🚀 NLP Advanced: Dịch máy Anh-Pháp với Attention & Beam Search

Phần mở rộng (Bonus Points)

- **Dataset**: Europarl (WMT 2014) - ~2 triệu cặp câu
- **Cải tiến 1**: Attention Mechanism (Bahdanau)
- **Cải tiến 2**: Beam Search Decoding
- **Cải tiến 3**: Bidirectional Encoder với Pack Padded Sequence
- **So sánh**: Hiệu suất Greedy vs Beam Search

1. Cài đặt & Import thư viện

In [1]:
```
# --- CÀI ĐẶT (Chỉ chạy 1 lần nếu cần) ---
# !pip install torch torchvision torchaudio --index-url https://download.pytorch
# !pip install spacy torchtext nltk
# !python -m spacy download en_core_web_sm
# !python -m spacy download fr_core_news_sm

import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence, pack_padded_sequence, pad_packed_se

from torchtext.data.utils import get_tokenizer
from collections import Counter
import spacy
import numpy as np
import random
import os
import time
import math

# Seed để tái lập kết quả
SEED = 1234
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed_all(SEED)
torch.backends.cudnn.deterministic = True

# Kiểm tra thiết bị
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"🖥️ Thiết bị: {device}")
if torch.cuda.is_available():
    print(f"   GPU: {torch.cuda.get_device_name(0)}")
    print(f"   VRAM: {torch.cuda.get_device_properties(0).total_memory / 1e9:.1f

print(f"📦 PyTorch version: {torch.__version__}")
```

```
🖥️ Thiết bị: cuda
   GPU: NVIDIA GeForce RTX 3050 Laptop GPU
   VRAM: 4.3 GB
📦 PyTorch version: 2.7.1+cu118
```

2. Cấu hình & Hyperparameters

In [2]:
```python
# ============================================================
# CẤU HÌNH CHÍNH
# ============================================================

# Đường dẫn dữ liệu
DATA_DIR = './data/advanced'
SRC_LANGUAGE = 'en'
TGT_LANGUAGE = 'fr'

# Giới hạn dữ liệu (để training nhanh hơn, có thể tăng lên)
MAX_SAMPLES = 200000  # Dùng 200k câu (có thể tăng lên 500k-2M nếu có GPU mạnh)
MAX_LENGTH = 50       # Giới hạn độ dài câu

# Hyperparameters
BATCH_SIZE = 64       # Giảm nếu hết VRAM
EMB_DIM = 256         # Embedding dimension
HID_DIM = 512         # Hidden dimension
N_LAYERS = 2          # Số layer LSTM
DROPOUT = 0.5
LEARNING_RATE = 0.001
N_EPOCHS = 15
CLIP = 1
PATIENCE = 3          # Early stopping

# Beam Search
BEAM_SIZE = 5

# Token đặc biệt
UNK_IDX, PAD_IDX, SOS_IDX, EOS_IDX = 0, 1, 2, 3
SPECIAL_SYMBOLS = ['<unk>', '<pad>', '<sos>', '<eos>']

print("✅ Cấu hình hoàn tất!")
print(f"   • Max samples: {MAX_SAMPLES:,}")
print(f"   • Max length: {MAX_LENGTH}")
print(f"   • Batch size: {BATCH_SIZE}")
print(f"   • Hidden dim: {HID_DIM}")
```

```
✅ Cấu hình hoàn tất!
   • Max samples: 200,000
   • Max length: 50
   • Batch size: 64
   • Hidden dim: 512
```

3. Xử lý dữ liệu Europarl

In [3]:
```python
# ============================================================
# XỬ LÝ DỮ LIỆU EUROPARL
# ============================================================

# 1. Tokenizers
print("📝 Khởi tạo Tokenizers...")
```

```python
tokenizers = {
    SRC_LANGUAGE: get_tokenizer('spacy', language='en_core_web_sm'),
    TGT_LANGUAGE: get_tokenizer('spacy', language='fr_core_news_sm')
}

# 2. Đọc và tiền xử lý dữ liệu
def load_data(data_dir, max_samples=None, max_length=50):
    """Đọc dữ liệu song song EN-FR"""

    src_file = os.path.join(data_dir, 'europarl-v7.fr-en.en')
    tgt_file = os.path.join(data_dir, 'europarl-v7.fr-en.fr')

    data = []

    print(f"📁 Đang đọc dữ liệu từ {data_dir}...")

    with open(src_file, 'r', encoding='utf-8') as f_src, \
         open(tgt_file, 'r', encoding='utf-8') as f_tgt:

        for i, (src_line, tgt_line) in enumerate(zip(f_src, f_tgt)):
            if max_samples and i >= max_samples:
                break

            src_line = src_line.strip()
            tgt_line = tgt_line.strip()

            # Bỏ qua câu rỗng hoặc quá dài
            if not src_line or not tgt_line:
                continue

            # Tokenize để kiểm tra độ dài
            src_tokens = tokenizers[SRC_LANGUAGE](src_line.lower())
            tgt_tokens = tokenizers[TGT_LANGUAGE](tgt_line.lower())

            # Lọc câu quá dài
            if len(src_tokens) > max_length or len(tgt_tokens) > max_length:
                continue

            # Lọc câu quá ngắn (ít nhất 3 từ)
            if len(src_tokens) < 3 or len(tgt_tokens) < 3:
                continue

            data.append((src_line, tgt_line))

            if (i + 1) % 100000 == 0:
                print(f"   Đã xử lý {i+1:,} dòng, thu được {len(data):,} cặp câu

    print(f"✅ Hoàn tất! Tổng số cặp câu: {len(data):,}")
    return data

# Đọc dữ liệu
all_data = load_data(DATA_DIR, max_samples=MAX_SAMPLES, max_length=MAX_LENGTH)

# Chia train/val/test (90/5/5)
random.shuffle(all_data)
train_size = int(0.9 * len(all_data))
val_size = int(0.05 * len(all_data))

train_data = all_data[:train_size]
val_data = all_data[train_size:train_size + val_size]
```

```
test_data = all_data[train_size + val_size:]

print(f"\n📊 Phân chia dữ liệu:")
print(f"   • Train: {len(train_data):,} cặp")
print(f"   • Val:   {len(val_data):,} cặp")
print(f"   • Test:  {len(test_data):,} cặp")
```

```
📝 Khởi tạo Tokenizers...
📁 Đang đọc dữ liệu từ ./data/advanced...
   Đã xử lý 100,000 dòng, thu được 84,917 cặp câu hợp lệ...
   Đã xử lý 200,000 dòng, thu được 170,056 cặp câu hợp lệ...
✅ Hoàn tất! Tổng số cặp câu: 170,056

📊 Phân chia dữ liệu:
   • Train: 153,050 cặp
   • Val:   8,502 cặp
   • Test:  8,504 cặp
```

In [4]:
```
# ============================================================
# XÂY DỰNG VOCABULARY
# ============================================================

class Vocabulary:
    """Class quản lý từ điển cho mỗi ngôn ngữ"""

    def __init__(self, min_freq=2, max_size=15000):
        self.min_freq = min_freq
        self.max_size = max_size
        self.stoi = {}  # string to index
        self.itos = {}  # index to string

    def build(self, data, language, tokenizer):
        """Xây dựng từ điển từ dữ liệu"""
        counter = Counter()

        idx = 0 if language == SRC_LANGUAGE else 1

        for pair in data:
            tokens = tokenizer(pair[idx].lower())
            counter.update(tokens)

        # Thêm special tokens
        for i, symbol in enumerate(SPECIAL_SYMBOLS):
            self.stoi[symbol] = i
            self.itos[i] = symbol

        # Thêm các từ phổ biến
        current_idx = len(SPECIAL_SYMBOLS)
        for token, freq in counter.most_common():
            if freq < self.min_freq:
                break
            if current_idx >= self.max_size:
                break
            if token not in self.stoi:
                self.stoi[token] = current_idx
                self.itos[current_idx] = token
                current_idx += 1

        print(f"   Vocabulary size: {len(self.stoi):,}")
        return self
```

```python
    def __len__(self):
        return len(self.stoi)

    def __getitem__(self, token):
        return self.stoi.get(token, UNK_IDX)

    def lookup_tokens(self, indices):
        return [self.itos.get(i, '<unk>') for i in indices]

    def encode(self, tokens):
        return [self[token] for token in tokens]

# Xây dựng từ điển
print("📚 Xây dựng Vocabulary...")
vocab_src = Vocabulary(min_freq=2, max_size=15000)
vocab_tgt = Vocabulary(min_freq=2, max_size=15000)

print(f"   Building English vocab...")
vocab_src.build(train_data, SRC_LANGUAGE, tokenizers[SRC_LANGUAGE])

print(f"   Building French vocab...")
vocab_tgt.build(train_data, TGT_LANGUAGE, tokenizers[TGT_LANGUAGE])

print("✅ Vocabulary hoàn tất!")
```

```
📚 Xây dựng Vocabulary...
   Building English vocab...
   Vocabulary size: 15,000
   Building French vocab...
   Vocabulary size: 15,000
✅ Vocabulary hoàn tất!
```

