

K-MELLODDY
Federated Drug Discovery
Platform Guide v1.0.1





목차

I. 연합학습

1. 연합학습 소개
2. 연합학습 워크플로우

II. 연합학습 준비

1. FDD 동작 워크플로우
2. 실행 스크립트 이해
3. 학습 파일 준비
 1. 필수 파일
 2. 비필수 파일
4. 클라이언트 데이터 제공 방식

III. FDD 플랫폼 이용방법

IV. 연합학습 주의사항

연합학습

연합학습 소개

연합학습의 기본 개념

- 연합학습은 기존의 인공지능 학습 방법과 달리, **분산된 환경에서 학습**을 진행합니다.
- 분산된 데이터(및 제공자)와 학습환경을 활용하며, 분산환경에서 학습이 완료된 인공지능 모델의 **가중치 정보만 모아 집계**합니다.
- 집계가 완료된 모델은 **모든 데이터를 학습한 것과 근사치의 성능**을 보입니다.
- 모델이 각 레이어별로 저장한 가중치와 편향은 결국 수치로 표현되므로, 이를 **평균내어 (FedAVG)** 서로 다른 데이터를 학습한 모델도 반영할 수 있다는 것이 핵심 idea 입니다.

연합학습 관련 용어

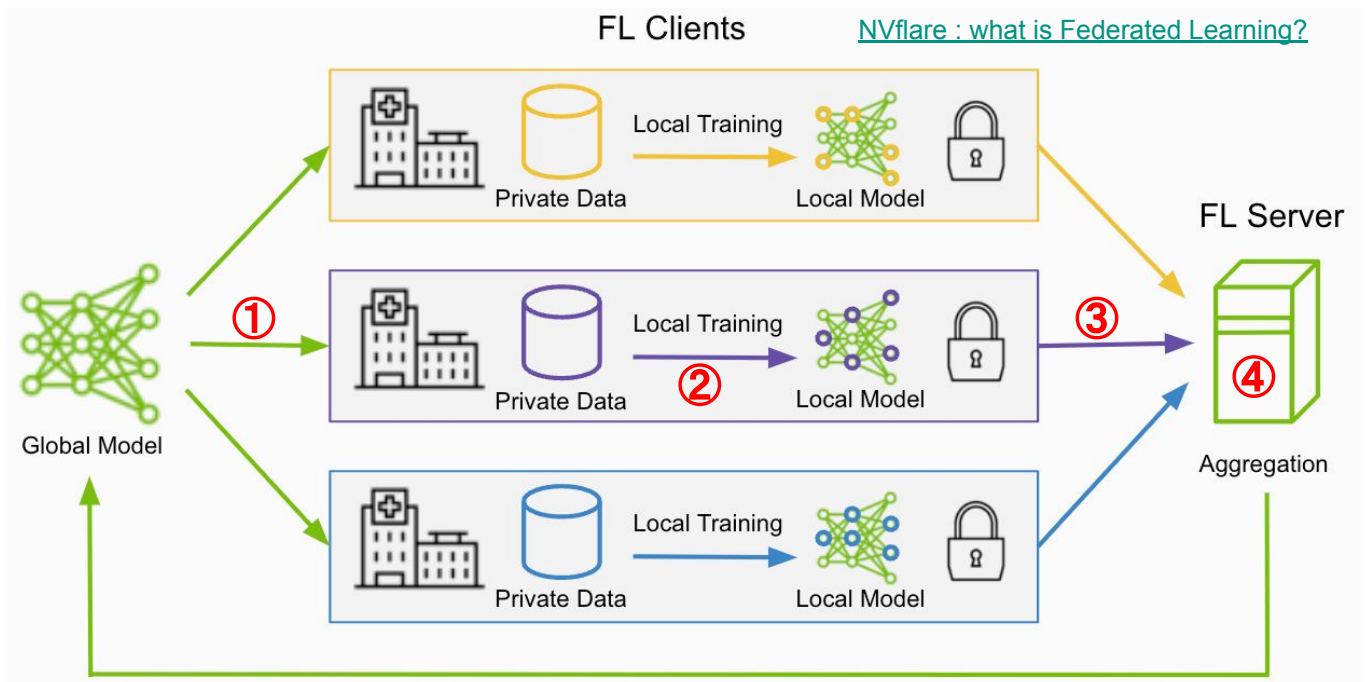
- **중앙집중식 학습** : 기존 방식대로 데이터를 수집/저장하여 하나의 서버에서 학습
- **연합학습** : 여러 서버에 분산된 데이터를 이동 없이 학습 가능한 학습 방법
- **연합학습 클라이언트** : 데이터를 보유하고 직접제공없이 공동 학습에 참여하는 참여자
- **연합학습 서버** : 각 클라이언트에서 학습된 모델을 집계하는 역할을 하는 서버
- **집계(aggregation)** : 각 클라이언트에서 학습된 모델 가중치를 합산하는 것
- **글로벌 모델** : 각 클라이언트에서 학습된 내용이 모두 집계, 반영된 공통 모델

연합학습 장점

1. 참여자는 데이터 공개 또는 유출위험 없이 학습과 모델생성만 참여 가능
2. 다기관 협력 학습을 통해 **모델의 일반화** 성능 향상
3. 특수한 **소수의 데이터**도 효과적으로 활용 가능
4. 데이터에 대한 **통제 권한을 유지**하면서 기관 간 신뢰 기반의 협력 가능
5. **참여자의 데이터 특성이 반영**되어 개인화된 AI 모델의 적용성 향상 기대
6. 협력 인프라 유지를 통해 **지속적인 학습과 모델 성능 향상** 기대

연합학습 워크플로우

연합학습의 진행 과정



⑤ 사용자가 설정된 Round 횟수에 따라 순환적 학습 가능

1. 서버는 학습할 모델을 각 클라이언트에 배포한다
2. 클라이언트는 수신받은 모델을 스스로가 보유한 데이터로 학습한다
3. 학습이 완료된 모델을 서버로 제출한다
4. 서버는 각 클라이언트의 모델 가중치를 집계하여 글로벌 모델을 생성한다
5. 1~4가 한번의 Round 과정이며, 남은 Round 횟수에 따라 반복한다

연합학습 워크플로우

연합학습에서 교환되는 정보

연합학습시에는 다음과 같은 정보를 교환합니다.

