, lw=2, label='Bahdanau train')
axes[0].plot(eps, vl_b, 'b--s', ms=3, lw=2, alpha=.7, label='Bahdanau val')
axes[0].plot(eps, tl_l, 'g-o',  ms=3, lw=2, label='Luong train')
axes[0].plot(eps, vl_l, 'g--s', ms=3, lw=2, alpha=.7, label='Luong val')
axes[0].set_title('Loss per Epoch'); axes[0].set_xlabel('Epoch')
axes[0].set_ylabel('Loss'); axes[0].legend(fontsize=8); axes[0].grid(True,alpha=.3)

axes[1].plot(eps, [math.exp(v) for v in vl_b], 'b-o', ms=3, lw=2, label='Bahdanau')
axes[1].plot(eps, [math.exp(v) for v in vl_l], 'g-o', ms=3, lw=2, label='Luong')
axes[1].set_title('Validation Perplexity'); axes[1].set_xlabel('Epoch')
axes[1].set_ylabel('PPL'); axes[1].legend(); axes[1].grid(True,alpha=.3)

bars = axes[2].bar(['Bahdanau','Luong'], [bleu_b, bleu_l],
                   color=['#2196F3','#4CAF50'], edgecolor='black', width=.4, alpha=.88)
for bar, score in zip(bars, [bleu_b, bleu_l]):
    axes[2].text(bar.get_x()+bar.get_width()/2, bar.get_height()+.2,
                 f'{score:.2f}', ha='center', fontweight='bold', fontsize=12)
axes[2].set_title('BLEU Score (test)'); axes[2].set_ylabel('BLEU')
axes[2].set_ylim(0, max(bleu_b,bleu_l)*1.3+2); axes[2].grid(True,alpha=.3,axis='y')

plt.suptitle('Attention Seq2Seq Comparison', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.savefig('attention_comparison.png', dpi=150, bbox_inches='tight')
plt.show()
print("Saved: attention_comparison.png")
```

Step 11 — Sample Translations

```
print("=" * 70)
print(f"  {'SRC':<24}  {'Bahdanau':<22}  {'Luong':<22}")
print("=" * 70)
for src, ref in test_pairs[:10]:
    hb, _ = translate(model_b, src)
    hl, _ = translate(model_l, src)
```

```
    print(f"  SRC      : {src}")
    print(f"  REF      : {ref}")
    print(f"  Bahdanau : {hb}")
    print(f"  Luong    : {hl}")
    print()
```

```
def plot_heatmap(attn_np, src_toks, tgt_toks, title, fname):
    """
    Plot attention weight matrix as a colour-coded heatmap.
    attn_np shape: (len(tgt_toks), len(src_toks))
    """
    fw = max(8, len(src_toks)*0.65 + 2)
    fh = max(5, len(tgt_toks)*0.60 + 2)
    fig, ax = plt.subplots(figsize=(fw, fh))
    im = ax.imshow(attn_np, cmap='YlOrRd', aspect='auto', vmin=0, vmax=attn_np.max())
    plt.colorbar(im, ax=ax, fraction=0.046, pad=0.04)
    ax.set_xticks(range(len(src_toks))); ax.set_yticks(range(len(tgt_toks)))
    ax.set_xticklabels(src_toks, rotation=40, ha='right', fontsize=10)
    ax.set_yticklabels(tgt_toks, fontsize=10)
    for i in range(len(tgt_toks)):
        for j in range(len(src_toks)):
            v = attn_np[i, j]
            c = 'white' if v > 0.5*attn_np.max() else 'black'
            ax.text(j, i, f'{v:.2f}', ha='center', va='center', fontsize=7, color=c)
    ax.set_xlabel('Source (English)', fontsize=11)
    ax.set_ylabel('Target (Spanish)', fontsize=11)
    ax.set_title(title, fontsize=11, fontweight='bold', pad=10)
    plt.tight_layout()
    plt.savefig(fname, dpi=150, bbox_inches='tight')
    plt.show(); plt.close()
    print(f"  Saved: {fname}")

examples = []   # store for side-by-side plot
count = 0
for src, ref in test_pairs:
    if count >= 3: break
    if len(src.split()) > 10: continue
    hb, ab = translate(model_b, src)
    hl, al = translate(model_l, src)
    if ab is None or al is None: continue
    if not hb.strip() or not hl.strip(): continue

    src_toks = ['<sos>'] + src.split() + ['<eos>']
    tb_toks  = hb.split()
    tl_toks  = hl.split()
    amb = ab.numpy()[:len(tb_toks), :len(src_toks)]
    aml = al.numpy()[:len(tl_toks), :len(src_toks)]

