

# 基于LLM的垃圾短信分类微调实战

第1-3部分：理论基础与数据流水线构建

LLM-Based Spam Classification Finetuning

# LLM生命周期的关键阶段：微调



## Stage 1 & 2: 基础构建

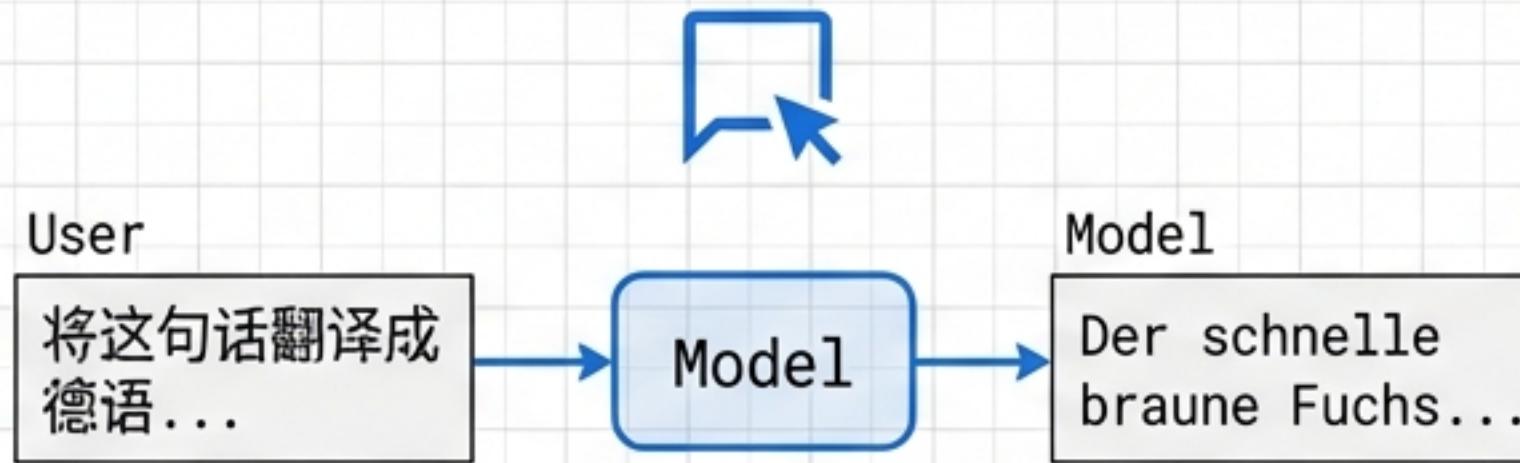
在通用文本数据集上进行预训练，构建基础模型。此时模型具备广泛的语言理解能力，但缺乏特定领域的专业性。

## Stage 3: 任务适配

本章重点。在较小的、带标签的数据集上进一步训练，使模型适应特定任务（如分类），将通用能力转化为专用能力。

# 两种核心微调范式：指令微调 vs 分类微调

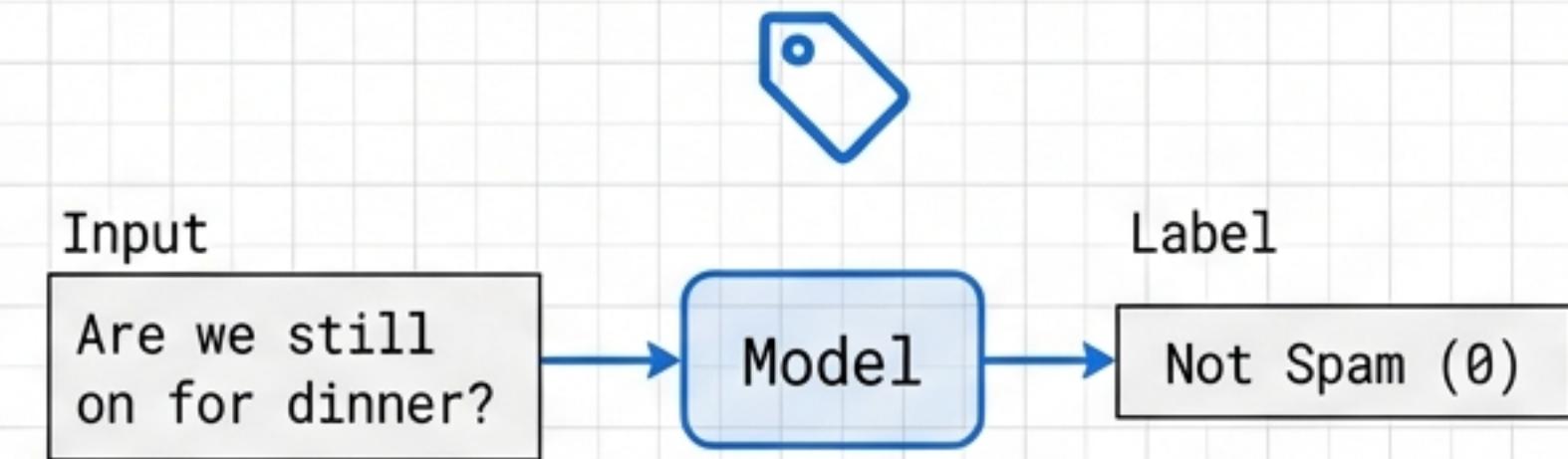
## 指令微调 (Instruction Finetuning)



