Giải thích chi tiết từng lệnh trong code

1. File `utils.py` - Các hàm tiện ích

Import statements

python

```python
import os          # Thao tác với hệ điều hành (đường dẫn, file, thư mục)
import torch       # Thư viện deep learning PyTorch
import json        # Xử lý dữ liệu JSON
```

Hàm `load_config()`

python

```python
def load_config():
    # Tạo đường dẫn đến file config.json
    config_path = os.path.join(os.path.dirname(__file__), "..", "config.json")
    # os.path.dirname(__file__): Lấy thư mục chứa file hiện tại
    # ".." : Đi lên thư mục cha
    # os.path.join(): Nối các phần đường dẫn một cách an toàn

    # Cấu hình mặc định
    default_config = {"project_root": os.path.abspath(os.path.join(os.path.dirname(__file__), '
    # os.path.abspath(): Chuyển đường dẫn tương đối thành đường dẫn tuyệt đối

    try:
        with open(config_path, "r", encoding="utf-8") as f:
            # Mở file config.json với encoding UTF-8
            return {**default_config, **json.load(f)}
            # {**dict1, **dict2}: Gộp 2 dictionary, dict2 sẽ ghi đè dict1 nếu có key trùng
            # json.load(f): Đọc và parse JSON từ file
    except FileNotFoundError:
        # Nếu không tìm thấy file config.json
        print(f"⚠️ Không tìm thấy config.json, sử dụng giá trị mặc định: {default_config}")
        return default_config
```

Hàm `save_model()` và `load_model()`

python

```python
def save_model(model, path):
    torch.save(model.state_dict(), path)
    # model.state_dict(): Lấy tất cả tham số (weights, biases) của mô hình
    # torch.save(): Lưu object Python vào file

def load_model(path):
    return torch.load(path)
    # torch.load(): Tải object Python từ file
```

Hàm `ensure_dir()`

python

```python
def ensure_dir(directory):
    os.makedirs(directory, exist_ok=True)
    # os.makedirs(): Tạo thư mục (và các thư mục cha nếu cần)
    # exist_ok=True: Không báo lỗi nếu thư mục đã tồn tại
```

2. File `data_augmentation.py` - Tăng cường dữ liệu

Import statements

python

```python
import pandas as pd                                          # Xử lý dữ liệu dạng bảng
from transformers import MarianMTModel, MarianTokenizer # Mô hình dịch máy (không sử dụng trong
import random                                                # Tạo số ngẫu nhiên
import py_vncorenlp                                         # Thư viện xử lý ngôn ngữ tiếng Việt
import torch                                                 # PyTorch
import os                                                    # Thao tác hệ điều hành
```

Khởi tạo VnCoreNLP

python

```python
py_vncorenlp.download_model(save_dir="C:/VnCoreNLP")
# Tải mô hình VnCoreNLP về máy (chỉ chạy lần đầu)

annotator = py_vncorenlp.VnCoreNLP(annotators=["wseg"], save_dir="C:/VnCoreNLP")
# Khởi tạo annotator với chức năng tách từ (word segmentation)
# annotators=["wseg"]: Chỉ sử dụng tách từ
```

Từ điển đồng nghĩa

python

```python
synonyms_dict = {
    "đẹp": ["xinh", "lộng lẫy", "mỹ miều"],
    "tuyệt vời": ["xuất sắc", "hoàn hảo", "tuyệt diệu"],
    # Dictionary mapping từ gốc -> danh sách từ đồng nghĩa
}
```

Hàm `synonym_replacement()`

python

```python
def synonym_replacement(comment):
    segmented_text = annotator.word_segment(comment)
    # annotator.word_segment(): Tách từ trong câu tiếng Việt
    # VD: "Tôi thích ăn phở" -> ["Tôi", "thích", "ăn", "phở"]

    if segmented_text:
        words = segmented_text[0].split()
        # segmented_text[0]: Lấy kết quả đầu tiên
        # .split(): Tách chuỗi thành list các từ
    else:
        words = comment.split()
        # Nếu tách từ thất bại, dùng cách tách thông thường

    new_words = words.copy()
    # .copy(): Tạo bản sao của list để không thay đổi list gốc

    for i, word in enumerate(words):
        # enumerate(): Trả về cả index và value
        # VD: enumerate(["a", "b"]) -> [(0, "a"), (1, "b")]

        if word in synonyms_dict and random.random() < 0.3:
            # word in synonyms_dict: Kiểm tra từ có trong từ điển không
            # random.random(): Tạo số thực ngẫu nhiên từ 0.0 đến 1.0
            # < 0.3: 30% cơ hội thay thế

            new_words[i] = random.choice(synonyms_dict[word])
            # random.choice(): Chọn ngẫu nhiên 1 phần tử từ list

    return " ".join(new_words)
    # " ".join(): Nối các từ thành chuỗi, cách nhau bởi dấu cách
```

