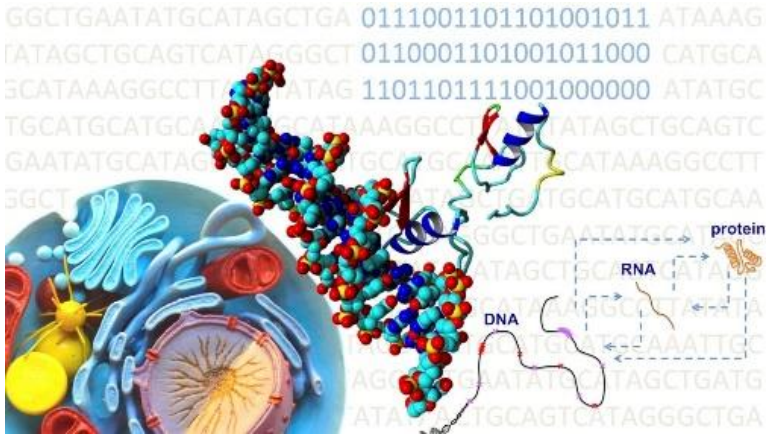
第四章 Transformer预训练语言模型

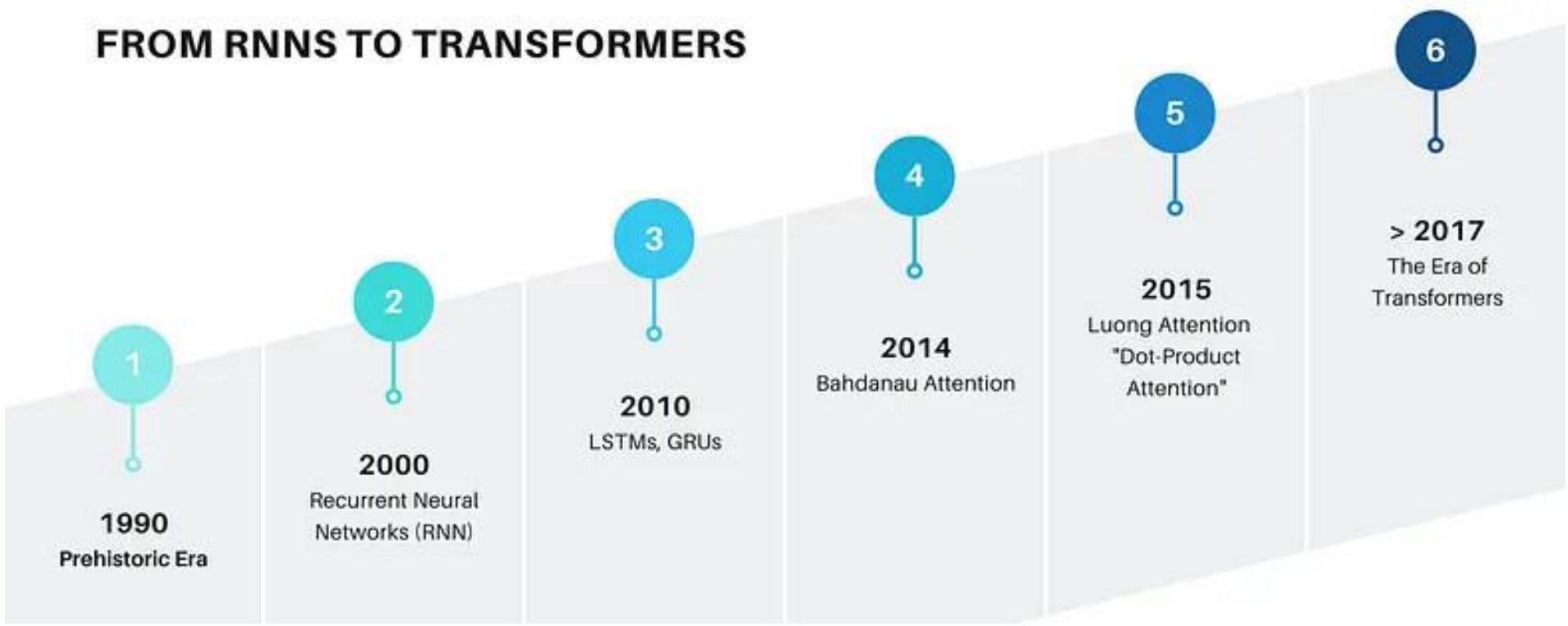
为什么需要预训练语言模型？



“自然语言：人类语言多样、复杂，句子结构长、上下文依赖强”

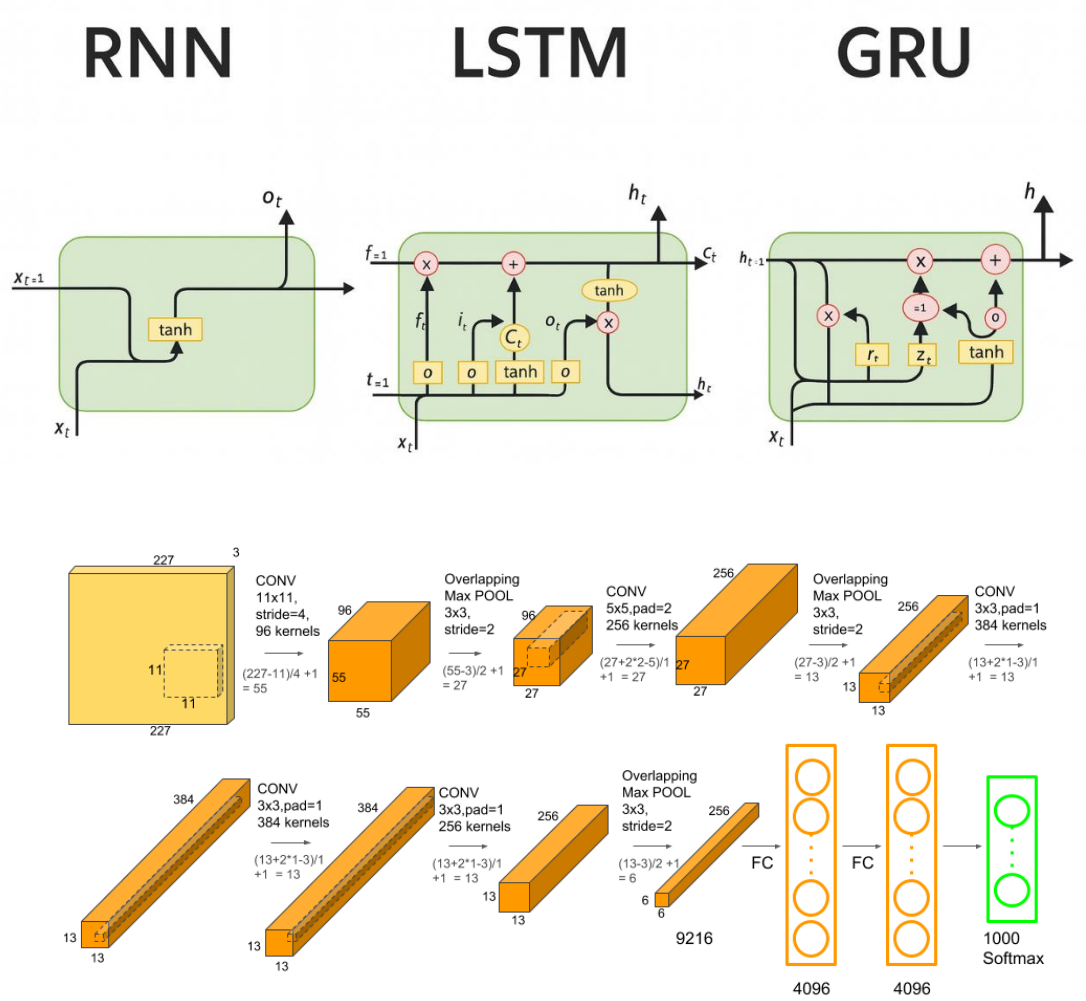
“生物序列：蛋白质/基因序列像语言，有语法（结构）、语义（功能）”

