
- 출처: LangChain 공식 문서, 조코딩의 랭체인으로 AI 에이전트 서비스 만들기
- 깃허브 저장소 출처: https://github.com/sw-woo/hanbit-langchain

---

01. `다국어 이메일 생성기 만들기`

- *출처: 위에 표기*

---

1) `교재 설명`

- **목표**
  - `Ollama` 기반의 `Llama 3.1 모델` + `로컬 환경`
  - `CTransformers` 이용한 `Local Llama 2` 모델 활용
  - **사용자가 선택한 언어에 맞춰 `이메일 주제`, `발신자`, `수신자 정보` 입력 → `자동으로 이메일 내용 생성` → 다국어 자동 이메일 생성기 구축하기**

- **핵심 키워드**
  - **`Ollama`**
  - **`Llama 3.1`**: *`Ollama` 통해 연결됨*
  - **`Local Llama 2`**: *`CTransformers` 라이브러리 통해 사용됨*
  - **`CTransformers`**

- **`LLaMa` 사용하는 이유**
  - **`Open source model`**: 오픈 가중치 모델 제공 → 대규모 호출 → 비용 부담 ↓
  - **`Local Llama 2`** → 데이터 외부 유출 ❌ → 프라이버시, 준법 규정 리스크 ↓
  - 인터넷 불안정한 환경에서도 네트워크 지연 없이 오프라인에서도 동작 → 일관된 응답 속도 제공

- **하이브리드** 전략: **요청 길이**, **언어**, **하드웨어 여유**에 따라 두 모델을 동적으로 바꿈 → 비용, 품질 동시에 최적화
    - a. **Ollama** + **Llama 3.1**: 고품질이 필요한 한국어 or 장문 이메일 → **GPU** 서버에서 실행 → 처리
    - b. **CTransformers** → **Local Llama 2**: 짧은 영문 알림 메일 or 오프라인 환경에서 사용
- **Ollama** = LLM을 로컬 환경에서 실행할 수 있도록 도와주는 오픈 소스 도구
    - 모델 레지스트리, 실행 런타임, langchain용 어댑터 통합
        - ollama pull llama3.1 → 모델 다운로드 → 바로 사용 가능
        - REST, gRPC, CLI 등 원하는 방식으로 호출 가능
        - **LangChain-Ollama 패키지 사용 시 → ChatOpenAI와 거의 동일한 인터페이스로 코드에 연결 가능**
    - Linux, macOS, Windows 등 지원 → 노트북 or 클라우드 GPU 서버 모두 배포 절차 동일 → Llama 3.1 손쉽게 활용할 수 있는 모델 허브 역할

- **Ollama 설치 및 환경 설정**
    - **Ollama 설치**: [Ollama 설치 다운로드 페이지]
        - 지원 플랫폼 (Linux, macOS, Windows 등)에 맞는 **Ollama 설치하기**

```
pip install langchain-ollama
```

*

- **원하는 모델 설치하기**
    - **터미널**로 이동 → Ollama pull <모델 이름> 입력

```
ollama pull llama3.1:8b                    # 교재 예시

# 70b 이상부터는 고성능 GPU 컴퓨팅 파워 필요함
```

*

- [8b 모델 다운로드]

*

- *추가 명령어*
  - *모델 목록 확인*: ***ollama list*** → 다운로드 된 모든 모델 확인하고 싶을 때
  - *모델과 직접 채팅*: ***ollama run <모델 이름>*** → 명령 줄에서 직접 모델과 대화하고 싶을 때
  - *추가 명령어 확인*: ***ollma help*** → 더 많은 명령어 확인하고 싶을 때
    - 혹은 Ollama 공식 문서 참조

- **CTransformers 환경 설정**
  - **CTransformers, langchain-community 라이브러리 설치 필요 !**
    - CPU 에서 Transformer 모델 실행위해 사용
    - LLaMa, GPT4ALL, MPT 등 다양한 모델 지원

```
pip install ctransformsers langchain-community
```

*

- **Llama-2-7B-Chat** 모델의 경량화 버전 다운로드
  - CTransformers = GGML 포맷으로 저장된 모델 사용
  - Llama-2-7B-Chat-GGML → llama-2-7b-chat.ggmlv3.q8_0.bin 다운로드 및 사용
  - 다운로드 → **로컬 디렉터리에 저장** → CTransformers 라이브러리와 함께 사용할 준비 마치기

