

Full Stack ML Engineer

구분	
링크	https://drive.google.com/file/d/1Q7mRbN7YlQcM6IYM7KTdsQxoLk1cHKF6/view?usp=sharing
발표자	이준엽 (Upstage)
비고	
완료	<input type="checkbox"/>
완료 예정일	@2021/09/24
유형	특강

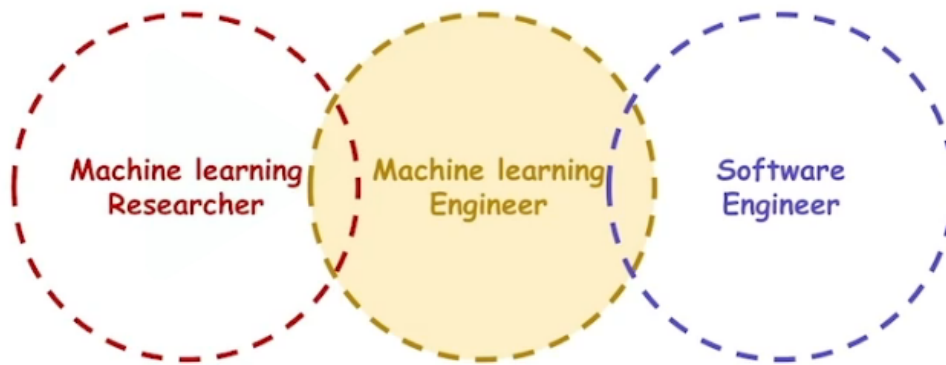


Full stack machine learning engineer (FSML-Engineer)에 대해 정의하고, 업계에서 FSML-Engineer의 역할과 실제 업무에 대해 공유합니다. ML 서비스란 무엇인지와 FSML-Engineer가 되면 좋은 점에 대해서도 간단하게 알아봅니다.

1. Full stack ML Engineer?

1.1 ML Engineer 란?

- Machine learning (Deep learning) 기술을 이해하고, 연구하고, Product 를 만드는 Engineer
- Deep learning 의 급부상으로 Product 에 Deep learning 을 적용하고자 하는 수요 발생
- 전통적인 기술의 경우 Research 영역과 Engineering 영역이 구분되지만, Deep learning 의 경우 폭발적 발전속도로 인해 그 경계가 모호함 (연구와 동시에 Product 에 적용)

