调试、性能分析与元编程实验报告

王杰（23160001074）

2025 年 9 月 21 日

1　实验目的

- 学习常见调试方法与性能分析思路：命令追踪、日志、交互式调试、CPU/时间开销分析等。
- 了解元编程理念：**用程序生成程序/配置**，使用模板、规则与构建系统减少重复劳动。
- 上手 PyTorch 基础：张量与自动求导、简单线性回归训练。

2　实验环境

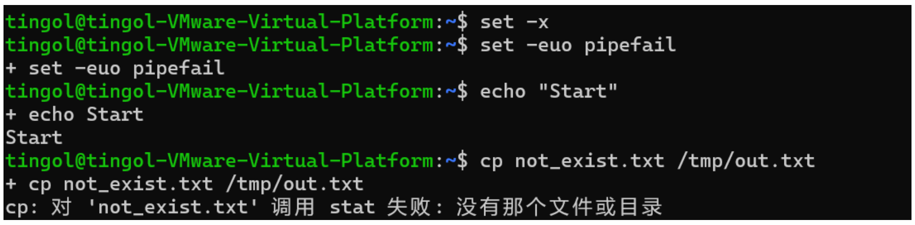
- 操作系统：Windows 11（WSL2: Ubuntu 24.04）
- Shell：`bash`（支持 `set -x`、`trap`、`xargs` 等）
- Python：`Python 3.11+`；常用模块：`logging`、`timeit`、`cProfile`、`pdb`
- 开发工具：`make`、`sed`、`awk`、
- 深度学习：`pytorch`
- LaTeX 编译器：**XeLaTeX**；在线：Overleaf

3　练习内容

3.1　调试与性能分析（9 个）

1. **Bash 行级追踪：快速定位脚本出错行**

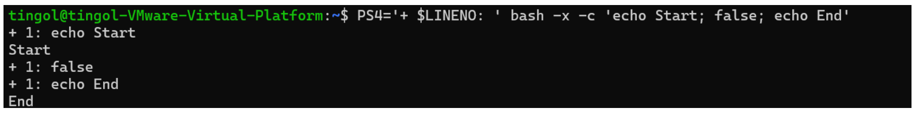
```
# script.sh
set -x              # 开启调试追踪
set -euo pipefail
echo "Start"
cp not_exist.txt /tmp/out.txt   # 故意制造错误
echo "End"
# 运行: bash script.sh
```



```
tingol@tingol-VMware-Virtual-Platform:~$ set -x
tingol@tingol-VMware-Virtual-Platform:~$ set -euo pipefail
+ set -euo pipefail
tingol@tingol-VMware-Virtual-Platform:~$ echo "Start"
+ echo Start
Start
tingol@tingol-VMware-Virtual-Platform:~$ cp not_exist.txt /tmp/out.txt
+ cp not_exist.txt /tmp/out.txt
cp: 对 'not_exist.txt' 调用 stat 失败: 没有那个文件或目录
```

2. 只临时追踪某条命令

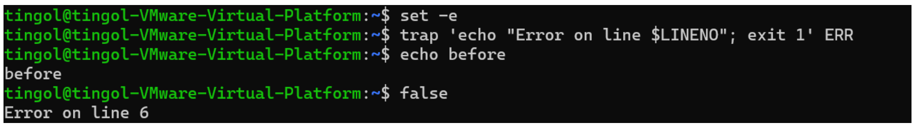
```
bash -x myscript.sh
PS4='+ $LINENO: ' bash -x myscript.sh
```



```
tingol@tingol-VMware-Virtual-Platform:~$ PS4='+ $LINENO: ' bash -x -c 'echo Start; false; echo End'
+ 1: echo Start
Start
+ 1: false
+ 1: echo End
End
```