- **Streamlit 및 기타 패키지 설치**
  - a. **Streamlit**
    - 터미널 (or 명령 프름프트)에서 다음 명령어 입력하기

```
pip install streamlit
```

*

- b. **그 외 필요한 패키지 설치**

- ***`langchain.prompts`***: `LLM`에 전달한 프롬프트 → 템플릿화 → 관리하는 데 사용
- ***`CTransformers`***: 다양한 오픈 소스 `LLM` → `CPU` 환경에서 실행할 수 있는 라이브러리
- ***`OllamaLLM`***: `Ollama` 통해 제공되는 `Llama 3.1` 모델 호출 → 사용 가능

```
import streamlit as st
from langchain.prompts import PromptTemplate

# CTransformers → Llama, GPT4ALL-J, MPT, Falcon 같은 다양한 오픈 소스 모델 지원
from langchain_community.llms.ctransformers import CTransformers

# Ollama llama 3.1 model 연결하기
from langchain_ollama.llms import OllamaLLM
```

- **`이메일 응답 생성하기`**
  - a. **`LLM 응답 생성 함수`**
    - `getLLMResponse()` = 입력 → `LLM` → 이메일 응답 생성

---

2) `환경 설정`

- a. **`Python 3.12 버전 설치 확인 및 설치`**

```
# 설치 가능한 Python 3.12 버전 목록 확인하기
pip intsll --list | grep "3.12"

# 최신 안정 버전인 Python 3.12.x 설치 (예시: 3.12.4)
pyenv install 3.12.4

# 이미 설치되어 있다면 "already installed" 메시지가 출력됨
```

- b. **`새 가상환경 생성하기`**

```
# 원하는 가상환경 이름으로 생성하기
```

```
# pyenv virtuenv <설치한 Python 버전> <원하는 가상환경 이름>

pyenv virtuenv 3.12.4. lc_multi_email_env
```

- **c.** **`가상 환경 활성화`**

```
pyenv activate lc_multi_email_env
```

- **d.** **`패키지 설치 및 안정화`**
  - 가상환경 활성화된 상태에서 `requirements_multi_email.txt`의 모든 패키지 설치하기
  - 원본

```
# requirements_multi_email.txt의 모든 패키지 설치
pip install -r requirements_multi_email.txt
```

- **`이메일 응답 생성하기`**
  - `LLM` 응답을 생성하는 함수

- **`Llama-2-7B-Chat용 래퍼: CPU에서 Llama 2 실행`** 보충 설명
  - `C Transformers is the Python library that provides bindings for transformer models implemented in C/C++ using the GGML library`
  - 모델 `local` 컴퓨터에 `다운` → `model="다운받은 모델명"`을 `작성` → 진행
  - 용량이 작을수록 `경량화` 된 버전 → *`※주의: 성능이 조금 떨어질 수 있음`*
  - `GPT`와 같은 언어 모델
    - 이러한 `파일 형식` 사용 → `모델`의 `학습된 가중치`와 `구조`를 `저장` → 이를 `추론` (`inference`) 시 `불러와서` `사용`
    - `모델`을 `효율적`으로 `저장`하고 `불러올` 수 있게 함 → **`다양한 플랫폼`과 `환경`에서 `모델 추론`을 `원활`하게 할 수 있게 함**
  - 따라서, **`GGUF` 및 `GGML` 파일 형식 = `GPT`와 같은 `언어 모델`의 `맥락`에서 `모델 추론`을 위해 `사용`되는 `중요한 파일 형식`**
- Llama-2-7B-Chat.ggmlv3.q8_0.bin