In [5]:
```python
# ============================================================
# DATASET & DATALOADER
# ============================================================

class TranslationDataset(Dataset):
    """Custom Dataset cho dịch máy"""

    def __init__(self, data, vocab_src, vocab_tgt, tokenizers):
        self.data = data
        self.vocab_src = vocab_src
        self.vocab_tgt = vocab_tgt
        self.tokenizers = tokenizers

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        src_text, tgt_text = self.data[idx]

        # Tokenize
        src_tokens = self.tokenizers[SRC_LANGUAGE](src_text.lower())
        tgt_tokens = self.tokenizers[TGT_LANGUAGE](tgt_text.lower())

        # Encode (thêm SOS và EOS)
        src_indices = [SOS_IDX] + self.vocab_src.encode(src_tokens) + [EOS_IDX]
        tgt_indices = [SOS_IDX] + self.vocab_tgt.encode(tgt_tokens) + [EOS_IDX]
        return torch.tensor(src_indices), torch.tensor(tgt_indices)

def collate_fn(batch):
    """Custom collate function với padding và sorting"""
    src_batch, tgt_batch = zip(*batch)

    # Lấy độ dài
    src_lengths = torch.tensor([len(s) for s in src_batch])

    # Sort theo độ dài giảm dần (cần cho pack_padded_sequence)
    sorted_indices = src_lengths.argsort(descending=True)

    src_batch = [src_batch[i] for i in sorted_indices]
    tgt_batch = [tgt_batch[i] for i in sorted_indices]
    src_lengths = src_lengths[sorted_indices]

    # Padding
    src_padded = pad_sequence(src_batch, padding_value=PAD_IDX)
    tgt_padded = pad_sequence(tgt_batch, padding_value=PAD_IDX)

    return src_padded, tgt_padded, src_lengths

# Tạo datasets
train_dataset = TranslationDataset(train_data, vocab_src, vocab_tgt, tokenizers)
val_dataset = TranslationDataset(val_data, vocab_src, vocab_tgt, tokenizers)
test_dataset = TranslationDataset(test_data, vocab_src, vocab_tgt, tokenizers)

# Tạo dataloaders
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True,
                          collate_fn=collate_fn, pin_memory=True, num_workers=0)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False,
                        collate_fn=collate_fn, pin_memory=True, num_workers=0)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False,
                         collate_fn=collate_fn, pin_memory=True, num_workers=0)

# Kiểm tra
src, tgt, lengths = next(iter(train_loader))
print(f"✅ DataLoader hoàn tất!")
print(f"   • Source shape: {src.shape}")
print(f"   • Target shape: {tgt.shape}")
print(f"   • Lengths sample: {lengths[:5]}")
```

```
✅ DataLoader hoàn tất!
   • Source shape: torch.Size([51, 64])
   • Target shape: torch.Size([52, 64])
   • Lengths sample: tensor([51, 45, 45, 45, 44])
```

3.1 Trực quan hóa Dữ liệu (Data Visualization)

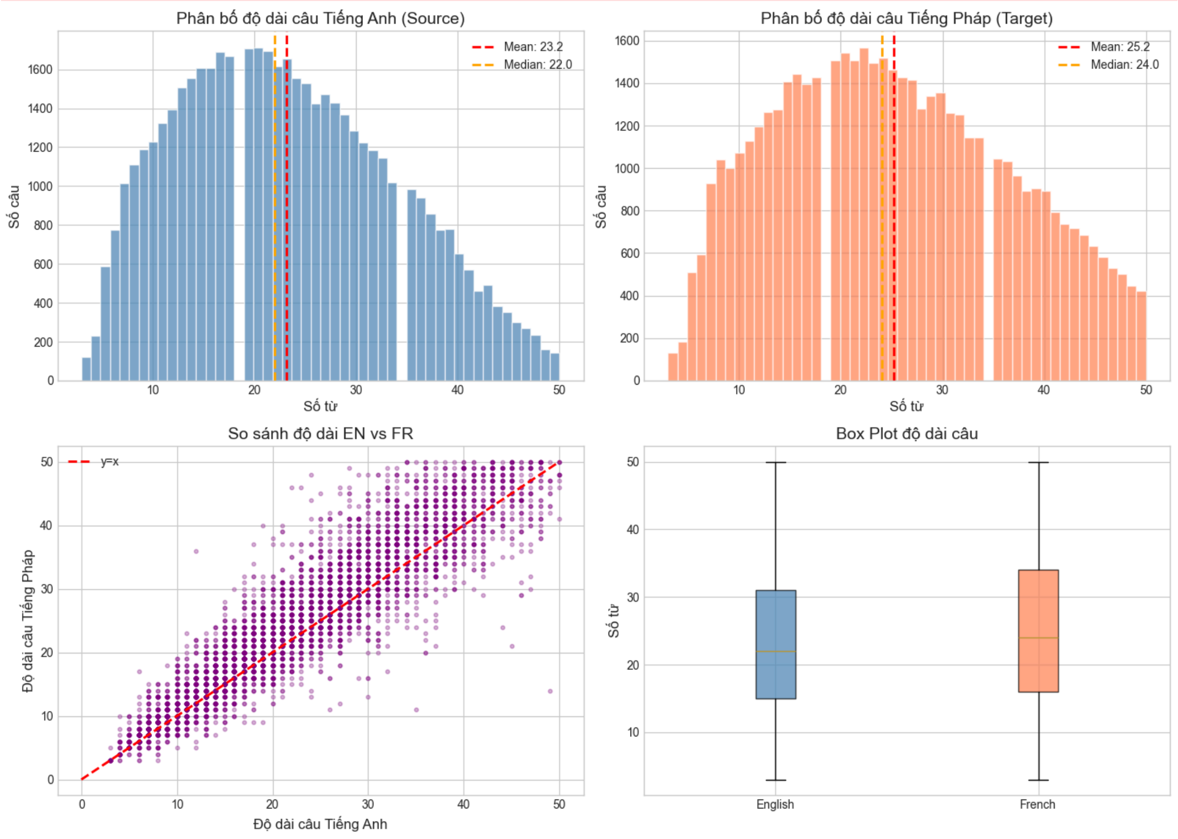
In [6]:

```python
# ============================================================
# TRỰC QUAN HÓA DỮ LIỆU
# ============================================================

import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import numpy as np

# Thiết lập style
```

```python
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette("husl")

# 1. PHÂN BỐ ĐỘ DÀI CÂU
# ============================================================
print("📊 Đang phân tích độ dài câu...")

src_lengths = []
tgt_lengths = []

for src, tgt in train_data[:50000]:  # Lấy mẫu 50k để phân tích nhanh
    src_tokens = tokenizers[SRC_LANGUAGE](src.lower())
    tgt_tokens = tokenizers[TGT_LANGUAGE](tgt.lower())
    src_lengths.append(len(src_tokens))
    tgt_lengths.append(len(tgt_tokens))

fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# Histogram độ dài câu nguồn (English)
ax1 = axes[0, 0]
ax1.hist(src_lengths, bins=50, color='steelblue', edgecolor='white', alpha=0.7)
ax1.axvline(np.mean(src_lengths), color='red', linestyle='--', linewidth=2, labe
ax1.axvline(np.median(src_lengths), color='orange', linestyle='--', linewidth=2,
ax1.set_xlabel('Số từ', fontsize=12)
ax1.set_ylabel('Số câu', fontsize=12)
ax1.set_title('Phân bố độ dài câu Tiếng Anh (Source)', fontsize=14)
ax1.legend()

# Histogram độ dài câu đích (French)
ax2 = axes[0, 1]
ax2.hist(tgt_lengths, bins=50, color='coral', edgecolor='white', alpha=0.7)
ax2.axvline(np.mean(tgt_lengths), color='red', linestyle='--', linewidth=2, labe
ax2.axvline(np.median(tgt_lengths), color='orange', linestyle='--', linewidth=2,
ax2.set_xlabel('Số từ', fontsize=12)
ax2.set_ylabel('Số câu', fontsize=12)
ax2.set_title('Phân bố độ dài câu Tiếng Pháp (Target)', fontsize=14)
ax2.legend()

# Scatter plot so sánh độ dài EN vs FR
ax3 = axes[1, 0]
ax3.scatter(src_lengths[:5000], tgt_lengths[:5000], alpha=0.3, s=10, c='purple')
ax3.plot([0, max(src_lengths)], [0, max(src_lengths)], 'r--', linewidth=2, label
ax3.set_xlabel('Độ dài câu Tiếng Anh', fontsize=12)
ax3.set_ylabel('Độ dài câu Tiếng Pháp', fontsize=12)
ax3.set_title('So sánh độ dài EN vs FR', fontsize=14)
ax3.legend()

# Box plot so sánh
ax4 = axes[1, 1]
box_data = [src_lengths, tgt_lengths]
bp = ax4.boxplot(box_data, labels=['English', 'French'], patch_artist=True)
colors = ['steelblue', 'coral']
for patch, color in zip(bp['boxes'], colors):
    patch.set_facecolor(color)
    patch.set_alpha(0.7)
ax4.set_ylabel('Số từ', fontsize=12)
ax4.set_title('Box Plot độ dài câu', fontsize=14)

plt.tight_layout()
plt.savefig('data_length_distribution.png', dpi=150, bbox_inches='tight')
plt.show()

print(f"\n📈 Thống kê độ dài câu:")
print(f"   English - Mean: {np.mean(src_lengths):.1f}, Median: {np.median(src_le
print(f"   French  - Mean: {np.mean(tgt_lengths):.1f}, Median: {np.median(tgt_le
print("✅ Đã lưu: data_length_distribution.png")
```

```
📊 Đang phân tích độ dài câu...
```

```
C:\Users\Scarlet\AppData\Local\Temp\ipykernel_11592\1287154487.py:61: MatplotlibD
eprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_lab
els' since Matplotlib 3.9; support for the old name will be dropped in 3.11.
  bp = ax4.boxplot(box_data, labels=['English', 'French'], patch_artist=True)
```



