

6th Assignment – Real Usage and Final Report

영수증 기반 가계부용 **Donut OCR + Discord Bot**

- 과목명: 기계학습의 이해
- 학번: 202403480
- 이름: 유보령

1. 프로젝트 개요

1.1 무엇을 만들었나?

이 프로젝트에서는 영수증 사진을 자동으로 읽고, 가계부 형식으로 정리해 주는 디스코드 봇 서비스를 구현해보고자 했습니다.

사용자는 디스코드 **DM**에서 봇에게 영수증 이미지를 보내기만 하면,

1. Donut OCR 모델이 영수증에서
 - 날짜(date)
 - 가게 이름(store)
 - 총 금액(cost)을 추출하고,
2. 추출된 가게 이름과 사전에 준비한 **vendor_map.csv**를 이용해 카테고리(편의점/마트, 카페, 식비, 문화/여가, 기타)를 추정합니다.
3. 봇이 추정 결과를 보여주고, 틀린 부분은 사용자가 직접 수정할 수 있게 한 뒤
4. 최종 결과를 **ledger.csv**에 저장하여
!가계부, !이번주, !오늘 명령으로 기간별 지출 내역을 확인할 수 있도록 했습니다.

1.2 어떤 문제를 해결했나?

평소에 소비 기록을 남기기 위해 가계부 앱이나 엑셀을 사용해 보려고 했지만, 영수증을 보고 일일이 날짜·가게·금액·카테고리를 입력하는 과정이 너무 번거로웠습니다.

- 결제할 때마다 영수증은 생기지만 집에 돌아와서 그 영수증을 다시 꺼내 항목을 하나씩 타이핑해야 한다는 점 때문에 몇 번 시도하다가 금방 포기하게 되고, 결국 실제 지출 내역과 가계부 기록이 서로 맞지 않는 문제가 반복되었습니다.

- 또한, 대부분의 가게부 서비스는 이미 텍스트로 정리되어 있는 카드 명세서 기반 혹은 사용자가 직접 입력하는 방식을 전제로 하고 있어서, 사진 형태의 영수증을 바로 처리해 주는 기능이 적다고 생각을 했습니다.
- 그래서 이번 프로젝트에서는 “영수증 사진을 그대로 보내기만 하면, 필요한 정보(날짜·금액·카테고리)를 자동으로 뽑아서 가게부에 기록해 주는 도구”를 만들어, “가게부를 쓰고 싶지만 입력이 귀찮아서 포기하게 되는 문제”를 조금이나마 해결해보았습니다.

2. 진행 과정

2.1 주제 선정 및 문제 정의

- 평소에 가게부를 작성하지 않아 지출이 관리가 안 되어, 과소비를 하는 문제가 있었습니다.
- 직접 가게부 앱에 날짜, 가게 이름, 카테고리, 금액 등을 일일이 입력하는 과정이 귀찮아서, 결국 기록이 끊기는 경우가 많았습니다.
- 그래서 자주 사용하는 디스코드 안에서 영수증 사진만 보내면 자동으로 가게부를 만들어주는 흐름을 만들어보고자 했습니다.

2.2 데이터 수집 및 분석

2.2.1 데이터 수집

- 직접 여러 번 결제한 실제 영수증 사진을 촬영해서 데이터를 만들었습니다.
- 서로 다른 가게(편의점, 카페, 식당 등)를 포함해서 총 **30장**의 영수증 이미지를 모았습니다.
- 각 이미지에 대해 다음과 같은 **JSON** 라벨을 수동으로 만들었습니다.
`{"date": "2025-11-13", "store": "알촌", "total": "8300"}`
- Donut 모델의 학습 형식에 맞추기 위해 실제 라벨은 `<s>`, `</s>` 토큰을 포함한 텍스트로 저장하였습니다.

`<s>{"date": "2025-11-13", "store": "알촌", "total": "8300"}</s>`

2.2.2 데이터 증강

원본 30장은 너무 적어서, 다음과 같은 간단한 증강(augmentation)을 적용해 데이터 수를 늘렸습니다.

- 밝기/대비 변화
- 약간의 회전 / 기울기
- 노이즈 추가

각 이미지당 5종류의 증강을 추가하여 원본 30장 + 증강 150장 = 총 180장의 이미지를 만들었습니다.

라벨(labels.json)은 이미지 파일명 기준으로 하나씩만 두고, DonutReceiptDataset 클래스에서 증강 이미지가 있을 경우 우선 사용하고, 없으면 원본 이미지를 사용하는 방식으로 데이터셋을 구성했다.

2.2.3 데이터 구조

- donut_dataset/images/ : 원본 이미지
- donut_dataset/images_aug/ : 증강 이미지
- donut_dataset/labels.json : 파일명 → <s>{...}</s> 형식 텍스트 매핑

데이터셋 클래스를 통해

- pixel_values (이미지 텐서)
- labels (토큰화된 정답 시퀀스)가 모델에 들어가도록 구현했습니다.

2.3 모델 선택 및 학습 방법

2.3.1 Donut 모델 선택

OCR + 구조화 정보를 동시에 뽑기 위해 Naver Clova의 Donut 모델 (naver-clova-ix/donut-base-finetuned-cord-v2) 을 사용했습니다.

선택 이유:

- 이미지 인코더 + 텍스트 디코더 구조로 되어 있어 “이미지 → JSON 텍스트” 형태의 학습이 잘 맞을 거라 판단했습니다.
- CORD 데이터셋으로 미리 학습되어 있어 영수증 및 영수증과 비슷한 문서에 특화되어 있다 판단했습니다.

