In [1]:
```
import os
import re
import time
import torch
```

```
d:\NLP\A2_AIT\.venv\Lib\site-packages\torch\_subclasses\functional_tensor.py:283:
UserWarning: Failed to initialize NumPy: No module named 'numpy' (Triggered inter
nally at C:\actions-runner\_work\pytorch\pytorch\pytorch\torch\csrc\utils\tensor_
numpy.cpp:84.)
  cpu = _conversion_method_template(device=torch.device("cpu"))
```

Task 1. Dataset Acquisition

In [ ]:
```
from pathlib import Path

RAW_DIR = Path("data_raw/harry_potter_books")
assert RAW_DIR.exists(), f"Folder not found: {RAW_DIR}"

txt_files = sorted(RAW_DIR.glob("*.txt"))
print("Found files:", len(txt_files))
for p in txt_files:
    print("-", p.name)
```

```
Found files: 3
- 01 Harry Potter and the Sorcerers Stone.txt
- 02 Harry Potter and the Chamber of Secrets.txt
- 03 Harry Potter and the Prisoner of Azkaban.txt
```

In [3]:
```
import re

def clean_book_text(text: str) -> str:
    text = text.replace("\r\n", "\n").replace("\r", "\n")
    # collapse excessive blank lines
    text = re.sub(r"\n{3,}", "\n\n", text)
    # remove trailing spaces on lines
    text = "\n".join([line.rstrip() for line in text.split("\n")])
    return text.strip()

books = []
for fp in txt_files:
    t = fp.read_text(encoding="utf-8", errors="ignore")
    t = clean_book_text(t)
    books.append(t)

print("Loaded books:", len(books))
print("Example preview:\n", books[0][:500])
```

```
Loaded books: 3
Example preview:
 M r. and Mrs. Dursley, of number four, Privet Drive, were proud to say that they
were perfectly normal, thank you very much. They were the last people you'd expec
t to be involved in anything strange or mysterious, because they just didn't hold
with such nonsense.

Mr. Dursley was the director of a firm called Grunnings, which made drills. He wa
s a big, beefy man with hardly any neck, although he did have a very large mustac
he. Mrs. Dursley was thin and blonde and had nearly twice the usual amoun
```

In [4]:
```
DATA_DIR = Path("data")
DATA_DIR.mkdir(exist_ok=True)

corpus_text = "\n\n".join(books)
CORPUS_PATH = DATA_DIR / "hp_kaggle_corpus.txt"
CORPUS_PATH.write_text(corpus_text, encoding="utf-8")

print("Saved:", CORPUS_PATH)
print("Chars:", len(corpus_text))
print("Tokens (whitespace):", len(corpus_text.split()))
```

```
Saved: data\hp_kaggle_corpus.txt
Chars: 1556494
Tokens (whitespace): 274252
```

Task 2. Model Training

Tokenize

In [5]:
```
def simple_tokenize(text: str):
    text = re.sub(r"([.,!?;:()\"'])", r" \1 ", text)
    text = re.sub(r"\s+", " ", text)
    return text.strip().split(" ")

text = CORPUS_PATH.read_text(encoding="utf-8", errors="ignore")
tokens = simple_tokenize(text)

print("Num tokens:", len(tokens))
print("First 40 tokens:", tokens[:40])
```

```
Num tokens: 326052
First 40 tokens: ['M', 'r', '.', 'and', 'Mrs', '.', 'Dursley', ',', 'of', 'numbe
r', 'four', ',', 'Privet', 'Drive', ',', 'were', 'proud', 'to', 'say', 'that', 't
hey', 'were', 'perfectly', 'normal', ',', 'thank', 'you', 'very', 'much', '.', 'T
hey', 'were', 'the', 'last', 'people', 'you’d', 'expect', 'to', 'be', 'involved']
```