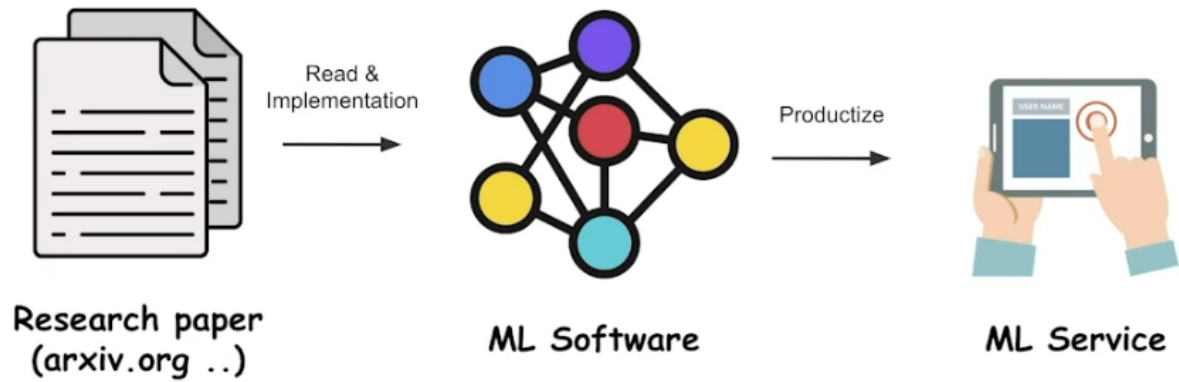


1.2 Full stack Engineer 란?

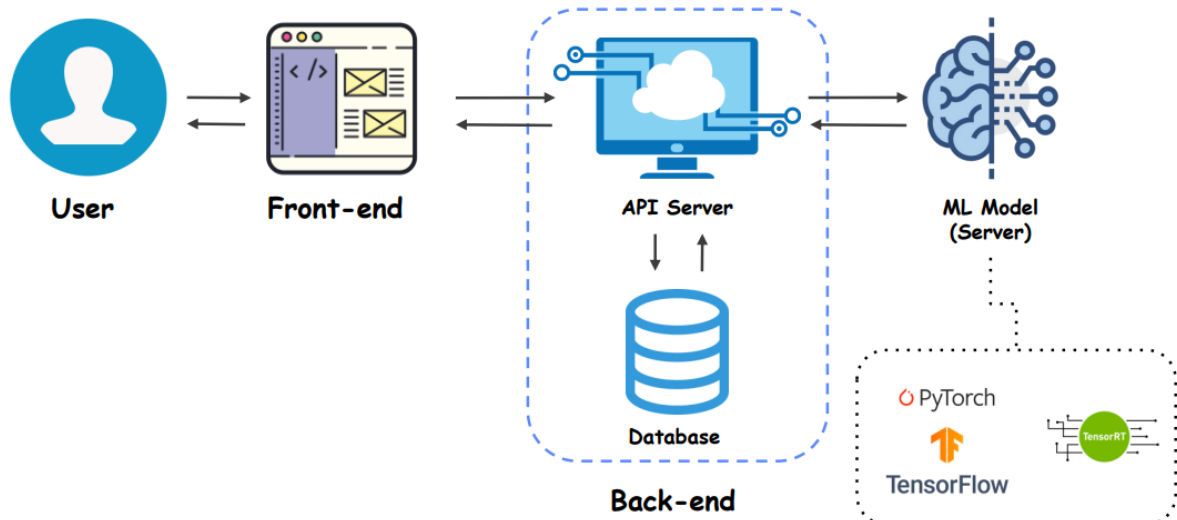
- Full stack engineer 란 표현은 각 포지션의 개발자들에게 달갑지 않은 표현일 수도 있다고 생각
(소개할 때 조심하자!)
- Full stack engineer 은 상태가 아니라 방향성이라고 생각.
- 모든 Stack을 잘 다루려는 방향으로 가고있다면 Full stack engineer
- 기준을 세우자면 내가 만들고 싶은 Product를 시간만 있다면 모두 혼자 만들수 있는 개발자

