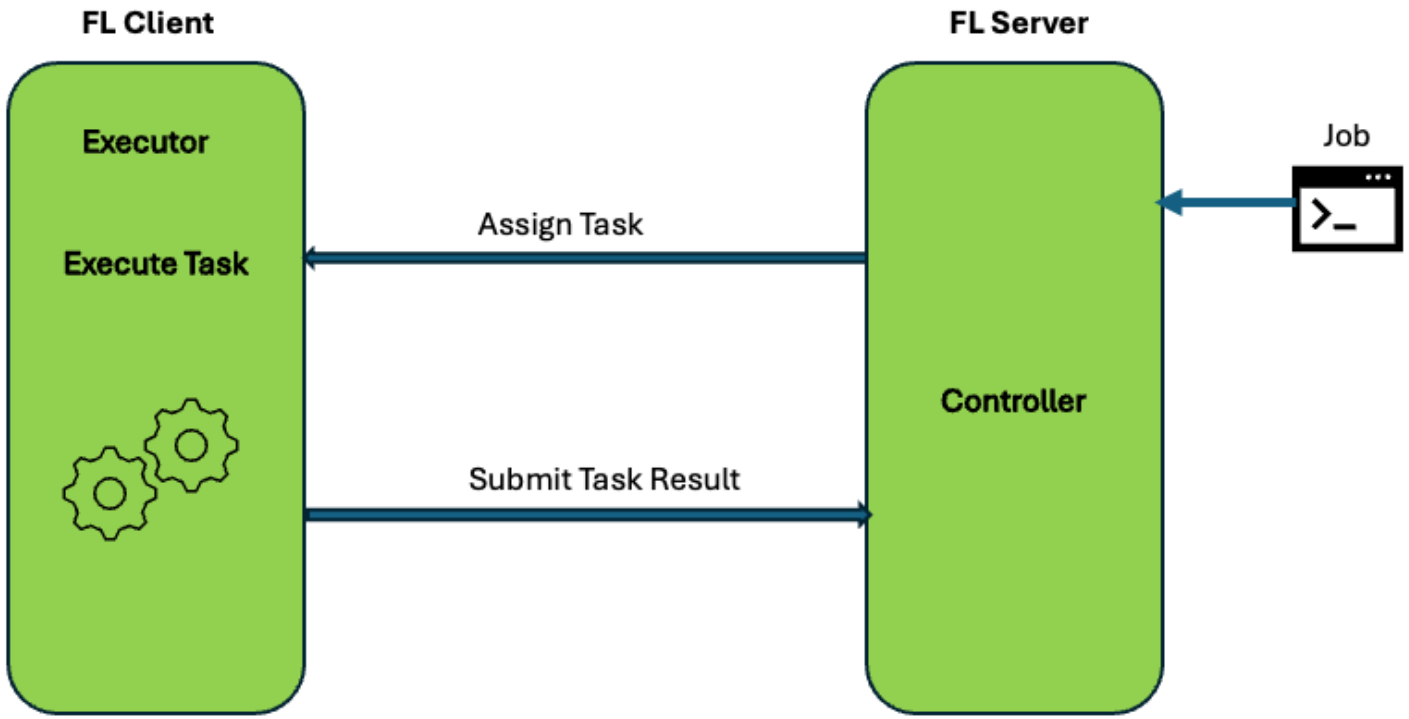
應用 NVFLARE
以PyTorch為範例
客製化聯邦式學習分析

邱彥榕

Device Require

- Unix command applications
  - Unix system
  - Using TWCC containers or VMs…
  - Docker
  - Anaconda or Miniconda related products
  - Products such as Mobaxterm

Federated Learning structure



https://docs.nvidia.com/clara/clara-train-archive/3.1/federated-learning/fl_background_and_arch.html