从RNN 到 Transformer



RNN (2000)：逐个读序列，容易忘前面内容。
LSTM/GRU (2010)：加了记忆，但训练慢。
Attention (2014–2015)：能“聚焦重点”，效果更好。
Transformer (2017-至今)：全面进入Transformer 时代。

为什么RNN/CNN 等不行？



RNN / LSTM / GRU

- 逐个读序列，速度慢
- 容易忘记前面的信息（长序列难）

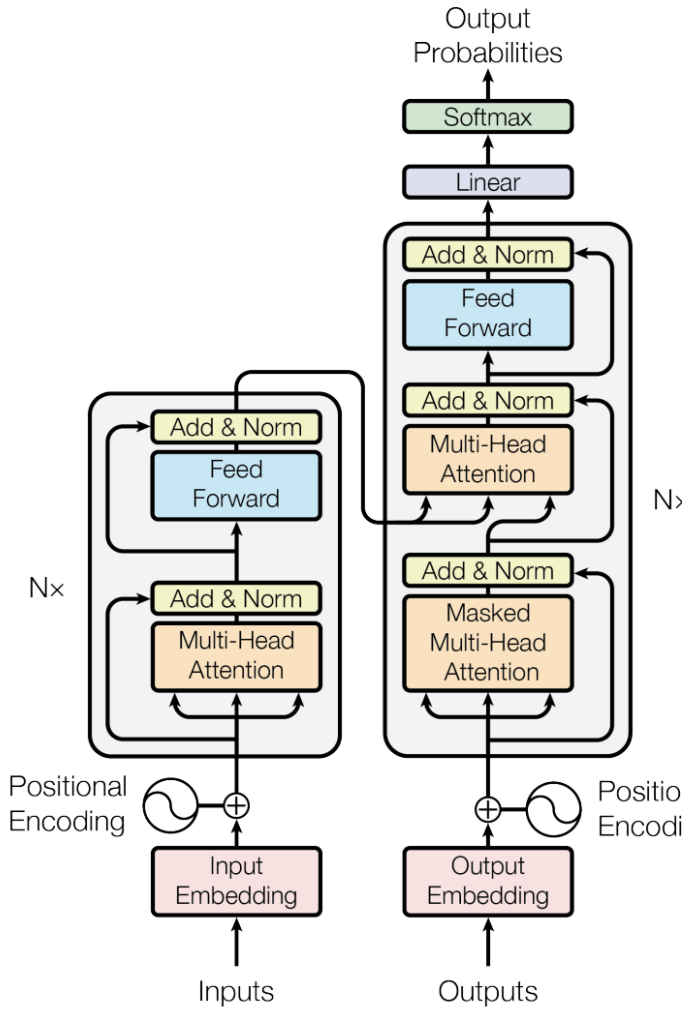
CNN

- 适合局部模式（如图像），但蛋白质功能常依赖全局关系
- 需要很多层才能捕捉长程依赖

如蛋白质序列很长

- RNN 容易 “记不住”
- CNN 只能看局部
- Transformer利用其Encoder 能**一次看到全局**

Attention Is All You Need



Attention 的核心思想

自注意力（Self-Attention）

每个位置都能和序列中其他位置交互
让模型“同时看到全局”

Transformer 的结构

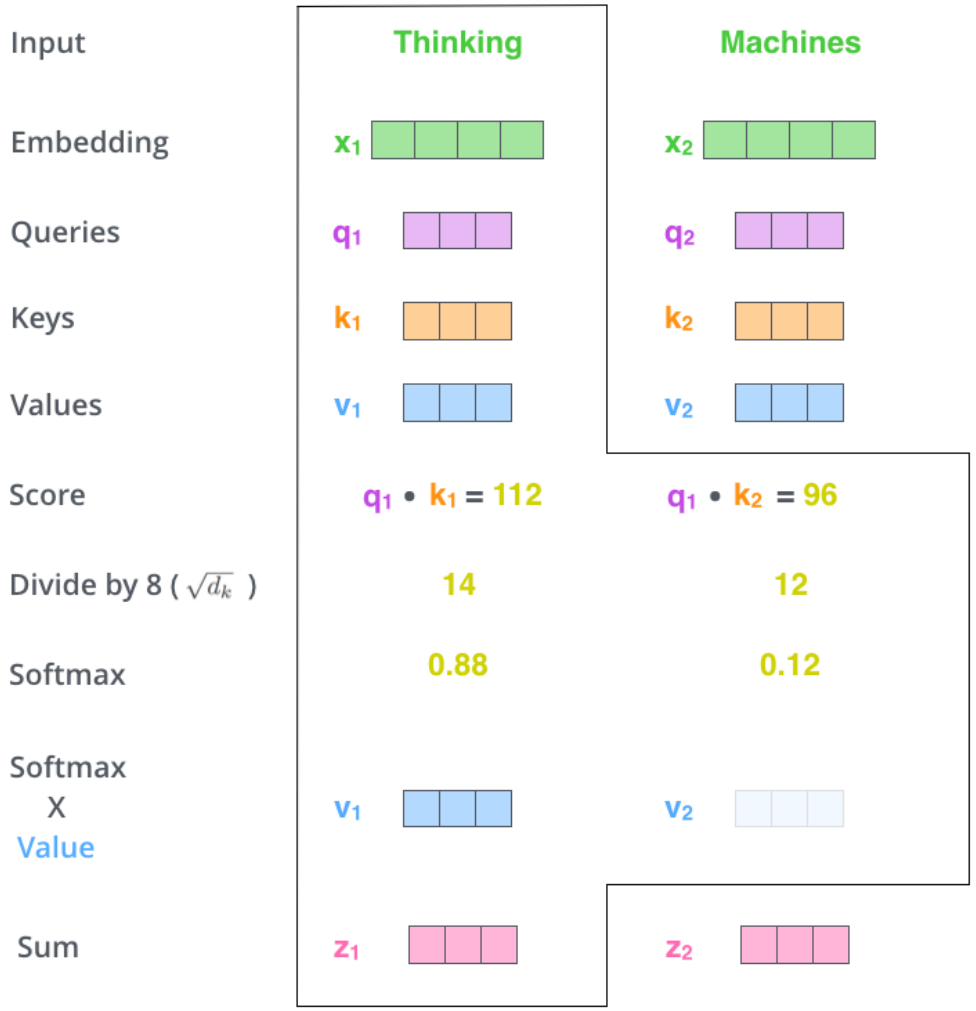
Encoder：把输入序列变成向量表示
Decoder：一步步生成输出序列（翻译/预测）

