PySyft

Release 0.2.3a1

Andrew Trask

Feb 06, 2020

CONTENTS:

1 API Reference **1**

1.1 syft . . . 1

1.1.1 Subpackages . . . 1

1.1.1.1 syft.federated . . . 1

1.1.1.1.1 Submodules . . . 1

1.1.1.1.1.1 syft.federated.federated_client . . . 1

1.1.1.1.1.2 Module Contents . . . 1

1.1.1.1.1.3 syft.federated.train_config . . . 2

1.1.1.1.1.4 Module Contents . . . 2

1.1.1.2 syft.frameworks . . . 4

1.1.1.2.1 Subpackages . . . 4

1.1.1.2.1.1 syft.frameworks.keras . . . 4

1.1.1.2.1.2 Subpackages . . . 4

1.1.1.2.1.3 syft.frameworks.keras.layers . . . 4

1.1.1.2.1.4 Submodules . . . 4

1.1.1.2.1.5 syft.frameworks.keras.layers.constructor . . . 4

1.1.1.2.1.6 Module Contents . . . 4

1.1.1.2.1.7 Package Contents . . . 4

1.1.1.2.1.8 syft.frameworks.keras.model . . . 4

1.1.1.2.1.9 Submodules . . . 4

1.1.1.2.1.10 syft.frameworks.keras.model.sequential . . . 4

1.1.1.2.1.11 Module Contents . . . 4

1.1.1.2.1.12 Package Contents . . . 5

1.1.1.2.1.13 Submodules . . . 5

1.1.1.2.1.14 syft.frameworks.keras.hook . . . 5

1.1.1.2.1.15 Module Contents . . . 5

1.1.1.2.1.16 syft.frameworks.tensorflow . . . 6

1.1.1.2.1.17 syft.frameworks.torch . . . 6

1.1.1.2.1.18 Subpackages . . . 6

1.1.1.2.1.19 syft.frameworks.torch.dp . . . 6

1.1.1.2.1.20 Submodules . . . 6

1.1.1.2.1.21 syft.frameworks.torch.dp.pate . . . 6

1.1.1.2.1.22 Module Contents . . . 6

1.1.1.2.1.23 syft.frameworks.torch.fl . . . 9

1.1.1.2.1.24 Submodules . . . 9

1.1.1.2.1.25 syft.frameworks.torch.fl.dataloader . . . 9

1.1.1.2.1.26 Module Contents . . . 9

1.1.1.2.1.27 syft.frameworks.torch.fl.dataset . . . 10

1.1.1.2.1.28 Module Contents . . . 10

1.1.1.2.1.29 syft.frameworks.torch.fl.utils . . . 11

1.1.1.2.1.30 Module Contents . . . 11
1.1.1.2.1.31 Package Contents . . . 13
1.1.1.2.1.32 syft.frameworks.torch.he . . . 14
1.1.1.2.1.33 Submodules . . . 14
1.1.1.2.1.34 syft.frameworks.torch.he.paillier . . . 14
1.1.1.2.1.35 Module Contents . . . 14
1.1.1.2.1.36 syft.frameworks.torch.hook . . . 15
1.1.1.2.1.37 Submodules . . . 15
1.1.1.2.1.38 syft.frameworks.torch.hook.hook . . . 15
1.1.1.2.1.39 Module Contents . . . 15
1.1.1.2.1.40 syft.frameworks.torch.hook.hook_args . . . 17
1.1.1.2.1.41 Module Contents . . . 17
1.1.1.2.1.42 syft.frameworks.torch.linalg . . . 18
1.1.1.2.1.43 Submodules . . . 18
1.1.1.2.1.44 syft.frameworks.torch.linalg.lr . . . 18
1.1.1.2.1.45 Module Contents . . . 18
1.1.1.2.1.46 syft.frameworks.torch.linalg.operations . . . 20
1.1.1.2.1.47 Module Contents . . . 20
1.1.1.2.1.48 Package Contents . . . 22
1.1.1.2.1.49 syft.frameworks.torch.mpc . . . 25
1.1.1.2.1.50 Submodules . . . 25
1.1.1.2.1.51 syft.frameworks.torch.mpc.beaver . . . 25
1.1.1.2.1.52 Module Contents . . . 25
1.1.1.2.1.53 syft.frameworks.torch.mpc.securenn . . . 25
1.1.1.2.1.54 Module Contents . . . 25
1.1.1.2.1.55 syft.frameworks.torch.mpc.spdz . . . 27
1.1.1.2.1.56 Module Contents . . . 27
1.1.1.2.1.57 syft.frameworks.torch.nn . . . 28
1.1.1.2.1.58 Submodules . . . 28
1.1.1.2.1.59 syft.frameworks.torch.nn.conv . . . 28
1.1.1.2.1.60 Module Contents . . . 28
1.1.1.2.1.61 syft.frameworks.torch.nn.pool . . . 28
1.1.1.2.1.62 Module Contents . . . 28
1.1.1.2.1.63 syft.frameworks.torch.nn.rnn . . . 29
1.1.1.2.1.64 Module Contents . . . 29
1.1.1.2.1.65 Package Contents . . . 30
1.1.1.2.1.66 syft.frameworks.torch.tensors . . . 32
1.1.1.2.1.67 Subpackages . . . 32
1.1.1.2.1.68 syft.frameworks.torch.tensors.decorators . . . 32
1.1.1.2.1.69 Submodules . . . 32
1.1.1.2.1.70 syft.frameworks.torch.tensors.decorators.logging . . . 32
1.1.1.2.1.71 Module Contents . . . 32
1.1.1.2.1.72 syft.frameworks.torch.tensors.interpreters . . . 33
1.1.1.2.1.73 Submodules . . . 33
1.1.1.2.1.74 syft.frameworks.torch.tensors.interpreters.additive_shared . . . 33
1.1.1.2.1.75 Module Contents . . . 33
1.1.1.2.1.76 syft.frameworks.torch.tensors.interpreters.autograd . . . 38
1.1.1.2.1.77 Module Contents . . . 38
1.1.1.2.1.78 syft.frameworks.torch.tensors.interpreters.build_gradients . . . 40
1.1.1.2.1.79 Module Contents . . . 40

1.1.1.2.1.80 syft.frameworks.torch.tensors.interpreters.crt_precision . . . 41
1.1.1.2.1.81 Module Contents . . . 41
1.1.1.2.1.82 syft.frameworks.torch.tensors.interpreters.gradients . . . 43
1.1.1.2.1.83 Module Contents . . . 43
1.1.1.2.1.84 syft.frameworks.torch.tensors.interpreters.gradients_core . . . 44
1.1.1.2.1.85 Module Contents . . . 44
1.1.1.2.1.86 syft.frameworks.torch.tensors.interpreters.hook . . . 45
1.1.1.2.1.87 Module Contents . . . 45
1.1.1.2.1.88 syft.frameworks.torch.tensors.interpreters.large_precision . . . 46
1.1.1.2.1.89 Module Contents . . . 46
1.1.1.2.1.90 syft.frameworks.torch.tensors.interpreters.native . . . 48
1.1.1.2.1.91 Module Contents . . . 48
1.1.1.2.1.92 syft.frameworks.torch.tensors.interpreters.numpy . . . 52
1.1.1.2.1.93 Module Contents . . . 52
1.1.1.2.1.94 syft.frameworks.torch.tensors.interpreters.paillier . . . 53
1.1.1.2.1.95 Module Contents . . . 53
1.1.1.2.1.96 syft.frameworks.torch.tensors.interpreters.placeholder . . . 54
1.1.1.2.1.97 Module Contents . . . 54
1.1.1.2.1.98 syft.frameworks.torch.tensors.interpreters.plusisminus . . . 56
1.1.1.2.1.99 syft.frameworks.torch.tensors.interpreters.polynomial . . . 56
1.1.1.2.1.100 syft.frameworks.torch.tensors.interpreters.precision . . . 56
1.1.1.2.1.101 Module Contents . . . 56
1.1.1.2.1.102 syft.frameworks.torch.tensors.interpreters.private . . . 60
1.1.1.2.1.103 Module Contents . . . 60
1.1.1.2.1.104 Submodules . . . 61
1.1.1.2.1.105 syft.frameworks.torch.functions . . . 61
1.1.1.2.1.106 Module Contents . . . 61
1.1.1.2.1.107 syft.frameworks.torch.torch_attributes . . . 61
1.1.1.2.1.108 Module Contents . . . 61
1.1.1.3 syft.generic . . . 62
1.1.1.3.1 Subpackages . . . 62
1.1.1.3.1.1 syft.generic.frameworks . . . 62
1.1.1.3.1.2 Subpackages . . . 62
1.1.1.3.1.3 syft.generic.frameworks.hook . . . 62
1.1.1.3.1.4 Submodules . . . 62
1.1.1.3.1.5 syft.generic.frameworks.hook.hook . . . 62
1.1.1.3.1.6 Module Contents . . . 62
1.1.1.3.1.7 syft.generic.frameworks.hook.hook_args . . . 65
1.1.1.3.1.8 Module Contents . . . 65
1.1.1.3.1.9 syft.generic.frameworks.hook.trace . . . 70
1.1.1.3.1.10 Module Contents . . . 70

1.1.1.3.1.11 Submodules . . . 71
1.1.1.3.1.12 syft.generic.frameworks.attributes . . . 71
1.1.1.3.1.13 Module Contents . . . 71
1.1.1.3.1.14 syft.generic.frameworks.overload . . . 72
1.1.1.3.1.15 Module Contents . . . 72
1.1.1.3.1.16 syft.generic.frameworks.remote . . . 72
1.1.1.3.1.17 Module Contents . . . 72
1.1.1.3.1.18 syft.generic.frameworks.types . . . 72
1.1.1.3.1.19 Module Contents . . . 72
1.1.1.3.1.20 syft.generic.pointers . . . 73
1.1.1.3.1.21 Submodules . . . 73
1.1.1.3.1.22 syft.generic.pointers.callable_pointer . . . 73
1.1.1.3.1.23 Module Contents . . . 73
1.1.1.3.1.24 syft.generic.pointers.multi_pointer . . . 74
1.1.1.3.1.25 Module Contents . . . 74
1.1.1.3.1.26 syft.generic.pointers.object_pointer . . . 76
1.1.1.3.1.27 Module Contents . . . 76
1.1.1.3.1.28 syft.generic.pointers.object_wrapper . . . 79
1.1.1.3.1.29 Module Contents . . . 79
1.1.1.3.1.30 syft.generic.pointers.pointer_plan . . . 80
1.1.1.3.1.31 Module Contents . . . 80
1.1.1.3.1.32 syft.generic.pointers.pointer_protocol . . . 81
1.1.1.3.1.33 Module Contents . . . 81
1.1.1.3.1.34 syft.generic.pointers.pointer_tensor . . . 82
1.1.1.3.1.35 Module Contents . . . 82
1.1.1.3.1.36 syft.generic.pointers.string_pointer . . . 86
1.1.1.3.1.37 Module Contents . . . 86
1.1.1.3.2 Submodules . . . 86
1.1.1.3.2.1 syft.generic.id_provider . . . 86
1.1.1.3.2.2 Module Contents . . . 86
1.1.1.3.2.3 syft.generic.metrics . . . 87
1.1.1.3.2.4 Module Contents . . . 87
1.1.1.3.2.5 syft.generic.object . . . 87
1.1.1.3.2.6 Module Contents . . . 87
1.1.1.3.2.7 syft.generic.object_storage . . . 89
1.1.1.3.2.8 Module Contents . . . 89
1.1.1.3.2.9 syft.generic.string . . . 90
1.1.1.3.2.10 Module Contents . . . 90
1.1.1.3.2.11 syft.generic.tensor . . . 92
1.1.1.3.2.12 Module Contents . . . 92
1.1.1.4 syft.grid . . . 93
1.1.1.4.1 Subpackages . . . 93
1.1.1.4.1.1 syft.grid.authentication . . . 93
1.1.1.4.1.2 Submodules . . . 93
1.1.1.4.1.3 syft.grid.authentication.account . . . 93
1.1.1.4.1.4 Module Contents . . . 93
1.1.1.4.1.5 syft.grid.authentication.credential . . . 94
1.1.1.4.1.6 Module Contents . . . 94
1.1.1.4.2 Submodules . . . 94
1.1.1.4.2.1 syft.grid.abstract_grid . . . 94
1.1.1.4.2.2 Module Contents . . . 94
1.1.1.4.2.3 syft.grid.private_grid . . . 94
1.1.1.4.2.4 Module Contents . . . 94
1.1.1.4.2.5 syft.grid.public_grid . . . 96