---

3) `새로운 버전으로 업그레이드` → 코드 재생성

- *기존에 설치해뒀던* `llama3.2:3b` *가 있어* `llama3.1:8b` *대신 사용 예정*

---

① 📝 다국어 이메일 생성기 코드 변경 사항

- 최신 `Ollama` 버전은 파일 메타데이터 관리가 더 안정적인 **`GGUF` 포맷** 공식적으로 권장
  - **a. `GGUF 포맷` 파일로 교체하기**
    - *반면, 교재 속 버전 및 코드:* `GGML` *파일 → 종종* `unknown type` *오류 반환함*
    - 따라서 최신 모델 파일로 교체하는 것을 권장: **`llama2-7b.gguf`**
      - 예시: `04_K_M.gguf`, `05_K_M.gguf` 등
  - **b. `Modelfile` 수정 및 저장**
    - `llama2-7b.ggub` 파일과 같은 폴더에 있어야 함
  - **c. `Ollama` 모델 재등록**
    - **`Ollama 서버 실행 중인 상태`** → 터미널 열기 → 파일들이 있는 폴더로 이동 → 다음 명령어 실행

```
ollama create llama2-7b-imported -f Modelfile
```

- ⚠️ `Modelfile` *저장 및 실행 과정에서 오류 발생으로 해당 모델 임시 보류*

- **app.py**

```
# app.py (새 코드)

import streamlit as st
from langchain.prompts import PromptTemplate
from langchain_ollama.llms import OllamaLLM

def getLLMResponse(form_input, email_sender, email_recipient, language
    """
    getLLMResponse 함수는 주어진 입력을 사용하여 LLM(대형 언어 모델)으로부터 이메일
```

```
        매개변수:
        - form_input: 사용자가 입력한 이메일 주제.
        - email_sender: 이메일을 보낸 사람의 이름.
        - email_recipient: 이메일을 받는 사람의 이름.
        - language: 이메일이 생성될 언어 (한국어 또는 영어).

        반환값:
        - LLM이 생성한 이메일 응답 텍스트.
        """

        # 💡 설치된 'llama3.2:3b' 모델을 사용하도록 지정
        llm = OllamaLLM(model="llama3.2:3b", temperature=0.7)

        # 이메일 생성 프롬프트
        if language == "한국어":
            template = """
            다음 정보를 사용하여 전문적인 이메일을 한국어로 작성해 주세요.
            \n\n이메일 주제: {email_topic}
            \n보낸 사람: {sender}\n받는 사람: {recipient}
            \n\n이메일 내용:
            """
        else:
            template = """
            Write a professional email in English using the following info
            \n\nEmail Topic: {email_topic}
            \nSender: {sender}\nRecipient: {recipient}
            \n\nEmail content:
            """

        # 최종 PROMPT 생성
        prompt = PromptTemplate(
            input_variables=["email_topic", "sender", "recipient", "langua
            template=template,
        )

        # LLM을 사용하여 응답 생성
        response = llm.invoke(prompt.format(email_topic=form_input, sende
        print(response)

        return response


    st.set_page_config(
        page_title="이메일 생성기 📮",
        page_icon='📮',
        layout='centered',
        initial_sidebar_state='collapsed'
```

```
)
st.header("이메일 생성기 📮 ")

# 이메일 작성 언어 선택
language_choice = st.selectbox('이메일을 작성할 언어를 선택하세요:', ['한국어',

form_input = st.text_area('이메일 주제를 입력하세요', height=100)

# 사용자 입력을 받기 위한 UI 열 생성
col1, col2 = st.columns([10, 10])
with col1:
    email_sender = st.text_input('보낸 사람 이름')
with col2:
    email_recipient = st.text_input('받는 사람 이름')

submit = st.button("생성하기")

# '생성하기' 버튼이 클릭되면, 아래 코드를 실행하기
if submit:
    if not form_input or not email_sender or not email_recipient:
        st.error("이메일 주제, 보낸 사람, 받는 사람 이름을 모두 입력해야 합니다.")
    else:
        with st.spinner('생성 중입니다...'):
            response = getLLMResponse(form_input, email_sender, email_
            st.write(response)


# 실행 방법: 해당 폴더로 이동
# 터미널: streamlit run app.py
```

- `app.py` - **`local 환경에 맞춰 실행`**
  - a. **`Ollama 모델 이름 일치`시키기**

```
model="llama2-7b-imported"          # 사용할 모델 이름
```

*

- b. **`Ollama 서버 실행`**

```
# 코드 실행 전
```

```
ollama serve                                    # 백그라운드에서 서버가 실행 중이어야 함
```