2.3.2 학습 환경

- Google Colab / Jupyter 기반 환경
- Python 3.10 이상

2.3.3 하이퍼파라미터 및 설정

- 배치 크기: 1

- 학습 epoch: 4
- learning rate: 1e-5 (소규모 데이터라 작은 값 사용)
- max_length: 80
- pad_token_id, decoder_start_token_id, eos_token_id를 Donut tokenizer 설정과 일치하도록 세팅
- gradient checkpointing 활성화로 메모리 절약

특히, epoch 수는 여러 번 시도 끝에 **4 epoch**로 결정하게 되었습니다.

- 2 epoch: 학습이 덜 돼서 가게 이름 인식이 많이 틀림
- 8 epoch 이상: train loss는 줄지만, 작은 데이터셋 특성상 과적합 위험이 커지고, 새로운 영수증에서 오히려 성능이 떨어지는 경향을 관찰 → 그래서 손실이 안정적으로 줄어드는 **4 epoch**에서 학습을 마무리하였습니다.

2.4 모델 평가 방법

2.4.1 데이터 분할

Donut을 미세조정(fine-tuning)할 때 사용한 영수증 데이터는 총 180장입니다. 이 중 약 80%는 학습(train), 20%는 평가용(test)으로 사용했습니다.

- 학습 세트: 144장 (원본 + 증강 포함)
- 테스트 세트: 36장

테스트 세트에는 편의점, 카페, 식당, 문구점 등 서로 다른 업종의 영수증을 포함시켜, 모델이 특정 가게에만 과적합되지 않았는지 확인하고자 했습니다.

2.4.2 평가 대상 필드

Donut 모델이 출력한 JSON에서 다음 세 가지 필드를 평가 대상으로 삼았다.

1. **date** : 결제 일자
2. **store** : 가게 이름
3. **cost** : 총 결제 금액

각 영수증에 대해 사람이 직접 만든 “정답 JSON”을 준비한 뒤, 모델의 출력과 필드별로 비교하였다.

2.4.3 평가 지표

필드별 성능을 정량화하기 위해 다음과 같은 지표를 사용했습니다.

1. 필드 단위 정확도(**Field-wise Accuracy**)

각 필드에 대해 모델 출력값이 정답과 완전히 일치하면 **1점**, 그렇지 않으면 **0점**으로 채점하고, 모든 테스트 샘플에 대해 평균을 내어 정확도를 계산했습니다.

2. 영수증 단위 완전 정확도(**Receipt-level Exact Match**)

하나의 영수증에 대해 **date**, **store**, **cost** 세 필드가 모두 정답과 일치하면 “성공”으로 간주하여, 전체 테스트 샘플 중 완전히 맞춘 영수증 비율을 별도로 계산했습니다.

3. 금액 오차(**Amount Error**)

cost 값은 숫자이기 때문에, 완전히 틀린 경우에도 얼마나 차이가 나는지를 보기 위해 평균 절대 퍼센트 오차(**MAPE**)를 추가로 계산했습니다.

4. 카테고리 분류 성능(**Category Accuracy**)

vendor_map.csv + 규칙 기반으로 예측한 카테고리를, 사람이 직접 부여한 “정답 카테고리(편의점/마트, 카페, 식비, 문화/여가, 기타)”와 비교하여 카테고리 정확도도 따로 계산하였습니다.

3. 모델을 서비스로 만든 구조

3.1 학습한 모델을 어떻게 실제 사용 가능한 서비스로 만들었나

- 직접 만든 디스코드 봇을 서버에 불러와 영수증 가계부 생성 모델과 연동을 하였습니다.

3.2 전체 시스템 구성

서비스는 다음 네 부분으로 나뉘게 됩니다.

1. 모델 학습 스크립트 (**donut_training.ipynb**)

- Donut 모델을 영수증 데이터에 맞게 fine-tuning
- 결과를 **donut_finetuned/** 폴더에 저장 (토큰라이저, **config**, **model** 파라미터 등)

2. OCR 모듈 (**ocr_model.py**)

- **load_ocr_model()** :
donut_finetuned 폴더를 읽어 Donut 모델과 **processor**를 로딩
- **run_ocr(image_path, processor, model, device)** :
이미지 한 장에 대해
 - Donut **generate()** 호출

- raw 텍스트 출력
- 텍스트에서 JSON 부분만 추출 후 dict로 파싱

3. CLI 서비스 (service_cli.py)

- 로컬에서 터미널로만 모델 기능을 테스트하기 위한 간단한 도구
- 사용자에게 이미지 경로를 입력받아 날짜/가게/금액을 추출하고, records.csv에 로그로 남긴다.

4. 디스코드 봇 (discord_bot.py)

- 핵심 서비스.
- 주요 흐름:
 - 사용자가 DM으로 영수증 이미지를 전송
 - 이미지를 tmp_images/에 임시 저장
 - run_ocr()로 날짜/가게/금액 추출
 - vendor_map.csv와 간단한 규칙을 이용해 카테고리 추정
 - 아래 내용을 포함한 임베드 메시지를 전송
 - 일시(추정)
 - 카테고리(추정)
 - 금액(추정)
 - 업체명
 - 원본 이미지 링크
 - 버튼 두 개 제공
 - [확정(저장)] : 그대로 ledger.csv에 기록
 - [수정] : 사용자가 "날짜, 카테고리, 금액, 업체명" 형식으로 수정값 입력

- 최종 확정된 내용은
ledger.csv에 date, category, store, cost 형식으로 저장
- 추가 명령어
 - !가계부 : 전체 기간에 대한 지출 요약
 - !이번주 : 이번 주(월~일) 지출 요약
 - !오늘 : 오늘 하루 지출 요약
→ 결과는 카테고리별 합계와, 최근 항목 몇 개를 함께 보여줌

3.3 vendor_map.csv와 카테고리 분류

vendor_map.csv는 다음과 같은 구조를 가집니다.