目标：增强对自然语言指令的理解。输出为开放式文本生成。

适用场景：聊天机器人、问答系统、翻译。

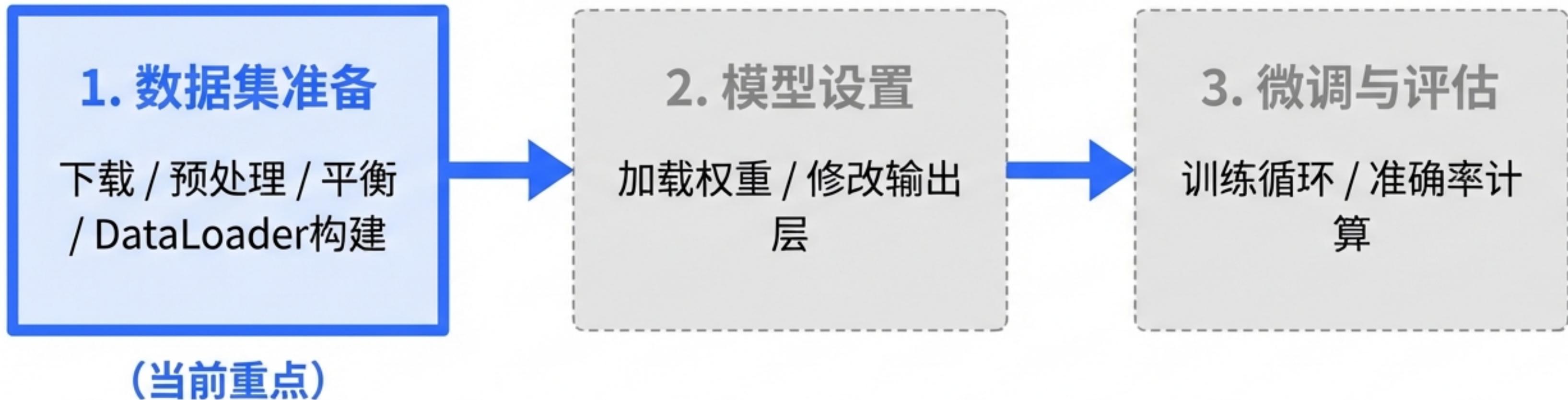
## 分类微调 (Classification Finetuning)



目标：识别特定的类别标签。输出为严格的类别ID。

适用场景：垃圾邮件检测、情感分析、医疗诊断。

# 本章技术路线图



最终目标：将一个通用的GPT架构模型转化为高效的二分类垃圾短信过滤器。

# 数据集概览：UCI SMS Spam Collection

**来源：**UCI Machine Learning Repository

**总量：**5,572 条短信

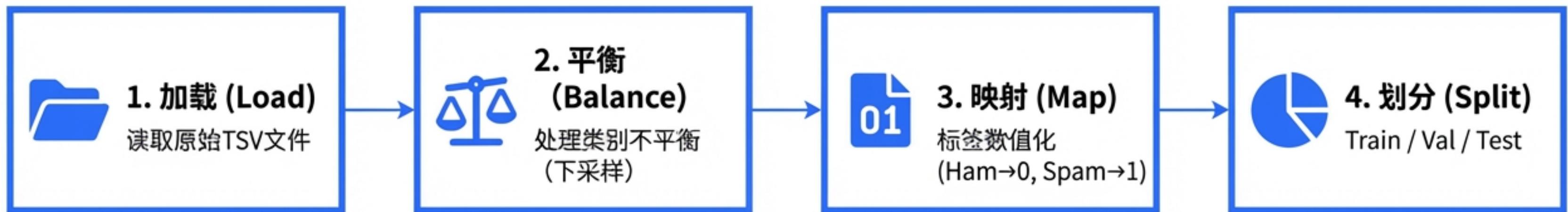
**格式：**TSV (制表符分隔)

**类别：**Ham (正常) / Spam (垃圾)

	Label	Text
1	ham	Go until jurong point, crazy.. Available only ...
2	ham	Ok lar... Joking wif u oni...
3	spam	Free entry in 2 a wkly comp to win FA Cup fina...
4	ham	U dun say so early hor... U c already then say...