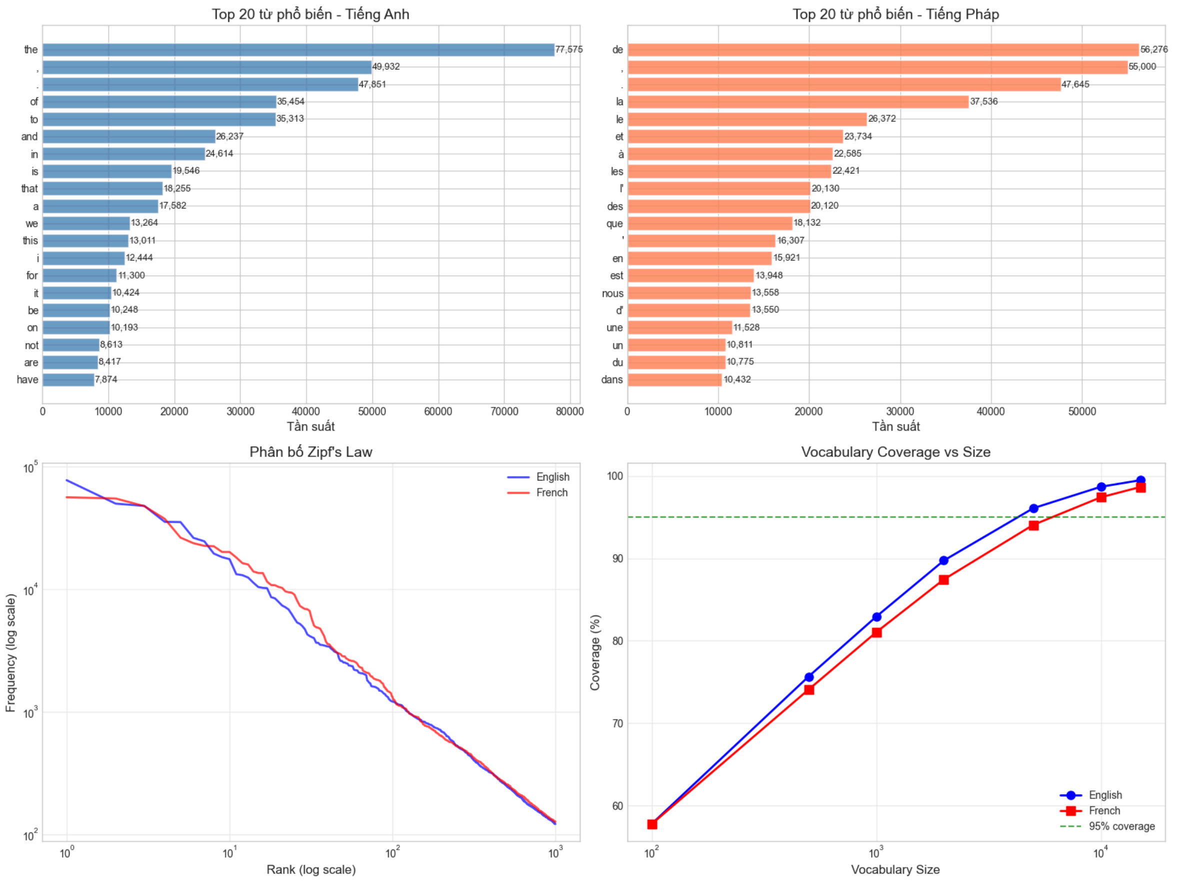
```
📈 Thống kê độ dài câu:
   English - Mean: 23.2, Median: 22.0, Max: 50
   French  - Mean: 25.2, Median: 24.0, Max: 50
✅ Đã lưu: data_length_distribution.png
```

In [7]:
```
# ============================================================
# 2. PHÂN TÍCH TỪ VỰNG (VOCABULARY ANALYSIS)
# ============================================================

print("📊 Đang phân tích từ vựng...")

# Đếm tần suất từ
en_word_freq = Counter()
fr_word_freq = Counter()

for src, tgt in train_data[:50000]:
    en_word_freq.update(tokenizers[SRC_LANGUAGE](src.lower()))
    fr_word_freq.update(tokenizers[TGT_LANGUAGE](tgt.lower()))

fig, axes = plt.subplots(2, 2, figsize=(16, 12))

# Top 20 từ phổ biến nhất - English
ax1 = axes[0, 0]
```

```python
top_en = en_word_freq.most_common(20)
words_en, counts_en = zip(*top_en)
bars1 = ax1.barh(range(len(words_en)), counts_en, color='steelblue', alpha=0.8)
ax1.set_yticks(range(len(words_en)))
ax1.set_yticklabels(words_en)
ax1.invert_yaxis()
ax1.set_xlabel('Tần suất', fontsize=12)
ax1.set_title('Top 20 từ phổ biến - Tiếng Anh', fontsize=14)
for bar, count in zip(bars1, counts_en):
    ax1.text(bar.get_width() + 100, bar.get_y() + bar.get_height()/2,
             f'{count:,}', va='center', fontsize=9)

# Top 20 từ phổ biến nhất - French
ax2 = axes[0, 1]
top_fr = fr_word_freq.most_common(20)
words_fr, counts_fr = zip(*top_fr)
bars2 = ax2.barh(range(len(words_fr)), counts_fr, color='coral', alpha=0.8)
ax2.set_yticks(range(len(words_fr)))
ax2.set_yticklabels(words_fr)
ax2.invert_yaxis()
ax2.set_xlabel('Tần suất', fontsize=12)
ax2.set_title('Top 20 từ phổ biến - Tiếng Pháp', fontsize=14)
for bar, count in zip(bars2, counts_fr):
    ax2.text(bar.get_width() + 100, bar.get_y() + bar.get_height()/2,
             f'{count:,}', va='center', fontsize=9)

# Phân bố tần suất từ (Zipf's Law)
ax3 = axes[1, 0]
en_freqs = sorted(en_word_freq.values(), reverse=True)[:1000]
fr_freqs = sorted(fr_word_freq.values(), reverse=True)[:1000]
ax3.loglog(range(1, len(en_freqs)+1), en_freqs, 'b-', alpha=0.7, label='English'
ax3.loglog(range(1, len(fr_freqs)+1), fr_freqs, 'r-', alpha=0.7, label='French',
ax3.set_xlabel('Rank (log scale)', fontsize=12)
ax3.set_ylabel('Frequency (log scale)', fontsize=12)
ax3.set_title("Phân bố Zipf's Law", fontsize=14)
ax3.legend()
ax3.grid(True, alpha=0.3)

# Vocabulary coverage
ax4 = axes[1, 1]
vocab_sizes = [100, 500, 1000, 2000, 5000, 10000, 15000]
en_coverage = []
fr_coverage = []
total_en = sum(en_word_freq.values())
total_fr = sum(fr_word_freq.values())

for size in vocab_sizes:
    top_en_words = [w for w, _ in en_word_freq.most_common(size)]
    top_fr_words = [w for w, _ in fr_word_freq.most_common(size)]
    en_cov = sum(en_word_freq[w] for w in top_en_words) / total_en * 100
    fr_cov = sum(fr_word_freq[w] for w in top_fr_words) / total_fr * 100
    en_coverage.append(en_cov)
    fr_coverage.append(fr_cov)

ax4.plot(vocab_sizes, en_coverage, 'b-o', label='English', linewidth=2, markersi
ax4.plot(vocab_sizes, fr_coverage, 'r-s', label='French', linewidth=2, markersiz
ax4.axhline(y=95, color='green', linestyle='--', alpha=0.7, label='95% coverage'
ax4.set_xlabel('Vocabulary Size', fontsize=12)
ax4.set_ylabel('Coverage (%)', fontsize=12)
ax4.set_title('Vocabulary Coverage vs Size', fontsize=14)
```

```
ax4.legend()
ax4.grid(True, alpha=0.3)
ax4.set_xscale('log')

plt.tight_layout()
plt.savefig('vocabulary_analysis.png', dpi=150, bbox_inches='tight')
plt.show()

print(f"\n📈 Thống kê từ vựng:")
print(f"   English - Unique words: {len(en_word_freq):,}, Total tokens: {total_e
print(f"   French  - Unique words: {len(fr_word_freq):,}, Total tokens: {total_f
print("✅ Đã lưu: vocabulary_analysis.png")
```

```
📊 Đang phân tích từ vựng...
```

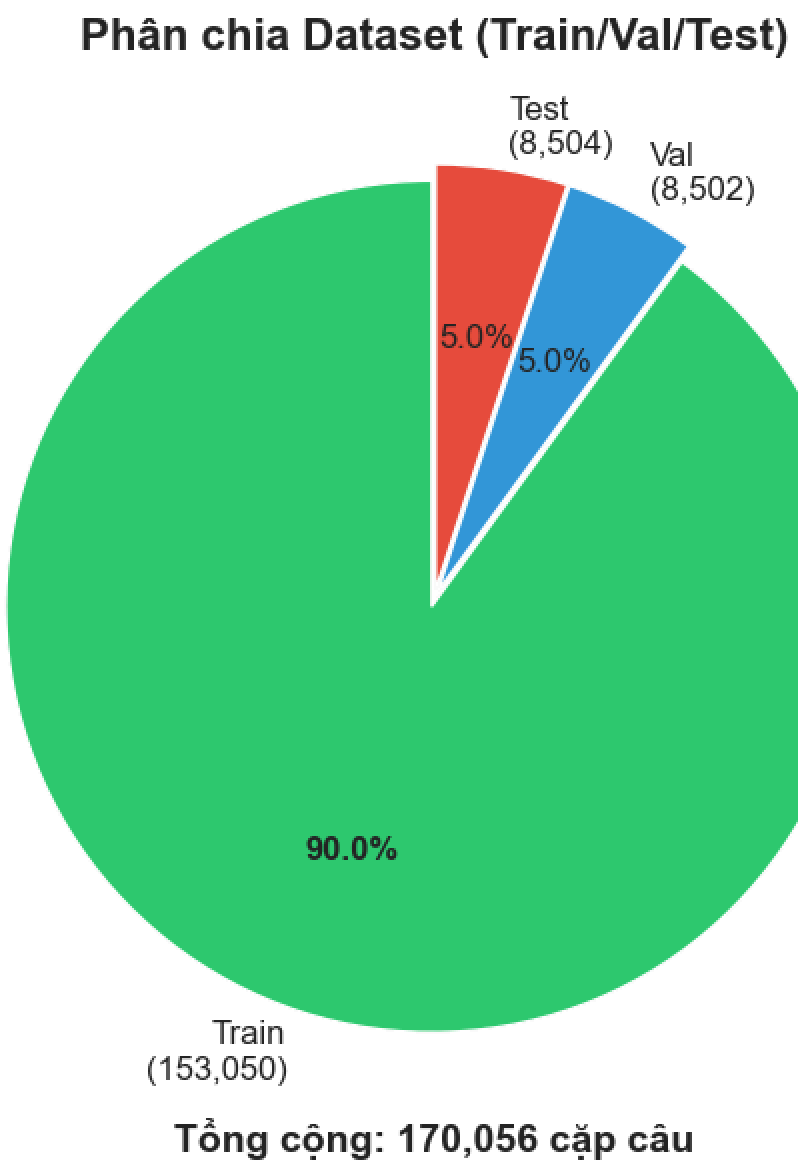


```
📈 Thống kê từ vựng:
   English - Unique words: 21,104, Total tokens: 1,159,513
   French  - Unique words: 29,232, Total tokens: 1,262,044
✅ Đã lưu: vocabulary_analysis.png
```