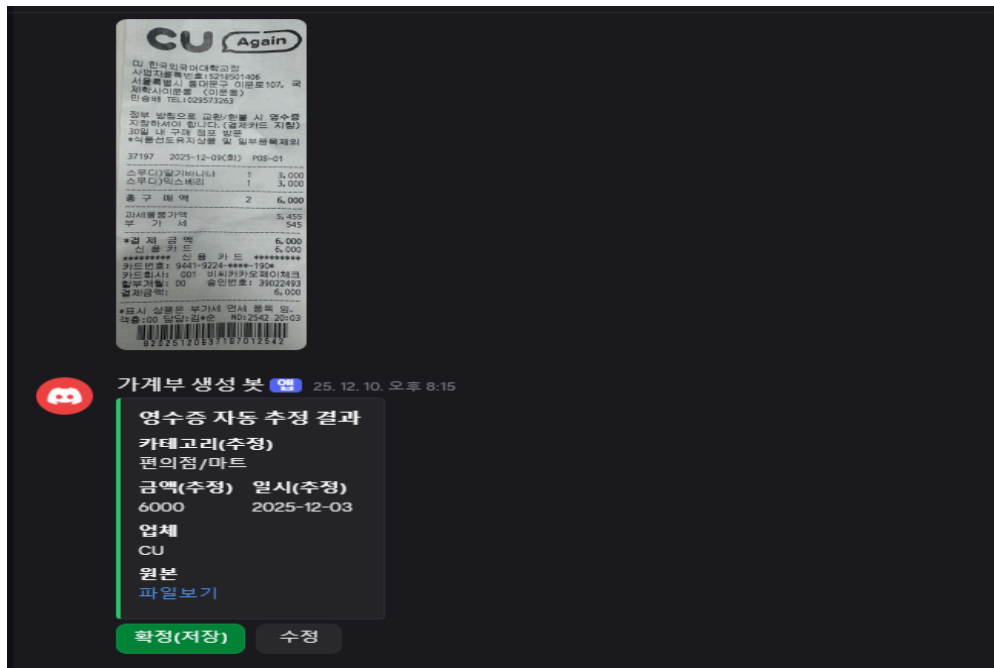
keyword	category
cu	편의점/마트
gs25	편의점/마트
starbucks	카페
artbox	기타

- 봇은 store_name과 OCR 원문 텍스트를 합친 문자열을 소문자로 변환한 뒤,
 - keyword가 포함되어 있으면 해당 category를 사용
 - 어떤 키워드와도 매칭되지 않으면 "기타"로 분류
- 카테고리는 편의점/마트, 카페, 식비, 문화/여가, 기타로 제한하였습니다.

4. 실제 사용 결과

4.1 사용 기록

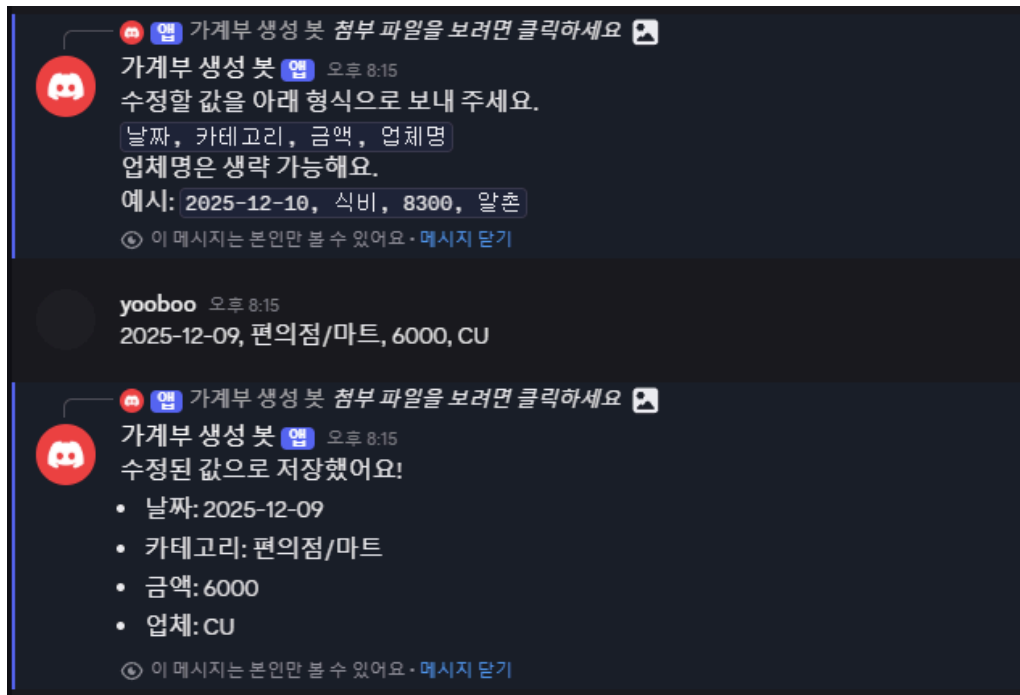
1. CU 편의점 영수증
 - 입력: CU 영수증 사진



○ 모델 추정:

- 날짜: 2025-12-03
- 가게: CU
- 총 금액: 6000
- 카테고리: 편의점/마트(`vendor_map.csv`에서 `cu` 매칭)

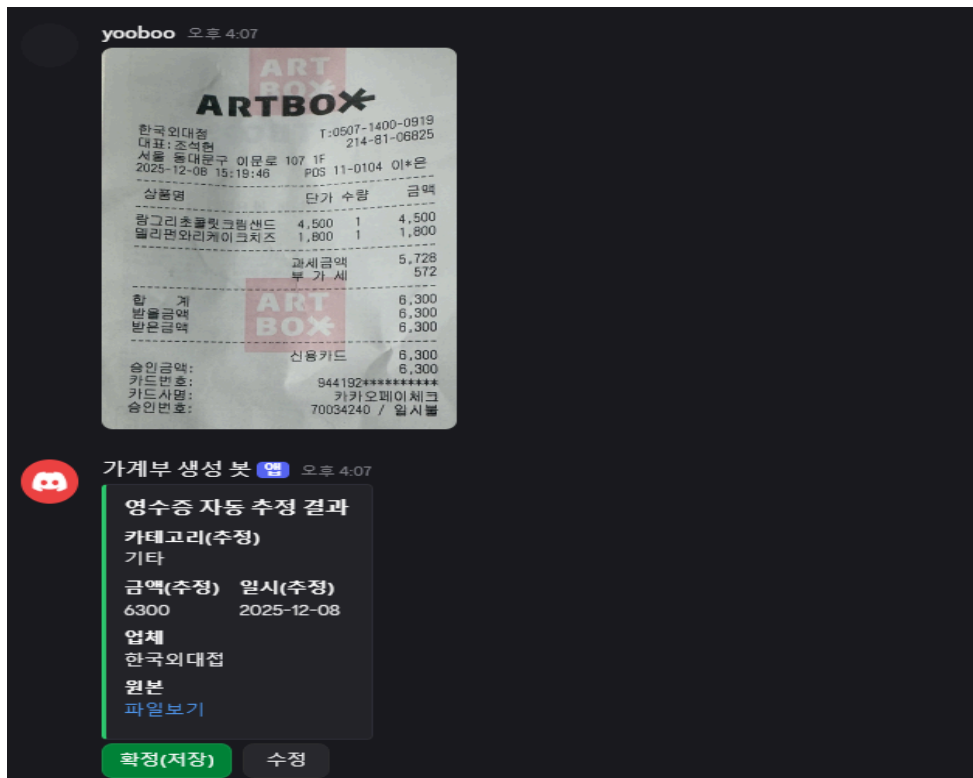
○ 날짜가 맞지 않아, 수정 버튼 클릭



- ledger.csv에 정상적으로 수정되어 기록됨

2. ARTBOX 영수증

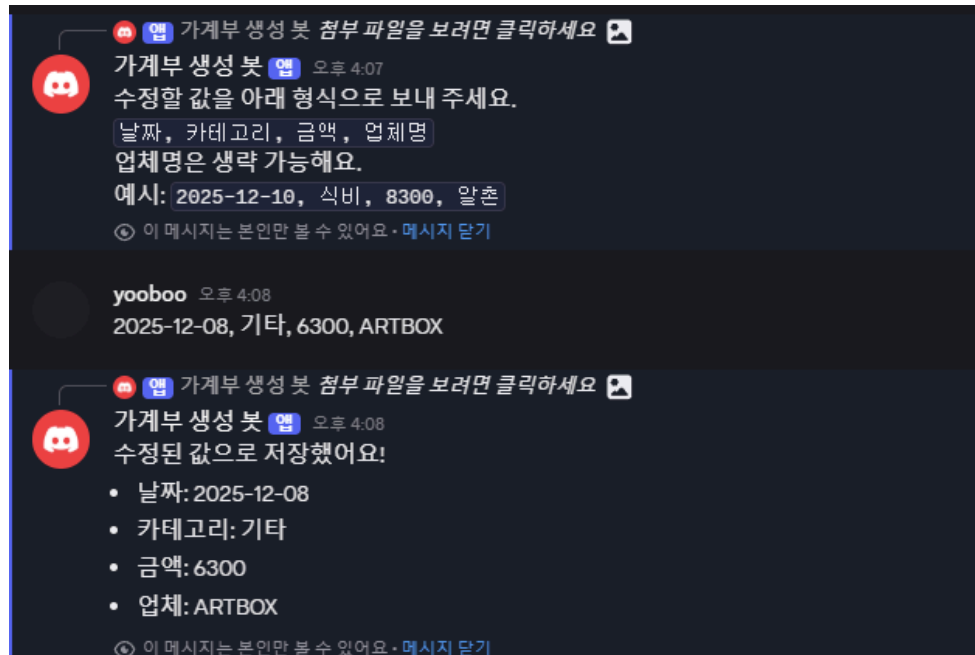
- 입력: ARTBOX 문구점 영수증 사진



○ 모델 추정:

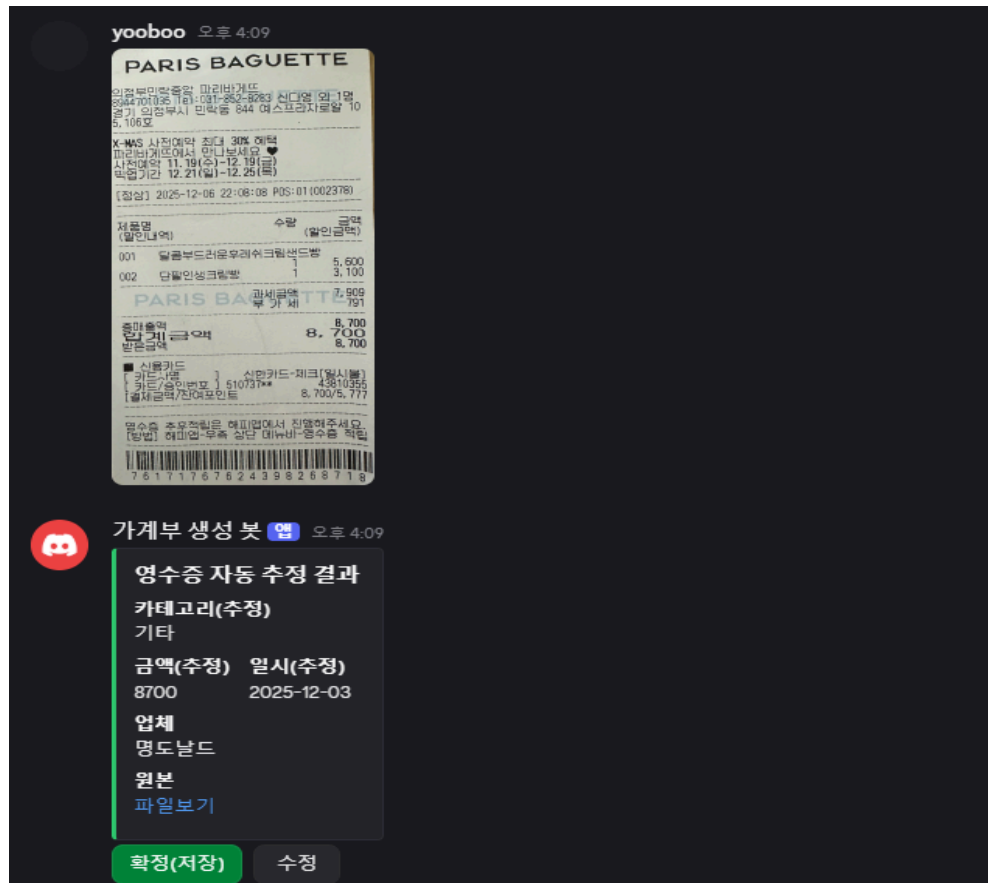
- 날짜: 2025-12-08
- 가게: 한국외대점 (주소 줄에서 인식)
- 금액: 6300
- 카테고리: 기타 (vendor map에 없어서 규칙에 따라 기타 처리)

○ 실제로는 “선물/문구” 성격이지만, 모델이 카테고리를 맞추기 어렵기 때문에
→ 수정 버튼을 눌러 카테고리를 기타로 직접 지정



3. 파riba게트 영수증

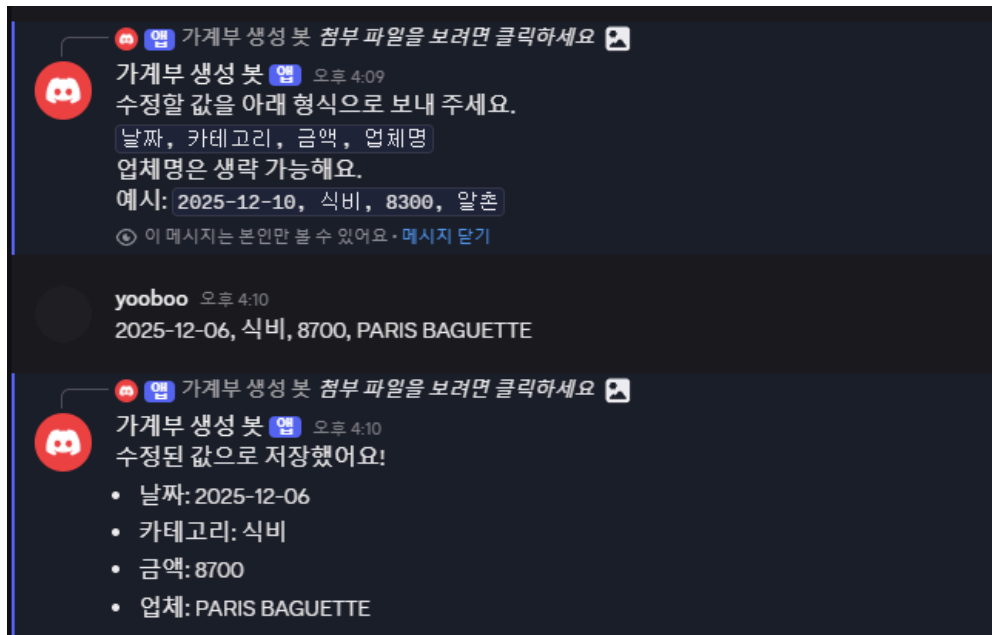
○ 입력: 파riba게트 영수증 사진



○ 모델 추정:

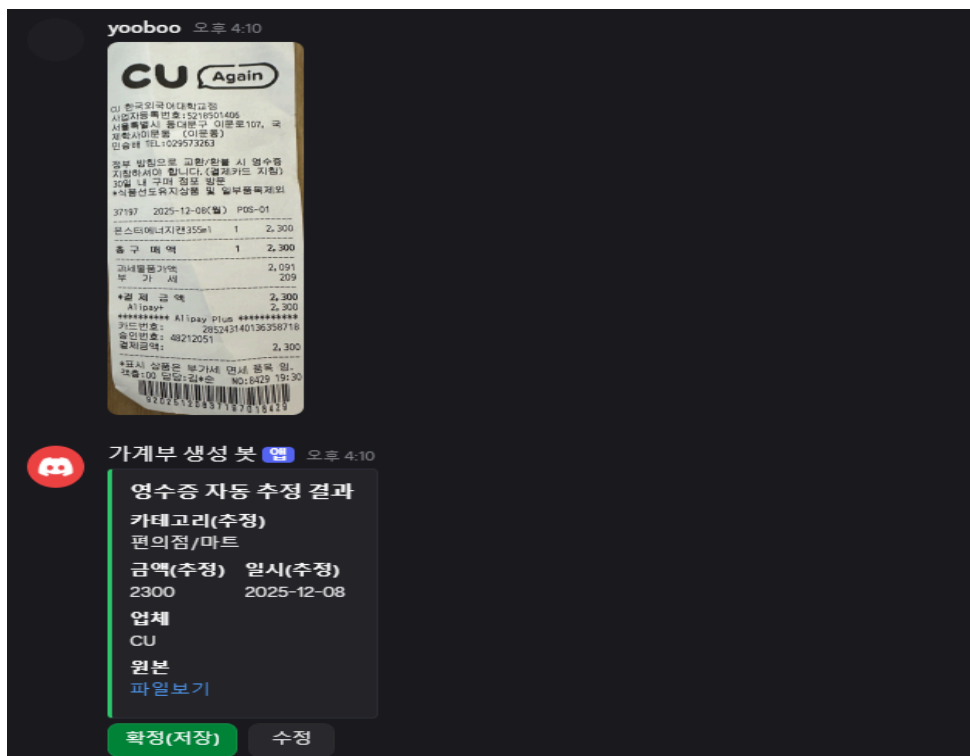
- 날짜: 2025-12-03
- 가게: 명도날드 (매우 이상하게 인식....)
- 금액: 8700
- 카테고리: 기타 (vendor map에 없어서 규칙에 따라 기타 처리)

○ 실제로는 식비 성격 → 수정 버튼을 눌러 카테고리를 식비로 직접 지정 / 가게명, 날짜 수정



4. CU 영수증

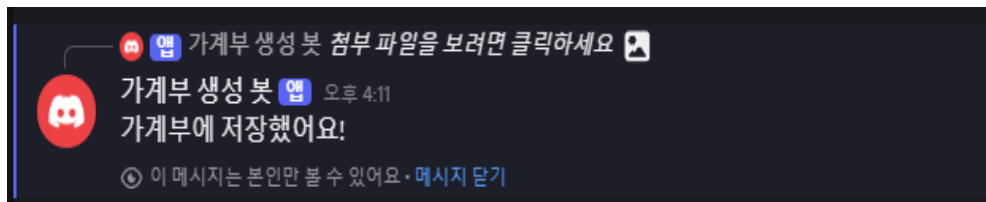
- 입력: CU 영수증 사진



- 모델 추정:

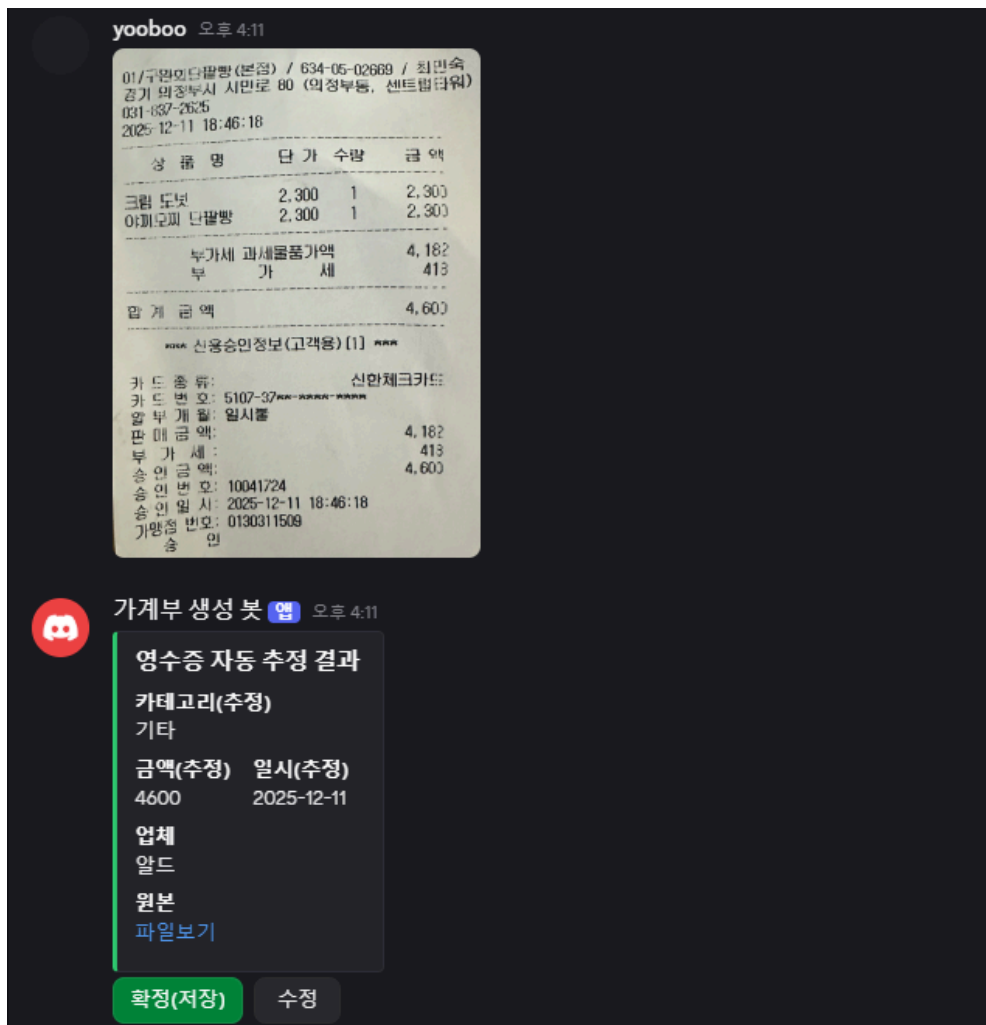
- 날짜: 2025-12-08
- 가게: CU
- 금액: 2300
- 카테고리: 편의점/마트(`vendor_map.csv`에서 `cu` 매칭)

