# Streamlit prototyping

2022.11.24. HAI 1팀

한양대학교

HANYANG UNIVERSITY

# 프로젝트 마무리

- 이번 시간에는 누군가 Huggingface에 업로드해둔 방언 번역 모델을 우리가 학습시킨 분류 모델과 조합하여 번역기 프로그램을 구성해 볼 거에요!

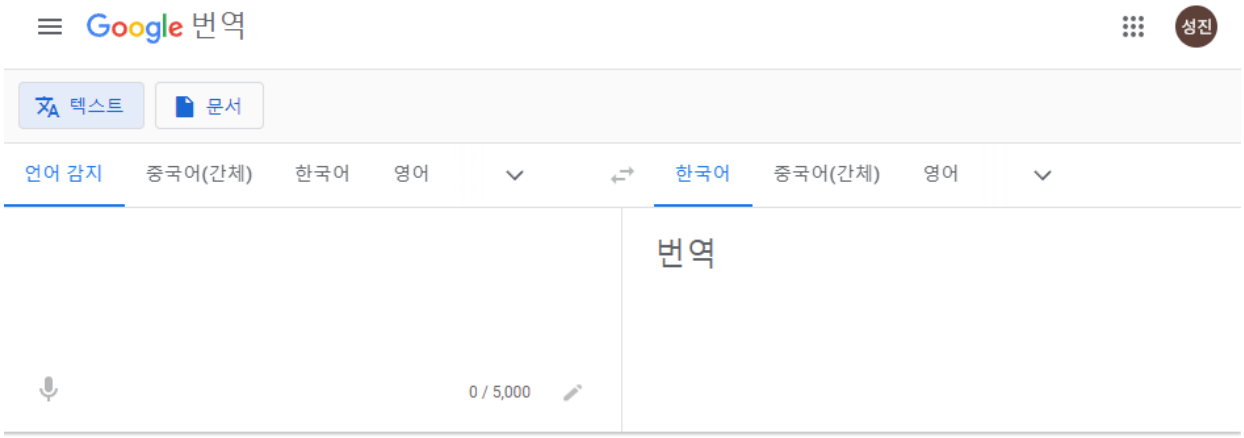
- 완성된 시스템은 **streamlit**이라는 데이터 사이언스 애플리케이션 프로토타입 개발 API를 이용하여 보기 쉽고 사용하기 쉬운 GUI 형태의 웹 애플리케이션으로 완성할 거에요.

- 출발 언어와 목표 언어가 매칭되는 조합은 상당히 여러 가지가 있을 텐데, 어떻게 하면 번역 정확도도 높이면서 서비스를 원활하게 제공할 수 있도록 할 수 있을까요?