1.1.1.4.2.6 Module Contents . . . 96
1.1.1.5 syft.messaging . . . 98
1.1.1.5.1 Subpackages . . . 98
1.1.1.5.1.1 syft.messaging.plan . . . 98
1.1.1.5.1.2 Submodules . . . 98
1.1.1.5.1.3 syft.messaging.plan.plan . . . 98
1.1.1.5.1.4 Module Contents . . . 98
1.1.1.5.1.5 syft.messaging.plan.state . . . 101
1.1.1.5.1.6 Module Contents . . . 101
1.1.1.5.1.7 Package Contents . . . 102
1.1.1.5.2 Submodules . . . 105
1.1.1.5.2.1 syft.messaging.message . . . 105
1.1.1.5.2.2 Module Contents . . . 105
1.1.1.5.2.3 syft.messaging.protocol . . . 111
1.1.1.5.2.4 Module Contents . . . 111
1.1.1.6 syft.serde . . . 114
1.1.1.6.1 Subpackages . . . 114
1.1.1.6.1.1 syft.serde.msgpack . . . 114
1.1.1.6.1.2 Submodules . . . 114
1.1.1.6.1.3 syft.serde.msgpack.native_serde . . . 114
1.1.1.6.1.4 Module Contents . . . 115
1.1.1.6.1.5 syft.serde.msgpack.proto . . . 119
1.1.1.6.1.6 Module Contents . . . 119
1.1.1.6.1.7 syft.serde.msgpack.serde . . . 120
1.1.1.6.1.8 Module Contents . . . 120
1.1.1.6.1.9 syft.serde.msgpack.torch_serde . . . 122
1.1.1.6.1.10 Module Contents . . . 122
1.1.1.6.1.11 Package Contents . . . 124
1.1.1.6.1.12 syft.serde.protobuf . . . 125
1.1.1.6.1.13 Submodules . . . 125
1.1.1.6.1.14 syft.serde.protobuf.native_serde . . . 125
1.1.1.6.1.15 Module Contents . . . 125
1.1.1.6.1.16 syft.serde.protobuf.proto . . . 125
1.1.1.6.1.17 Module Contents . . . 125
1.1.1.6.1.18 syft.serde.protobuf.serde . . . 126
1.1.1.6.1.19 Module Contents . . . 126
1.1.1.6.1.20 syft.serde.protobuf.torch_serde . . . 128
1.1.1.6.1.21 Module Contents . . . 128
1.1.1.6.1.22 Package Contents . . . 130
1.1.1.6.1.23 syft.serde.torch . . . 131
1.1.1.6.1.24 Submodules . . . 131
1.1.1.6.1.25 syft.serde.torch.serde . . . 131
1.1.1.6.1.26 Module Contents . . . 131
1.1.1.6.2 Submodules . . . 132
1.1.1.6.2.1 syft.serde.compression . . . 132
1.1.1.6.2.2 Module Contents . . . 132
1.1.1.6.2.3 syft.serde.serde . . . 133
1.1.1.6.2.4 Module Contents . . . 133
1.1.1.6.3 Package Contents . . . 134
1.1.1.7 syft.workers . . . 135
1.1.1.7.1 Submodules . . . 135
1.1.1.7.1.1 syft.workers.abstract . . . 135
1.1.1.7.1.2 Module Contents . . . 135
1.1.1.7.1.3 syft.workers.base . . . 135

1.1.1.7.1.4 Module Contents . . . 135
1.1.1.7.1.5 syft.workers.node_client . . . 143
1.1.1.7.1.6 Module Contents . . . 143
1.1.1.7.1.7 syft.workers.tfe . . . 144
1.1.1.7.1.8 Module Contents . . . 144
1.1.1.7.1.9 syft.workers.virtual . . . 145
1.1.1.7.1.10 Module Contents . . . 145
1.1.1.7.1.11 syft.workers.websocket_client . . . 145
1.1.1.7.1.12 Module Contents . . . 145
1.1.1.7.1.13 syft.workers.websocket_server . . . 147
1.1.1.7.1.14 Module Contents . . . 147
1.1.2 Submodules . . . 148
1.1.2.1 syft.codes . . . 148
1.1.2.1.1 Module Contents . . . 148
1.1.2.2 syft.dependency_check . . . 149
1.1.2.2.1 Module Contents . . . 149
1.1.2.3 syft.exceptions . . . 149
1.1.2.3.1 Module Contents . . . 149
1.1.2.4 syft.sandbox . . . 151
1.1.2.4.1 Module Contents . . . 151
1.1.2.5 syft.version . . . 151
1.1.2.5.1 Module Contents . . . 151

2 Indices and tables **153**

Python Module Index **155**

Index **157**

CHAPTER

ONE

API REFERENCE

This page contains auto-generated API reference documentation[1].

1.1 syft

PySyft is a Python library for secure, private Deep Learning. PySyft decouples private data from model training, using Federated Learning, Differential Privacy, and Multi-Party Computation (MPC) within PyTorch.

1.1.1 Subpackages

1.1.1.1 syft.federated

1.1.1.1.1 Submodules

1.1.1.1.1.1 syft.federated.federated_client

1.1.1.1.1.2 Module Contents

class syft.federated.federated_client.**FederatedClient**(*datasets=None*)

Bases: *syft.generic.object_storage.ObjectStorage*

A Client able to execute federated learning in local datasets.

add_dataset(*self*, *dataset*, *key: str*)

remove_dataset(*self*, *key: str*)

set_obj(*self*, *obj: object*)

Registers objects checking if which objects it should cache.

Parameters **obj** – An object to be registered.

_check_train_config(*self*)

_build_optimizer(*self*, *optimizer_name: str*, *model*, *optimizer_args: dict*)

Build an optimizer if needed.

Parameters

- **optimizer_name** – A string indicating the optimizer name.
- **optimizer_args** – A dict containing the args used to initialize the optimizer.

[1] Created with sphinx-autoapi

Returns A Torch Optimizer.

fit(*self*, *dataset_key: str*, *device: str = 'cpu'*, ***kwargs*)
Fits a model on the local dataset as specified in the local TrainConfig object.

Parameters

- **dataset_key** – Identifier of the local dataset that shall be used for training.
- ****kwargs** – Unused.

Returns Training loss on the last batch of training data.

Return type loss

_create_data_loader(*self*, *dataset_key: str*, *shuffle: bool = False*)

_fit(*self*, *model*, *dataset_key*, *loss_fn*, *device='cpu'*)

evaluate(*self*, *dataset_key: str*, *return_histograms: bool = False*, *nr_bins: int = -1*, *return_loss: bool = True*, *return_raw_accuracy: bool = True*, *device: str = 'cpu'*)
Evaluates a model on the local dataset as specified in the local TrainConfig object.

Parameters

- **dataset_key** – Identifier of the local dataset that shall be used for training.
- **return_histograms** – If True, calculate the histograms of predicted classes.
- **nr_bins** – Used together with calculate_histograms. Provide the number of classes/bins.
- **return_loss** – If True, loss is calculated additionally.
- **return_raw_accuracy** – If True, return nr_correct_predictions and nr_predictions

Returns

- loss: avg loss on data set, None if not calculated.
- nr_correct_predictions: number of correct predictions.
- nr_predictions: total number of predictions.
- histogram_predictions: histogram of predictions.
- histogram_target: histogram of target values in the dataset.

Return type Dictionary containing depending on the provided flags

1.1.1.1.1.3 syft.federated.train_config

1.1.1.1.1.4 Module Contents

class syft.federated.train_config.**TrainConfig**(*model: torch.jit.ScriptModule*, *loss_fn: torch.jit.ScriptModule*, *owner: AbstractWorker = None*, *batch_size: int = 32*, *epochs: int = 1*, *optimizer: str = 'SGD'*, *optimizer_args: dict = {'lr': 0.1}*, *id: Union[int, str] = None*, *max_nr_batches: int = -1*, *shuffle: bool = True*, *loss_fn_id: int = None*, *model_id: int = None*)
TrainConfig abstraction.

A wrapper object that contains all that is needed to run a training loop remotely on a federated learning setup.

__str__(*self*)
Returns the string representation of a TrainConfig.

_wrap_and_send_obj(*self*, *obj*, *location*)
Wrappers object and send it to location.

send(*self*, *location: BaseWorker*)
Gets the pointer to a new remote object.

One of the most commonly used methods in PySyft, this method serializes the object upon which it is called (self), sends the object to a remote worker, creates a pointer to that worker, and then returns that pointer from this function.

Parameters **location** – The BaseWorker object which you want to send this object to. Note that this is never actually the BaseWorker but instead a class which instantiates the BaseWorker abstraction.

Returns A weakref instance.

get(*self*, *location*)

get_model(*self*)

get_loss_fn(*self*)

static simplify(*worker: AbstractWorker*, *train_config: TrainConfig*)
Takes the attributes of a TrainConfig and saves them in a tuple.

Attention: this function does not serialize the model and loss_fn attributes of a TrainConfig instance, these are serialized and sent before. TrainConfig keeps a reference to the sent objects using _model_id and _loss_fn_id which are serialized here.

Parameters

- **worker** – the worker doing the serialization
- **train_config** – a TrainConfig object

Returns a tuple holding the unique attributes of the TrainConfig object

Return type tuple

static detail(*worker: AbstractWorker*, *train_config_tuple: tuple*)
This function reconstructs a TrainConfig object given it's attributes in the form of a tuple.

Parameters

- **worker** – the worker doing the deserialization
- **train_config_tuple** – a tuple holding the attributes of the TrainConfig

Returns A TrainConfig object

Return type train_config

1.1.1.2 syft.frameworks

1.1.1.2.1 Subpackages

1.1.1.2.1.1 syft.frameworks.keras

1.1.1.2.1.2 Subpackages

1.1.1.2.1.3 syft.frameworks.keras.layers

1.1.1.2.1.4 Submodules

1.1.1.2.1.5 syft.frameworks.keras.layers.constructor

1.1.1.2.1.6 Module Contents

syft.frameworks.keras.layers.constructor.**add_constructor_registration**(*layer_cls*)
This method rewires the layer's constructor to record arguments passed to it.

syft.frameworks.keras.layers.constructor.**filter_layers**(*layers_module*, *tfe_layers_module*)
Returns all layer types in module.

1.1.1.2.1.7 Package Contents

syft.frameworks.keras.layers.**add_constructor_registration**(*layer_cls*)
This method rewires the layer's constructor to record arguments passed to it.

syft.frameworks.keras.layers.**filter_layers**(*layers_module*, *tfe_layers_module*)
Returns all layer types in module.

1.1.1.2.1.8 syft.frameworks.keras.model

1.1.1.2.1.9 Submodules

1.1.1.2.1.10 syft.frameworks.keras.model.sequential

1.1.1.2.1.11 Module Contents

syft.frameworks.keras.model.sequential.**_args_not_supported_by_tfe = ['activity_regularizer**

syft.frameworks.keras.model.sequential.**share**(*model*, *cluster*, *target_graph=None*)
Secret share the model between *workers*.

This is done by rebuilding the model as a TF Encrypted model inside *target_graph* and pushing this graph to TFEWorkers running the cluster.

Note that this keeps a TensorFlow session alive that must later be shutdown using *model.stop()*.

syft.frameworks.keras.model.sequential.**serve**(*model*, *num_requests=5*)
Serve the specified number of predictions using the shared model, blocking until completed.

syft.frameworks.keras.model.sequential.**stop**(*model*)
Shutdown the TensorFlow session that was used to serve the shared model.

syft.frameworks.keras.model.sequential.**_configure_tfe**(*cluster*)

syft.frameworks.keras.model.sequential.**_rebuild_tfe_model**(*keras_model*, *stored_keras_weights*)
Rebuild the plaintext Keras model as a TF Encrypted Keras model from the plaintext weights in *stored_keras_weights* using the current TensorFlow graph, and the current TF Encrypted protocol and configuration.

syft.frameworks.keras.model.sequential.**_instantiate_tfe_layer**(*keras_layer*, *stored_keras_weights*)

syft.frameworks.keras.model.sequential.**_get_layer_type**(*keras_layer_cls*)

syft.frameworks.keras.model.sequential.**_trim_params**(*params*, *filter_list*)

1.1.1.2.1.12 Package Contents

syft.frameworks.keras.model.**serve**(*model*, *num_requests=5*)
Serve the specified number of predictions using the shared model, blocking until completed.

syft.frameworks.keras.model.**share**(*model*, *cluster*, *target_graph=None*)
Secret share the model between *workers*.

This is done by rebuilding the model as a TF Encrypted model inside *target_graph* and pushing this graph to TFEWorkers running the cluster.

Note that this keeps a TensorFlow session alive that must later be shutdown using *model.stop()*.

syft.frameworks.keras.model.**stop**(*model*)
Shutdown the TensorFlow session that was used to serve the shared model.

1.1.1.2.1.13 Submodules

1.1.1.2.1.14 syft.frameworks.keras.hook

1.1.1.2.1.15 Module Contents

class syft.frameworks.keras.hook.**KerasHook**(*keras*)

_hook_layers(*self*)

_hook_sequential(*self*)

1.1.1.2.1.16 syft.frameworks.tensorflow

1.1.1.2.1.17 syft.frameworks.torch

1.1.1.2.1.18 Subpackages

1.1.1.2.1.19 syft.frameworks.torch.dp

1.1.1.2.1.20 Submodules

1.1.1.2.1.21 syft.frameworks.torch.dp.pate

This script computes bounds on the privacy cost of training the student model from noisy aggregation of labels predicted by teachers. It should be used only after training the student (and therefore the teachers as well). We however include the label files required to reproduce key results from our paper (https://arxiv.org/abs/1610.05755): the epsilon bounds for MNIST and SVHN students.

1.1.1.2.1.22 Module Contents

syft.frameworks.torch.dp.pate.**compute_q_noisy_max**(*counts*, *noise_eps*)
Returns ~ Pr[outcome != winner].

Parameters

- **counts** – a list of scores
- **noise_eps** – privacy parameter for noisy_max

Returns the probability that outcome is different from true winner.

Return type q

syft.frameworks.torch.dp.pate.**compute_q_noisy_max_approx**(*counts*, *noise_eps*)
Returns ~ Pr[outcome != winner].

Parameters

- **counts** – a list of scores
- **noise_eps** – privacy parameter for noisy_max

Returns the probability that outcome is different from true winner.

Return type q

syft.frameworks.torch.dp.pate.**logmgf_exact**(*q*, *priv_eps*, *l*)
Computes the logmgf value given q and privacy eps.

The bound used is the min of three terms. The first term is from https://arxiv.org/pdf/1605.02065.pdf. The second term is based on the fact that when event has probability (1-q) for q close to zero, q can only change by exp(eps), which corresponds to a much smaller multiplicative change in (1-q) The third term comes directly from the privacy guarantee. :param q: pr of non-optimal outcome :param priv_eps: eps parameter for DP :param l: moment to compute.

