文本数据准备

文本数据准备

数据预处理流程

原始语料库

质量过滤

- 语种过滤
- 统计过滤
- 关键词过滤
- 分类器过滤

Alice is writing a paper about LLMs. ~~#$^&~~ Alice is writing a paper about LLMs.

敏感内容过滤

- 有毒内容
- 隐私内容（PII）

替换('Alice') is writing a paper about LLMs.

数据去重

- 句子级别
- 文档级别
- 数据集级别

Alice is writing a paper about LLMs. ~~Alice is writing a paper about LLMs.~~

词元化（分词）

- BPE 分词
- WordPiece 分词
- Unigram 分词

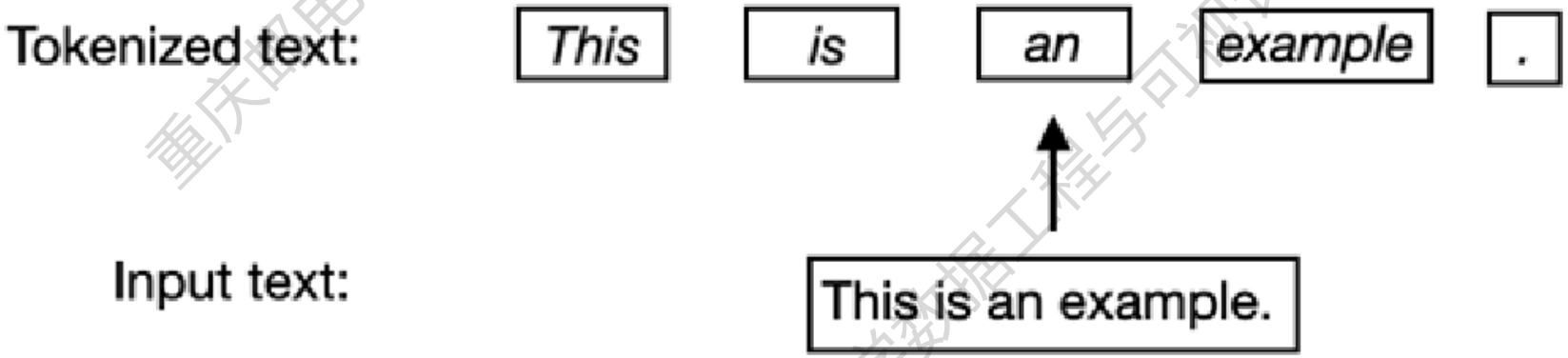
编码('[Somebody] is writing a paper about LLMs.')

准备预训练！

32, 145, 66, 79, 12, 56, ...

文本数据预处理

文本分词（词元化，Tokenization）是数据预处理的一个关键步骤，旨在将原始文本切分为模型可识别的词元（token）序列。 Token是大语言模型处理文本的最小单位，可以是单词或特殊字符，如标点符号等。



文本分词

本次实验用到的文本数据集是一部由Edith Wharton创作的英文短篇小说*The Verdict*，已将其复制到the-verdict.txt文件中。

实验任务1：从文件中读取文本数据，输出文本包含的字符数和前100个字符

```
with open("the-verdict.txt", "r", encoding="utf-8") as f:
    raw_text = f.read()

print("Total number of character:", len(raw_text))
print(raw_text[:100])
```

实验输出结果：

```
Total number of character: 20479
I HAD always thought Jack Gisburn rather a cheap genius--though a good fellow enough--so it was no g
```

文本分词

实验任务2: 使用正则表达式对读取的文本数据进行分词

```
import re
preprocessed = re.split(r'([,.:;?_!"()\']|--|\s)', raw_text)
preprocessed = [item.strip() for item in preprocessed if item.strip()]
print(preprocessed[:30])
```