1.3 Full stack ML Engineer

- Deep Learning research를 이해하고 ML Product로 만들 수 있는 Engineer

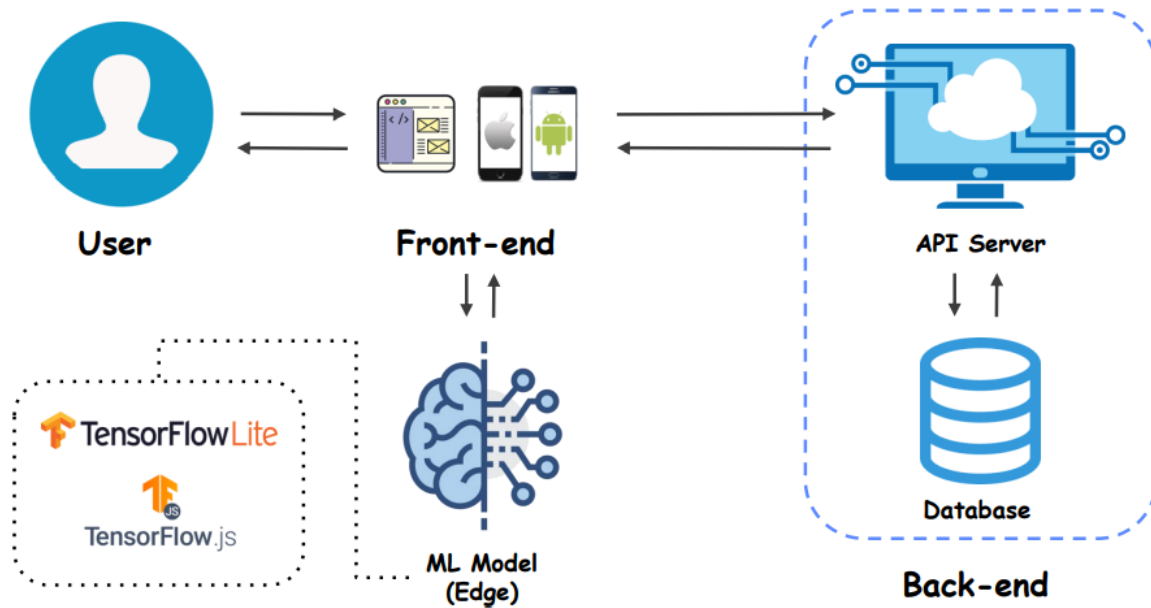


1.3.1 ML Service



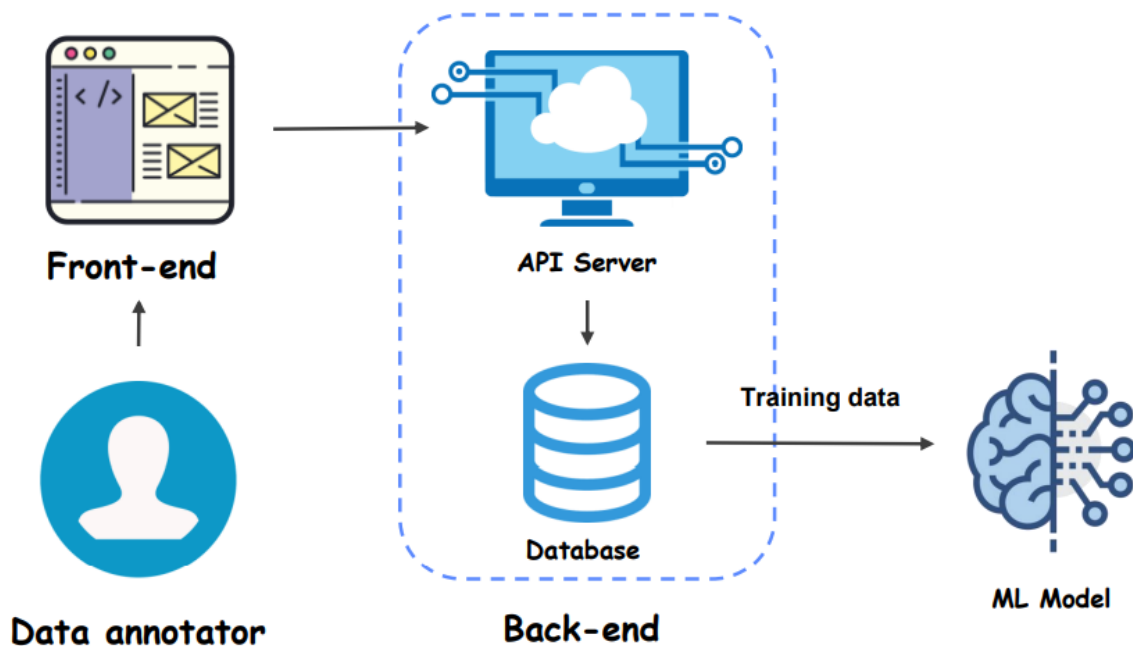
1.3.2 Edge device service

- Front-end 에서 처리



1.3.3 Data collection

- ML Model을 만들기 위한 pipeline



2. Pros, cons of Full stack ML engineer

- Full stack ML engineer의 좋은점(pros)과 안 좋은점(cons)

2.1 장점 (Pros)

- 아래의 장점들이 반드시 Full stack engineer가 되어야만 갖출 수 있는 것은 아니다.

2.1.1 재밌다

- 소프트웨어 개발은 컴퓨터에서 돌아가는 작은 세상을 만드는 작업

2.1.2 빠른 프로토타이핑

- 프로토타이핑을 협업하기는 곤란한 경우가 많음

2.1.3 기술간 시너지

- 연결에 대한 고려가 들어간 개발 결과물은 팀의 불필요한 로스를 줄여줌

2.1.4 팀플레이



- 서로 갈등이 생길 법한 부분에서의 기술적인 이해가 매우 도움이 됨
- 기술에 대한 이해 == 잠재적 위험에 대한 고려

2.1.5 성장의 다각화

- 연구자 + 개발자 + 기획자가 모인 회의에서 모든 내용이 성장의 밑거름이 됨
- 사람에 따라서는 매너리즘을 떨치는 방법이 되기도 함

2.2 단점 (Cons)

2.2.1 깊이가 없어 질 수도 있다.

- 아무래도 하나의 스택에 집중한 개발자 보다는 해당 스택에 대한 깊이가 없어질 수 있다.

2.2.2 시간이 많이 들어간다.

- 공부할 분야가 많다보니 절대적으로 시간이 많이 들어간다.
- 절대적인 시간을 많이 투자한 만큼 깊이를 갖게 될 것.

3. ML Product, ML Team, ML Engineer

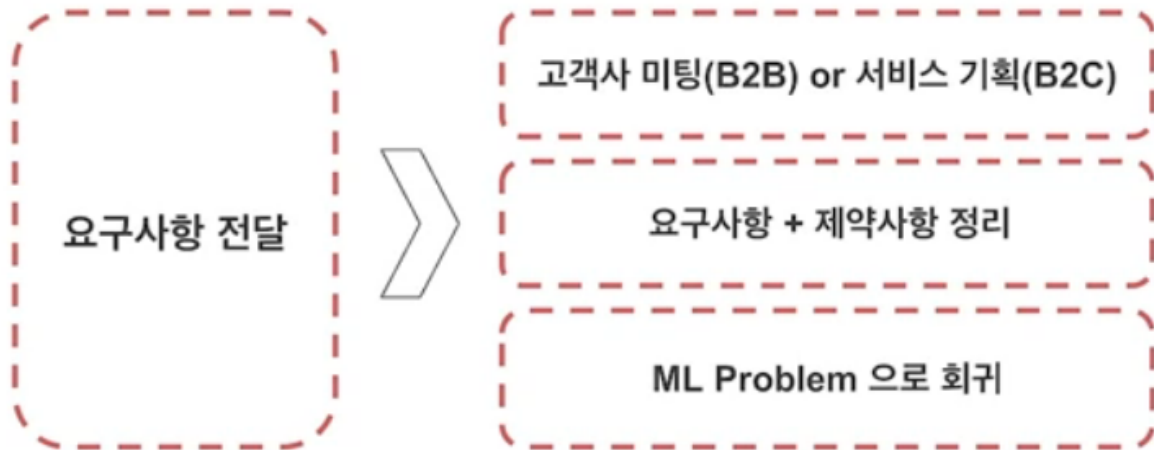
- 회사에서 ML Product 는 어떻게 만들어지는지, ML 팀은 어떻게 구성되어있는지,
- 팀에서 Full stack ML 엔지니어의 역할은 무엇인지 대한 저의 경험을 나눕니다.