3. 出错时打印上下文: trap

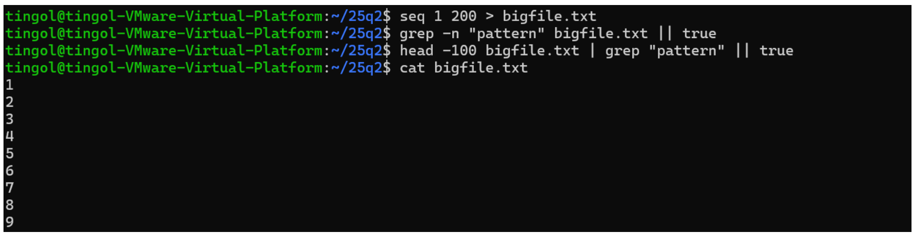
```
#!/usr/bin/env bash
set -euo pipefail
trap 'echo "Error on line $LINENO"; exit 1' ERR
do_something() { false; }
echo "before"; do_something; echo "after"
```



```
tingol@tingol-VMware-Virtual-Platform:~$ set -e
tingol@tingol-VMware-Virtual-Platform:~$ trap 'echo "Error on line $LINENO"; exit 1' ERR
tingol@tingol-VMware-Virtual-Platform:~$ echo before
before
tingol@tingol-VMware-Virtual-Platform:~$ false
Error on line 6
```

4. 最小可复现样例（MRE）: 收敛问题拆小

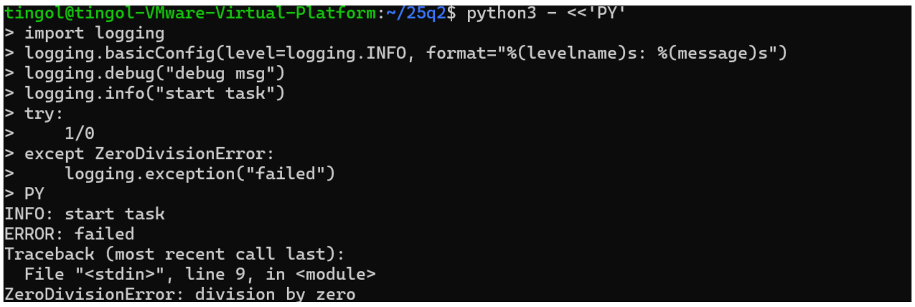
```
# 将复杂命令拆成最小示例（示意）
grep -n "pattern" bigfile.txt  # => 失败
head -100 bigfile.txt | grep "pattern"  # 在小范围先复现
```



```
tingol@tingol-VMware-Virtual-Platform:~/25q2$ seq 1 200 > bigfile.txt
tingol@tingol-VMware-Virtual-Platform:~/25q2$ grep -n "pattern" bigfile.txt || true
tingol@tingol-VMware-Virtual-Platform:~/25q2$ head -100 bigfile.txt | grep "pattern" || true
tingol@tingol-VMware-Virtual-Platform:~/25q2$ cat bigfile.txt
1
2
3
4
5
6
7
8
9
```

5. **记录日志：Python logging 基础用法**

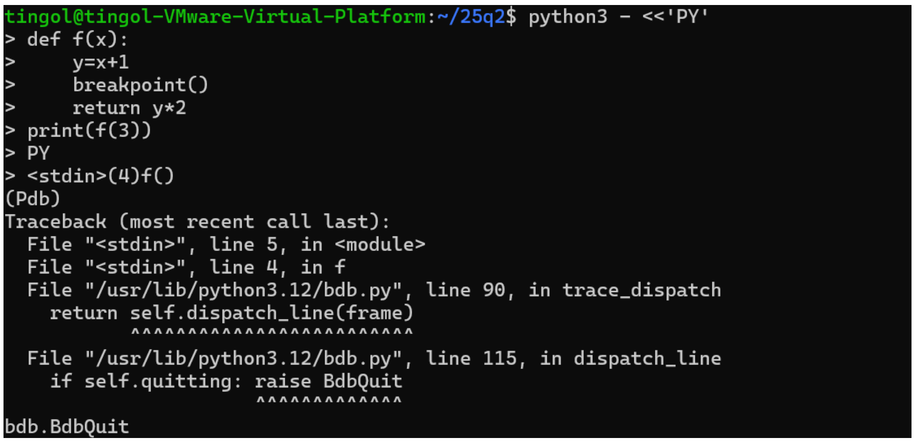
```
import logging
logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)
    s")
logging.debug("debug msg (默认不可见)")
logging.info("start task")
try:
    1/0
except ZeroDivisionError:
    logging.exception("failed")
```



```
tingol@tingol-VMware-Virtual-Platform:~/25q2$ python3 - <<'PY'
> import logging
> logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
> logging.debug("debug msg")
> logging.info("start task")
> try:
>     1/0
> except ZeroDivisionError:
>     logging.exception("failed")
> PY
INFO: start task
ERROR: failed
Traceback (most recent call last):
  File "<stdin>", line 9, in <module>
ZeroDivisionError: division by zero
```

