SW/AI 동아리 추가 보고서

30923 함태준

세부 프로그램 동작 (WEB)

사용자가 웹사이트에 접속하면 Flask 서버가 요청을 받고 index.html를 사용자에게 보낸다.

```
14      @app.route("/")
15      def main():
16          return render_template("index.html")
```

화면에 뽀로로와 대화할 수 있는 페이지가 나타난다.
사용자가 아래 input box에 글자를 입력하고 전송 버튼을 누르거나, 엔터 키를 누르면 지정된 함수를 실행한다.

```
37      sendBtn.addEventListener('click', sendMessage); // 전송 버튼 클릭 시 sendMessage 실행
38      userInput.addEventListener('keydown', (e) => {
39          if (e.key === 'Enter') sendMessage(); // 엔터키 누르면 sendMessage 실행
40      });
```

Javascript로 지정된 함수는 Flask 웹 서버에 fetch로 /chat 주소에 POST 요청을 보낸다. 이 때, input box의 값을 상수 변수 text에 저장하고 이를 FormData 객체에 넣어서 요청 보낼 때 같이 넣는다.

```
async function sendMessage() {
    const text = userInput.value.trim(); // 입력값 앞뒤 공백 제거
    userInput.value = ''; // 입력 필드 비우기

    if (text) {
        const formData = new FormData(); // 폼 데이터 객체 생성 (파일 업로드나 POST 요청에 사용)

        formData.append("message", text); // 폼 데이터에 'message' 필드로 사용자 입력 추가
        addMessage(text, 'user'); // 사용자 메시지를 화면에 추가

        const response = await fetch("/chat", { // 서버에 POST 요청 (비동기)
            method: "POST", // HTTP 메서드 지정
            body: formData, // 요청 본문에 폼 데이터 포함
        });

        if (response.ok) { // 응답이 성공적이면
            const json = await response.json(); // 응답을 JSON으로 파싱 (비동기)
            addMessage(json["response"], 'bot'); // 서버 응답 메시지를 화면에 추가
        }
    } else {
        return; // 입력값이 없으면 함수 종료
    }
}
```

웹 서버에서 /chat으로 요청을 받았을 때 요청받은 form 데이터에서 message를 불러오고, 이를 LLM 객체의 ask 메소드로 처리하여, JSON 방식으로 리턴한다.

```python
@app.route("/chat", methods=['POST'])
def chat():
    message = request.form.get("message")
    logger.info(message)

    answer = LLM.ask(message)

    return jsonify({"response": answer})
```

LLM 객체에서는 파인튜닝된 모델을 불러오고, ask가 호출되면 들어온 값을 인코드 하고, Tensor로 변환해 메시지를 생성한다.

```python
class Model:

    def __init__(self) -> None:
        pass

    def load_file(self, file):
        self.file = file
        self.tokenizer = AutoTokenizer.from_pretrained("./pororo_model")
        self.model = AutoModelForCausalLM.from_pretrained("./pororo_model")

    def ask(self, q):
        return self.model.generate(
            input_ids=torch.tensor([self.tokenizer.encode(q)]),
            max_length=512,
            num_return_sequences=1,
            do_sample=True,
            top_p=0.95,
            top_k=50
        )[0]
```

이렇게 생성된 메시지를 JSON 형식으로 보내면, Javascript에서 이를 받아 HTML에 값을 삽입하여 유저에게 보여준다. (Javascript를 활용한 HTML 동적 로딩)

```
function addMessage(content, className) {
    const msg = document.createElement('div');
    msg.classList.add('message', className);
    msg.textContent = content;
    chatContainer.appendChild(msg);
    chatContainer.scrollTop = chatContainer.scrollHeight;
}

```

세부 프로그램 동작 (finetuning)