Build vocab

In [6]:
```
from collections import Counter

PAD = "<pad>"
UNK = "<unk>"

def build_vocab(tokens, min_freq=2):
    counts = Counter(tokens)
    itos = [PAD, UNK]
    for tok, c in counts.most_common():
        if c >= min_freq and tok not in (PAD, UNK):
            itos.append(tok)
    stoi = {tok: i for i, tok in enumerate(itos)}
    return stoi, itos, counts

min_freq = 2
stoi, itos, counts = build_vocab(tokens, min_freq=min_freq)
pad_idx = stoi[PAD]
unk_idx = stoi[UNK]
vocab_size = len(itos)
```

```
print("Vocab size:", vocab_size)
print("Most common:", counts.most_common(10))
```

```
Vocab size: 8705
Most common: [(',', 20667), ('.', 17591), ('the', 11793), ('to', 6467), ('and', 6
304), ('a', 5302), ('of', 4923), ('”', 4571), ('Harry', 4467), ('was', 4109)]
```

Numericalize

In [7]:
```
def numericalize(tokens, stoi, unk_idx):
    return [stoi.get(tok, unk_idx) for tok in tokens]

ids = numericalize(tokens, stoi, unk_idx)
print("First 30 ids:", ids[:30])
```

```
First 30 ids: [2540, 1, 3, 6, 222, 3, 599, 2, 8, 904, 625, 2, 1055, 1056, 2, 39,
2115, 5, 194, 23, 35, 39, 1556, 1319, 2, 2022, 19, 73, 196, 3]
```

Sequences

In [8]:
```
import torch

seq_len = 40

def make_sequences(ids, seq_len):
    n = (len(ids) - 1) // seq_len * seq_len
    x = torch.tensor(ids[:n], dtype=torch.long)
    y = torch.tensor(ids[1:n+1], dtype=torch.long)
    x = x.view(-1, seq_len)
    y = y.view(-1, seq_len)
    return x, y

X, Y = make_sequences(ids, seq_len)
print("X:", X.shape, "Y:", Y.shape)
```

```
X: torch.Size([8151, 40]) Y: torch.Size([8151, 40])
```

DataLoaders

In [9]:
```
from torch.utils.data import TensorDataset, DataLoader

def make_loaders(X, Y, batch_size=64, train_ratio=0.9):
    n = X.size(0)
    perm = torch.randperm(n)
    X, Y = X[perm], Y[perm]
    n_train = int(n * train_ratio)
    X_train, Y_train = X[:n_train], Y[:n_train]
    X_val, Y_val = X[n_train:], Y[n_train:]
    train_loader = DataLoader(TensorDataset(X_train, Y_train), batch_size=batch_
    val_loader = DataLoader(TensorDataset(X_val, Y_val), batch_size=batch_size,
    return train_loader, val_loader

train_loader, val_loader = make_loaders(X, Y, batch_size=64)
print("Train batches:", len(train_loader))
print("Val batches:", len(val_loader))
```

```
Train batches: 115
Val batches: 13
```

In [ ]:
```
import math
from torch import nn
```

```
from torch.nn import functional as F

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device
```

Out[ ]:
```
device(type='cpu')
```

Hyperparameters

In [ ]:
```
embed_dim  = 128
hidden_dim = 256
num_layers = 1
dropout    = 0.4

learning_rate = 1e-3
num_epochs    = 45
grad_clip     = 1.0
```

Define Model

In [ ]:
```
class LanguageModelLSTM(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim, num_layers, dropout, p
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=pad_idx

        self.lstm = nn.LSTM(
            input_size=embed_dim,
            hidden_size=hidden_dim,
            num_layers=num_layers,
            batch_first=True,
            dropout=dropout if num_layers > 1 else 0.0
        )

        self.fc = nn.Linear(hidden_dim, vocab_size)

    def forward(self, x, hidden=None):
        """
        x: [B, T] integer token ids
        returns:
          logits: [B, T, V]
          hidden: (h_n, c_n)
        """
        emb = self.embedding(x)
        out, hidden = self.lstm(emb, hidden)
        logits = self.fc(out)
        return logits, hidden
```

Initialize model, loss, optimizer

In [13]:
```
model = LanguageModelLSTM(
    vocab_size=vocab_size,
    embed_dim=embed_dim,
    hidden_dim=hidden_dim,
    num_layers=num_layers,
    dropout=dropout,
    pad_idx=pad_idx
).to(device)
```