3.1 ML Product



3.1.1 요구사항 전달

- 고객의 요구사항을 수집하는 단계



고객사 미팅(B2B) or 서비스 기획(B2C)

- 제품에 대한 요구사항 전달
- 추상적인 단계

요구사항 + 제약사항 정리

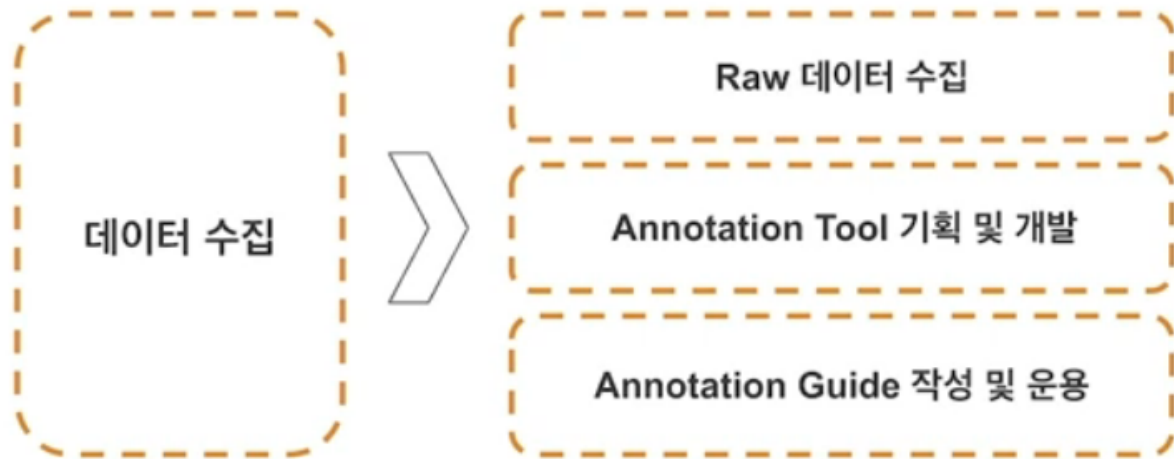
- 기능 상세 요구 사항 정리
- 제약 사항 정리
 - ex) GPU 하나에 몇 개의 모델이 올라가야 하는 지
- 일정 정리

ML Problem 으로 회귀 (가장 중요!)

- 요구사항은 실 생활의 문제
- ML이 풀 수 있는 형태의 문제로 회귀하는 작업

3.1.2 데이터 수집

- Model을 훈련/평가할 데이터를 취득하는 단계



Raw 데이터 수집

- 요구사항에 맞는 데이터 수집
- Bias 없도록 분포 주의
- 저작권 주의

Annotation Tool 기획 및 개발

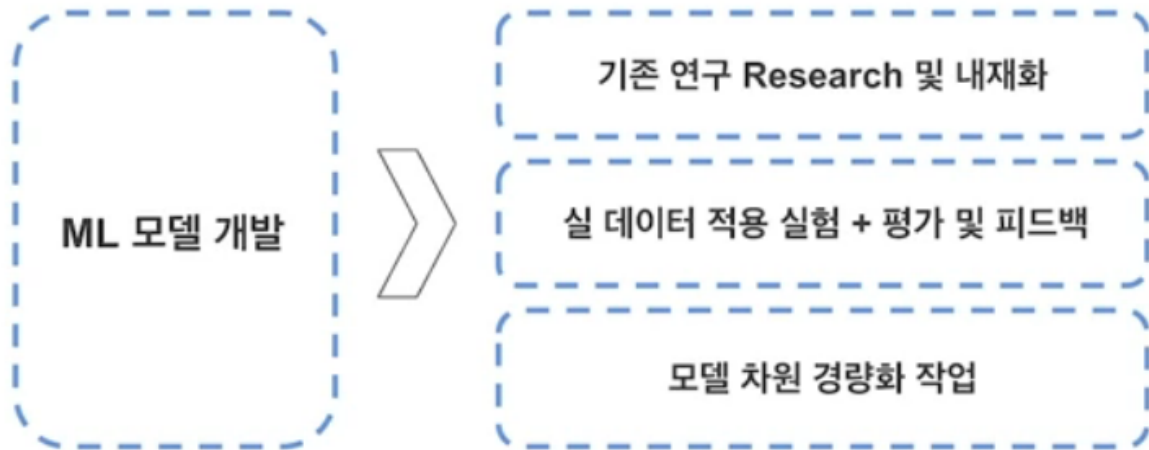
- 데이터에 대한 정답 입력 톨
- 모델 Input / Output 고려
- 작업속도 / 정확도 극대화