1. 모델의 가중치 및 편향. 즉, 모델 레이어에 저장된 수치
2. 해당 모델 학습에 사용된 데이터 수
3. 모델 학습 도중 측정된 평가지표(metric)

이 중 가장 중요한 것은 흔히 **파라미터**라고 부르는 **모델의 가중치 및 편향**으로,

딥러닝 모델에는 각 층(Layer)마다 다음과 같은 내용이 저장되어있습니다.

```

▶ ~
● cnn.state_dict()
[8] ✓ 0.0s

... OrderedDict([('conv1.weight',
                    tensor([[[[-9.3193e-02, -1.8830e-01, -5.4252e-02],
                               [-3.2006e-02,  8.2363e-02, -5.3273e-02],
                               [ 6.2063e-02,  1.5661e-01,  1.9180e-01]],

                               [[ 3.6877e-02, -1.3477e-01, -8.0246e-02],
                               [-6.1978e-02, -2.3163e-02,  1.4348e-02],
                               [-1.7355e-01, -1.6389e-01, -1.1512e-02]],

                               [[-4.3043e-02,  1.0520e-01,  2.7751e-02],
                               [ 2.6104e-02,  1.0300e-01, -6.2245e-02],
                               [ 1.7043e-01, -1.4381e-01, -6.2790e-02]]]]),

```

결국 연합학습이란 위와 같은 수치로 이루어진 텐서 행렬을 수집해서, 집계하는 방법입니다.

따라서 모델이 저장하는 파라미터만 전송될 뿐, **실제 데이터가 이동하지는 않습니다.**

연합학습 워크플로우

연합학습 집계 알고리즘 상세

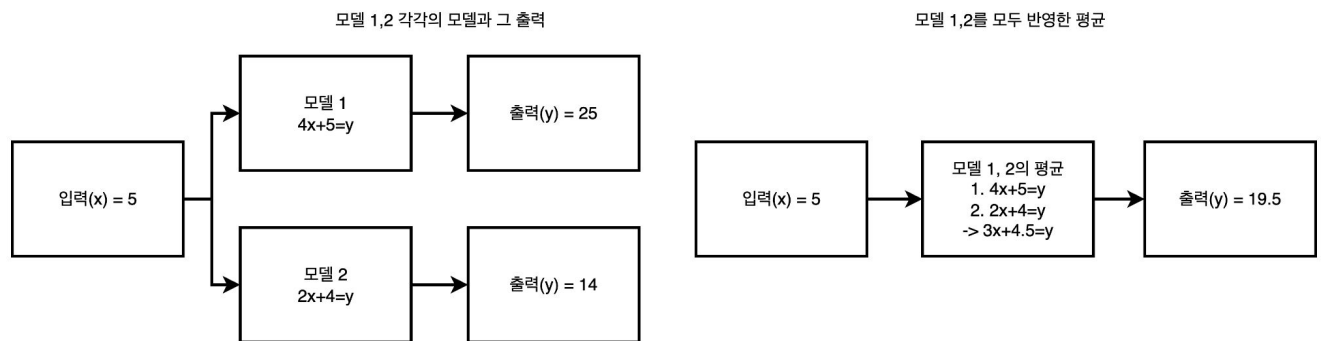
그렇다면, 아까와 같은 텐서 행렬을 수집하여 어떻게 사용하는 것일까요?

모델의 가중치와 편향이란 결국, 어떠한 **입력으로부터 출력으로 변환하는 보정값**이라는 데 그 핵심 idea가 있습니다

$ax+b=y$ 라는 아주 간단한 선형회귀 모델이 있다고 가정해 봅시다.

모델 1,2는 각각 다른 가중치를 가지고 있습니다.

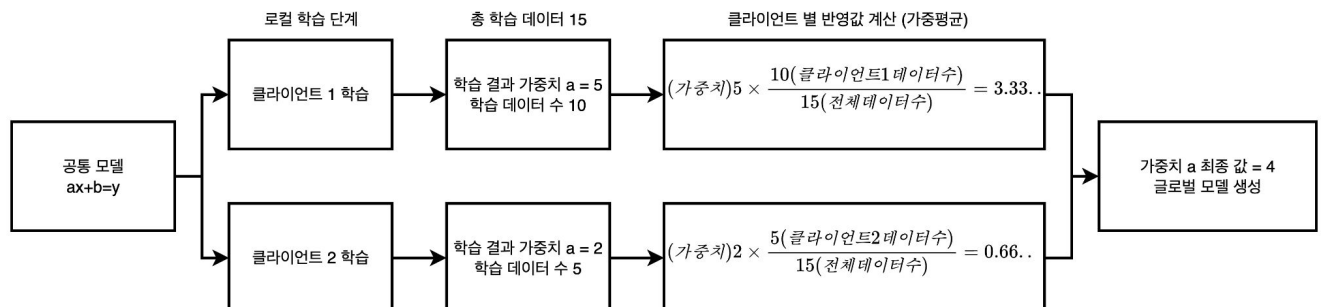
여기서 우리는 두 모델의 **가중치와 편향만 가져와 이용**할 수 있습니다.



우측처럼 두 모델을 평균내어 **새로운 모델을 생성하는 것이 연합학습의 집계**입니다.

실제 집계시에는 수만~수억개에 달하는 파라미터를 모두 연산하게 됩니다.

아래는 연합학습 집계알고리즘 중 주로 사용되는 FedAVG에 대한 예시입니다.



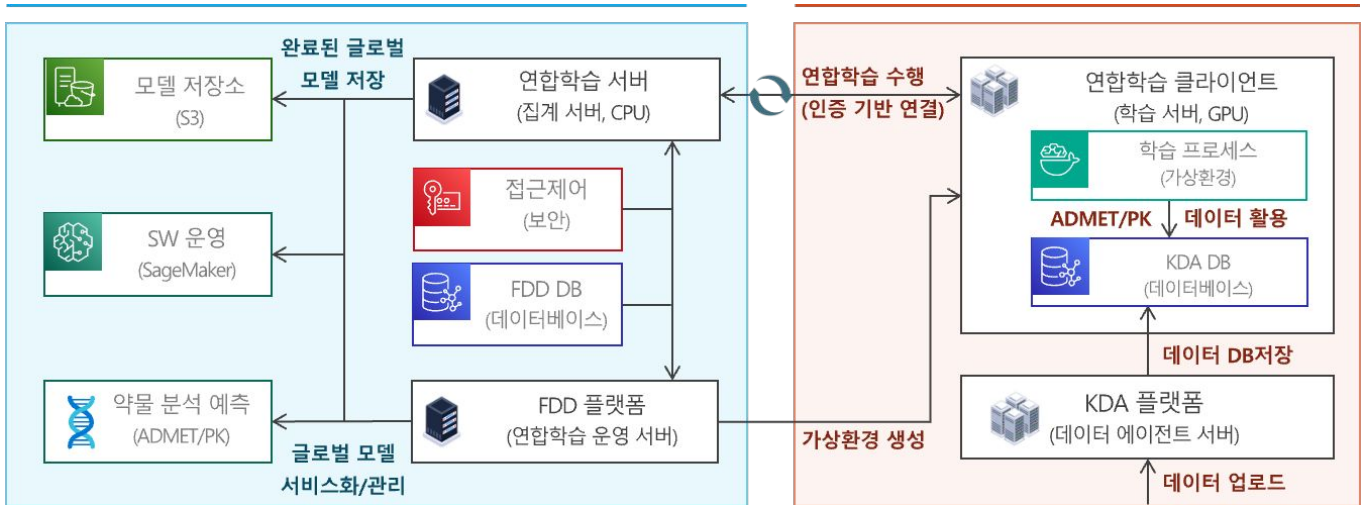
FedAVG는 단순 평균이 아닌 **학습량에 따라 가중평균을 사용**한다는 특징이 있습니다.