In [8]:
```
# ============================================================
# 3. XEM MẪU DỮ LIỆU
# ============================================================

print("📊 Mẫu dữ liệu từ Dataset:")
print("=" * 80)

# Hiển thị 5 cặp câu mẫu
for i in range(5):
    src, tgt = train_data[i]
    print(f"\n🔹 Cặp câu {i+1}:")
    print(f"   EN: {src}")
    print(f"   FR: {tgt}")

print("\n" + "=" * 80)
```

```python
# Thống kê dataset
fig, ax = plt.subplots(figsize=(10, 6))

# Pie chart cho phân chia train/val/test
sizes = [len(train_data), len(val_data), len(test_data)]
labels = [f'Train\n({len(train_data):,})', f'Val\n({len(val_data):,})', f'Test\r
colors = ['#2ecc71', '#3498db', '#e74c3c']
explode = (0.02, 0.02, 0.02)

wedges, texts, autotexts = ax.pie(sizes, explode=explode, labels=labels, colors=
                                   autopct='%1.1f%%', startangle=90,
                                   textprops={'fontsize': 12})
autotexts[0].set_fontweight('bold')

ax.set_title('Phân chia Dataset (Train/Val/Test)', fontsize=16, fontweight='bold

# Thêm tổng số
total = sum(sizes)
ax.text(0, -1.3, f'Tổng cộng: {total:,} cặp câu', ha='center', fontsize=14, font

plt.tight_layout()
plt.savefig('dataset_split.png', dpi=150, bbox_inches='tight')
plt.show()

print("✅ Đã lưu: dataset_split.png")
```

```
📊 Mẫu dữ liệu từ Dataset:
================================================================================

🔹 Cặp câu 1:
   EN: This recognition of culture as a vehicle for European identity should be s
hared not only by the European Parliament and the Commission, but also by the Cou
ncil, currently discussing a motion for a resolution from the Belgian Presidency 
on the role of culture in Europe.
   FR: Cette reconnaissance de la culture comme vecteur de l'identité européenne 
doit être partagée non seulement par le Parlement européen et la Commission, mais
aussi par le Conseil qui discute actuellement d'un projet de résolution de la pré
sidence belge sur la place de la culture en Europe.

🔹 Cặp câu 2:
   EN: There is an urgent need for a single patent that is legally valid througho
ut the EU, and there should be a single jurisdiction.
   FR: Les inventeurs ont besoin d'urgence de disposer d'un brevet unique, juridi
quement valide dans toute l'Union européenne, ainsi que d'une juridiction unique.

🔹 Cặp câu 3:
   EN: It is self-evident that liberalisation is not an end in itself, but only a
means towards improving services.
   FR: Il va de soi que la libéralisation n'est pas une fin en soi mais ne consti
tue qu'un moyen pour améliorer les services.

🔹 Cặp câu 4:
   EN: Very briefly on the subject of swine fever: the Commission is of the same 
mind as those who feel that after the last epidemic, there is an urgent need to r
evise the relevant legal provisions.
   FR: Très brièvement sur la sujet de la peste porcine : la Commission estime au
ssi qu'il y a maintenant un besoin urgent, après la dernière épidémie, de retrava
iller les dispositions juridiques en la matière.

🔹 Cặp câu 5:
   EN: The Ombudsman has already addressed many of the points made in my report.
   FR: Le Médiateur a déjà abordé de nombreux éléments repris dans mon rapport.

================================================================================
```



✅ Đã lưu: dataset_split.png

4. Mô hình Seq2Seq với Attention (Bahdanau)

Kiến trúc:

- **Encoder**: Bidirectional LSTM với pack_padded_sequence
- **Attention**: Bahdanau (Additive) Attention
- **Decoder**: LSTM với context vector từ attention

In [9]:
```
# ============================================================
# ENCODER (Bidirectional LSTM)
# ============================================================

class Encoder(nn.Module):
    """
    Encoder với Bidirectional LSTM
    - Sử dụng pack_padded_sequence để xử lý hiệu quả
    - Output: encoder_outputs (cho attention) và hidden state
    """

    def __init__(self, input_dim, emb_dim, hid_dim, n_layers, dropout):
        super().__init__()

        self.hid_dim = hid_dim
        self.n_layers = n_layers

        self.embedding = nn.Embedding(input_dim, emb_dim, padding_idx=PAD_IDX)

        # Bidirectional LSTM
        self.rnn = nn.LSTM(emb_dim, hid_dim, n_layers,
                           dropout=dropout if n_layers > 1 else 0,
                           bidirectional=True,
                           batch_first=False)

        # Linear để kết hợp 2 hướng
        self.fc_hidden = nn.Linear(hid_dim * 2, hid_dim)
        self.fc_cell = nn.Linear(hid_dim * 2, hid_dim)

        self.dropout = nn.Dropout(dropout)

    def forward(self, src, src_lengths):
        # src: [src_len, batch_size]

        embedded = self.dropout(self.embedding(src))
        # embedded: [src_len, batch_size, emb_dim]

        # Pack sequence
        packed = pack_padded_sequence(embedded, src_lengths.cpu(), enforce_sorte

        # LSTM
        packed_outputs, (hidden, cell) = self.rnn(packed)

        # Unpack
        encoder_outputs, _ = pad_packed_sequence(packed_outputs)
        # encoder_outputs: [src_len, batch_size, hid_dim * 2]

        # Kết hợp hidden state từ 2 hướng
        # hidden: [n_layers * 2, batch_size, hid_dim]
        # -> [n_layers, batch_size, hid_dim]

        hidden = torch.tanh(self.fc_hidden(
            torch.cat((hidden[-2,:,:], hidden[-1,:,:]), dim=1)
        ))
        cell = torch.tanh(self.fc_cell(
            torch.cat((cell[-2,:,:], cell[-1,:,:]), dim=1)
        ))

        # Expand để match số layers của decoder
        hidden = hidden.unsqueeze(0).repeat(self.n_layers, 1, 1)
        cell = cell.unsqueeze(0).repeat(self.n_layers, 1, 1)

        return encoder_outputs, hidden, cell

print("✅ Encoder với Bidirectional LSTM đã định nghĩa!")
```

```
✅ Encoder với Bidirectional LSTM đã định nghĩa!
```

In [10]:

```
# ============================================================
# ATTENTION MECHANISM (Bahdanau / Additive)
# ============================================================
class Attention(nn.Module):
    """
    Bahdanau (Additive) Attention

    score(s_t, h_i) = v^T * tanh(W_s * s_t + W_h * h_i)

    Cho phép decoder "nhìn" vào toàn bộ encoder outputs
    """

    def __init__(self, enc_hid_dim, dec_hid_dim):
        super().__init__()

        # enc_hid_dim * 2 vì bidirectional
        self.attn = nn.Linear((enc_hid_dim * 2) + dec_hid_dim, dec_hid_dim)
        self.v = nn.Linear(dec_hid_dim, 1, bias=False)

    def forward(self, hidden, encoder_outputs, mask):
        """
        Args:
            hidden: [batch_size, dec_hid_dim] - decoder hidden state hiện tại
            encoder_outputs: [src_len, batch_size, enc_hid_dim * 2]
            mask: [batch_size, src_len] - mask cho padding

        Returns:
            attention_weights: [batch_size, src_len]
        """
        src_len = encoder_outputs.shape[0]
        batch_size = encoder_outputs.shape[1]

        # Repeat hidden state cho mỗi source token
        hidden = hidden.unsqueeze(1).repeat(1, src_len, 1)
        # hidden: [batch_size, src_len, dec_hid_dim]

        encoder_outputs = encoder_outputs.permute(1, 0, 2)
        # encoder_outputs: [batch_size, src_len, enc_hid_dim * 2]

        # Tính attention scores
        energy = torch.tanh(self.attn(torch.cat((hidden, encoder_outputs), dim=2
        # energy: [batch_size, src_len, dec_hid_dim]

        attention = self.v(energy).squeeze(2)
        # attention: [batch_size, src_len]

        # Áp dụng mask (đặt padding positions thành -inf)
        attention = attention.masked_fill(mask == 0, -1e10)

        return F.softmax(attention, dim=1)

print("✅ Bahdanau Attention đã định nghĩa!")
```

```
✅ Bahdanau Attention đã định nghĩa!
```

In [11]:
```
# ============================================================
# DECODER với ATTENTION
# ============================================================

class DecoderWithAttention(nn.Module):
    """
    Decoder sử dụng Attention mechanism
```

```python
    Tại mỗi bước:
    1. Tính attention weights dựa trên hidden state hiện tại
    2. Tính context vector (weighted sum của encoder outputs)
    3. Kết hợp context với embedding làm input cho LSTM
    """

    def __init__(self, output_dim, emb_dim, enc_hid_dim, dec_hid_dim, n_layers,
        super().__init__()

        self.output_dim = output_dim
        self.attention = attention
        self.n_layers = n_layers

        self.embedding = nn.Embedding(output_dim, emb_dim, padding_idx=PAD_IDX)

        # Input: embedding + context vector
        self.rnn = nn.LSTM((enc_hid_dim * 2) + emb_dim, dec_hid_dim, n_layers,
                           dropout=dropout if n_layers > 1 else 0)

        # Output layer
        self.fc_out = nn.Linear((enc_hid_dim * 2) + dec_hid_dim + emb_dim, outpu

        self.dropout = nn.Dropout(dropout)

    def forward(self, input, hidden, cell, encoder_outputs, mask):
        """
        Args:
            input: [batch_size] - token hiện tại
            hidden: [n_layers, batch_size, dec_hid_dim]
            cell: [n_layers, batch_size, dec_hid_dim]
            encoder_outputs: [src_len, batch_size, enc_hid_dim * 2]
            mask: [batch_size, src_len]
        """
        input = input.unsqueeze(0)
        # input: [1, batch_size]

        embedded = self.dropout(self.embedding(input))
        # embedded: [1, batch_size, emb_dim]

        # Tính attention weights (dùng top layer hidden)
        a = self.attention(hidden[-1], encoder_outputs, mask)
        # a: [batch_size, src_len]

        a = a.unsqueeze(1)
        # a: [batch_size, 1, src_len]

        encoder_outputs_permuted = encoder_outputs.permute(1, 0, 2)
        # encoder_outputs_permuted: [batch_size, src_len, enc_hid_dim * 2]

        # Context vector (weighted sum)
        context = torch.bmm(a, encoder_outputs_permuted)
        # context: [batch_size, 1, enc_hid_dim * 2]

        context = context.permute(1, 0, 2)
        # context: [1, batch_size, enc_hid_dim * 2]

        # Kết hợp embedding và context
        rnn_input = torch.cat((embedded, context), dim=2)
        # rnn_input: [1, batch_size, emb_dim + enc_hid_dim * 2]
        output, (hidden, cell) = self.rnn(rnn_input, (hidden, cell))
        # output: [1, batch_size, dec_hid_dim]

        # Loại bỏ dimension 0
        embedded = embedded.squeeze(0)
        output = output.squeeze(0)
        context = context.squeeze(0)

        # Prediction
        prediction = self.fc_out(torch.cat((output, context, embedded), dim=1))
        # prediction: [batch_size, output_dim]

        return prediction, hidden, cell, a.squeeze(1)

print("✅ Decoder với Attention đã định nghĩa!")
```

```
✅ Decoder với Attention đã định nghĩa!
```