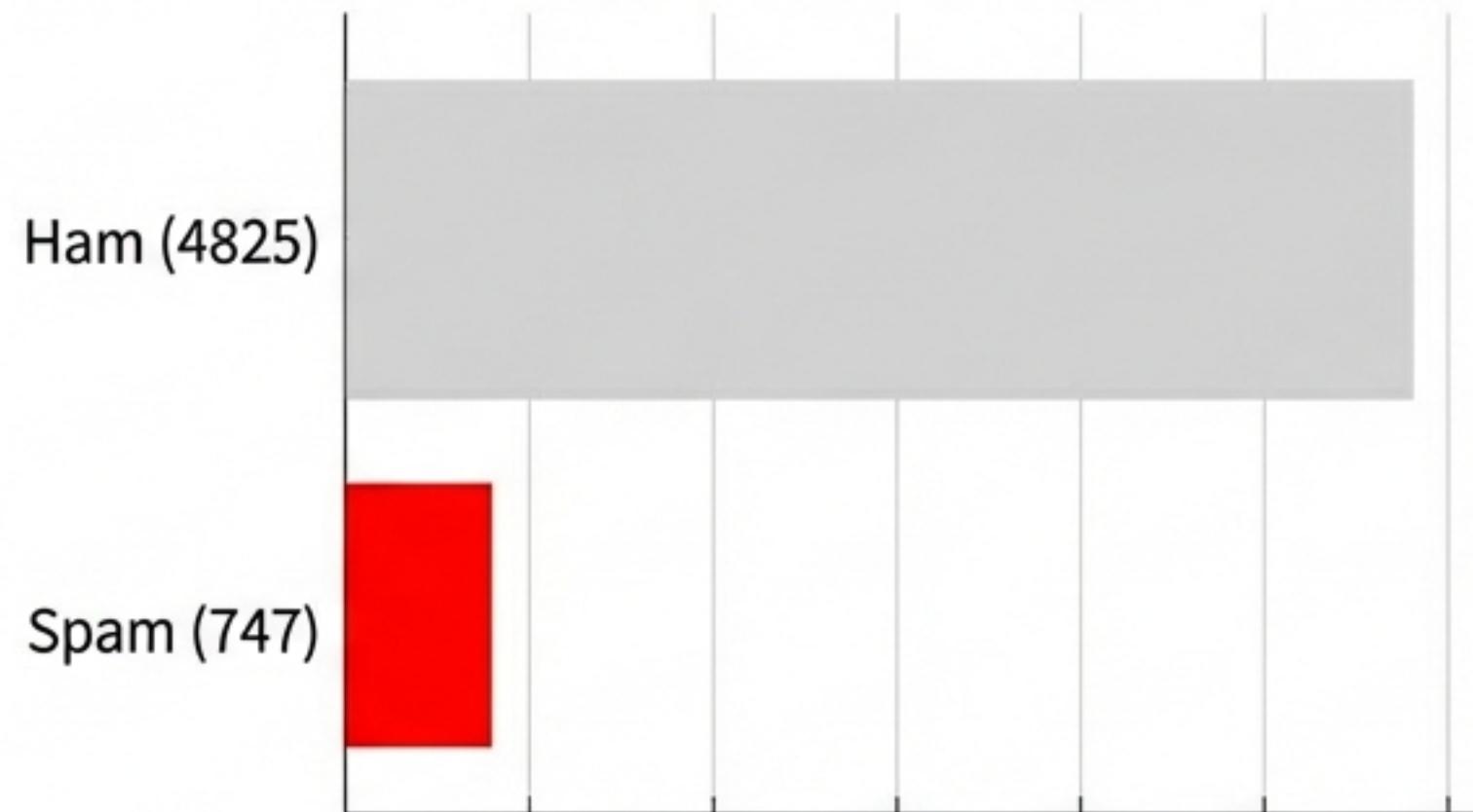
**Observation:** 真实数据是非结构化的，长度不一，需要清洗。

# 数据预处理流水线

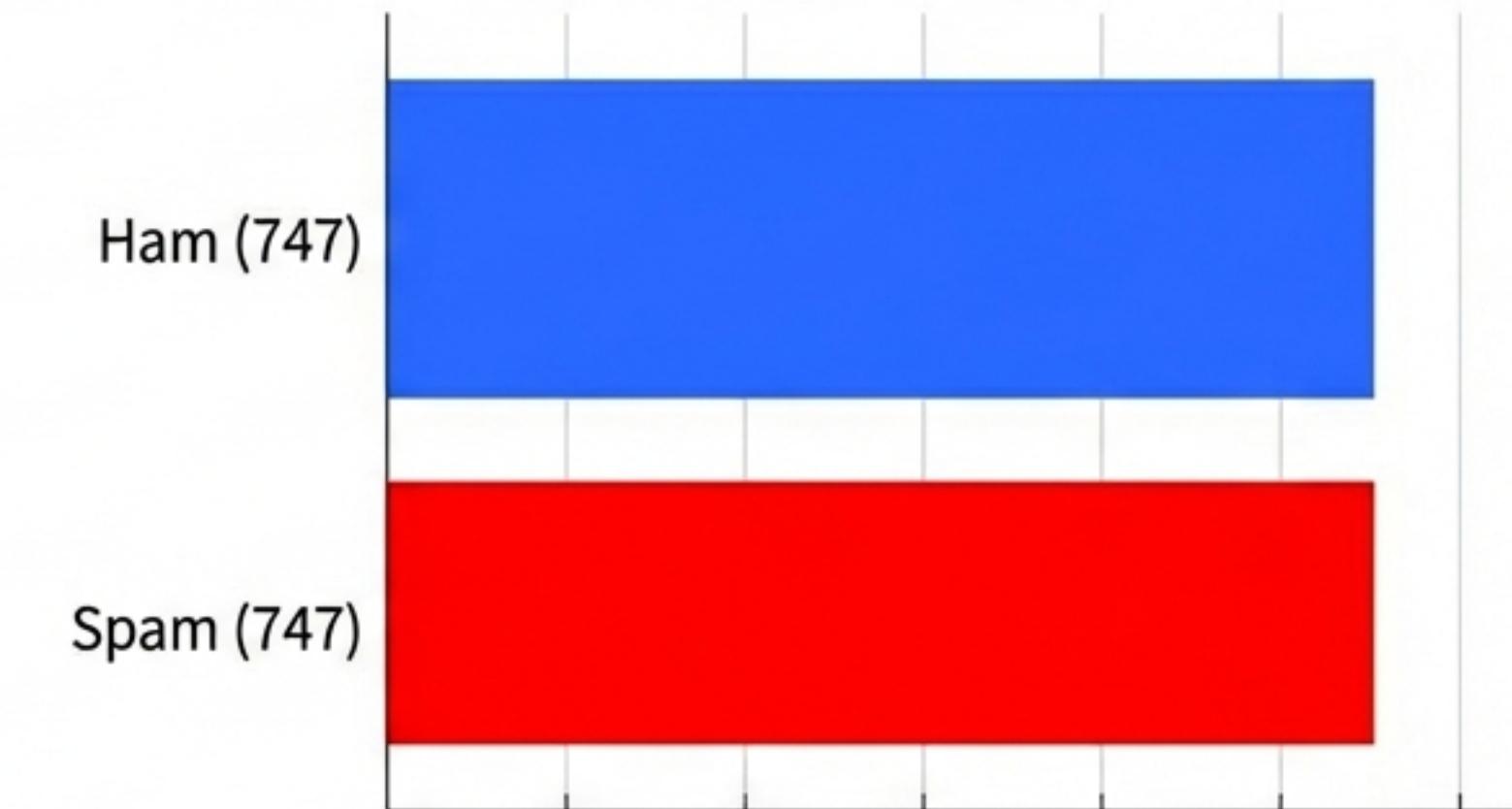


# 类别平衡处理：下采样策略

原始分布严重不平衡



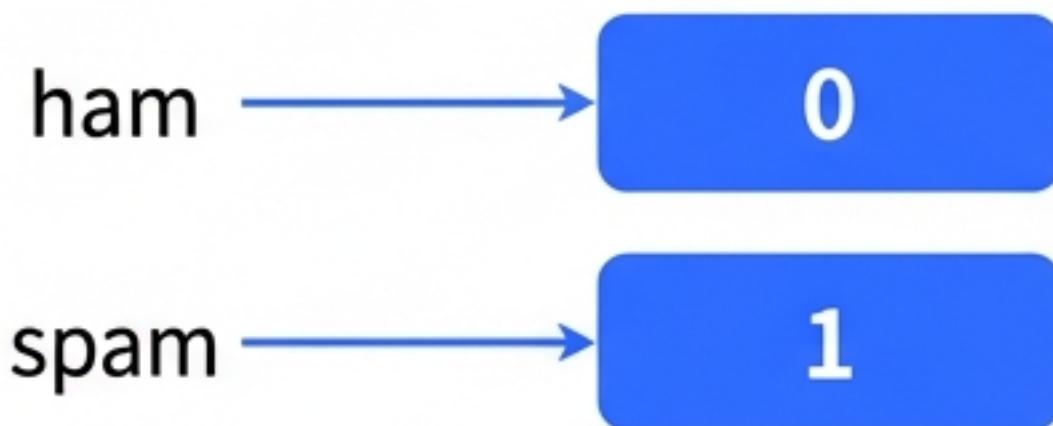
下采样后的平衡分布



```
# 核心逻辑: 创建平衡数据集
def create_balanced_dataset(df):
    num_spam = df[df["Label"] == "spam"].shape[0]
    # 随机抽取 ham 样本以匹配 spam 数量
    ham_subset = df[df["Label"] == "ham"].sample(num_spam, random_state=123)
    balanced_df = pd.concat([ham_subset, df[df["Label"] == "spam"]])
    return balanced_df
```