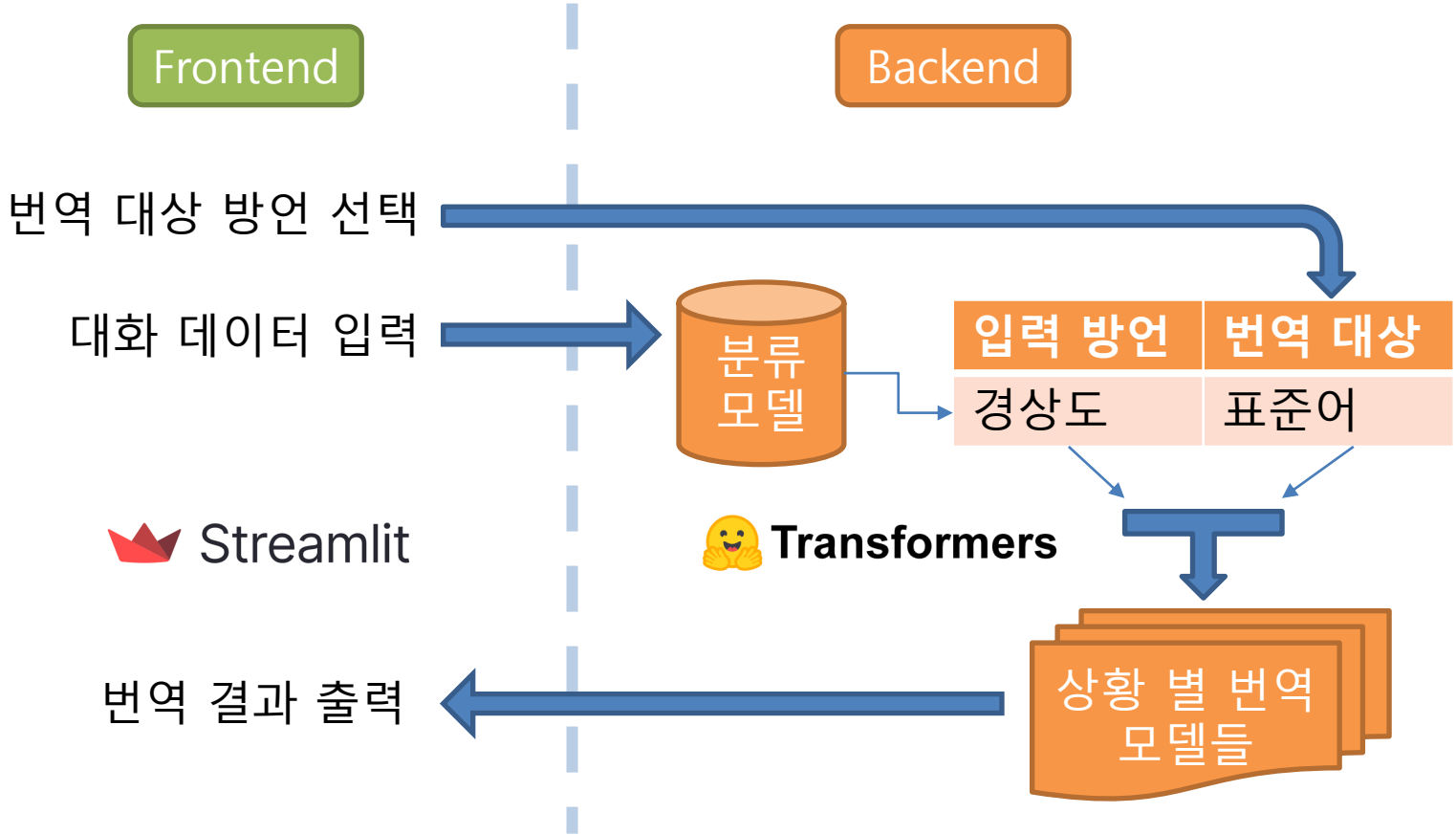
- 실제 AI 모델을 활용한 서비스를 제공하기 위해서는 어떤 요소들을 고려해야 할까요?

# Recall - 우리가 만들 친구는 어떤 프로그램인가?

- 사람의 발화 텍스트를 입력하면, 표준어인지 아니면 특정 지역 방언인지 자동으로 분류해줘요(구글 번역기의 언어 자동 인식 기능).
- 현재 입력된 텍스트를 원하는 지역의 방언이나 표준어로 번역할 수 있어요.
- 웹 애플리케이션으로 구현하여 누구나 접근해서 사용 가능해요.