Multi-Head Attention：多角度同时关注不同部分

优势

并行计算（比 RNN 快很多）
捕捉长程依赖（远距离氨基酸关系）

Self-Attention 机制

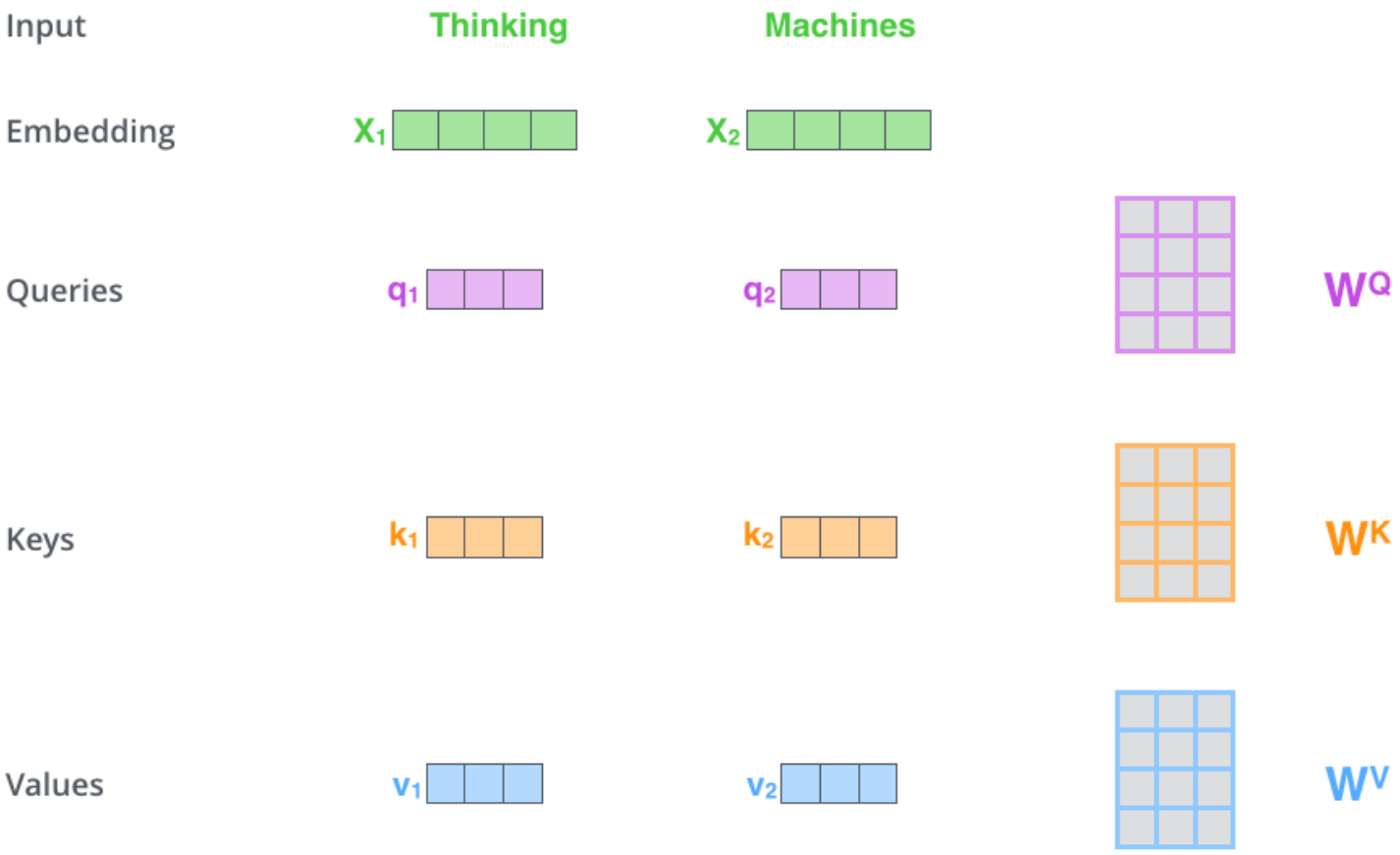


Self-Attention 核心步骤

1.每个输入（x） → 变成三个向量：Query (Q)、Key (K)、Value (V)
2.计算相关性：Q · K，表示“这个词和别的词有多相关”
3.归一化：除以维度，再做 Softmax → 得到权重
4.加权求和：权重 × V，得到新的表示 z
5.结果：每个位置的表示，融合了全序列的信息

大家可以把 Attention 想成开会。每个人（序列中的位置）都带着**自己的问题 Q**，去问**其他人的身份 K**，根据**匹配程度分配注意力**，再把**大家的答案 V** 汇总成一个新的表示。这样，每个位置不再只看自己，还能参考**全局**

Self-Attention 机制

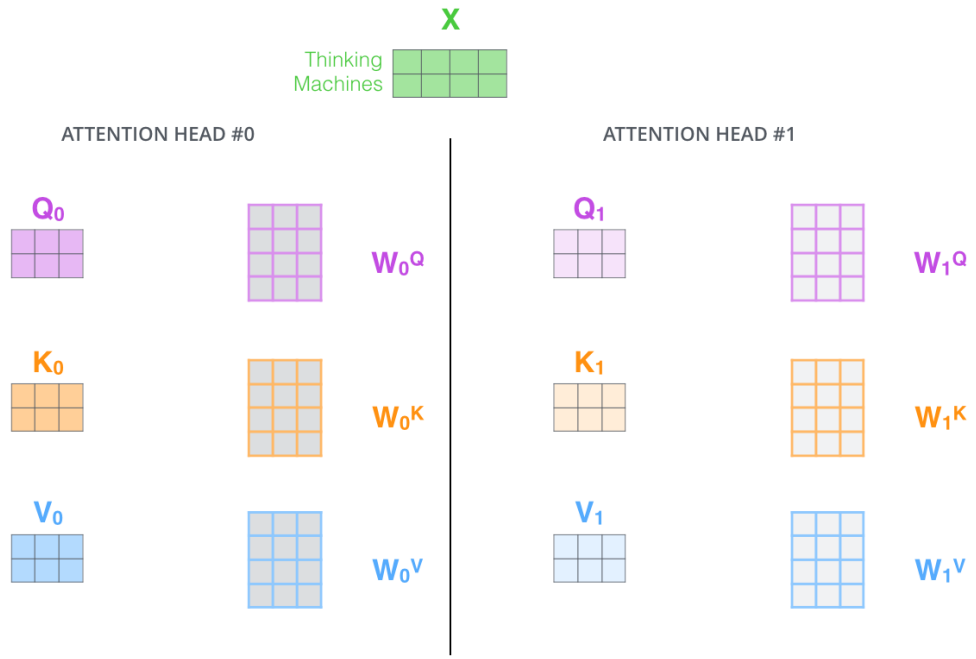


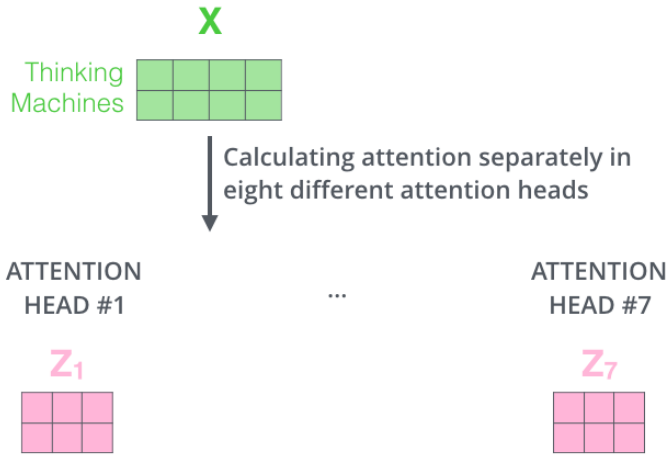
$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$$

