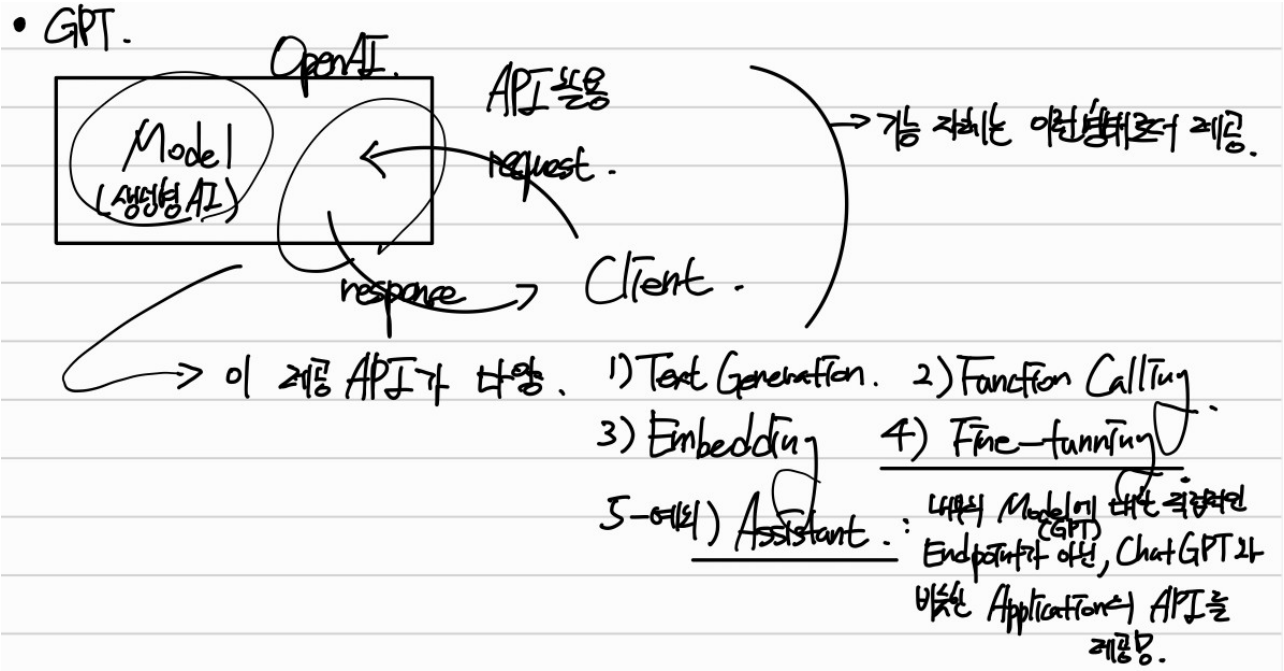
LLM 개발 환경 구축 (GPT, LIama, Gemini, HyperClovaX)

1. GPT

- OpenAI. Documentations - Get Started, Capabilities, Assistant 를 통해 다음과 같은 내용을 확인할 수 있음.



- 즉, Model 을 활용할 수 있는 API 가, 일반적인 답변 생성, Fine-Tuning, Assistant 까지 제공되므로, 우리가 개발 환경을 별도로 구축할 필요가 없음.
  - Model 을 사용할 수 있는 API 에 접근하여 사용해주기만 하면 됨!

Q. fine-tuning 이란 무엇인가? (위 링크의 내용과 동일. 한번 더 언급.) : A. 기존에 사전 학습된 머신러닝 모델을 특정 작업이나 데이터셋에 맞게 추가적으로 학습시키는 과정을 의미.

- GPT 가 통상적인 방법으로는 Model 을 직접 제공하는 것이 아닌, Fine-Tuning 만 제공하므로, 이를 기준으로 살펴보는 것이 좋을 듯 함.
- *Q1. 인공지능에서 학습이란 무엇을 의미하는가? => A. 기계가 데이터를 통해 패턴을 인식하고, 이를 바탕으로 결정을 내리거나 예측을 하는 능력을 개발하는 과정. 데이터 수집 및 처리, 모델 훈련 (모델(알고리즘)이 데이터에서 패턴을 찾는 과정), 학습된 모델의 평가 및 조정, 예측 또는 의사결정의 과정을 거침.*
- *Q2. Fine-Tuning 외에, 다른 학습 방법에는 무엇이 있는가? => A. 학습 : Training model from scratch (처음부터 학습) / Fine Tuning (기존 학습된 Model을 학습). 또한, Training model from scratch의 종류로, 지도학습, 비지도학습, 강화 학습 등이 존재.*

