2학기 스타트캠프의 ai미니프로젝트

완벽합니다! 이제 **ai-dev-deployment MLOps 프로젝트 제출을 위한 체크리스트**를 만들어드리겠습니다.

🚀 ai-dev-deployment MLOps 프로젝트 제출 가이드

📋 전체 작업 체크리스트

Phase 1: 환경 설정 및 기본 실행 (필수)

✅ 1-1. 로컬 환경 구축

```
cd ai-dev-deployment
pip install -r requirements.txt
```

확인 사항:

- ☐ PyTorch GPU 버전 정상 설치 확인
- ☐ MLflow 설치 확인
- ☐ BentoML 설치 확인

✅ 1-2. MLflow 서버 실행

```
# 터미널 1: MLflow UI 서버 시작
mlflow server --host 127.0.0.1 --port 8080
```

확인 사항:

- ☐ http://localhost:8080 접속 가능
- ☐ MLflow UI 화면 캡처

Phase 2: 모델 학습 및 추적 (필수)

✅ 2-1. 첫 번째 모델 학습 실행

```
# 터미널 2: 모델 학습
python train_app.py
```

확인 사항:

- [ ] MNIST 데이터셋 다운로드 완료
- [ ] 5 epoch 학습 완료
- [ ] MLflow UI에서 실험 결과 확인
- [ ] `mnist-classifier` 모델 등록 확인

스크린샷 필수:

- [ ] 터미널 학습 로그
- [ ] MLflow UI - Experiments 페이지
- [ ] MLflow UI - Models 페이지

✅ 2-2. 하이퍼파라미터 튜닝 실험 (3회 이상)

model_train.py 수정:

실험 1 (Baseline):

```
params = {
    "epochs": 5,
    "batch_size": 64,
    "learning_rate": 0.001,
    "optimizer": "Adam"
}
```

실험 2 (Learning Rate 변경):

```
params = {
    "epochs": 5,
    "batch_size": 64,
    "learning_rate": 0.0001,  # 변경
    "optimizer": "Adam"
}
```

실험 3 (Batch Size 변경):

```
params = {
    "epochs": 5,
    "batch_size": 128,  # 변경
    "learning_rate": 0.001,
    "optimizer": "Adam"
}
```

각 실험마다:

```
python train_app.py
```

확인 사항:

- [ ] MLflow에 3개 이상의 Run 생성
- [ ] 각 Run의 파라미터 차이 확인

- [ ] Loss 그래프 비교
- [ ] 최적 모델 선택 및 근거 문서화

스크린샷 필수:

- [ ] MLflow Experiments 비교 화면
- [ ] 각 Run의 Metrics 그래프

Phase 3: BentoML 모델 서빙 (필수)

✅ 3-1. BentoML 서비스 실행

```
cd bentoml
bentoml serve app.py:PyTorchModelService --reload
```

확인 사항:

- [ ] 서비스 시작 (http://localhost:3000)
- [ ] Swagger UI 접속 가능
- [ ] `/predict` 엔드포인트 확인

✅ 3-2. API 테스트

테스트 1: 단일 예측

```
curl -X POST http://localhost:3000/predict \
  -H "Content-Type: application/json" \
  -d @../sample-predict-data.json
```

테스트 2: 배치 예측

```
curl -X POST http://localhost:3000/batch_predict \
  -H "Content-Type: application/json" \
  -d @../sample-predict-data.json
```

확인 사항:

- [ ] API 응답 정상 (200 OK)
- [ ] 예측 결과 확인 (10개 클래스 확률)
- [ ] 응답 시간 측정

스크린샷 필수:

- [ ] BentoML Swagger UI
- [ ] API 요청/응답 결과

Phase 4: Docker 컨테이너화 (필수)

✅ 4-1. Docker Compose 설정 수정

docker-compose.yml 수정:

```
# MLflow S3 설정 제거 (로컬 테스트용)
environment:
  - MLFLOW_BACKEND_STORE_URI=sqlite:///mlflow.db  # 변경
  # S3 관련 설정 주석 처리
```

✅ 4-2. Docker Compose 실행

```
# GPU 없이 실행하는 경우 docker-compose.yml에서 nvidia 관련 설정 제거
docker-compose up -d mlflow postgres prometheus grafana
```

확인 사항:

- [ ] 모든 컨테이너 정상 실행 (`docker ps`)
- [ ] MLflow: http://localhost:5000
- [ ] Prometheus: http://localhost:9090
- [ ] Grafana: http://localhost:3001 (admin/admin)

스크린샷 필수:

- [ ] `docker ps` 결과
- [ ] 각 서비스 웹 UI

Phase 5: 모니터링 설정 (권장)