# 标签数值化与数据集划分

## Step 1: Label Mapping (标签映射)



```
df["Label"] = df["Label"].map({"ham": 0, "spam": 1})
```

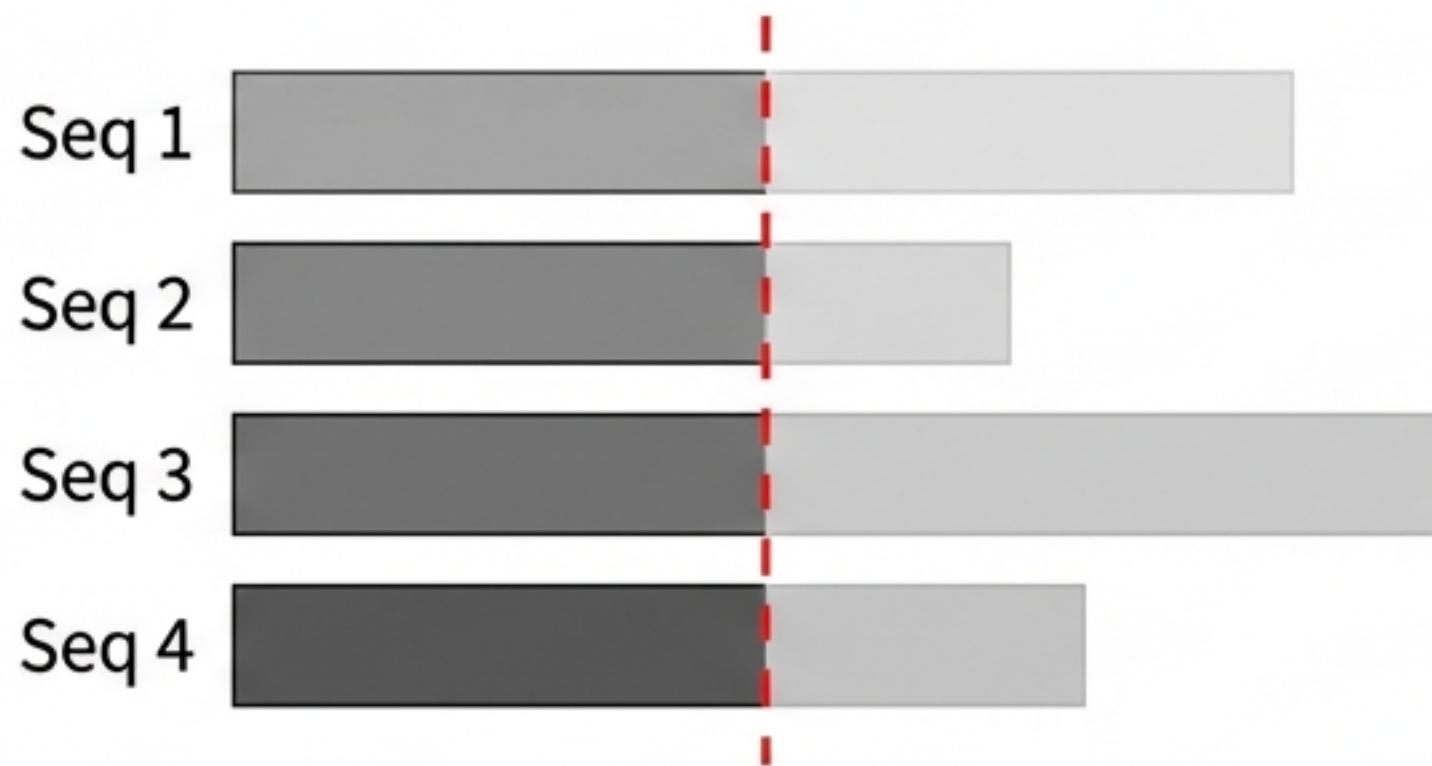
## Step 2: Dataset Splitting (数据集划分)



```
random_split(df, 0.7, 0.1)
```