Hàm `augment_data()`

python

```python
def augment_data(data):
    augmented_data = []

    for _, row in data.iterrows():
        # data.iterrows(): Duyệt qua từng dòng của DataFrame
        # Trả về (index, Series), ta chỉ quan tâm Series nên dùng _

        comment = row["comment"]  # Lấy giá trị cột "comment"
        label = row["label"]      # Lấy giá trị cột "label"
        rate = row["rate"]        # Lấy giá trị cột "rate"

        augmented_data.extend([
            # .extend(): Thêm nhiều phần tử vào list
            {"comment": comment, "label": label, "rate": rate},                          # Dữ
            {"comment": synonym_replacement(comment), "label": label, "rate": rate},     # Dữ
        ])

    return pd.DataFrame(augmented_data)
    # pd.DataFrame(): Tạo DataFrame từ list các dictionary
```

Phần chạy chính

python

```python
if __name__ == "__main__":
    # Chỉ chạy khi file được execute trực tiếp, không chạy khi import

    try:
        project_root = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
        # Lấy đường dẫn tuyệt đối đến thư mục gốc dự án

        input_path = os.path.join(project_root, "data", "raw", "test_5k.csv")
        output_dir = os.path.join(project_root, "data", "processed")
        output_path = os.path.join(output_dir, "augmented_test_2k.csv")
        # Tạo các đường dẫn file input và output

        print(f"📄 Đang đọc file từ: {input_path}")
        if not os.path.exists(input_path):
            # os.path.exists(): Kiểm tra file/thư mục có tồn tại không
            raise FileNotFoundError(f"Không tìm thấy file: {input_path}")
            # raise: Ném exception để dừng chương trình

        data = pd.read_csv(input_path, usecols=["comment", "label", "rate"], on_bad_lines='skip
        # pd.read_csv(): Đọc file CSV thành DataFrame
        # usecols: Chỉ đọc các cột cần thiết
        # on_bad_lines='skip': Bỏ qua các dòng bị lỗi format

        required_columns = {"comment", "label", "rate"}
        if not required_columns.issubset(data.columns):
            # .issubset(): Kiểm tra tất cả phần tử có trong tập khác không
            # data.columns: Danh sách tên cột của DataFrame
            raise ValueError("❌ File CSV phải có đầy đủ các cột: comment, label, rate")

        augmented_data = augment_data(data)
        # Gọi hàm tăng cường dữ liệu

        os.makedirs(output_dir, exist_ok=True)
        # Tạo thư mục output nếu chưa có

        augmented_data.to_csv(output_path, index=False)
        # .to_csv(): Lưu DataFrame thành file CSV
        # index=False: Không lưu chỉ số dòng

    except Exception as e:
        # Bắt tất cả các exception khác
        print(f"❌ Lỗi: {e}")
```

3. File `train_phobert_classifier.py` - Huấn luyện mô hình

Import statements

python

```python
import torch                                                    # PyTorch
import torch.nn as nn                                           # Neural network modules
from transformers import AutoModelForSequenceClassification, AutoTokenizer  # Hugging Face trar
from torch.utils.data import Dataset, DataLoader                # Xử lý dữ liệu
from sklearn.metrics import accuracy_score                      # Tính độ chính xác
import pandas as pd                                             # Xử lý dữ liệu
import os                                                       # Thao tác hệ điều hành
from utils import load_config, save_model                       # Import từ file utils.py
```