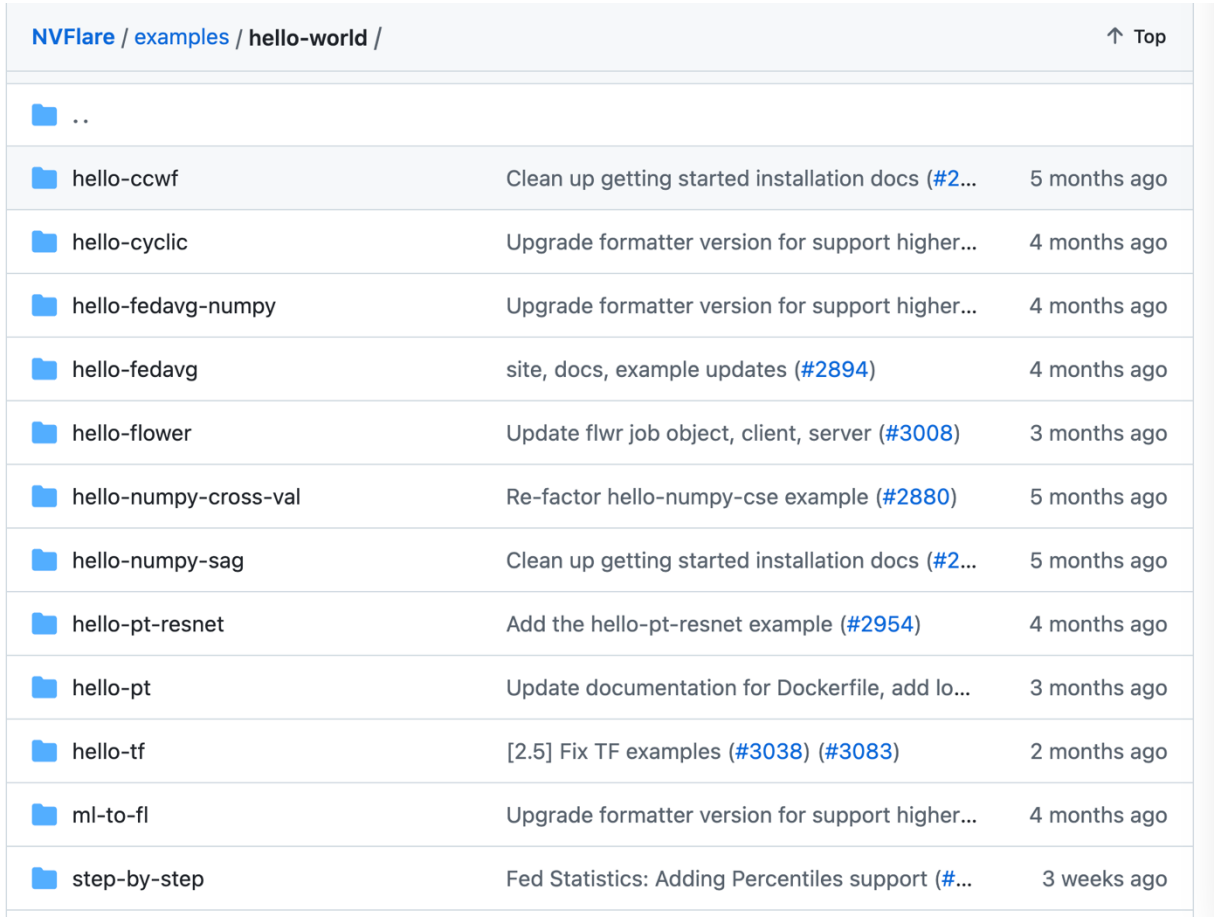
Install NVFLARE

- git clone https://github.com/NVIDIA/NVFlare.git
- python3 -m pip install --user --upgrade pip
- python3 -m pip install --user virtualenv
- python3 -m venv nvflare_YOURNAME
- source nvflare_YOURNAME /bin/activate
- python3 -m pip install nvflare

NVFlare operational mode

- Simulator and POC mode for rapid development and prototyping
  - simulator
  - poc
- FLARE API

