AI 텀프로젝트
시스템 동작 가이드라인

202000826

김연범

1. 텀프로젝트 A. Multimodal RAG

(0) 사전 준비 사항

– 평가 환경은 colab이라고 가정.

– 평가자 드라이브에 Movie_DB 파일 업로드(ChromaDB).

(https://drive.google.com/file/d/12_j7wd_skfTXIbgVq9QjutdoLQZTHr1Z/view?usp=sharing)

– mkdir 과정은 이미 돼 있다고 가정.

(1) 드라이브 마운트

0. 드라이브 마운트

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

: 평가자의 드라이브에 마운트 진행.

(2) 필요 라이브러리 설치 및 임포트

1. 필요 라이브러리 설치 및 임포트

```
!pip install -q torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cpu
!pip install -q transformers sentence-transformers gradio chromadb
!pip install -U bitsandbytes
```

```
from PIL import Image
import os
import io
import torch
import numpy as np
from transformers import CLIPProcessor, CLIPModel, AutoTokenizer, AutoModelForCausalLM
from sentence_transformers import SentenceTransformer
import gradio as gr
import chromadb
from chromadb.config import Settings
import json
```

: 필요한 라이브러리 설치 및 라이브러리 임포트 진행.

(3) 텍스트/이미지 임베딩 모델 임포트

2. 텍스트/이미지 임베딩 모델 임포트

```
# 텍스트 임베딩 모델
text_embedder = SentenceTransformer("BAAI/bge-m3")

# 이미지 임베딩 모델
clip_model_name = "openai/clip-vit-base-patch32"
clip_model = CLIPModel.from_pretrained(clip_model_name)
clip_processor = CLIPProcessor.from_pretrained(clip_model_name)
```

: 온라인 환경에서 텍스트/이미지 임베딩 모델 로드 진행.

(4) ChromaDB 불러오기

3. ChromaDB 불러오기

```
# ChromaDB 불러오기
client = chromadb.PersistentClient(path="drive/MyDrive/Movie_DB")

text_collection = client.get_collection("movie_text_embeddings")
image_collection = client.get_collection("movie_image_embeddings")
```

: 평가자의 드라이브 파일에 업로드된 ChromaDB 불러오기.

: 평가자의 드라이브 파일에 해당 벡터 데이터베이스가 존재해야 함.

: 드라이브 경로: drive/MyDrive/Movie_DB

(5) ChromaDB 데이터 search 함수 정의

4. ChromaDB 데이터 search 함수 정의

```
def search_with_metadata(query_text=None, query_image=None, k=3):
    if query_image:
        inputs = clip_processor(images=query_image, return_tensors="pt")
        with torch.no_grad():
            q_emb = clip_model.get_image_features(**inputs).cpu().numpy().tolist()

        out = image_collection.query(
            query_embeddings=q_emb,
            n_results=k
        )
    else:
        q_emb = text_embedder.encode(query_text).tolist()

        out = text_collection.query(
            query_embeddings=[q_emb],
            n_results=k
        )

    ids = out["ids"][0]
    metas = out["metadatas"][0]

    return {
        "ids": ids,
        "metadatas": metas,
    }
```

: 벡터 데이터베이스 내에서 search를 진행하는 함수 정의하기.

(6) Q&A 상호작용을 위한 LLM 모델 로드

5. Q&A 상호작용을 위한 LLM 모델 로드

```
model_name = "Qwen/Qwen2.5-3B-Instruct"

tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    load_in_8bit=True,
    device_map="auto",
)
```

: 온라인 환경에서, 챗봇 상호작용을 위한 LLM 모델 로드.

: 추론 모델로는 Qwen2.5-3B-Instruct 모델 선택.