2. Llama (Open Source LLM의 대표라 생각하고 찾기.)

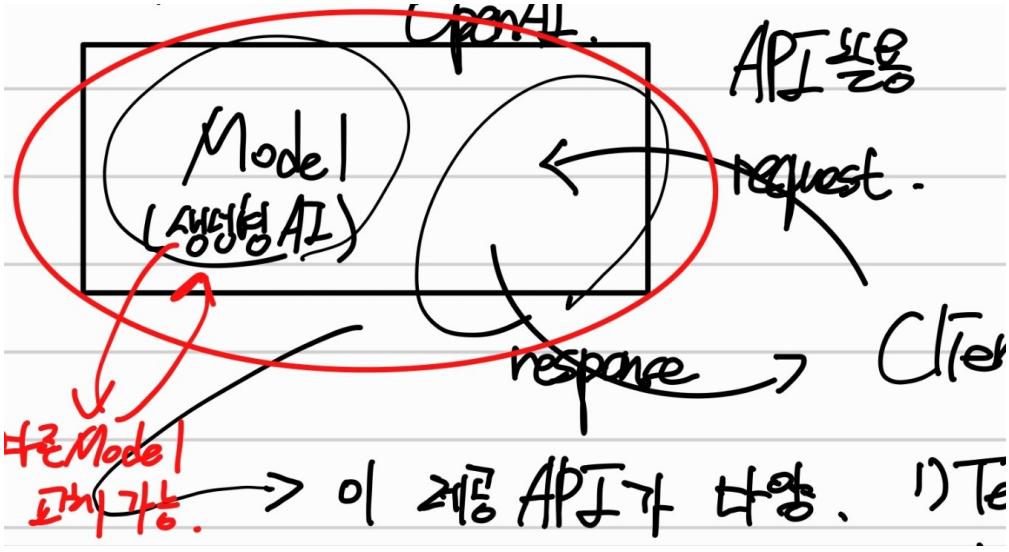
1. Open Source LLM 개발 환경 구축 탐색

- Llama 는 Open Source Model 로서, Model 의 형태로 직접 제공됨. 이에 따라, 내가 Llama 관련 개발, 사용 환경을 구추갈 때 가장 중점적인 요건은 다음과 같았음.
    1. Unity Client 로 API Request 등을 요청하여 Chat 기능을 사용할 수 있어야 하므로, API Server 로서의 구축 혹은, Cloud 적인 특성을 가지고 있어야 함.
    2. LLM Model 을 갈아낌으로써, 쉽게 다른 LLM Model 을 사용할 수 있어야 함.
- 이에 따라, 고안하거나 예상할 수 있는 방법은 다음과 같았음.
    1. Cloud 활용 : API Server, 다른 LLM Model로의 전환 이라는 주요 기능이 이미 구축되어, 이를 Service 로서 제공하는 Cloud 를 찾을 수 있는 것이 제일 베스트.
    2. From Scratch : Model 을 사용할 수 있는 환경을 구축. (예상했던 대로, 다른 Model 로 갈아낄 수 있도록 확장성 있게) 이후, 해당 내용을 기반으로 API Server 연결 및 제작.
- 1번이 더 쉽고 빠르게 만들 수 있음. 2번은 학술적으로나, 개발적으로 유의미한 성과를 가질 가능성이 높지만, 제한 시간 내에 개발하는 것이 불가능할 가능성이 큼.

1. Cloud 활용

> 아무리 찾아도 안 떠서 조금 의아했었는데, 역시 지속적으로, 제대로 탐색했을 때 많은 방법이 존재함.
>
> - <"Platform 이름" open source llm api> 라고 검색 시, 모두 찾음.

- 아래 내용 사용할 시 해당 내용까지는 쉽게 만들 수 있을 듯.



1) Google

- Serve open-source LLMs on Google Cloud

2) Azure

- MS, '애저 AI 스튜디오' 대폭 확장...오픈 소스 LLM 40개 추가 < 산업일반 < 산업 < 기사본문 - AI타임스 (aitimes.com)
- 시작 - Azure AI | Machine Learning 스튜디오

> Azure 에서, Model 들이 Chat Completion 과 Text Generation 이 갈리던데 둘의 차이?
>
> - Chat Completion : 대화 중에 주어진 문장을 기반으로 대화를 이어가는 작업.
> - Text Generation : 주어진 프롬프트, 시작 텍스트 바탕 새로운 텍스트를 생성하는 작업.
> - 즉, 문맥 기반 예측 (연속성) 이냐, 주어진 프롬프트에 따른 생성물이냐. 이것만 다름.

3) Hugging Face

- Open-Source Text Generation & LLM Ecosystem at Hugging Face > Tools in the Hugging Face Ecosystem for LLM Serving > TGI
- TGI 는 A Rust, Python and gRPC server for text generation inference 이다.
- Hugging Face, Inference Endpoints, and Inference API 에 통합되어 있어, API Endpoint 로 해당 내용을 제공하며, HuggingFace 안에 있는 LLM Model 들을 사용하여 구축할 수 있다.

> Hugging Face 가 제공하는 OpenSource Model 의 수가 압도적. 또한, 한국어용 LLM 들의 경우, Open Ko-LLM Leaderboard - a Hugging Face Space by upstage 을 통해, 제공되고 있으므로, 해당 내용을 사용하는 것이 훨씬 효과적일 것이라는 생각임.

2. From Scratch

1. 🤗 Transformers (huggingface.co) 를 이용하여 NLP Model 을 가져와 사용.
    - *NLP 란? : 컴퓨터가 인간의 언어를 이해항도록 하는 자연어 처리의 한 분야. (01. 자연어 처리(natural language processing) 준비하기 - 딥 러닝을 이용한 자연어 처리 입문 (wikidocs.net))*

    - *Transformer 란? : NLP 작업에서 사용하는 심층 모델 구조 중 하나. Attention 등에 집중함. (16. 트랜스포머(Transformer) - 딥 러닝을 이용한 자연어 처리 입문 (wikidocs.net))*
    - GPT 과 같은 최근 Chat 을 위한 Model 의 경우, Transformer 를 기반으로 만들어져 있기 때문에, 해당 Model 을 사용하기 위해서 HuggingFace Transformer 를 이용하는 것.
2. Text 생성 등 필수 기능을 만들거나 한 후, Flask 혹은 FastAPI 등을 사용하여 API 화.
    - Flutter, Fast API 에서 LLM API 를 활용하기-2 (Bard-API vs OpenAI-API vs kakaoTrans)
3. 이후 해당 내용을 Docker 를 이용하여 Container 화 및 배포(Azure 등 이용해서 Publish 하면 됨).
    - Docker : 여러가지를 의미함.
        - 컨테이너를 만들고 사용할 수 있도록 해주는 기술.
        - 빌드, 배포, 실행, 업데이트 등 가능한, 오픈 소스 플랫폼.
    - Containerization : 어플리케이션과 그 어플리케이션이 필요로 하는 라이브러리 및 설정을 하나의 패키지로 묶는 과정을 의미함. 이를 기반으로 만든 하나의 묶음을 Container 라고 함.

3. 결론

1. "1. Cloud 활용" 방법 사용.
2. "Hugging Face" 를 활용할 것.
    - 세부적 환경 구축과 관련하여서는 "Open-Source Text Generation & LLM Ecosystem at Hugging Face > Tools in the Hugging Face Ecosystem for LLM Serving > TGI"를 활용할 것.
    - Open Ko-LLM Leaderboard : Open Ko-LLM Leaderboard - a Hugging Face Space by upstage : Model 은 해당 Model 들을 활용하고자 함.

- 즉, 추가 탐색 필요 내용은 다음과 같을 듯.
  1. TGI - Model 등 기반으로 해서 어덯게 만들어 두고 어떻게 쓰는지.
  2. Open Ko-LLM Leaderboard 분류가 어떻게 되고 뭘 써야 하는지.