Annotation Guide 작성 및 운용

- "이럴 땐 어떻게 Annotation 하나요?" 에 대한 대답이 되는 문서
- 간단하고 명확한 Guide 문서를 작성하도록 노력

3.1.3 ML 모델 개발

- Machine learning 모델을 개발하는 단계



기존 연구 Research 및 내재화

- 논문을 찾아 보고 이해하는 단계
- 적절한 연구를 재현하여 내재화
- baseline 모델 확보

실 데이터 적용 실험 + 평가 및 피드백

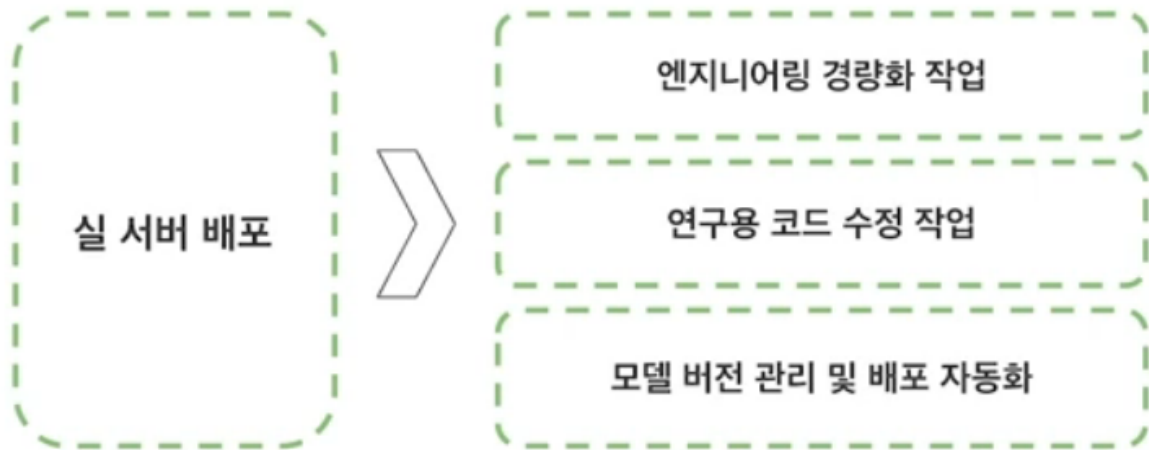
- 수집된 데이터 적용
- 평가 및 모델 수정

모델 차원 경량화 작업

- 모델 단계의 경량화
- Distillation
- Network surgery

3.1.4 실 서버 배포

- 서비스 서버에 적용하는 단계



엔지니어링 경량화 작업

- TensorRT 등의 프레임워크 적용
- Quantization

연구용 코드 수정 작업

- 연구용 코드 정리
- 배포용 코드와 Inference 맞추는 작업

모델 버전 관리 및 배포 자동화

- 모델 버전 관리 시스템
- 모델 배포 주기 결정 & 업데이트 배포 자동화 작업

3.2 ML Team 구성

- 프로젝트 매니저 (1)
- 개발자 (2)
- 연구자 (2)
- 기획자 (1)
- 데이터 관리자 (1)

- 각각의 역할을 겸하게 된다.

3.3 Full stack ML Engineer in ML Team

Job 1. 실생활 문제를 ML 문제로 Formulation

- 고객 / 서비스 의 요구사항은 실 생활 문제!
- Machine learning 모델이 해결 가능한 형태로 문제를 쪼개는 작업 / 가능한지 판단하는 작업
- 기존 연구에 대한 폭 넓은 이해와 최신 연구의 수준을 파악하고 있어야 함

Job 2. Raw Data 수집

- 웹에서 학습데이터를 모아야 하는 경우도 있음
- Web Crawler (Scraper) 개발해서 데이터 수집 (저작권 주의)

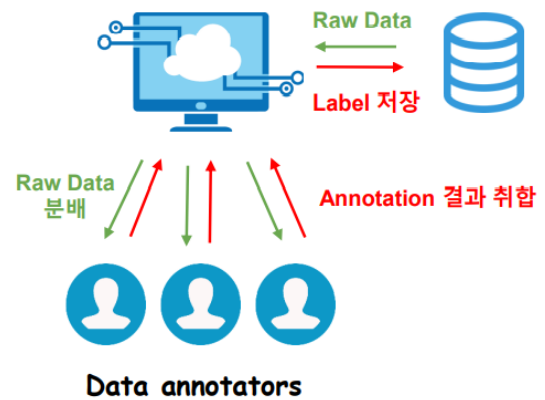
Job 3. Annotation tool 개발

- 수집/제공 받은 데이터의 정답을 입력하는 작업을 수행하는 web application 개발
- 작업 속도와 정확성을 고려한 UI 디자인이 필요
- 다수의 Annotator 들이 Client 를 통해 동시에 서버로 접속. Annotation 작업을 수행.
- 새로운 Task 에 대한 Annotation tool 기획시 모델에 대한 이해가 필요할 수 있었음



