Funktion dokumentation

Certificates:

P2pfl/certificates

Mutual TLS for gRPC so far just for testing not for actual use

* Client\_ext (client def)
* Gen-certs (generate cert)
* Openssl (define cert)
* Server\_ext (server def)

Communication:

Defined set of commands and protocols.

P2pfl/communication/commands

Include message commands, weight commands.

* Message/heartbeat\_command (checks on nodes state and detects failures)
* Message/metrics\_command (sends evaluation metrics)
* Message/model\_initialized\_command (signals model initialization on a node)
* Message/models\_agregated\_command (informs neighbors that the aggregation of the model is finished)
* Message/models\_ready\_command (signals that the aggregation is complete and the readiness for the next stage)
* Message/start\_learning\_command (initiates the learning process across the network)
* Message/stop\_learning\_comman (Terminates the learning process across the network)
* Message/vote\_train\_set\_command (manages voting for training set selection)
* Weights/full\_model\_command (communicates the complete aggregated model)
* Weights/init\_model\_command (Distibutes initial model weights)
* Weights/partial\_model\_command (communicates a partial model update used during aggregation)

P2pfl/communication/protocols

Include grpc protocol, memory protocol. Defines interaction of all components.

P2pfl/communication/protocols

* Grpc/proto/generate\_proto
* Grpc/proto/node.proto
* Grpc/proto/node\_pb2
* Grpc/proto\_pb2\_grpc

(specific Example protocol generated ?)

* Grpc/address (address parser (Determines if the address is a Unix domain address or an IP address (IPv4 or IPv6). handles different address types)
* Grpc/grpc\_client (Implementation of the client side (i.e. who initiates the communication) of the GRPC communication protocol. (build\_message, build\_weights, send, broadcast)
* Grpc/grpc\_communication\_protocol (define and manage GRPC communication protocol. (get\_address, start, stop, add\_command, connect, disconnect, build\_msg, build\_weights, send, broadcast, get\_neighbors, wait\_for\_termination, gossip\_weights) handling grpc protocol)
* Grpc/grpc\_neighbors (define grpc\_neighbors (refresh\_or\_add, connect, build\_direct\_neighbor, build\_non\_direct\_neighbor, disconnect))
* Grpc/grpc\_server (Implementation of the server side of a GRPC communication protocol. (start, stop, wait\_for\_termination, is\_running, handshake, disconnect, send, add\_command))
* Memory/memory\_client (Implementation of the client side of an in-memory communication protocol. (build\_message, build\_weights, send, broadcast))
* Memory/memory\_comunication\_protocol (defines and manages the In-memory communication protocol. (get\_address, start, stop, add\_command, connect, disconnect, build\_msg, build\_weights, send, broadcast, get\_neighbors, wait\_for\_termination, gossip\_weights)
* Memory/memory\_neighbors (Implementation of the neighbors side of an in-memory communication protocol. (refresh\_or\_add, connect, build\_direct\_neighbor, build\_non\_direct\_neighbor, disconnect))
* Memory/memory\_server (Implementation of the server side of an in-memory communication protocol. (start, stop, wait\_for\_termination, is\_running, handshake, disconnect, send\_message, send\_weights, add\_command))
* Memory/server\_singleton (Server singleton class. (reset\_instance, wait\_for\_termination))
* Client (Protocol agnostic client. Client interface. It is used as a interface to help to decoulple communication protocols based on clien-server architecutes. (build\_message, build\_weights, send, broadcast))
* Communication\_protocol (Communication protocol interface. (start, stop, add\_command, build\_msg, build\_weights, send, broadcast, connect, disconnect, get\_neighbors, get\_address, wait\_for\_termination, gossip\_weights))
* Exceptions (Communication exceptions.)
* Gossiper (Protocol agnostic gossiper. Gossiper for agnostic communication protocol. (start, stop, add\_message, check\_and\_set\_processed, run, gossip\_weights))
* Heartbeater (Protocol agnostic heartbeater. Heartbeater for agnostic communication protocol. Send and update fresh heartbeats. (run, stop, beat, heartbeater))
* Neighbors (Protocol agnostic neighbor management. Neighbor management class for agnostic communication protocol. (connect, disconnect, refresh\_or\_add, add, remove, get, get\_all, exists, clear\_neighbors))

Examples:

Preconstructed examples to try and learn from.

P2pfl/examples

* Mnist (defines and creats a set amount of nodes and starts the learning process)
* Node1 (defines and creates a single node)
* Node2 (defines and creates a second node and connects it to a first node then starts the learning process)

Learning:

Includes the components relevant for learning with the network. The included components are part of three categories (aggregator, dataset, frameworks).