2. TGI 구축 및 사용법

- HuggingFace. Text Generation Inference 구축 및 사용법 참조.

Spaces

> 위 링크 내의 서술에서는, HuggingFace 의 TGI 를 기반으로 구성한 Container 를 배포하는 방법으로서, AWS, Azure 같은 친구들만 논의 됨. 하지만, Open-Source Text Generation & LLM Ecosystem at Hugging Face : 이 내용의 아랫부분 확인했을 때 Docker template for HuggingChat 이 released 되었으며, 누구나 their instance based on a large language model을 배포할 수 있다 함. 따라서, Spaces 관련 내용을 조사 및 정리하고자 하였음.

- HuggingFace. Spaces 정리 참조.

> 24.01.27 어후; 생각보다 스터디 및 정리 과정이 더더더 빡세다;

3. Open Ko-LLM Leaderboard 분류 및 사용

- 해당 항목 내 평가 지표

평가지표	주요내용
추론능력 (ARC)	· (ARC, AI2 Reasoning Challenge) AI가 질문에 대한 답변이 얼마나 적절한지를 측정 ※ 초등학교 수준의 과학 질문지로만 구성
상식능력 (HellaSwag)	· (HellaSwag) AI가 짧은 글 및 지시사항에 알맞은 문장을 생성하는지 여부 측정 ※ 인간에게는 사소한 질문이지만, AI에게는 답변하기 어려운 질문지로 구성
언어이해력 (MMLU)	· (MMLU, Massive Multitask Language Understanding) 방대한 분야의 질문에 대한 답변이 얼마나 정확한지를 측정 ※ 57개 다양한 분야(초등 수학, 역사, 컴퓨터 과학, 법학 등)에 대한 질문지로 구성
환각방지능력 (TruthfulQA)	· (TruthfulQA) AI가 생성한 답변이 얼마나 진실한지 측정 ※ 인간이 잘못 인지 혹은 거짓으로 대답할 수 있는 질문지로 구성
한국어 상식생성능력	· (Korean-CommonGEN-V2) AI가 주어진 조건의 질문에 대한 답변이 한국어 사용자라면 보유하고 있을 일반 상식에 부합하는지 여부 측정 ※ 역사 왜곡, 환각 오류, 형태소 부착 오류, 불규칙 활용 오류, 혐오 표현 등에대한 광범위한 유형을 포함한 질문지로 구성 (고려대 임희석 교수 데이터 구축)

  - 지금 용도에서는 추론 능력(Ko-ARC), 한국어 상식 생성 능력(Ko-CommonGen) 이 둘을 가장 신경쓰는 것이 나을 것으로 보임.
- ex - Model 명이, "hyeogi/SOLAR-10.7B-dpo-v1"라 할 때, 앞은 만든 사람, 회사, 조직. 뒤는 기반이 된 Model 혹은 Model 이름인 것으로 보인다.
  - 해당 모델의 경우, SOLAR 라는 모델을 사용했는데, Upstage - AGI for work 에서 만든 소규모 매개변수 모델인 것으로 보인다.
  - 뒤에 있는 10.7B 등은, 해당 모델을 학습하기 위해 사용한 매개변수.
    - *매개변수란? : Transformer Architecture 에서 사용되는 Attention 이라는 개념이 있음. (문맥에서 집중되는 관계성) 학습에서 사용하는 이 Attention의 수를 매개 변수라 함.(대규모 언어 모델이란?)*

궁금한 건 아니고 개인 의견 : Upstage 확인했을 때, 업스테이지(AI PACK) - 제품/서비스 3개 - THE VC 결국 제공하는 기능은 두 가지임. Solar(자체 개발 LLM 모델)와 아숙업(Solar 기반 (2024.1.11 기준으로 LLM 변경됨.) ChatGPT 처럼 Chat 가능한 친구). 그런데 Solar Model 은 사실상 Open Source 고. 아숙업은 카카오톡 챗봇인 만큼 따로 판매하거나 하는 것이 아닌 그냥 사용할 수 있는 친구임.

- 이때 오는 궁금증은 이거임. **이익을 어디서 얻으려고 하는 건가?** OpenAI는 GPT Model 을 소유하고 있으며, 이를 기반으로 한 ChatGPT 서비스, ChatGPT 기반의 API 호출 서비스를 바탕으로 그 수익성을 유지하고 있다.
  - 또한, Microsoft 와의 협업으로 Azure 내 API 호출 등이 가능하도록 되긴 했지만, Assistant 와 같은 강력한 기능의 경우, 결국 OpenAI 내에서만 사용할 수 있다. 이런 차별점을 가지고 있다.
- 하지만, Upstage 의 경우, 모델을 독점하고 있지도 않다. (Solar 는 OpenSource) 이에 따라서 Upstage 에서 만드는 API 등의 내용의 경우, 따로 독자적인 역량등을 구축하지 않는 이상, 다른 기업들도 모두 따라할 수 있다. 또한 그렇다고 해서, 이를 기반으로해서 기똥찬 LLM 이용 Service (Client) 를 제공하는 것도 아니다. 아숙업 하나만 제공하고 있다.
  - 수익이 지금 투자 말고 1도 없고. 경쟁력도 스스로 낮춰 가진다.
  - 제품의 목적 을 기반으로 보자. 사용자의 수도 딱히 안 많고LLM 모델이 주요라, 개발자 대상임.), 사용자와의 연결성을 공고히 하는 것도 떨어진다. (Model 을 공개했으므로, 수요층은 많지만, 공급층이 늘어날 가능성이 높다.)
  - 가지고 있는 게 **역량**만 있지 않나.
  - 혹은 Llama2 기반으로 해서 Fine Tuning 으로 어떻게든 얻고, 결국 이를 기반으로 했기 때문에, 공개할 수 밖에 없었다 하면, 기업 가치가 정말 별로 많이 높지 않나 싶긴 하다.
- 아이디어 : 오히려 LangChain 인데, Server 로 동작해서 사용하기 쉽게 만드는 게 더 도움이 되지 않을까? 모델 갈아끼거나, Gemini, GPT 등 다른 것도 API Key 등 넣으면 바로 사용할 수 있고, 근데 Server 로 배포해놔서 사용자체는 일반적인 API 호출만 하면 할 수 있도록 할 수 있는!
  - 이거 만들어도 재밌을 것 같긴 하네.