즉, 데이터를 많이 학습한 클라이언트는 모델에 더 많은 영향을 주게 됩니다.

연합학습 준비

FDD 동작 워크플로우

FDD 플랫폼의 전체 구조



사용자의 준비 사항

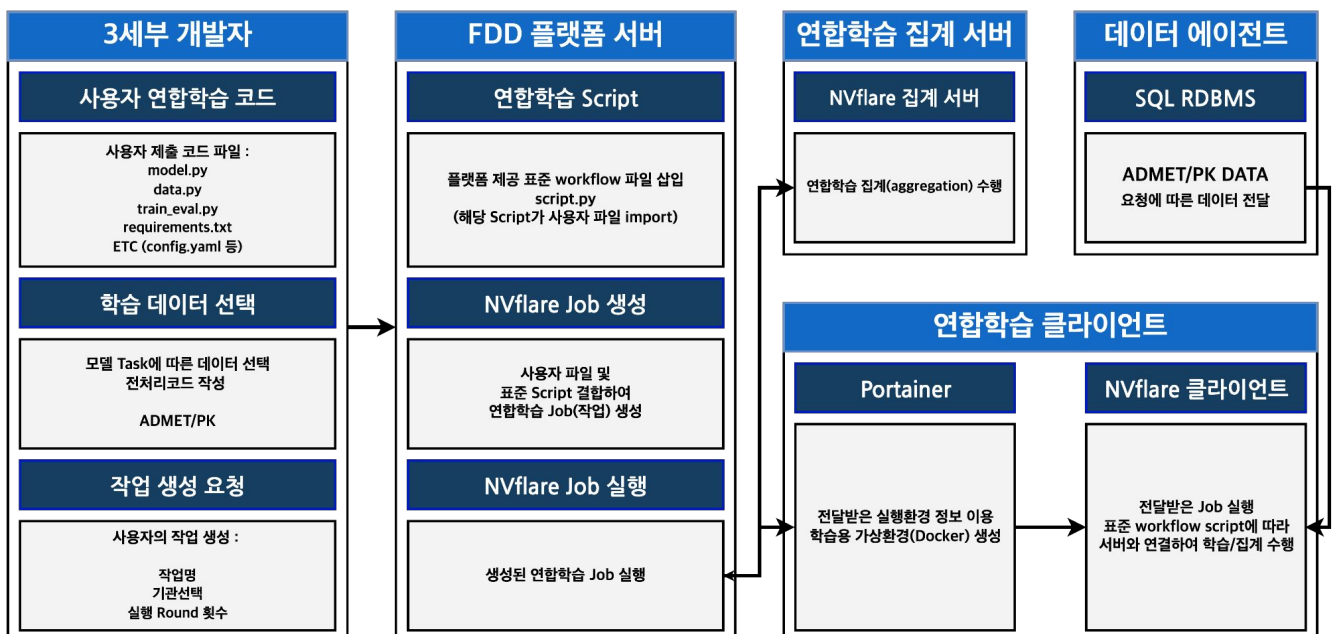
필수 준비요소

1. model.py
2. data.py
3. train_eval.py
4. requirements.txt

비필수 준비요소 (선택사항)

1. config.yaml
2. 모델 사전학습 파일
3. 기타 추가 파일

FDD 플랫폼의 워크플로우



실행 스크립트 이해

script.py

script.py는 FDD 플랫폼 시스템에서 사용자 코드에 **자동으로 추가하는 파일**입니다

해당 파일에는 클라이언트 내에서 이뤄지는 연합학습 동작이 정의되어 있습니다.

script.py의 동작과 순서

1. 사용자 업로드 파일 **import**
2. 연합학습 서버와의 연결/통신
3. 사용자의 설정 파일 내용을 **load**
4. KDA로부터 데이터 가져오기
5. 사용자의 **data_loader()** 함수 실행
6. 사용자의 **model_loader()** 함수 실행
7. 설정된 Round 동안 반복
 - 7.1. 서버로부터 글로벌 모델을 받아 파라미터 덮어쓰기
 - 7.2. 사용자의 **train()**, **evaluate()** 함수 실행
 - 7.3. 모델 파라미터 추출하여 서버로 전송

script.py 는 사용자의 모델, 데이터, 학습 로직을 받아 표준 동작을 하도록 합니다

script의 사용자 코드 **import** 및 관련 **args / return**에 대하여 [코드 예시](#)를 참조하세요

학습 파일 준비

필수 파일

FDD 플랫폼에서 연합학습을 진행하려면, **필수적으로 준비해야 하는 파일**이 있습니다

해당 파일들은 **고정된 파일명과 양식, 필수 포함 항목**이 존재합니다.

model.py

model.py 는 사용자가 **연합학습에 사용할 모델**을 정의하는 파일입니다

NVflare 모델 관리자에 등록할 모델의 클래스가 정의되어 있어야 하며,

script.py 에 모델 인스턴스를 반환하기 위하여 `model_loader()` 함수가 필요합니다.

연합학습 시, 서버와 클라이언트는 각각 모델을 불러오는 방법이 다릅니다

- **서버** : NVflare model persistor 프로세스가 모델 클래스 직접 호출, 초기화
- **클라이언트** : script.py에 작성된 `model = model_loader()` 를 사용해 반환받음

model.py의 작성방법과 주의사항에 대하여 [코드예시](#)를 참조하세요

data.py

data.py 는 사용자가 **연합학습에 사용할 데이터** 로더와 전처리를 정의하는 파일입니다

KDA로부터 전달받은 데이터를 전처리하는 과정이 정의되어 있어야 하며,

script.py 에 원하는 형태의 데이터를 반환하기 위하여 `data_loader()` 함수가 필요합니다.

`data_loader()` 함수 인자는 KDA로부터 불러온 데이터의 원본 데이터프레임,

반환은 원하는 데이터셋을 딕셔너리로 묶어 반환하여 사용합니다.