6. **交互式单步：pdb 与 breakpoint()**

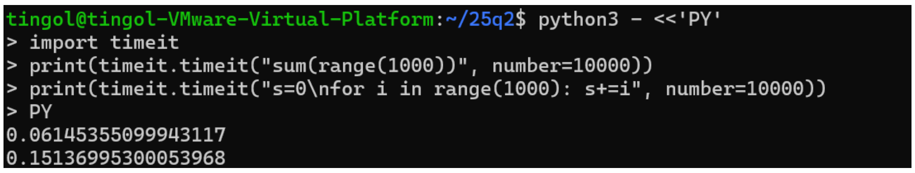
```
def f(x):
    y = x + 1
    breakpoint()  # 运行 python -O 将跳过；平时可进入 pdb
    return y * 2

print(f(3))
# 也可命令行：python -m pdb your_script.py
```



```
tingol@tingol-VMware-Virtual-Platform:~/25q2$ python3 - <<'PY'
> def f(x):
>     y=x+1
>     breakpoint()
>     return y*2
> print(f(3))
> PY
> <stdin>(4)f()
(Pdb)
Traceback (most recent call last):
  File "<stdin>", line 5, in <module>
  File "<stdin>", line 4, in f
  File "/usr/lib/python3.12/bdb.py", line 90, in trace_dispatch
    return self.dispatch_line(frame)
           ^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/lib/python3.12/bdb.py", line 115, in dispatch_line
    if self.quitting: raise BdbQuit
                      ^^^^^^^^^^^^^
bdb.BdbQuit
```

7. **微基准：`timeit` 对比两种写法**

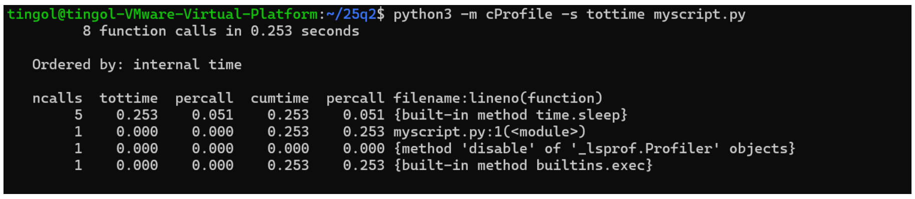
```
import timeit
print(timeit.timeit("sum(range(1000))", number=10000))
print(timeit.timeit("s=0\nfor i in range(1000): s+=i", number=10000))
```



```
tingol@tingol-VMware-Virtual-Platform:~/25q2$ python3 - <<'PY'
> import timeit
> print(timeit.timeit("sum(range(1000))", number=10000))
> print(timeit.timeit("s=0\nfor i in range(1000): s+=i", number=10000))
> PY
0.061453550099943117
0.15136995300053968
```

8. **整程序 CPU 分析：`cProfile`**

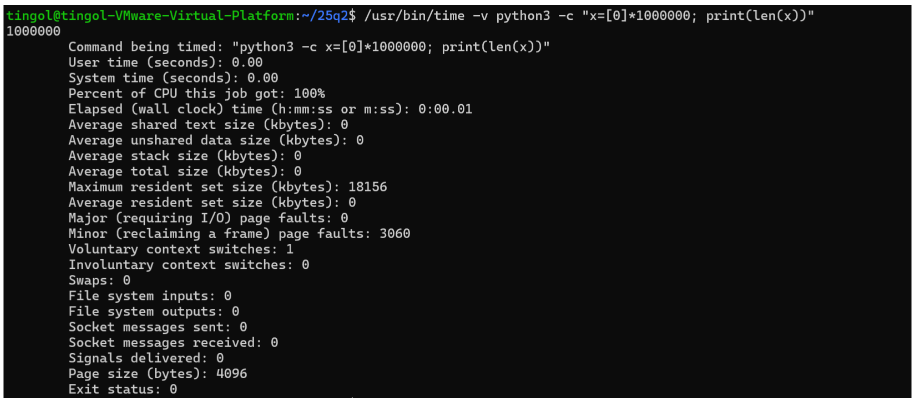
```
# save as work.py
import time
def slow():
    for _ in range(5):
        time.sleep(0.05)
slow()
# 运行并排序耗时：python -m cProfile -s tottime work.py
```



```
tingol@tingol-VMware-Virtual-Platform:~/25q2$ python3 -m cProfile -s tottime myscript.py
         8 function calls in 0.253 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        5    0.253    0.051    0.253    0.051 {built-in method time.sleep}
        1    0.000    0.000    0.253    0.253 myscript.py:1(<module>)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
        1    0.000    0.000    0.253    0.253 {built-in method builtins.exec}
```