P2pfl/learning

* Aggregator/aggregator (abstract aggregator for defining aggregators which are crucial for the aggregation process)
* Aggregator/fedavg (Federated Averaging combines updates using a weighted average based on sample size. Supports partial aggregation)
* Aggregator/fedmedian (Computes the median of updates for robustness against outliers or adversarial contributions. Supports partial aggregation)
* Aggregator/scaffold (Uses control variates to reduce variance and correct client drift in non-IID data scenarios. No support for partial aggregation)
* Dataset/p2pfl\_dataset (provides a consistent API for working with datasets, regardless of their original format or source. Loads data from various sources, including CSV, JSON, and Parquet files, Pandas DataFrames, Python dictionaries and lists, and the Hugging Face Hub. Easily partition your data for federated learning using built-in strategies. Export your data in formats readily usable by popular machine learning frameworks like PyTorch, TensorFlow, and Flax. Apply custom transformations to your data before training.)
* Dataset/partition\_strategies (RandomIIDPartitionStrategy (Distributes data randomly across clients, creating an **Independent and Identically Distributed (IID) scenario**.), DirichletPartitionStrategy (Non-IID) (Distributes data based on a Dirichlet distribution, controlled by the alpha parameter, creating a **non-IID scenario**.)
* Framework/flax/flax\_dataset (Flax Dataset export strategy.)
* Framework/flax/flax\_learner (Flax Learner for P2PFL. (flax\_model, get\_flax\_data, calculate\_loss\_acc, train\_step, fit, evaluate, interrupt\_fit, get\_framework))
* Framework/flax/flax\_model (Flax Model for P2PFL. (dict\_to\_np, np\_to\_dict, get\_parameters, set\_parameters, encode\_parameters, decode\_parameters, build\_copy, get\_framework, MLP))
* Framework/pytorch/callbacks/scaffold\_callback (Callback for SCAFFOLD operations (PyTorch Lighting). (get\_name, on\_train\_start, on\_train\_batch\_start, on\_before\_zero\_grad, on\_train\_end, get\_parameters, set\_parameters))
* Framework/pytorch/utils/torchvision\_to\_dataset (Example script that allows you to upload a PyTorch Vision dataset to the Hugging Face Hub. The idea is to directly use parquet and datasets to reduce the overhead of creating a TorchVision dataset and then transforming it to a Hugging Face dataset. (create\_huggingface\_dataset\_from\_torchvision, generate\_examples, get\_args))
* Framework/pytorch/lightning\_dataset (PyTorch dataset integration. (get\_mnist, get\_generator, export))
* Framework/pytorch/lightning\_learner (Lightning Learner for P2PFL." (get\_pt-model\_data, fit, interrupt\_fit, evaluate, get\_framework))
* Framework/pytorch/lightning\_logger (Lightning Logger for P2PFL. Pytorch Lightning Logger for Federated Learning. Handles local training loggin. (name, version, log\_hyperparams, log\_metrics, save, finalize))
* Framework/pytorch/lightning\_model (Convolutional Neural Network (for MNIST) with PyTorch Lightning. P2PFL model abstraction for PyTorch Lightning. (get\_parameters, set\_parameters. get\_framework, MLP, get\_activation, forward, configure\_optimizers, training\_step, validation\_step, test\_step))
* Framework/simulation/actor\_pool (Actor pool for distributed computing using Ray. (terminate, fit, evaluate, superactualpool, new, reduce, create\_actor, add\_actor, submit, submit\_learner\_job, flag\_future\_as\_ready, reset\_addr\_to\_future\_dict, is\_future\_ready, fetch\_future\_result, flag\_actor\_for\_removal, check\_and-remove\_actor\_from\_pool, check\_actor\_fits\_in\_pool, process\_unordered\_future, get\_learner\_result))
* Framework/simulation/utils (Utility functions for the simulation. (check\_client\_resources, pool\_size\_from\_resources))
* Framework/simulation/virtual\_learner (Virtual Node Learner. Decorator for the learner to be used in the simulation. (set\_addr, set\_model, get\_model, set\_data, get\_data, set\_epochs, fit, interrupt\_fit, evaluate, get\_framework))
* Framework/tensorflow/callbacks/keras\_logger (Keras Logger for P2PFL. Keras Logger for Federated Learning. Handles local training logging. (get\_name, on\_epoch\_end, on\_train\_batch\_send))
* Framework/tensorflow/callbacks/scaffold\_callback (Callback for SCAFFOLD operations (Keras). Wraps an optimizer to only redefine apply\_gradients, delegating other calls. (apply\_gradients, getattr, SCAFFOLDCallback, get\_name, on\_train\_begin, on\_train\_batch\_end, on\_train\_end, set\_additional\_info))
* Framework/tensorflow/keras\_dataset (Export strategy for TensorFlow/Keras datasets. (export))
* Framework/tensorflow/keras\_learner (Learner for TensorFlow/Keras models in P2PFL. (get\_tf-model\_data, fit, interrupt\_fit, evaluate, get\_framework))
* Framework/tensorflow/keras\_model (Keras model abstraction for P2PFL. (get\_parameters, set\_parameters, get\_framework, MLP, call, get\_framework))
* Framework/callback (P2PFL Callbacks for federated learning. A callback for the P2PFL learning process. The callback can generate additional information that can be used by the aggregator to affect the learning process. In order to affect the learning process, the callback must be registered with the `CallbackFactory`. (get\_name, get\_info, set\_info))
* Framework/callback\_factory (P2PFLCallback factory. Factory for creating callbacks based on learner framework and aggregator requirements. (register\_callback, create\_callbacks))
* Framework/exceptions (Learning Exceptions.)
* Framework/learner (NodeLearning Interface - Template Pattern. Template to implement learning processes, including metric monitoring during training. (set\_addr, set\_model, get\_model, set\_data, get\_data, set\_epochs, update\_callbacks\_with\_model\_info, fit, interrupt\_fit, evaluate, get\_framework))
* Framework/learner\_factory (P2PFLCallback factory. Factory for creating learners based on the model framework. (create\_learner))
* Framework/p2pfl\_model (P2PFL model abstraction. This class encapsulates the different models across all the possible frameworks. The key concept is the extraction of the model weights in a common format for all the frameworks. (get\_model, encode\_parameters, decode\_parameters, get\_parameters, set\_parameters, add\_info, get\_info, set\_contribution, get\_contributors, get\_num\_samples, build\_copy, get\_framework))