Returns Upper bound on logmgf

syft.frameworks.torch.dp.pate.**logmgf_from_counts**(*counts*, *noise_eps*, *l*)
ReportNoisyMax mechanism with noise_eps with 2*noise_eps-DP in our setting where one count can go up by one and another can go down by 1.

syft.frameworks.torch.dp.pate.**sens_at_k**(*counts*, *noise_eps*, *l*, *k*)
Return sensitivity at distane k.

Parameters

- **counts** – an array of scores
- **noise_eps** – noise parameter used
- **l** – moment whose sensitivity is being computed
- **k** – distance

Returns at distance k

Return type sensitivity

syft.frameworks.torch.dp.pate.**smoothed_sens**(*counts*, *noise_eps*, *l*, *beta*)
Compute beta-smooth sensitivity.

Parameters

- **counts** – array of scors
- **noise_eps** – noise parameter
- **l** – moment of interest
- **beta** – smoothness parameter

Returns a beta smooth upper bound

Return type smooth_sensitivity

syft.frameworks.torch.dp.pate.**perform_analysis**(*teacher_preds*, *indices*, *noise_eps*, *delta=1e-05*, *moments=8*, *beta=0.09*)
"Performs PATE analysis on predictions from teachers and combined predictions for student.

Parameters

- **teacher_preds** – a numpy array of dim (num_teachers x num_examples). Each value corresponds to the index of the label which a teacher gave for a specific example
- **indices** – a numpy array of dim (num_examples) of aggregated examples which were aggregated using the noisy max mechanism.
- **noise_eps** – the epsilon level used to create the indices
- **delta** – the desired level of delta
- **moments** – the number of moments to track (see the paper)
- **beta** – a smoothing parameter (see the paper)

Returns first value is the data dependent epsilon, then the data independent epsilon

Return type tuple

syft.frameworks.torch.dp.pate.**tensors_to_literals**(*tensor_list*)
Converts list of torch tensors to list of integers/floats. Fix for not having the functionality which converts list of tensors to tensors

Parameters **tensor_list[List]** – List of torch tensors

Returns List of floats/integers

Return type literal_list[List]

syft.frameworks.torch.dp.pate.**logmgf_exact_torch**(*q*, *priv_eps*, *l*)
Computes the logmgf value given q and privacy eps. The bound used is the min of three terms. The first term is from https://arxiv.org/pdf/1605.02065.pdf. The second term is based on the fact that when event has probability (1-q) for q close to zero, q can only change by exp(eps), which corresponds to a much smaller multiplicative change in (1-q) The third term comes directly from the privacy guarantee. :param q: pr of non-optimal outcome :param priv_eps: eps parameter for DP :param l: moment to compute.

Returns Upper bound on logmgf

syft.frameworks.torch.dp.pate.**compute_q_noisy_max_torch**(*counts*, *noise_eps*)
Returns ~ Pr[outcome != winner]. :param counts: a list of scores :param noise_eps: privacy parameter for noisy_max

Returns the probability that outcome is different from true winner.

Return type q

syft.frameworks.torch.dp.pate.**logmgf_from_counts_torch**(*counts*, *noise_eps*, *l*)
ReportNoisyMax mechanism with noise_eps with 2*noise_eps-DP in our setting where one count can go up by one and another can go down by 1.

syft.frameworks.torch.dp.pate.**sens_at_k_torch**(*counts*, *noise_eps*, *l*, *k*)
Return sensitivity at distane k. :param counts: an array of scores :param noise_eps: noise parameter used :param l: moment whose sensitivity is being computed :param k: distance

Returns at distance k

Return type sensitivity

syft.frameworks.torch.dp.pate.**smooth_sens_torch**(*counts*, *noise_eps*, *l*, *beta*)
Compute beta-smooth sensitivity.

Parameters

- **counts** – array of scors
- **noise_eps** – noise parameter
- **l** – moment of interest
- **beta** – smoothness parameter

Returns a beta smooth upper bound

Return type smooth_sensitivity

syft.frameworks.torch.dp.pate.**perform_analysis_torch**(*preds*, *indices*, *noise_eps=0.1*, *delta=1e-05*, *moments=8*, *beta=0.09*)
Performs PATE analysis on predictions from teachers and combined predictions for student. :param teacher_preds: a torch tensor of dim (num_teachers x num_examples). Each value corresponds to the

index of the label which a teacher gave for a specific example

Parameters

- **indices** – a torch tensor of dim (num_examples) of aggregated examples which were aggregated using the noisy max mechanism.
- **noise_eps** – the epsilon level used to create the indices
- **delta** – the desired level of delta
- **moments** – the number of moments to track (see the paper)

- **beta** – a smoothing parameter (see the paper)

Returns first value is the data dependent epsilon, then the data independent epsilon

Return type tuple

1.1.1.2.1.23 syft.frameworks.torch.fl

1.1.1.2.1.24 Submodules

1.1.1.2.1.25 syft.frameworks.torch.fl.dataloader

1.1.1.2.1.26 Module Contents

syft.frameworks.torch.fl.dataloader.**numpy_type_map**

syft.frameworks.torch.fl.dataloader.**default_collate**(*batch*)
Puts each data field into a tensor with outer dimension batch size

class syft.frameworks.torch.fl.dataloader.**_DataLoaderIter**(*loader*, *worker_idx*)
Bases: object

Iterates once over the DataLoader's dataset, as specified by the samplers

__len__(*self*)

_get_batch(*self*)

__next__(*self*)

__iter__(*self*)

stop(*self*)

class syft.frameworks.torch.fl.dataloader.**_DataLoaderOneWorkerIter**(*loader*, *worker_idx*)
Bases: object

Iterates once over the worker's dataset, as specified by its sampler

_get_batch(*self*)

__next__(*self*)

__iter__(*self*)

stop(*self*)

class syft.frameworks.torch.fl.dataloader.**FederatedDataLoader**(*federated_dataset*, *batch_size=8*, *shuffle=False*, *num_iterators=1*, *drop_last=False*, *collate_fn=default_collate*, *iter_per_worker=False*, ***kwargs*)
Bases: object

Data loader. Combines a dataset and a sampler, and provides single or several iterators over the dataset.

Parameters

- **federated_dataset** (FederatedDataset) – dataset from which to load the data.
- **batch_size** (*int, optional*) – how many samples per batch to load (default: 1).
- **shuffle** (*bool, optional*) – set to True to have the data reshuffled at every epoch (default: False).
- **collate_fn** (*callable, optional*) – merges a list of samples to form a mini-batch.
- **drop_last** (*bool, optional*) – set to True to drop the last incomplete batch, if the dataset size is not divisible by the batch size. If False and the size of dataset is not divisible by the batch size, then the last batch will be smaller. (default: False)
- **num_iterators** (*int*) – number of workers from which to retrieve data in parallel. num_iterators <= len(federated_dataset.workers) - 1 the effect is to retrieve num_iterators epochs of data but at each step data from num_iterators distinct workers is returned.
- **iter_per_worker** (*bool*) – if set to true, __next__() will return a dictionary containing one batch per worker

__initialized = False

__iter__(*self*)

__next__(*self*)

__len__(*self*)

1.1.1.2.1.27 syft.frameworks.torch.fl.dataset

1.1.1.2.1.28 Module Contents

syft.frameworks.torch.fl.dataset.**logger**

class syft.frameworks.torch.fl.dataset.**BaseDataset**(*data*, *targets*, *transform=None*)

This is a base class to be used for manipulating a dataset. This is composed of a .data attribute for inputs and a .targets one for labels. It is to be used like the MNIST Dataset object, and is useful to avoid handling the two inputs and label tensors separately.

Parameters

- **tensors]** (*data[list,torch*) – the data points
- **targets** – Corresponding labels of the data points
- **transform** – Function to transform the datapoints

fix_precision

float_precision

__len__(*self*)

__getitem__(*self*, *index*)

Parameters **index[integer]** – index of item to get

Returns Data points corresponding to the given index targets: Targets correspoding to given datapoint

Return type data

transform(*self*, *transform*)
Allows a transform to be applied on given dataset. :param transform: The transform to be applied on the data

send(*self*, *worker*)

Parameters **class]** (*worker[worker*) – worker to which the data must be sent

Returns Return the object instance with data sent to corresponding worker

Return type self

get(*self*)
Gets the data back from respective workers.

fix_prec(*self*, **args*, ***kwargs*)
Converts data of BaseDataset into fixed precision

float_prec(*self*, **args*, ***kwargs*)
Converts data of BaseDataset into float precision

share(*self*, **args*, ***kwargs*)
Share the data with the respective workers

property location(*self*)
Get location of the data

syft.frameworks.torch.fl.dataset.**dataset_federate**(*dataset*, *workers*)
Add a method to easily transform a torch.Dataset or a sy.BaseDataset into a sy.FederatedDataset. The dataset given is split in len(workers) part and sent to each workers

syft.frameworks.torch.fl.dataset.**federate**

syft.frameworks.torch.fl.dataset.**federate**

class syft.frameworks.torch.fl.dataset.**FederatedDataset**(*datasets*)

property workers(*self*)
Returns: list of workers

__getitem__(*self*, *worker_id*)

Parameters **worker_id[str,int]** – ID of respective worker

Returns: Get Datasets from the respective worker

__len__(*self*)

__repr__(*self*)

1.1.1.2.1.29 syft.frameworks.torch.fl.utils

1.1.1.2.1.30 Module Contents

syft.frameworks.torch.fl.utils.**logger**

syft.frameworks.torch.fl.utils.**extract_batches_per_worker**(*federated_train_loader: sy.FederatedDataLoader*)
Extracts the batches from the federated_train_loader and stores them in a dictionary (keys = data.location).

Args: federated_train_loader: the connection object we use to send responses.

back to the client.

syft.frameworks.torch.fl.utils.**add_model**(*dst_model*, *src_model*)
Add the parameters of two models.

Parameters

- **dst_model** (*torch.nn.Module*) – the model to which the src_model will be added.
- **src_model** (*torch.nn.Module*) – the model to be added to dst_model.

Returns the resulting model of the addition.

Return type torch.nn.Module

syft.frameworks.torch.fl.utils.**scale_model**(*model*, *scale*)
Scale the parameters of a model.

Parameters

- **model** (*torch.nn.Module*) – the models whose parameters will be scaled.
- **scale** (*float*) – the scaling factor.

Returns the module with scaled parameters.

Return type torch.nn.Module

syft.frameworks.torch.fl.utils.**federated_avg**(*models: List[torch.nn.Module]*) → torch.nn.Module
Calculate the federated average of a list of models.

Parameters **models** (*List[torch.nn.Module]*) – the models of which the federated average is calculated.

Returns the module with averaged parameters.

Return type torch.nn.Module

syft.frameworks.torch.fl.utils.**accuracy**(*pred_softmax*, *target*)
Calculate the accuray of a given prediction.

This functions assumes pred_softmax to be converted into the final prediction by taking the argmax.

Parameters

- **pred_softmax** – array type(float), providing nr_classes values per element in target.
- **target** – array type(int), correct classes, taking values in range [0, nr_classes).

Returns float, fraction of correct predictions.

Return type accuracy

syft.frameworks.torch.fl.utils.**create_gaussian_mixture_toy_data**(*nr_samples: int*)
Create a simple toy data for binary classification

The data is drawn from two normal distributions target = 1: mu = 2, sigma = 1 target = 0: mu = 0, sigma = 1
The dataset is balanced with an equal number of positive and negative samples

Parameters **nr_samples** – number of samples to generate

Returns data, targets

syft.frameworks.torch.fl.utils.**iris_data_partial**()
Returns: 30 samples from the iris data set: https://archive.ics.uci.edu/ml/datasets/iris

1.1.1.2.1.31 Package Contents

class syft.frameworks.torch.fl.**BaseDataset**(*data*, *targets*, *transform=None*)

This is a base class to be used for manipulating a dataset. This is composed of a .data attribute for inputs and a .targets one for labels. It is to be used like the MNIST Dataset object, and is useful to avoid handling the two inputs and label tensors separately.

Parameters

- **tensors]** (*data[list,torch*) – the data points
- **targets** – Corresponding labels of the data points
- **transform** – Function to transform the datapoints

fix_precision

float_precision

__len__(*self*)

__getitem__(*self*, *index*)

Parameters **index[integer]** – index of item to get

Returns Data points corresponding to the given index targets: Targets correspoding to given datapoint

Return type data

transform(*self*, *transform*)

Allows a transform to be applied on given dataset. :param transform: The transform to be applied on the data

send(*self*, *worker*)

Parameters **class]** (*worker[worker*) – worker to which the data must be sent

Returns Return the object instance with data sent to corresponding worker

Return type self

get(*self*)

Gets the data back from respective workers.

fix_prec(*self*, **args*, ***kwargs*)

Converts data of BaseDataset into fixed precision

float_prec(*self*, **args*, ***kwargs*)

Converts data of BaseDataset into float precision

share(*self*, **args*, ***kwargs*)

Share the data with the respective workers

property location(*self*)

Get location of the data

class syft.frameworks.torch.fl.**FederatedDataset**(*datasets*)

property workers(*self*)

Returns: list of workers

__getitem__(*self*, *worker_id*)

Parameters **worker_id[str,int]** – ID of respective worker

Returns: Get Datasets from the respective worker

__len__(*self*)

__repr__(*self*)

class syft.frameworks.torch.fl.**FederatedDataLoader**(*federated_dataset*, *batch_size=8*, *shuffle=False*, *num_iterators=1*, *drop_last=False*, *collate_fn=default_collate*, *iter_per_worker=False*, ***kwargs*)

Bases: object

Data loader. Combines a dataset and a sampler, and provides single or several iterators over the dataset.