9. **测内存/时间峰值：/usr/bin/time**

```
/usr/bin/time -v python -c "x=[0]*10_000_000; print(len(x))"
# 关键指标：Elapsed (wall clock) time / Maximum resident set size
```



```
tingol@tingol-VMware-Virtual-Platform:~/25q2$ /usr/bin/time -v python3 -c "x=[0]*1000000; print(len(x))"
1000000
        Command being timed: "python3 -c x=[0]*1000000; print(len(x))"
        User time (seconds): 0.00
        System time (seconds): 0.00
        Percent of CPU this job got: 100%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.01
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 18156
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 3060
        Voluntary context switches: 1
        Involuntary context switches: 0
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

3.2 元编程与自动化（9 个）

1. **花括号展开：一次生成多段相似代码/文件**

```
echo {dev,staging,prod}-config.yaml
touch file-{a..c}{1..3}.txt
```

2. **Here-Doc 模板：变量注入生成配置**

```
APP=demo
cat > config.yaml <<EOF
app: $APP
port: 8080
log_level: INFO
EOF
```

3. **sed 模板替换：占位符批量注入**

```
# template.conf 含 {{PORT}} 与 {{MODE}}
sed -e "s/{{PORT}}/8080/g" -e "s/{{MODE}}/release/g" \
    template.conf > app.conf
```

4. **awk 生成命令 + 管道执行（小心使用）**

```
# names.txt 每行一个目录名
awk '{print "mkdir -p out/"$1}' names.txt | sh
```

5. **xargs 并行批处理（避免手写循环）**

```
ls *.jpg | xargs -n1 -P4 -I{} convert "{}" -resize 512x512 "out/{}"
# -P4 表示并行 4 个任务（需 ImageMagick 的 convert）
```

6. **用 make 管理“规则”：自动根据依赖更新**

```
# Makefile
out/%.txt: src/%.txt
  @mkdir -p out
  cp $< $@
# 使用: make out/a.txt    ($< 为源, $@ 为目标)
```

7. **批量文档转换规则（配合 pandoc）**

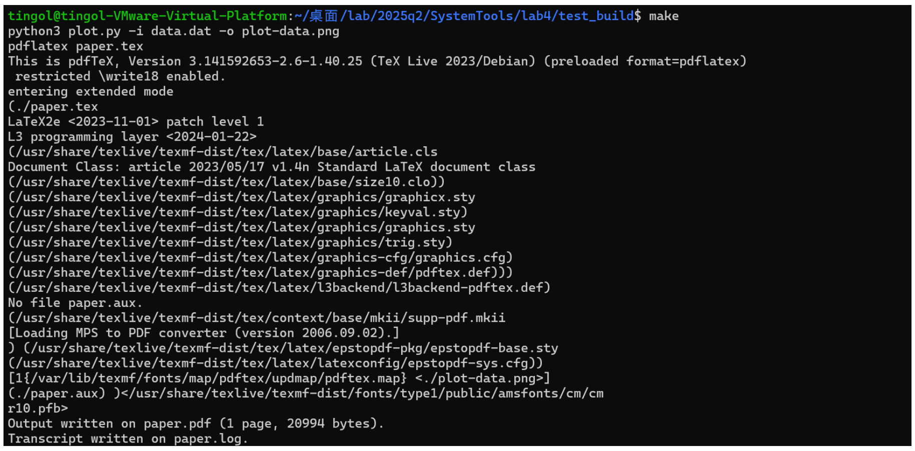
```
MD := $(wildcard docs/*.md)
HTML := $(MD:.md=.html)

all: $(HTML)

docs/%.html: docs/%.md
  pandoc $< -o $@
# 使用: make 或 make -j
```

8. **Python 装饰器：为函数“自动加功能”（计时）**

```
import time, functools
def timed(fn):
    @functools.wraps(fn)
    def wrap(*a, **kw):
        t0=time.time(); r=fn(*a,**kw)
        print(f"{fn.__name__} took {time.time()-t0:.3f}s")
        return r
    return wrap