*

- **c.** `모델 파일 형식`
  - `GGML` 파일 ❌ → `GGUF` 파일 ⭕

- `변경 사항 요약` - *`교재 코드`* *vs* *`app.py`*

구분	교재 초기 코드	수정된 app.py 코드
`LLM` 모듈	`from langchain.llms import CTransformers`	`from langchain_ollama.llms import Olla`
`LLM` 객체	`llm = CTransformers(...)`	`llm = OllamaLLM(model="llama3.2:3b",`
프롬프트	`LLM`에 더 명확한 지침을 제공하도록 템플릿 미세 변경	LLM에 더 명확한 지침을 제공하도록 템플릿 미세 변경
불필요한 코드	`CTransformers` 및 `Llama-2-7b-chat` 관련 주석 및 코드	*모두 제거* 또는 *주석 처리*

---

② 📝 LLM 효율성 분석

- `Llama-2-7b-chat` (*CTransformers*) **모델을 제거하는 것이 효율적인 이유**
  - **a.** `통합 환경 구성` **및** `관리의 단순화` *(Single Source of Truth)*
    - `Ollama`**의 역할:** `Ollama` = 다양한 `LLM`을 `다운로드`, `관리` 및 `실행`하는 `표준화`된 `서버` 역할
      - 예시: `llama3.2:3b`, `exaone3.5:7.8b` 등
    - `비효율성`: 만약 OllamaLLM (Ollama를 통해 모델 실행)과 CTransformers (별도의 라이브러리를 통해 모델 파일 직접 로드 및 실행)를 동시에 사용하려고 하면, `다음과 같은 문제 발생`
      - *`자원 중복`*: 두 라이브러리가 *`메모리`* / *`CPU`* / *`GPU`* *자원을 놓고* *`경쟁`*하거나 *`충돌`*할 *가능성*
      - *`설정 복잡성`*: *`모델 파일 경로`*, *`로드 설정`* *(quantization, CPU/GPU 할당 등)*을 `Ollama`와 `CTransformers` 두 곳 모두에서 관리 → *`설정`*이 *`복잡`*해지고 *`오류 가능성`*이 *`높아짐`*
    - `효율성`: `OllamaLLM` 하나만 사용 = `모든 모델 실행`을 `Ollama 서버`에 맡김 → `관리 포인트`가 `하나로 통일`되어 `매우` `깔끔`하고 `효율적`
  - **b.** `성능` **및** `기능 측면`**의** `이점`
    - `최신 기술 활용`: `Ollama`는 `지속적`으로 `업데이트` → `최신 모델`과 `최적화`된 `실행 환경`을 `지원`함
      - *최신 모델 예시:* *`Llama 3.2`*, *`Exaone 3.5`* 등
      - *최적화된 실행 환경 예시:* *`CUDA`*, *`Metal`* 등

    - **모델 교체의 용이성**:
        - **Ollama 환경**: **코드 크게 변경 ❌ → `model="모델명"`만 바꿈 → **즉시 다른 모델** *(예: llama3.2:3b → exaone3.5:7.8b)* 로 **실험 가능**
        - *CTransformers*: *모델을 변경할 때마다* *새로운 .gguf 파일* *(혹은 .ggml 파일)을* *다운로드* → *로드 설정을* *다시 구성해야 하는 번거로움 발생*
- 결론적으로, **현재 Ollama를 사용하고 있는 경우, OllamaLLM만을 사용 → 통합된 환경에서 최신 모델을 활용하는 것 = 가장 효율적이고 안정적인 로컬 LLM 개발 방식**