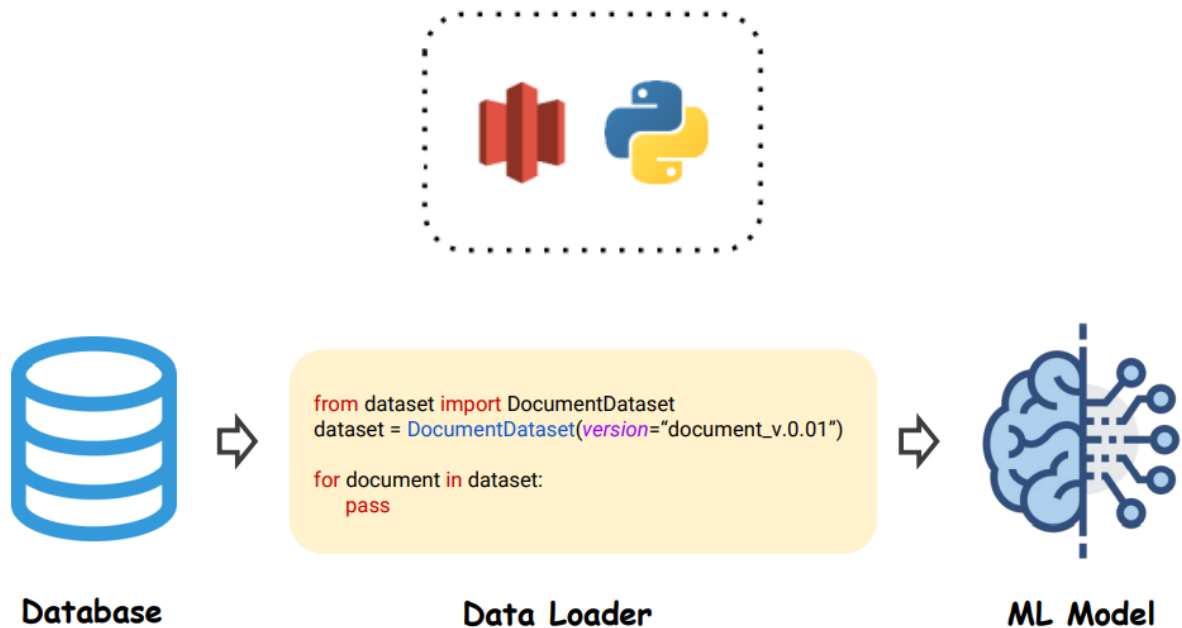


<Object detection annotation tool>



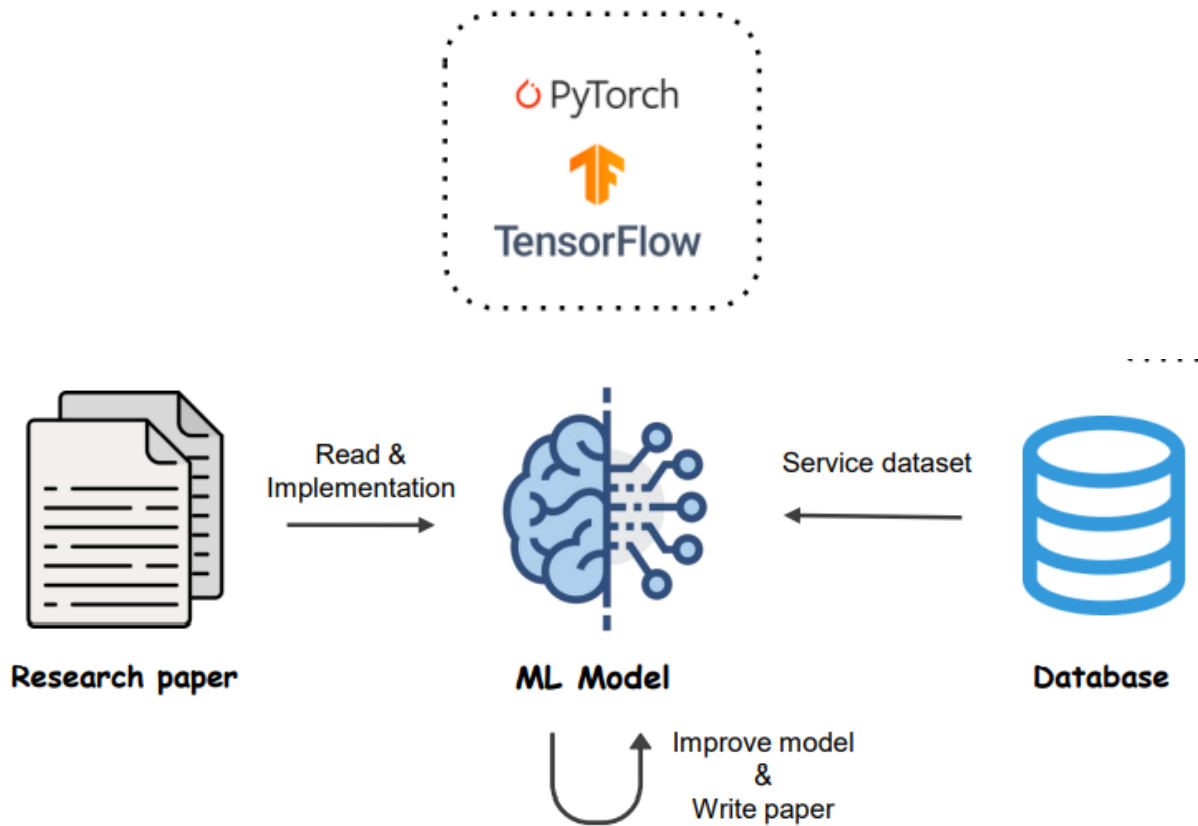
Job 4. Data version 관리 및 loader 개발

- 쌓인 데이터의 Version 관리
- Database에 있는 데이터를 Model로 Load 하기 위한 Loader package 개발



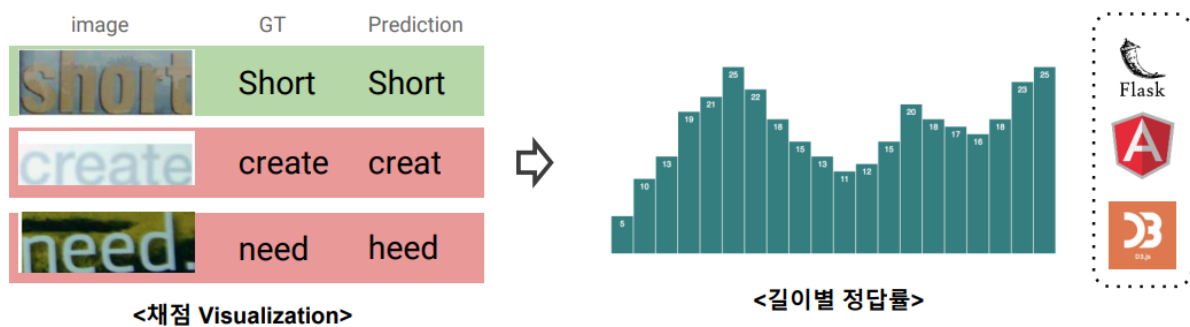
Job 5. Model 개발 및 논문 작성