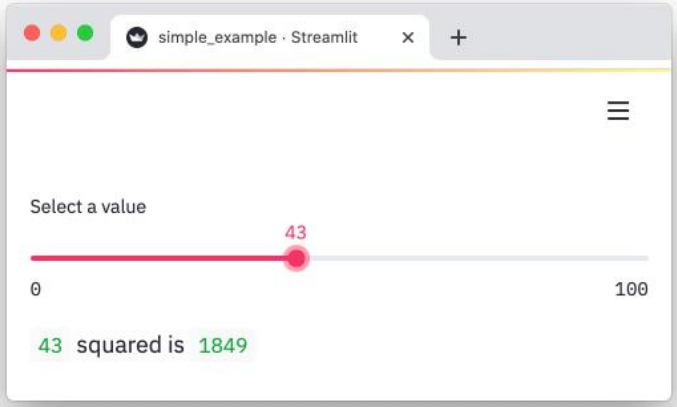


# Recall - 실제 개발해야 할 구조



# 프로토타입 웹 애플리케이션 개발

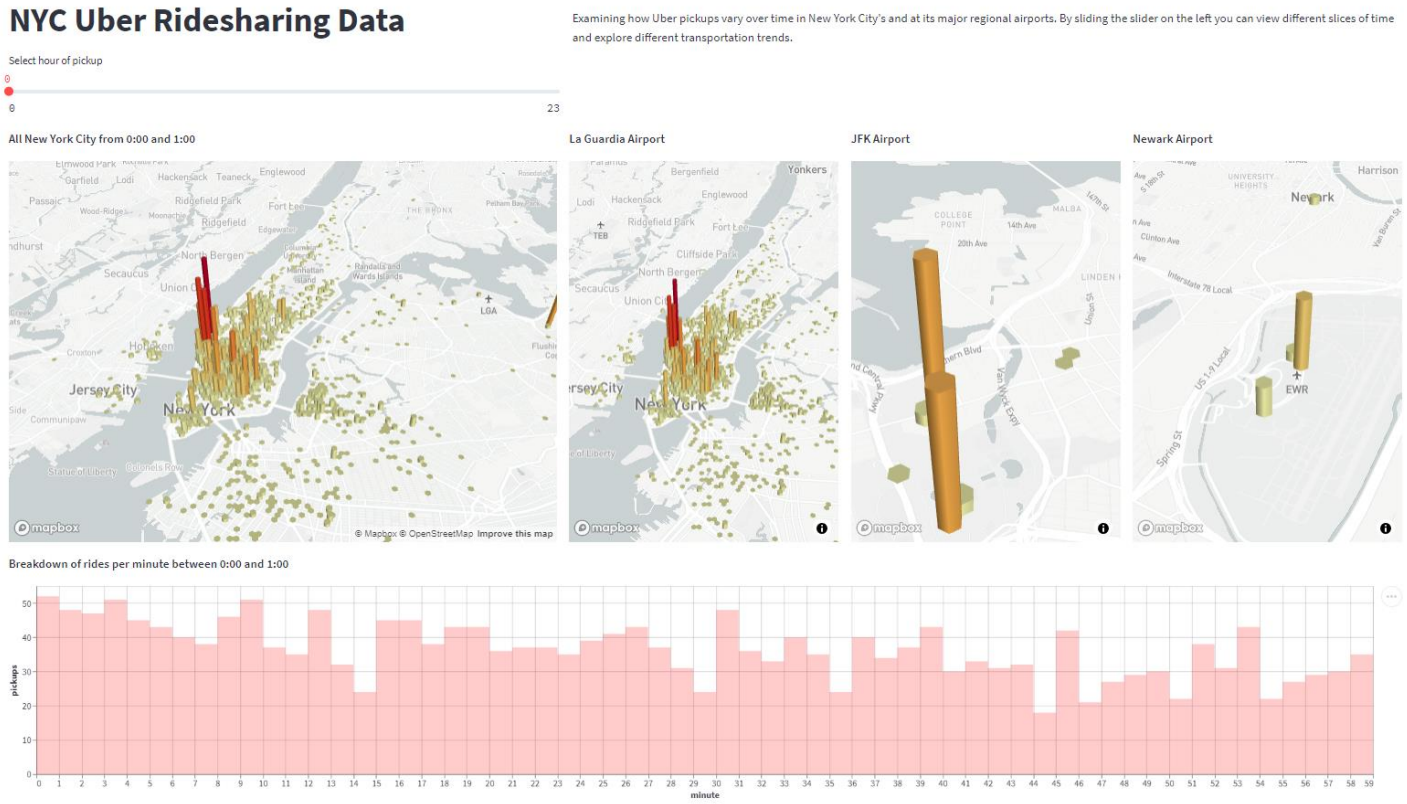
- 머신 러닝 모델을 학습하고 잘 동작하는지 확인하는 것 뿐만 아니라, 다른 사람들과 공유하기 위한 도구인 **Streamlit**을 활용해 보겠습니다.
- 간단한 프로토타입을 웹 형식으로 빠르게 만들어 공유하기에 최적화된 도구로, 대화 텍스트를 입력하고 번역 타겟 방언을 선택하고 결과를 확인할 수 있는 UI를 구현해야 해요.