simulator example



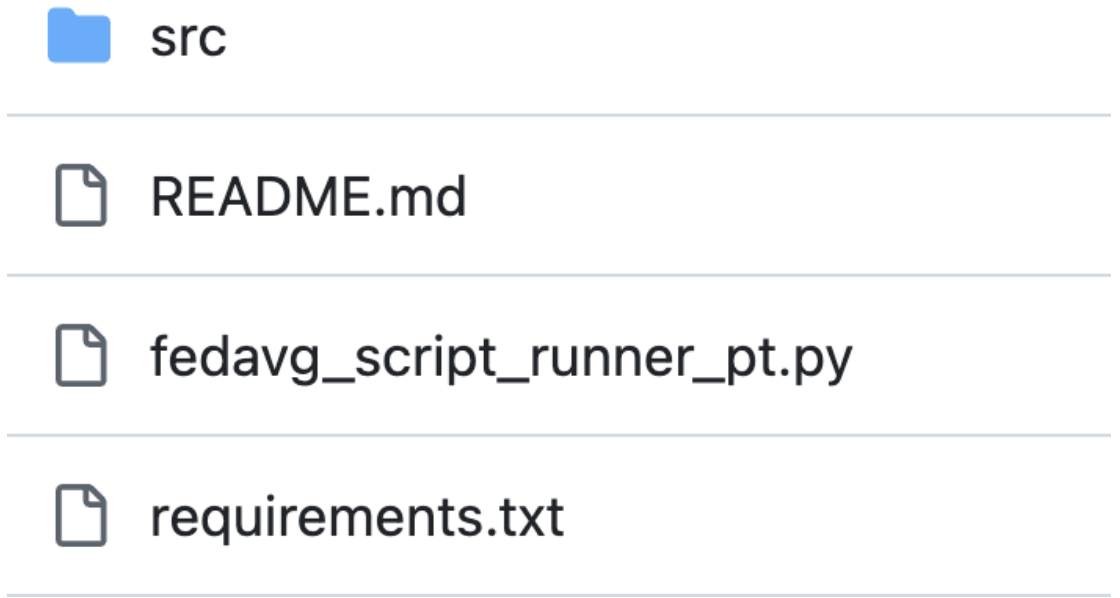
NVFlare / examples / hello-world / ↑ Top

..		
hello-ccwf	Clean up getting started installation docs (#2...	5 months ago
hello-cyclic	Upgrade formatter version for support higher...	4 months ago
hello-fedavg-numpy	Upgrade formatter version for support higher...	4 months ago
hello-fedavg	site, docs, example updates (#2894)	4 months ago
hello-flower	Update flwr job object, client, server (#3008)	3 months ago
hello-numpy-cross-val	Re-factor hello-numpy-cse example (#2880)	5 months ago
hello-numpy-sag	Clean up getting started installation docs (#2...	5 months ago
hello-pt-resnet	Add the hello-pt-resnet example (#2954)	4 months ago
hello-pt	Update documentation for Dockerfile, add lo...	3 months ago
hello-tf	[2.5] Fix TF examples (#3038) (#3083)	2 months ago
ml-to-fl	Upgrade formatter version for support higher...	4 months ago
step-by-step	Fed Statistics: Adding Percentiles support (#...	3 weeks ago

