Artificial Intelligence Experiment

최종 보고서

전공: 인공지능학과
학년: 2학년
학번: 20231834
이름: 김민선

1 프로젝트 소개

먼저 해당 프로젝트는 이전 12주차 결과보고서에서 설명한 페이지의 연장으로, 해당 페이지를 그대로 최종 프로젝트에 활용하였다. 따라서 해당 보고서에 이미 작성한 내용은 중복 작성하지 않았다.

기존의 언어 학습 챗봇 서비스를 개선한 프로젝트를 구현하고자 하였다. 평소 언어 학습에 관심이 있어 이 주제를 택하였으며, 기존 서비스를 사용하며 느낀 단점들을 이번 프로젝트에서 개선하고자 하였다.

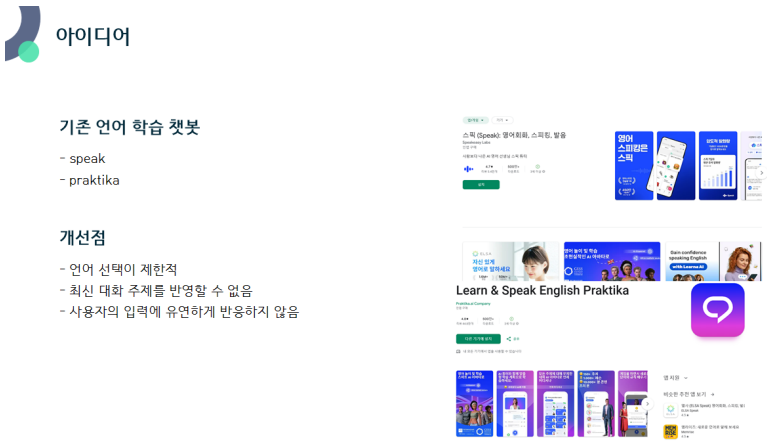

Figure 1: 프로젝트 주제 선정 계기

챗봇에서는 사용자가 선택한 언어에 따라 해당 언어로, 사용자가 입력한 주제에 대해 대화를 하며, 사용자의 입력을 자연스럽게 고쳐주는 기능을 제공한다(intermediate advanced 선택 시). 또는 사용자가 해당 언어로 대화할 수 없는 경우 (beginner 레벨) 영어로 대화를 하되, 사용자의 입력을 선택 언어로 번역하는 기능을 제공하도록 했다. 또한, 최신 정보를 검색해 답변에 활용하는 기술을 활용하여 최근 발생한 사건이나, 최신 방영 드라마, 영화 등에 대해서도 폭넓은 대화가 가능한 chatbot을 제공하고자 여러 기능을 추가하였다.

이 때 페이지에서 구현해야 하는 것들은 아래와 같다.

1. 새로운 채팅을 만들 수 있는 기능
2. 새로운 채팅에서 학습하고자 하는 언어, 해당 언어의 습득 수준, 대화하고자 하는 주제를 입력하는 기능
3. 메인 화면에서 [채팅 / 프롬프트 확인 및 추가 작성 / 사용자의 입력을 수정한 결과 혹은 번역한 결과를 확인] 하는 창을 선택하는 기능

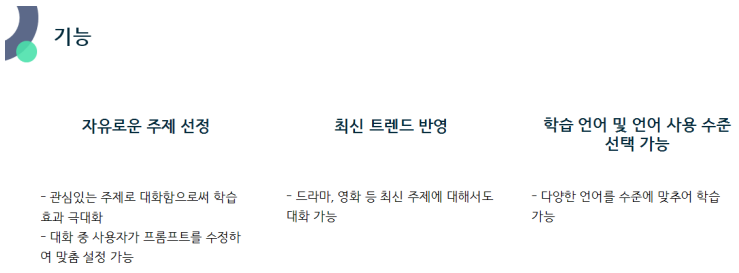

Figure 2: 프로젝트 요약

2 기존에 구현한 기능

메인 화면을 출력하는 기능, 그리고 새로운 채팅의 설정을 위한 창은 12주차에서 완성하였다. 따라서 이번 보고서에는 그 외의 기능들을 설명한다.