데이터의 in/output과 관련하여 상세한 예시는 [코드 예시](#)를 참조하세요

학습 파일 준비

train_eval.py

train_eval.py 는 사용자가 **모델의 학습과 평가 방법**을 정의하는 파일입니다

script.py 에서 사용될 **train()**, **evaluate()** 함수가 필요하며,

evaluate() 함수는 python 딕셔너리 형태의 metric을 반환하여야 합니다.

evaluate() 의 반환 metric은 FDD 플랫폼 UI의 성능 및 결과화면에 표시됩니다.

상세한 예시는 [코드 예시](#)를 참조하세요

requirements.txt

requirements.txt는 사용자가 작성한 코드의 **실행환경을 구성**하기 위하여 필요합니다

해당 파일은 클라이언트에서 **Docker 컨테이너 가상환경을 생성**할때 사용됩니다

작성시, 다음 요건에 맞게 작성하여야 합니다.

1. 패키지 목록 간소화
2. 각 패키지별 버전 명시
3. 패키지 URL 사용시 작성 방법 준수

자세한 사항과 예시에 대하여 [4장 연합학습 주의사항](#) 에서 확인하세요

학습 파일 준비

비필수 파일

FDD 플랫폼에서 연합학습을 진행할 때, 필수적으로 작성해야 하는 파일 이외에도, 사용자는 여러가지 유형의 파일들을 포함하여 효과적으로 학습을 수행 할 수 있습니다.

업로드 가능한 파일 유형

- 모델, 데이터, 학습 관련 config 적용
 - `config.yaml` (파일명 고정)
 - 파일이 존재할 경우, 자동으로 감지하여 설정 파일로 사용됩니다
 - 서버 : NVflare 모델 관리자가 모델 init 인자로 활용, dict 객체 전달
 - 클라이언트 : omegaconf 객체로 변환되어 제공
- 사전학습 모델 가중치 파일
 - `.safetensors`, `.ckpt`, `.pt`, `.pth`
 - 사전학습된 모델의 가중치 파일을 함께 업로드 가능합니다
 - 현재 플랫폼 환경에서는 총 1GB 까지 가능합니다 (변경 예정)
- 기타 참조, 사전, 설정 등을 위한 파일
 - `.txt`, `.json`, (config이 아닌 파일명의).`yaml`, `yaml`
 - 일반적으로 참조용 문서 또는 모델 이외의 설정을 원할 때 사용합니다

학습 파일 준비

학습 파일 구성

이전 장에서 설명드린 필수/비필수 파일 유형과 준비방법에 따라 요약하면,

업로드할 파일의 구성은 다음과 같습니다

- model, data, train_eval의 3가지 필수 python 파일
- `uv pip install -r requirements.txt` 로 실행 가능한 패키지 목록
- 필요시 설정값이 작성된 `config.yaml` 파일
- 기타 추가적인 모델 가중치 파일과 txt, json, yaml, yml 등

사용자는 이러한 파일들을 “하위 디렉토리 없이” 업로드 하여야 합니다

구체적인 좋은/나쁜 업로드 예시는 [4장 연합학습 주의사항](#) 에서 확인하세요

추가 업로드 파일 활용 방법

함께 업로드시 자동으로 감지되는 requirements.txt 또는 config.yaml 등을 제외하고,

기타 추가파일을 사용하고자 할 경우 사용방법은 다음과 같습니다.

1. `os.path.dirname(os.path.abspath(__file__))` 를 통해 현위치 절대경로 획득
2. “절대경로/파일명” 형태로 사용자 코드에서 파일 읽기 사용

자세한 사항은 [4장 연합학습 주의사항](#) 에서 확인하세요

클라이언트 데이터 제공 방식

데이터 활용 방법

실제 모델 학습이 진행될 클라이언트 내에는 데이터가 파일로 존재하지 않습니다.

따라서 KDA(K-melloddy Data Agent)의 DB로부터 데이터를 불러와 제공합니다.

사용자의 `data.data_loader()` 함수에 인자로 제공될 변수는 다음과 같은 형태입니다.

```
raw_datas = {
    "solubility" : solubility_dataframe,
    "permeability" : permeability_dataframe,
    "toxicity" : toxicity_dataframe,
    ...
}
```

이와같이 사용자에게 제공되는 데이터는 파이썬 딕셔너리 내 {테이블명 : 데이터프레임} 형태로 전달됩니다.

모든 데이터 및 테이블은 별도의 분할 세트(test, train 등)로 분할되어있지는 않으며,

사용자는 인자로 제공된 해당 변수를 활용하여 원하는 데이터 전처리 과정을 진행하면 됩니다.

데이터 구조 예시

```
raw_datas["solubility"].head(5)
```

	smiles_structure_parent	smiles_salt	chemical_id	chemical_name	smiles_id	test	test_type	test_subject	measurement_type	measurement_relation	measurement_value
0	*****	None	CHEMBL5267910	CHEMBL5267910	537	Solubility	Solubility	pH7.4	-	=	0.00076
1	*****	None	CHEMBL5268032	CHEMBL5268032	538	Solubility	Solubility	pH7.4	-	=	0.0045
2	*****	None	CHEMBL5268713	CHEMBL5268713	539	Solubility	Solubility	pH7.4	-	=	0.0002
3	*****	None	CHEMBL5270444	CHEMBL5270444	540	Solubility	Solubility	pH7.4	-	=	0.00057
4	*****	None	CHEMBL5271330	CHEMBL5271330	541	Solubility	Solubility	pH7.4	-	=	0.0011

2세부 업로드 데이터의 상세 구조는 [2025 표준 데이터포맷](#) 을 참조하세요

FDD 플랫폼 이용방법

FDD 플랫폼 가이드

데이터 정보 페이지

연합학습에 사용할 데이터의 통계정보를 확인하는 페이지입니다

[Data Information](#)
[AI Model Submit](#)
[Federated Learning](#)
[ADMET Software](#)

기관 선택 [Reset](#)






기관01 기관02 기관03 기관04 기관05 기관06 기관07 기관08 기관09 기관10 기관11 기관12 기관13 기관14 기관15
 기관16 기관17

실험 데이터 항목 선택 [Reset](#)

Solubility Permeability Brain penetration Efflux Transporter Physicochem Plasma protein binding Metabolic stability CYP Inhibition
 Plasma stability In vivo TK Human TK In vivo PK Human PK