파일을 전처리 하기 위해서 demucs 모델으로 파일의 배경음을 전부 제거해 목소리만 남게 한다.

```
        # 파일 로드
        wav, sr = torchaudio.load(file_path)

        # 모노 → 스테레오 변환 필요 시
        if wav.shape[0] == 1:
            wav = wav.repeat(2, 1) #1채널을 2번 복제하여 2채널로 변환함

        wav = wav.unsqueeze(0)
        #wav = wav.unsqueeze(0).to("cuda") #CUDA로 연산할때는 이거 쓰기

        # demucs 모델은 [batch, channels, sales] 형태로 입력을 받음
        #unsqueeze(0) 을 하면 기존 (2, samples) 형태가 (1, 2, samples)로 변환됨

        # 모델 적용
        with torch.no_grad(): #추론 모드에서 연산 수행 demucs 모델을 사전학습 모델이므로, gradient 계산이 필요 X
            sources = apply_model(model, wav, split=True, overlap=0.25, progress=True) #demucs 라이브러리의 apply_model 함수 사용

        # 분리된 소스 저장
        sources_names = model.sources #모델이 분리한 트랙 이름 리스트 가져옴
        for idx, name in enumerate(sources_names):
            output_path = os.path.join(output_dir, f"{i}_{name}.wav")
            torchaudio.save(output_path, sources[0, idx], sample_rate=sr)  #sources[0, idx]는 분리된 특정 트랙 오디오 tensor..(?)
            print(f"  → 저장됨: {output_path}")
```

남은 목소리 파일에서 pyannote 모델을 사용하여 화자를 구분하고 대사를 JSON 형식으로 저장한다.

```
def wav_to_json(CurrentVoiceFile):
    print(f"Processing {CurrentVoiceFile}...")

    # 전체 파일에 대해 diarization 수행
    diarization = pipeline(os.path.join("separated_outputs", CurrentVoiceFile))
    print("전체 diarization 결과:")
    print(diarization)

    with open(f"DiarizationResultsJson/{CurrentVoiceFile[0:-4]}.json", "w") as f:
        json.dump(diar_to_json(diarization), f) # Json 저장

```

JSON을 불러와서 얻은 대사를 토크나이즈 하고, 이를 모델에 학습시켜 저장한다.

```
# 4. 토큰화
tokenized = dataset.map(lambda x: tokenizer(x["text"], truncation=True, padding="max_length", max_length=512), batched=True)
tokenized.set_format("torch")

# 5. 학습 인자 설정
training_args = TrainingArguments(
    output_dir="./pororo_model",
    per_device_train_batch_size=2,
    num_train_epochs=3,
    save_strategy="epoch",
    logging_steps=10,
    learning_rate=2e-5,
    fp16=True,  # GPU가 있다면 True
)

# 6. Trainer 구성 및 학습
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized,
)
trainer.train()

# 7. 모델 저장
trainer.save_model("./pororo_model")
```

협업 내용

프로젝트에서 웹 서버, 클라이언트 제작과 데이터 전처리를 담당하였다.
Flask로 서버 프로그램을 개발하고, 이를 학습된 모델과 연결하여 클라이언트와 통신할 수 있도록 제작하였다.
HTML, CSS, Javascript로 클라이언트를 작성하였으며, fetch를 통해 서버와 POST 방식으로 통신할 수 있게 구성하였다.
추가로 2학년 후배, 이준혁에게 Javascript를 이용한 HTML 동적 로딩하는 법, HTML 기본 작성 및 원리에 대해 알려주는 멘토 역할도 수행했으며, 이 과정에서 이준혁이 작성한 HTML 동적 로딩 Javascript 코드를 리뷰하고 실제 프로젝트에 사용할 수 있도록 수정을 도왔다.
데이터 전처리에서는 팀원들이 수집한 wav 파일을 화자 구분 및 구분한 데이터를 슬라이싱하는 역할을 주로 맡아 다른 팀원들이 인공지능 모델을 학습시키는 데 큰 도움을 주었다.