@timed
def job(): time.sleep(0.2)
job()
```

9. **数据驱动生成：用列表驱动批量创建文件**

```
from pathlib import Path
cfgs = [{"name":"dev","port":8000},{"name":"prod","port":80}]
tpl = "name: {name}\nport: {port}\n"
for c in cfgs:
    Path(f"{c['name']}.yaml").write_text(tpl.format(**c), encoding="utf-8
    ")
```



```
tingol@tingol-VMware-Virtual-Platform:~/桌面/lab/2025q2/SystemTools/lab4/test_build$ make
python3 plot.py -i data.dat -o plot-data.png
pdflatex paper.tex
This is pdfTeX, Version 3.141592653-2.6-1.40.25 (TeX Live 2023/Debian) (preloaded format=pdflatex)
 restricted \write18 enabled.
entering extended mode
(./paper.tex
LaTeX2e <2023-11-01> patch level 1
L3 programming layer <2024-01-22>
(/usr/share/texlive/texmf-dist/tex/latex/base/article.cls
Document Class: article 2023/05/17 v1.4n Standard LaTeX document class
(/usr/share/texlive/texmf-dist/tex/latex/base/size10.clo))
(/usr/share/texlive/texmf-dist/tex/latex/graphics/graphicx.sty
(/usr/share/texlive/texmf-dist/tex/latex/graphics/keyval.sty)
(/usr/share/texlive/texmf-dist/tex/latex/graphics/graphics.sty
(/usr/share/texlive/texmf-dist/tex/latex/graphics/trig.sty)
(/usr/share/texlive/texmf-dist/tex/latex/graphics-cfg/graphics.cfg)
(/usr/share/texlive/texmf-dist/tex/latex/graphics-def/pdftex.def)))
(/usr/share/texlive/texmf-dist/tex/latex/l3backend/l3backend-pdftex.def)
No file paper.aux.
(/usr/share/texlive/texmf-dist/tex/context/base/mkii/supp-pdf.mkii
[Loading MPS to PDF converter (version 2006.09.02).]
) (/usr/share/texlive/texmf-dist/tex/latex/epstopdf-pkg/epstopdf-base.sty
(/usr/share/texlive/texmf-dist/tex/latex/latexconfig/epstopdf-sys.cfg))
[1{/var/lib/texmf/fonts/map/pdftex/updmap/pdftex.map} <./plot-data.png>]
(./paper.aux) )</usr/share/texlive/texmf-dist/fonts/type1/public/amsfonts/cm/cm
r10.pfb>
Output written on paper.pdf (1 page, 20994 bytes).
Transcript written on paper.log.
```

3.3 PyTorch 简例（2 个）

1. **张量与自动求导：二次函数梯度**

```python
import torch
x = torch.tensor(3.0, requires_grad=True)
y = x**2 + 2*x + 1
y.backward()
print("dy/dx at x=3 =", x.grad.item())  # 2x+2 => 8
```

2. **文本情感分析**

```python
#   python test2.py --fast
#   python test2.py --epochs 3 --batch-size 64 --limit 12000 --seed 42

import os
import re
import io
import tarfile
import random
import argparse
import urllib.request
from collections import Counter
from typing import List, Tuple

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader

# tqdm 可选
try:
    from tqdm import tqdm
except Exception:
    def tqdm(x, **kwargs):
        return x

IMDB_URL = "https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.
    gz"


# ------------------------------
# 下载与解压 IMDb
# ------------------------------
def download_and_extract_imdb(root: str = "./data") -> str:
    os.makedirs(root, exist_ok=True)
    tar_path = os.path.join(root, "aclImdb_v1.tar.gz")
    target_dir = os.path.join(root, "aclImdb")

    if os.path.isdir(target_dir) and os.path.isdir(os.path.join(
    target_dir, "train")):
        print(f"[Info] IMDb 已存在: {target_dir}")
        return target_dir

    if not os.path.exists(tar_path):
        print(f"[Info] 正在下载 IMDb 数据集到 {tar_path} ... (约 80MB) ")
        urllib.request.urlretrieve(IMDB_URL, tar_path)
        print("[Info] 下载完成。")

    print(f"[Info] 正在解压到 {root} ...")
    with tarfile.open(tar_path, "r:gz") as tar:
        tar.extractall(path=root)
    print("[Info] 解压完成。")
    return target_dir