每个词都会去问其他词 ‘你跟我相关吗？’（**Q•K**），算出分数后做 softmax 变成权重，再用这些权重去**加权**取别人的信息 **V**，最后得到一个新的表示 **Z**。

这就是 **Attention 的本质**：根据**相关性**计算聚合信息。让模型在一堆信息里，挑出最**重要的部分**去**重点关注**。

Multi-Head Attention

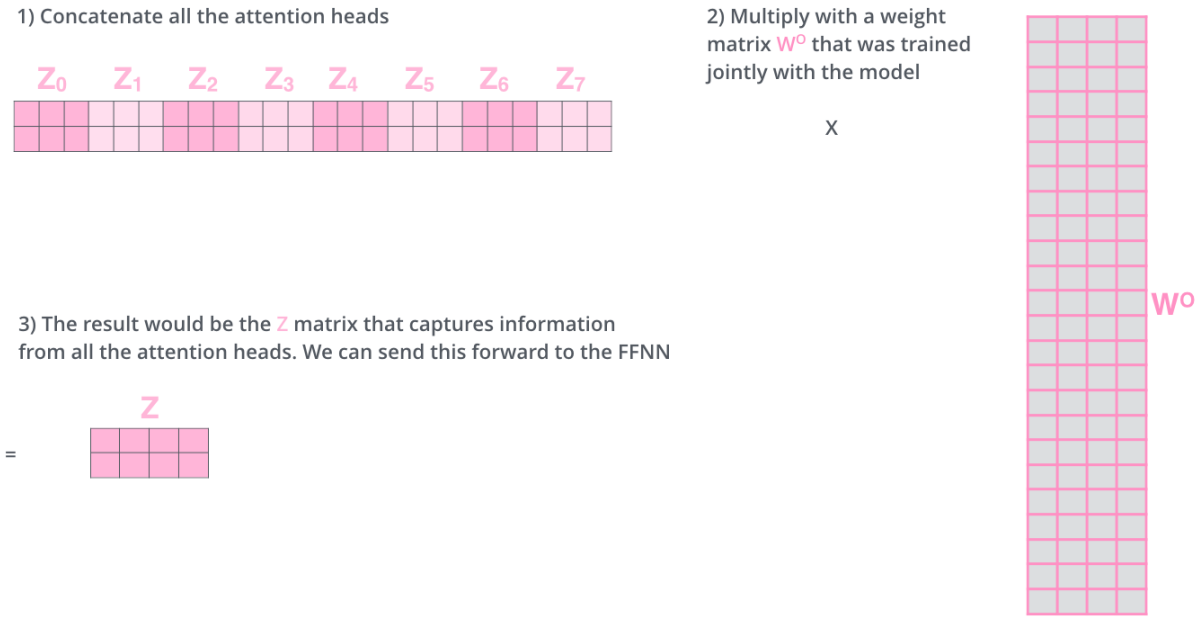


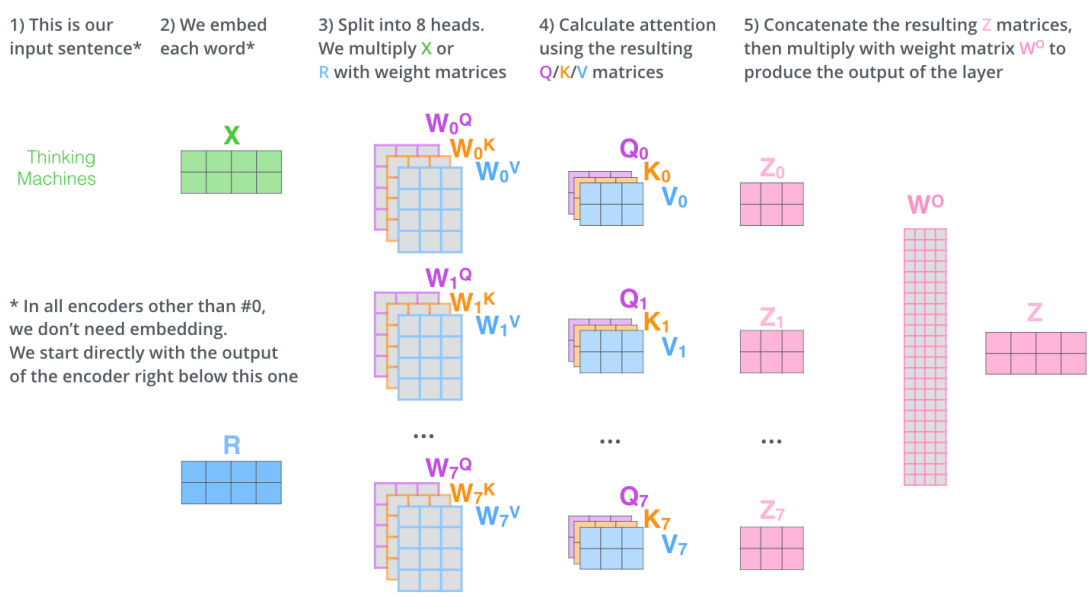


Multi-Head Attention 的作用

单个 Attention Head：只能从一个角度看序列关系
多个 Head：相当于多副眼镜，同时从不同角度去关注
输出：每个 head 得到自己的表示 Z_0、Z_1...，再拼接起来
→ 更全面的特征

Multi-Head Attention





Multi-Head Attention 输出过程

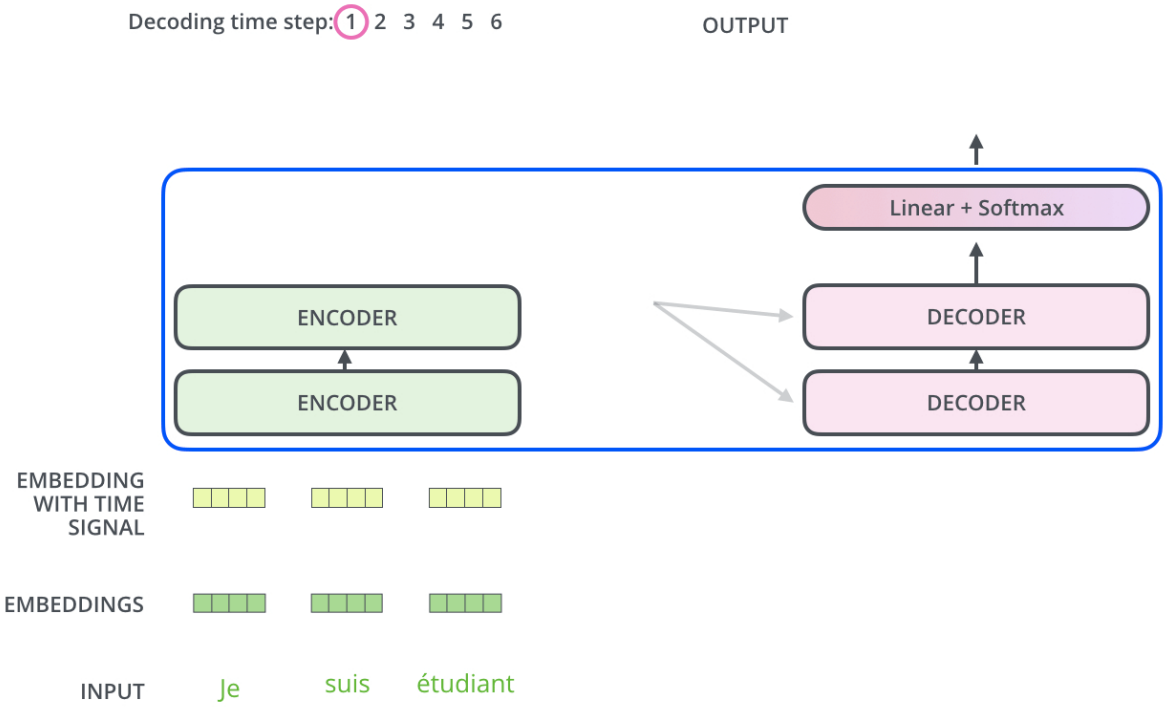
1.分头计算：每个 head 得到自己的表示 $Z_0, Z_1, \ldots, Z_7$