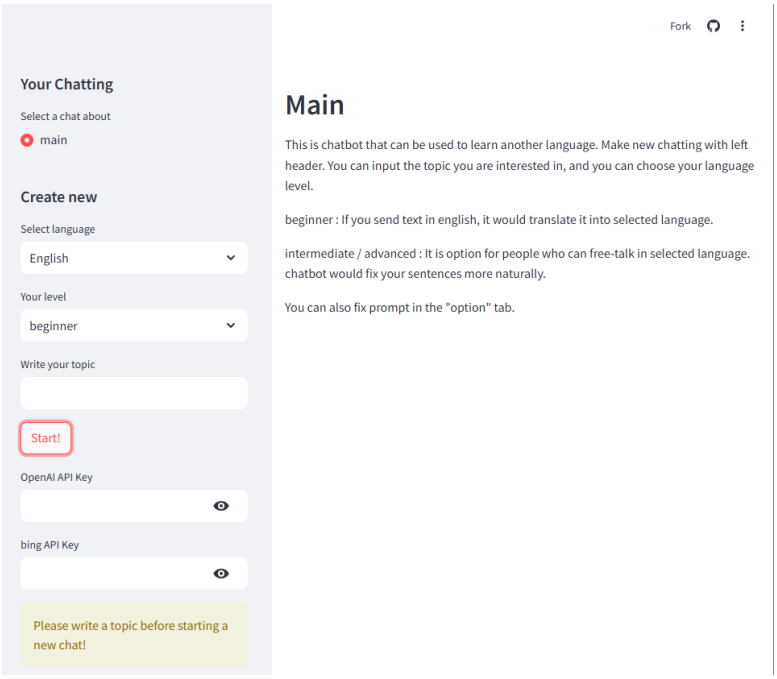

Figure 3: 기존 구현 화면

3 코드 설명 및 실행 화면

3.1 라이브러리 불러오기

```
import streamlit as st
from openai import OpenAI
import bing
```

위에서 bing은 RAG 검색을 위해 따로 작성한 python 파일이다.

```
    st.header("Create new")
    language = st.selectbox('Select language', ['English', 'korean', 'japanese'])
    level = st.selectbox('Your level', ['beginner', 'intermediate', 'advanced'])
    topic = st.text_input('Write your topic')
    new_chat = st.button('Start!')

    openai_api_key = st.text_input("OpenAI API Key", type="password")
    bing_api_key = st.text_input("bing API Key", type="password")

    if new_chat:
        # Add the new chat to the list if the topic is not empty
        if topic:
            if topic in st.session_state.chat_list :
                st.warning("Chatting already exists. Please choose another topic!")
            else :
                st.session_state.chat_list.append(topic)
                st.session_state.chat_list_option[topic] = {'language':language, 'level':level
    , 'topic':topic}
        else:
            st.warning("Please write a topic before starting a new chat!")
```

아래의 채팅 선택에 이어 위의 코드에서는 채팅의 설정을 변경하는 기능 및 채팅 생성 버튼이 눌렸을 경우 새 채팅을 생성한다. 언어 선택은 영어, 한국어, 일본어가 가능하도록 하였고, 단계는 beginner, intermediate, advanced의 세 가지로 나누었다. topic이 입력되었으며 버튼이 눌렸을 경우 해당 입력에 따라 새로운 채팅을 생성한다. topic 입력이 없거나 기존 topic과 중복될 경우 채팅을 생성하지 못한다. 또한 사이드바에서 api키를 입력받도록 하였다.

3.2 기능 3 : 메인 화면

3.2.1 채팅 화면

```
else :
    chat, option, fixing = st.tabs(['chatting', 'option', 'fixed inputs'])
```