- 기존 연구 조사 및 재현 (재현 성능은 Public benchmark 데이터로 검증)
- 수집된 서비스 데이터 적용
- 모델 개선 작업 + 아이디어 적용 → 논문 작성



Job 6. Evaluation tool 혹은 Demo 개발

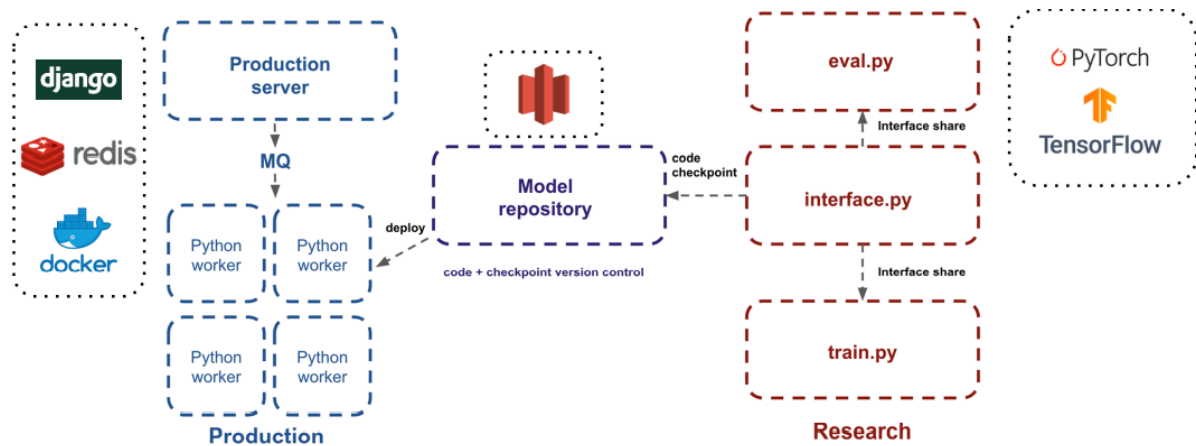
- 모델의 Prediction 결과를 채점하는 Web application 개발
- OCR 프로젝트 중 혼자 사용하려고 개발 (정/오답 케이스 분석) → 이후 팀에서 모두 사용.
- 모두 사용하다보니 모델 특성 파악을 위한 요구사항 발생
→ 반영하다보니 모델 발전의 경쟁력이 됨