---

③ 환경 설정

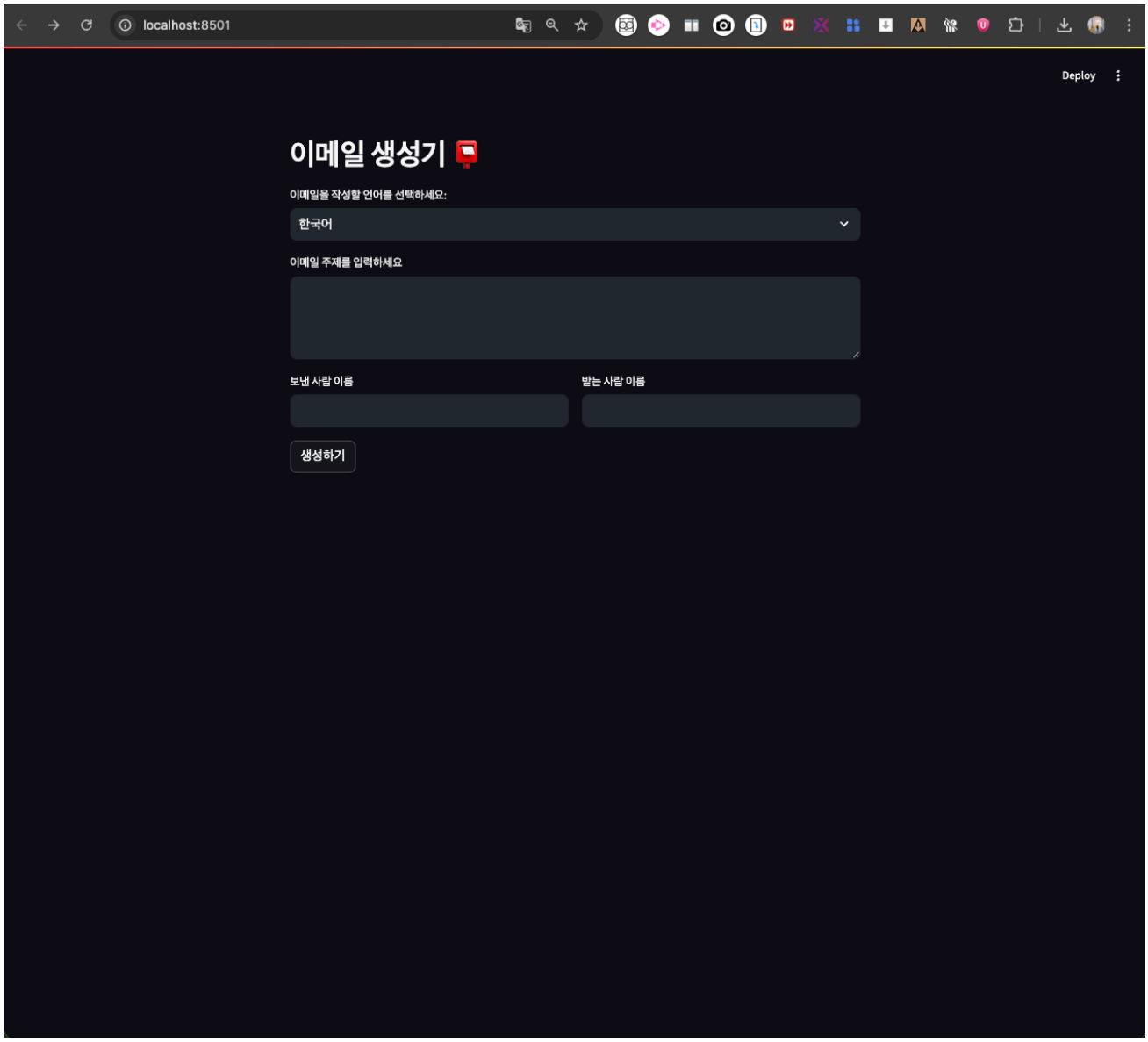
- ⚠️ 실행 전 필수 준비 사항
    - mac 터미널

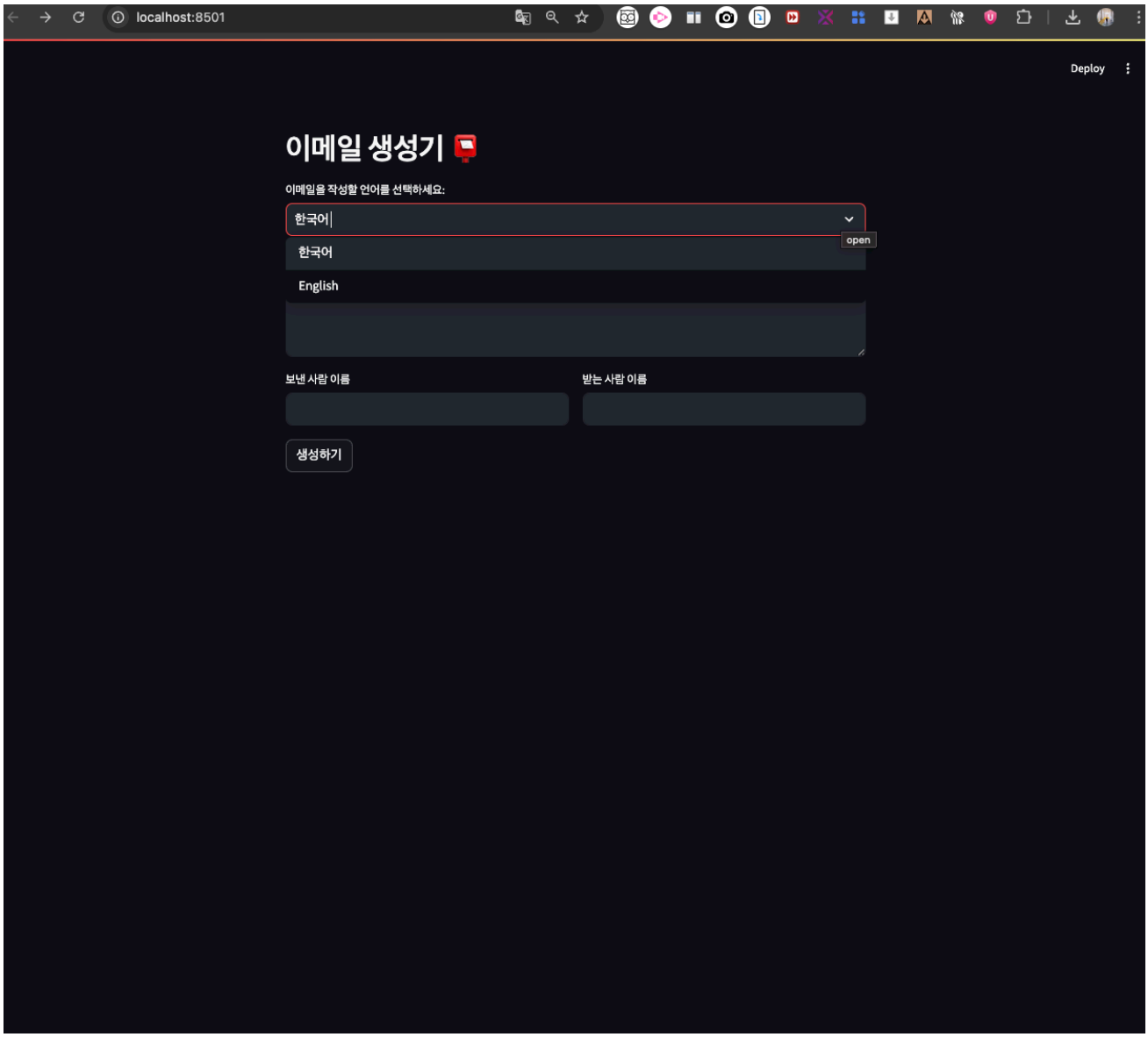
```
# Ollama 서버가 실행 중이어야 함
ollama serve

# `llama3.2:3b` 모델이 설치되어 있어야 함
ps aux | grep ollama              # ollama 설치된 모델 확인하기

# 설치된 모델이 없다면 설치하기
ollama run llama3.2:3b

# 설치된 목록 확인하기
ollama list                       # 테스트로 정상 동작하는지 확인하기

# `langchain`, `langchain-ollama` 라이브러리가 설치되어 있어야 함

pip install langchain langchain-ollama
```