(7) Q&A LLM 파이프라인 함수 정의

6. Q&A LLM 파이프라인 함수 정의

6-1. 질의용 전체 파이프라인 함수

```
def answer_query(user_text=None, user_image=None):

    # 1. 검색 결과 top-k row index 가져오기
    match_indices = search_with_metadata(query_text=user_text, query_image=user_image, k=1)

    # 2. 영화 정보 context 구성
    meta = match_indices["metadatas"][0]  # top-1 결과
    title = meta.get("title", "")
    genres = meta.get("genres", "")
    overview = meta.get("overview", "")
    popularity = meta.get("popularity", "")
    release_date = meta.get("release_date", "")
    budget = meta.get("budget", "")
    revenue = meta.get("revenue", "")
    runtime = meta.get("runtime", "")
    context = ""

    context = (
        f"title: {title}₩n"
        f"genres: {genres}₩n"
        f"overview: {overview}₩n"
        f"popularity: {popularity}₩n"
        f"release_date: {release_date}₩n"
        f"budget: {budget}₩n"
        f"revenue: {revenue}₩n"
        f"runtime: {runtime}₩n"
    )

    # 3. 사용자 입력 텍스트 구성
    user_prompt_text = user_text if user_text else "User upload the image."

      # 4. Qwen2용 messages 구성
      messages = [
          {
              "role": "system",
              "content": (
                  "You are a movie expert."
                  "Answer the user's questions in natural, smooth english only, using the [reference data] as your knowledge source. "
                  "The [reference data] contains movie information you should consult to find the most relevant details for your answer."
                  "If the movie the user is asking about does not have sufficiently similar information in the [reference data], using your pre
                  "Always summarize your answer in a single concise sentence."
              )
          },
          {
              "role": "user",
              "content": (
                  f"[Question]₩n{user_prompt_text}₩n₩n"
                  f"[reference data]₩n{context}₩n₩n"
                  "[Answer]"
              )
          }
      ]

      # 5. apply_chat_template로 모델 입력 생성
      inputs = tokenizer.apply_chat_template(
          messages,
        add_generation_prompt=True,
        tokenize=True,
        return_dict=True,
        return_tensors="pt",
      ).to(model.device)


      # 6. 생성
      outputs = model.generate(
          **inputs,
          max_new_tokens=64,
      )

      # 7. 모델 출력 디코딩
      generated = tokenizer.decode(
          outputs[0][inputs["input_ids"].shape[-1]:],
          skip_special_tokens=True
      )
      return generated.strip()
```

: Q&A를 위한 상호작용 파이프라인 실행.

(8) json 파일 생성용 추가 파이프라인

6-2. json 파일 생성용 추가 파이프라인

```
INPUT_DIR = "./data"
OUTPUT_DIR = "./release"

# json 파일 로드용 함수
def load_json(path):
    with open(path, "r", encoding="utf-8") as f:
        return json.load(f)

# json 파일 저장용 함수
def save_json(path, data):
    with open(path, "w", encoding="utf-8") as f:
        json.dump(data, f, ensure_ascii=False, indent=2)

########################################################################
FILENAME = "test.json" # 불러올 입력 데이터 파일 이름
FILENAME_SAVE = "202000826.test.json"
########################################################################

input_path = os.path.join(INPUT_DIR, FILENAME)
data = load_json(input_path)

output = [] # 저장할 Q&A 데이터 쌍

for item in data:
  # json 파일에서 질문 가져오기
  question = item['question']
  print(f"▶ Generating answer for: {question}")

  # 각 질문에 대한 대답 생성
  answer = answer_query(question)
  print(f"▶ Generated answer: {answer}")

  # json 파일로 저장하기 위한 포맷 설정
  output_data = {
      "question": question,
      "answer": answer
  }
  output.append(output_data)

output_path = os.path.join(OUTPUT_DIR, FILENAME_SAVE)
save_json(output_path, output)

print(f"✓ Saved: {output_path}")
```

: 평가자가 평가를 위한 json 파일 생성 파이프라인.

: 평가자의 평가용 데이터는 /data 경로의 'test.json' 파일 데이터라고 가정.

: 모델의 출력 결과는 터미널 및 /release 경로의 '202000826.test.json' 파일 데이터라고 가정.

(0) -> (8) 까지 순서대로 코드 실행.