main 화면 선택중이 아닐 경우, 화면을 세 개의 tab(chatting, option, fixed inputs)으로 나누었다.

```
    with chat :

        if "messages" not in st.session_state:
            st.session_state.messages = {}
            st.session_state.fix_messages = {}
        if selected_chat not in st.session_state.messages :
            text = f'you are chatting with user in context of '+ selected_chat
            if (st.session_state.chat_list_option[selected_chat]['level'] == 'beginner') :
                text += '. Talk with user in English, and suggest better chance for user to
    speak ' +st.session_state.chat_list_option[selected_chat]['language']
            else :
                text += '. Talk with user in '+st.session_state.chat_list_option[selected_chat
    ]['language']
            text += ". You don't have to answer too long. You are friend-like chat bot."
            st.session_state.messages[selected_chat] = [{'role':'system', 'content' : text}]

            if st.session_state.chat_list_option[selected_chat]['level'] == 'beginner' :
                fixing_require = f"You are translator. Translate input texts in {st.
    session_state.chat_list_option[selected_chat]['language']}. you are not supposed to answer
    , but translate the input. TRANSLATE ENGLISH TO {st.session_state.chat_list_option[
    selected_chat]['language']}."
            else :
                fixing_require = "You are writing assistant. You are not supposed to answer
    the question, but fixing the inpuy text."
            st.session_state.fix_messages[selected_chat] = [{'role':'system', 'content' :
    fixing_require}]
```

먼저 chatting 탭을 선택했을 경우 실행되는 코드이다. 위의 코드는 chatting이 시작되었을 때, 기존의 채팅이 없을 경우 실행되는 코드이다. 먼저 채팅 이력 및 프롬프트 설정을 저장하는 messages 변수가 존재하지 않을 경우 이를 생성한다. 다음으로 선택 채팅창의 메세지가 messages 변수에 존재하지 않을 경우 messages에 해당 선택 채팅창의 설정에 따라 프롬프트를 작성한 후 messages에 저장한다. beginner일 경우와 그렇지 않은 경우 프롬프트를 다르게 작성하는 것을 볼 수 있다.
이번 프로젝트에서 LLM 모델은 크게 두 가지로 사용된다. 첫 번째는 사용자와 대화를 하는 모델이며, 두 번째는 사용자의 입력을 고쳐주거나 번역하는 모델이다. 코드에서 messages는 첫 번째 경우를 위한 프롬프트이며, fix_messages가 두 번째 경우를 위한 프롬프트이다.

```
 if not openai_api_key or not bing_api_key:
            st.info("Please add your API key to continue.", icon="*")
```

위의 코드에서는 api 키가 입력되지 않을 경우 채팅이 시작되지 못하도록 한다.

```
        else :
            client = OpenAI(api_key=openai_api_key)


            for message in st.session_state.messages[selected_chat][1:]:
                with st.chat_message(message["role"]):
                    st.markdown(message["content"])
```

먼저 api키를 설정한 후 기존 messages를 출력한다.

```
            if prompt := st.chat_input("What is up?"):

                result_texts = bing.get_relevant_texts(f"{selected_chat} {prompt}",
    bing_api_key)

                # Store and display the current prompt.
                prompt_with_context = f"""
                here is the context about {selected_chat}.

                answer in context below :
                {str(result_texts)}

                Question :
                {prompt}
                """

                with st.chat_message("user"):
                    st.markdown(prompt)
```

사용자의 입력을 저장한 후 대화 챗봇의 rag에 사용할 데이터를 추가로 입력해 prompt_with_context로 저장하였다. 위에서 사용한 bing.get_relevant_texts는 관련 페이지를 검색한 후 텍스트를 크롤링하는 함수로, 후의 bing.py 코드 설명에서 자세히 다루겠다.

```
                if st.session_state.chat_list_option[selected_chat]['level'] == 'beginner' :
                    prompt_for_fix = "translate this in " + st.session_state.chat_list_option[
    selected_chat]['language'] + ". " + prompt
                    st.session_state.fix_messages[selected_chat].append({"role": "user", "
    content": prompt_for_fix})

                else :
                    st.session_state.fix_messages[selected_chat].append({"role": "user", "
    content": prompt})
                st.session_state.messages[selected_chat].append({"role": "user", "content":
    prompt_with_context})
```