Level 4

Absorption Distribution Metabolism Excretion Toxicity PK kinetics

Level	Test	Data	Avg.	Std.	Min.	Adjusted Min	Median	Adjusted Max	Max.	Low Bound	Q1	IQR	Q3	High Bound	Outlier	Outlier(%)	분포도
1	⊖ CYP Inhibition	100↓	60.09	33.52	0.14	0.14	50.00	100.00	100.00	-39.73	40.00	53.15	93.15	172.88	0	0.0%	
2	⊖ CYP2C9	100↓	57.43	17.95	0.14	0.14	50.00	100.00	100.00	-98.42	20.00	78.95	98.95	217.37	0	0.0%	
3	⊖ IC50	100↓	33.61	10.34	0.14	0.14	40.00	100.00	100.00	-66.35	3.46	46.54	50.00	119.81	0	0.0%	
4	Human	100↓	33.61	10.34	0.14	0.14	40.00	100.00	100.00	-66.35	3.46	46.54	50.00	119.81	0	0.0%	
3	⊖ Inhibition_10uM	100↓	97.80	95.90	90.40	92.60	100.00	100.00	100.00	90.50	96.20	3.80	100.00	105.70	1	4.3%	

1. 기관 선택을 통해 기관별 데이터 통계 확인이 가능합니다
2. 실험 데이터 항목 선택을 통해 실험별 데이터 통계 확인이 가능합니다
3. Level 설정을 통해 실험별 하위 범주를 확인 가능합니다
 - Level 1 : Test
 - Level 2 : Test_Type
 - Level 3 : Measurement_Type
 - Level 4 : Test_Subject
4. Test 이름 옆 색상별 마크는 위상단 ADMET/PK 데이터 유형 라벨을 나타냅니다
5. 데이터의 갯수는 0개, 100개 이하, 100~1000개, 1000개 이상 4가지 범주만 표시합니다
6. 평균, 최대/최소, 중앙값 등의 통계정보를 표시합니다

FDD 플랫폼 가이드

AI Model 관리 페이지

연합학습에 사용할 모델과 학습용 코드파일을 관리하는 페이지입니다

Data Information <u>AI Model Submit</u> Federated Learning ADMET Software				
		① 신규 등록		🔍 검색어를 입력하세요
번호	AI 모델명 ↑↓	내용	등록일 ↑↓	수정 / 삭제
9	CNN.v1.0	CNN	2025.10.15	✏️ 🗑️

1. 신규등록을 통해 학습할 모델의 파일을 등록합니다
2. AI 모델명 란에는 등록된 모델의 클래스명과 버전이 표기됩니다
3. 수정시 신규등록과 동일한 모달 화면이 표시됩니다
4. 모델 파일 삭제가 가능하며, 복구가 불가능합니다

FDD 플랫폼 가이드

AI Model 관리 페이지

관리페이지에서 신규 등록을 선택하면 아래와 같은 화면이 표시됩니다

신규 등록

검색어를 입력하세요.

번호 AI 모델명 등록일 수정 / 삭제

AI 모델 등록

1 AI 모델 이름 *

AI 모델의 명칭을 지정해주세요.

model.py내 클래스 이름을 작성해 주세요. 상이한 경우 모델 객체를 찾아 사용할 수 없습니다.
예) class.MLP() > MLP

2 버전 *

v1.0

3 학습 개요 *

모델의 특징을 잘 설명해주세요. 핵심 키워드를 포함하여 작성하면 검색이 용이합니다.

model.py requirements.txt data.py train_eval.py 필수 파일이 모두 필요합니다.

4

클릭 또는 드래그 앤 드롭으로 파일을 추가할 수 있습니다.

.txt, .py, .pt, .pth, .yaml, .yml, .json 파일만 등록할 수 있습니다.

취소하기 등록하기

1. 학습할 모델의 클래스 이름을 입력합니다
 - 학습할 모델은 **Pytorch**로 작성되어야 합니다
 - model.py에 작성된 **학습할 모델 클래스 명**을 정확히 작성합니다
 - 작성된 이름은 NVflare가 학습할 모델을 찾는데 사용됩니다
2. 버전은 같은 이름의 클래스 등록시 중복 방지를 위하여 사용자가 버전지정 가능합니다.
 - **이미 존재하는 모델명 + 버전**은 제한됩니다 (전역 네임스페이스)
3. 학습 개요 작성을 통해 관리화면에서 간편한 확인이 가능합니다
4. 드롭박스를 사용하여 학습할 파일을 업로드 가능합니다

1. 신규등록을 통해 새로운 연합학습 작업을 생성합니다
2. 생성한 연합학습 작업명이 표시됩니다
3. AI 모델명 란에는 해당 작업에 사용될 모델명이 보여집니다
4. 기관수는 연합학습 참여 클라이언트의 숫자를 의미합니다
5. 진행상태는 여러 상태로 표시됩니다
 - **학습시작** : 아직 학습이 시작되지 않았으며, 사용자 시작 명령 대기
 - **학습대기** : 학습 시작을 위하여 다른 작업 종료를 대기중 (**사용자당 최대 2건** 대기)
 - **학습진행** : 학습이 현재 진행중임
 - **작업취소** : 사용자가 학습을 중단하였음 (USER-Interrupt)
 - **작업실패** : 오류로 인하여 학습이 중단되었음 (ERROR)
 - **결과보기** : 학습이 성공적으로 종료되어 성능지표와 학습 모델 확인가능
6. 동작 확인창이 표시되면, Confirm을 선택하여 학습을 시작할 수 있습니다

FDD 플랫폼 가이드

연합학습 작업 관리 페이지

관리페이지에서 신규 등록을 선택하면 아래와 같은 화면이 표시됩니다



The screenshot shows the 'Federated Learning' registration form. The form is titled '연합학습 등록하기' (Federated Learning Registration). It contains the following fields and elements:

- 1**: '연합학습명' (Federated Learning Name) input field with the value 'test_1'.
- 2**: '등록된 AI 모델 선택' (Registered AI Model Selection) dropdown menu with the value 'CNN.v1.0'.
- 3**: '학습설명' (Learning Description) text area with the placeholder text '주요 키워드를 포함하여 설명글을 적어주세요'.
- 4**: '학습 ROUND' (Learning Round) input field with the value '5'.
- 5**: '저장할 북마크 제목을' (Bookmark Title) input field.
- 6**: 'ALL' button and '북마크 등록' (Bookmark Registration) button.
- 7**: Grid of buttons for selecting agencies, labeled '기관01' through '기관17'.

At the bottom of the form, there are two buttons: '취소하기' (Cancel) and '등록하기' (Register).