(9) 별첨 1. Gradio 앱 실행 코드

별첨. Gradio 앱 실행

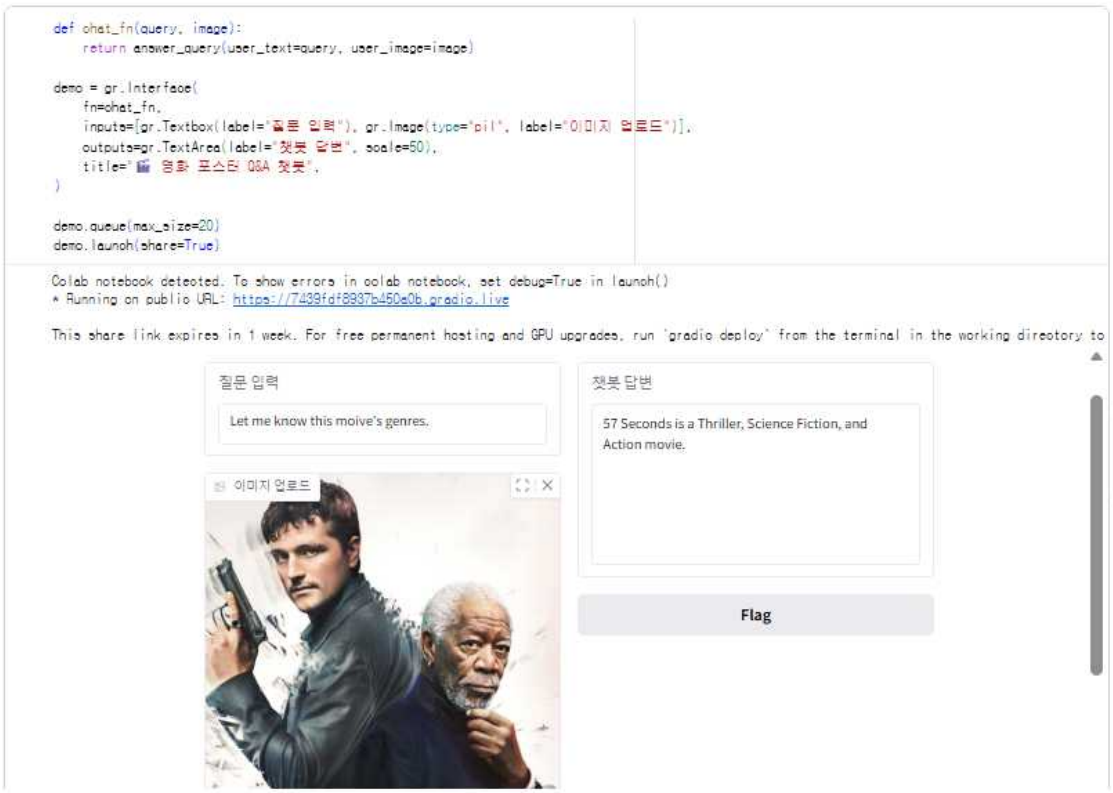
가능한 질의 내용: 제목, 장르, 줄거리, 인기 정도, 출시일, 예산, 수익, 상영시간

(1) 채팅으로만 질의할 경우

- 텍스트 칸에 영화 관련 질문 작성 후 send

(2) 이미지(포스터)와 함께 질의할 경우

- 이미지 업로드 후, 텍스트 칸에 질문 작성 후 send
- 예: 포스터 업로드 후, 'Let me know this movie's genres.'



: Gradio 앱 실행을 통한 인터페이스 상호작용 가능 코드.

: (i) 채팅으로만 질의 시, 채팅 후 submit.

: (ii) 포스터와 함께 질의 시, 채팅 + 포스터 업로드 후 submit.

(10) 별첨 2. ChromaDB 생성 코드

: ChromaDB 생성을 위해 사용했던 코드들 첨부.

2. 텀프로젝트 B.
Prompt based Sound Classification

(0) 사전 준비 사항

– 평가 환경은 colab이라고 가정.

– 평가자 드라이브에 모델이 저장된 pth 파일
(“audio_cnn_model.pth”) 업로드.

(https://drive.google.com/file/d/1ABctx9sa47ik9RUy7ZXpEhcrWhw4Wfjx/view?usp=sharing)

(압축 파일에 포함돼 있으므로, 이상이 있는 경우에만 위 경로를 통해 다운로드)

(1) 드라이브 마운트

0. 드라이브 마운트

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

: 평가자 드라이브에 마운트.

