实验1：基于 Transformer 的机器翻译

1. 实验目标

- 熟悉并实操 PyTorch 原生的 Transformer API
  - `nn.Transformer`
  - `nn.TransformerEncoderLayer`
  - `nn.TransformerDecoderLayer`。
- 完成中英小规模机器翻译的实现流程
  - 数据预处理 → tokenizer → 模型搭建→ 训练 → 评估（BLEU）。
- 正确理解各种 mask在 `nn.Transformer` 中的用法
  - padding mask
  - causal mask
  - memory padding mask
- 做至少一项消融或拓展实验
  - 如：position encoding 比较 / head 数量改变 / label smoothing / beam search

2. 实验环境

- Python 版本：3.10+
- 必需库（建议版本）：
  - `torch`（GPU 环境下安装对应 CUDA 版本）
  - `sentencepiece` 或 `tokenizers`（用于 BPE / SentencePiece）
  - `sacrebleu`或 `nltk`（计算 BLEU 指标）
  - `tqdm`, `numpy`, `pandas`, `matplotlib`（可视化）
- 建议使用 GPU 环境

3. 数据集

- 共有30k左右的翻译数据对，每对翻译数据在同一行：第一列是英文，第二列是中文，用`\t`分割
- 下载地址：https://www.manythings.org/anki/cmn-eng.zip
- 划分方式：随机划分

- 训练集+验证：80%
- 测试集：20%

4. 实验任务

- 实现中英翻译（Chinese → English 或 English → Chinese，任选方向），训练并给出 test 集 BLEU指标。
- **要求**：
  - 使用 PyTorch 原生的 `nn.Transformer` 完成
    - 不得使用外部封装的 Transformer 模型，如 HuggingFace 等。
  - 解释所有 mask 在 `nn.Transformer` 中的作用方式（见下节细节）。
  - 评价指标：BLEU。
- **可选拓展**：beam search，label smoothing，lr schedule，embedding 共享，不同position encoding 方式比较等。

5. PyTorch `nn.Transformer` 实现要点

PyTorch 的 `nn.Transformer` 以及 `nn.TransformerEncoderLayer/nn.TransformerDecoderLayer` 的输入输出格式与 mask 使用方法有严格要求。下面按要点列出：

（1）Embedding 与位置编码

- 输入：将 token ids 转为 `d_model` 维 embedding（`nn.Embedding`），并加上 position encoding（**可选**可学习 `nn.Embedding`）。
- 注意：若共享 src/tgt embedding 或共享 output projection，根据实现调整（共享可节省参数）。

（2）Masks 的区别与构造

- **tgt_mask (causal mask)**：保证 decoder 在时间步 t 只能看到 ≤ t 的 token。通常用 `torch.triu(torch.ones(T, T) * float('-inf'), diagonal=1)` 构建（或布尔掩码）。传入 `tgt_mask`。
- **src_mask / memory_mask**：一般不需要（除非要阻断 src 内部位置之间的注意力）。通常设置为 `None`。
- **key_padding_mask（src_key_padding_mask / tgt_key_padding_mask / memory_key_padding_mask）**：是 (N, S) 的布尔 mask，标注 padding tokens；用于在 attention 计算时屏蔽 padding positions。**非常重要**：必须传入正确的 padding mask 否则模型会学习到错误对齐。
  - `src_key_padding_mask` 用于 encoder self-attn
  - `tgt_key_padding_mask` 用于 decoder self-attn
  - `memory_key_padding_mask` 用于 encoder-decoder attention（key/value 来自 encoder）

（3）输出 head 与 loss

- Transformer 输出后通常接线性层 `nn.Linear(d_model, vocab_size)`
- 然后用 `nn.CrossEntropyLoss(ignore_index=pad_idx)` 计算损失（训练阶段使用 teacher forcing）。

6. 推荐起始超参数参考

- `d_model = 128`
- `nhead = 4`
- `dim_feedforward = 512`
- `num_encoder_layers = num_decoder_layers = 2`
- `dropout = 0.1`
- `vocab_size ≈ 8000`（使用 BPE / SentencePiece）
- `batch_size = 32`（视显存可减）
- `optimizer = Adam`（`lr = 3e-4` 起）
- `epochs = 10-30`（小数据集）
- `label_smoothing = 0.1`（可选）

> *说明：若在 CPU 上训练，可考虑将 d_model 降至 64、batch 减小到 8。*

7. 实验步骤

1. **准备数据**
   - 下载数据集（或抽样子集）；训练 SentencePiece/BPE（vocab ~8k）。
   - Tokenize 源/目标并生成 train/valid/test 文件（token ids）。
   - 生成 pad/cls/bos/eos 索引并写入 vocab 或 config。
2. **文件结构建议**
   - `data/`：原始与 tokenized 数据
   - `src/model.py`：模型定义
   - `src/train.py`：训练循环、日志输出等
   - `src/eval.py`：贪婪解码、beam search（可选）、BLEU 评估
   - `src/utils.py`：mask 构造、tokenizer 辅助等工具函数
3. **构造 mask 与 batch**
   - 在 batch 中得到 `src` (N, S) 和 `tgt` (N, T)，在调用 `nn.Transformer` 前把其转成 (S, N) 与 (T, N)。
   - 构造 `tgt_mask`（T, T causal）和 `src_key_padding_mask` / `tgt_key_padding_mask`（N, S / N, T）。
4. **训练**
   - 使用 teacher forcing，把 `tgt_input = tgt[:, :-1]` 传入 decoder，预测 `tgt_out = tgt[:, 1:]`。

- 计算交叉熵，忽略 `pad_idx`。记录 train/val loss 与 val BLEU。
- 按照 epoch 保存 checkpoint（保存最优 val BLEU 的模型）。

5. **推理（eval）**
   - 实现 greedy decode（循环地将上一步预测作为下一个 step 的 input）并输出句子。
   - 可选：实现 beam search（beam=4）并比较效果。
6. **消融 / 拓展实验（至少选一项并结果写入报告）**
   - 去掉 position encoding VS 使用 learnable position embedding。
   - 改变 `nhead`（2 vs 8）或 `d_model`（64 vs 256）。
   - label smoothing 比较。
   - 实现并比较 greedy VS beam search 的 BLEU 与样例输出。

8. 实验报告要求

1. **摘要（100–150 字）**：问题、方法、数据、主要结果。
2. **方法**：简要说明 `nn.Transformer` 的结构、mask 的传递（可以画一个小图说明 src/tgt/memory 与 mask 的对应关系）。
3. **数据与预处理**：数据来源、训练/验证/测试 大小、tokenizer 类型与 vocab 大小、过滤策略。
4. **实现细节与超参**：模型架构表（d_model/nhead/layers/d_ff/dropout）、优化器设置、lr schedule（如有）。
5. **实验结果**：表格列出 baseline 与每个消融/拓展的指标（BLEU/chrF/训练时间）。
6. **可视化与分析**：训练曲线、attention heatmaps、案例分析（成功/失败）。
7. **结论与反思**：总结主要发现、存在问题、未来改进方向。
8. **附录**：运行命令、环境依赖、关键代码片段（或给出文件/函数名定位）。

9. 评分方式

- **实现正确性（35）**：是否正确使用 `nn.Transformer`与mask 。
- **结果与可视化（25）**：验证 & 测试 BLEU 、训练验证曲线。
- **分析深度（20）**：消融/拓展实验设计合理并给出结果分析。
- **报告质量（10）**：表达清楚、结构完整、图表规范。
- **加分项（10）**：实现 beam search / Noam lr schedule / weight tying / 更大数据 fine-tune 等。