위에서는 사용자의 입력을 수정해 줄 모델을 위한 프롬프트 입력을 수정하는 코드이다. beginner 레벨에서, 사용자의 입력을 번역하라는 명령이 제대로 입력되지 않아 추가로 수정하는 단계를 거쳤다. 프롬프트 수정이 끝난 후 대화 기록을 저장하는 messages, fix_messages 딕셔너리에 저장한다.

```
                fix_stream = client.chat.completions.create(
                    model="gpt-4o-mini",
                    messages=[
                        {"role": m["role"], "content": m["content"]}
                        for m in st.session_state.fix_messages[selected_chat]
                    ],
                    stream=True,
                )

                with st.chat_message("assistant"):
                    response = st.write_stream(fix_stream)
                st.session_state.fix_messages[selected_chat].append({"role": "assistant", "
    content": response})

                stream = client.chat.completions.create(
                    model="gpt-4o-mini",
                    messages=[
                        {"role": m["role"], "content": m["content"]}
                        for m in st.session_state.messages[selected_chat]
                    ],
                    stream=True,
                )

                with st.chat_message("assistant"):
                    response = st.write_stream(stream)
                st.session_state.messages[selected_chat].append({"role": "assistant", "content
    ": response})

                st.session_state.messages[selected_chat][-2]['content'] = prompt
                st.session_state.fix_messages[selected_chat][-2]['content'] = prompt
```

위에서 작성한 프롬프트에 따라 메세지를 생성한 후 출력하는 코드이다. 코드 작성 순서에 따라 사용자의 입력을 수정하는 코드가 먼저 시행되며, 다음으로 사용자의 입력에 답변하는 코드가 시행된다. 이를 실행한 결과는 아래와 같다.