Parameters

- **federated_dataset** (FederatedDataset) – dataset from which to load the data.
- **batch_size** (*int, optional*) – how many samples per batch to load (default: 1).
- **shuffle** (*bool, optional*) – set to True to have the data reshuffled at every epoch (default: False).
- **collate_fn** (*callable, optional*) – merges a list of samples to form a mini-batch.
- **drop_last** (*bool, optional*) – set to True to drop the last incomplete batch, if the dataset size is not divisible by the batch size. If False and the size of dataset is not divisible by the batch size, then the last batch will be smaller. (default: False)
- **num_iterators** (*int*) – number of workers from which to retrieve data in parallel. num_iterators <= len(federated_dataset.workers) - 1 the effect is to retrieve num_iterators epochs of data but at each step data from num_iterators distinct workers is returned.
- **iter_per_worker** (*bool*) – if set to true, __next__() will return a dictionary containing one batch per worker

__initialized = False

__iter__(*self*)

__next__(*self*)

__len__(*self*)

1.1.1.2.1.32 syft.frameworks.torch.he

1.1.1.2.1.33 Submodules

1.1.1.2.1.34 syft.frameworks.torch.he.paillier

1.1.1.2.1.35 Module Contents

syft.frameworks.torch.he.paillier.**keygen**

1.1.1.2.1.36 syft.frameworks.torch.hook

1.1.1.2.1.37 Submodules

1.1.1.2.1.38 syft.frameworks.torch.hook.hook

1.1.1.2.1.39 Module Contents

class syft.frameworks.torch.hook.hook.**TorchHook**(*torch*, *local_worker: BaseWorker = None*, *is_client: bool = True*, *verbose: bool = True*)

Bases: *syft.generic.frameworks.hook.hook.FrameworkHook*

A Hook which Overrides Methods on PyTorch Tensors.

The purpose of this class is to:

- extend torch methods to allow for the moving of tensors from one

worker to another. * override torch methods to execute commands on one worker that are called on tensors controlled by the local worker.

This class is typically the first thing you will initialize when using PySyft with PyTorch because it is responsible for augmenting PyTorch with PySyft's added functionality (such as remote execution).

Parameters

- **local_worker** – An optional BaseWorker instance that lets you provide a local worker as a parameter which TorchHook will assume to be the worker owned by the local machine. If you leave it empty, TorchClient will automatically initialize a `workers.VirtualWorker` under the assumption you're looking to do local experimentation or development.
- **is_client** – An optional boolean parameter (default True), indicating whether TorchHook is being initialized as an end-user client.This can impact whether or not variables are deleted when they fall out of scope. If you set this incorrectly on a end user client, Tensors and Variables will never be deleted. If you set this incorrectly on a remote machine (not a client), tensors will not get saved. It's really only important if you're not initializing the local worker yourself.
- **verbose** – An optional boolean parameter (default True) to indicate whether or not to print the operations as they occur.
- **queue_size** – An integer optional parameter (default 0) to specify the max length of the list that stores the messages to be sent.

Example

```
>>> import torch as th
>>> import syft as sy
>>> hook = sy.TorchHook(th)
Hooking into Torch...
Overloading Complete.
# constructing a normal torch tensor in pysyft
>>> x = th.Tensor([-2,-1,0,1,2,3])
>>> x
-2
```

(continues on next page)

(continued from previous page)

```
-1
0
1
2
3
[syft.core.frameworks.torch.tensor.FloatTensor of size 6]
```

create_shape(*cls*, *shape_dims*)

create_wrapper(*cls*, *wrapper_type*)

create_zeros(*cls*, **shape*, *dtype=None*, ***kwargs*)

_hook_native_tensor(*self*, *tensor_type: type*, *syft_type: type*)
Adds PySyft Tensor Functionality to the given native tensor type.

Overloads the given native Torch tensor to add PySyft Tensor Functionality. Overloading involves modifying the tensor type with PySyft's added functionality. You may read about what kind of modifications are made in the methods that this method calls.

Parameters

- **tensor_type** – The type of tensor being hooked (in this refactor this is only ever torch.Tensor, but in previous versions of PySyft this iterated over all tensor types.
- **syft_type** – The abstract type whose methods should all be added to the tensor_type class. In practice this is always TorchTensor. Read more about it there.

__hook_properties(*self*, *tensor_type*)

_hook_syft_tensor_methods(*self*, *syft_type: type*)

_hook_private_tensor_methods(*self*, *syft_type: type*)

_hook_worker_methods(*self*)

_get_hooked_base_worker_method(*hook_self*, *attr*)

_hook_additive_shared_tensor_methods(*self*)
Add hooked version of all methods of the torch Tensor to the Additive Shared tensor: instead of performing the native tensor method, it will be forwarded to each share when it is relevant

_hook_parameters(*self*)
This method overrides the torch Parameter class such that it works correctly with our overridden tensor types. The native torch Parameter class kept deleting all of our attributes on our custom tensors, so we wrote our own.

_hook_torch_module(*self*)
Overloads functions in the main torch modules. The way this is accomplished is by first moving all existing module functions in the torch module to native_<function_name_here>.

Example

the real `torch.cat()` will become `torch.native_cat()` and `torch.cat()` will have our hooking code.

classmethod _get_hooked_func(*cls*, *public_module_name*, *func_api_name*, *attr*)
Torch-specific implementation. See the subclass for more.

_get_hooked_additive_shared_method(*hook_self*, *attr*)
Hook a method to send it multiple remote workers

Parameters **attr** (*str*) – the method to hook

Returns the hooked method

_hook_tensor(*hook_self*)
Hooks the function torch.tensor() We need to do this seperately from hooking the class because internally torch does not pick up the change to add the args :param hook_self: the hook itself

classmethod _transfer_methods_to_native_tensor(*cls*, *tensor_type: type*, *syft_type: type*)
Adds methods from the TorchTensor class to the native torch tensor.

The class TorchTensor is a proxy to avoid extending directly the torch tensor class.

Parameters **tensor_type** – The tensor type to which we are adding methods from TorchTensor class.

_hook_module(*self*)
Overloading torch.nn.Module with PySyft functionality, the primary module responsible for core ML functionality such as Neural network layers and loss functions. It is important to note that all the operations are actually in-place.

_hook_optim(*self*)
Overloading torch.optim.Optimizer with PySyft functionality. Optimizer hyper-parameters should indeed be converted to fixed precision to interact with fixed precision or additive shared tensors. It is important to note that all the operations are actually in-place.

1.1.1.2.1.40 syft.frameworks.torch.hook.hook_args

1.1.1.2.1.41 Module Contents

syft.frameworks.torch.hook.hook_args.**type_rule**

syft.frameworks.torch.hook.hook_args.**forward_func**

syft.frameworks.torch.hook.hook_args.**backward_func**

syft.frameworks.torch.hook.hook_args.**ambiguous_methods**

syft.frameworks.torch.hook.hook_args.**ambiguous_functions**

1.1.1.2.1.42 syft.frameworks.torch.linalg

1.1.1.2.1.43 Submodules

1.1.1.2.1.44 syft.frameworks.torch.linalg.lr

1.1.1.2.1.45 Module Contents

class syft.frameworks.torch.linalg.lr.**EncryptedLinearRegression**(*crypto_provider: BaseWorker*, *hbc_worker: BaseWorker*, *precision_fractional: int = 6*, *fit_intercept: bool = True*)

Multi-Party Linear Regressor based on Jonathan Bloom's algorithm. It performs linear regression using Secure Multi-Party Computation. While the training is performed in SMPC, the final regression coefficients are public at the end and predictions are made in clear on local or pointer Tensors.

Reference: Section 2 of https://arxiv.org/abs/1901.09531

Parameters

- **crypto_provider** – a BaseWorker providing crypto elements for AdditiveSharingTensors (used for SMPC) such as Beaver triples
- **hbc_worker** – The "Honest but Curious" BaseWorker. SMPC operations in PySyft use SecureNN protocols, which are based on 3-party computations. In order to apply it for more than 3 parties, we need a "Honest but Curious" worker. To perform the Encrypted Linear Regression, the algorithm chooses randomly one of the workers in the pool and secret share all tensors with the chosen worker, the crypto provider and the "Honest but Curious" worker. Its main role is to avoid collusion between two workers in the pool if the algorithm secred shared the tensors with two randomly chosen workers and the crypto provider. The "Honest but Curious" worker is essentially a legitimate participant in a communication protocol who will not deviate from the defined protocol but will attempt to learn all possible information from legitimately received messages.
- **precision_fractional** – precision chosen for FixedPrecisionTensors
- **fit_intercept** – whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (e.g. data is expected to be already centered)

coef
: torch.Tensor of shape (n_features,). Estimated coefficients for the linear regression problem.

intercept
: torch.Tensor of shape (1,) if fit_intercept is set to True, None otherwise. Estimated intercept for the linear regression.

pvalue_coef
: numpy.array of shape (n_features,). Two-sided p-value for a hypothesis test whose null hypothesis is that the each coeff is zero.

pvalue_intercept
: numpy.array of shape (1,) if fit_intercept is set to True, None otherwise. Two-sided p-value for a hypothesis test whose null hypothesis is that the intercept is zero.

fit (*self, X_ptrs: List[torch.Tensor], y_ptrs: List[torch.Tensor]*)
Fits the linear model using Secured Multi-Party Linear Regression. The final results (i.e. coefficients and p-values) will be public.

predict (*self*, *X: torch.Tensor*)
Performs predicion of linear model on X, which can be a local torch.Tensor or a wrapped PointerTensor. The result will be either a local torch.Tensor or a wrapped PointerTensor, depending on the nature of X.

summarize (*self*)
Prints a summary of the coefficients and its statistics. This method should be called only after training of the model.

_check_ptrs (*self*, *X_ptrs*, *y_ptrs*)
Method that check if the lists of pointers corresponding to the explanatory and explained variables have their elements as expected. It also computes parallelly some Regressor's attributes such as number of features and total sample size.

static _add_intercept (*X_ptrs*)
Adds a column-vector of 1's at the beginning of the tensors X_ptrs

static _get_workers (*ptrs*)
Method that returns the pool of workers in a tuple

static _remote_dot_products (*X_ptrs*, *y_ptrs*)
This method computes the aggregated dot-products remotely. It corresponds to the Compression stage (or Compression within) of Bloom's algorithm

_share_ptrs (*self*, *ptrs*, *worker_idx*)
Method that secret share a list of remote tensors between a worker of the pool and the 'honest but curious' worker, using a crypto_provider worker

_compute_pvalues (*self*)
Compute p-values of coefficients (and intercept if fit_intercept==True)

class syft.frameworks.torch.linalg.lr.**DASH**(*crypto_provider: BaseWorker*, *hbc_worker: BaseWorker*, *precision_fractional: int = 6*)
Distributed Association Scan Hammer (DASH) algorithm based on Jonathan Bloom's algorithm. It uses Secured Multi-Party Computation at combine phase. While the training is performed in SMPC, the final regression coefficients are public at the end.

Reference: Section 2 of https://arxiv.org/abs/1901.09531

Parameters

- **crypto_provider** – a BaseWorker providing crypto elements for ASTs such as Beaver triples
- **hbc_worker** – The "Honest but Curious" BaseWorker. SMPC operations in PySyft use SecureNN protocols, which are based on 3-party computations. In order to apply it for more than 3 parties, we need a "Honest but Curious" worker. To perform the DASH algorithm, we choose randomly one of the workers in the pool and secret share all tensors with the chosen worker,the crypto provider and the "Honest but Curious" worker. Its main role is to avoid collusion between two workers in the pool if the algorithm secred shared the tensors with two randomly chosen workers and the crypto provider. The "Honest but Curious" worker is essentially a legitimate participant in a communication protocol who will not deviate from the defined protocol but will attempt to learn all possible information from legitimately received messages.
- **precision_fractional** – precision chosen for FixedPrecisionTensors

coef
torch.Tensor of shape (n_features,). Estimated coefficients for DASH algorithm.

pvalue
numpy.array of shape (n_features,). Two-sided p-value for a hypothesis test whose null hypothesis is that the each coeff is zero.

fit (*self, X_ptrs: List[torch.Tensor], C_ptrs: List[torch.Tensor], y_ptrs: List[torch.Tensor]*)

get_coeff (*self*)

get_standard_errors (*self*)

get_p_values (*self*)

_check_ptrs (*self*, *X_ptrs*, *C_ptrs*, *y_ptrs*)
Method that check if the lists of pointers corresponding to the response vector, transient covariate vectors and independent permanent covariate vectors have their elements as expected. It also computes parallelly some Regressor's attributes such as degrees of freedom and total sample size.

static _get_workers (*ptrs*)
Method that returns the pool of workers in a tuple

static _remote_dot_products (*X_ptrs*, *C_ptrs*, *y_ptrs*)
This method computes the aggregated dot-products remotely. It corresponds to the Compression stage (or Compression within) of DASH algorithm

static _remote_qr (*C_ptrs*)
Performs the QR decompositions of permanent covariate matrices remotely. It returns a list with the upper right matrices located in each worker

static _inv_upper (*R*)
Performs the inversion of a right upper matrix (2-dim tensor) in MPC by solving the linear equation R * R_inv = I with backward substitution.