In [12]:

```
# ============================================================
# SEQ2SEQ với ATTENTION
# ============================================================

class Seq2SeqAttention(nn.Module):
    """Seq2Seq với Attention mechanism"""

    def __init__(self, encoder, decoder, src_pad_idx, device):
        super().__init__()

        self.encoder = encoder
        self.decoder = decoder
        self.src_pad_idx = src_pad_idx
        self.device = device

    def create_mask(self, src):
        """Tạo mask cho source (1 = valid, 0 = padding)"""
        mask = (src != self.src_pad_idx).permute(1, 0)
        return mask

    def forward(self, src, src_lengths, trg, teacher_forcing_ratio=0.5):
        """
        Args:
            src: [src_len, batch_size]
            src_lengths: [batch_size]
            trg: [trg_len, batch_size]
            teacher_forcing_ratio: xác suất dùng ground truth
        """
        batch_size = src.shape[1]
        trg_len = trg.shape[0]
        trg_vocab_size = self.decoder.output_dim

        # Tensor lưu outputs
        outputs = torch.zeros(trg_len, batch_size, trg_vocab_size).to(self.devic

        # Tạo mask
        mask = self.create_mask(src)

        # Encode
        encoder_outputs, hidden, cell = self.encoder(src, src_lengths)

        # Decode
        input = trg[0, :]  # <sos> token

        for t in range(1, trg_len):
            output, hidden, cell, _ = self.decoder(input, hidden, cell, encoder_

            outputs[t] = output

            # Teacher forcing
            teacher_force = random.random() < teacher_forcing_ratio
            top1 = output.argmax(1)

            input = trg[t] if teacher_force else top1

        return outputs

print("✅ Seq2Seq với Attention đã định nghĩa!")
```

```
✅ Seq2Seq với Attention đã định nghĩa!
```

In [13]:
```
# ============================================================
# KHỞI TẠO MODEL
# ============================================================

INPUT_DIM = len(vocab_src)
OUTPUT_DIM = len(vocab_tgt)

# Khởi tạo các components
attention = Attention(HID_DIM, HID_DIM)

encoder = Encoder(INPUT_DIM, EMB_DIM, HID_DIM, N_LAYERS, DROPOUT)

decoder = DecoderWithAttention(OUTPUT_DIM, EMB_DIM, HID_DIM, HID_DIM, N_LAYERS,

model = Seq2SeqAttention(encoder, decoder, PAD_IDX, device).to(device)

# Khởi tạo weights
def init_weights(m):
    for name, param in m.named_parameters():
        if 'weight' in name:
            nn.init.normal_(param.data, mean=0, std=0.01)
        else:
            nn.init.constant_(param.data, 0)

model.apply(init_weights)

# Đếm parameters
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

print(f"✅ Model đã khởi tạo!")
print(f"   • Tổng parameters: {count_parameters(model):,}")
print(f"   • Input vocab: {INPUT_DIM:,}")
print(f"   • Output vocab: {OUTPUT_DIM:,}")
```

```
✅ Model đã khởi tạo!
   • Tổng parameters: 51,640,984
   • Input vocab: 15,000
   • Output vocab: 15,000
```

5. Huấn luyện Model

In [14]:
```
# ============================================================
# OPTIMIZER & LOSS
# ============================================================

optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
criterion = nn.CrossEntropyLoss(ignore_index=PAD_IDX)

# Learning rate scheduler
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0

# ============================================================
# TRAIN & EVALUATE FUNCTIONS
# ============================================================

def train_epoch(model, iterator, optimizer, criterion, clip):
    model.train()
    epoch_loss = 0

    for i, (src, trg, src_lengths) in enumerate(iterator):
        src = src.to(device)
        trg = trg.to(device)

        optimizer.zero_grad()

        output = model(src, src_lengths, trg)

        # Reshape cho loss
        output_dim = output.shape[-1]
        output = output[1:].view(-1, output_dim)
        trg = trg[1:].view(-1)

        loss = criterion(output, trg)
        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), clip)
        optimizer.step()

        epoch_loss += loss.item()

        # Progress
        if (i + 1) % 100 == 0:
            print(f"   Batch {i+1}/{len(iterator)} - Loss: {loss.item():.4f}", e

    return epoch_loss / len(iterator)

def evaluate(model, iterator, criterion):
    model.eval()
    epoch_loss = 0

    with torch.no_grad():
        for src, trg, src_lengths in iterator:
            src = src.to(device)
            trg = trg.to(device)

            output = model(src, src_lengths, trg, teacher_forcing_ratio=0)

            output_dim = output.shape[-1]
            output = output[1:].view(-1, output_dim)
            trg = trg[1:].view(-1)
            loss = criterion(output, trg)
            epoch_loss += loss.item()

    return epoch_loss / len(iterator)

def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs

print("✅ Training functions đã định nghĩa!")
```

```
✅ Training functions đã định nghĩa!
```

In [ ]:
```
# ============================================================
# MAIN TRAINING LOOP VỚI VISUALIZATION
# ============================================================

from IPython.display import clear_output

best_valid_loss = float('inf')
patience_counter = 0
train_losses = []
valid_losses = []
train_ppls = []
valid_ppls = []
learning_rates = []
epoch_times = []

print("=" * 60)
print("🚀 BẮT ĐẦU HUẤN LUYỆN MODEL VỚI ATTENTION")
print("=" * 60)
print(f"   • Device: {device}")
print(f"   • Epochs: {N_EPOCHS}")
print(f"   • Patience: {PATIENCE}")
print(f"   • Batches per epoch: {len(train_loader)}")
print("=" * 60)

for epoch in range(N_EPOCHS):
    start_time = time.time()

    if device.type == 'cuda':
        torch.cuda.empty_cache()

    train_loss = train_epoch(model, train_loader, optimizer, criterion, CLIP)
    valid_loss = evaluate(model, val_loader, criterion)

    end_time = time.time()
    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    # Lưu metrics
    train_losses.append(train_loss)
    valid_losses.append(valid_loss)
    train_ppls.append(math.exp(train_loss))
    valid_ppls.append(math.exp(valid_loss))
    learning_rates.append(optimizer.param_groups[0]['lr'])
    epoch_times.append(end_time - start_time)
    # Learning rate scheduling
    scheduler.step(valid_loss)

    # Check improvement
    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        patience_counter = 0
        torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'train_loss': train_loss,
            'valid_loss': valid_loss,
            'train_losses': train_losses,
            'valid_losses': valid_losses,
        }, 'best-model-attention.pth')
        status = "✅ SAVED BEST"
    else:
        patience_counter += 1
        status = f"⏳ No improvement ({patience_counter}/{PATIENCE})"

    print(f'Epoch {epoch+1:02}/{N_EPOCHS} | Time: {epoch_mins}m {epoch_secs}s |
    print(f'   Train Loss: {train_loss:.4f} | Train PPL: {math.exp(train_loss):8
    print(f'   Val. Loss:  {valid_loss:.4f} | Val. PPL:  {math.exp(valid_loss):8
    print(f'   LR: {optimizer.param_groups[0]["lr"]:.6f}')
    print("-" * 60)

    # Early stopping
    if patience_counter >= PATIENCE:
        print(f'\n⚠️ EARLY STOPPING sau {PATIENCE} epochs không cải thiện!')
        break

print("\n" + "=" * 60)
print("✅ HUẤN LUYỆN HOÀN TẤT!")
print(f"   Best Validation Loss: {best_valid_loss:.4f}")
print(f"   Total training time: {sum(epoch_times)/60:.1f} minutes")
print("=" * 60)
```

```
============================================================
🚀 BẮT ĐẦU HUẤN LUYỆN MODEL VỚI ATTENTION
============================================================
   • Device: cuda
   • Epochs: 15
   • Patience: 3
   • Batches per epoch: 2392
============================================================
   Batch 100/2392 - Loss: 6.0939
```

