

오픈소스SW기여

**테스트 계획**

**박찬호 32181928**

**허찬용 32184939**

***Easy Diffuser***

https://github.com/Easy-Diffuser

**Contents 목차**

**Ⅰ 테스트 방법 01**

**Ⅱ 테스트 코드 02**

일반적인 테스트 방법 01

단위 테스트 01

통합 테스트 01

ML 모델 02

크롬 익스텐션 03

**Ⅰ 테스트 방법**

일반적인 개발 상황에서 주로 사용되는 테스트는 단위 테스트와 통합 테스트이다.

단위 테스트는 메소드나 클래스 정도의 작은 동작 단위에서 구현 검증을 수행한다. 메소드 단위에서, 단위 테스트는 메소드의 호출에 따른 input과 output을 검증한다. Input에 맞는 output이 도출된다면 내부 구현에 문제가 없다고 가정한다.

통합 테스트는 구현된 메소드와 클래스 뿐만 아니라, 외부 라이브러리를 포함하여 전체 코드를 통합하여 검증한다. User input이 여러 메소드와 라이브러리를 거쳐 최종 output을 잘 도출하는지 확인한다.

**단위 테스트**

우리 서비스에서 단위테스트는 적용하기 어렵다. 첫째로, 머신러닝 모델의 경우 input과 output이 특정한 로직을 거쳐 도출되는 것이 아니라, 모델의 학습에 따른 결과를 도출한다. 또한, input과 output의 형태가 정형화된 request / response 형태가 아니라 이미지 또는 텍스트 토큰이라는 문제가 있다. 따라서, 머신러닝 모델에서의 단위 테스트는 적용하지 않는다.

크롬 익스텐션의 경우 일반적인 웹 개발과 유사한 점이 많아 일견 단위 테스트를 적용하기 쉬울 것으로 보인다. 그러나, 실제로는 chrome api를 호출하고, 단위에서의 동작이 추상화된 로직으로 감추어져 단위 테스트에서의 input과 output을 정의하기 어렵다. 따라서, 크롬 익스텐션에서도 단위테스트는 적용하지 않는다.

**통합 테스트**

현재의 로직을 다시 점검해보면 통합 테스트를 적용하는 것이 여러 장점이 있음을 알 수 있다. 현재의 로직은 유저 🡪 크롬 익스텐션 🡪 ML 모델 🡪 크롬 익스텐션 🡪 유저 🡪 크롬 익스텐션 🡪 WebUI (외부 서비스) 로 이뤄진다. 즉, 유저는 크롬 익스텐션을 통해 ML 모델에 접근하고, 이 결과를 확인 후 다시 크롬 익스텐션을 통해 WebUI에 결과를 보낸다. 그런데, 크롬 익스텐션의 동작은 실제로는 ML 모델의 api 호출 정도의 간단한 역할임에도, chrome brower에 직접 조작을 가해야 하므로 자동화된 테스트는 어려움이 있다. 이러한 이유로, 유저와 크롬 익스텐션을 로직에서 분리하여 테스트 코드를 구성하면, 자동화된 코드로 학습 및 추론 결과를 테스트 할 수 있다. 이 경우 로직은 다음과 같다. 자동화 코드 실행 🡪 ML 모델 호출 🡪 WebUI 전달 🡪 생성.

이 경우 크롬 익스텐션은 별개의 테스트를 적용한다. 코드를 이용한 통합 테스트를 적용하지 않고, 사용자의 input에 따른 테스트를 사용한다.

**Ⅱ 테스트 방법**

**ML 모델**

ML 모델은 자동화된 코드로 테스트를 진행한다.

기본적인 모델은 아래와 같다. BERT모델을 사용하고 pretrain된 모델을 가져온다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Img2Txt 코드는 아래와 같다. 먼저 이미지가 들어오면 Caption 형식으로 바꿔줘야 한다.

이미지 파일을 local에서 열거나 링크로 열어준 뒤 이미지를 모델에 맞게 변환을 해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Unsqueeze(0)를 통해 1인 차원을 생성한다.

변환된 이미지를 evaluate()함수에 넣어 이미지와 Tagging된 단어 중 가장 적합한 단어를 찾는다.

순서에 맞게 단어를 찾아주고 caption이라는 list에 넣어준다.

Prediction\_neg는 임계치 -90보다 큰 값 중 가장 작은 값을 가져와 neg\_prompt에 넣어준다. 여기서 -90이라는 값은, 우리가 사용하는 모델이 가지는 Image Encoder – Text Encoder pair에서 급격하게 연관성이 떨어지는 지점을 threshold로 잡은 것이다. 이렇게 처리한 이유는, 최저 값은 아예 엉뚱한 단어를 도출하고, 말이 되는 단어 중 가장 낮은 값을 가지는 단어를 선택해야 실제 이미지 생성에 도움이 되는 negative prompt를 생성할 수 있기 때문이다.

이 부분을 caption\_neg에 저장한다.