_share_ptrs (*self*, *ptrs*, *worker_idx*)
Method that secret share a list of remote tensors between a worker of the pool and the 'honest but curious' worker, using a crypto_provider worker

_compute_pvalues (*self*)
Compute p-values of coefficients

1.1.1.2.1.46 syft.frameworks.torch.linalg.operations

1.1.1.2.1.47 Module Contents

syft.frameworks.torch.linalg.operations.**inv_sym** (*t*)
This function performs the inversion of a symmetric matrix (2-dim tensor) in MPC. It uses LDLt decomposition, which is better than Cholensky decomposition in our case since it doesn't use square root. Algorithm reference: https://arxiv.org/abs/1111.4144 - Section IV

Parameters **t** – symmetric 2-dim tensor

Returns inverse of t as 2-dim tensor

Return type t_inv

syft.frameworks.torch.linalg.operations.**_ldl** (*t*)
This function performs the LDLt decomposition of a symmetric matrix (2-dim tensor)

Parameters **t** – symmetric 2-dim tensor

Returns

lower triangular matrix as a 2-dim tensor with same type as t d: 1-dim tensor which represents the diagonal in the LDLt decomposition inv_d: 1-dim tensor which represents the inverse of the diagonal d. It is usefull

when computing inverse of a symmetric matrix, by caching it we avoid repeated computations with division, which is very slow in MPC

Return type l

syft.frameworks.torch.linalg.operations.**qr**(*t*, *mode='reduced'*, *norm_factor=None*)

This function performs the QR decomposition of a matrix (2-dim tensor). The decomposition is performed using Householder Reflection.

Parameters

- **t** – 2-dim tensor, shape(M, N). It should be whether a local tensor, a pointer to a remote tensor or an AdditiveSharedTensor
- **mode** – {'reduced', 'complete', 'r'}. If K = min(M, N), then - 'reduced' : returns q, r with dimensions (M, K), (K, N) (default) - 'complete' : returns q, r with dimensions (M, M), (M, N) - 'r' : returns r only with dimensions (K, N)
- **norm_factor** – float. The normalization factor used to avoid overflow when performing QR decomposition on an AdditiveSharedTensor. For example in the case of the DASH algorithm, this norm_factor should be of the order of the square root of number of entries in the original matrix used to perform the compression phase assuming the entries are standardized.

Returns orthogonal matrix as a 2-dim tensor with same type as t r: lower triangular matrix as a 2-dim tensor with same type as t

Return type q

syft.frameworks.torch.linalg.operations.**_norm_mpc**(*t*, *norm_factor*)

Computation of a norm of a vector in MPC. The vector should be an AdditiveSharedTensor.

It performs the norm calculation by masking the tensor with a multiplication by a big random number drawn from a uniform distribution, and computing the square root of the squared norm of the masked tensor, which is computed beforehand with a dot product in MPC.

In order to maintain stability and avoid overflow, this functions uses a norm_factor that scales down the tensor for MPC computations and rescale it at the end. For example in the case of the DASH algorithm, this norm_factor should be of the order of the square root of number of entries in the original matrix used to perform the compression phase assuming the entries are standardized.

Parameters

- **t** – 1-dim AdditiveSharedTensor, representing a vector.
- **norm_factor** – float. The normalization factor used to avoid overflow

Returns the norm of the vector as an AdditiveSharedTensor

1.1.1.2.1.48 Package Contents

syft.frameworks.torch.linalg.**inv_sym**(*t*)

This function performs the inversion of a symmetric matrix (2-dim tensor) in MPC. It uses LDLt decomposition, which is better than Cholensky decomposition in our case since it doesn't use square root. Algorithm reference: https://arxiv.org/abs/1111.4144 - Section IV

Parameters **t** – symmetric 2-dim tensor

Returns inverse of t as 2-dim tensor

Return type t_inv

syft.frameworks.torch.linalg.**qr**(*t*, *mode='reduced'*, *norm_factor=None*)

This function performs the QR decomposition of a matrix (2-dim tensor). The decomposition is performed using Householder Reflection.

Parameters

- **t** – 2-dim tensor, shape(M, N). It should be whether a local tensor, a pointer to a remote tensor or an AdditiveSharedTensor
- **mode** – {'reduced', 'complete', 'r'}. If K = min(M, N), then - 'reduced' : returns q, r with dimensions (M, K), (K, N) (default) - 'complete' : returns q, r with dimensions (M, M), (M, N) - 'r' : returns r only with dimensions (K, N)
- **norm_factor** – float. The normalization factor used to avoid overflow when performing QR decomposition on an AdditiveSharedTensor. For example in the case of the DASH algorithm, this norm_factor should be of the order of the square root of number of entries in the original matrix used to perform the compression phase assuming the entries are standardized.

Returns orthogonal matrix as a 2-dim tensor with same type as t r: lower triangular matrix as a 2-dim tensor with same type as t

Return type q

class syft.frameworks.torch.linalg.**EncryptedLinearRegression**(*crypto_provider: BaseWorker*, *hbc_worker: BaseWorker*, *precision_fractional: int = 6*, *fit_intercept: bool = True*)

Multi-Party Linear Regressor based on Jonathan Bloom's algorithm. It performs linear regression using Secure Multi-Party Computation. While the training is performed in SMPC, the final regression coefficients are public at the end and predictions are made in clear on local or pointer Tensors.

Reference: Section 2 of https://arxiv.org/abs/1901.09531

Parameters

- **crypto_provider** – a BaseWorker providing crypto elements for AdditiveSharingTensors (used for SMPC) such as Beaver triples
- **hbc_worker** – The "Honest but Curious" BaseWorker. SMPC operations in PySyft use SecureNN protocols, which are based on 3-party computations. In order to apply it for more than 3 parties, we need a "Honest but Curious" worker. To perform the Encrypted Linear Regression, the algorithm chooses randomly one of the workers in the pool and secret share all tensors with the chosen worker, the crypto provider and the "Honest but Curious" worker. Its main role is to avoid collusion between two workers in the pool if the algorithm secred

shared the tensors with two randomly chosen workers and the crypto provider. The "Honest but Curious" worker is essentially a legitimate participant in a communication protocol who will not deviate from the defined protocol but will attempt to learn all possible information from legitimately received messages.

- **precision_fractional** – precision chosen for FixedPrecisionTensors
- **fit_intercept** – whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (e.g. data is expected to be already centered)

coef
torch.Tensor of shape (n_features,). Estimated coefficients for the linear regression problem.

intercept
torch.Tensor of shape (1,) if fit_intercept is set to True, None otherwise. Estimated intercept for the linear regression.

pvalue_coef
numpy.array of shape (n_features,). Two-sided p-value for a hypothesis test whose null hypothesis is that the each coeff is zero.

pvalue_intercept
numpy.array of shape (1,) if fit_intercept is set to True, None otherwise. Two-sided p-value for a hypothesis test whose null hypothesis is that the intercept is zero.

fit(*self, X_ptrs: List[torch.Tensor], y_ptrs: List[torch.Tensor]*)
Fits the linear model using Secured Multi-Party Linear Regression. The final results (i.e. coefficients and p-values) will be public.

predict(*self*, *X: torch.Tensor*)
Performs predicion of linear model on X, which can be a local torch.Tensor or a wrapped PointerTensor. The result will be either a local torch.Tensor or a wrapped PointerTensor, depending on the nature of X.

summarize(*self*)
Prints a summary of the coefficients and its statistics. This method should be called only after training of the model.

_check_ptrs(*self*, *X_ptrs*, *y_ptrs*)
Method that check if the lists of pointers corresponding to the explanatory and explained variables have their elements as expected. It also computes parallelly some Regressor's attributes such as number of features and total sample size.

static _add_intercept(*X_ptrs*)
Adds a column-vector of 1's at the beginning of the tensors X_ptrs

static _get_workers(*ptrs*)
Method that returns the pool of workers in a tuple

static _remote_dot_products(*X_ptrs*, *y_ptrs*)
This method computes the aggregated dot-products remotely. It corresponds to the Compression stage (or Compression within) of Bloom's algorithm

_share_ptrs(*self*, *ptrs*, *worker_idx*)
Method that secret share a list of remote tensors between a worker of the pool and the 'honest but curious' worker, using a crypto_provider worker

_compute_pvalues(*self*)
Compute p-values of coefficients (and intercept if fit_intercept==True)

class syft.frameworks.torch.linalg.**DASH**(*crypto_provider: BaseWorker*, *hbc_worker: BaseWorker*, *precision_fractional: int = 6*)
Distributed Association Scan Hammer (DASH) algorithm based on Jonathan Bloom's algorithm. It uses Secured

Multi-Party Computation at combine phase. While the training is performed in SMPC, the final regression coefficients are public at the end.

Reference: Section 2 of https://arxiv.org/abs/1901.09531

Parameters

- **crypto_provider** – a BaseWorker providing crypto elements for ASTs such as Beaver triples
- **hbc_worker** – The "Honest but Curious" BaseWorker. SMPC operations in PySyft use SecureNN protocols, which are based on 3-party computations. In order to apply it for more than 3 parties, we need a "Honest but Curious" worker. To perform the DASH algorithm, we choose randomly one of the workers in the pool and secret share all tensors with the chosen worker,the crypto provider and the "Honest but Curious" worker. Its main role is to avoid collusion between two workers in the pool if the algorithm secred shared the tensors with two randomly chosen workers and the crypto provider. The "Honest but Curious" worker is essentially a legitimate participant in a communication protocol who will not deviate from the defined protocol but will attempt to learn all possible information from legitimately received messages.
- **precision_fractional** – precision chosen for FixedPrecisionTensors

coef
torch.Tensor of shape (n_features,). Estimated coefficients for DASH algorithm.

pvalue
numpy.array of shape (n_features,). Two-sided p-value for a hypothesis test whose null hypothesis is that the each coeff is zero.

fit (*self, X_ptrs: List[torch.Tensor], C_ptrs: List[torch.Tensor], y_ptrs: List[torch.Tensor]*)

get_coeff (*self*)

get_standard_errors (*self*)

get_p_values (*self*)

_check_ptrs (*self*, *X_ptrs*, *C_ptrs*, *y_ptrs*)
Method that check if the lists of pointers corresponding to the response vector, transient covariate vectors and independent permanent covariate vectors have their elements as expected. It also computes parallelly some Regressor's attributes such as degrees of freedom and total sample size.

static _get_workers (*ptrs*)
Method that returns the pool of workers in a tuple

static _remote_dot_products (*X_ptrs*, *C_ptrs*, *y_ptrs*)
This method computes the aggregated dot-products remotely. It corresponds to the Compression stage (or Compression within) of DASH algorithm

static _remote_qr (*C_ptrs*)
Performs the QR decompositions of permanent covariate matrices remotely. It returns a list with the upper right matrices located in each worker

static _inv_upper (*R*)
Performs the inversion of a right upper matrix (2-dim tensor) in MPC by solving the linear equation R * R_inv = I with backward substitution.

_share_ptrs (*self*, *ptrs*, *worker_idx*)
Method that secret share a list of remote tensors between a worker of the pool and the 'honest but curious' worker, using a crypto_provider worker

_compute_pvalues(*self*)
Compute p-values of coefficients

1.1.1.2.1.49 syft.frameworks.torch.mpc

1.1.1.2.1.50 Submodules

1.1.1.2.1.51 syft.frameworks.torch.mpc.beaver

1.1.1.2.1.52 Module Contents

syft.frameworks.torch.mpc.beaver.**request_triple**(*crypto_provider: AbstractWorker*, *cmd: Callable*, *field: int*, *a_size: tuple*, *b_size: tuple*, *locations: list*)
Generates a multiplication triple and sends it to all locations.

Parameters

- **crypto_provider** – worker you would like to request the triple from
- **cmd** – An equation in einsum notation.
- **field** – An integer representing the field size.
- **a_size** – A tuple which is the size that a should be or a torch.Size instance
- **b_size** – A tuple which is the size that b should be or a torch.Size intance
- **locations** – A list of workers where the triple should be shared between.

Returns A triple of AdditiveSharedTensors such that c_shared = cmd(a_shared, b_shared).

1.1.1.2.1.53 syft.frameworks.torch.mpc.securenn

This is an implementation of the SecureNN paper https://eprint.iacr.org/2018/442.pdf

Note that there is a difference here in that our shares can be negative numbers while they are always positive in the paper

1.1.1.2.1.54 Module Contents

syft.frameworks.torch.mpc.securenn.**p = 67**

syft.frameworks.torch.mpc.securenn.**Q_BITS = 62**

syft.frameworks.torch.mpc.securenn.**no_wrap**

syft.frameworks.torch.mpc.securenn.**decompose**(*tensor*)
decompose a tensor into its binary representation.

syft.frameworks.torch.mpc.securenn.**flip**(*x*, *dim*)
Reverse the order of the elements in a tensor

syft.frameworks.torch.mpc.securenn.**_random_common_bit**(**workers*)
Return a bit chosen by a worker and sent to all workers, in the form of a MultiPointerTensor

syft.frameworks.torch.mpc.securenn.**_random_common_value**(*max_value*, **workers*)
Return n in [0, max_value-1] chosen by a worker and sent to all workers, in the form of a MultiPointerTensor

syft.frameworks.torch.mpc.securenn.**_shares_of_zero**(*size*, *field*, *crypto_provider*, **workers*)
Return n in [0, max_value-1] chosen by a worker and sent to all workers, in the form of a MultiPointerTensor

syft.frameworks.torch.mpc.securenn.**select_share**(*alpha_sh*, *x_sh*, *y_sh*)
Performs select share protocol If the bit alpha_sh is 0, x_sh is returned If the bit alpha_sh is 1, y_sh is returned

Parameters

- **x_sh** (AdditiveSharingTensor) – the first share to select
- **y_sh** (AdditiveSharingTensor) – the second share to select
- **alpha_sh** (AdditiveSharingTensor) – the bit to choose between x_sh and y_sh

Returns z_sh = (1 - alpha_sh) * x_sh + alpha_sh * y_sh

syft.frameworks.torch.mpc.securenn.**private_compare**(*x_bit_sh*, *r*, *beta*)
Perform privately x > r

Parameters

- **x** (*AdditiveSharedTensor*) – the private tensor
- **r** (MultiPointerTensor) – the threshold commonly held by alice and bob
- **beta** (MultiPointerTensor) – a boolean commonly held by alice and bob to hide the result of computation for the crypto provider