In [ ]:

```
# ============================================================
# BIỂU ĐỒ TRAINING METRICS CHI TIẾT
# ============================================================

if len(train_losses) > 0:
    fig, axes = plt.subplots(2, 3, figsize=(18, 10))
    epochs_range = range(1, len(train_losses) + 1)

    # 1. Loss curves
    ax1 = axes[0, 0]
    ax1.plot(epochs_range, train_losses, 'b-o', label='Train Loss', linewidth=2,
    ax1.plot(epochs_range, valid_losses, 'r-s', label='Val Loss', linewidth=2, m
    ax1.fill_between(epochs_range, train_losses, valid_losses, alpha=0.2, color=
    ax1.set_xlabel('Epoch', fontsize=12)
    ax1.set_ylabel('Loss', fontsize=12)
    ax1.set_title('Training & Validation Loss', fontsize=14, fontweight='bold')
    ax1.legend(fontsize=10)
    ax1.grid(True, alpha=0.3)

    # 2. Perplexity curves
    ax2 = axes[0, 1]
    ax2.plot(epochs_range, train_ppls, 'b-o', label='Train PPL', linewidth=2, ma
    ax2.plot(epochs_range, valid_ppls, 'r-s', label='Val PPL', linewidth=2, mark
    ax2.set_xlabel('Epoch', fontsize=12)
    ax2.set_ylabel('Perplexity', fontsize=12)
    ax2.set_title('Training & Validation Perplexity', fontsize=14, fontweight='b
    ax2.legend(fontsize=10)
    ax2.grid(True, alpha=0.3)

    # 3. Learning Rate Schedule
    ax3 = axes[0, 2]
    ax3.plot(epochs_range, learning_rates, 'g-^', linewidth=2, markersize=8)
    ax3.set_xlabel('Epoch', fontsize=12)
    ax3.set_ylabel('Learning Rate', fontsize=12)
    ax3.set_title('Learning Rate Schedule', fontsize=14, fontweight='bold')
    ax3.grid(True, alpha=0.3)
    ax3.set_yscale('log')

    # 4. Training Time per Epoch
    ax4 = axes[1, 0]
    bars = ax4.bar(epochs_range, epoch_times, color='teal', alpha=0.7, edgecolor
    ax4.axhline(y=np.mean(epoch_times), color='red', linestyle='--', linewidth=2
    ax4.set_xlabel('Epoch', fontsize=12)
    ax4.set_ylabel('Time (seconds)', fontsize=12)
    ax4.set_title('Training Time per Epoch', fontsize=14, fontweight='bold')
    ax4.legend()
    ax4.grid(True, alpha=0.3, axis='y')

    # 5. Loss Improvement
    ax5 = axes[1, 1]
    loss_improvement = [0] + [valid_losses[i-1] - valid_losses[i] for i in range
    colors = ['green' if x > 0 else 'red' for x in loss_improvement]
    ax5.bar(epochs_range, loss_improvement, color=colors, alpha=0.7, edgecolor='
    ax5.axhline(y=0, color='black', linewidth=1)
    ax5.set_xlabel('Epoch', fontsize=12)
    ax5.set_ylabel('Loss Improvement', fontsize=12)
    ax5.set_title('Validation Loss Improvement (Green=Better)', fontsize=14, fon
    ax5.grid(True, alpha=0.3, axis='y')

    # 6. Overfitting Gap
    ax6 = axes[1, 2]
    gap = [v - t for t, v in zip(train_losses, valid_losses)]
    ax6.fill_between(epochs_range, gap, alpha=0.5, color='orange')
    ax6.plot(epochs_range, gap, 'o-', color='darkorange', linewidth=2, markersiz
    ax6.axhline(y=0, color='black', linewidth=1, linestyle='--')
    ax6.set_xlabel('Epoch', fontsize=12)
    ax6.set_ylabel('Val Loss - Train Loss', fontsize=12)
    ax6.set_title('Overfitting Gap (Lower=Better)', fontsize=14, fontweight='bol
    ax6.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.savefig('training_metrics_detailed.png', dpi=150, bbox_inches='tight')
    plt.show()

    # Summary statistics
    print("\n📊 TRAINING SUMMARY:")
    print(f"   • Best Val Loss: {min(valid_losses):.4f} (Epoch {valid_losses.ind
    print(f"   • Best Val PPL: {min(valid_ppls):.2f}")
    print(f"   • Final Train Loss: {train_losses[-1]:.4f}")
    print(f"   • Final Val Loss: {valid_losses[-1]:.4f}")
    print(f"   • Total Training Time: {sum(epoch_times)/60:.1f} minutes")
    print(f"   • Avg Time per Epoch: {np.mean(epoch_times):.1f} seconds")
    print("✅ Đã lưu: training_metrics_detailed.png")
else:
    print("⚠️ Chưa có dữ liệu training để vẽ")
```

6. Beam Search Decoding

Beam Search xem xét nhiều khả năng cùng lúc thay vì chỉ chọn token có xác suất cao nhất (greedy).

In [ ]:

```
# ============================================================
# BEAM SEARCH DECODING
# ============================================================

def translate_sentence_greedy(sentence, model, vocab_src, vocab_tgt, tokenizer,
    """Greedy decoding (baseline)"""
    model.eval()

    # Tokenize
    if isinstance(sentence, str):
        tokens = tokenizer(sentence.lower())
    else:
        tokens = [t.lower() for t in sentence]

    # Encode
    src_indices = [SOS_IDX] + vocab_src.encode(tokens) + [EOS_IDX]
    src_tensor = torch.LongTensor(src_indices).unsqueeze(1).to(device)
    src_lengths = torch.LongTensor([len(src_indices)])

    # Create mask
    mask = model.create_mask(src_tensor)

    with torch.no_grad():
        encoder_outputs, hidden, cell = model.encoder(src_tensor, src_lengths)

    # Decode
    trg_indices = [SOS_IDX]

    for _ in range(max_len):
        trg_tensor = torch.LongTensor([trg_indices[-1]]).to(device)

        with torch.no_grad():
            output, hidden, cell, _ = model.decoder(trg_tensor, hidden, cell, en

        pred_token = output.argmax(1).item()
        trg_indices.append(pred_token)

        if pred_token == EOS_IDX:
            break
    # Decode to text
    trg_tokens = vocab_tgt.lookup_tokens(trg_indices[1:])  # Skip SOS
    if '<eos>' in trg_tokens:
        trg_tokens = trg_tokens[:trg_tokens.index('<eos>')]

    return trg_tokens


def translate_sentence_beam(sentence, model, vocab_src, vocab_tgt, tokenizer, de
    """Beam Search decoding"""
    model.eval()

    # Tokenize
    if isinstance(sentence, str):
        tokens = tokenizer(sentence.lower())
    else:
        tokens = [t.lower() for t in sentence]

    # Encode
    src_indices = [SOS_IDX] + vocab_src.encode(tokens) + [EOS_IDX]
    src_tensor = torch.LongTensor(src_indices).unsqueeze(1).to(device)
    src_lengths = torch.LongTensor([len(src_indices)])

    # Create mask
    mask = model.create_mask(src_tensor)

    with torch.no_grad():
        encoder_outputs, hidden, cell = model.encoder(src_tensor, src_lengths)

    # Initialize beams: (score, sequence, hidden, cell)
    beams = [(0, [SOS_IDX], hidden, cell)]
    completed = []

    for _ in range(max_len):
        new_beams = []

        for score, seq, h, c in beams:
            if seq[-1] == EOS_IDX:
                completed.append((score, seq))
                continue

            trg_tensor = torch.LongTensor([seq[-1]]).to(device)

            with torch.no_grad():
                output, new_h, new_c, _ = model.decoder(trg_tensor, h, c, encode

            # Get top-k tokens
            log_probs = F.log_softmax(output, dim=1)
            topk_probs, topk_indices = log_probs.topk(beam_size)

            for i in range(beam_size):
                new_score = score + topk_probs[0, i].item()
                new_seq = seq + [topk_indices[0, i].item()]
                new_beams.append((new_score, new_seq, new_h, new_c))

        # Keep top beam_size
        beams = sorted(new_beams, key=lambda x: x[0], reverse=True)[:beam_size]

        # Stop if all beams are completed
        if all(b[1][-1] == EOS_IDX for b in beams):
            break

    # Add remaining beams to completed
    completed.extend([(s, seq) for s, seq, _, _ in beams])

    # Get best sequence
    if completed:
        best_score, best_seq = max(completed, key=lambda x: x[0] / len(x[1]))  #
    else:
        best_seq = beams[0][1]

    # Decode to text
    trg_tokens = vocab_tgt.lookup_tokens(best_seq[1:])  # Skip SOS
    if '<eos>' in trg_tokens:
        trg_tokens = trg_tokens[:trg_tokens.index('<eos>')]

    return trg_tokens


def translate(sentence: str) -> str:
    """
    Hàm translate theo yêu cầu đề bài
    Sử dụng Beam Search để dịch
    """
    tokens = translate_sentence_beam(
        sentence, model, vocab_src, vocab_tgt,
        tokenizers[SRC_LANGUAGE], device,
        beam_size=BEAM_SIZE
    )
    return ' '.join(tokens)


print("✅ Beam Search và translate functions đã định nghĩa!")
```

7. Đánh giá & So sánh

In [ ]:
```
# ============================================================
# LOAD BEST MODEL & ĐÁNH GIÁ
# ============================================================

# Load best model
try:
    checkpoint = torch.load('best-model-attention.pth', map_location=device)
    model.load_state_dict(checkpoint['model_state_dict'])
    print(f"✅ Đã load model từ epoch {checkpoint['epoch']+1}")
    print(f"   Train Loss: {checkpoint['train_loss']:.3f}")
    print(f"   Valid Loss: {checkpoint['valid_loss']:.3f}")
except FileNotFoundError:
    print("⚠️ Không tìm thấy checkpoint, sử dụng model hiện tại")

# Load spacy cho tiếng Pháp (để tokenize reference)
spacy_fr = spacy.load('fr_core_news_sm')
```