4. 실전 환경 구축 - TGI

이전 탐색 자료 정리

- 아예 좋았던 강의 / 내용들임.
  - 딥 러닝을 이용한 자연어 처리 입문 - WikiDocs
  - 생성 AI 활용기 - WikiDocs
  - 강의 소개 - Hugging Face NLP Course

1. TGI 구축

- Solar 를 사용할 예정이기 때문에, Solar TGI 처럼 칠 필요 있음.
- 구축하는 것은 아마 어렵지 않을 것. 구축으로 인해 나오는 결과물은 Docker Container.
- 현재 Spaces 를 기반으로 배포하려 하나, 이거 지금 확인했을 때, 문제있을 가능성도 있어보임. 유의해서 진행할 것.
  - 이유 : Spaces 랑 TGI 랑 별개의 개념으로 보이므로. 이를 따로 연결해줘야 할 가능성이 있을 수도 있을 것 같아서. (Open-Source Text Generation & LLM Ecosystem at Hugging Face - 이거 기반.)
  - 맞네. 확인해보니까, Hugging Chat 이랑 TGI 와의 연계 / Hugging Chat 과 Spaces 의 연계를 별도로 내세움. 그리고 내가 생각했을 때도, Spaces 자체는 배포만 담당하고, API Server 의 구축은 TGI 기반으로 해서 별도로 관리하는 게 합리적
    - 근데, Spaces 에는 또 처음 Model 을 입력하는 값이 존재하므로?
    - **될 가능성도 분명 존재한다는 말이지?** (Hugging Chat 자체가 TGI 를 이용해서 Powered 되기 때문에)
- TheBloke/Solar-10.7B-SLERP-AWQ · Hugging Face : 여기 Solar 이용했을 때, TGI 등 구축 자세히 설명되어 있어, 이거 함 살펴봐도 될 듯.
- 전략
  1. Spaces 부터 제작해보기.
  2. 만약 따로 TGI 가 내부에서 활용되지 않은 경우, TGI 를 따로 만들어, Docker 를 해당 Spaces 에 올리는 작업 해보기.

- 사용 모델 : hyeogi/SOLAR-10.7B-dpo-v1 · Hugging Face
- HuggingFace 비싸다 생각했는데. 전체적 사용량이나 호출에도 돈 내야하는 MS 보다, GPU 빵빵한 거 제공해주면서 어떤 호출하든 상관없는 HuggingFace 가 선녀라는 생각이 들기도 하고?

본문 1 : Spaces 를 ChatUI 기반으로 하여 제작.

- 과정이 굉장히 간단해서 그냥 했는데 안 됨.

문제 발생 1 : MongoDB URL 미 입력

- ChatUI on Spaces (huggingface.co)
- chat-ui/README.md at main · huggingface/chat-ui (github.com)
    - chat-ui/README.md / Database
- The chat history is stored in a MongoDB instance, and having a DB instance available is needed for Chat UI to work.
- 즉, ChatUI 를 사용하는 경우, Chat History 를 저장하기 위한 MongoDB Url 이 무조건적으로 있어야 함.
    - 근데 난 Chat Completion 만 사용해도 되는데?
    - 일단 간편하게 쓸 수 있으므로, MongoDB 의 Free tier 를 사용해보고자 함.
- *아 근데 MongoDB Cloud 에서 쓰려면 이거 Access Control 해야하네.* => 좀 복합적인 문제다. 솔~직히 여기 부분은 하나도 모르겠음!

본문 2 : TGI 를 따로 만들기. Spaces Docker Blank에 올리기.

- TGI 를 만드는 건 따로 문제가 없을 것 같긴 해! 문제가 되는 건, 올리는 것!
- Docker Deploy 관련
    - Best Platform for Python Apps Deployment - Hugging Face Spaces with Docker (youtube.com)
    - Docker Spaces Examples (huggingface.co)
    - SpacesExamples/fastapi_t5 at main (huggingface.co)
    - SpacesExamples/fastapi_t5 at main (huggingface.co)
    - Explore Llamav2 With TGI - a Hugging Face Space by ysharma
- **Docker Deploy 가 꽤 난항이었음. 또한, TGI 내부적으로는 OAuth 등 인증을 처리하는 부분이 없어, Docker 자체적으로 이를 추가해주어야 한다고 판단. 이게 좀 많이 골 때렸.**
- **TGI 는 Docker 를 만들어서 제공한다면, 애초에 🤗 Inference Endpoints (huggingface.co) 는 Docker 및 배포 필요 없이, API Endpoints 를 제공하는 구나! 하,,,,**

문제 2 : Docker 및, Spaces 에 대한 이해 부족

- Docker 로 인해 Container 라는 File 이 나오게 되고, 이를 upload 하면 되는 것이라고 생각했었는데, 이해 자체를 꽤나 잘못하고 있었음.

- 바로 잡은 내용은 HuggingFace. Text Generation Inference 구축 및 사용법 > 이해 요약 및 정리 에 서술해 둠. Docker 에 대한 자세한 내용은 Docker 란 무엇인가 에 정리.

본문 3 : Docker Space 제작 및 TGI 구성, 이후 Test.