Returns = (x > r).

syft.frameworks.torch.mpc.securenn.**msb**(*a_sh*)
Compute the most significant bit in a_sh, this is an implementation of the SecureNN paper https://eprint.iacr.org/2018/442.pdf

Parameters **a_sh** (AdditiveSharingTensor) – the tensor of study

Returns the most significant bit

syft.frameworks.torch.mpc.securenn.**share_convert**(*a_sh*)
Convert shares of a in field L to shares of a in field L - 1

Parameters **a_sh** (AdditiveSharingTensor) – the additive sharing tensor who owns the shares in field L to convert

Returns An additive sharing tensor with shares in field L-1

syft.frameworks.torch.mpc.securenn.**relu_deriv**(*a_sh*)
Compute the derivative of Relu

Parameters **a_sh** (AdditiveSharingTensor) – the private tensor on which the op applies

Returns 0 if Dec(a_sh) < 0 1 if Dec(a_sh) > 0 encrypted in an AdditiveSharingTensor

syft.frameworks.torch.mpc.securenn.**relu**(*a_sh*)
Compute Relu

Parameters **a_sh** (AdditiveSharingTensor) – the private tensor on which the op applies

Returns Dec(a_sh) > 0 encrypted in an AdditiveSharingTensor

syft.frameworks.torch.mpc.securenn.**division**(*x_sh*, *y_sh*, *bit_len_max=Q_BITS // 2*)
Performs division of encrypted numbers

Parameters

- **y_sh** (*x_sh,*) – the private tensors on which the op applies

- **bit_len_max** – the number of bits needed to represent the highest value in the tensors we may want to avoid giving this value so Q_BITS is the default value

Returns element-wise integer division of x_sh by y_sh

syft.frameworks.torch.mpc.securenn.**maxpool**(*x_sh*)
Compute MaxPool: returns fresh shares of the max value in the input tensor and the index of this value in the flattened tensor

Parameters **x_sh** (AdditiveSharingTensor) – the private tensor on which the op applies

Returns maximum value as an AdditiveSharingTensor index of this value in the flattened tensor as an AdditiveSharingTensor

syft.frameworks.torch.mpc.securenn.**maxpool_deriv**(*x_sh*)
Compute derivative of MaxPool

Parameters **x_sh** (AdditiveSharingTensor) – the private tensor on which the op applies

Returns an AdditiveSharingTensor of the same shape as x_sh full of zeros except for a 1 at the position of the max value

syft.frameworks.torch.mpc.securenn.**maxpool2d**(*a_sh*, *kernel_size: int = 1*, *stride: int = 1*, *padding: int = 0*)
Applies a 2D max pooling over an input signal composed of several input planes. This interface is similar to torch.nn.MaxPool2D. :param kernel_size: the size of the window to take a max over :param stride: the stride of the window :param padding: implicit zero padding to be added on both sides

1.1.1.2.1.55 syft.frameworks.torch.mpc.spdz

1.1.1.2.1.56 Module Contents

syft.frameworks.torch.mpc.spdz.**no_wrap**

syft.frameworks.torch.mpc.spdz.**spdz_mul**(*cmd: Callable*, *x_sh*, *y_sh*, *crypto_provider: AbstractWorker*, *field: int*)
Abstractly multiplies two tensors (mul or matmul)

Parameters

- **cmd** – a callable of the equation to be computed (mul or matmul)
- **x_sh** (AdditiveSharingTensor) – the left part of the operation
- **y_sh** (AdditiveSharingTensor) – the right part of the operation
- **crypto_provider** (AbstractWorker) – an AbstractWorker which is used to generate triples
- **field** (*int*) – an integer denoting the size of the field

Returns an AdditiveSharingTensor

1.1.1.2.1.57 syft.frameworks.torch.nn

1.1.1.2.1.58 Submodules

1.1.1.2.1.59 syft.frameworks.torch.nn.conv

1.1.1.2.1.60 Module Contents

class syft.frameworks.torch.nn.conv.**Conv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=False*, *padding_mode='zeros'*)

Bases: torch.nn.Module

This class is the beginning of an exact python port of the torch.nn.Conv2d module. Because PySyft cannot hook into layers which are implemented in C++, our special functionalities (such as encrypted computation) do not work with torch.nn.Conv2d and so we must have python ports available for all layer types which we seek to use.

Note that this module has been tested to ensure that it outputs the exact output values that the main module outputs in the same order that the main module does.

However, there is often some rounding error of unknown origin, usually less than 1e-6 in magnitude.

This module has not yet been tested with GPUs but should work out of the box.

forward(*self*, *data*)

1.1.1.2.1.61 syft.frameworks.torch.nn.pool

1.1.1.2.1.62 Module Contents

class syft.frameworks.torch.nn.pool.**AvgPool2d**(*kernel_size*, *stride=None*, *padding=0*, *ceil_mode=False*, *count_include_pad=True*, *divisor_override=None*)

Bases: torch.nn.Module

This class is the beginning of an exact python port of the torch.nn.AvgPool2d module. Because PySyft cannot hook into layers which are implemented in C++, our special functionalities (such as encrypted computation) do not work with torch.nn.AvgPool2d and so we must have python ports available for all layer types which we seek to use.

Note that this module has been tested to ensure that it outputs the exact output values that the main module outputs in the same order that the main module does.

However, there is often some rounding error of unknown origin, usually less than 1e-6 in magnitude.

This module has not yet been tested with GPUs but should work out of the box.

forward(*self*, *data*)

1.1.1.2.1.63 syft.frameworks.torch.nn.rnn

1.1.1.2.1.64 Module Contents

class syft.frameworks.torch.nn.rnn.**RNNCellBase**(*input_size*, *hidden_size*, *bias*, *num_chunks*, *nonlinearity=None*)

Bases: torch.nn.Module

Cell to be used as base for all RNN cells, including GRU and LSTM This class overrides the torch.nn.RNNCellBase Only Linear and Dropout layers are used to be able to use MPC

reset_parameters(*self*)

This method initializes or reset all the parameters of the cell. The paramaters are initiated following a uniform distribution.

init_hidden(*self*, *input*)

This method initializes a hidden state when no hidden state is provided in the forward method. It creates a hidden state with zero values.

class syft.frameworks.torch.nn.rnn.**RNNCell**(*input_size*, *hidden_size*, *bias=True*, *nonlinearity='tanh'*)

Bases: *syft.frameworks.torch.nn.rnn.RNNCellBase*

Python implementation of RNNCell with tanh or relu non-linearity for MPC This class overrides the torch.nn.RNNCell

forward(*self*, *x*, *h=None*)

class syft.frameworks.torch.nn.rnn.**GRUCell**(*input_size*, *hidden_size*, *bias=True*, *nonlinearity=None*)

Bases: *syft.frameworks.torch.nn.rnn.RNNCellBase*

Python implementation of GRUCell for MPC This class overrides the torch.nn.GRUCell

forward(*self*, *x*, *h=None*)

class syft.frameworks.torch.nn.rnn.**LSTMCell**(*input_size*, *hidden_size*, *bias=True*, *nonlinearity=None*)

Bases: *syft.frameworks.torch.nn.rnn.RNNCellBase*

Python implementation of LSTMCell for MPC This class overrides the torch.nn.LSTMCell

reset_parameters(*self*)

forward(*self*, *x*, *hc=None*)

class syft.frameworks.torch.nn.rnn.**RNNBase**(*input_size*, *hidden_size*, *num_layers*, *bias*, *batch_first*, *dropout*, *bidirectional*, *base_cell*, *nonlinearity=None*)

Bases: torch.nn.Module

Module to be used as base for all RNN modules, including GRU and LSTM This class overrides the torch.nn.RNNBase Only Linear and Dropout layers are used to be able to use MPC

forward(*self*, *x*, *h=None*)

_swap_axis(*self*, *x*, *h*)

This method swap the axes for batch_size and seq_len. It is used when batch_first==True.

_init_hidden(*self*, *input*)

This method initializes a hidden state when no hidden state is provided in the forward method. It creates a hidden state with zero values for each layer of the network.

_apply_time_step(*self*, *x*, *h*, *c*, *t*, *reverse_direction=False*)
Apply RNN layers at time t, given input and previous hidden states

class syft.frameworks.torch.nn.rnn.**RNN**(*input_size*, *hidden_size*, *num_layers=1*, *nonlinearity='tanh'*, *bias=True*, *batch_first=False*, *dropout=0*, *bidirectional=False*)
Bases: *syft.frameworks.torch.nn.rnn.RNNBase*

Python implementation of RNN for MPC This class overrides the torch.nn.RNN

class syft.frameworks.torch.nn.rnn.**GRU**(*input_size*, *hidden_size*, *num_layers=1*, *bias=True*, *batch_first=False*, *dropout=0*, *bidirectional=False*)
Bases: *syft.frameworks.torch.nn.rnn.RNNBase*

Python implementation of GRU for MPC This class overrides the torch.nn.GRU

class syft.frameworks.torch.nn.rnn.**LSTM**(*input_size*, *hidden_size*, *num_layers=1*, *bias=True*, *batch_first=False*, *dropout=0*, *bidirectional=False*)
Bases: *syft.frameworks.torch.nn.rnn.RNNBase*

Python implementation of LSTM for MPC This class overrides the torch.nn.LSTM

1.1.1.2.1.65 Package Contents

class syft.frameworks.torch.nn.**RNN**(*input_size*, *hidden_size*, *num_layers=1*, *nonlinearity='tanh'*, *bias=True*, *batch_first=False*, *dropout=0*, *bidirectional=False*)
Bases: *syft.frameworks.torch.nn.rnn.RNNBase*

Python implementation of RNN for MPC This class overrides the torch.nn.RNN

class syft.frameworks.torch.nn.**GRU**(*input_size*, *hidden_size*, *num_layers=1*, *bias=True*, *batch_first=False*, *dropout=0*, *bidirectional=False*)
Bases: *syft.frameworks.torch.nn.rnn.RNNBase*

Python implementation of GRU for MPC This class overrides the torch.nn.GRU

class syft.frameworks.torch.nn.**LSTM**(*input_size*, *hidden_size*, *num_layers=1*, *bias=True*, *batch_first=False*, *dropout=0*, *bidirectional=False*)
Bases: *syft.frameworks.torch.nn.rnn.RNNBase*

Python implementation of LSTM for MPC This class overrides the torch.nn.LSTM

class syft.frameworks.torch.nn.**RNNCell**(*input_size*, *hidden_size*, *bias=True*, *nonlinearity='tanh'*)
Bases: *syft.frameworks.torch.nn.rnn.RNNCellBase*

Python implementation of RNNCell with tanh or relu non-linearity for MPC This class overrides the torch.nn.RNNCell

forward(*self*, *x*, *h=None*)

class syft.frameworks.torch.nn.**GRUCell**(*input_size*, *hidden_size*, *bias=True*, *nonlinearity=None*)
Bases: *syft.frameworks.torch.nn.rnn.RNNCellBase*

Python implementation of GRUCell for MPC This class overrides the torch.nn.GRUCell

forward(*self*, *x*, *h=None*)

class syft.frameworks.torch.nn.**LSTMCell**(*input_size*, *hidden_size*, *bias=True*, *nonlinearity=None*)
Bases: *syft.frameworks.torch.nn.rnn.RNNCellBase*

Python implementation of LSTMCell for MPC This class overrides the torch.nn.LSTMCell

reset_parameters(*self*)

forward(*self*, *x*, *hc=None*)

class syft.frameworks.torch.nn.**Conv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=False*, *padding_mode='zeros'*)

Bases: torch.nn.Module

This class is the beginning of an exact python port of the torch.nn.Conv2d module. Because PySyft cannot hook into layers which are implemented in C++, our special functionalities (such as encrypted computation) do not work with torch.nn.Conv2d and so we must have python ports available for all layer types which we seek to use.

Note that this module has been tested to ensure that it outputs the exact output values that the main module outputs in the same order that the main module does.

However, there is often some rounding error of unknown origin, usually less than 1e-6 in magnitude.

This module has not yet been tested with GPUs but should work out of the box.

forward(*self*, *data*)

class syft.frameworks.torch.nn.**AvgPool2d**(*kernel_size*, *stride=None*, *padding=0*, *ceil_mode=False*, *count_include_pad=True*, *divisor_override=None*)

Bases: torch.nn.Module

This class is the beginning of an exact python port of the torch.nn.AvgPool2d module. Because PySyft cannot hook into layers which are implemented in C++, our special functionalities (such as encrypted computation) do not work with torch.nn.AvgPool2d and so we must have python ports available for all layer types which we seek to use.

Note that this module has been tested to ensure that it outputs the exact output values that the main module outputs in the same order that the main module does.

However, there is often some rounding error of unknown origin, usually less than 1e-6 in magnitude.

This module has not yet been tested with GPUs but should work out of the box.

forward(*self*, *data*)

1.1.1.2.1.66 syft.frameworks.torch.tensors

1.1.1.2.1.67 Subpackages

1.1.1.2.1.68 syft.frameworks.torch.tensors.decorators

1.1.1.2.1.69 Submodules

1.1.1.2.1.70 syft.frameworks.torch.tensors.decorators.logging

1.1.1.2.1.71 Module Contents

class syft.frameworks.torch.tensors.decorators.logging.**LoggingTensor**(*owner=None*, *id=None*, *tags=None*, *description=None*)

Bases: *syft.generic.tensor.AbstractTensor*

add(*self*, *_self*, **args*, ***kwargs*)

Here is an example of how to use the @overloaded.method decorator. To see what this decorator do, just look at the next method manual_add: it does exactly the same but without the decorator.