In [ ]:
```
# ============================================================
# SO SÁNH GREEDY vs BEAM SEARCH
# ============================================================
```

```python
from torchtext.data.metrics import bleu_score as torchtext_bleu
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction

def calculate_bleu_comparison(test_data, model, n_samples=200):
    """So sánh BLEU score giữa Greedy và Beam Search"""

    greedy_outputs = []
    beam_outputs = []
    targets = []

    print(f"📊 Đang tính BLEU trên {n_samples} câu...")

    for i, (src, tgt) in enumerate(test_data[:n_samples]):
        # Greedy
        greedy_pred = translate_sentence_greedy(
            src, model, vocab_src, vocab_tgt,
            tokenizers[SRC_LANGUAGE], device
        )

        # Beam Search
        beam_pred = translate_sentence_beam(
            src, model, vocab_src, vocab_tgt,
            tokenizers[SRC_LANGUAGE], device,
            beam_size=BEAM_SIZE
        )

        # Target
        tgt_tokens = [t.text.lower() for t in spacy_fr(tgt)]

        greedy_outputs.append(greedy_pred)
        beam_outputs.append(beam_pred)
        targets.append([tgt_tokens])

        if (i + 1) % 50 == 0:
            print(f"   Đã xử lý {i+1}/{n_samples}...")

    # Tính BLEU
    greedy_bleu = torchtext_bleu(greedy_outputs, targets)
    beam_bleu = torchtext_bleu(beam_outputs, targets)

    return greedy_bleu, beam_bleu

# Tính BLEU
greedy_bleu, beam_bleu = calculate_bleu_comparison(test_data, model, n_samples=2

print("\n" + "=" * 60)
print("📊 KẾT QUẢ SO SÁNH")
print("=" * 60)
print(f"   Greedy Decoding BLEU:     {greedy_bleu*100:.2f}")
print(f"   Beam Search BLEU (k={BEAM_SIZE}):  {beam_bleu*100:.2f}")
print(f"   Cải thiện:                +{(beam_bleu - greedy_bleu)*100:.2f}")
print("=" * 60)
```

In [ ]:

```python
# ============================================================
# TEST DỊCH VỚI CÁC CÂU MẪU
# ============================================================

print("=" * 70)
print("🔤 TEST HÀM translate()")
print("=" * 70)
```

```
test_sentences = [
    "The weather is beautiful today.",
    "I love learning new languages.",
    "The European Parliament is meeting today.",
    "We need to protect the environment.",
    "Education is very important for children."
]

for sentence in test_sentences:
    result = translate(sentence)
    print(f"\n📥 EN: {sentence}")
    print(f"📤 FR: {result}")

print("\n" + "=" * 70)
```

8. Biểu đồ & Visualizations

In [ ]:
```
# ============================================================
# BIỂU ĐỒ TRAINING LOSS (TỪ CHECKPOINT)
# ============================================================

# Load losses từ checkpoint nếu có
try:
    checkpoint = torch.load('best-model-attention.pth', map_location=device)
    if 'train_losses' in checkpoint:
        train_losses = checkpoint['train_losses']
        valid_losses = checkpoint['valid_losses']
        print("✅ Đã load training history từ checkpoint")
except:
    pass

if len(train_losses) > 0:
    fig, axes = plt.subplots(1, 2, figsize=(14, 5))
    epochs_range = range(1, len(train_losses) + 1)

    # Loss plot
    ax1 = axes[0]
    ax1.plot(epochs_range, train_losses, 'b-o', label='Train Loss', linewidth=2.
    ax1.plot(epochs_range, valid_losses, 'r-s', label='Validation Loss', linewid

    # Highlight best epoch
    best_epoch = valid_losses.index(min(valid_losses)) + 1
    ax1.axvline(x=best_epoch, color='green', linestyle='--', linewidth=2, alpha=
    ax1.scatter([best_epoch], [min(valid_losses)], color='green', s=200, zorder=

    ax1.set_xlabel('Epoch', fontsize=13)
    ax1.set_ylabel('Loss', fontsize=13)
    ax1.set_title('Training & Validation Loss', fontsize=15, fontweight='bold')
    ax1.legend(fontsize=11, loc='upper right')
    ax1.grid(True, alpha=0.3)
    ax1.set_xticks(epochs_range)

    # Perplexity plot
    ax2 = axes[1]
    train_ppl = [math.exp(l) for l in train_losses]
    val_ppl = [math.exp(l) for l in valid_losses]
    ax2.plot(epochs_range, train_ppl, 'b-o', label='Train PPL', linewidth=2.5, m
    ax2.plot(epochs_range, val_ppl, 'r-s', label='Validation PPL', linewidth=2.5
    ax2.axvline(x=best_epoch, color='green', linestyle='--', linewidth=2, alpha=
    ax2.scatter([best_epoch], [min(val_ppl)], color='green', s=200, zorder=5, ma

    ax2.set_xlabel('Epoch', fontsize=13)
    ax2.set_ylabel('Perplexity', fontsize=13)
    ax2.set_title('Training & Validation Perplexity', fontsize=15, fontweight='b
    ax2.legend(fontsize=11, loc='upper right')
    ax2.grid(True, alpha=0.3)
    ax2.set_xticks(epochs_range)

    plt.tight_layout()
    plt.savefig('training_curves.png', dpi=200, bbox_inches='tight')
    plt.show()
    print("✅ Đã lưu: training_curves.png")
else:
    print("⚠️ Chưa có dữ liệu training để vẽ")
```

In [ ]:

```
# ============================================================
# VISUALIZE ATTENTION WEIGHTS - ENHANCED
# ============================================================

def visualize_attention(sentence, model, vocab_src, vocab_tgt, tokenizer, device
    """Visualize attention weights cho một câu với enhanced display"""
    model.eval()

    # Tokenize
    tokens = tokenizer(sentence.lower())
    src_indices = [SOS_IDX] + vocab_src.encode(tokens) + [EOS_IDX]
    src_tensor = torch.LongTensor(src_indices).unsqueeze(1).to(device)
    src_lengths = torch.LongTensor([len(src_indices)])

    # Create mask
    mask = model.create_mask(src_tensor)

    with torch.no_grad():
        encoder_outputs, hidden, cell = model.encoder(src_tensor, src_lengths)

    # Decode và lưu attention weights
    trg_indices = [SOS_IDX]
    attentions = []

    for _ in range(50):
        trg_tensor = torch.LongTensor([trg_indices[-1]]).to(device)

        with torch.no_grad():
            output, hidden, cell, attention = model.decoder(
                trg_tensor, hidden, cell, encoder_outputs, mask
            )

        attentions.append(attention.cpu().numpy())

        pred_token = output.argmax(1).item()
        trg_indices.append(pred_token)

        if pred_token == EOS_IDX:
            break

    # Convert to arrays
    attentions = np.concatenate(attentions, axis=0)
    src_tokens = ['<sos>'] + tokens + ['<eos>']
    trg_tokens = vocab_tgt.lookup_tokens(trg_indices)
    if '<eos>' in trg_tokens:
        eos_idx = trg_tokens.index('<eos>')
        trg_tokens = trg_tokens[:eos_idx + 1]

    # Plot
    fig, ax = plt.subplots(figsize=(max(12, len(src_tokens)*0.8), max(8, len(trg

    # Chỉ lấy phần attention tương ứng
    attn_to_plot = attentions[:len(trg_tokens)-1, :len(src_tokens)]

    # Heatmap với colormap đẹp hơn
    im = ax.imshow(attn_to_plot, cmap='YlOrRd', aspect='auto')

    # Thêm text annotation cho các giá trị cao
    for i in range(attn_to_plot.shape[0]):
        for j in range(attn_to_plot.shape[1]):
            if attn_to_plot[i, j] > 0.2:  # Chỉ hiện giá trị > 0.2
                ax.text(j, i, f'{attn_to_plot[i, j]:.2f}', ha='center', va='cent
                       fontsize=8, color='black' if attn_to_plot[i, j] < 0.5 els

    ax.set_xticks(range(len(src_tokens)))
    ax.set_xticklabels(src_tokens, rotation=45, ha='right', fontsize=11)
    ax.set_yticks(range(len(trg_tokens)-1))
    ax.set_yticklabels(trg_tokens[1:], fontsize=11)

    ax.set_xlabel('Source (English)', fontsize=13)
    ax.set_ylabel('Target (French)', fontsize=13)
    ax.set_title('Attention Weights Heatmap', fontsize=15, fontweight='bold')

    # Colorbar
    cbar = plt.colorbar(im, ax=ax, shrink=0.8)
    cbar.set_label('Attention Weight', fontsize=11)

    plt.tight_layout()

    if save_name:
        plt.savefig(save_name, dpi=200, bbox_inches='tight')
    plt.show()

    return ' '.join(trg_tokens[1:-1]) if '<eos>' in vocab_tgt.lookup_tokens(trg_


# Test với nhiều câu
test_sentences_attn = [
    "The weather is beautiful today.",
    "I love learning new languages.",
    "The European Parliament meets every month."
]

print("=" * 70)
print("🔍 ATTENTION VISUALIZATION")
print("=" * 70)

for i, sentence in enumerate(test_sentences_attn):
    result = visualize_attention(
        sentence, model, vocab_src, vocab_tgt,
        tokenizers[SRC_LANGUAGE], device,
        save_name=f'attention_example_{i+1}.png'
    )
    print(f"\n📥 Input: {sentence}")
    print(f"📤 Output: {result}")
    print(f"✅ Đã lưu: attention_example_{i+1}.png")
    print("-" * 70)
```