Streamlit

# Streamlit 살펴보기

- Streamlit은 Python 파일을 작성한 후 streamlit run [파일명].py 를 실행하는 것 만으로도 간단하게 원하는 데이터와 모델을 표시하는 웹 페이지를 만들 수 있어요.
- Streamlit 갤러리(https://streamlit.io/gallery)에 들어가서 여러 주제의 데이터 사이언스 & 머신러닝 애플리케이션이 구현된 것을 구경해보면 어떤 라이브러리인지 감이 올 거에요.



# streamlit.cache

- Streamlit으로 구현된 웹 애플리케이션에서 transformers 라이브러리를 활용한 모델을 불러와서 사용할 때는, 매번 실행할 때 마다 불필요하게 모델이 로드되지 않도록 cache 할 필요가 있습니다.
- streamlit..cache라는 function decoerator를 모델을 로드하는 함수 앞에 붙여주면 모델을 로드한 결과를 cache하여 반복해서 사용할 수 있도록 해 실행 시간을 줄일 수 있습니다.

```python
@st.cache(show_spinner=False, allow_output_mutation=True, suppress_st_warning=True, max_entries=1)
def load_classification_model():
    with st.spinner("Loading model for classification..."):
        os.makedirs('./models', exist_ok=True)

        model_url = "https://github.com/GirinMan/HAI-DialectTranslator/raw/main/classifier_training/model.pth"
        model_response = requests.get(model_url, allow_redirects=True)

        open('./models/model.pth', 'wb').write(model_response.content)

        model_ckpt = "monologg/koelectra-small-v3-discriminator"
        model = AutoModelForSequenceClassification.from_pretrained(model_ckpt, num_labels=2).to('cuda')
        tokenizer = AutoTokenizer.from_pretrained(model_ckpt)

        state = torch.load('./models/model.pth', map_location='cuda')
        model.load_state_dict(state, strict=False)
        return model, tokenizer
```

# 모델 inference를 위한 함수 정의

- Streamlit으로 구현된 텍스트 입력 필드, 버튼 등과 상호작용하여 원하는 시점에 원하는 모델(지난 시간동안 구현한 사투리 지역 구분 모델 및 huggingface hub에 업로드되어 있는 사투리 번역 모델)들을 적절히 사용하기 위한 helper function들을 정의합니다.
- Classification task와 translation task에 사용되는 모델과 토크나이저가 서로 다르기 때문에, 각각 적절한 것을 파라미터로 입력하여 사용할 수 있도록 합니다.

```python
def classification(model, tokenizer, input_txt):
    input_tensor = torch.tensor([tokenizer.encode(input_txt)]).to('cuda')

    with torch.no_grad():
        preds = model(input_tensor).logits.cpu()

    result = np.argmax(preds, axis=1).item()

    return result

def translation(model, tokenizer, input_txt):
    input_tensor = torch.tensor([tokenizer.encode(input_txt)]).to('cuda')

    generated_ids = model.generate(input_tensor, num_beams=4)
    result = tokenizer.decode(generated_ids[0], skip_special_tokens=True)

    return result
```