实验输出结果:

```
4690
['I', 'HAD', 'always', 'thought', 'Jack', 'Gisburn', 'rather', 'a', 'cheap', 'genius', '--', 'though', 'a', 'good', 'fellow', 'en
ough', '--', 'so', 'it', 'was', 'no', 'great', 'surprise', 'to', 'me', 'to', 'hear', 'that', ',', 'in']
```

文本分词

实验任务3：根据文本分词结果构建词汇表，并输出词汇表大小和前10个词

```
all_words = sorted(set(preprocessed))
vocab_size = len(all_words)
vocab = {token:integer for integer,token in enumerate(all_words)}
print(vocab_size)
for i, item in enumerate(vocab.items()):
    print(item)
    if i >= 10:
        break
```

实验输出结果：

```
1130
('!', 0)
('"', 1)
("'", 2)
('(', 3)
(')', 4)
(',', 5)
('--', 6)
('.', 7)
(':', 8)
(';', 9)
('?', 10)
```

文本分词

实验任务4：实现一个简单的分词器，其中包含一个编码方法(encode)，负责将文本拆分为token，并通过词汇表将token字符转换为整数(token ID)。此外，还将实现一个解码方法(decode)，负责将token ID转换回文本。最后，在短篇小说文本数据上对实现的文本分词器进行编码和解码测试。

```python
class SimpleTokenizerV1:
    def __init__(self, vocab):
        self.str2id = vocab
        self.id2str = {i:s for s,i in vocab.items()}

    def encode(self, text):
        preprocessed = re.split(r'([,.?_!"()\']|--|\s)', text)
        preprocessed = [item for item in preprocessed if item.strip()]
        ids = [self.str2id[s] for s in preprocessed]
        return ids

    def decode(self, ids):
        text = " ".join(self.id2str[i] for i in ids)
        text = re.sub(r'\s+([,.?!"()\'])', r'\1', text)
        return text
```

文本分词

```
tokenizer = SimpleTokenizerV1(vocab)
text = raw_text[:99]
ids = tokenizer.encode(text)
print(ids)
print(tokenizer.decode(ids))
```

实验输出结果：

```
[53, 44, 149, 1003, 57, 38, 818, 115, 256, 486, 6, 1002, 115, 500, 435, 392, 6, 908, 585, 1077, 709]
I HAD always thought Jack Gisburn rather a cheap genius -- though a good fellow enough -- so it was no
```

文本分词

实验任务5：使用上一步创建的分词器对新的文本进行分词

```
text = "Hello, do you like tea?"
print(tokenizer.encode(text))
```

实验输出结果：

```
KeyError: 'Hello'
```

原因是我们根据短篇小说文本构建的词汇表没有包含Hello，一种简单的解决方法是在词汇表中加入一个特殊token（例如，<|unk|>）来表示不在词汇表中的词。而想要从根本上解决这一问题，则需要采用更先进的文本分词和编码方式，如GPT模型使用的字节对编码方法。

文本分词

字节对编码(Byte Pair Encoding)