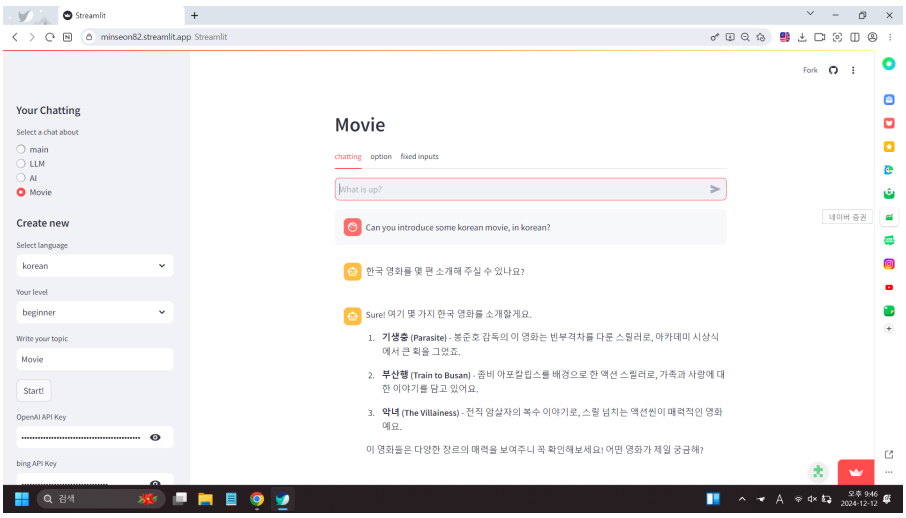

Figure 4: 실행 화면 - beginner

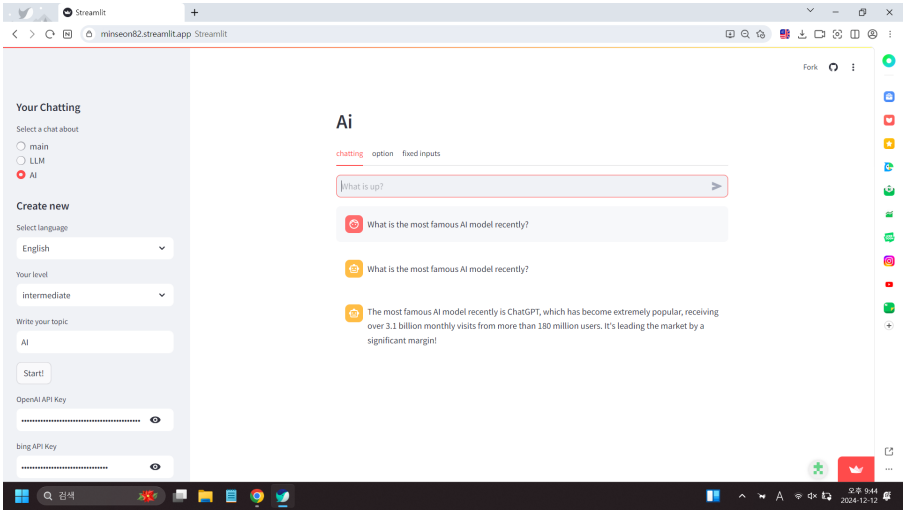

Figure 5: 실행 화면 - intermediate

3.2.2 option 화면

```
    with option :
        st.write("Your option :")
        st.write("-selected language :", st.session_state.chat_list_option[selected_chat]['language'],
        " \n -selected level :",st.session_state.chat_list_option[selected_chat]['level'])

        st.write("".join(st.session_state.messages[selected_chat][0]['content'].split("chatbot.")[1:]))
        fixed_prompt = st.text_input("add text into your prompt")
        if fixed_prompt :
            st.session_state.messages[selected_chat][0]['content'] += fixed_prompt
```

option에서는 기존의 설정을 확인하는 기능과 prompt를 추가 입력하는 기능을 다룬다. 위의 prompt에서 추가된 내용은 챗봇에 바로 반영되며, 다음 채팅부터 수정된 prompt를 사용하게 된다.

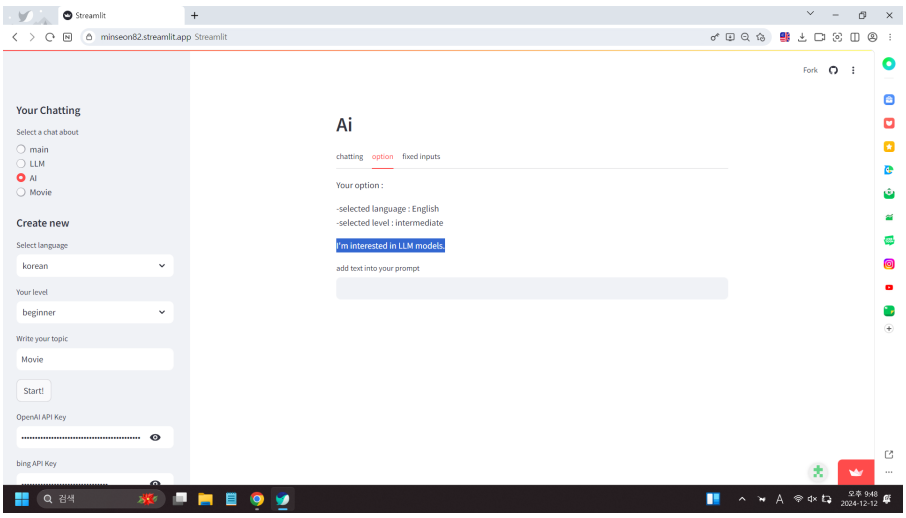

Figure 6: option 화면

3.2.3 fixing inputs 화면

```
    with fixing :
        for message in st.session_state.fix_messages[selected_chat][1:]:
            with st.chat_message(message["role"]):
                st.markdown(message["content"])
```

fixing 화면에서는 이전에 수정된 사용자 입력을 나열하는 역할만을 수행한다. 시행 결과 화면은 아래와 같다.