- 오류 없이 잘 추정해냄 → 저장 버튼 눌러 ledger.csv에 저장



5. 빵집 영수증

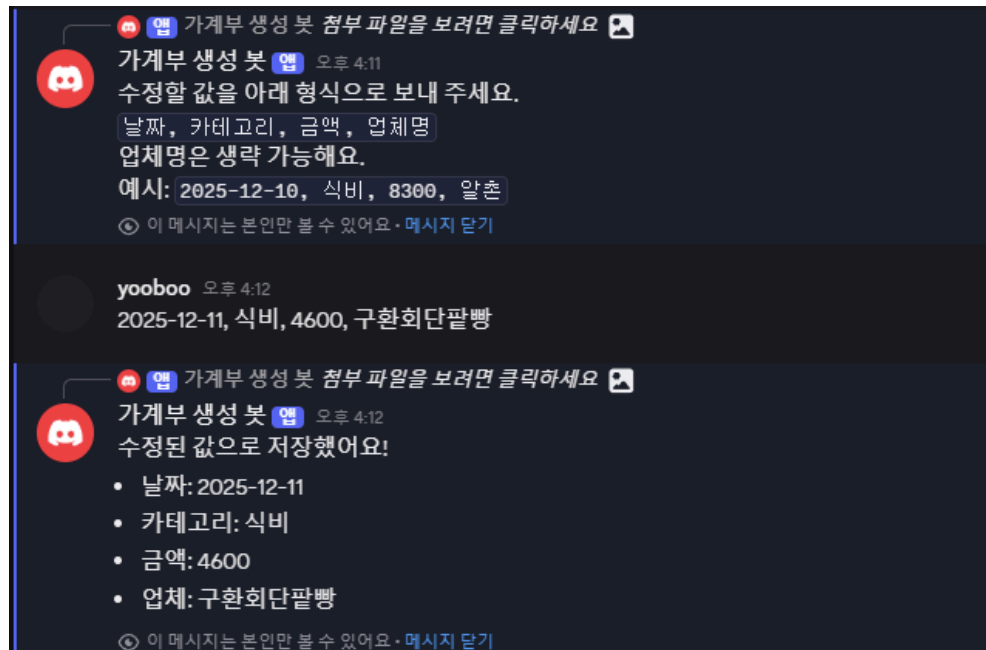
- 입력: 구환회단팔빵 영수증 사진



- 모델 추정:

- 날짜: 2025-12-11
- 가게: 알드 (매우 이상하게 추정해냄 ...)
- 금액: 4600
- 카테고리: 기타

- 카테고리, 업체명 오류 → 수정 버튼 눌러 수정하여 ledger.csv에 저장



결론: 대부분 금액은 항상 정확하게 추정해내는 모습을 보였습니다. 처음에 모델을 만들었을 때는 금액조차도 추측을 못해냈었는데, 금액 부분은 발전된 모습을 보였습니다. 날짜는 가끔씩 추측을 잘 못해내는 모습을 보였는데, 이는 영수증마다 품질이 달라 숫자가 훼손되어 그런 경향이 있는 것 같습니다. 가게명은 아무래도 사전에 학습되어 있지 않은 새로운 가게명이면 추측을 해내기가 어렵고, 이에 따라 카테고리 추측도 어려워 새로운 가게명 및 카테고리 추측 부분에서는 좋은 결과를 보여주지 못하였습니다.

!가계부, !이번주, !오늘 명령어로 가계부가 제대로 생성되는 것을 확인하였습니다.



yooboo 오후 4:12

!가계부



가계부 생성 봇 앱 오후 4:12

📅 2025-12 월 가계부 요약

총 지출: 27,900원

카테고리별 지출:

- 식비 : 13,300원 (47.7%)
- 편의점/마트 : 8,300원 (29.7%)
- 기타 : 6,300원 (22.6%)

최근 5건:

- 2025-12-06 PARIS BAGUETTE 8,700원 (식비)
- 2025-12-08 ARTBOX 6,300원 (기타)
- 2025-12-08 CU 2,300원 (편의점/마트)
- 2025-12-09 CU 6,000원 (편의점/마트)
- 2025-12-11 구환회단팔빵 4,600원 (식비)



yooboo 오후 4:12

!이번주



가계부 생성 봇 앱 오후 4:12

📅 2025-12-08 ~ 2025-12-14 주간 가계부

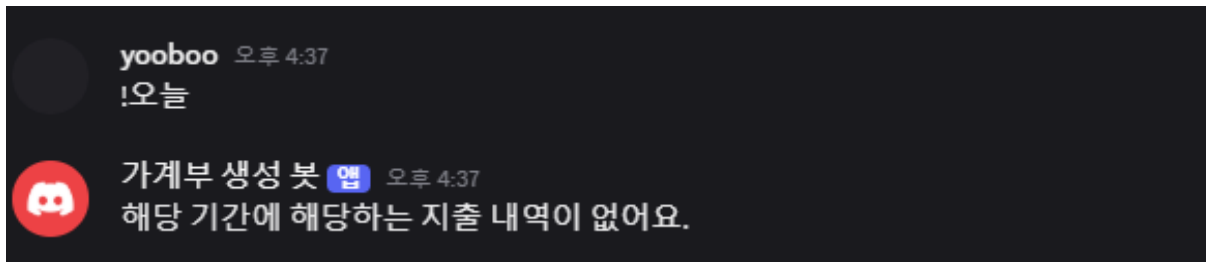
총 지출: 19,200원

카테고리별 지출:

- 편의점/마트 : 8,300원 (43.2%)
- 기타 : 6,300원 (32.8%)
- 식비 : 4,600원 (24.0%)

최근 5건:

- 2025-12-08 ARTBOX 6,300원 (기타)
- 2025-12-08 CU 2,300원 (편의점/마트)
- 2025-12-09 CU 6,000원 (편의점/마트)
- 2025-12-11 구환회단팔빵 4,600원 (식비)



5. 배운 점 및 개선 방향

5.1 문제점 및 한계

1. 데이터가 매우 적음

- 실제 영수증 30장 + 증강 150장 수준의 작은 데이터셋이라 모델이 특정 가게/패턴에 과도하게 특화될 위험이 있습니다.
- epoch를 4로 제한해 과적합을 어느 정도 막았지만, 아직도 새로운 형태의 영수증에는 약한 모습을 보였습니다.

2. 가게 이름 인식의 불안정성

- 한글 가게 이름을 완벽하게 인식하지 못하는 경우가 많았습니다.
- 이 때문에, `vendor_map`에 등록되지 않은 가게들은 카테고리 분류가 잘 되지 않았습니다.
- 결국 “기타” 비율이 늘어나며, 사용자가 직접 카테고리를 수정해야 하는 상황이 자주 발생하게 되었습니다.

3. 에러 처리 및 예외 상황

- OCR 결과에서 JSON이 깨져 나오는 경우 파싱에 실패하게 됩니다. 이런 경우 “JSON을 찾지 못했다”는 메시지를 띄우고 있지만, 사용자 관점에서 더 친절한 안내가 필요하다고 생각이 됩니다.

5.2 개선 방향

1. 데이터 확장

- 더 많은 영수증을 수집하고, 실제로 자주 사용하는 가게(학교 주변, 프랜차이즈 등)를 집중적으로 추가 학습시키면 성능을 높일 수 있을 거라 생각합니다.
- 실제 사용 중 쌓이는 `ledger.csv`를 활용해 `semi-supervised` 방식으로 추가 학습도 가능할거라 생각합니다.

2. 가게 이름 → 카테고리 분류 모델 분리

- 지금은 단순 키워드/사전 기반인데, 별도의 텍스트 분류 모델(예: BERT 기반)을 두어 “가게명 + 일부 문구”를 입력하면 카테고리를 예측하도록 만들 수 있을거라 생각합니다.

*발표 시간 교수님께서 주신 질문에 대한 답변을 고민해본 결과

Q. 기존 토스에 있는 기능과 다른 점이 무엇인지 ?

A. 토스에 해당 기능이 존재하는지 물렸었습니다. 그래서 발표가 끝난 뒤 토스에서 해당 기능을 찾아보았습니다. 토스는 카드사에서 오는 거래 데이터를 기반으로 자동 분류를 해주는 서비스이고, 제 모델은 영수증 이미지 자체를 읽어서 디스코드 안에서 가게부를 만드는 이미지 기반 및 커스터마이징이 가능한 시스템이라고 생각이 되어 입력 자체가 다른 개념이라고 생각합니다.

또한 토스의 기능에서는 현금으로 결제한 정보, 친구와의 **1/N** 결제 등의 정보는 자동 반영이 어렵지만, 제 모델은 이런 부분을 조금이나마 수월하게 반영할 수 있을거라 생각이 됩니다.

5.3 개인적으로 배운 점

- 딥러닝 모델을 “학습만 시키는 것”과 “실제 서비스로 만드는 것”의 차이를 체감하였습니다.
- 작은 버그(경로, 토큰 ID, 디바이스 설정)가 전체 파이프라인을 쉽게 망가뜨릴 수 있으므로 단계별로 분리해서 디버깅하는 습관이 중요하다는 것을 배웠습니다.
- ML 모델만으로 완벽한 해결을 기대하기보다는, 규칙 기반 방식(**vendor_map**, 키워드 규칙)과 적절히 조합하는 하이브리드 접근이 실제 서비스에서는 훨씬 실용적일 수 있다는 점을 경험하였습니다.
- 마지막으로, 디스코드 봇이라는 친숙한 인터페이스 덕분에 모델을 “매일 실제로 사용할 수 있는 형태”로 연결하는 경험을 할 수 있었습니다.