https://github.com/NVIDIA/NVFlare/tree/main/examples/hello-world

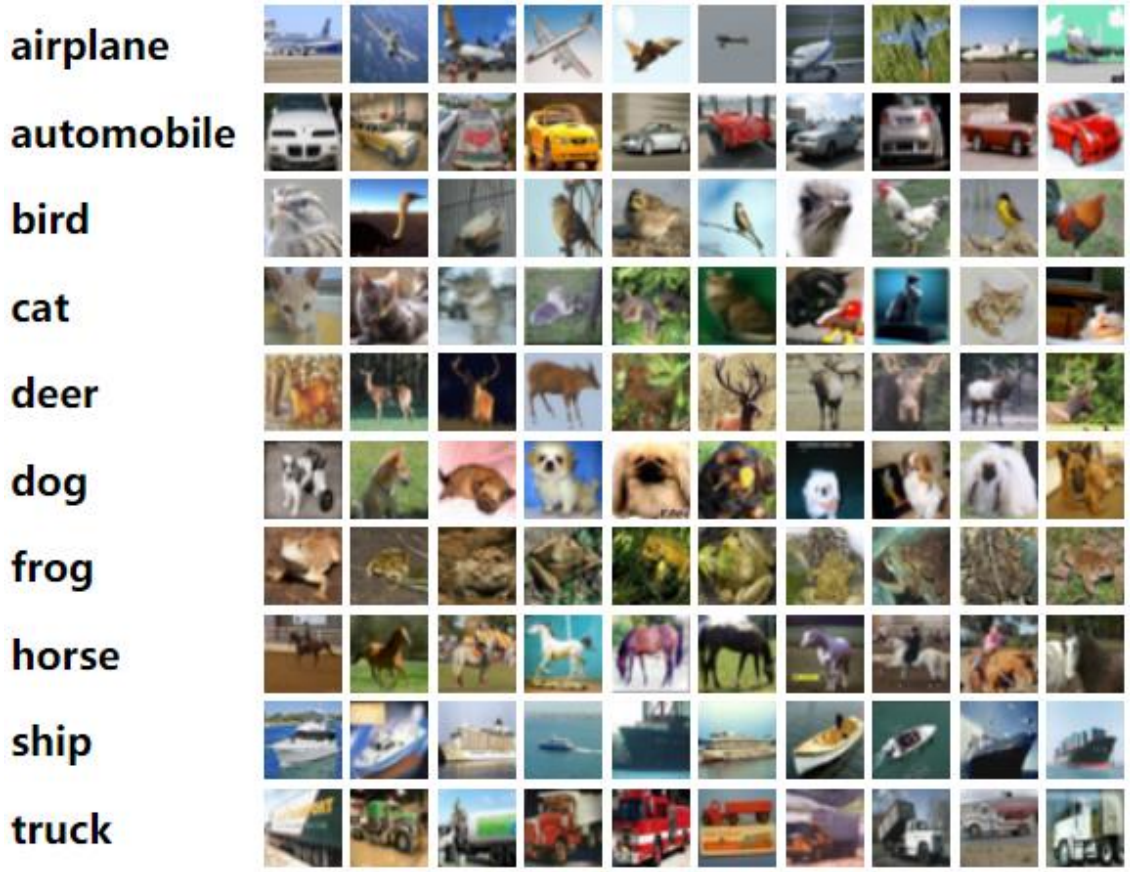
hello-pt-resnet

- cd ./NVFlare/examples/hello-world/hello-pt-resnet
- pip3 install -r requirements.txt
- pip3 install tensorboard
- python3 fedavg_script_runner_pt.py



CIFAR10

- 60000 images
  - 32x32 colour images in 10 classes
  - 6000 images per class
  - 50000 training images
  - 10000 test images