1. 연합학습 작업명은 자유롭게 작성가능합니다
 - 영문 및 언더바(_)를 제외한 한글, 특수문자, 공백은 제한됩니다
 - 이미 존재하는 연합학습 작업명은 제한됩니다 (전역 네임스페이스)
2. 사전에 등록하였던 AI 모델 파일을 선택 가능하며, 이름+버전 목록이 표시됩니다
3. 학습 설명을 작성하여 간편 관리가 가능합니다
4. 연합학습의 Round 횟수를 설정합니다
5. 아래 기관목록을 선택하고 북마크 제목을 입력하여 선택그룹 저장이 가능합니다
6. 생성한 북마크를 선택할 수 있습니다
7. 기관 목록 버튼을 클릭하여 하나씩 참여를 설정할 수 있습니다

FDD 플랫폼 가이드

연합학습 작업 정보 페이지

연합학습 작업을 실행하고, 진행상태 버튼을 클릭하면 작업 정보 페이지가 표시됩니다

① 기본정보

성능 & 결과

② 학습 중지

연합학습 개요 ③

연합학습명	TEST
학습 내용	TEST
학습 등록일	2025-09-17 01:51:47
등록 모델 정보	<div>모델 코드 4</div> <div>AI 모델명 CNN</div> <div>모델 설명 CNN</div> <div>모델 등록일 2025-09-17 01:51:47</div>
Data 선택 기관	2

진행 현황 ④

시작

완료

시작: 2025-09-17 04:28:50

연합학습 환경 생성

완료

소요시간: 0시간 2분

시작: 2025-09-17 04:30:34

종료: 2025-09-17 04:31:58

Round별 학습

완료

소요시간: 0시간 1분

접기 ^

시작: 2025-09-17 13:33:17

종료: 2025-09-17 13:33:31

⑤ Round 1

소요시간: 0시간 1분

시작: 2025-09-17 13:33:17

종료: 2025-09-17 13:33:24

Round 2

소요시간: 0시간 1분

시작: 2025-09-17 13:33:27

종료: 2025-09-17 13:33:28

Round 3

소요시간: 0시간 1분

시작: 2025-09-17 13:33:30

종료: 2025-09-17 13:33:31

결과 생성

완료

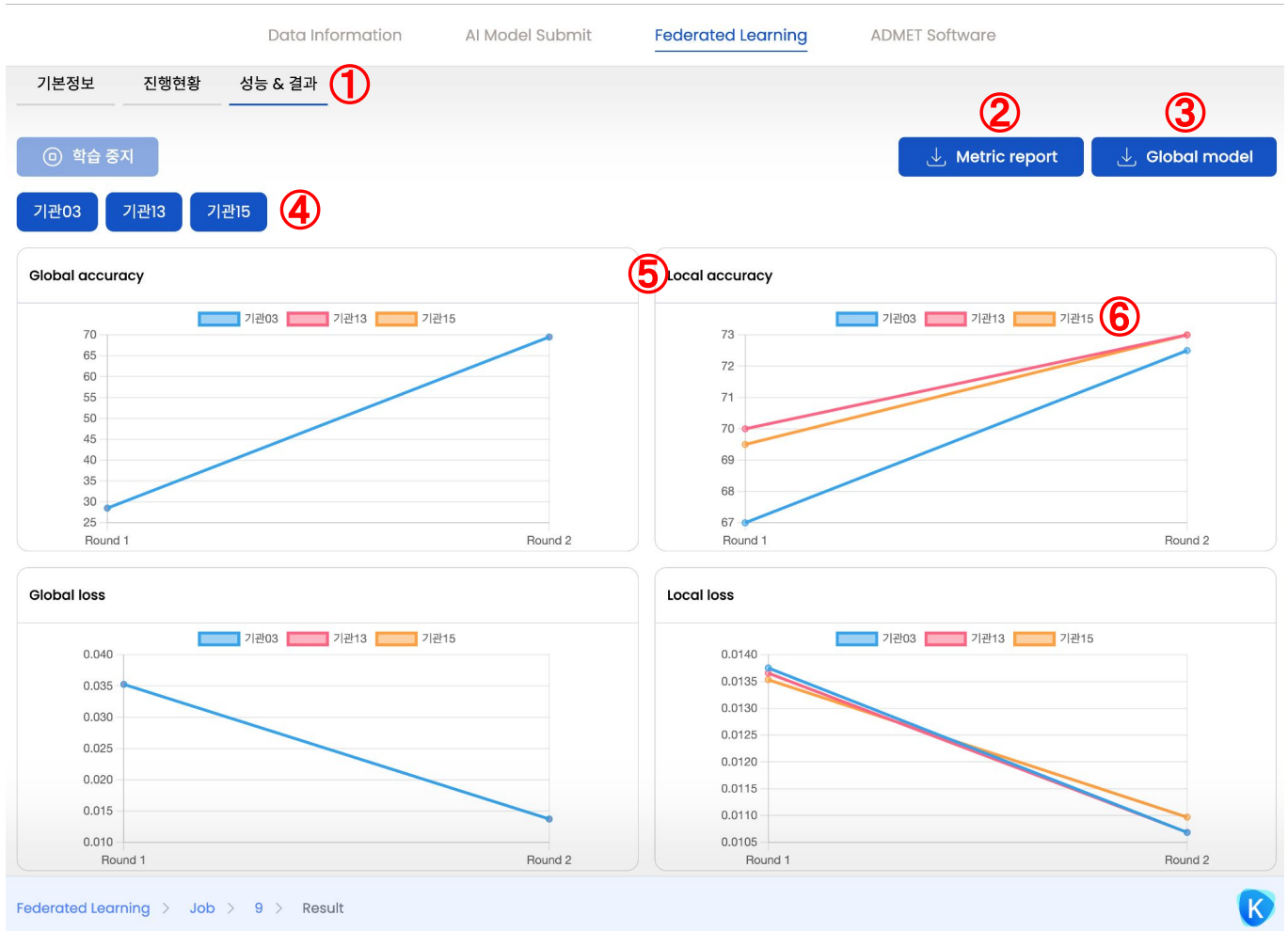
시작: 2025-09-17 04:37:30

- 기본정보 탭에서는 실행된 작업에 대한 기본정보 확인이 가능합니다
- 학습이 진행중일때는 사용자가 학습 중지를 선택하여 종료할 수 있습니다
- 연합학습 개요는 해당 작업의 기본 정보와 등록 시간 등을 표시합니다
- 진행현황 탭에서는 실행된 작업에 대한 진행상황을 표시합니다
 - 시작** : 연합학습 작업 시작 명령이 성공적으로 전달됨
 - 연합학습 환경 생성** : 각 클라이언트에 학습용 Docker 가상환경 생성
 - Round별 학습** : 연합학습 작업이 진행중
 - 결과 생성** : 학습이 완료된 모델의 .pt 파일과 성능지표를 제공중
- Round 별 진행상황 목록은 각 라운드별 경과와 소요시간을 표시합니다

FDD 플랫폼 가이드

연합학습 작업 결과 페이지

연합학습 작업이 실행되면 성능 및 결과 페이지가 활성화 됩니다



1. 성능 및 결과 페이지는 1Round 이상 진행되어 성능지표 응답을 받으면 활성화됩니다
2. 모든 학습이 종료된 후 성능지표를 요약한 csv파일을 받을 수 있습니다
3. 모든 학습이 종료된 후 결과물인 글로벌 모델을 pt파일로 받을 수 있습니다
4. 기관선택 버튼을 통해 페이지내 모든 그래프에서 일괄적으로 on/off 가능합니다
5. 동일한 성능지표의 Global Eval, Local Eval 값을 확인 가능합니다
 - **Global** : 동일한 글로벌 모델에 대해 기관의 데이터로 Evaluation 진행
 - **Local** : 기관의 데이터로 학습한 모델을 기관의 데이터로 Evaluation 진행
6. 하나의 그래프 내에서 기관별 표시를 on/off 가능합니다

연합학습 주의사항

연합학습 주의사항

requirements.txt 작성시 주의사항

사용자의 실행환경 패키지 목록 작성 방법에 대한 설명입니다.