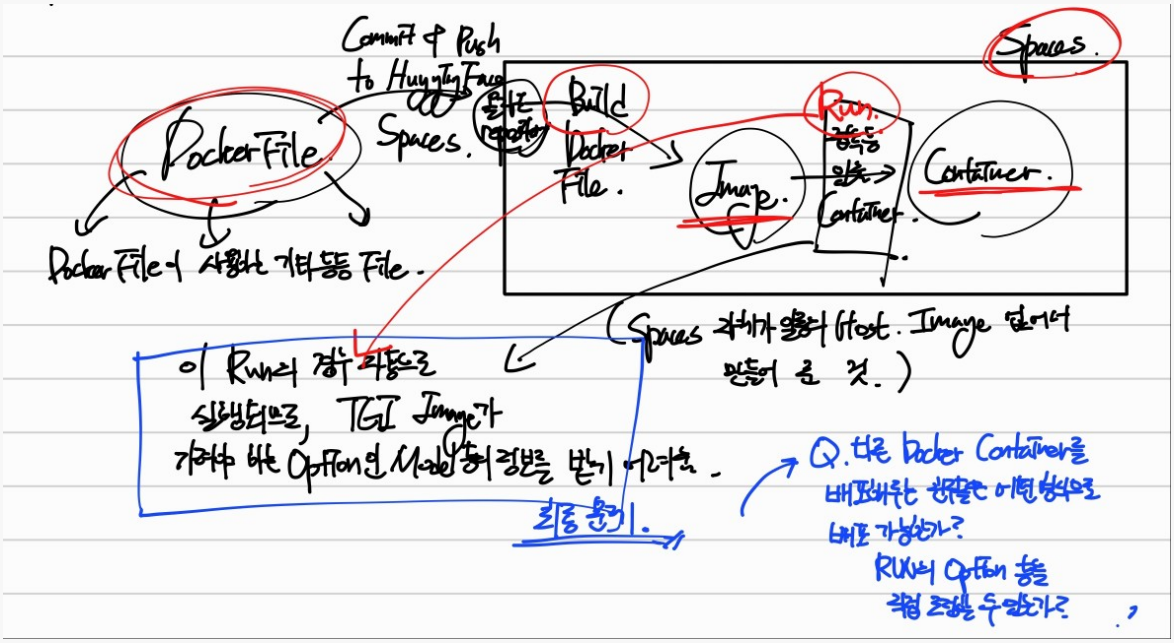
> Blank Docker Space 만들고, 거기다가 명령어 직접 입력해 볼 거임. 그리고 해당 URL (어떻게 설정되어 있는지 확인할 필요 있을 듯.) 에 요청 보내서 Docker Container 가 정상적으로 작동하는지 확인.
>
> - URL 확인해야 함.
> - Request 예시가 현재 존재하지 않으므로, 해당 내용도 추가로 개발해야 함.
> - Docker 를 구성할 경우, 해당 원격 컴퓨터에 접속하는 OAuth 나 인증을 어떻게 해야할지 약간 의문스러움. (Private 로 구성하면 진짜 안 될 수도 있을 것 같은데.)
>   - 아마 Docker 자체적으로 OAuth 나 인증을 구성하는 절차가 존재할 것으로 보임.

1. 일단 Blank Docker Space 를 만들었지만, 따로 명령어를 입력하는 칸이 존재하는 것이 아닌, Dockerfile 을 구성하라는 내용만 뜸.
   - 내가 갖고 있는 것 : Docker Image URL, Image 기반으로 Container 를 어떻게 구축하는지에 대한 지식. 그럼 이 둘을 기반으로, HuggingFace Spaces 에서 Image 기반 Container 를 어떻게 실행시키는지만 알면 되겠네?
   - How to deploy a Panel app to Hugging Face using Docker? (youtube.com) : app.py, requirements.text, dockerfile 을 구성해야 하는 것으로 보임.
- *Q. Docker 파일이 뭐냐? => A. Docker 란 무엇인가 에 함께 기록.* / *Q. Space 구성 할 때 작성해줘야 하는 세 가지는 무엇을 의미하는가? => HuggingFace. Spaces 정리 에 추가적으로 기록.*
2. DockerFile 을 구성. 어차피 Build 랑 Container 화 같이 실행됨. CMD 부분에다가 내가 넣고 싶은 내용 짚어넣으면 될 것.

전략 :

1. TGI 에 Docker 내용이 포함되어 있었으므로 전체 복사해서 내 Repository 에 집어 넣음.
2. DockerFile 이 해당 Repository 에서 자동 Build 및, Build 된 Image 파일을 실행한다는 것을 확인했으므로, CMD 부분에 model 관련 부분을 집어넣음. => 아 근데 이건 안 될 가능성이 높은데. => *DockerFile 을 올리고 난 후, Build 와 Build 된 Image 를 기반으로 한 Run 이 자동으로 처리가 되므로, 발생하는 문제임.*

- ChatUI 는 아마 Internet 을 통해 받은 Model 및 기타 등등 바탕으로, 해당 내용 내부에서 Customizing 하는 게 있을 듯. *안 될 가능성이 너무 높다.*



노선 전환

- 기존 노선 : Spaces 활용을 위해 DockerFile 등에 존재하는 Configuration 등을 변경해보기
  - text-generation-launcher 이용. DockerFile 을 다르게 구성하여 model 이름까지는 지정이 가능할 것으로 보이나, 나머지 RUN 설정의 경우, DockerCompose 를 활용해야 하거나, 해도 안 될 가능성이 큼.

```
FROM ghcr.io/huggingface/text-generation-inference:1.4

ENTRYPOINT ["/usr/bin/env"]

CMD ["text-generation-launcher", "--model-id", "{원하는 모델 이름}"]
```

- **변경 노선 : TGI 사용부터 Deployments (외부 URL, Server 로 배포) 까지 있는 Tutorial 하나를 그냥 새롭게 파기.** 그게 더 효과적일 것으로 보임.

2. TGI 구축 및 배포 (새마음 새 뜻으로)

> Deploy Your Private Llama 2 Model to Production with Text Generation Inference and RunPod (youtube.com)

기반 지식 정리

- *RUNPOD 는 뭐하는 사이트?* : GPU 기반 클라우드 컴퓨팅 서비스 제공 사이트.
  - Pods Overview : RUNPOD 가 제공하는 PODs 는 running container instances 들임. 바로 내가 RUN 해준 Container 를 Hosting 해줄 수 있다는 점이, 이 구조가 가지는 강점.
    - 궁금한 거 더 있으면, 해당 Documentation 확인하면 될 듯!
  - *클라우드 컴퓨팅?* : 원격으로 컴퓨팅 자원을 제공, 혹은 원격 컴퓨팅 자원을 이용하는 서비스(사람들이 더 쉽게 쓸 수 있도록).
- *Colab* : "Colaboratory"의 준말. Google이 제공하는 무료 Jupyter Notebook 환경임.
  - Jupyter Notebook을 돌릴 수 있는 원격 컴퓨터와, Browser 에서 사용할 수 있는 Jupyter Notebook 을 제공해주는 친구.
  - Colab 사용법 좀 익힐 필요 있을 듯 : Colab 사용법

과정

1. RUNPOD 가입 및 결제. API Keys 받아두고, RUNPOD를 Client에서 쓸 준비.
2. Colab Notebook 에서 POD 를 만들어 주는 Python 코드 실행.
   - **이것까지 진행 완료 했음! 잘 된다!**
3. 이후 해당 Container 에 접근하여 바로 Request 호출 및 이용해주기만 하면 됨!