# ------------------------------
# 文本清洗与分词
# ------------------------------
def clean_text(s: str) -> str:
    s = s.lower().replace("<br />", " ").replace("<br>", " ")
    return s
```

```
_TOKEN_RE = re.compile(r"[a-z0-9']+")

def simple_tokenize(s: str) -> List[str]:
    s = clean_text(s)
    toks = _TOKEN_RE.findall(s)
    if not toks:
        return ["<unk>"]
    return toks


# -----------------------------
# 词表
# -----------------------------
class Vocab:
    def __init__(self, counter: Counter, max_size: int = 25000, min_freq:
     int = 2,
                 specials: List[str] = None):
        if specials is None:
            specials = ["<pad>", "<unk>"]
        self.itos = list(specials)

        for word, freq in counter.most_common():
            if freq < min_freq:
                continue
            if word in specials:
                continue
            self.itos.append(word)
            if len(self.itos) >= max_size:
                break
        self.stoi = {w: i for i, w in enumerate(self.itos)}
        self.pad_index = self.stoi["<pad>"]
        self.unk_index = self.stoi["<unk>"]

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

    def numericalize(self, tokens: List[str]) -> List[int]:
        unk = self.unk_index
        stoi = self.stoi
        return [stoi.get(tok, unk) for tok in tokens]


# ----------------------------
# 读取 IMDb (tokens, label)
# ----------------------------
def read_split(split_dir: str, limit: int = None, seed: int = 42) -> List
    [Tuple[List[str], int]]:
    pos_dir = os.path.join(split_dir, "pos")
    neg_dir = os.path.join(split_dir, "neg")
    data = []

    def read_dir(d, label):
        files = [os.path.join(d, f) for f in os.listdir(d) if f.endswith(
    ".txt")]
        for fp in files:
            with io.open(fp, "r", encoding="utf-8") as f:
                text = f.read()
            tokens = simple_tokenize(text)
            data.append((tokens, label))

    read_dir(pos_dir, 1)
    read_dir(neg_dir, 0)

    random.Random(seed).shuffle(data)
    if limit is not None:
        data = data[:limit]
    return data


# ----------------------------
# Dataset & collate
# ----------------------------
class TextClsDataset(Dataset):
    def __init__(self, examples: List[Tuple[List[str], int]], vocab:
    Vocab):
        self.examples = examples
        self.vocab = vocab

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

    def __getitem__(self, idx):
        tokens, label = self.examples[idx]
        ids = self.vocab.numericalize(tokens)
        return torch.tensor(ids, dtype=torch.long), torch.tensor(label,
   dtype=torch.float32)


def pad_sequences(seqs: List[torch.Tensor], pad_idx: int):
    lengths = torch.tensor([len(s) for s in seqs], dtype=torch.long)
    max_len = int(lengths.max().item())
    padded = torch.full((len(seqs), max_len), pad_idx, dtype=torch.long)
    for i, s in enumerate(seqs):
        padded[i, : len(s)] = s
    return padded, lengths


def collate_fn(batch, pad_idx):
    seqs, labels = zip(*batch)
    padded, lengths = pad_sequences(list(seqs), pad_idx)
    labels = torch.stack(labels, dim=0)
    return padded, lengths, labels


# -----------------------------
# 模型: 双向 LSTM
# -----------------------------
class SentimentLSTM(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim
   =1, n_layers=2, dropout=0.5):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim,
   padding_idx=0)
        self.lstm = nn.LSTM(
            embedding_dim,
            hidden_dim,
            num_layers=n_layers,
            batch_first=True,
            bidirectional=True,
            dropout=dropout if n_layers > 1 else 0.0,
        )
        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(hidden_dim * 2, output_dim)

    def forward(self, text, lengths):
        # text: [B, T]
        embedded = self.dropout(self.embedding(text))  # [B, T, E]
        packed = nn.utils.rnn.pack_padded_sequence(
            embedded, lengths.cpu(), batch_first=True, enforce_sorted=
    False
        )
        _, (hidden, _) = self.lstm(packed)
        # hidden: [num_layers*2, B, H] -> 取最后一层双向的正/反向
        last_fw = hidden[-2, :, :]
        last_bw = hidden[-1, :, :]
        cat = torch.cat((last_fw, last_bw), dim=1)  # [B, 2H]
        cat = self.dropout(cat)
        logits = self.fc(cat).squeeze(1)  # [B]
        return logits