https://www.cs.toronto.edu/~kriz/cifar.html

Server submit job

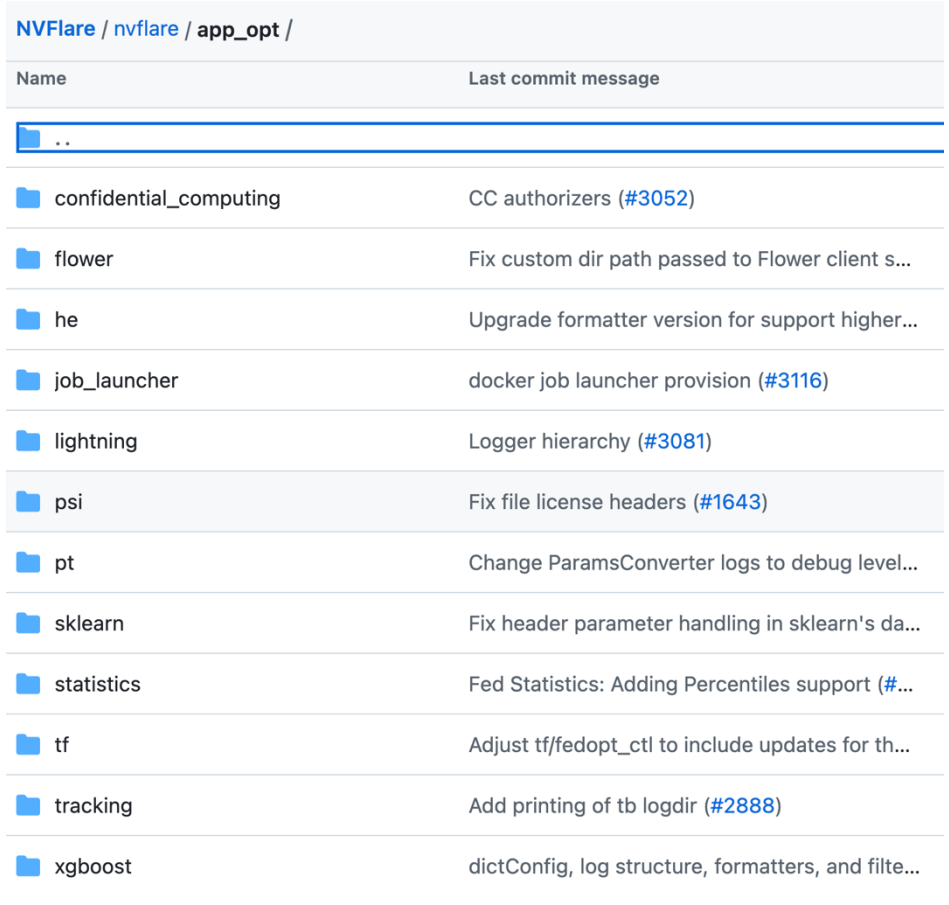
fedavg_script_runner_pt.py

GPU usage

- GPU is available
  - job.simulator_run("/tmp/nvflare/jobs/workdir", gpu="0")
- GPU is not available
  - job.simulator_run("/tmp/nvflare/jobs/workdir")

FedAvgJob

- from nvflare.app_opt.pt.job_config.fed_avg import FedAvgJob



NVFlare / nvflare / app_opt /

Name	Last commit message
..	
confidential_computing	CC authorizers (#3052)
flower	Fix custom dir path passed to Flower client s...
he	Upgrade formatter version for support higher...
job_launcher	docker job launcher provision (#3116)
lightning	Logger hierarchy (#3081)
psi	Fix file license headers (#1643)
pt	Change ParamsConverter logs to debug level...
sklearn	Fix header parameter handling in sklearn's da...
statistics	Fed Statistics: Adding Percentiles support (#...
tf	Adjust tf/fedopt_ctl to include updates for th...
tracking	Add printing of tb logdir (#2888)
xgboost	dictConfig, log structure, formatters, and filte...

```python
from typing import List, Optional

import torch.nn as nn

from nvflare.app_common.workflows.fedavg import FedAvg
from nvflare.app_opt.pt.job_config.base_fed_job import BaseFedJob
```

and…and…and…and…

```python
def get_result(self):
    """Divide weighted sum by sum of weights."""
    with self.lock:
        aggregated_dict = {k: v * (1.0 / self.counts[k]) for k, v in self.total.items()}
        self.reset_stats()
        return aggregated_dict
```

https://github.com/NVIDIA/NVFlare/blob/main/nvflare/app_common/aggregators/weighted_aggregation_helper.py#L20

ScriptRunner

- from nvflare.job_config.script_runner import ScriptRunner
- default for pytorch
- script_args for train_script

```
executor = ScriptRunner(
    script=train_script,
    script_args="",  # f"--batch_size 32 --data_path /tmp/data/site-{i}"
    framework=FrameworkType.TENSORFLOW,
)
```

fedavg_script_runner_pt.py

```python
from src.resnet_18 import Resnet18

from nvflare.app_opt.pt.job_config.fed_avg import FedAvgJob
from nvflare.job_config.script_runner import ScriptRunner

if __name__ == "__main__":
    n_clients = 2
    num_rounds = 2
    train_script = "src/hello-pt_cifar10_fl.py"

    job = FedAvgJob(
        name="hello-pt_cifar10_fedavg",
        n_clients=n_clients,
        num_rounds=num_rounds,
        initial_model=Resnet18(num_classes=10),
    )

    # Add clients
    for i in range(n_clients):
        executor = ScriptRunner(
            script=train_script,
            script_args="",  # f"--batch_size 32 --data_path /tmp/data/site-{i}"
        )
        job.to(executor, f"site-{i + 1}")

    # job.export_job("/tmp/nvflare/jobs/job_config")
    job.simulator_run("/tmp/nvflare/jobs/workdir", gpu="0")
```

Set clients and model weight

Send script to clients

Run Job
Modify /tmp/YOURNAME/nvflare/...

Client training

src/hello-pt_cifar10_fl.py

hello-pt_cifar10_fl.py

- import nvflare.client as flare
- from nvflare.client.tracking import SummaryWriter
- flare.init() # Establish a connection with the FL server
- flare.system_info() # Retrieve client system information
- while flare.is_running(): # Start execution loop
- flare.receive() # Receive the global model from the server
- flare.FLModel() # Set up the process to handle the model
- flare.send() # Send the updated model back to the server

```
27
28      DATASET_PATH = "/tmp/nvflare/data"
29
```

/tmp/YOURNAME/nvflare/data

```
model = Resnet18(num_classes=10)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
loss = nn.CrossEntropyLoss()
optimizer = SGD(model.parameters(), lr=lr, momentum=0.9)
transforms = Compose(
    [
        ToTensor(),
        Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    ]
)
```

```
train_dataset = CIFAR10(
    root=os.path.join(DATASET_PATH, client_name),
    transform=transforms,
    download=True,
    train=True,
)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
```

Where change?