9. Phân tích 5 Ví dụ & So sánh với Baseline

In [ ]:
```
# ============================================================
# PHÂN TÍCH 5 VÍ DỤ DỊCH VỚI BIỂU ĐỒ
# ============================================================

from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction

smoother = SmoothingFunction()

# Chọn 5 câu từ test set
random.seed(42)
sample_indices = random.sample(range(len(test_data)), 5)

print("=" * 80)
print("📊 PHÂN TÍCH 5 VÍ DỤ DỊCH (ADVANCED MODEL)")
print("=" * 80)

results = []

for idx, i in enumerate(sample_indices, 1):
    src_sentence = test_data[i][0]
    ref_sentence = test_data[i][1]

    # Greedy
    greedy_tokens = translate_sentence_greedy(
        src_sentence, model, vocab_src, vocab_tgt,
        tokenizers[SRC_LANGUAGE], device
    )
    greedy_pred = ' '.join(greedy_tokens)

    # Beam Search
    beam_tokens = translate_sentence_beam(
        src_sentence, model, vocab_src, vocab_tgt,
        tokenizers[SRC_LANGUAGE], device, beam_size=BEAM_SIZE
    )
    beam_pred = ' '.join(beam_tokens)

    # Reference tokens
    ref_tokens = [t.text.lower() for t in spacy_fr(ref_sentence)]

    # BLEU scores
    greedy_bleu = sentence_bleu([ref_tokens], greedy_tokens, smoothing_function=
    beam_bleu = sentence_bleu([ref_tokens], beam_tokens, smoothing_function=smoc

    results.append({
        'idx': idx,
        'src': src_sentence,
        'ref': ref_sentence,
        'greedy': greedy_pred,
        'beam': beam_pred,
        'greedy_bleu': greedy_bleu,
        'beam_bleu': beam_bleu
    })

    print(f"\n--- Ví dụ {idx} ---")
    print(f"📥 Source (EN):     {src_sentence[:80]}...")
    print(f"🎯 Reference (FR):  {ref_sentence[:80]}...")
    print(f"🤖 Greedy:          {greedy_pred[:80]}...")
    print(f"🔍 Beam (k={BEAM_SIZE}):       {beam_pred[:80]}...")
    print(f"📊 BLEU - Greedy: {greedy_bleu:.4f} | Beam: {beam_bleu:.4f}")

    better = "Beam ✓" if beam_bleu > greedy_bleu else ("Greedy ✓" if greedy_ble
    print(f"   ➡️ {better}")

# Vẽ biểu đồ so sánh
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Bar chart so sánh BLEU
ax1 = axes[0]
x = np.arange(5)
width = 0.35
greedy_scores = [r['greedy_bleu'] for r in results]
beam_scores = [r['beam_bleu'] for r in results]

bars1 = ax1.bar(x - width/2, greedy_scores, width, label='Greedy', color='steelb
bars2 = ax1.bar(x + width/2, beam_scores, width, label=f'Beam (k={BEAM_SIZE})',

ax1.set_xlabel('Ví dụ', fontsize=12)
ax1.set_ylabel('BLEU Score', fontsize=12)
ax1.set_title('So sánh BLEU: Greedy vs Beam Search', fontsize=14, fontweight='bo
ax1.set_xticks(x)
ax1.set_xticklabels([f'Ex {i+1}' for i in range(5)])
ax1.legend()
ax1.grid(True, alpha=0.3, axis='y')

# Thêm giá trị lên bar
for bar, score in zip(bars1, greedy_scores):
    ax1.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.01,
             f'{score:.3f}', ha='center', va='bottom', fontsize=9)
for bar, score in zip(bars2, beam_scores):
    ax1.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.01,
             f'{score:.3f}', ha='center', va='bottom', fontsize=9)

# Improvement chart
ax2 = axes[1]
improvements = [r['beam_bleu'] - r['greedy_bleu'] for r in results]
colors = ['green' if imp > 0 else 'red' if imp < 0 else 'gray' for imp in improv
bars = ax2.bar(x, improvements, color=colors, alpha=0.8, edgecolor='white')
ax2.axhline(y=0, color='black', linewidth=1)
ax2.set_xlabel('Ví dụ', fontsize=12)
ax2.set_ylabel('BLEU Improvement (Beam - Greedy)', fontsize=12)
ax2.set_title('Beam Search Improvement', fontsize=14, fontweight='bold')
ax2.set_xticks(x)
ax2.set_xticklabels([f'Ex {i+1}' for i in range(5)])
ax2.grid(True, alpha=0.3, axis='y')

# Thêm annotation
for bar, imp in zip(bars, improvements):
    color = 'green' if imp > 0 else 'red'
    sign = '+' if imp > 0 else ''
    ax2.text(bar.get_x() + bar.get_width()/2,
             bar.get_height() + (0.005 if imp >= 0 else -0.015),
             f'{sign}{imp:.3f}', ha='center', va='bottom' if imp >= 0 else 'top'
             fontsize=10, color=color, fontweight='bold')

plt.tight_layout()
plt.savefig('bleu_comparison.png', dpi=150, bbox_inches='tight')
plt.show()

# Summary
avg_greedy = np.mean(greedy_scores)
avg_beam = np.mean(beam_scores)
print("\n" + "=" * 80)
print("📈 SUMMARY:")
print(f"   Average Greedy BLEU: {avg_greedy:.4f}")
print(f"   Average Beam BLEU:   {avg_beam:.4f}")
print(f"   Average Improvement: {avg_beam - avg_greedy:+.4f} ({(avg_beam/avg_gre
print("✅ Đã lưu: bleu_comparison.png")
print("=" * 80)
```

10. Tổng kết & Kết luận

```
In [ ]:
# ============================================================
# TỔNG KẾT KẾT QUẢ VỚI BIỂU ĐỒ
# ============================================================

print("=" * 80)
print("📋 TỔNG KẾT PHẦN MỞ RỘNG (ADVANCED)")
print("=" * 80)

# Tạo figure tổng hợp
fig = plt.figure(figsize=(16, 10))

# 1. Model Architecture Summary (text box)
ax1 = fig.add_subplot(2, 2, 1)
ax1.axis('off')
arch_text = """
🏗️ MODEL ARCHITECTURE

┌─────────────────────────────────────┐
│  ENCODER (Bidirectional LSTM)       │
│  • Input: English tokens            │
│  • Embedding: 256 dim               │
│  • Hidden: 512 dim x 2 directions   │
│  • Layers: 2                        │
│  • Output: encoder_outputs + states │
└─────────────────────────────────────┘
            ↓
┌─────────────────────────────────────┐
│  BAHDANAU ATTENTION                 │
│  • Query: decoder hidden state      │
│  • Key/Value: encoder outputs       │
│  • Output: context vector           │
└─────────────────────────────────────┘
            ↓
┌─────────────────────────────────────┐
│  DECODER (LSTM + Attention)         │
│  • Input: embedding + context       │
│  • Hidden: 512 dim                  │
```

```python
│  • Layers: 2                          │
│  • Output: French tokens              │
└───────────────────────────────────────┘
"""
ax1.text(0.1, 0.5, arch_text, transform=ax1.transAxes, fontsize=10,
         verticalalignment='center', fontfamily='monospace',
         bbox=dict(boxstyle='round', facecolor='lightblue', alpha=0.5))
ax1.set_title('Model Architecture', fontsize=14, fontweight='bold')

# 2. Training Stats
ax2 = fig.add_subplot(2, 2, 2)
ax2.axis('off')

if len(train_losses) > 0:
    stats_text = f"""
📊 TRAINING STATISTICS

• Dataset: Europarl (WMT 2014)
• Training samples: {len(train_data):,}
• Validation samples: {len(val_data):,}
• Test samples: {len(test_data):,}

• Total parameters: {count_parameters(model):,}
• Embedding dim: {EMB_DIM}
• Hidden dim: {HID_DIM}
• LSTM Layers: {N_LAYERS}

• Epochs trained: {len(train_losses)}
• Best Val Loss: {min(valid_losses):.4f}
• Best Val PPL: {math.exp(min(valid_losses)):.2f}
• Training time: {sum(epoch_times)/60:.1f} minutes
"""
else:
    stats_text = """
📊 TRAINING STATISTICS

• Dataset: Europarl (WMT 2014)
• Training samples: --
• Model not yet trained

Run training cells to see statistics.
"""

ax2.text(0.1, 0.5, stats_text, transform=ax2.transAxes, fontsize=11,
         verticalalignment='center', fontfamily='monospace',
         bbox=dict(boxstyle='round', facecolor='lightyellow', alpha=0.5))
ax2.set_title('Training Statistics', fontsize=14, fontweight='bold')

# 3. Features implemented
ax3 = fig.add_subplot(2, 2, 3)
ax3.axis('off')

features_text = """
✅ CÁC CẢI TIẾN ĐÃ TRIỂN KHAI

1. ✅ DATASET LỚN (Europarl ~2M sentences)
   → Đa dạng domain (politics, law...)

2. ✅ BIDIRECTIONAL ENCODER
   → Hiểu context cả 2 chiều
```

```
3. ✅ BAHDANAU ATTENTION
   → Decoder "nhìn" toàn bộ encoder outputs
   → Giải quyết vấn đề câu dài

4. ✅ PACK_PADDED_SEQUENCE
   → Xử lý padding hiệu quả
   → Tăng tốc training

5. ✅ BEAM SEARCH (k=5)
   → Xem xét nhiều hypotheses
   → Giảm lỗi lặp từ

6. ✅ ATTENTION VISUALIZATION
   → Hiểu model đang "chú ý" vào đâu
"""

ax3.text(0.05, 0.5, features_text, transform=ax3.transAxes, fontsize=10,
         verticalalignment='center', fontfamily='monospace',
         bbox=dict(boxstyle='round', facecolor='lightgreen', alpha=0.5))
ax3.set_title('Implemented Features', fontsize=14, fontweight='bold')

# 4. Files generated
ax4 = fig.add_subplot(2, 2, 4)
ax4.axis('off')

files_text = """
📁 CÁC FILE ĐÃ TẠO

Model & Checkpoints:
  • best-model-attention.pth

Data Visualizations:
  • data_length_distribution.png
  • vocabulary_analysis.png
  • dataset_split.png

Training Visualizations:
  • training_metrics_detailed.png
  • training_curves.png

Analysis Visualizations:
  • attention_example_1.png
  • attention_example_2.png
  • attention_example_3.png
  • bleu_comparison.png

Summary:
  • final_summary.png
"""

ax4.text(0.1, 0.5, files_text, transform=ax4.transAxes, fontsize=10,
         verticalalignment='center', fontfamily='monospace',
         bbox=dict(boxstyle='round', facecolor='lavender', alpha=0.5))
ax4.set_title('Generated Files', fontsize=14, fontweight='bold')

plt.suptitle('🚀 NLP Advanced - Final Summary', fontsize=18, fontweight='bold',
plt.tight_layout()
plt.savefig('final_summary.png', dpi=150, bbox_inches='tight')
plt.show()
```

```python
print("\n" + "=" * 80)
print("✅ HOÀN THÀNH PHẦN MỞ RỘNG!")
print("=" * 80)
print("""
📝 ĐIỂM CỘNG (Bonus Points):

1. ✅ Dataset WMT 2014 (Europarl)     → +0.25đ
2. ✅ Attention Mechanism (Bahdanau)  → +0.25đ
3. ✅ Beam Search Decoding            → +0.25đ
4. ✅ Visualizations (Attention maps) → +0.25đ

   Tổng điểm cộng dự kiến: +1.0đ
""")
print("=" * 80)
```