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

model
```

Out[13]:
```
LanguageModelLSTM(
  (embedding): Embedding(8705, 128, padding_idx=0)
  (lstm): LSTM(128, 256, batch_first=True)
  (fc): Linear(in_features=256, out_features=8705, bias=True)
)
```

In [ ]:
```
def run_epoch(model, data_loader, train=True):
    if train:
        model.train()
    else:
        model.eval()

    total_loss = 0.0
    total_tokens = 0

    for xb, yb in data_loader:
        xb = xb.to(device)
        yb = yb.to(device)

        logits, _ = model(xb)

        loss = criterion(
            logits.view(-1, logits.size(-1)),
            yb.view(-1)
        )

        if train:
            optimizer.zero_grad()
            loss.backward()
            nn.utils.clip_grad_norm_(model.parameters(), grad_clip)
            optimizer.step()

        total_loss += loss.item() * xb.numel()
        total_tokens += xb.numel()

    avg_loss = total_loss / total_tokens
    ppl = math.exp(avg_loss) if avg_loss < 20 else float("inf")
    return avg_loss, ppl
```

Training Loop

In [ ]:
```
best_val_loss = float("inf")
best_epoch = 0

os.makedirs("artifacts", exist_ok=True)

for epoch in range(1, num_epochs + 1):
    train_loss, train_ppl = run_epoch(model, train_loader, train=True)
    val_loss, val_ppl = run_epoch(model, val_loader, train=False)

    print(
        f"Epoch {epoch:02d} | "
        f"Train Loss: {train_loss:.4f}, PPL: {train_ppl:.2f} | "
        f"Val Loss: {val_loss:.4f}, PPL: {val_ppl:.2f}"
```

```
    )

    if val_loss < best_val_loss:# save best model based on validation loss
        best_val_loss = val_loss
        best_epoch = epoch
        torch.save(
            {
                "model_state_dict": model.state_dict(),
                "stoi": stoi,
                "itos": itos,
                "pad_idx": pad_idx,
                "unk_idx": unk_idx,
                "vocab_size": vocab_size,
                "embed_dim": embed_dim,
                "hidden_dim": hidden_dim,
                "num_layers": num_layers,
                "dropout": dropout,
                "seq_len": seq_len
            },
            "artifacts/lm_lstm_best.pt"
        )