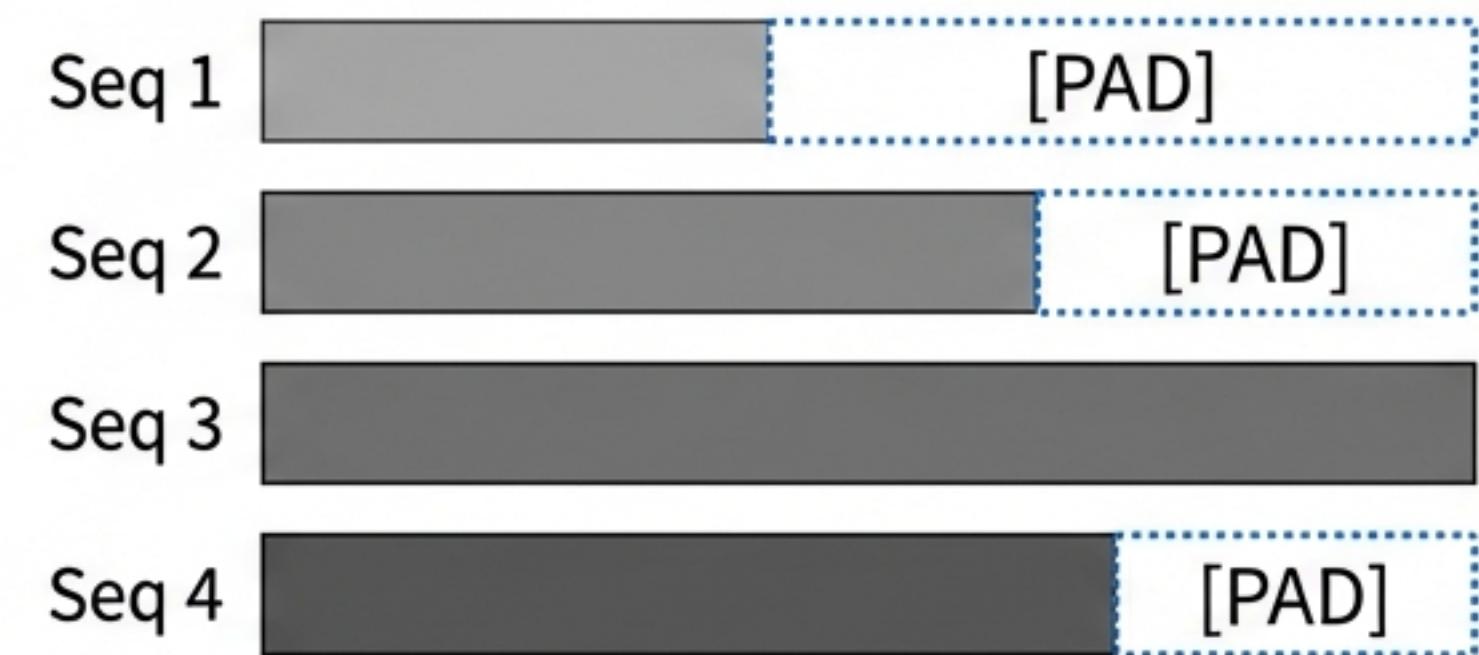
# 序列长度统一：截断 vs 填充

## 截断 (Truncation)



将所有序列切断至最短长度。  
缺点：丢失大量信息。

## 填充 (Padding) [Selected]



使用特殊token填充至最长长度。  
优点：保留完整信息。

**Decision:** 采用填充策略

# 填充机制可视化

Input Text → This is the first text

Token IDs → [1212, 318, 262, 717] → Padding Token <|endoftext|>

Padded IDs → [1212, 318, 262, 717, 50256, 50256...]

```
import tiktoken  
tokenizer = tiktoken.get_encoding("gpt2")  
print(tokenizer.encode("<|endoftext|>"))  
# Output: [50256]
```

# 构建 SpamDataset 类：初始化

```
class SpamDataset(Dataset):
    def __init__(self, csv_file, tokenizer, max_length=None,
                 pad_token_id=50256):
        self.data = pd.read_csv(csv_file)

        # A: 预先对所有文本进行分词
        self.encoded_texts = [
            tokenizer.encode(text) for text in self.data["Text"]
        ]

        # B: 确定最大序列长度
        if max_length is None:
            self.max_length = self._longest_encoded_length()
        else:
            self.max_length = max_length
```