- Adapting the example from use the MNIST of Pytorch tutorial
- Modify the code above ：
  - Model structure
    - src/resnet_18.py
  - Load model
    - fedavg_script_runner_pt.py
  - transforms
    - fedavg_script_runner_pt.py
    - src/hello-pt_cifar10_fl.py
  - Load data
    - src/hello-pt_cifar10_fl.py

https://github.com/pytorch/examples/blob/main/mnist/main.py

Model structure

```python
class Resnet18(ResNet):
    def __init__(self, num_classes, weights: Optional[ResNet18_Weights] = None, progress:
        self.num_classes = num_classes

        weights = ResNet18_Weights.verify(weights)

        if weights is not None:
            _ovewrite_named_param(kwargs, "num_classes", len(weights.meta["categories"]))

        super().__init__(BasicBlock, [2, 2, 2, 2], num_classes=num_classes, **kwargs)

        if weights is not None:
            super().load_state_dict(weights.get_state_dict(progress=progress))
```

Source

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

mnist

Load model

```
initial_model=Resnet18(num_classes=10),
```

Source

```
initial_model=Resnet18(),
```

Complete

transforms

```
transforms = Compose(
    [
        ToTensor(),
        Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    ]
)
```

Source

```
transforms = Compose(
    [
        ToTensor(),
        Normalize((0.5), (0.5)),
    ]
)
```

Complete

Load data

```
train_dataset = CIFAR10(
    root=os.path.join(DATASET_PATH, client_name),
    transform=transforms,
    download=True,
    train=True,
)
```

Source

```
train_dataset = MNIST(
    root=os.path.join(DATASET_PATH, client_name),
    transform=transforms,
    download=True,
    train=True,
)
```

Complete

NVFlare operational mode

- Simulator and POC mode for rapid development and prototyping
  - simulator
  - poc
- FLARE API

nvflare provision -p ./secure_project.yml

project file for creating the workspace

```yaml
api_version: 3
name: secure_project
description: NVIDIA FLARE sample project yaml file for CIFAR-10 example

participants:
  - name: overseer.example.com
    type: overseer
    org: nvidia
    protocol: https
    api_root: /api/v1
    port: 8443
  - name: localhost
    type: server
    org: nvidia
    fed_learn_port: 8102
    admin_port: 8103
  - name: site-1
    type: client
    org: nvidia
```

https://github.com/NVIDIA/NVFlare/blob/main/examples/advanced/cifar10/cifar10-real-world/workspaces/secure_project.yml

```
├── admin@nvidia.com
│   ├── local
│   └── startup
│       ├── client.crt
│       ├── client.key
│       ├── fed_admin.json
│       ├── fl_admin.sh
│       ├── readme.txt
│       └── rootCA.pem
│   └── transfer
├── overseer.example.com
│   ├── local
│   └── startup
│       ├── gunicorn.conf.py
│       ├── overseer.crt
│       ├── overseer.key
│       ├── privilege.yml
│       ├── rootCA.pem
│       ├── signature.json
│       └── start.sh
│   └── transfer
├── site-1
│   ├── audit.log
│   ├── local
│   │   ├── authorization.json.default
│   │   ├── log.config.default
│   │   ├── privacy.json.sample
│   │   ├── resources.json
│   │   └── resources.json.default
│   ├── log.txt
│   ├── pid.fl
│   ├── readme.txt
│   ├── startup
│   │   ├── client.crt
│   │   ├── client.key
│   │   ├── client_context.tenseal
│   │   ├── fed_client.json
│   │   ├── rootCA.pem
│   │   ├── signature.json
│   │   ├── start.sh
│   │   ├── stop_fl.sh
│   │   └── sub_start.sh
│   └── transfer
```

config_fed_client.json

```
{
  "format_version": 2,                  // Specifies the config version (must be 2)
  "TRAIN_SPLIT_ROOT": "/tmp/cifar10_splits",    // Path to local training index files
  "AGGREGATION_EPOCHS": 4,                  // Number of local epochs per round
  "executors": [],                      // How to handle tasks from the server
  "task_result_filters": [],              // No post-processing filters applied
  "task_data_filters": [],                // No input filters applied
  "components": []                      // All modules the client uses
}
```