Note the subtlety between self and _self: you should use _self and NOT self.

manual_add(*self*, **args*, ***kwargs*)

Here is the version of the add method without the decorator: as you can see it is much more complicated. However you might need sometimes to specify some particular behaviour: so here what to start from :)

static torch(*module*)

We use the @overloaded.module to specify we're writing here a function which should overload the function with the same name in the <torch> module :param module: object which stores the overloading functions

Note that we used the @staticmethod decorator as we're in a class

classmethod on_function_call(*cls*, *command*)

Override this to perform a specific action for each call of a torch function with arguments containing syft tensors of the class doing the overloading

static simplify(*worker: AbstractWorker*, *tensor: LoggingTensor*)

This function takes the attributes of a LogTensor and saves them in a tuple :param tensor: a LogTensor :type tensor: LoggingTensor

Returns a tuple holding the unique attributes of the log tensor

Return type tuple

Examples

data = _simplify(tensor)

static detail(*worker: AbstractWorker*, *tensor_tuple: tuple*)
This function reconstructs a LogTensor given it's attributes in form of a tuple. :param worker: the worker doing the deserialization :param tensor_tuple: a tuple holding the attributes of the LogTensor

Returns a LogTensor

Return type *LoggingTensor*

Examples

logtensor = detail(data)

1.1.1.2.1.72 syft.frameworks.torch.tensors.interpreters

1.1.1.2.1.73 Submodules

1.1.1.2.1.74 syft.frameworks.torch.tensors.interpreters.additive_shared

1.1.1.2.1.75 Module Contents

syft.frameworks.torch.tensors.interpreters.additive_shared.**no_wrap**

class syft.frameworks.torch.tensors.interpreters.additive_shared.**AdditiveSharingTensor**(*shared_dict = None owner id=N field= n_bit crypt tags= de- scrip tion=*

Bases: *syft.generic.tensor.AbstractTensor*

__add__

__radd__

__sub__

__pow__

__truediv__

__repr__(*self*)

__str__(*self*)

__bool__(*self*)
Prevent evaluation of encrypted tensor

property locations(*self*)
Provide a locations attribute

property shape(*self*)
Return the shape which is the shape of any of the shares

dim(*self*)

clone(*self*)
Clone should keep ids unchanged, contrary to copy

get_class_attributes(*self*)
Specify all the attributes need to build a wrapper correctly when returning a response, for example precision_fractional is important when wrapping the result of a method on a self which is a fixed precision tensor with a non default precision_fractional.

property grad(*self*)
Gradient makes no sense for Additive Shared Tensor, so we make it clear that if someone query .grad on a Additive Shared Tensor it doesn't error but returns grad and can't be set

backward(*self*, **args*, ***kwargs*)
Calling backward on Additive Shared Tensor doesn't make sense, but sometimes a call can be propagated downward the chain to an AST (for example in create_grad_objects), so we just ignore the call.

get(*self*)
Fetches all shares and returns the plaintext tensor they represent

virtual_get(*self*)
Get the value of the tensor without calling get - Useful for debugging, only for VirtualWorkers

init_shares(*self*, **owners*)
Initializes shares and distributes them amongst their respective owners

Parameters **the list of shareholders. Can be of any length.**
(**owners*) –

static generate_shares(*secret*, *n_workers*, *field*, *random_type*)
The cryptographic method for generating shares given a secret tensor.

Parameters

- **secret** – the tensor to be shared.
- **n_workers** – the number of shares to generate for each value (i.e., the number of tensors to return)
- **field** – 1 + the max value for a share
- **random_type** – the torch type shares should be encoded in (use the smallest possible given the choise of mod"

reconstruct(*self*)
Reconstruct the shares of the AdditiveSharingTensor remotely without its owner being able to see any sensitive value

Returns A MultiPointerTensor where all workers hold the reconstructed value

zero(*self*)
Build an additive shared tensor of value zero with the same properties as self

refresh(*self*)
Refresh shares by adding shares of zero

_getitem_multipointer(*self*, *self_shares*, *indices_shares*)
Support x[i] where x is an AdditiveSharingTensor and i a MultiPointerTensor

Parameters

- **self_shares** (*dict*) – the dict of shares of x
- **indices_shares** (*dict*) – the dict of shares of i

Returns an AdditiveSharingTensor

_getitem_public(*self*, *self_shares*, *indices*)
Support x[i] where x is an AdditiveSharingTensor and i a MultiPointerTensor

Parameters

- **self_shares** (*dict*) – the dict of shares of x
- **indices_shares** (*tuples of ints*) – integers indices

Returns an AdditiveSharingTensor

__getitem__(*self*, *indices*)

add(*self*, *shares: dict*, *other*)
Adds operand to the self AST instance.

Parameters

- **shares** – a dictionary <location_id -> PointerTensor) of shares corresponding to self. Equivalent to calling self.child.
- **other** – the operand being added to self, can be: - a dictionary <location_id -> PointerTensor) of shares - a torch tensor - a constant

sub(*self*, *shares: dict*, *other*)
Subtracts an operand from the self AST instance.

Parameters

- **shares** – a dictionary <location_id -> PointerTensor) of shares corresponding to self. Equivalent to calling self.child.
- **other** – the operand being subtracted from self, can be: - a dictionary <location_id -> PointerTensor) of shares - a torch tensor - a constant

__rsub__(*self*, *other*)

_private_mul(*self*, *other*, *equation: str*)
Abstractly Multiplies two tensors

Parameters

- **self** – an AdditiveSharingTensor
- **other** – another AdditiveSharingTensor
- **equation** – a string representation of the equation to be computed in einstein summation form

_public_mul(*self*, *shares*, *other*, *equation*)
Multiplies an AdditiveSharingTensor with a non-private value (int, torch tensor, MultiPointerTensor, etc.)

When other is a constant equal to zero, the shares vanish so we need to add fresh shares of zero.

Parameters

- **shares** (*dict*) – a dictionary <location_id -> PointerTensor) of shares corresponding to self. Equivalent to calling self.child.
- **other** (*dict of int*) – operand being multiplied with self, can be: - a dictionary <location_id -> PointerTensor) of shares - a torch tensor (Int or Long) - or an integer
- **equation** – a string representation of the equation to be computed in einstein summation form

mul (*self*, *other*)
Multiplies two tensors together

Parameters

- **self** (AdditiveSharingTensor) – an AdditiveSharingTensor
- **other** – another AdditiveSharingTensor, or a MultiPointerTensor, or an integer

__mul__ (*self*, *other*, ***kwargs*)

__imul__ (*self*, *other*)

pow (*self*, *power*)
Compute integer power of a number by recursion using mul

This uses the following trick:

- Divide power by 2 and multiply base to itself (if the power is even)
- Decrement power by 1 to make it even and then follow the first step

matmul (*self*, *other*)
Multiplies two tensors matrices together

Parameters

- **self** – an AdditiveSharingTensor
- **other** – another AdditiveSharingTensor or a MultiPointerTensor

mm (*self*, **args*, ***kwargs*)
Multiplies two tensors matrices together

__matmul__ (*self*, **args*, ***kwargs*)
Multiplies two tensors matrices together

__itruediv__ (*self*, **args*, ***kwargs*)

_private_div (*self*, *divisor*)

_public_div (*self*, *shares: dict*, *divisor*)

div (*self*, *divisor*)

mod (*self*, *shares: dict*, *modulus: int*)

__mod__ (*self*, **args*, ***kwargs*)

chunk (*self*, *shares*, **args*, ***kwargs*)
This method overrides the torch.Tensor.chunk() method of Pytorch

static torch (*module*)

relu (*self*)

positive (*self*)

gt (*self*, *other*)

__gt__(*self*, *other*)

ge(*self*, *other*)

__ge__(*self*, *other*)

lt(*self*, *other*)

__lt__(*self*, *other*)

le(*self*, *other*)

__le__(*self*, *other*)

eq(*self*, *other*)

__eq__(*self*, *other*)

max(*self*, *dim=None*, *return_idx=False*)
Return the maximum value of an additive shared tensor

Parameters

- **dim** (*None or int*) – if not None, the dimension on which the comparison should be done
- **return_idx** (*bool*) – Return the index of the maximum value Note that if dim is specified then the index is returned anyway to match the Pytorch syntax.

Returns the maximum value (possibly across an axis) and optionally the index of the maximum value (possibly across an axis)

argmax(*self*, *dim=None*)

static select_worker(*args*, *worker*)
utility function for handle_func_command which help to select shares (seen as elements of dict) in an argument set. It could perhaps be put elsewhere

Parameters

- **args** – arguments to give to a functions
- **worker** – owner of the shares to select

Returns args where the AdditiveSharedTensors are replaced by the appropriate share

classmethod handle_func_command(*cls*, *command*)
Receive an instruction for a function to be applied on a Syft Tensor, Replace in the args all the LogTensors with their child attribute, forward the command instruction to the handle_function_command of the type of the child attributes, get the response and replace a Syft Tensor on top of all tensors found in the response.

Parameters

- **command** – instruction of a function command: (command name,
- **self>, arguments[, kwargs])** (*<no*) –

Returns the response of the function command

set_garbage_collect_data(*self*, *value*)

get_garbage_collect_data(*self*)

static simplify(*worker: AbstractWorker*, *tensor: AdditiveSharingTensor*)
This function takes the attributes of a AdditiveSharingTensor and saves them in a tuple :param tensor: a AdditiveSharingTensor :type tensor: AdditiveSharingTensor

Returns a tuple holding the unique attributes of the additive shared tensor

Return type tuple

Examples

data = simplify(tensor)

static detail(*worker: AbstractWorker*, *tensor_tuple: tuple*)
This function reconstructs a AdditiveSharingTensor given it's attributes in form of a tuple. :param worker: the worker doing the deserialization :param tensor_tuple: a tuple holding the attributes of the AdditiveSharingTensor

Returns a AdditiveSharingTensor

Return type *AdditiveSharingTensor*

Examples

shared_tensor = detail(data)

static bufferize(*worker: AbstractWorker*, *tensor: AdditiveSharingTensor*)
This function takes the attributes of a AdditiveSharingTensor and saves them in a protobuf object :param tensor: a AdditiveSharingTensor :type tensor: AdditiveSharingTensor

Returns a protobuf object holding the unique attributes of the additive shared tensor

Return type protobuf

Examples

data = protobuf(tensor)

static unbufferize(*worker: AbstractWorker*, *protobuf_tensor: AdditiveSharingTensorPB*)
This function reconstructs a AdditiveSharingTensor given its' attributes in form of a protobuf object. :param worker: the worker doing the deserialization :param protobuf_tensor: a protobuf object holding the attributes of the AdditiveSharingTensor

Returns a AdditiveSharingTensor

Return type *AdditiveSharingTensor*

Examples

shared_tensor = unprotobuf(data)

1.1.1.2.1.76 syft.frameworks.torch.tensors.interpreters.autograd

1.1.1.2.1.77 Module Contents

syft.frameworks.torch.tensors.interpreters.autograd.**backwards_grad**(*grad_fn*, *in_grad=None*)

class syft.frameworks.torch.tensors.interpreters.autograd.**AutogradTensor**(*data=None*, *requires_grad=True*, *owner=None*, *id=None*, *preinitialize_grad=False*, ***kwargs*)

Bases: *syft.generic.tensor.AbstractTensor*

A tensor that tracks operations to build a dynamic graph and backprops through the graph to calculate gradients.

backward(*self*, *grad=None*)

property data(*self*)

property grad(*self*)

attr(*self*, *attr_name*)

__add__(*self*, *other*)

__iadd__(*self*, *other*)

__sub__(*self*, *other*)

__isub__(*self*, *other*)

__mul__(*self*, *other*)

__matmul__(*self*, *other*)

__pow__(*self*, *power*, ***kwargs*)

__truediv__(*self*, *other*)

__gt__(*self*, *_self*, *other*)

__ge__(*self*, *_self*, *other*)

__lt__(*self*, *_self*, *other*)

__le__(*self*, *_self*, *other*)

eq(*self*, *_self*, *other*)

relu(*self*, *self_*)

__getattribute__(*self*, *name*)

static torch(*module*)

classmethod handle_func_command(*cls*, *command*)

Receive an instruction for a function to be applied on a AutogradTensor, Perform some specific action (like logging) which depends of the instruction content, replace in the args all the LogTensors with their child attribute, forward the command instruction to the handle_function_command of the type of the child attributes, get the response and replace a AutogradTensor on top of all tensors found in the response. :param command: instruction of a function command: (command name, <no self>, arguments[, kwargs]) :return: the response of the function command

get(*self*)

Just a pass through. This is most commonly used when calling .get() on a AutogradTensor which has also been shared.

float_precision(*self*)
Just a pass through. This is most commonly used when calling .float_precision() on a AutogradTensor which has also been shared.

static simplify(*worker: AbstractWorker*, *tensor: AutogradTensor*)

Takes the attributes of an AutogradTensor and saves them in a tuple. Or simply said, it serializes an AutogradTensor

Parameters **tensor** – an AutogradTensor.

Returns a tuple holding the unique attributes of the AutogradTensor.