从零开始实现BPE可能相对复杂，我们将使用一个名为tiktoken的现有Python开源库，该库非常高效地实现了BPE算法。

实验任务6：使用tiktoken工具库实现BPE分词

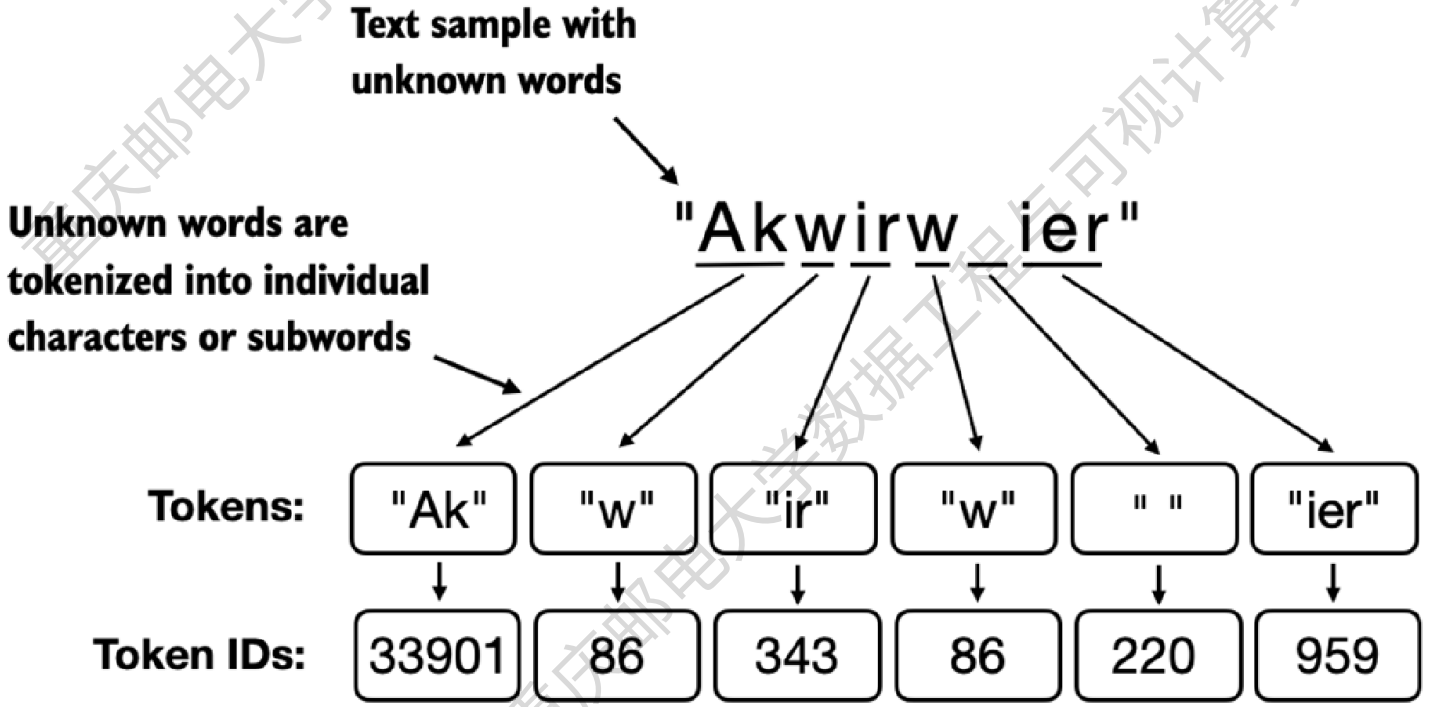
```
import tiktoken

tokenizer = tiktoken.get_encoding("gpt2")
text = "Hello, do you like tea? <|endoftext|> In the sunlit terraces of someunknownPlace."
integers = tokenizer.encode(text, allowed_special={"<|endoftext|>"})
print(integers)
strings = tokenizer.decode(integers)
print(strings)
```

实验输出结果：

```
[15496, 11, 466, 345, 588, 8887, 30, 220, 50256, 554, 262, 4252, 18250, 8812, 2114, 286, 617, 34680, 27271, 13]
Hello, do you like tea? <|endoftext|> In the sunlit terraces of someunknownPlace.
```

文本分词

通过上面的实验可以发现，BPE分词器能够正确编码和解码未知词汇，例如“someunknownPlace”。特殊标记<|endoftext|>可以用于分隔不同的文本，可以发现<|endoftext|>被分配了一个相对较大的token ID，即50256。这是因为BPE分词器被用于训练GPT-2、GPT-3以及ChatGPT的初始模型，它的词汇表包含了50256个token。BPE将不在其预定义词汇表中的单词分解为更小的子词单元甚至单个字符，使其能够处理超出词汇表的单词。