# 화면에 표시될 UI 구성

- Streamlit의 기본 API인 title, text_area, button, selectbox 등을 이용하여 입력 언어를 선택하고 발화 텍스트를 입력할 수 있도록 하는 UI를 구성합니다.
- 커맨드 라인에서 **streamlit run app.py** 명령을 실행하면 바로 웹 애플리케이션 형태로 실행되어 간단하게 배포할 수 있습니다. (Anaconda 환경에서 사용 권장)

```python
def main():
    st.title("2022-2학기 HAI 1팀 프로젝트")
    st.subheader("표준어-경상도 방언 번역기")
    st.markdown("""
        #### Description
        - 이 웹 애플리케이션은 PyTorch와 Transformers 라이브러리를
        - 모델 서빙 및 현재 보여지는 웹 페이지는 Streamlit을 활용하
        """)

    class_model, class_tokenizer = load_classification_model()
    trans_models = load_translation_models()

    options = ['표준어', '경상도 방언']
    input_options = st.selectbox("입력 언어", ['자동'] + options)

    message = st.text_area("발화 텍스트 입력", "여기에 입력")

    output_options = st.selectbox("목표 언어", options)
```

```python
if st.button("번역하기"):
    st.subheader("번역 결과")

    init = -1
    target = 1
    for i in range(len(options)):
        if input_options == options[i]:
            init = i
        if output_options == options[i]:
            target = i

    if init == -1:
        init = classification(class_model, class_tokenizer, message)
        st.success("입력 텍스트 자동 분류: " + options[init])

    same = False
    if init == target:
        same = True
    elif init == 0 and target == 1:
        selected = trans_models[0]
    elif init == 1 and target == 0:
        selected = trans_models[1]

    if same:
        translation_result = message
    else:
        translation_result = translation(selected[0], selected[1], message)

    st.text_area("", translation_result, label_visibility="collapsed")
```

# 번역 기능 테스트

- 입력 언어를 자동으로 설정하면 미리 만들어 두었던 방언 분류 모델이 자동으로 경상도 방언이라는 것을 인식합니다.
- 인식한 결과를 바탕으로 **[경상도 방언 -> 표준어]** 번역 모델에 입력한 결과를 출력하는 것을 볼 수 있습니다.
- 실제 데모 페이지가 운영되고 있습니다. http://girinman.ddns.kr:8501로 접속해서 직접 테스트 해보세요!
- 완벽하지는 않지만, 표준어-경상도 방언 번역 기능이 나름 잘 동작하는 것을 볼 수 있습니다!

# 4가지 지역 방언이 모두 사용 가능하도록 고치기

- 소스 코드와 데모 페이지를 보면 알겠지만, 지금 만들어져 있는 프로토타입은 표준어와 경상도 두 가지 방언 종류만 번역이 가능해요.

- 입력된 언어를 분류하는 모델도 binary classification을 수행하고 있고, 번역을 수행하는 모델도 [표준어 <-> 경상도 방언] 작업만 지원해요.

- 저번 시간에 성능을 끌어올렸던 multi-label classification 모델을 적용하고, huggingface에서 다른 지역 방언을 번역할 수 있는 모델들을 찾아와 적용해서 우리의 사투리 번역기가 지원 가능한 방언의 종류를 늘려 보도록 합시다!

- Streamlit 애플리케이션을 구현하는 app.py의 소스 코드를 잘 분석하고 수정해서 다음 주 마지막 회합 시간에 발표할 수 있도록 완성본을 만들어 보아요...

# 슬슬 끝이 보이는 것 같기도 하고...?

- 9월 초부터 열심히 함께 정신없이 달려온 프로젝트의 결과물이 눈에 보이는 것 같네요! 우리가 지금까지 해본 일들을 정리해보자면...

- **자연어 처리의 개념**과 이 분야에서 다루고 있는 여러 문제들(**텍스트 분류, 기계 번역, 텍스트 요약 등등..**) 그리고 최신 자연어처리 알고리즘과 **모델**들에 대해 알아보는 시간을 가졌어요.

- 자연어 처리 작업을 수행하는 모델을 학습시키기 위해 **데이터 준비**부터 **학습 및 추론**, 성능 향상을 위한 **하이퍼파라미터 조정**까지 같이 해 봤어요.

- 완성된 모델과 기존에 누군가 학습해서 업로드해 놓은 모델을 Streamlit을 활용하여 누구나 쉽게 사용할 수 있도록 웹 서비스 형태로 만들어 보았어요.

- 함께 공부하느라 고생 많았어요! 여러분도 이제 어엿한 NLP 엔지니어가 되었습니다!