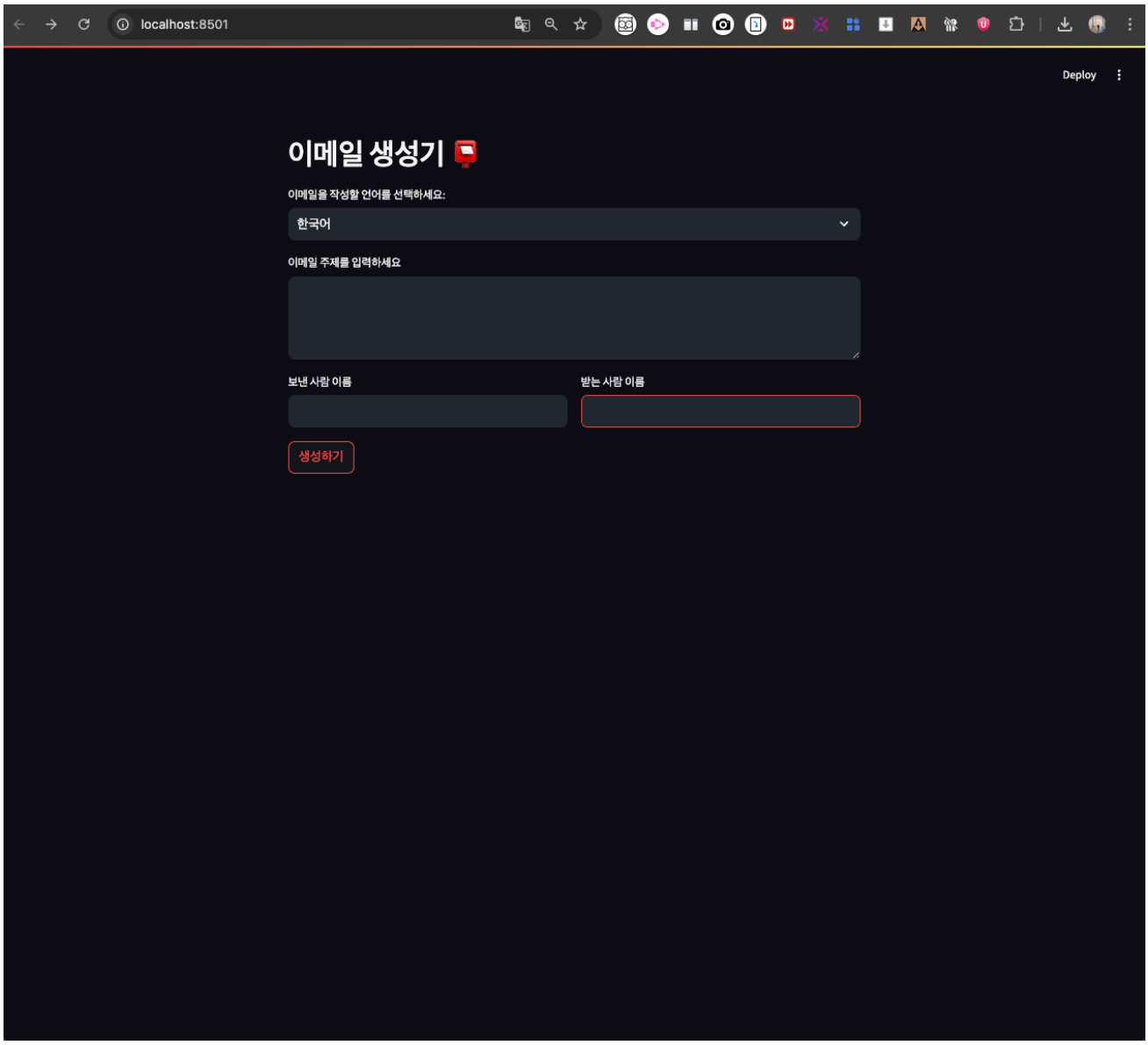
---

④ Python Script로 실행하기

- app.py

- 
localhost:8501
Deploy
이메일 생성기
이메일을 작성할 언어를 선택하세요:
한국어
이메일 주제를 입력하세요
보낸 사람 이름
받는 사람 이름
생성하기


localhost:8501
Deploy
이메일 생성기
이메일을 작성할 언어를 선택하세요:
한국어
open
한국어
English
보낸 사람 이름
받는 사람 이름
생성하기



---

⑤ 주피터 노트북으로 실행해보기

```
# 필요한 패키지 임포트 해보기
import sys
from langchain.prompts import PromptTemplate
from langchain_ollama.llms import OllamaLLM
print("✅ 필요한 라이브러리 임포트 완료.")
```