사용자가 업로드한 requirements.txt 파일은 다음 명령을 통해 설치됩니다

```
uv pip install -r requirements.txt --system --index-strategy unsafe-best-match
```

다음과 같은 경우, 실행환경 구성이 불가능합니다

1. requirements.txt 와 별도로 특정 파일을 다운로드, 직접 설치하는 bash shell 파일
2. requirements.txt 에 로컬 파일, 즉 `package @ file:///myfile...` 과 같이 작성된 경우
3. 패키지 종속성으로 인하여 `uv pip install` 명령 한번으로 설치 불가능한 경우

다음과 같은 사항을 준수하여 작성하여야 합니다

1. 업로드 전 `uv pip install -r requirements.txt` 명령을 통해 설치 가능 테스트
2. 특정 패키지 버전 지정과 저장소 특정을 원할 경우, 다음방법을 사용
 - 2.1. 저장소 주소 `--extra-index-url` 또는 `-f` flag를 사용하여 파일 상단에 작성

- `--extra-index-url https://download.pytorch.org/whl/cu126`

- `-f https://data.pyg.org/whl/torch-2.6.0+cu126.html`

- 2.2. 개별 패키지에 대한 URL 작성

- `torch @ https://download.pytorch.org/whl/cu126/torch-2.6.0%2Bcu126-cp311-cp311-linux_x86_64.whl`

연합학습 주의사항

파일 업로드 예시 및 주의사항

파일 업로드 시 주의해야 할 사항과 **좋은/나쁜 예시** 입니다

파일 구성의 좋은 예시

```

  ok
  ├── additional_utils.py
  ├── ! config.yaml
  ├── data.py
  ├── model.py
  ├── model1.pt
  ├── model2.pth
  ├── requirements.txt
  ├── tokenizer1_config.json
  ├── tokenizer2_config.json
  ├── train_eval.py
  └── voca.txt
  
```

1. 모든 파일이 하위 디렉토리 없이 flat 구성
2. model, data, train_eval 등 필수 파일명 준수
3. config.yaml 파일명 준수
4. 기타 txt, json, pt 파일 등 정상 업로드

파일 구성의 안좋은 예시

```

  no
  ├── model1
  │   ├── model1.pt
  │   └── tokenizer1_config.json
  ├── model2
  │   ├── model2.pth
  │   └── tokenizer2_config.json
  ├── utils
  │   ├── additional_utils.py
  │   ├── ! config.yaml
  │   ├── data.py
  │   ├── model.py
  │   ├── requirements.txt
  │   ├── tokenizer1_config.json
  │   ├── tokenizer2_config.json
  │   ├── train_eval_v2.py
  │   └── voca.txt
  
```

1. 하위 디렉토리에 계층별 파일 정리
2. tokenizer1_config.json 등 중복되는 파일명 존재
3. train_eval_v2.py 필수파일명 이름 준수X
4. additional_utils 하위 디렉토리 존재, import 불가

연합학습 주의사항

추가 업로드 파일 사용시 주의사항

사용자의 추가 업로드 파일 사용 방법에 대한 설명입니다.

사용자가 업로드한 파일은 작업실행시 NVflare의 job 단위로 관리, 전송 및 실행됩니다.

전송된 모든 파일은 클라이언트의 `workspace/UUID/(작업명)` 디렉토리 내 존재합니다.

학습 시작 스크립트는 workspace 위치에서,

`python3 ./UUID/(작업명)/script.py` 명령으로 시작됩니다.

따라서 프로세스의 `pwd`는 `.workspace/`이며, 다음과 같은 문제가 발생합니다.

1. 추가 python 파일 import시, `import util.additional` 와 같은 경로 탐색 불가
2. 파일을 `load("myfile.txt")` 와 같이 사용시 `workspace/myfile.txt` 경로 입력됨

사용자 정의 코드가 원하는 파일 경로를 찾을 수 있도록, 다음과 같이 실행해야 합니다

1. `os.path.dirname(os.path.abspath(__file__))` 를 통해 현위치 절대경로 획득
2. “절대경로/파일명” 형태로 사용자 코드에서 파일 읽기 사용

코드 예시

script.py

```
# 1. 사용자 코드 import
from data import data_loader
from model import model_loader
from train_eval import train, evaluate

# 2. GPU 디바이스 설정, 사용자 코드에 .to(device) 작성필요 없음
DEVICE = torch.device("cuda:0")

def main():
    # 3. 사용자 설정 파일 config.yaml 로드, 파일이 없으면 configs = None
    configs = get_config()

    # 4. KDA로부터 데이터 GET, raw_datas : Dict[str, DataFrame]
    raw_datas = get_data()

    # 5. 사용자의 data_loader 함수 실행, data : Dict[str, Any]
    data = data_loader(raw_datas, configs)

    # 6. 사용자의 model_loader 함수 실행, model : torch.nn.Module Class
    model = model_loader(configs)

    # 7. 1회 Round 마다 반복
    while flare.is_running():

        # 7.1 서버에서 배포한 글로벌 모델 및 현재 진행 정보 수신(현재 라운드 회차 등)
        input_model = flare.receive()

        # 7.2 서버에서 배포한 글로벌 모델의 params를 현재 클라이언트 model에 로드
        model.load_state_dict(input_model.params)

        # 학습/평가 전 모델을 GPU 디바이스로 이동하므로, 사용자 코드에서 하지 않아도 좋음
        model.to(DEVICE)

        # 7.3 글로벌 모델에 대하여, 클라이언트가 가진 데이터로 평가 진행
        glob_metric = evaluate(model, data, configs)

        # 7.4 클라이언트 보유 데이터를 학습
        train(model, data, configs)

        # 7.5 클라이언트 보유 데이터를 학습한 모델에 대하여 평가 진행
        local_metric = evaluate(model, data, configs)

        # 7.6 1회 학습이 완료된 모델의 파라미터를 CPU로 이동, state_dict()를 사용하여 추출 후 전송
        output_model = flare.FLModel(params=model.cpu().state_dict())
        flare.send(output_model)
```