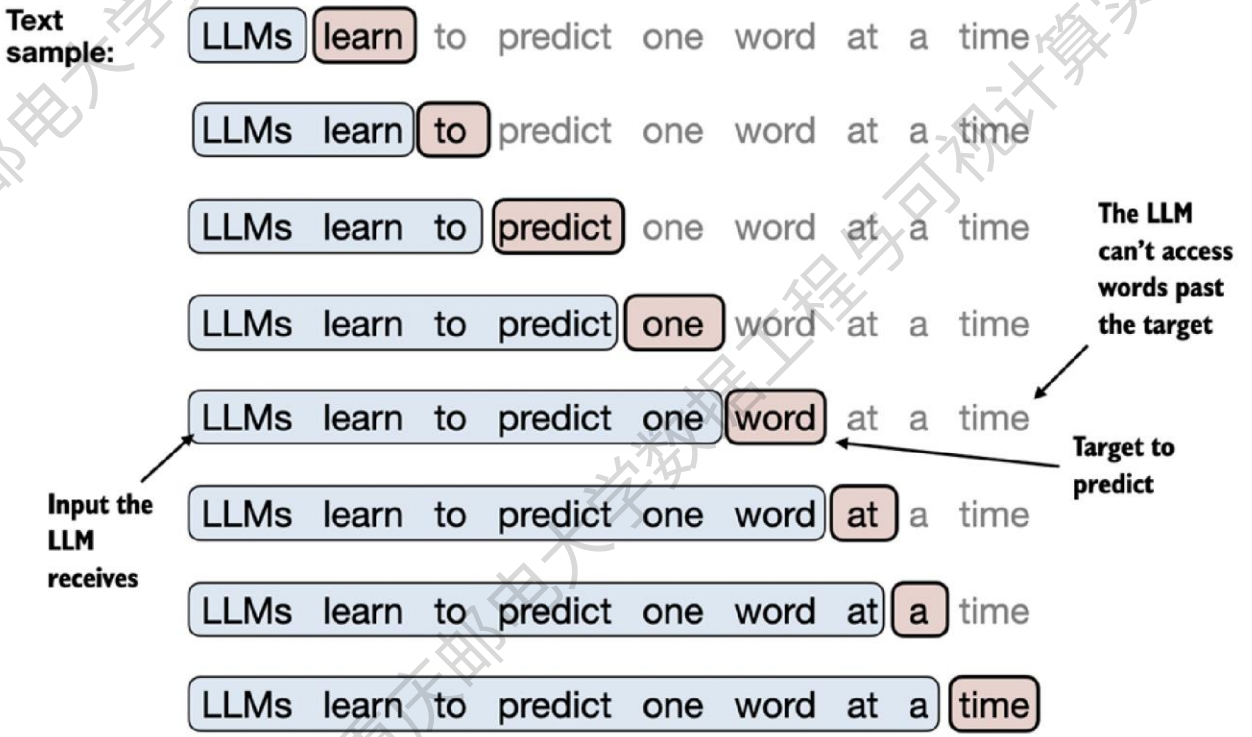


文本分词

课堂练习1：使用tiktoken 库中的BPE分词器对未知单词 “Akwirw ier” 进行分词，并输出token ID。然后，调用decode函数将每个token ID转换为字符串，查看输出结果，并自行查阅相关文献资料理解BPE的分词原理。

构建预训练数据

上一节详细介绍了分词步骤以及将字符串分词成token再转换为整数token ID的过程。但是，这些数据还不能直接用于大语言模型的预训练，我们需要生成训练大语言模型所需的训练样本，每个训练样本由输入-目标（input-target）构成，其中目标是输入的下一个词。



构建预训练数据

在本节中，我们将实现一个数据加载器，通过滑动窗口方法从训练数据集中提取输入-目标。

实验任务7：使用前一节中介绍的BPE分词器对短篇小说进行分词，并输出token总数

```
with open("the-verdict.txt", "r", encoding="utf-8") as f:
    raw_text = f.read()

enc_text = tokenizer.encode(raw_text)
print(len(enc_text))
```