print(f"Best epoch: {best_epoch}, best val loss: {best_val_loss:.4f}")
```

```
Epoch 01 | Train Loss: 6.7126, PPL: 822.67 | Val Loss: 6.1423, PPL: 465.14
Epoch 02 | Train Loss: 5.8888, PPL: 360.96 | Val Loss: 5.7221, PPL: 305.56
Epoch 03 | Train Loss: 5.5252, PPL: 250.93 | Val Loss: 5.4634, PPL: 235.90
Epoch 04 | Train Loss: 5.2760, PPL: 195.59 | Val Loss: 5.2901, PPL: 198.36
Epoch 05 | Train Loss: 5.0969, PPL: 163.52 | Val Loss: 5.1707, PPL: 176.04
Epoch 06 | Train Loss: 4.9580, PPL: 142.31 | Val Loss: 5.0846, PPL: 161.51
Epoch 07 | Train Loss: 4.8441, PPL: 126.99 | Val Loss: 5.0177, PPL: 151.07
Epoch 08 | Train Loss: 4.7455, PPL: 115.06 | Val Loss: 4.9619, PPL: 142.86
Epoch 09 | Train Loss: 4.6587, PPL: 105.50 | Val Loss: 4.9178, PPL: 136.71
Epoch 10 | Train Loss: 4.5812, PPL: 97.63 | Val Loss: 4.8824, PPL: 131.95
Epoch 11 | Train Loss: 4.5098, PPL: 90.90 | Val Loss: 4.8542, PPL: 128.28
Epoch 12 | Train Loss: 4.4442, PPL: 85.13 | Val Loss: 4.8261, PPL: 124.73
Epoch 13 | Train Loss: 4.3835, PPL: 80.12 | Val Loss: 4.8059, PPL: 122.23
Epoch 14 | Train Loss: 4.3256, PPL: 75.61 | Val Loss: 4.7877, PPL: 120.02
Epoch 15 | Train Loss: 4.2709, PPL: 71.59 | Val Loss: 4.7763, PPL: 118.66
Epoch 16 | Train Loss: 4.2195, PPL: 68.00 | Val Loss: 4.7626, PPL: 117.04
Epoch 17 | Train Loss: 4.1702, PPL: 64.73 | Val Loss: 4.7579, PPL: 116.50
Epoch 18 | Train Loss: 4.1224, PPL: 61.71 | Val Loss: 4.7489, PPL: 115.45
Epoch 19 | Train Loss: 4.0765, PPL: 58.94 | Val Loss: 4.7462, PPL: 115.14
Epoch 20 | Train Loss: 4.0322, PPL: 56.38 | Val Loss: 4.7437, PPL: 114.86
Epoch 21 | Train Loss: 3.9885, PPL: 53.97 | Val Loss: 4.7450, PPL: 115.01
Epoch 22 | Train Loss: 3.9453, PPL: 51.69 | Val Loss: 4.7433, PPL: 114.81
Epoch 23 | Train Loss: 3.9034, PPL: 49.57 | Val Loss: 4.7451, PPL: 115.02
Epoch 24 | Train Loss: 3.8631, PPL: 47.61 | Val Loss: 4.7472, PPL: 115.26
Epoch 25 | Train Loss: 3.8233, PPL: 45.75 | Val Loss: 4.7527, PPL: 115.89
Epoch 26 | Train Loss: 3.7845, PPL: 44.01 | Val Loss: 4.7587, PPL: 116.59
Epoch 27 | Train Loss: 3.7461, PPL: 42.35 | Val Loss: 4.7651, PPL: 117.34
Epoch 28 | Train Loss: 3.7074, PPL: 40.75 | Val Loss: 4.7727, PPL: 118.24
Epoch 29 | Train Loss: 3.6700, PPL: 39.25 | Val Loss: 4.7790, PPL: 118.98
Epoch 30 | Train Loss: 3.6330, PPL: 37.83 | Val Loss: 4.7896, PPL: 120.25
Epoch 31 | Train Loss: 3.5969, PPL: 36.49 | Val Loss: 4.7982, PPL: 121.29
Epoch 32 | Train Loss: 3.5598, PPL: 35.16 | Val Loss: 4.8100, PPL: 122.73
Epoch 33 | Train Loss: 3.5250, PPL: 33.95 | Val Loss: 4.8198, PPL: 123.94
Epoch 34 | Train Loss: 3.4899, PPL: 32.78 | Val Loss: 4.8321, PPL: 125.47
Epoch 35 | Train Loss: 3.4549, PPL: 31.65 | Val Loss: 4.8460, PPL: 127.24
Epoch 36 | Train Loss: 3.4208, PPL: 30.59 | Val Loss: 4.8555, PPL: 128.44
Epoch 37 | Train Loss: 3.3872, PPL: 29.58 | Val Loss: 4.8687, PPL: 130.16
Epoch 38 | Train Loss: 3.3530, PPL: 28.59 | Val Loss: 4.8837, PPL: 132.12
Epoch 39 | Train Loss: 3.3196, PPL: 27.65 | Val Loss: 4.8987, PPL: 134.12
Epoch 40 | Train Loss: 3.2857, PPL: 26.73 | Val Loss: 4.9150, PPL: 136.31
Epoch 41 | Train Loss: 3.2523, PPL: 25.85 | Val Loss: 4.9292, PPL: 138.27
Epoch 42 | Train Loss: 3.2199, PPL: 25.03 | Val Loss: 4.9447, PPL: 140.43
Epoch 43 | Train Loss: 3.1886, PPL: 24.25 | Val Loss: 4.9634, PPL: 143.08
Epoch 44 | Train Loss: 3.1560, PPL: 23.48 | Val Loss: 4.9823, PPL: 145.81
Epoch 45 | Train Loss: 3.1236, PPL: 22.73 | Val Loss: 4.9961, PPL: 147.83
Best epoch: 22, best val loss: 4.7433
```

Validation loss decreased steadily during the early training phase and reached its minimum at epoch 22. After this point, training loss continued to decrease while validation loss began to increase, indicating the onset of overfitting. Therefore, the final model was selected based on the lowest validation loss using an early stopping criterion rather than the last training epoch.

Task 3. Text Generation - Web Application Development

In [ ]:
```
import torch
from torch import nn
from torch.nn import functional as F
```