코드 예시

model.py

```
# 학습할 모델은 PyTorch의 torch.nn.Module 상속 또는 torch lightning이어야 함
import torch.nn as nn
import torch.nn.functional as F

# 1. 모델 init의 인자 설정 전달이 필요하지 않은 간단한 모델일 경우
# 사용자 모델 클래스와 레이어 정의, NVflare model persistor는 아래 클래스를 model.MLP 형태로 호출
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(28*28, 512)
        ...
        self.fc3 = nn.Linear(256, 10)

    def forward(self, x):
        x = self.fc1(x)
        ...
        x = self.fc3(x)
        return x

# configs 인자를 받아 모델 init 후 반환하는 model_loader 함수 정의
# 학습을 진행하는 클라이언트의 script에서 호출
def model_loader(configs: DictConfig): # configs 인자가 필요하지 않더라도, script에서는 인자 전달
    model = MLP() # configs 인자를 받았으나 사용하지 않음
    return model

# -----

# 2. 모델 init의 인자 설정 전달이 필요한 모델일 경우
class MLP(nn.Module):
    def __init__(self, configs: DictConfig): # init에 configs 인자를 전달받도록 작성합니다.
        super(MLP, self).__init__()
        # nvflare model persistor가 호출시에는 configs를 dict 타입으로 전달하므로, dict 형태로 사용
        # 또는 configs = OmegaConf.create(configs) 를 상단에 배치하여 OmegaConf 타입으로 사용

        self.args_1 = configs["args_1"] # Dict[str, Any] 형태인 configs 객체를 원하는 인수와 매칭합니다
        self.args_2 = configs["args_2"]
        ...
        self.fc3 = nn.Linear(256, 10)

    def forward(self, x):
        x = self.fc1(x)
        ...
        x = self.fc3(x)
        return x

def model_loader(configs: DictConfig): # configs 인자가 필요하지 않더라도, script에서는 인자 전달
    model = MLP(configs) # configs 인자를 받아 init에 사용
    return model
```

코드 예시

model.py

3. 사전학습 파일로부터 load_state_dict 사용이 필요한모델일 경우

```
class MLP(nn.Module):
    def __init__(self, configs:DictConfig):
        super(MLP, self).__init__()
        self.args_1 = configs["args_1"]
        self.args_2 = configs["args_2"]
        ...
        self.fc3 = nn.Linear(256, 10)

    def forward(self,x):
        x = self.fc1(x)
        ...
        x = self.fc3(x)
        return x
```

사용자 모델을 감싸는 WrapperClass 클래스 정의, NVflare model persistor는 해당 클래스 등록

```
class WrapperClass(nn.Module):
    def __init__(self, configs:DictConfig):
        self.model = MLP(configs) # configs 인자를 전달하여 모델 인스턴스 생성
        # 파일로부터 가중치 load
        self.model.load_state_dict(torch.load(f"{os.path.dirname(os.path.abspath(__file__))}/~.pt"))
        # NVflare에서 호출할 경우 class 객체만 가능하며 함수나 반환된 변수 등 객체 불가
        # 부득이하게 init 동작을 수행 가능한 wrapper class를 사용함

    def forward(self,x):
        return self.model(x)

def model_loader(configs: DictConfig):
    model = WrapperClass(configs)
    return model
```

4. huggingface로부터 from_pretrained 사용이 필요한모델일 경우

huggingface 모델 인스턴스를 감싸는 WrapperClass 클래스 정의, NVflare model persistor는 해당 클래스 등록

```
class WrapperClass(nn.Module):
    def __init__(self, configs:DictConfig):
        self.model = AutoModelForCausalLM.from_pretrained(configs.model_name) # huggingface로부터 모델 load
        # NVflare에서 호출할 경우 class 객체만 가능하며 함수나 반환된 변수 등 객체 불가
        # 부득이하게 init 동작을 수행 가능한 wrapper class를 사용함

    def forward(self,x):
        return self.model(x)

def model_loader(configs: DictConfig):
    model = WrapperClass(configs)
    return model
```

코드 예시

data.py

```
from torch.utils.data.dataloader import DataLoader

... # 전처리 과정에서 호출할 함수 정의 자유

# 원본 데이터의 데이터프레임을 전달받고, 사용자 모델에 맞게 전처리하여 반환할 data_loader 함수 정의
# 데이터활용방법 참고
def data_loader(raw_datas, configs):
    train_loader = DataLoader(raw_datas['train'], batch_size=32, shuffle=True, num_workers=4)
    test_loader = DataLoader(raw_datas['test'], batch_size=32, shuffle=True, num_workers=4)

    # 반환은 언제나 하나의 딕셔너리 변수로 묶어 반환
    data = {"train": train_loader, "test": test_loader}
    return data
```

코드 예시

train_eval.py

```
def train(model, data, configs):
    # data 딕셔너리로부터 사용할 train 세트를 선택, 사용
    train_data = data["train"]

    # configs 딕셔너리로부터 사용할 하이퍼파라미터를 선택, 사용
    epochs = configs["epochs"]
    learning_rate = configs["learning_rate"]

    for epoch in range(EPOCHS):
        model.train()
        ...

def evaluate(model, data, configs):
    # data 딕셔너리로부터 사용할 test 세트를 선택, 사용
    test_data = data["test"]

    # configs 딕셔너리로부터 사용할 하이퍼파라미터를 선택, 사용
    epochs = configs["epochs"]
    learning_rate = configs["learning_rate"]

    model.eval()
    with torch.no_grad():
        for image, label in test_data:
            ...

    # evaluation 함수는 결과물로 metrics 딕셔너리를 반환해야 합니다.
    # 사용자 정의 metric 변수를 아래 딕셔너리에 추가하여 사용 가능합니다.
    metrics = {"loss": test_loss, "accuracy": test_accuracy}
    # metrics = {"loss": test_loss, "accuracy": test_accuracy, "user_metric_1": user_metric_1}
    return metrics

# -----

# lightning 및 huggingface Trainer도 물론 사용 가능합니다
from lightning.pytorch import Trainer

def train(model, data, configs):
    # 함수호출시 trainer_build 되도록 구성해주세요
    trainer = trainer_build(configs)
    trainer.fit(model, data)

def evaluate(model, data, configs):
    # 함수호출시 trainer_build 되도록 구성해주세요
    trainer = trainer_build(configs)
    metrics = trainer.test(model, data)
    # lightning 및 huggingface 등 트레이너의 경우 기본 반환 metric 형식을 dict 형태로 변환 필요
    return metrics

def trainer_build(configs: Dict[str, Any]):
    trainer = Trainer(configs=configs["trainer"])
    return trainer
```