实验输出结果：5145

构建预训练数据

实验任务8：根据上一步的分词结果创建输入-目标样本，设上下文窗口大小为4，删除了原始文本中前50个token，以便观察输出结果。

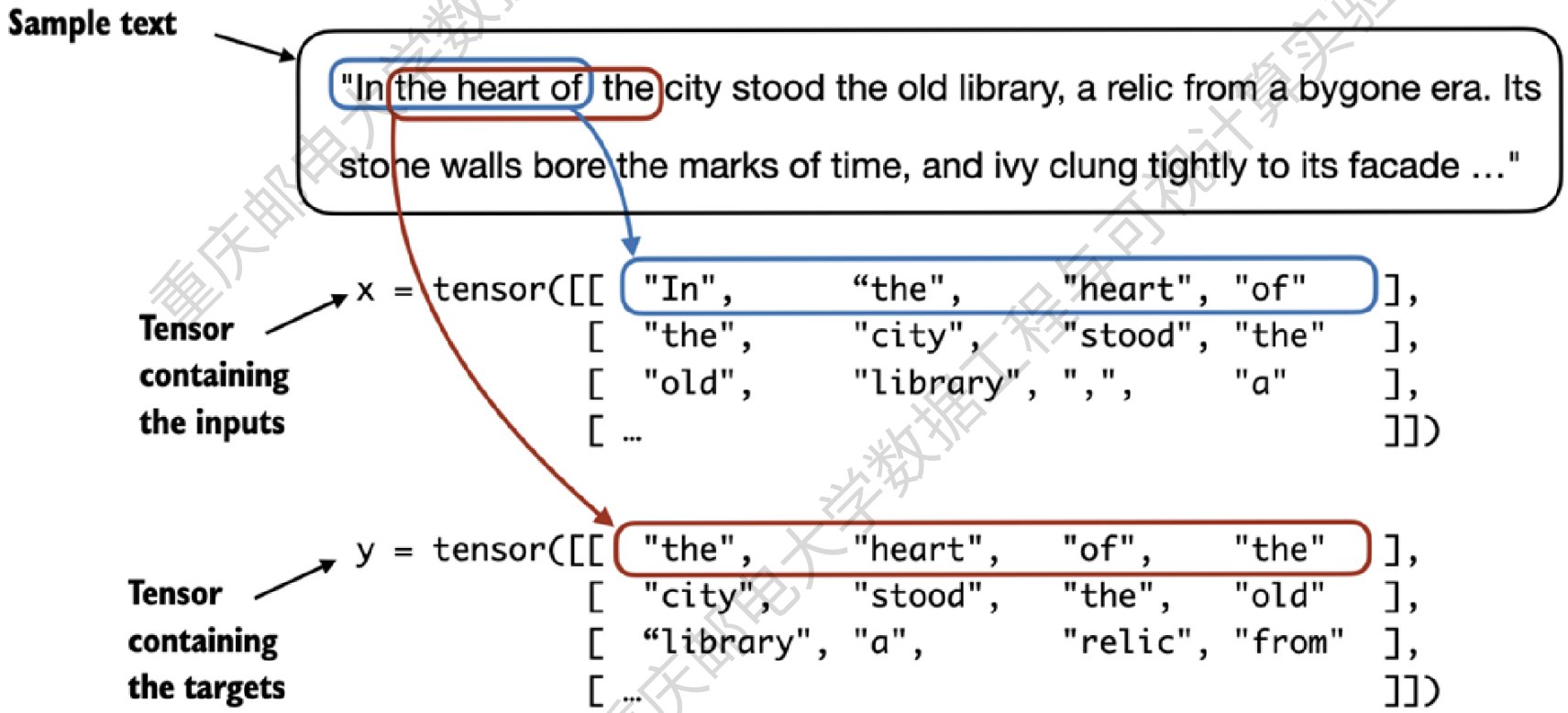
```
enc_sample = enc_text[50:]
context_size = 4
for i in range(1, context_size+1):
    context = enc_sample[:i]
    desired = enc_sample[i]
    print(tokenizer.decode(context), "---->", tokenizer.decode([desired]))
```