Class `TextDataset`

python

```python
class TextDataset(Dataset):
    # Kế thừa từ torch.utils.data.Dataset

    def __init__(self, data, tokenizer, max_len):
        self.sentences = data["comment"].values
        # .values: Chuyển pandas Series thành numpy array

        self.labels = data["label"].map({"POS": 0, "NEG": 1, "NEU": 2}).fillna(0).values
        # .map(): Thay thế giá trị theo dictionary
        # .fillna(0): Thay thế giá trị NaN bằng 0

        self.tokenizer = tokenizer  # Lưu tokenizer
        self.max_len = max_len      # Độ dài tối đa của câu

    def __len__(self):
        return len(self.sentences)
        # Trả về số lượng mẫu trong dataset

    def __getitem__(self, idx):
        # Được gọi khi truy cập dataset[idx]

        sentence = self.sentences[idx]

        inputs = self.tokenizer.encode_plus(
            sentence,                   # Câu cần encode
            max_length=self.max_len,    # Độ dài tối đa
            padding="max_length",       # Đệm để đạt max_length
            truncation=True,            # Cắt bớt nếu quá dài
            return_tensors="pt"         # Trả về PyTorch tensors
        )
        # encode_plus(): Chuyển text thành tokens, sau đó thành numbers

        label = torch.tensor(int(self.labels[idx]), dtype=torch.long)
        # torch.tensor(): Tạo tensor từ data
        # dtype=torch.long: Kiểu số nguyên 64-bit

        return {
            "input_ids": inputs["input_ids"].squeeze(),        # Token IDs
            "attention_mask": inputs["attention_mask"].squeeze(), # Mask để phân biệt token thậ
            "label": label
        }
        # .squeeze(): Loại bỏ dimension có size = 1
```

Class `PhoBERTClassifier`

python

```python
class PhoBERTClassifier(nn.Module):
    # Kế thừa từ torch.nn.Module

    def __init__(self, phobert_model, num_labels=3):
        super().__init__()
        # super().__init__(): Gọi constructor của class cha

        self.phobert = phobert_model            # Mô hình PhoBERT
        self.dropout = nn.Dropout(0.1)         # Dropout layer với tỷ lệ 0.1
        # Dropout: Ngẫu nhiên "tắt" 10% neurons để tránh overfitting

    def forward(self, input_ids, attention_mask):
        # Hàm được gọi khi thực hiện forward pass

        outputs = self.phobert(input_ids, attention_mask=attention_mask)
        # Truyền input qua mô hình PhoBERT

        return outputs.logits
        # .logits: Điểm số thô cho mỗi class (chưa qua softmax)
```

Hàm `train_model()`

python

```python
def train_model(model, dataloader, optimizer, device, epochs=3):
    model.train()  # Chuyển mô hình sang chế độ huấn luyện

    for epoch in range(epochs):
        total_loss = 0

        for batch in dataloader:
            # Duyệt qua từng batch dữ liệu

            input_ids = batch["input_ids"].to(device)
            attention_mask = batch["attention_mask"].to(device)
            labels = batch["label"].to(device)
            # .to(device): Chuyển tensor sang GPU/CPU

            optimizer.zero_grad()
            # Xóa gradient từ iteration trước

            outputs = model(input_ids, attention_mask)
            # Forward pass: tính output

            loss = nn.CrossEntropyLoss()(outputs, labels)
            # Tính loss sử dụng Cross Entropy Loss

            loss.backward()
            # Backward pass: tính gradient

            optimizer.step()
            # Cập nhật parameters dựa trên gradient

            total_loss += loss.item()
            # .item(): Lấy giá trị scalar từ tensor

        print(f"Epoch {epoch+1}, Loss: {total_loss / len(dataloader)}")
        # In loss trung bình của epoch
```

Hàm `evaluate_model()`

python

```python
def evaluate_model(model, dataloader, device):
    model.eval()  # Chuyển mô hình sang chế độ đánh giá
    predictions, true_labels = [], []

    with torch.no_grad():
        # Không tính gradient để tiết kiệm memory và tăng tốc

        for batch in dataloader:
            input_ids = batch["input_ids"].to(device)
            attention_mask = batch["attention_mask"].to(device)
            labels = batch["label"].to(device)

            outputs = model(input_ids, attention_mask)
            # Forward pass

            preds = torch.argmax(outputs, dim=1)
            # argmax(): Lấy index của giá trị lớn nhất
            # dim=1: Theo chiều columns (mỗi sample)

            predictions.extend(preds.cpu().numpy())
            true_labels.extend(labels.cpu().numpy())
            # .cpu(): Chuyển tensor từ GPU về CPU
            # .numpy(): Chuyển tensor thành numpy array
            # .extend(): Thêm nhiều phần tử vào list

    accuracy = accuracy_score(true_labels, predictions)
    # Tính độ chính xác
    print(f"Accuracy: {accuracy}")
    return accuracy
```