Management:

Includes the logger component that manages the learning process and manages the interaction of the components and other management functions relevant for the process.

P2pfl/Management/

* logger/decorators/async\_logger (A decorator that wraps the logger to support asynchronous logging. This ensures that logging does not block the main processes in high-performance, distributed environments, improving the overall efficiency of the system.)
* logger/decorators/file\_logger (A decorator that wraps a logger class, automatically writing log entries to a specified file. This decorator enhances the base logger by directing the log output to a file in addition to the standard output.)
* logger/decorators/logger\_decorator (decorator for logger to be used in simulation. Decorator class for logging. Works by wrapping the a previous logger, so new funcionalities can be added. By default, it does nothing, just delegates the calls to the wrapped logger.)
* logger/decorators/ray\_logger (A decorator that wraps a logger class and sends logs to a Ray cluster. This is particularly useful in federated learning setups using Ray, where logs need to be captured at both the node and system level in a distributed environment.)
* logger/decorators/singleton\_logger (logger for singleton)
* logger/decorators/web\_logger (A decorator that wraps a logger class and sends log entries to P2PFL web services. This is useful for centralized logging when working with distributed systems, allowing logs to be monitored remotely.)
* logger/logger (defines the logger and logger level and functions to use for the logger)
* metric\_storage (manage/get local and global logs)
* node\_monitor (run and stop the node\_monitor (stop, run, report\_system\_resources, report\_status))
* p2pfl\_web\_services (manages communication with webservice (build\_header, register\_node, unregister\_node, send\_log, send\_local\_metric, send\_global\_metric, send\_system\_metric, get\_pending\_actions))

Stages:

Includes functionality to define stages in and the workflow of the learning process.

P2pfl/stages/

* base\_node/gossip\_model\_stage
* base\_node/round\_finished\_stage
* base\_node/start\_learning\_stage
* base\_node/train\_stage
* base\_node/vote\_train\_set\_stage
* Base\_node/wait\_agg\_models\_stage

Defines the stages and there interaction in a learning process

A(StartLearningStage) --> B(VoteTrainSetStage)

B -- Node in trainset? --> C(TrainStage)

B -- Node not in trainset? --> D(WaitAggregatedModelsStage)

C --> E(GossipModelStage)

D --> E

E --> F(RoundFinishedStage)

F -- No more rounds? --> Finished

F -- More rounds? --> B

* Stage (abstract class for stages)
* Stage\_factory (create stages for a workflow)
* Workflows (defines and manages the workflow)

Utils:

Includes additional functionality created to make things easier.

P2pfl/utils

* Check\_ray (checks whether ray is installed and initializes it if not initialized)
* Singleton (call singleton)
* Topologies (create a set of defined network topologies (star, full, line, ring))
* Utils (included utility functions wait\_convergence (wait till all nodes have n neighbors), full\_conection, wait\_to\_finish (wait till all nodes have finished the workflow), check\_equal\_models (check that all nodes have the same model)

Other:

* cli (define functionality of the cli)
* exceptions (define possible exceptions)
* experiment (define and execute experiments)
* node (defines the functionality (commands) and model etc. of a node)
* node\_state (store and set node state)
* settings (stores and set hyperparameter for learning process and network)