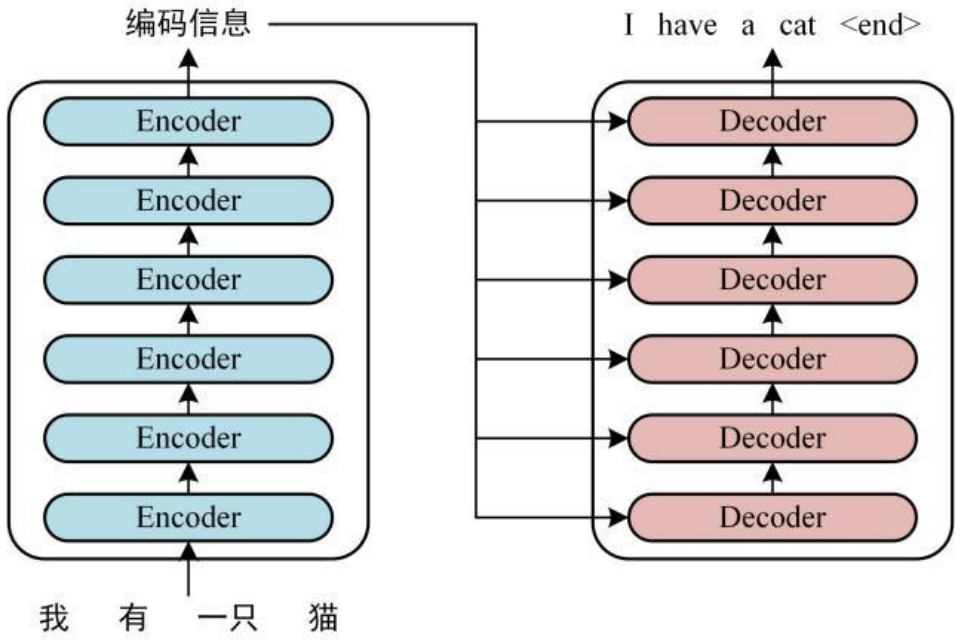
2.拼接：把这些 Z 拼在一起，得到更丰富的特征

3.线性变换：再乘一个矩阵 W^0，把拼接后的结果压缩成统一维度

4.最终输出 Z：融合了多角度信息，传给后续层

Encoder-Decoder 框架





Encoder-Decoder 框架

Encoder：把输入序列（法语/蛋白质）压缩成向量表示
Decoder：逐步生成输出序列（英语/预测标签）

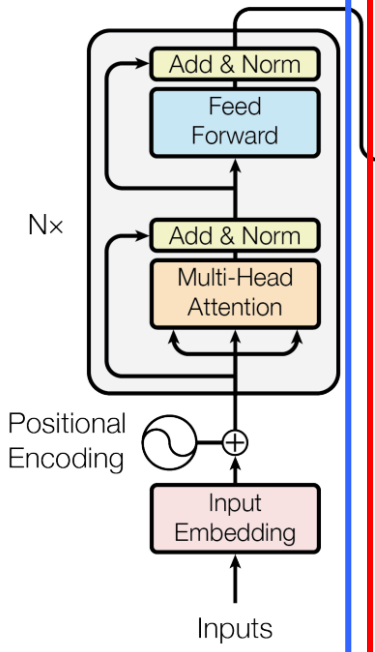
Attention Is All You Need

BERT

编码器

遮蔽词元预测

在训练过程中随机遮蔽一些词元，然后让模型去预测这些被遮蔽的词元

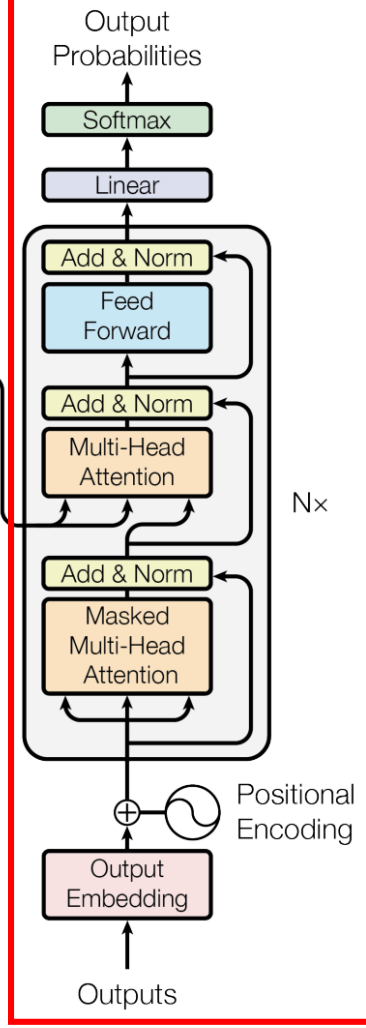


GPT

解码器

下一个词元预测

根据前面的词元来预测下一个词元



Nat Biotechnol, 2024, 42, 200-202

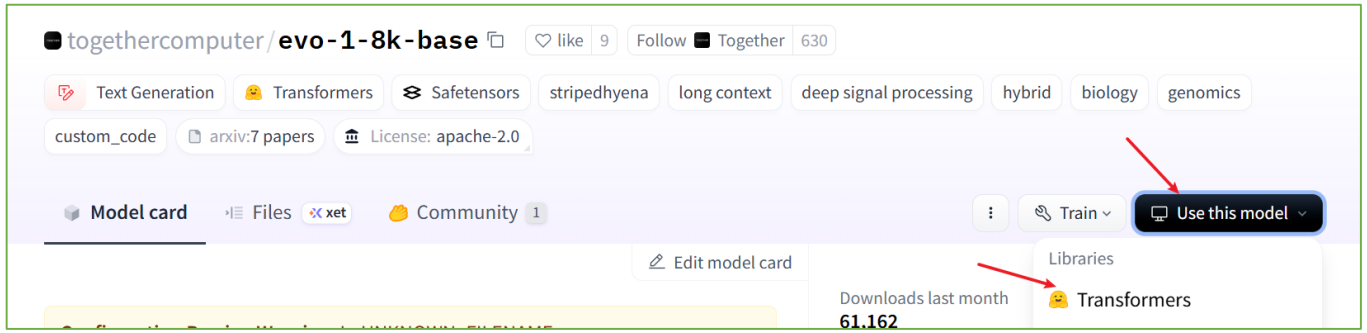
Transformers库介绍和使用示例

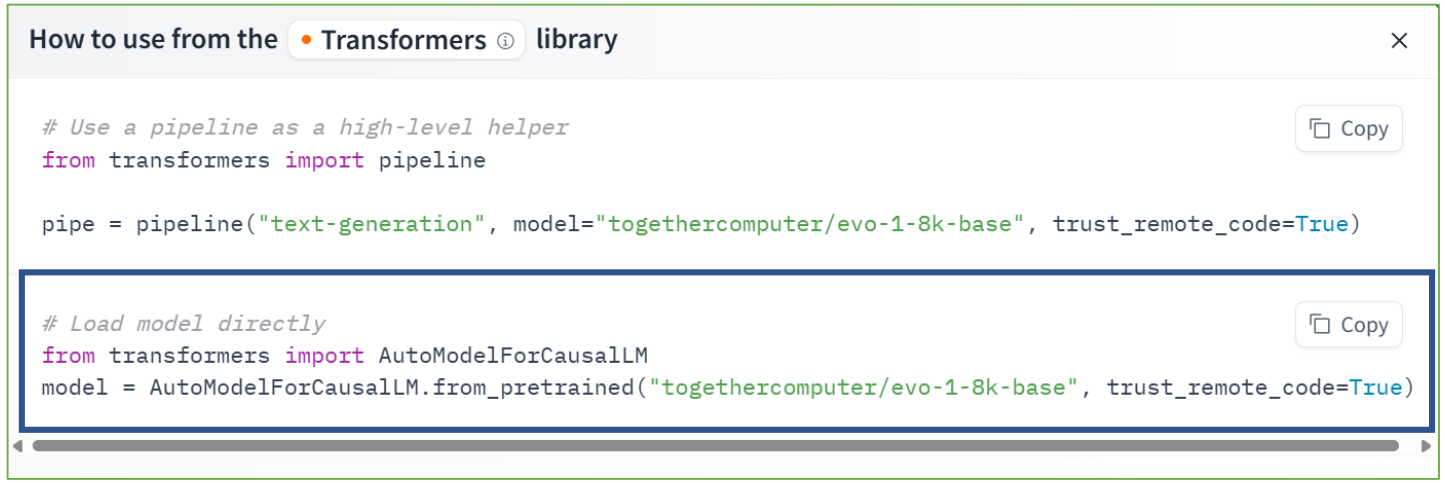
Transformers 库的基本介绍

- 来自 Hugging Face
- 可以直接地帮我们完成以下事情
  - 根据标准化的配置文件(标准化的意思是有特定的参数)，帮我们自动化地定义模型很分词器
  - 通过一行代码，帮我们自动下载别人的模型结构 和 权重，然后帮我们载入预训练好的模型(会存在科学上网问题)
  - 仅需少量代码(官方定义的接口)，完成模型的训练(预训练+微调)

- 集成度高，直接调用别人预训练好的模型非常简单
- 适合快速上手、实验验证

使用Hugging Face 载入别人的预训练模型





在国外的话，这两行代码就能够很方便地帮我们完成以下工作
自动下载模型权重和分词器到 ~/.cache/huggingface/hub/ 这一个默认路径

然后导入模型的时候，就会默认去这个路径看看有无下载使用该模型需要的文件，包括分词器，模型权重，模型配置

```
(base) [zyd fyp-X10DAi /home/zyd] WORK 0
# ls ~/.cache/huggingface/hub/
datasets--HuggingFace-CN-community--Diffusion-book-cn      models--Rostlab--prot_bert_bfd                 models--togethercomputer--evo-1-8k-base
datasets--HuggingFace-CN-community--Diffusion-book-cn.tar  models--Rostlab--prot_t5_xl_half_uniref50-enc  version.txt
evo-1-8k-base                                              models--togethercomputer--evo-1-131k-base
```

transformers自动pull下来的示例

main ▾ evo-1-8k-base

Zymrael Update README.md a9be7b6 VERIFIED

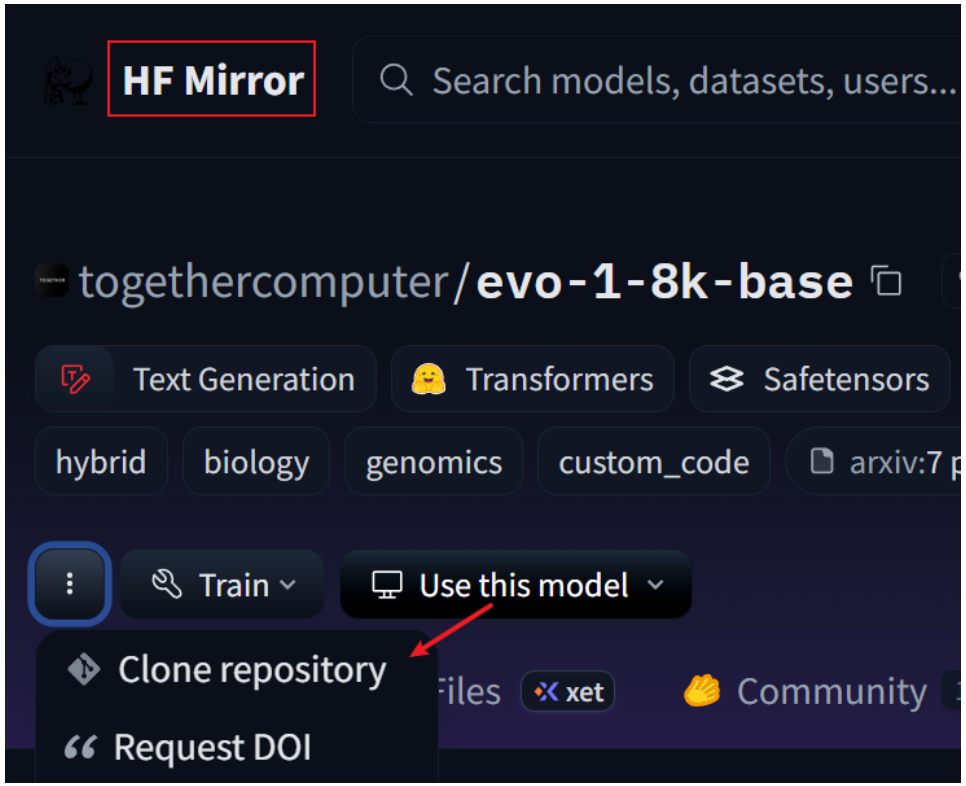
.gitattributes	Safe	1.52 kB
README.md	Safe	5.27 kB
config.json	Safe	1.86 kB
generation_config.json	Safe	69 Bytes
model-00001-of-00003.safetensors	Safe	4.98 GB xet
model-00002-of-00003.safetensors	Safe	4.93 GB xet