(2) 초기 환경 설정

1. 초기 환경 설정

1-1. 라이브러리 설치, 임포트

```
!pip install datasets librosa matplotlib seaborn scikit-learn torch torchaudio torchcodec
```

```
import torch.nn.functional as F
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from datasets import load_dataset, load_from_disk
import librosa
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from tqdm import tqdm
```

1-2. 중요 하이퍼파라미터 설정

```
SAMPLE_RATE = 22050
DURATION = 4 # 오디오 길이 통일 (4초)
N_MFCC = 40 # MFCC 개수
BATCH_SIZE = 32
EPOCHS = 50 # 학습 횟수
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# 클래스 이름 정의 (UrbanSound8K 10개 클래스)
classes = [
    "air_conditioner", "car_horn", "children_playing", "dog_bark",
    "drilling", "engine_idling", "gun_shot", "jackhammer",
    "siren", "street_music"
]

print(f"Using device: {device}")
```

```
Using device: cuda
```

: 필수 라이브러리 설치, 임포트 및 중요 하이퍼파라미터 설정.

: 하이퍼파라미터의 경우 훈련 이외에도, 평가에서도 활용되므로 꼭 실행.

(3) 오디오 데이터셋 전처리

2-1. 오디오 데이터셋 전처리 함수

```
class AudioDataset(Dataset):
    def __init__(self, data_list):
        self.data = data_list

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

    def __getitem__(self, idx):
        item = self.data[idx]
        # 오디오 처리
        audio_info = item['audio']
        audio = audio_info['array']
        sr = audio_info['sampling_rate']
        label = item['classID']

        # 전처리 (위의 로직과 동일하게 적용)
        if sr != SAMPLE_RATE:
            audio = librosa.resample(audio, orig_sr=sr, target_sr=SAMPLE_RATE)

        max_len = SAMPLE_RATE * DURATION
        if len(audio) < max_len:
            pad_width = max_len - len(audio)
            audio = np.pad(audio, (0, pad_width), mode='constant')
        else:
            audio = audio[:max_len]

        mfcc = librosa.feature.mfcc(y=audio, sr=SAMPLE_RATE, n_mfcc=N_MFCC)
        mfcc = librosa.power_to_db(mfcc)

        # 텐서 변환 (Channel, Height, Width) -> (1, n_mfcc, time_steps)
        return torch.tensor(mfcc, dtype=torch.float32).unsqueeze(0), torch.tensor(label, dtype=torch.long)
```

: 오디오 데이터셋 전처리 함수.

(4) 오디오 분류 모델 정의

3. 모델 정의 - Residual Block + CNN 기반 오디오 분류 모델

```
# A. Squeeze-and-Excitation (SE) Block 정의
class SEBlock(nn.Module):
    def __init__(self, channel, reduction=16):
        super(SEBlock, self).__init__()
        # 1. Squeeze: Global Average Pooling을 통해 채널 정보를 압축
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        # 2. Excitation: 두 개의 FC Layer로 채널별 중요도를 학습
        self.fc = nn.Sequential(
            nn.Linear(channel, channel // reduction, bias=False),
            nn.ReLU(inplace=True),
            nn.Linear(channel // reduction, channel, bias=False),
            nn.Sigmoid() # Sigmoid를 통해 0과 1 사이의 가중치 생성
        )

    def forward(self, x):
        b, c, _, _ = x.size()
        # Squeeze
        y = self.avg_pool(x).view(b, c)
        # Excitation
        y = self.fc(y).view(b, c, 1, 1)
        # Re-weighting: 입력 특징 맵에 가중치(y)를 곱함
        return x * y.expand_as(x)

# B. Residual Block 정의 (Conv + BN + ReLU + SE)
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, use_se=True):
        super(ResidualBlock, self).__init__()

        # 잔차 연결을 위한 Conv (입력 채널과 출력 채널이 다를 경우 사용)
```

: ResNet + SE-Net 기반의 오디오 분류 모델 정의.

(5) 저장된 모델 로드

6. 저장된 모델 불러오기

```
# 1. 모델 클래스 정의 (AudioCNN 클래스는 코드가 실행되는 파일에 반드시 존재해야 함!)
model = ResSEAudioCNN(num_classes=10).to(device)

# 2. 저장된 상태 사전(state_dict) 로드
# 모델이 저장된 파일명을 사용.
MODEL_PATH = "drive/MyDrive/audio_cnn_model.pth"
loaded_state_dict = torch.load(MODEL_PATH, weights_only=True, map_location=device) # 저장 시와 다른 장치(CPU/GPU)에서 로드할 경우 map_locatior

# 3. 모델 객체에 상태 사전 적용
model.load_state_dict(loaded_state_dict)

# 4. 모델을 평가 모드로 전환
model.eval()

print(f"Model loaded successfully from {MODEL_PATH} and ready for inference.")
```

: 이미 학습된 파라미터를 로컬 경로에서 불러와 적용.