# -----------------------------
# 训练 / 评估 / 准确率
# -----------------------------
def binary_accuracy_from_logits(logits, labels):
    preds = (torch.sigmoid(logits) >= 0.5).float()
    correct = (preds == labels).float().sum()
    return correct / labels.numel()


@torch.no_grad()
def evaluate(model, loader, device, criterion):
    model.eval()
    total_loss, total_acc, n_batch = 0.0, 0.0, 0
    for x, lengths, y in loader:
        x, lengths, y = x.to(device), lengths.to(device), y.to(device)
        logits = model(x, lengths)
        loss = criterion(logits, y)
        acc = binary_accuracy_from_logits(logits, y)
```

```
            total_loss += loss.item()
            total_acc += acc.item()
            n_batch += 1
    return total_loss / n_batch, total_acc / n_batch


def train(model, train_loader, valid_loader, device, epochs=2, lr=1e-3,
    clip=1.0):
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
    criterion = nn.BCEWithLogitsLoss().to(device)

    best_valid = float("inf")
    for ep in range(1, epochs + 1):
        model.train()
        pbar = tqdm(train_loader, desc=f"Epoch {ep}/{epochs}")
        total_loss, total_acc, n_batch = 0.0, 0.0, 0
        for x, lengths, y in pbar:
            x, lengths, y = x.to(device), lengths.to(device), y.to(device
    )

            optimizer.zero_grad()
            logits = model(x, lengths)
            loss = criterion(logits, y)
            acc = binary_accuracy_from_logits(logits, y)
            loss.backward()
            if clip is not None:
                nn.utils.clip_grad_norm_(model.parameters(), clip)
            optimizer.step()

            total_loss += loss.item()
            total_acc += acc.item()
            n_batch += 1
            if hasattr(pbar, "set_postfix"):
                pbar.set_postfix(loss=f"{total_loss/n_batch:.4f}", acc=f"
    {total_acc/n_batch:.4f}")

        val_loss, val_acc = evaluate(model, valid_loader, device,
    criterion)
        print(f"[Eval] valid_loss={val_loss:.4f}, valid_acc={val_acc:.4f}
    ")
```

```
        if val_loss < best_valid:
            best_valid = val_loss
            torch.save(model.state_dict(), "best_model.pt")
            print("[Info] 已保存最佳模型 -> best_model.pt")


# -----------------------------
# 推理
# -----------------------------
@torch.no_grad()
def predict_sentiment(model, sentence: str, vocab: Vocab, device: str = "
   cpu"):
    model.eval()
    tokens = simple_tokenize(sentence)
    ids = vocab.numericalize(tokens)
    x = torch.tensor(ids, dtype=torch.long).unsqueeze(0)  # [1, T]
    lengths = torch.tensor([len(ids)], dtype=torch.long)
    x, lengths = x.to(device), lengths.to(device)
    logit = model(x, lengths)
    prob = torch.sigmoid(logit).item()  # 概率越接近 1 越正面
    return prob


# -----------------------------
# 主函数
# -----------------------------
def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--data", type=str, default="./data", help="数据根目录
   ")
    ap.add_argument("--max-vocab", type=int, default=25000)
    ap.add_argument("--min-freq", type=int, default=2)
    ap.add_argument("--embedding-dim", type=int, default=100)
    ap.add_argument("--hidden-dim", type=int, default=256)
    ap.add_argument("--layers", type=int, default=2)
    ap.add_argument("--dropout", type=float, default=0.5)
    ap.add_argument("--batch-size", type=int, default=64)
    ap.add_argument("--epochs", type=int, default=2)
    ap.add_argument("--lr", type=float, default=1e-3)
    ap.add_argument("--limit", type=int, default=None, help="每个 split 使
    用的样本数上限（train/test 分别截取）")
    ap.add_argument("--valid-ratio", type=float, default=0.1)
    ap.add_argument("--seed", type=int, default=2025)
    ap.add_argument("--fast", action="store_true", help="快速模式：自动
    limit=12000, epochs=1")
    args = ap.parse_args()

    if args.fast:
        if args.limit is None:
            args.limit = 12000
        if args.epochs == 2:
            args.epochs = 1

    random.seed(args.seed)
    torch.manual_seed(args.seed)

    device = "cuda" if torch.cuda.is_available() else "cpu"
    print(f"[Info] 使用设备：{device}")

    imdb_dir = download_and_extract_imdb(args.data)

    train_examples = read_split(os.path.join(imdb_dir, "train"), limit=
    args.limit, seed=args.seed)
    test_examples = read_split(os.path.join(imdb_dir, "test"), limit=args
    .limit, seed=args.seed)