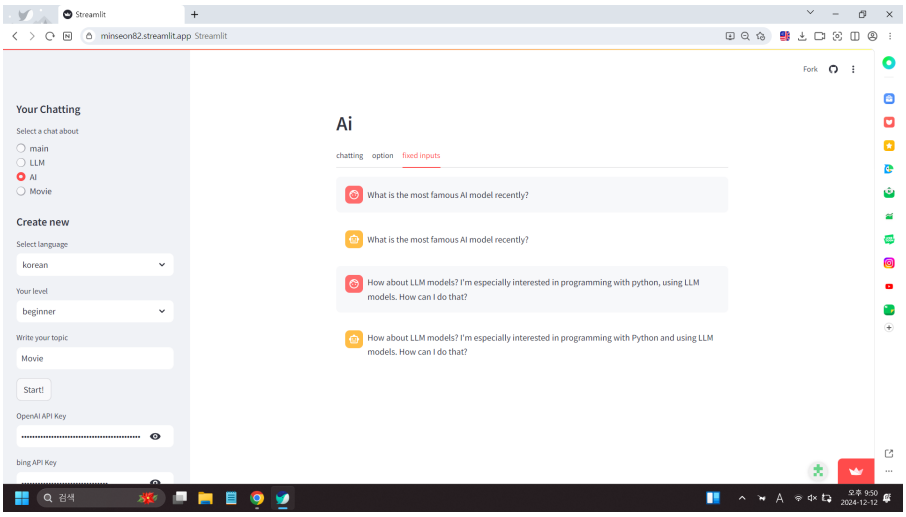

Figure 7: fixing inputs 화면

3.3 bing.py

```
import requests
from bs4 import BeautifulSoup

# Function to search Bing for relevant websites
def search_bing(query, api_key, endpoint="https://api.bing.microsoft.com/v7.0/search",
    num_results=10):
    headers = {"Ocp-Apim-Subscription-Key": api_key}
    params = {"q": query, "count": num_results}
    response = requests.get(endpoint, headers=headers, params=params)
    response.raise_for_status()
    results = response.json()
    return [result["url"] for result in results.get("webPages", {}).get("value", [])]

# Function to extract textual content from a webpage
```

```python
def scrape_website(url):
    try:
        response = requests.get(url, timeout=10)
        response.raise_for_status()
        soup = BeautifulSoup(response.text, "html.parser")

        # Extract visible text
        paragraphs = soup.find_all("p")
        content = "\n".join(p.get_text() for p in paragraphs if p.get_text())
        return content.strip()
    except Exception as e:
        print(f"Failed to scrape {url}: {e}")
        return None

# Main workflow
def get_relevant_texts(query, api_key):
    success = 0
    print(f"Searching for: {query}")
    urls = search_bing(query, api_key)

    print("\nRetrieved URLs:")
    for url in urls:
        print(url)

    texts = []
    for url in urls:
        print(f"\nScraping: {url}")
        text = scrape_website(url)
        if text:
            print(text[:100])
            if len(text) > 10000 :
                text = text[:10000]
            texts.append(text)
            success += 1
        if success >= 3 :
            break
    return texts
```

위 코드 파일은 검색 및 크롤링을 위해 작성한 사용자 정의 함수들을 작성한 파일이다. search_bing() 코드에서 검색 및 상위 10개 링크를 추출하며, scrap_website()에서는 받아온 링크에 대해 크롤링을 한다. chatbot 파일에서 시행되는 함수인 get_relevant_texts() 함수는 query와 api 키를 입력받아 해당 입력으로 검색을 수행한 후, 받아온 10개 링크에서 3개의 크롤링이 성공할 때까지 크롤링을 수행한다. 또한, gpt 모델에 입력 길이 제한이 있기 때문에, text의 길이가 너무 긴 경우 최대 10000자까지 반환할 수 있도록 했다.

4 개선점

현재 LLM 모델들이 영어로 대화하는 것에 특화되어 있기 때문에 사용자가 기본적으로 어느 정도 영어를 사용한다는 가정 하에 프로젝트를 진행하였다. 따라서 beginner 단계에서 영어를 기준으로 대화를 이어나가게 되는데, 사용자의 모어를 입력받아 기본 언어를 변경할 수 있는 기능을 추가해 서비스를 개선할 수 있을 것이다. 또한 사용자가 언어를 자유롭게 선택할 수 있도록 하는 기능을 추가할 수 있을 것 같다.

또한, 현재 intermediate 단계와 advanced 단계는 차이가 없는데, 여기에서 역시 별도로 프롬프트를 조정하여 advanced 단계에서 더욱 고차원적으로 언어 학습이 가능하도록 할 수 있을 것이다.

코드 상의 문제로는, LLM이 정보를 아는지 여부에 관계 없이 모든 입력에 대해 검색 및 크롤링을 수행한다. 이는 입력의 크기를 키워 서비스의 효율성을 떨어트리므로, LLM이 정보를 알고 있는지 판단하여 사전에 입력되지 않은 정보의 경우에만 RAG를 활용하도록 해 효율성을 올릴 수 있을 것이다. 또는, 주제가 최초로 선정되었을 때 다수의 페이지를 크롤링한 후 임베딩하여 모델이 해당 정보를 토대로 RAG를 수행할 수 있도록 해 효율성을 높일 수 있다.