Phần chạy chính

python

```python
if __name__ == "__main__":
    config = load_config()  # Tải cấu hình
    project_root = config["project_root"]

    data_path = os.path.join(project_root, "data", "processed", "augmented_test_2k.csv")
    model_path = os.path.join(project_root, "models", "phobert_best.pt")

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    # Sử dụng GPU nếu có, không thì dùng CPU

    data = pd.read_csv(data_path, on_bad_lines='skip')
    # Đọc dữ liệu đã tăng cường

    tokenizer = AutoTokenizer.from_pretrained("vinai/phobert-base")
    # Tải tokenizer của PhoBERT

    phobert = AutoModelForSequenceClassification.from_pretrained("vinai/phobert-base", num_labe
    # Tải mô hình PhoBERT với 3 lớp đầu ra

    dataset = TextDataset(data, tokenizer, max_len=128)
    dataloader = DataLoader(dataset, batch_size=16, shuffle=True)
    # Tạo DataLoader với batch_size=16, shuffle dữ liệu

    model = PhoBERTClassifier(phobert, num_labels=3)
    model = model.to(device)  # Chuyển mô hình sang device

    optimizer = torch.optim.Adam(model.parameters(), lr=2e-5)
    # Tạo optimizer Adam với learning rate = 0.00002

    train_model(model, dataloader, optimizer, device)      # Huấn luyện
    accuracy = evaluate_model(model, dataloader, device)   # Đánh giá
    save_model(model, model_path)                          # Lưu mô hình
```

4. File `inference.py` - Dự đoán

Hàm `predict()`

python

```python
def predict(model, tokenizer, text, device, max_len=128):
    model.eval()  # Chế độ đánh giá

    inputs = tokenizer.encode_plus(
        text,
        max_length=max_len,
        padding="max_length",
        truncation=True,
        return_tensors="pt"
    )
    # Encode văn bản giống như khi huấn luyện

    input_ids = inputs["input_ids"].to(device)
    attention_mask = inputs["attention_mask"].to(device)

    with torch.no_grad():
        outputs = model(input_ids, attention_mask)
        pred = torch.argmax(outputs, dim=1).cpu().item()
        # .item(): Chuyển tensor 1 phần tử thành số Python

    label_map = {0: "POS", 1: "NEG", 2: "NEU"}
    return label_map.get(pred, "unknown")
    # .get(): Lấy value từ dictionary, trả về "unknown" nếu không tìm thấy key
```

Phần chạy chính

python

```python
if __name__ == "__main__":
    config = load_config()
    project_root = config["project_root"]
    model_path = os.path.join(project_root, "models", "phobert_best.pt")
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    tokenizer = AutoTokenizer.from_pretrained("vinai/phobert-base")
    model = PhoBERTClassifier(AutoModelForSequenceClassification.from_pretrained("vinai/phobert

    model.load_state_dict(load_model(model_path))
    # .load_state_dict(): Tải trọng số đã lưu vào mô hình

    model = model.to(device)

    # Dự đoán mẫu
    test_text = "Câu này có tự nhiên không?"
    result = predict(model, tokenizer, test_text, device)
    print(f"📝 Văn bản: {test_text}")
    print(f"🔍 Dự đoán: {result}")

    # Dự đoán từ file (tuỳ chọn)
    input_path = os.path.join(project_root, "data", "raw", "test_5k.csv")
    if os.path.exists(input_path):
        data = pd.read_csv(input_path, usecols=["comment"])
        predictions = [predict(model, tokenizer, text, device) for text in data["comment"]]
        # List comprehension: tạo list bằng cách duyệt qua iterable

        data["prediction"] = predictions  # Thêm cột mới vào DataFrame
        output_path = os.path.join(project_root, "data", "processed", "predictions.csv")
        data.to_csv(output_path, index=False)
```

Tóm tắt các lệnh quan trọng

Lệnh	Chức năng
`torch.device()`	Chọn device (GPU/CPU)
`model.train()` / `model.eval()`	Chuyển chế độ huấn luyện/đánh giá
`torch.no_grad()`	Tắt tính gradient
`optimizer.zero_grad()`	Xóa gradient cũ
`loss.backward()`	Tính gradient
`optimizer.step()`	Cập nhật parameters
`torch.argmax()`	Lấy index của giá trị lớn nhất
`.to(device)`	Chuyển tensor sang device
`pd.read_csv()`	Đọc file CSV
`tokenizer.encode_plus()`	Chuyển text thành tokens
`AutoTokenizer.from_pretrained()`	Tải tokenizer đã train
`AutoModelForSequenceClassification.from_pretrained()`	Tải mô hình đã train