: 파라미터가 저장된 로컬 경로는

“drive/MyDrive/audio_cnn_model.pth” 라고 가정.

(6) 추론 및 혼동행렬 생성 파이프라인

7. 추론 및 혼동행렬 생성 파이프라인

```
def evaluate_and_plot_cm(model, loader, device, class_names):
    y_pred = []
    y_true = []

    # 1. 예측 수행
    model.eval()  # 평가 모드
    print("Evaluating model on Test Set...")

    with torch.no_grad():
        # tqdm을 사용하여 진행 상황바 표시
        for inputs, labels in tqdm(loader, desc="Inference"):
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)

            # CPU로 이동 후 리스트에 저장
            y_pred.extend(predicted.cpu().numpy())
            y_true.extend(labels.cpu().numpy())

    # 2. 혼동행렬 계산
    cm = confusion_matrix(y_true, y_pred)

    # 3. 시각화 (Heatmap)
    plt.figure(figsize=(12, 10))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names,
                yticklabels=class_names)

    plt.xlabel('Predicted Label', fontsize=12, fontweight='bold')
    plt.ylabel('True Label', fontsize=12, fontweight='bold')
    plt.title('UrbanSound8K Confusion Matrix', fontsize=15)
    plt.xticks(rotation=45)
    plt.yticks(rotation=0)
    plt.tight_layout()
    plt.show()

    # 4. 클래스별 정확도 출력
    # 대각선 값(정답) / 해당 클래스의 총 개수(행의 합)
    class_accuracy = cm.diagonal() / cm.sum(axis=1)

    print("\n" + "="*30)
    print("Class-wise Accuracy")
    print("="*30)
    for i, acc in enumerate(class_accuracy):
        print(f"{class_names[i]:<20}: {acc*100:.2f}%")
    print("="*30)
```

: 학습된 모델을 추론시키고, 이를 바탕으로 혼동행렬을 생성해주는 파이프라인.

(7) 평가

8. Test 데이터(평가)

8-1. 테스트 데이터 전처리

(i) 가정 1. 테스트 데이터 역시, 훈련 데이터처럼 dataset 형태로 가져온다고 가정!

(i) 가정 2. 테스트 데이터 역시, 훈련 데이터처럼 'audio' 컬럼(array + sampling_rate)의 데이터와, 'classID' 컬럼의 정답 데이터가 존재한다고 가정!

(i) 가정 3. 테스트 데이터 역시, 훈련 데이터처럼 다음과 같은 클래스 이름과 'classID' 컬럼의 숫자가 순서대로 0부터 9까지 대응된다고 가정!

```
classes = [ "air_conditioner", "car_horn", "children_playing", "dog_bark", "drilling", "engine_idling",
"gun_shot", "jackhammer", "siren", "street_music" ]
```

```
# 1. dataset 형태로 데이터 가져오기
dataset = load_dataset("danavery/urbansound8K", split="train") # 데이터

# 2. dataset list 형태로 변환하기
dataset_list = list(dataset)

# 3. 데이터셋 전처리
test_data = AudioDataset(dataset_list)

# 4. data loader
loader = DataLoader(test_data, batch_size=BATCH_SIZE, shuffle=False)
```

: 평가자가 평가를 위해 실행하는 코드.

: 테스트 데이터는 훈련 데이터와 동일한 포맷을 가진다고 가정.

: 이후 밑의 혼동 매트릭스 출력 함수를 실행하여 추론&혼동 매트릭스 출력.