Job 7. 모델 실 서버 배포

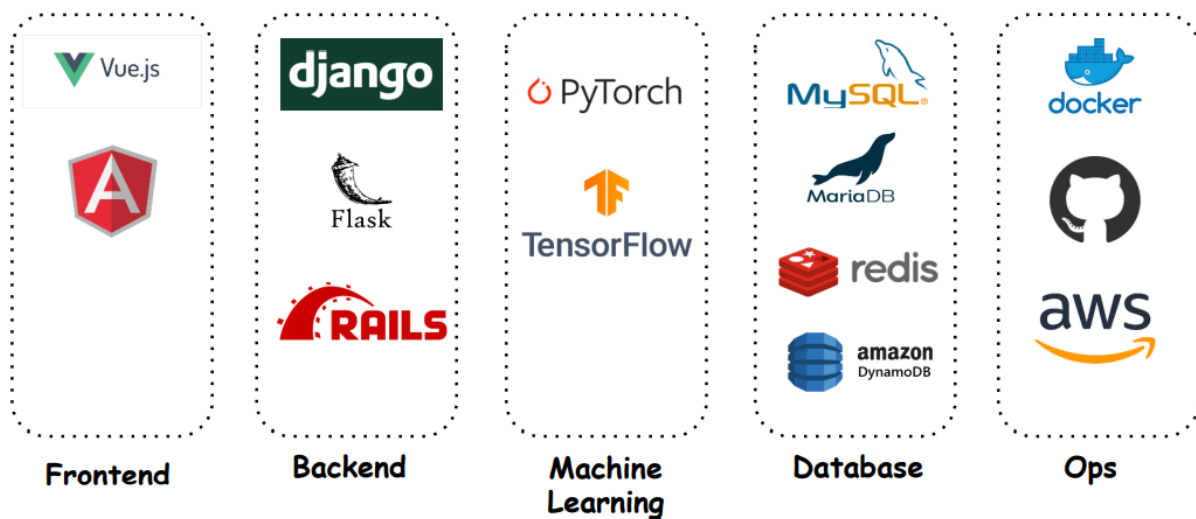
- 연구용 코드를 Production server 에서 사용 가능하도록 정리하는 작업

- File server 에 코드 + Weight 파일 압축해서 Version 관리
- Production server 에서는 Python worker 에게 MQ를 통해 job 을 전달



4. Roadmap

4.1 Stack share



4.2 시작하는 방법

- 각 스택에서 점점더 framework 의 발전이 점점 더 interface 가 쉬워지는 방향으로 발전
- 특정 스택이 쉽다는 것은 아님.
- 하지만 초~중급수준의 구현을 하는것이 점점 쉬워질 것

- ML Engineer 라면, 하나의 논문을 구현하고, Demo page 를 만들어보는 것을 추천합니다.



4.3 염두할 점

4.3.1 시작이 80%이다.

- 시작이 반이다? → 시작이 80% 다
- 모든 Stack이 공통적으로 **시작이 가장 어렵습니다.**
- **익숙한 언어 + 가장 적은 기능 + 가장 쉬운 Framework** 로 시작하세요.
- 하나를 배우고 나면 나머지를 배우는 것은 훨씬 쉽다! Google + Stackoverflow 의 도움으로 짹짹!
- 시작의 허들을 넘은 뒤, 필요에 의해 원론을 공부하세요.

4.3.2 빨리 완성해보기


- **처음부터 너무 잘 만들려고 하지 마세요. 최대한 빨리 완성하세요**
- 많은 개발자들이 Full stack 으로 개발하다보면 가장 중요한 "완성" 이라는 가치에 도달하지 못하는 이유
- 모든 stack의 안티 패턴을 모두 신경써서 개발하다가 포기하기보다는 완성에 집중하세요
- Full stack 개발의 매력은 "완성" 에서 오는 "재미" 입니다.
- 완성된 코드에 기능을 더 하다보면 자연스레 리팩토링의 중요성에 대해 알게 됩니다.
- 재미를 느끼고 반복적으로 완성하다보면 실력이 늘게 됩니다.

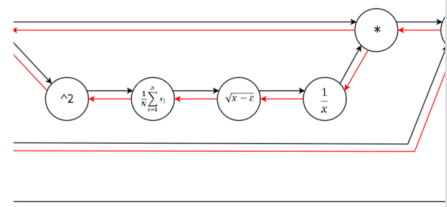
4.3.3 전문 분야 정하기

- 전문 분야를 정하세요!
- 모든 Stack 에서 초~중급 개발수준을 계속 유지하는 것은 좋지 않은 선택입니다.
- Andrej Karpathy 의 “Yes you should understand backprop” 이라는 글을 읽어보시길 추천합니다.

Yes you should understand backprop

When we offered CS231n (Deep Learning class) at Stanford, we intentionally designed the programming assignments to include explicit calculations involved in backpropagation on the lowest

 <https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b>



4.3.4 새로운 것에 대한 두려움 없애기

- 새로운 것에 대한 두려움 없애기 위해 반복적으로 접하세요
- Frontend 개발이 어렵고 자신 없다면, 회사에서 Frontend 개발을 할 수 있는 기회를 찾으세요
- 배우고 싶었던 Stack 에 대한 문서 + 유튜브를 쉬운 것 부터 재미로 보면 좋습니다.
- 해당 Stack 을 잘 하시는 분에게 많이 질문하세요.