实验输出结果：

```
and ---->  established
and established ---->  himself
and established himself ---->  in
and established himself in ---->  a
```

构建预训练数据

在使用生成的训练样本对大语言模型进行预训练之前，还需要实现一个高效的数据加载器，该加载器遍历输入数据集并将输入和目标作为PyTorch 张量返回，这些张量可以视为多维数组。具体来说，我们的目标是返回两个张量：一个输入张量，作为大语言模型的输入，另一个目标张量，包含大语言模型需要预测的目标。此外，加载器能够很方便的批量生成训练数据。



构建预训练数据

实验任务9：使用 PyTorch 内置的 Dataset 和 DataLoader 类实现高效的数据加载器，并通过这个加载器从原始文本中生成预训练数据。

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

class GPTDatasetV1(Dataset):
    def __init__(self, text, tokenizer, max_length, stride):
        self.input_ids = []
        self.target_ids = []
        token_ids = tokenizer.encode(text)
        for i in range(0, len(token_ids)-max_length, stride):
            input = token_ids[i:i+max_length]
            target = token_ids[i+1:i+max_length+1]
            self.input_ids.append(torch.tensor(input))
            self.target_ids.append(torch.tensor(target))

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

    def __getitem__(self, idx):
        return self.input_ids[idx], self.target_ids[idx]
```

构建预训练数据

```python
# 从原始文本数据中生成预训练数据集，并通过DataLoader加载数据集
with open("the-verdict.txt", "r", encoding="utf-8") as f:
    raw_text = f.read()
tokenizer = tiktoken.get_encoding("gpt2")
dataset = GPTDatasetV1(raw_text, tokenizer, max_length=4, stride=1)

dataloader = DataLoader(dataset, batch_size=3, shuffle=False)
for i, (input, target) in enumerate(dataloader):
    if i > 2:
        break
    print(f'Batch {i+1} Input:', input)
    print(f'Batch {i+1} Target:', target)
```

batch_size指定了加载器每次生成训练样本的数量，大语言模型在训练时每次读入一批训练样本，可以通过GPU并行计算，从而加速训练过程。
思考：为什么不一次读入所有训练样本？或者每次读入一个数据样本？

实验输出结果：

```
Batch 1 Input: tensor([[  40,  367, 2885, 1464],
        [ 367, 2885, 1464, 1807],
        [2885, 1464, 1807, 3619]])
Batch 1 Target: tensor([[ 367, 2885, 1464, 1807],
        [2885, 1464, 1807, 3619],
        [1464, 1807, 3619,  402]])
Batch 2 Input: tensor([[ 1464,  1807,  3619,   402],
        [ 1807,  3619,   402,   271],
        [ 3619,   402,   271, 10899]])
Batch 2 Target: tensor([[ 1807,  3619,   402,   271],
        [ 3619,   402,   271, 10899],
        [  402,   271, 10899,  2138]])
Batch 3 Input: tensor([[  402,   271, 10899,  2138],
        [  271, 10899,  2138,   257],
        [10899,  2138,   257,  7026]])
Batch 3 Target: tensor([[  271, 10899,  2138,   257],
        [10899,  2138,   257,  7026],
        [ 2138,   257,  7026, 15632]])
```

构建预训练数据

课堂练习2：为了更好地理解数据加载器的工作原理，请尝试使用不同的参数设置进行测试，例如，设置不同的步幅(stride)、上下文大小(max_length)和批量大小(batch_size)。

课后任务：查阅相关文献资料学习理解Transformer结构和自注意力机制