config_fed_server.json

```
{
  "format_version": 2,
  // Global variables used in args below
  "TRAIN_SPLIT_ROOT": "/path/to/split_dir",
  "AGGREGATION_EPOCHS": 4,
  "executors": [ ], // Task handlers (e.g. train/validate) mapped to executors
  "task_result_filters": [], // Optional filters for task results (e.g. compression, encryption)
  "task_data_filters": [], // Optional filters for incoming task data
  "components": [] // Modular components like learners, loggers, metrics, etc.
}
```

config

```
"executor": {
    "id": "Executor",
    "path": "nvflare.app_common.executors.model_learner_executor.ModelLearnerExecutor",
    "args": {
      | "learner_id": "cifar10-learner"
    }
```

```
    learner_id,
    train_task=AppConstants.TASK_TRAIN,
    submit_model_task=AppConstants.TASK_SUBMIT_MODEL,
    validate_task=AppConstants.TASK_VALIDATION,
    configure_task=AppConstants.TASK_CONFIGURE,
):
    """Key component to run learner on clients.

    Args:
        learner_id (str): id of the learner object
        train_task (str, optional): task name for train. Defaults to AppConstants.TASK_TRAIN.
        submit_model_task (str, optional): task name for submit model. Defaults to AppConstants.TASK_SUBMIT_MODEL.
        validate_task (str, optional): task name for validation. Defaults to AppConstants.TASK_VALIDATION.
        configure_task (str, optional): task name for configure. Defaults to AppConstants.TASK_CONFIGURE.
    """
    super().__init__()
    self.learner_id = learner_id
    self.learner = None
    self.learner_name = ""
    self.is_initialized = False
    self.learner_exe_lock = threading.Lock()  # used ensure only one execution at a time
```

learner_id is request

https://github.com/NVIDIA/NVFlare/blob/main/nvflare/app_common/executors/model_learner_executor.py

```
class CIFAR10ModelLearner(ModelLearner):  # A custom federated learning trainer for CIFAR-10

  def __init__(self, ...):
    # Initializes hyperparameters, dataset paths, training configs, and placeholders.

  def initialize(self):
    # Sets up model, optimizer, loss functions, device (CPU/GPU),
    # and analytics writer (e.g., TensorBoard or AnalyticsSender).

  def _create_datasets(self):
    # Loads the training and validation datasets based on site-specific indices
    # (used in federated settings) or uses full dataset if in central mode.

  def finalize(self):
    # (Optional) Final cleanup after training is complete (currently empty).

  def local_train(self, train_loader, model_global, val_freq=0):
    # Performs local training for a given number of epochs using SGD optimizer.
    # Optionally runs validation during training. Supports FedProx loss.
```

```
def save_model(self, is_best=False):
    # Saves the local model to disk.
    # If `is_best=True`, stores it as the best-performing model so far.

def train(self, model: FLModel) -> Union[str, FLModel]:
    # Main method triggered by the "train" task.
    # Loads the received global model, performs local training,
    # and returns the model delta (differences).

def get_model(self, model_name: str) -> Union[str, FLModel]:
    # Triggered by the "submit_model" task.
    # Loads and returns the best local model stored during training.

def local_valid(self, valid_loader, tb_id=None):
    # Performs evaluation on the validation dataset.
    # Computes accuracy and optionally logs metrics using the writer.

def validate(self, model: FLModel) -> Union[str, FLModel]:
    # Triggered by the "validate" task.
    # Evaluates the received global model on local training and validation sets.
    # Returns metrics (e.g., accuracy).
```

Config to code

Task in Server	executor code (https://github.com/NVIDIA/NVFlare/blob/main/nvflare/app_common/executors/model_learner_executor.py)	Own code
"train"	self.train_task（default "train"）	train()
"submit_model"	self.submit_model_task（default "submit_model"）	get_model()
"validate"	self.validate_task（default "validate"）	validate()

Connect to server

- git clone https://github.com/holiday01/nvflare_2.5.2_workspace
- cd ./nvflare_2.5.2_workspace/cifar10-real-world
- site-1~site8
  - export PYTHONPATH=${PWD}
  - workspaces/secure_workspace/site-1/startup/start.sh