- `✅ 필요한 라이브러리 임포트 완료.` - (`0.9s`)

```
def getLLMResponse(form_input, email_sender, email_recipient, language):
    """
    getLLMResponse 함수는 주어진 입력을 사용하여 LLM(대형 언어 모델)으로부터 이메일 응답을 생성함

    매개변수:
    - form_input: 사용자가 입력한 이메일 주제.
    - email_sender: 이메일을 보낸 사람의 이름.
    - email_recipient: 이메일을 받는 사람의 이름.
    - language: 이메일이 생성될 언어 (한국어 또는 영어).
```

```
    반환값:
    - LLM이 생성한 이메일 응답 텍스트.
    """

    # OllamaLLM 객체 생성 = 'llama3.2:3b' 모델 사용
    # 참고: Ollama가 다른 포트에서 실행 중이라면 base_url을 지정해야 함
    try:
        llm = OllamaLLM(model="llama3.2:3b", temperature=0.7)
    except Exception as e:
        print(f"Ollama 연결 오류: {e}", file=sys.stderr)
        print("Ollama 서버가 실행 중인지, 모델이 설치되어 있는지 확인해 주세요.", file=sys.stder
        return "ERROR: Ollama LLM connection failed."


    # 이메일 생성 프롬프트
    if language == "한국어":
        template = """
        다음 정보를 사용하여 전문적인 이메일을 한국어로 작성해 주세요.

        이메일 주제: {email_topic}
        보낸 사람: {sender}
        받는 사람: {recipient}

        이메일 내용:
        """
    else:
        template = """
        Write a professional email in English using the following information.

        Email Topic: {email_topic}
        Sender: {sender}
        Recipient: {recipient}

        Email content:
        """

    # 최종 PROMPT 생성
    prompt = PromptTemplate(
        input_variables=["email_topic", "sender", "recipient", "language"],
        template=template,
    )

    # LLM → 응답 생성
    response = llm.invoke(prompt.format(email_topic=form_input, sender=email_sende
    return response
```

```
# ------------------------------------------------------------------
# 테스트 실행
# ------------------------------------------------------------------

# === 1차 테스트: 한국어 이메일 ===

email_topic_kr = "다음 분기 마케팅 전략 변경 사항 보고"
email_sender_kr = "김영희 부장"
email_recipient_kr = "이철수 팀장"
```

```
language_choice_kr = "한국어"

print(f"\n--- [1차 테스트: {language_choice_kr}] ---")
print("생성 중... (Ollama 응답 대기)")
response_kr = getLLMResponse(email_topic_kr, email_sender_kr, email_recipient_kr,

print("\n[생성된 한국어 이메일 내용]\n")
print(response_kr)
```

- **`test_1`**(***`ver_ko`***) - (`31.8s`)

*

- *`Llama 모델이 한국어에 가진 한계점이 보임`*

```
--- [1차 테스트: 한국어] ---
생성 중... (Ollama 응답 대기)

[생성된 한국어 이메일 내용]

subject: 다음 분기 마케팅 전략 변경 사항 보고

dear 이철수 팀장,

안녕하세요. kimyounghee입니다.

이번 달은 새로운 분기로进入합니다. 이에 따라 마케팅 전략도 변경되어야 합니다.

제가 제안한 변경 사항은 다음과 같습니다.

1.  **마케팅 objetivo의 변경** : 현재의 objectives를 review하고, 새로 정의하는 objective가 
2.  **Target audience의 확장** : 새로운 target audience를 추가하여 마케팅 전략을 변경할 수 있
3.  **마케팅 채널의 diversification** : 기존의 channel에 새로운 channel을 추가하여 마케팅 전략
4.  **마케팅 budget의 increase** : 현재의 budget를 review하고, 새로 증가하는 budget가 필요합

이 변경 사항은 팀과 동등한 의견을 benöt습니다.因此, I propose a meeting to discuss these ch

If you have any questions or concerns, please don't hesitate to reach out to me.

Thank you for your time and consideration.

Best regards,

kimyounghee 부장
```

```
# === 2차 테스트: 영어 이메일 ===
email_topic_en = "Follow-up on the Q3 Financial Review meeting action items"
email_sender_en = "Alice Johnson"
email_recipient_en = "Bob Williams"
language_choice_en = "English"

print(f"\n--- [2차 테스트: {language_choice_en}] ---")
print("생성 중... (Ollama 응답 대기)")
response_en = getLLMResponse(email_topic_en, email_sender_en, email_recipient_en,

print("\n[생성된 영어 이메일 내용]\n")
```