A

使用 GPT-2 tokenizer 将文本转换为整数列表。

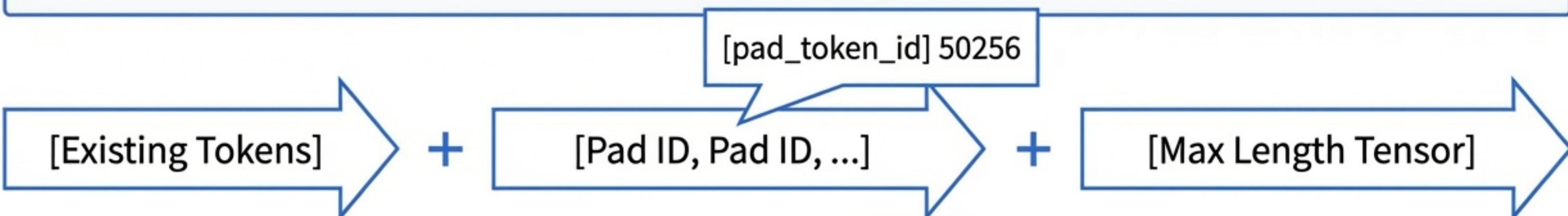
B

计算整个数据集中最长的序列，作为统一的 Tensor 尺寸。

# 核心实现：序列截断与填充逻辑

```
# C: 截断逻辑（若超过 max_length）
self.encoded_texts = [
    encoded_text[:self.max_length]
    for encoded_text in self.encoded_texts
]

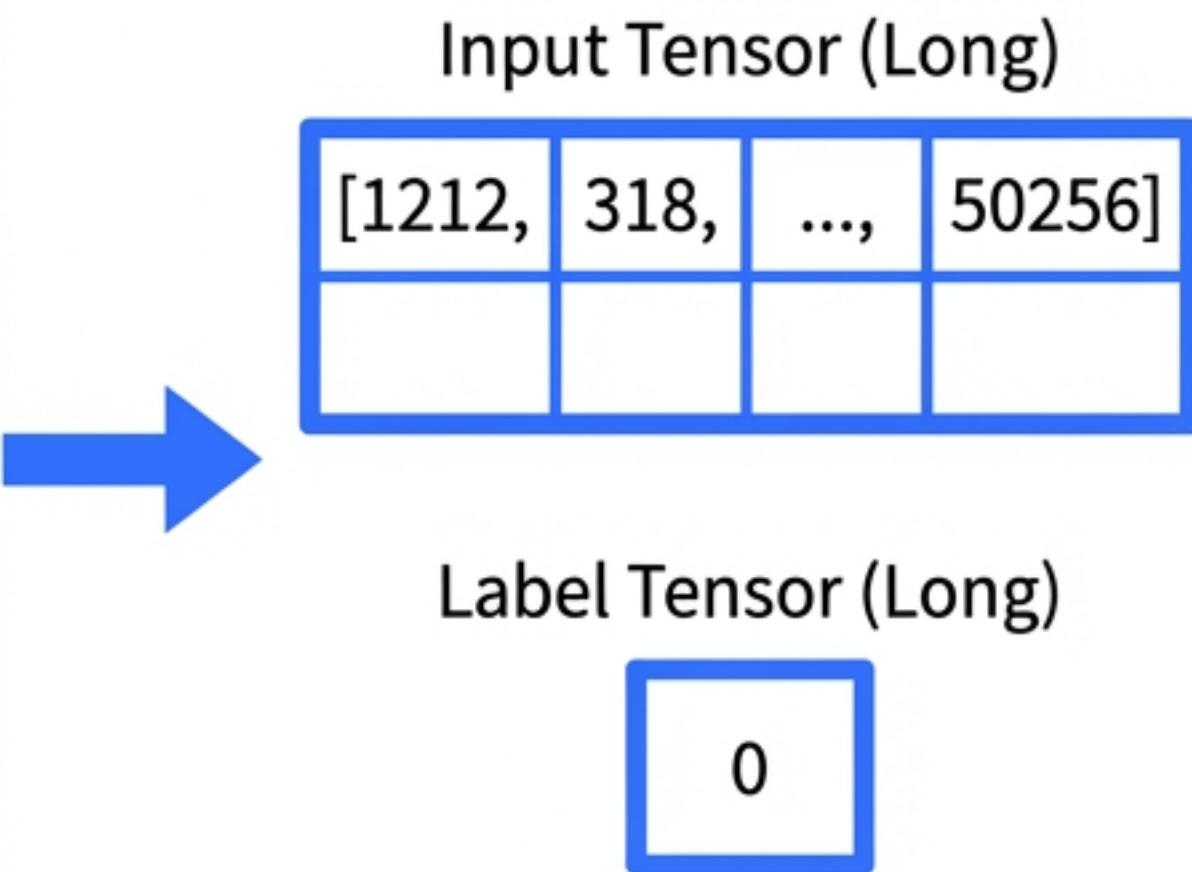
# D: 填充逻辑（核心代码）
self.encoded_texts = [
    encoded_text + [pad_token_id] * (self.max_length - len(encoded_text))
    for encoded_text in self.encoded_texts
]
```



# 数据获取：转换为 PyTorch 张量

```
def __getitem__(self, index):
    encoded = self.encoded_texts[index]
    label = self.data.iloc[index]["Label"]

    return (
        torch.tensor(encoded, dtype=torch.long),
        torch.tensor(label, dtype=torch.long)
    )
```