    # Individual heatmaps
    plot_heatmap(amb, src_toks, tb_toks,
                 f'Bahdanau Attention — Example {count+1}\nSRC: "{src}"  HYP: "{hb}"',
                 f'attn_bahdanau_ex{count+1}.png')
    plot_heatmap(aml, src_toks, tl_toks,
                 f'Luong Attention — Example {count+1}\nSRC: "{src}"  HYP: "{hl}"',
                 f'attn_luong_ex{count+1}.png')

    examples.append((src, src_toks, hb, tb_toks, amb, hl, tl_toks, aml))
    count += 1
```

Step 13 — Side-by-Side: Bahdanau vs Luong

```
for i, (src, src_toks, hb, tb, amb, hl, tl, aml) in enumerate(examples):
```

```
    fig, axes = plt.subplots(1, 2, figsize=(18, max(6, len(src_toks)*0.7)))
    fig.suptitle(f'Attention Comparison — Example {i+1}\n'
                 f'Source: "{src}"', fontsize=13, fontweight='bold')

    for ax, mat, tgt_toks, name, cmap in zip(
        axes,
        [amb, aml], [tb, tl],
        ['Bahdanau (Additive)\nscore = V*tanh(W_enc*h_s + W_dec*s_t)',
         'Luong (Multiplicative)\nscore = h_s * W * s_t'],
        ['YlOrRd', 'YlGnBu']
    ):
        im = ax.imshow(mat, cmap=cmap, aspect='auto', vmin=0)
        plt.colorbar(im, ax=ax, fraction=0.046, pad=0.04)
        ax.set_xticks(range(len(src_toks))); ax.set_yticks(range(len(tgt_toks)))
        ax.set_xticklabels(src_toks, rotation=40, ha='right', fontsize=9)
        ax.set_yticklabels(tgt_toks, fontsize=9)
        ax.set_title(name, fontsize=10, fontweight='bold')
        ax.set_xlabel('Source (English)', fontsize=10)
        ax.set_ylabel('Target (Spanish)', fontsize=10)
        for ii in range(len(tgt_toks)):
            for jj in range(len(src_toks)):
                v = mat[ii, jj]
                col = 'white' if v > 0.5*mat.max() else 'black'
                ax.text(jj, ii, f'{v:.2f}', ha='center', va='center',
                        fontsize=6.5, color=col)
    plt.tight_layout()
    plt.savefig(f'attn_compare_ex{i+1}.png', dpi=150, bbox_inches='tight')
    plt.show(); plt.close()
    print(f"  Saved: attn_compare_ex{i+1}.png")
```

Step 14 — Translate Your Own Sentences

```
my_sentences = [
    "How are you?",
    "I love you.",
    "What is your name?",
    "Good morning.",
    "Where is the library?",
    "I don't understand.",
]
print(f"  {'English':<28} {'Bahdanau':<25} {'Luong'}")
print("-" * 80)
for s in my_sentences:
    hb, _ = translate(model_b, s)
    hl, _ = translate(model_l, s)
    print(f"  {s:<28} {hb:<25} {hl}")
```

Step 15 — Save & Download

```
import zipfile

def save_model(model, name, bleu, tl_hist, vl_hist):
    torch.save({
        'model_state': model.state_dict(),
        'src_vocab': src_vocab.word2idx,
        'tgt_vocab': tgt_vocab.word2idx,
        'bleu': bleu,
        'train_losses': tl_hist,
        'val_losses': vl_hist,
        'hparams': {'EMBED_DIM':EMBED_DIM,'HIDDEN_DIM':HIDDEN_DIM,
                    'NUM_LAYERS':NUM_LAYERS,'DROPOUT':DROPOUT}
```

```
                    NUM_LAYERS':NUM_LAYERS, 'DROPOUT':DROPOUT},
    }, f'{name}.pt')
    print(f"  Saved: {name}.pt")

save_model(model_b, 'bahdanau_model', bleu_b, tl_b, vl_b)
save_model(model_l, 'luong_model',    bleu_l, tl_l, vl_l)

output_files = (
    ['bahdanau_model.pt', 'luong_model.pt', 'attention_comparison.png'] +
    [f'attn_bahdanau_ex{i+1}.png'  for i in range(len(examples))] +
    [f'attn_luong_ex{i+1}.png'     for i in range(len(examples))] +
    [f'attn_compare_ex{i+1}.png'   for i in range(len(examples))]
)

with zipfile.ZipFile('attention_outputs.zip', 'w') as zf:
    for fn in output_files:
        if os.path.exists(fn):
            zf.write(fn)
            print(f"  Added to zip: {fn}")

from google.colab import files
files.download('attention_outputs.zip')
print("Downloaded: attention_outputs.zip")
```