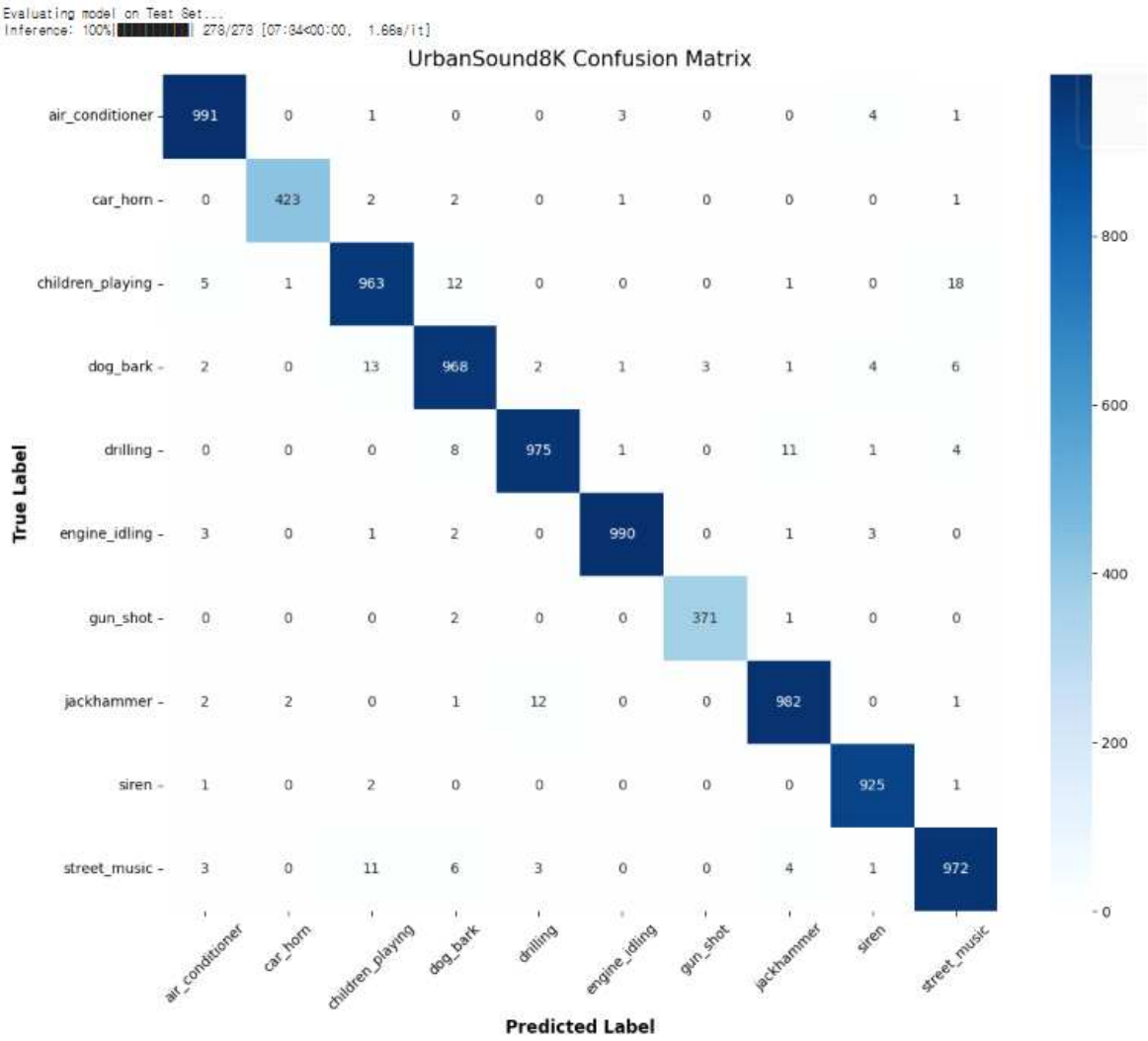
혼동 매트릭스 출력

```
evaluate_and_plot_cm(model, loader, device, classes)
```

문서에 적힌 코드들만 선택하여,

(0) -> (7) 까지 순서대로 코드 실행.

(8) 별첨.




```
--------------------------------------
Class-wise Accuracy
--------------------------------------
air_conditioner     : 99.10%
car_horn            : 98.60%
children_playing    : 96.30%
dog_bark            : 96.80%
drilling            : 97.50%
engine_idling       : 99.00%
gun_shot            : 99.20%
jackhammer          : 98.20%
siren               : 99.67%
street_music        : 97.20%
--------------------------------------
```

: 예시 (학습/평가용) 데이터 전체를 활용해 '(7)번－평가'를 진행한 결과물.