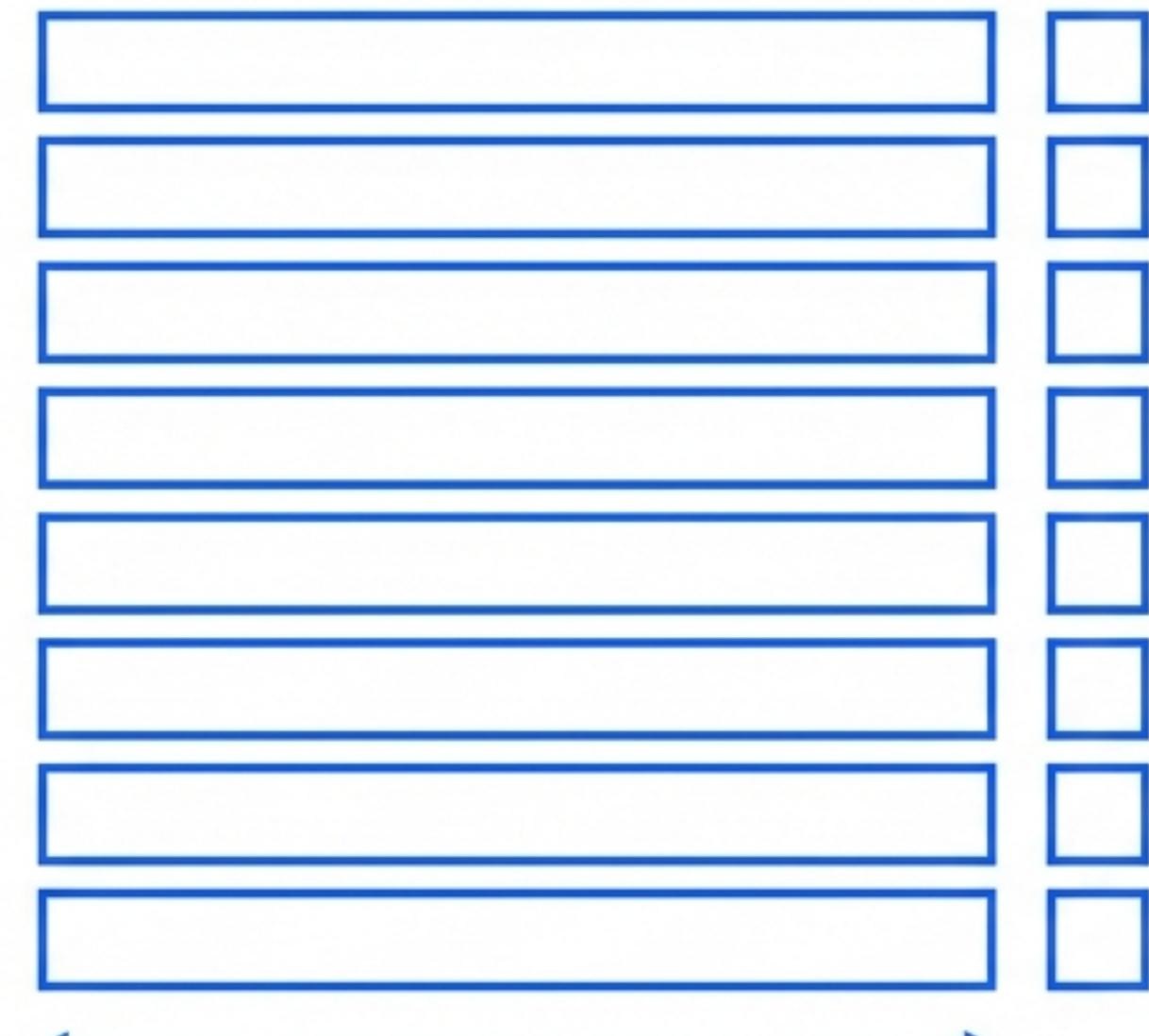


准备好输入 GPU (dtype=long is required for Embeddings)

# 配置数据加载器 (DataLoader)

```
train_loader = DataLoader(  
    dataset=train_dataset,  
    batch_size=8,      # 每批 8 条  
    shuffle=True,     # 训练集打乱  
    drop_last=True,   # 丢弃不完整批次  
    num_workers=0  
)
```

Batch Size = 8



Padded Sequence Length

# 验证数据流水线

```
>>> for input_batch, target_batch in train_loader:  
>>>     print("Input batch dimensions:", input_batch.shape)  
>>>     print("Label batch dimensions", target_batch.shape)  
>>>     break
```

```
Input batch dimensions: torch.Size([8, 120])  
Label batch dimensions: torch.Size([8])
```

Max Sequence Length  
(填充后的最大长度)

Batch Size  
(批大小)

数据流水线构建完成。Ready for Model Finetuning.