✅ 5-1. Prometheus 설정 확인

config/prometheus/prometheus.yml 확인

✅ 5-2. Grafana 대시보드 구성

1. Grafana 접속 (http://localhost:3001)
2. Data Source 추가:
   - Type: Prometheus
   - URL: http://prometheus:9090
3. 대시보드 Import:
   - config/grafana/dashboards_json/ai_dashboard.json 업로드

확인 사항:

- [ ] Prometheus 데이터 수집 확인
- [ ] Grafana 대시보드 표시
- [ ] 시스템 메트릭 (CPU, Memory) 확인

스크린샷 필수:

- [ ] Grafana 대시보드 전체 화면

Phase 6: DVC 데이터 버전 관리 (선택)

✅ 6-1. DVC 초기화

```
pip install dvc dvc-s3
dvc init
```

✅ 6-2. 대용량 파일 추적

```
# 모델 파일 DVC로 관리
dvc add models/model.pth
dvc add data/large_dataset.csv

# Git에 메타데이터만 커밋
git add models/model.pth.dvc data/large_dataset.csv.dvc
git commit -m "Add DVC tracking for models and data"
```

✅ 6-3. DVC 파이프라인 실행

```
# 파이프라인 실행 (dvc.yaml 기반)
dvc repro
```

확인 사항:

- [ ] `.dvc` 파일 생성 확인
- [ ] DVC 파이프라인 실행 성공
- [ ] 메트릭 파일 생성 (metrics/train_metrics.json)

Phase 7: 결과 정리 및 문서화 (필수)

✅ 7-1. 실험 결과 분석 문서 작성

작성 내용:

```
## 실험 결과 요약

### 1. 모델 학습 실험
| 실험 번호 | Learning Rate | Batch Size | Epochs | Final Loss | 소요 시간 |
```

```
|----------|--------------|-----------|--------|-----------|----------|
| Run 1    | 0.001        | 64        | 5      | 0.xxxx    | xx초     |
| Run 2    | 0.0001       | 64        | 5      | 0.xxxx    | xx초     |
| Run 3    | 0.001        | 128       | 5      | 0.xxxx    | xx초     |

### 2. 최적 모델 선택
- 선택한 모델: Run X
- 선택 이유: [Loss가 가장 낮고, 학습 안정성이 높음]

### 3. BentoML API 성능
- 평균 응답 시간: xx ms
- 처리량: xx requests/sec

### 4. 발견한 문제 및 해결
1. [문제]: MLflow 연결 실패
   [해결]: 서버 포트 충돌 해결 (8080 → 8081)

2. [문제]: Docker GPU 마운트 오류
   [해결]: CPU 모드로 전환
```

✅ 7-2. README 업데이트

README.md에 추가:

- [ ] 실제 실행 스크린샷 5개 이상
- [ ] 발생한 오류 및 해결 방법
- [ ] 하이퍼파라미터 실험 결과
- [ ] 성능 벤치마크 결과

✅ 7-3. 최종 체크

제출 전 확인:

- [ ] 모든 스크린샷 저장 (screenshots/ 폴더)
- [ ] 코드 변경 사항 Git 커밋
- [ ] requirements.txt 최신화
- [ ] 실행 방법 문서화 완료

📊 최종 제출물 구성

```
ai-dev-deployment/
├── README.md                    # ✅ 업데이트 완료
├── EXPERIMENT_REPORT.md         # ✅ 새로 작성 (실험 결과)
├── screenshots/                 # ✅ 새로 생성
│   ├── mlflow-ui.png
│   ├── mlflow-experiments.png
│   ├── bentoml-swagger.png
```

```
│   ├── grafana-dashboard.png
│   └── docker-containers.png
├── models/                    # ✅ 학습된 모델들
├── metrics/                   # ✅ 실험 메트릭
└── [기존 파일들]
```

⏱️ 예상 소요 시간

Phase	작업	예상 시간
1	환경 설정	30분
2	모델 학습 3회	1시간
3	BentoML 서빙	30분
4	Docker 실행	30분
5	모니터링 설정	30분
6	DVC (선택)	30분
7	문서화	1시간
총합		**4-5시간**

🎯 평가 포인트 예상

1. **MLflow 실험 추적** (30점)
   - 3개 이상의 실험 Run
   - 파라미터/메트릭 로깅
   - 모델 레지스트리 활용
2. **BentoML 배포** (30점)
   - 서비스 정상 작동
   - API 테스트 완료
   - 성능 측정
3. **Docker 컨테이너화** (20점)
   - Docker Compose 실행
   - 모니터링 스택 구성
4. **문서화** (20점)
   - 실험 결과 정리

- 스크린샷 첨부
- 문제 해결 과정 기록