model-00003-of-00003.safetensors	Safe	3 GB xet
model.safetensors.index.json	Safe	34.9 kB
pytorch_model.pt	Safe pickle	16.8 GB xet
special_tokens_map.json		3 Bytes
tokenizer_config.json	Safe	299 Bytes

```
(base) [zyd fyp-X10DAi /home/zyd/.cache/huggingface/hub] WORK 0
# tree -alh ./models--togethercomputer--evo-1-8k-base/
./models--togethercomputer--evo-1-8k-base/
├── [4.0K]  blobs
│   ├── [1.1G]  4a08792f22697584c4b0c6cd1729902bc993ad7396b76f5caf6d7cc2b32ab882.incomplete
│   ├── [ 34K]  b08da632da9a41606e63792f414afcd7153488cf
│   └── [1.8K]  f5c2e2cb7bedd60b2d77672093bab04906281c0a
├── [4.0K]  .no_exist
│   └── [4.0K]  a9be7b66485080893399ade87c7d34f81ad3e249
│       └── [   0]  model.safetensors
├── [4.0K]  refs
│   └── [  40]  main
└── [4.0K]  snapshots
    └── [4.0K]  a9be7b66485080893399ade87c7d34f81ad3e249
        ├── [  34]  config.json -> ../../../evo-1-8k-base/config.json
        ├── [  45]  generation_config.json -> ../../../evo-1-8k-base/generation_config.json
        ├── [  55]  model-00001-of-00003.safetensors -> ../../../evo-1-8k-base/model-00001-of-00003.safetensors
        ├── [  55]  model-00002-of-00003.safetensors -> ../../../evo-1-8k-base/model-00002-of-00003.safetensors
        ├── [  55]  model-00003-of-00003.safetensors -> ../../../evo-1-8k-base/model-00003-of-00003.safetensors
        ├── [  51]  model.safetensors.index.json -> ../../../evo-1-8k-base/model.safetensors.index.json
        ├── [  39]  pytorch_model.pt -> ../../../evo-1-8k-base/pytorch_model.pt
        ├── [  32]  README.md -> ../../../evo-1-8k-base/README.md
        ├── [  46]  special_tokens_map.json -> ../../../evo-1-8k-base/special_tokens_map.json
        └── [  44]  tokenizer_config.json -> ../../../evo-1-8k-base/tokenizer_config.json
```