- Q. Request 호출 시 인증 및 보안과 관련해서는 어떻게 처리되나?
- => A. RUNPOD 를 통해서 Pytorch 를 만들거나, 내가 코딩하는 장소로서 사용하는 경우 SSH Public Key 를 따로 만들어서, 이를 해당 POD 에 등록함으로써 인가받는 방식으로 사용하는 것으로 보임. // 또한 사실 JWK (JWT Validating(공개 키, 개인 키, 암호화와 복호화, 서명 생성 및 검증 > 디지털 서명 생성 (Signing) / 디지털 서명 검증 (Verification))할 때 사용되는 비밀 키)를 받아오고, 이를 기반으로 Header 로 들어온 Authentication 의 JWT(OpenAI. API Key 숨기기 > 문제 해결을 위한 배경 지식 > JWT)의 무결성을 인증하여, 사용하는 것은, Docker로 이미 완성되어 배포된 곳에서 체크하기에는 아무래도 무리가 있긴 할 것이라 생각한다.
  - A. **인증과 관련하여서는,** 기존 사용하던 Proxy Server 를 이용해서, API Key 를 숨기는 것처럼, RUNPOD 의 URL 을 숨기는 방식이 지금으로서는 제일 가능성이 있지 않을까 싶다!
  - A. 근데 이것도 OPENAI 의 경우처럼, 직접적으로 Model 이 들어있는 은닉

4. POD 를 사용하지 않을 때는 Terminate(Delete) 해주어야 함! 그냥 STOP 만 하면, 계속 자원이 소모.

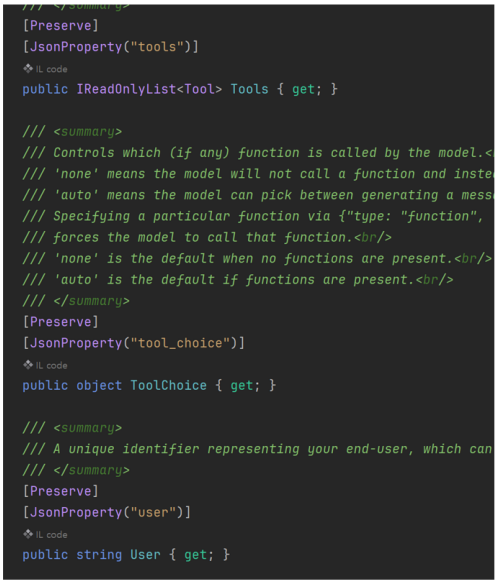
> 이 사람 꽤 잘 알려줘서 Fine Tuning 관련해서도 [Fine-tuning Llama 2 on Your Own Dataset | Train an LLM for Your Use Case with QLoRA on a Single GPU - YouTube] 이걸로 도움 받을 수 있으면 좋을 듯.

3. RUNPOD 에 배포된 Tgi 를 활용할 Client 코드 구축

- 기존 GPT Interface 를 기반으로, Tgi 를 사용할 수 있는 간단한 코드를 제작 및 배포.

문제 1 : `RestException: [POST:422]`

- 단, 이 경우 문제 발생. 내가 마주친 422 에러 (FastAPI) (velog.io)
- 422 Unprocessable Entity 로 볼 때, json 내부에 처리할 수 없는 entity 의 종류가 존재하여 문제가 발생한 것으로 보임.
- 실제 확인해 봤을 때도, Swagger API 내에 표시된 Schema List 와 ChatGPT 에서 포함하는 List 가 서로 다름.



  - TGI 에서 구성한 Swagger API 에 존재하는 문제. 이에 따라, Unity Client 내에서 해당 Request 를 호출하는 내용을 따로 만들어 줘야 할 듯.
- *이 문제 해결 못해서, 지금 엄 청 붙잡고 있었음!*
  - 보통 422 ERROR 의 경우, 일반적인 오류와 달리 bigscience/bloom · Getting HTTP Error Code: 422 when using Inference API (huggingface.co) 넣어준 json contents (parameter) 부분에 오류가 있어 발생하는 것이라는데...

C#

```
// 마지막 Version

using System.Collections;
using System.Collections.Generic;
using Newtonsoft.Json;
using OpenAI;
using OpenAI.Chat;
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.Networking;
using Utils;

namespace LLM.TGI
{
    public class Message
    {
        public string content;
        public string role;

        public Message(string inputRole, string inputContent)
        {            role = inputRole;
            content = inputContent;
        }    }
    public class ChatRequest
    {
        public double frequency_penalty;
        public List<float> logit_bias;
        public bool logprobs;
        public int max_tokens;
        public List<Message> messages;
        public string model;
        public int n;
        public double presence_penalty;
        public int seed;
        public bool stream;
        public double temperature;
        public int top_logprobs;
```

```
        public double top_p;
    }    public class TgiInterface : Singleton<TgiInterface>
    {        protected override void Awake()
        {            base.Awake();
            StartCoroutine(SendChatRequest());
        }
        public IEnumerator SendChatRequest()
        {            var url = "https://8iti6x1uan53v0-
80.proxy.runpod.net/v1/chat/completions"; // API 엔드포인트 URL
var request = new ChatRequest
            {
                frequency_penalty = 1,
                logit_bias = new List<float> { 0 },
                logprobs = false,
                max_tokens = 32,
                messages = new List<Message>
                {                    new Message("system", "You
are a helpful assistant."),
                    new Message("user", "Who won the world series
in 2020?"),
                    new Message("assistant", "The Los Angeles
Dodgers won the World Series in 2020."),
                    new Message("user", "Where was it played?"),
                },
                model = "tgi",
                n = 2,
                presence_penalty = 0.1,
                seed = 42,
                stream = true,
                temperature = 1,
                top_logprobs = 5,
                top_p = 0.95
            };

            var json = JsonConvert.SerializeObject(request);
            var postData =
System.Text.Encoding.UTF8.GetBytes(json);
            var webRequest = UnityWebRequest.PostWwwForm(url,
```

```
json);
            webRequest.uploadHandler = new
UploadHandlerRaw(postData);
                        webRequest.SetRequestHeader("Content-
Type", "application/json");

            yield return webRequest.SendWebRequest();

            if (webRequest.result ==
UnityWebRequest.Result.ConnectionError || webRequest.result ==
UnityWebRequest.Result.ProtocolError)
            {                Debug.LogError(webRequest.error);
            }           else
            {
                var jsonResponse =
webRequest.downloadHandler.text;
                var response =
JsonConvert.DeserializeObject<ChatResponse>(jsonResponse);
                var choice = response.FirstChoice;
            Debug.Log($"[{choice.Index}] {choice.Message.Role}:
{choice.Message} | Finish Reason: {choice.FinishReason}");
            }       }    }}
```