텍스트, 실내, 블랙, 스크린샷이(가) 표시된 사진

자동 생성된 설명

문장이 끝났다는 토큰이 전달되면 해당 함수 종료 뒤 pos\_caption과 neg\_caption을 return 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

모델에서 나온 caption을 이용하여 webui api가 이미지 생성을 하도록 코드 작성한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이러한 과정을 통해, WebUI를 포함하여 통합 테스트를 진행할 수 있다.

통합 테스트의 결과는 두 가지로 평가할 수 있다.

1. Inception score를 이용한 정량화

Inception Score는 생성 모델의 결과를 평가하는 척도이다. 이는 아래의 코드를 통해 도출할 수 있다.

**>>> import** **torch**

**>>> \_** **=** **torch.manual\_seed(**123**)**

**>>> from** **torchmetrics.image.inception** **import** **InceptionScore**

**>>> inception** **=** **InceptionScore()**

**>>>** *# generate some images*

**>>> imgs** **=** **torch.randint(**0**,** 255**,** **(**100**,** 3**,** 299**,** 299**),** **dtype=torch.uint8)**

**>>> inception.update(imgs)**

**>>> inception.compute()**

(tensor(1.0544), tensor(0.0117))

그러나 기존의 논문들이 지적하듯, inception score는 diversity를 제대로 반영하지 못하고 실제 사람이 느끼는 semantic한 특징을 제대로 읽어낼 수는 없다.

2. Input output 결과 비교.=

따라서, 사람이 느끼는 semantic한 특징을 읽어내기 위해 직접 input output을 비교한다. 테스트 코드를 사용하여 충분한 반복을 시행한 후, 유사도를 직접 매기고 이 결과를 평균 내어 모델의 결과를 평가한다.

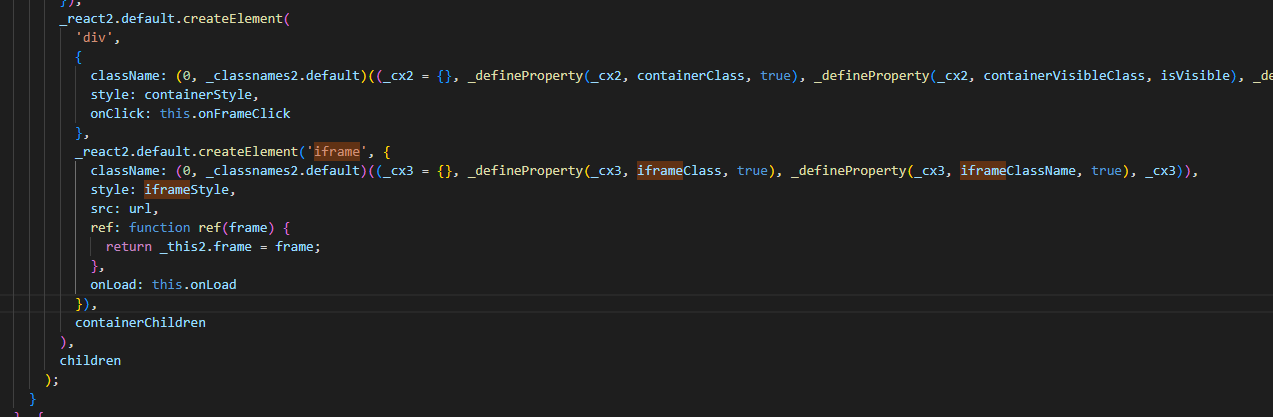
구현 및 학습과정에서 위의 두 테스트로 얻어지는 점수를 향상시키는 것을 목표로 한다.

**크롬 익스텐션**

앞서 설명한 것과 같이, 크롬 익스텐션은 코드를 통한 테스트를 적용하는 데에 어려움이 있다. 따라서, 사용자의 동작에 의한 테스트를 적용해야 하는데 이를 위해서는 UI 파트의 구현이 필수적이다. 실제 API를 호출하는 영역은 ML 모델의 테스트에서 다뤄지기 때문에, UI 파트의 동작이 보증된다면 서비스 배포 직전의 시나리오 테스트 과정에서 크롬 익스텐션의 기능이 충분히 검증된다고 생각할 수 있다. 따라서, 크롬 익스텐션의 UI 구현을 먼저 수행하였다.