    # 训练/验证划分
    n_valid = max(1, int(len(train_examples) * args.valid_ratio))
    valid_examples = train_examples[:n_valid]
    train_examples = train_examples[n_valid:]

    # 构建词表（基于训练集）
    counter = Counter()
    for tokens, _ in train_examples:
        counter.update(tokens)
    vocab = Vocab(counter, max_size=args.max_vocab, min_freq=args.
    min_freq)
    print(f"[Info] 词表大小：{len(vocab)}（含 <pad>/<unk>）")

    # Dataset & DataLoader
```

```
train_ds = TextClsDataset(train_examples, vocab)
valid_ds = TextClsDataset(valid_examples, vocab)
test_ds  = TextClsDataset(test_examples,  vocab)

collate = lambda batch: collate_fn(batch, vocab.pad_index)
train_loader = DataLoader(train_ds, batch_size=args.batch_size,
shuffle=True,  collate_fn=collate)
valid_loader = DataLoader(valid_ds, batch_size=args.batch_size,
shuffle=False, collate_fn=collate)
test_loader  = DataLoader(test_ds,  batch_size=args.batch_size,
shuffle=False, collate_fn=collate)

# 模型
model = SentimentLSTM(
    vocab_size=len(vocab),
    embedding_dim=args.embedding_dim,
    hidden_dim=args.hidden_dim,
    output_dim=1,
    n_layers=args.layers,
    dropout=args.dropout,
).to(device)
print(model)

# 训练
train(model, train_loader, valid_loader, device, epochs=args.epochs,
lr=args.lr, clip=1.0)

# 测试集评估 (使用最佳权重)
criterion = nn.BCEWithLogitsLoss().to(device)
if os.path.exists("best_model.pt"):
    model.load_state_dict(torch.load("best_model.pt", map_location=
device))
    print("[Info] 已加载最佳模型参数进行测试评估。")
test_loss, test_acc = evaluate(model, test_loader, device, criterion)
print(f"[Test] loss={test_loss:.4f}, acc={test_acc:.4f}")

# 示例预测
pos_review = "This movie was fantastic! I really enjoyed it."
neg_review = "The film was terrible and boring."
p1 = predict_sentiment(model, pos_review, vocab, device)
```

```
    p2 = predict_sentiment(model, neg_review, vocab, device)
    print(f"[Demo] Positive review score: {p1:.4f}")
    print(f"[Demo] Negative review score: {p2:.4f}")
    print("[Done] 训练与推理完成。")


if __name__ == "__main__":
    main()
```

4 常见问题与解决

- **追踪信息过多**：用 `PS4='+ $LINENO: '` 精简并带上行号；仅在可疑片段前后插入 `set -x` 与 `set +x`。
- **`make` 报错 "missing separator"**：规则行必须以 **Tab** 开头；Windows 文本行尾导致异常时建议在 WSL/Unix 下编辑。
- **微基准失真**：使用 `timeit` 多次迭代，避免 I/O 干扰；与 `cProfile` 结合看热点函数。

5 心得体会

调试优先考虑 "**尽可能小的可复现样例**"，配合 Shell 追踪、日志与交互式调试快速缩小问题范围；性能分析先**测量**再优化，`timeit`/`/usr/bin/time`/`cProfile` 能给出可靠依据。元编程理念强调 "**避免重复**"，用模板、规则与构建系统把一次性的手工命令固化下来，长期收益显著。PyTorch 入门只需掌握张量与自动求导，最小可用训练闭环能帮助理解更复杂的模型。

6 参考资料

- Missing Semester（中文）：调试及性能分析：https://missing-semester-cn.github.io/2020/debugging-profiling/
- Missing Semester（中文）：元编程：https://missing-semester-cn.github.io/2020/metaprogramming/
- 菜鸟教程：PyTorch 入门：https://www.runoob.com/pytorch/pytorch-tutorial.html

7 github 地址

https://github.com/tingol666/git2025q2.git