결론 - 실패.

- 422 Error 가 뜨는 것을 볼 때, 다음 문제 중 하나라고 추정할 수 있음.
    1. UnityWebRequest 로 보내는 Request 가 잘못되었을 경우.
    2. RUNPOD 에 올린 TGI Server 가 잘못되었을 경우.

1. UnityWebRequest 로 보내는 Request 가 잘못되었을 경우 : 내가 바꿀 수 있는 모든 변인을 다 시도해봤는데 안 됨. 주로 model 명, 인자의 수, JSON Encoding 등을 달리 해봤는데, 안 됨. 또한, Web Request 에 들어가는 인수 등 또한 아예 예시에 나와있는 것과 동일하게 맞췄는데 제대로 동작하지 않음. => 이에 따라 UnityWebRequest 에는 별도의 문제가 존재하지 않고, RUNPOD 의 TGI Server 에 문제가 있다고 생각하였음.
2. RUNPOD 의 TGI Server에서 변경 가능한 내용은 Colab 에 나와있는 RUNPOD 실행할 때 작성할 수 있는 인수들임. 주로, cloud_type, model_id 등을 변경하였는데, 내가 가변 가능한 변수를 변경한다 하더라도 여기서 제대로 동작하지 않음.

> 결론 : Docker 로 TGI Container 를 RUNPOD 를 통해 배포까지는 가능하였으나, 배포한 TGI Server 를 이용하는 것이 모종의 이유로 실패. 이에 따라, vLLM 등 다른 방법을 찾아보고자 함. (Serving LLM 과 관련하여 찾아보면 될 듯 함.)

- 마지막으로, Generate 일 때는 어떻게 되나 한번 확인해볼 것.

5. 실전 환경 구축 - vLLM

- vLLM 기본 개념 정리 참조.

> Model 기반 API Server 구축의 경우, Tutorial 등이 잘 나와있는 걸 따라가 실질적으로 바로 이용할 수 있도록 하는 것을 목적으로 함. 이때, 다음과 같은 것이 존재할 듯.
>
> 1. Colab 기반 배포 : Go Production: ⚡ Super FAST LLM (API) Serving with vLLM !!!
> 2. Google Cloud 에 Instance 만들어서 배포 : Setup vLLM with Google Cloud
> 3. Runpod 에서 빌린 Computer 에서 Jupyter Notebook 켜서 배포 : 이건 1번에서 Colab 을 기반으로 하여 실행했고, 나는 Jupyter Notebook 을 RUNPOD 에서 실행하는 방법을 찾아놨기 때문에 동일한 것을 이를 통해 진행하면 되지 않을까 하여 한 것!
>   - GPU가 없어요? GPU 메모리가 부족해요? 런팟을 사용해보세요 | jiogenes
> - 2번 영상을 약간 살펴봤을 때, Instance 를 생성해서, 이후 해당 Instance(Cloud Computer) 에 SSH 나 뭐 그런 거 이용해서 Console 로 접근해서 조작하던데, 이것도 맹 원격으로 Computer 를 조작한다는 점에서 동일하구나! 싶다!

- 현재 1번은 시도 했는데, T4 GPU 의 Memory 등이 부족해서 실행되지 않은 상황임. RUNPOD 와 위 코드를 응용하면 바로 가능하지 않을까 싶긴 함!

1. Colab 기반 배포 코드 분석

PYTHON

```
# 복사 자체는 드래그로 하니까 쉽게 되더라!
! pip install vllm

from vllm import LLM

prompts = ["Hello, my name is", "The capital of France is"]  # Sample prompts.
llm = LLM(model="hyeogi/SOLAR-10.7B-dpo-v1")  # Create an LLM.
outputs = llm.generate(prompts)  # Generate texts from the prompts.

! curl ipv4.icanhazip.com

! python -m vllm.entrypoints.openai.api_server --host 127.0.0.1 --port 8888 --model bigscience/bloomz-560m & npx localtunnel --port 8888

!curl https://every-peaches-bake.loca.lt/v1/completions -H "Content-Type: application/json" -d '{"model": "bigscience/bloomz-560m","prompt": "San Francisco is a","max_tokens": 7,"temperature": 0}'
```

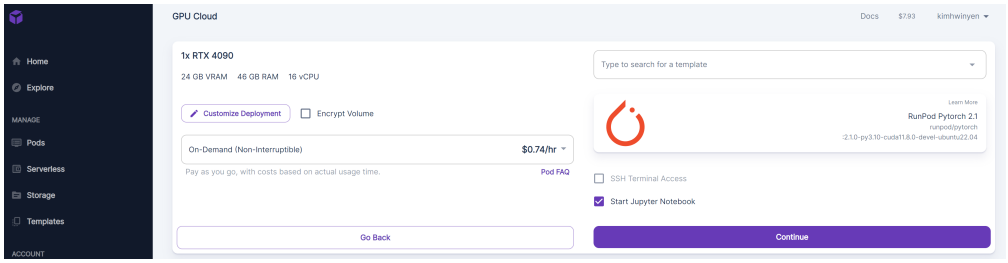
1. `pip` : 파이썬 pip(Package Installer of python)의 뜻, 사용법 (tistory.com) : "파이썬 모듈 및 모듈에 필요한 모든 파일(합하여, Pacakge)"을 다운로드 및 관리해주는 친구.
   - `!` : Jupyter Notebook 과 같은 python 환경에서 사용함. 이를 붙이면 python code 가 아닌 shell 명령어 로 해석되고 실행됨.
2. `from vllm import LLM` : vllm 라이브러리에서 LLM 클래스를 가져옴.
3. `LLM(model="hyeogi/SOLAR-10.7B-dpo-v1")` : 이러한 모델 생성.
4. `! curl ipv4.icanhazip.com` : `curl` 명령어를 사용하여 `ipv4.icanhazip.com` 웹사이트에 요청을 보냄. 이 사이트는 요청을 보낸 시스템의 공용 IPv4 주소를 반환.
5. `! python -m vllm.entrypoints.openai.api_server --host 127.0.0.1 --port 8888 --model bigscience/bloomz-560m` :
   - `vllm` 라이브러리의 `openai.api_server` 모듈을 실행(python 명령어가 그 의미.)