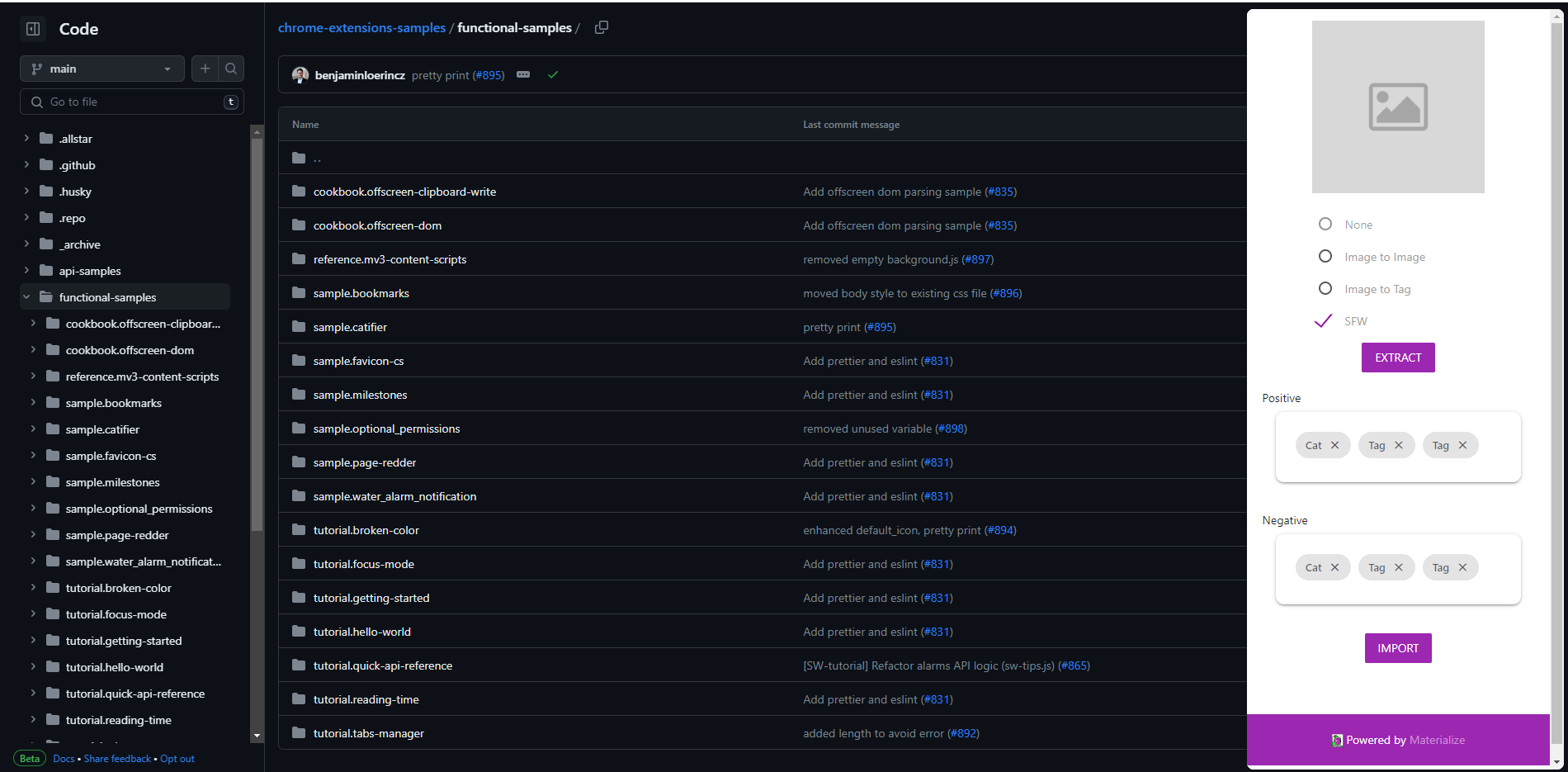
크롬 익스텐션은 manifest v2 기반으로 구현되며, https://github.com/segmentio/chrome-sidebar

를 boilerplate로 하여 구현하였다. 위의 라이브러리는 iframe 내부에 url을 받아 불러오는 방식으로 동작한다. 최초 구현 시에는 iframe을 부르는 코드를 직접 고쳐 react app으로 대체하려고 시도했으나 이러한 방식은 의존성과 에러 문제가 있었다.

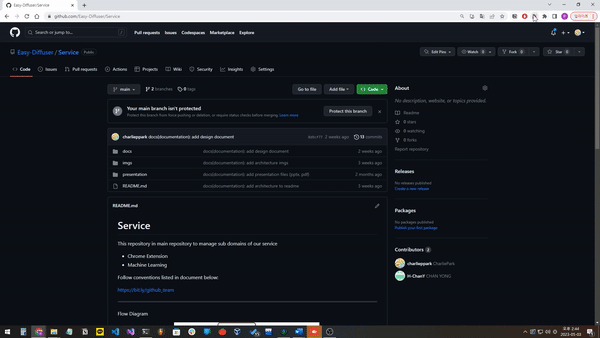


따라서, 최종적으로 boilerplate 외부에 추가로 Single Page 서비스를 만들고, 이를 호스팅 하여 사이드바와 연결하였다. UI는 material 테마를 사용하였고, materialize css를 사용하여 구현하였다.

그 결과는 다음과 같다.



현재의 동작은 아래와 같다



크롬 익스텐션의 검증을 위해 추가로 구현되어야 하는 부분은 아래와 같다.

1. 모델 api 호출
2. Tag 제거 되돌리기.