```
import re

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

ckpt = torch.load("artifacts/lm_lstm_best.pt", map_location=device)

stoi = ckpt["stoi"]
itos = ckpt["itos"]
pad_idx = ckpt["pad_idx"]
unk_idx = ckpt["unk_idx"]
vocab_size = ckpt["vocab_size"]

embed_dim  = ckpt["embed_dim"]
hidden_dim = ckpt["hidden_dim"]
num_layers = ckpt["num_layers"]
dropout    = ckpt["dropout"]
seq_len    = ckpt["seq_len"]

class LanguageModelLSTM(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim, num_layers, dropout, p
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=pad_idx
        self.lstm = nn.LSTM(
            input_size=embed_dim,
            hidden_size=hidden_dim,
            num_layers=num_layers,
            batch_first=True,
            dropout=dropout if num_layers > 1 else 0.0
        )
        self.fc = nn.Linear(hidden_dim, vocab_size)

    def forward(self, x, hidden=None):
        emb = self.embedding(x)
        out, hidden = self.lstm(emb, hidden)
        logits = self.fc(out)
        return logits, hidden

model = LanguageModelLSTM(vocab_size, embed_dim, hidden_dim, num_layers, dropout
model.load_state_dict(ckpt["model_state_dict"])
model.eval()

print("Loaded model ✅", "device:", device)
```

```
Loaded model ✅ device: cpu
```

In [23]:
```
def simple_tokenize(text: str):
    text = re.sub(r"([.,!?;:()\"'])", r" \1 ", text)
    text = re.sub(r"\s+", " ", text)
    return text.strip().split(" ") if text.strip() else []
```

In [ ]:
```
def top_k_filter(logits: torch.Tensor, k: int) -> torch.Tensor:
    if k is None or k <= 0 or k >= logits.numel():
        return logits
    values, idx = torch.topk(logits, k)
    filtered = torch.full_like(logits, float("-inf"))
    filtered[idx] = logits[idx]
    return filtered

@torch.no_grad()
def generate_text(prompt: str, max_new_tokens: int = 60, temperature: float = 1.
```

```
    model.eval()

    prompt_tokens = simple_tokenize(prompt)
    prompt_ids = [stoi.get(t, unk_idx) for t in prompt_tokens]
    if len(prompt_ids) == 0:
        prompt_ids = [unk_idx]

    x = torch.tensor(prompt_ids, dtype=torch.long, device=device).unsqueeze(0)
    logits, hidden = model(x, None)

    generated = list(prompt_ids)

    for _ in range(max_new_tokens):
        next_logits = logits[0, -1]
        next_logits = next_logits / max(temperature, 1e-6)
        next_logits = top_k_filter(next_logits, top_k)

        probs = F.softmax(next_logits, dim=-1)
        next_id = torch.multinomial(probs, num_samples=1).item()

        generated.append(next_id)

        x_next = torch.tensor([[next_id]], dtype=torch.long, device=device)
        logits, hidden = model(x_next, hidden)

    return " ".join(itos[i] if i < len(itos) else "<unk>" for i in generated)
```

In [21]:
```
print(generate_text("Harry Potter is", max_new_tokens=60, temperature=1.0, top_k
```

```
Harry Potter is . We’re going to be getting into , while I’d find out of all this
. They told him what the most is at Hogwarts School of wizards <unk> you coming .
…” Dumbledore came to Harry , who was holding a Slytherin table before he’d gone
over , and he had never heard a moment , the Snitch he
```

Github Link https://github.com/Thiri203/NLP_A2.git