如果不解决科学上网问题，那么就会因为下载不下来直接报错

transformers的科学上网问题简单解决方案

使用 hf-mirror.com 镜像，git clone 到本地，然后通过本地绝对路径导入




```
git lfs install
git clone https://hf-mirror.com/togethercomputer/evo-1-8k-base
```

假如 我 git clone 到 /data12T/evo-1-8k-base目录

```
from transformers import AutoModelForCausalLM
model = AutoModelForCausalLM.from_pretrained("/data12T/evo-1-8k-base", trust_remote_code=True)
```

```
./evo-1-8k-base/
├── [1.8K]  config.json
├── [  69]  generation_config.json
├── [1.5K]  .gitattributes
├── [4.6G]  model-00001-of-00003.safetensors
├── [4.6G]  model-00002-of-00003.safetensors
├── [2.8G]  model-00003-of-00003.safetensors
├── [ 34K]  model.safetensors.index.json
├── [ 16G]  pytorch_model.pt
├── [5.1K]  README.md
├── [   3]  special_tokens_map.json
└── [ 299]  tokenizer_config.json
```

预训练语言模型ESM进行蛋白质定位分类

数据讲解

ACC	Kingdom	Mitochondrion	Sequence
Q28165	Metazoa	0	MAAAAAAAAAAGAAGGRGSGPGRRRHLVPGAG(
Q86U42	Metazoa	0	MAAAAAAAAAAGAAGGRGSGPGRRRHLVPGAG(
Q0GA42	Metazoa	0	MAAAAAAAAALGVRLRDCCSRGAVLLLFFSLSPRP
P82349	Metazoa	0	MAAAAAAAAATEQQGSNGPVKKSMREKAVERRN
Q7L5N1	Metazoa	0	MAAAAAAAAATNGTGGSSGMEVDAAVVPSVMA
Q96S94	Metazoa	0	MAAAAAAAGAAGSAAPAAAAGAPGSGGAPSGS(
Q9CQ25	Metazoa	0	MAAAAAAAGGAALAVSTGLETATLQKLALRRKKVI
Q96P70	Metazoa	0	MAAAAAAGAASGLPGPVAQGLKEALVDTLTGILSP
P63086	Metazoa	0	MAAAAAAGPEMVRGQVFDVGPRYTNLSYIGEGAY
Q9UID3	Metazoa	0	MAAAAAAGPSPGSGPGDSPEGPEGEAPERRRKAH
Q9SE42	Viridiplanta	0	MAAAAAAKIAPSMLSSDFANLAAEADRMVRLGAD
Q01167	Metazoa	0	MAAAAAALSGAGTPPAGGGAGGGGAGGGGSPPG
P21708	Metazoa	0	MAAAAAAPGGGGGEPRGTAGVVPVVPGEVEVVKC
Q8CH72	Metazoa	0	MAAAAAASHLNLDALREVLECPICMESFTEEQLRPH
P51788	Metazoa	0	MAAAAAEEGMEPRALQYEQTLMYGRYTQDLGAFA
Q8BKC5	Metazoa	0	MAAAAAEQQQFYLLLGNLLSPDNVVRKQAEETYEN
Q9NXW9	Metazoa	0	MAAAAAETPEVLRECGCKGIRTCLICERQRGSDPPV
P86790	Metazoa	0	MAAAAAGAGSGPWAAQEKQFPPALLSFFIYNPRFC
P23610	Metazoa	0	MAAAAAGLGGGGAGPGPEAGDFLARYRLVSNKLK
Q9Y2Z0	Metazoa	0	MAAAAAGTATSQRFFQSFSDALIDEDPQAALEELT
Q3UCQ1	Metazoa	0	MAAAAALSGAGAPPAGGGAGGGGSPPGGWAVA
Q7L5D6	Metazoa	0	MAAAAAMAEQESARNGGRNRGGVQRVEGKLRAS
O75439	Metazoa	1	MAAAAARVVLSSAARRRLWGFSESLLIRGAAGRSL
Q6RFH5	Metazoa	0	MAAAAARWNHVWVGTETGILKGVNLQRKQAANI
Q9Y2U8	Metazoa	0	MAAAAASAPQQLSDEELFSQLRRYGLSPGPVTESTI

ACC

就像学生的学号，是每个蛋白的唯一 ID。

我们不直接用它做预测，只是用来区分样本。

Mitochondrion

这是我们要预测的标签（label）。

1 = 这个蛋白在“线粒体”里

0 = 不在

所以这是一个标准的**二分类任务**。

Sequence

蛋白的氨基酸序列，相当于一段由字母组成的“句子”。

我们要用 ESM 这种语言模型来读懂它。

加载与清洗

```python
import pandas as pd
df = pd.read_csv("deeploc.csv")
df = df[["ACC","Sequence","Mitochondrion"]].dropna()
df = df[df["Sequence"].str.len() > 0]
print(df.shape, df["Mitochondrion"].value_counts())
```

划分数据

```
from sklearn.model_selection import train_test_split
train, test = train_test_split( *arrays: df, test_size=0.2, stratify=df["Mitochondrion"], random_state=42)
train, val  = train_test_split( *arrays: train, test_size=0.1, stratify=train["Mitochondrion"], random_state=42)
```