- 해당 서버는 local host 의 127.0.0.1 주소의 8888 port 에서 실행되며.
- model 은 bigscience/bloomz-560m 를 사용. (이건 왜 썼는지 모르겠네;)

6. `& npx localtunnel --port 8888` :
    1. `&` : 첫 번째 명령이 잘 끝나면 실행.
    2. `npx localtunnel --port 8888`: `npx`는 Node.js 패키지 실행자로, 여기서는 `localtunnel` 패키지를 실행. `localtunnel`은 로컬 서버를 인터넷에 공개하는 툴. `--port 8888` 옵션으로 인해, 8888 포트에서 실행 중인 로컬 서버(이전에 Python 명령으로 설정한 서버)를 인터넷에 공개.

2. RUNPOD 와 연결된 Jupyter Notebook 실행

1. RUNPOD 에서 사용량 구매
2. RUNPOD 에서 GPU Cloud POD 생성 (Container)



- Jupyter Notebook 사용할 거라 Template 으로 Pytorch 설정해 둠.
- 또한, 보안을 강화하기 위해 SSH Terminal 설정할 것임.
    1. Local (RUNPOD 의 Jupyter Notebook 에 접속할 컴퓨터) 에서 ssh-keygen 하기.
        - ssh-keygen -t ed25519 -c "Email" 하면, 알아서 됨. pub 파일 text 로 열어서 붙여넣으면 됨.
    2. SSH key 를 RUNPOD 내의 Account Setting 부분에다가 집어넣기.
- 그리고, 여기서 Volume을 좀 유의 해야 함!

    “Pasted image 20240131003309.png” could not be found.

    - 나는 지금 Docker Container 내부에 Model 을 직접 다운로드 받아 쓰려고 하기 때문에 이렇게 하는데, 정석적으로는 Persistant Data 로 다운로드 받아 쓰는 것이 좋긴 할 듯!
        - 또, Volume 명을 설정할 수 있는 것을 볼 때, 해당 할당한 Volume 을 다른 곳에서 쓰는 것도 가능한 것으로 보임!
        - *Q. Container 면 독립된 환경으로서 만들어지는데, 어캐쓰냐 싶긴 해!*

3. 이후, POD 가 Container 로서 만들어짐! 다 만들어진 후, connect > connect to jupyterlab 을 누르면, 해당 computer 의 jupyter notebook으로 잘 연결되는 것을 확인할 수 있음!
    - 아래에 떠있는 ssh 명령어를 입력해도, Terminal 에서 잘 연결 됨!

3. RUNPOD 에서 Colab 과 동일한 내용 실행.

- 동일 코드 약간 변경하여 실행함.
- 캬~ 잘 되는 거 보소~!
- *Q. Container 에서 Server 작동하는 것이므로 Port Number 등이 맞지 않아 문제가 발생하는 것이 아닐까? => A. 문제 발생 없을 듯. 왜? npx 기반으로 해서 local 로 만든 친구를 새롭게 deploy 해줄 거거든!*
- ㅋㅋㅋㅋㅋ rtx4090 도 llm 초기화 시, solar 모델에서 구동이 안 되네. => rtx6000 ada 로 구동하기로 함.
  - 다운로드 속도도 다른 것보다 더 빠르네! 대강 1GB 정도 된다!
- npx 를 여기서 실행할 수 없어 발생하는 문제가 존재! (npx 는 nodejs 의 library. Colab 에서는 지원하나, 일반적인 Jupyter 에서는 이를 지원하지 않음.)
- ngrok 을 이용하려 하였으나, authtoken 이 별도로 필요.
  - pyngrok - a Python wrapper for ngrok — pyngrok 7.0.5 documentation
  - authtoken 배정에 관련된 거 위 내용에 포함되어 있음.
- cuda memory 부족 발생!
  - **이건 vllm 으로 client 내에 검증하는 llm 을 만들고 나서, 이후 또 server 에 배속해주려고 했어서 그럼!**
  - => Server 만들어서 배포하는 code 를 바로 사용하면 문제 없음!
- Container 여서 문제 발생하는 건지 (RUNPOD 자체가 아무래도 그거니까.), 아니면 진짜 XSRF 라고, Jupyter 에서 Server 를 만들어서 문제 발생하는 건지, 모르겠네 ㅋㅋ...
- 그리고 결과적으로 훨씬 훨씬 쉬운 친구 찾았음.
  - runpod-workers/worker-vllm: The RunPod worker template for serving our large language model endpoints. Powered by vLLM. (github.com)
  - vLLM Docker Container 를 활용하여 RUNPOD Endpoint 를 구축할 수 있게 해주는 친구.
  - 이걸로 하면 될 듯?

> **결론 : 문제 다량 발생으로 인한 노선 전환!**

6. 실전 환경 구축 - vLLM with RunPod Worker!

- RUNPOD Worker 에 해당 내용 정리해놨으므로 참고할 것!
- 가격 확인 : 그냥 GPU 빌리는 거에 비해 거의 3배임. 대신, 초 단위다 보니까 안 돌아갈 때는 멈춰있음.
- Request 대략 구성하여, 해당 내용을 실행함. Request Code 도 아직 부족한 점이 많긴 하지만, 일단 이렇게 구성.

- 잘 된다!!!!!!!!!!!!!!!! ㅠㅠㅠㅠㅠㅠㅠㅠㅠ