划分数据

为什么要划分？
我们需要让模型“学”一部分数据（训练集），
“考”一部分数据（验证集），
最后“真正考试”的部分（测试集）。

常见比例

训练集：大约 70%
验证集：大约 10%
测试集：大约 20%

注意

用 stratify 保证正样本 / 负样本比例一致。
否则可能会出现训练集几乎没有 1 的情况，模型学不动。

把序列变成“模型能看懂的词”（Tokenizer）

```python
from transformers import AutoTokenizer

# HuggingFace 模型仓库 ID
MODEL_ID = "facebook/esm2_t12_35M_UR50D"  # 小模型，国内环境推荐先试

# 如果直接下载很慢，可以换成镜像：
# 清华镜像：  https://hf-mirror.com/
# export HF_ENDPOINT=https://hf-mirror.com   # Linux / Mac
# set HF_ENDPOINT=https://hf-mirror.com      # Windows PowerShell

# 加载分词器
tok = AutoTokenizer.from_pretrained(MODEL_ID, do_lower_case=False)

# 随便取一个蛋白序列试试
seq = "IEDPELEAIKARVREMEEEAEKLKELQNEVEKQMNMSPPPGNAGPVIMSIEEKMEADARSIYVGNVDYGATAEELEAHFHGCGSVNRV"
out = tok(seq)
print(out.keys())      # ['input_ids', 'attention_mask']
print(out["input_ids"][:10])  # 看看前10个token id
```

分词器（Tokenizer）

把蛋白质序列（字母串）切成模型认识的“词”

每个氨基酸/氨基酸片段 → 一个数字 ID

输出结果

input_ids：序列被编码后的数字

attention_mask：哪些位置是真实数据，哪些是填充

科学上网提醒

HuggingFace 在国内可能下载慢

可以设置环境变量：

HF_ENDPOINT=https://hf-mirror.com

或者提前把模型下好，放到本地再加载

数据集包装

```python
import torch
from torch.utils.data import Dataset

# 定义一个简单的数据集类
class SeqDataset(Dataset):
    def __init__(self, df):
        self.x = df["Sequence"].tolist()          # 输入：序列
        self.y = df["Mitochondrion"].tolist()     # 输出：标签 (0/1)

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

    def __getitem__(self, i):
        return {"text": self.x[i], "labels": int(self.y[i])}

# collate_fn：把一批数据打包成模型输入
def collate(batch):
    seqs = [b["text"] for b in batch]
    y    = [b["labels"] for b in batch]
    toks = tok(
        seqs,
        padding=True,
        truncation=True,
        max_length=1024,      # ESM 最大长度
        return_tensors="pt"
    )
    toks["labels"] = torch.tensor(y)
    return toks
```

为什么需要 Dataset？

我们的原始数据是表格格式（DataFrame）
PyTorch 训练时需要一个能 __getitem__ 的类，
按 index 取数据

SeqDataset

self.x → 蛋白质序列
self.y → 是否在线粒体（0/1）

collate

负责把一批序列打包
通过 tokenizer 转成 input_ids / attention_mask

加上 labels 一起喂给模型

模型 = ESM 表征 + 线性分类头

```python
import torch, torch.nn as nn
from transformers import AutoModel

class ESMBinary(nn.Module):
    def __init__(self, model_id, freeze=True):
        super().__init__()
        self.esm = AutoModel.from_pretrained(model_id)
        if freeze:  # 入门建议先冻结，稳定、省显存
            for p in self.esm.parameters():
                p.requires_grad = False
        h = self.esm.config.hidden_size
        self.head = nn.Linear(h, out_features: 2)  # 二分类：不在线粒体/在线粒体

    def forward(self, input_ids, attention_mask, labels=None):
        out = self.esm(input_ids=input_ids, attention_mask=attention_mask)
        cls = out.last_hidden_state[:, 0, :]          # 取“句向量”
        logits = self.head(cls)
        loss = None
        if labels is not None:
            loss = nn.CrossEntropyLoss()(logits, labels)
        return {"loss": loss, "logits": logits}
```

思路：ESM 把序列→向量；我们只在顶层接一个线性分类头。

为什么冻结：减少训练量，避免过拟合，演示更稳。

输出：两个 logit（0/1），训练时走 CrossEntropyLoss

三步开训（Trainer）

```python
from transformers import Trainer, TrainingArguments
from torch.utils.data import DataLoader
from sklearn.model_selection import train_test_split
import numpy as np

# 构建 Dataset
train_ds, val_ds, test_ds = SeqDataset(train), SeqDataset(val), SeqDataset(test)

# 评测指标（尽量简单直观）
1 usage
def metrics(eval_pred):
    logits, labels = eval_pred
    preds = logits.argmax(-1)
    acc = (preds == labels).mean()
    return {"accuracy": acc}
```

```python
# 训练参数（入门稳妥配置）
args = TrainingArguments(
    output_dir="./out",
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=32,
    learning_rate=1e-3,                 # 只训线性头 → 稍大点 OK
    evaluation_strategy="epoch",
    logging_steps=50,
    save_strategy="epoch",
    report_to="none"                    # 课堂演示不用 W&B
)

model = ESMBinary(MODEL_ID, freeze=True)
trainer = Trainer(
    model=model, args=args,
    train_dataset=train_ds, eval_dataset=val_ds,
    data_collator=collate, tokenizer=tok,
    compute_metrics=metrics
)

trainer.train()
```

Trainer 自动帮我们做训练循环、评估与保存。

关键参数：batch_size / lr / epochs / evaluation_strategy。

课堂建议：先跑 2–3 个 epoch，看验证集准确率即可。

pip 安装**慢**可用清华：pip install -i https://pypi.tuna.tsinghua.edu.cn/simple transformers

HF 下载**慢**：设置 HF_ENDPOINT=https://hf-mirror.com；或提前把模型放到本地目录再 from_pretrained("本地路径")。

测试集评估（Accuracy / F1 / AUC）

```python
from sklearn.metrics import classification_report, roc_auc_score

pred = trainer.predict(test_ds)
y_true = pred.label_ids
logits = pred.predictions
y_prob = logits[:, 1]                    # 类1（在线粒体）的 logit
y_pred = logits.argmax(1)

print(classification_report(y_true, y_pred, digits=3))
try:
    # 若两类都有样本，计算 AUC
    auc = roc_auc_score(y_true, y_prob)
    print("ROC-AUC:", round(auc, 3))
except Exception as e:
    print("AUC 无法计算（可能是测试集单类）")
```

Accuracy：整体对不对；

F1：更关注少数类的效果；

AUC：概率排序能力；

课堂提醒：若类别极不平衡，可报 F1 和 AUC 更客观。

看几个真实样本（增强直觉）

```python
import torch

sub = test.sample(5, random_state=0)

tokd = tok(sub["Sequence"].tolist(), padding=True, truncation=True, return_tensors="pt")
model.eval()
with torch.no_grad():
    out = model(**{k: v for k, v in tokd.items()}, labels=None)
    prob = out["logits"].softmax(-1)[:, 1].cpu().numpy()

for acc, y, p in zip(sub["ACC"], sub["Mitochondrion"], prob):
    print(f"{acc}: 真实={y}，预测P(mito)={p:.2f}")
```

随机抽 5 条：展示 真实标签 vs 预测概率；

直观看到：0.87 的概率更像“在线粒体”，0.12 更像“不在”。

Nat Biotechnol, 2024, 42, 200-202

扩展

随堂小测 & 课外学习

- 作业：
  - 1. 思考题：为什么我们在蛋白质亚细胞定位预测任务中，不直接用 RNN/CNN，而选择 Transformer？
  - 2. 讨论题：在蛋白质语言模型设计中，如果样本量不够大，你会采取哪些策略来解决？请提出至少两种方法。

- 参考资料：
  - 1. Attention Is All You Need，Vaswani 等，2017，NeurIPS.
  - 2. Evolutionary-scale prediction of atomic-level protein structure from primary sequence using a large language model ，Lin 等，2023，Science