Return type tuple

static detail(*worker: AbstractWorker*, *tensor_tuple: tuple*)
This function reconstructs (deserializes) an AutogradTensors given its attributes in form of a tuple. :param worker: the worker doing the deserialization :param tensor_tuple: a tuple holding the attributes of the AutogradTensor

Returns an AutogradTensor

Return type *AutogradTensor*

Examples

shared_tensor = detail(data)

1.1.1.2.1.78 syft.frameworks.torch.tensors.interpreters.build_gradients

1.1.1.2.1.79 Module Contents

syft.frameworks.torch.tensors.interpreters.build_gradients.**tab**

syft.frameworks.torch.tensors.interpreters.build_gradients.**split_signature**(*signature*)

syft.frameworks.torch.tensors.interpreters.build_gradients.**construct_grad_fn_class**(*grad_def*)

syft.frameworks.torch.tensors.interpreters.build_gradients.**derivatives**

1.1.1.2.1.80 syft.frameworks.torch.tensors.interpreters.crt_precision

1.1.1.2.1.81 Module Contents

class syft.frameworks.torch.tensors.interpreters.crt_precision.**CRTPrecisionTensor**(*residues: dict = None, base=None, precision_fraction re-construction_coeffs= owner=None id=None, tags: set = None, description: str = None*)

Bases: *syft.generic.tensor.AbstractTensor*

A CRT tensor is a tensor whose values are represented as their remainders modulo several pairwise coprime numbers. The true tensor values lie in the Zq field where q is the product of all the moduli. A CRT tensor then represent a modular system of equations:

x = a_0 mod f_0 ... x = a_n mod f_n

The Chinese Remainder Theorem (this is where CRT in CRTPrecisionTensor comes from) asserts that exactly one x in Zq with q = f_0 * ... * f_n satisfies this system. This x is the real value represented by the tensor. This tensor makes it possible to represent big numbers and to avoid overflows while manipulating them. It also makes additions, subtractions, and multiplications of huge numbers quite efficient.

__eq__

__add__

__radd__

__sub__

__mul__

__rmul__

__truediv__

float_precision(*self*)

property grad(*self*)

Gradient makes no sense for CRT Tensor, so we make it clear that if someone query .grad on a CRT Tensor

it doesn't error but returns grad and can't be set

__str__(*self*)

__repr__(*self*)

property shape(*self*)

eq(*self*, *self_*, *other_*)

__neg__(*self*)

add(*self*, *self_*, *other*)

sub(*self*, *self_*, *other*)

__rsub__(*self*, *other*)

mul(*self*, *self_*, *other*)

abstract div(*self*, *other*)

reconstruct(*self*)
Build the tensor in Zq with q = prod(self.child.keys()) satisfying the modular system represented by the tensor. The algorithm consists in: 1) Compute N = prod(self.child.keys()) 2) yi = N / ni where the ni's are moduli in the system 3) zi = yi^(-1) mod ni (we know zi exists because all the original moduli are pairwise coprime) 4) The result is x = sum(ai * yi * zi) where ai is the residue modulo ni in the system

We can see that this boils down to a linear combination of ai's with coefficients yi * zi. These coefficients depend only on the moduli used to represent the tensor so we don't need to compute them several times for the same tensor. We can also reuse them for a tensor produced by operations on self because 1) the operations can only be done between tensors with the same moduli and 2) the output tensor will have the same moduli. This is why we store them after having computed them once.

static torch(*module*)

share(*self*, **owners*, *field=None*, *crypto_provider=None*)
Share the tensor between several workers. This gives an AdditiveSharingTensor wrapped around the original CRT tensor

get(*self*)
Get back a tensor shared between several workers.

static simplify(*worker: AbstractWorker*, *tensor: CRTPrecisionTensor*)
This function takes the attributes of a CRTPrecisionTensor and saves them in a tuple :param worker: the worker doing the serialization :param tensor: a CRTPrecisionTensor

Returns a tuple holding the unique attributes of the tensor

Return type tuple

static detail(*worker: AbstractWorker*, *tensor_tuple: tuple*)
This function reconstructs a CRTPrecisionTensor given its attributes in form of a tuple. :param worker: the worker doing the deserialization :param tensor_tuple: a tuple holding the attributes of the CRTPrecisionTensor

Returns a CRTPrecisionTensor

Return type *CRTPrecisionTensor*

get_class_attributes(*self*)

syft.frameworks.torch.tensors.interpreters.crt_precision.**_moduli_for_fields**

syft.frameworks.torch.tensors.interpreters.crt_precision.**_sizes_for_fields**

1.1.1.2.1.82 syft.frameworks.torch.tensors.interpreters.gradients

1.1.1.2.1.83 Module Contents

class syft.frameworks.torch.tensors.interpreters.gradients.**AddBackward**(*self_*, *other*)
Bases: *syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc*

gradient(*self*, *grad*)

class syft.frameworks.torch.tensors.interpreters.gradients.**SubBackward**(*self_*, *other*)
Bases: *syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc*

gradient(*self*, *grad*)

class syft.frameworks.torch.tensors.interpreters.gradients.**SumBackward**(*self_*)
Bases: *syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc*

gradient(*self*, *grad*)

class syft.frameworks.torch.tensors.interpreters.gradients.**AsinBackward**(*self_*)
Bases: *syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc*

gradient(*self*, *grad*)

class syft.frameworks.torch.tensors.interpreters.gradients.**MulBackward**(*self_*, *other*)
Bases: *syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc*

gradient(*self*, *grad*)

class syft.frameworks.torch.tensors.interpreters.gradients.**DivBackward**(*self_*, *other*)
Bases: *syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc*

gradient(*self*, *grad*)

class syft.frameworks.torch.tensors.interpreters.gradients.**PowBackward**(*self_*, *power*)
Bases: *syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc*

gradient(*self*, *grad*)

class syft.frameworks.torch.tensors.interpreters.gradients.**MatmulBackward**(*self_*, *other*)
Bases: *syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc*

gradient(*self*, *grad*)

class syft.frameworks.torch.tensors.interpreters.gradients.**TBackward**(*self_*)
Bases: *syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc*

gradient(*self*, *grad*)

class syft.frameworks.torch.tensors.interpreters.gradients.**SigmoidBackward**(*self_*)
Bases: *syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc*

gradient(*self*, *grad*)

class syft.frameworks.torch.tensors.interpreters.gradients.**SinBackward**(*self_*)
Bases: *syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc*

gradient(*self*, *grad*)

class syft.frameworks.torch.tensors.interpreters.gradients.**SinhBackward**(*self_*)
Bases: *syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc*

gradient(*self*, *grad*)

class syft.frameworks.torch.tensors.interpreters.gradients.**SqrtBackward**(*self_*)
Bases: *syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc*

gradient(*self*, *grad*)

class syft.frameworks.torch.tensors.interpreters.gradients.**TanhBackward**(*self_*)
Bases: *syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc*

gradient(*self*, *grad*)

class syft.frameworks.torch.tensors.interpreters.gradients.**ReluBackward**(*self_*)
Bases: *syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc*

gradient(*self*, *grad*)

1.1.1.2.1.84 syft.frameworks.torch.tensors.interpreters.gradients_core

1.1.1.2.1.85 Module Contents

class syft.frameworks.torch.tensors.interpreters.gradients_core.**GradFunc**(**args*)

abstract gradient(*self*, *grad*)

__call__(*self*, *grad*)

__repr__(*self*)

__setattr__(*self*, *name*, *value*)

static simplify(*worker: AbstractWorker*, *grad_fn*)

Takes the attributes of a grad_fn object and saves them in a tuple Every gradient function class that extends *GradFunc* uses this function to simplify the attributes.

Parameters **grad_fn** – gradient function object (AddBackward, SubBackward, etc.)

Returns a tuple containing all the simplified attributes of gradient function

Return type grad_fn_attrs

static detail(*worker: AbstractWorker*, *gradfn_tuple*)

This function reconstructs (deserializes) the gradient function object, given its attributes in the form of a tuple

Args: gradfn_tuple: a tuple containing all the simplified attributes of a grad function (along with the class name at begining)

Returns A correct gradient function object

syft.frameworks.torch.tensors.interpreters.gradients_core.**apply_dim_transformations**(*grad_self*, *grad_other*, *self_shape*, *other_shape*)

Given computed gradients and initial shapes, reshape the gradients to match the initial shapes by reverse engineering the expansion operations made by PyTorch when operating two tensors with different shapes.

Parameters

- **grad_self** – computed gradient for self
- **grad_other** – computed gradient for other
- **self_shape** – initial shape for self
- **other_shape** – initial shape for other

Returns grad_self, grad_other with the proper shape

1.1.1.2.1.86 syft.frameworks.torch.tensors.interpreters.hook

1.1.1.2.1.87 Module Contents

class syft.frameworks.torch.tensors.interpreters.hook.**HookedTensor**(*owner=None*, *id=None*, *tags=None*, *description=None*, *verbose=False*)

Bases: *syft.generic.tensor.AbstractTensor*

HookedTensor is an abstraction which should not be used directly on its own. Its purpose is only to allow other tensors to extend it so that they automatically have all of the Torch method hooked without having to add it to the hook.py file.

1.1.1.2.1.88 syft.frameworks.torch.tensors.interpreters.large_precision

1.1.1.2.1.89 Module Contents

class syft.frameworks.torch.tensors.interpreters.large_precision.**LargePrecisionTensor**(*owner id=No tags=N description=N field: int = 2**51 base: int = 10, precision_fi internal_typ verbose=*

Bases: *syft.generic.tensor.AbstractTensor*

LargePrecisionTensor allows handling of numbers bigger than LongTensor

Some systems using Syft require larger types than those supported natively. This tensor type supports arbitrarily large values by packing them in smaller ones. Typically a user will require to enlarge a float number by fixing its precision

tensor.fix_prec()

The large value is defined by *precision_fractional*.

The smaller values are of type *internal_type*. The split of the large number into the smaller values is in the range ±2**(size - 1).

By default operations are done with NumPy. This implies unpacking the representation and packing it again.

Sharing a LPT requires using a arithmetic field where the shares will live. This field cannot bigger than 2 ** 62 or the process would trigger a RuntimeError: Overflow when unpacking long. Note that this field will be applied to the internal representation and not to the scaled tensor.

Check the tests to see how to play with the different parameters.

__add__

add_

__sub__

sub_

__mul__

mul_

__mod__

__gt__

__lt__

_create_internal_representation(*self*)
Decompose a tensor into an array of numbers that represent such tensor with the required precision

static _expand_item(*a_number*, *max_length*)

property internal_precision(*self*)
"The internal precision used to decompose the large numbers is the size of the type - 1. The large number is decomposed into positive smaller numbers. This could provoke overflow if any of these smaller parts are bigger than type_precision/2.

get_class_attributes(*self*)
Specify all the attributes need to build a wrapper correctly when returning a response.

add(*self*, *self_*, *other*)

__iadd__(*self*, *other*)
Add two fixed precision tensors together.

sub(*self*, *self_*, *other*)

__isub__(*self*, *other*)
Add two fixed precision tensors together.

mul(*self*, *self_*, *other*)

__imul__(*self*, *other*)

mod(*self*, *self_*, *other*)

gt(*self*, *self_*, *other*)

lt(*self*, *self_*, *other*)

fix_large_precision(*self*)

float_precision(*self*)
Restore the tensor from the internal representation.

Returns the original tensor.

Return type tensor

static create_tensor_from_numpy(*ndarray*, ***kwargs*)
Decompose a NumPy array into an array of numbers that represent such tensor with the required precision.

Typically this method is called on the result of an operation.

static _split_numbers(*numbers*, *bits*, *internal_type*)
Splits a tensor of numbers in numbers of a smaller power.

Parameters

- **numbers** (*array*) – the tensor to split.
- **bits** (*int*) – the bits to use in the split.

Returns a tensor with one more dimension representing the original one.

Return type array

_internal_representation_to_large_ints(*self*)
Creates an numpy array containing the objective large numbers.

static _restore_large_number(*number_parts*, *bits*)
Rebuilds a number from a numpy array.

Parameters

- **number_parts** (*ndarray*) – the numpy array of numbers representing the original one.
- **bits** (*int*) – the bits used in the split.

Returns the large number represented by this tensor

Return type Number

static _forward_func(*tensor*)

static _backward_func(*tensor*, ***kwargs*)

share(*self*, **owners*, *field=None*, *crypto_provider=None*)

syft.frameworks.torch.tensors.interpreters.large_precision.**type_precision**

1.1.1.2.1.90 syft.frameworks.torch.tensors.interpreters.native

1.1.1.2.1.91 Module Contents

syft.frameworks.torch.tensors.interpreters.native.**_get_maximum_precision**()
This function returns the maximum value allowed for precision fractions before the chain decides to use LPT.

This function can be overridden if the setup requires the use of LargePrecisionTensor from a smaller precision.

The default value is the size of torch.long

Returns The maximum value for precision allowed in this setup

syft.frameworks.torch.tensors.interpreters.native.**default_pytorch_maximum_precision**()
Dealing with integers > 2**62-1 is not fun with precision tensors.

class syft.frameworks.torch.tensors.interpreters.native.**TorchTensor**
Bases: *syft.generic.tensor.AbstractTensor*

Add methods to this tensor to have them added to every torch.Tensor object.

This tensor is simply a more convenient way to add custom functions to all Torch tensor types. When you add a function to this tensor, it will be added to EVERY native torch tensor type (i.e. torch.Torch) automatically by the TorchHook (which is in frameworks/torch/hook.py).

Note: all methods from AbstractTensor will also be included because this tensor extends AbstractTensor. So, if you're looking for a method on the native torch tensor API but it's not listed here, you might try checking AbstractTensor.

float_precision

float_precision_

fix_precision

fix_precision_

has_child(*self*)

property tags(*self*)

property description(*self*)

property shape(*self*)

property data(*self*)

property grad(*self*)

__str__(*self*)

__repr__(*self*)

__eq__(*self*, *other*)

property id(*self*)

property gc(*self*)

property disable_gc(*self*)

property garbage_collection(*self*)

_is_parameter(*self*)
Utility method to test if the tensor is in fact a Parameter

static torch(*module*)

classmethod handle_func_command(*cls*, *command*)
Operates as a router for functions. A function call always starts by being handled here and 3 scenarii must be considered:

Real Torch tensor: The arguments of the function are real tensors so we should run the native torch command

Torch wrapper: The arguments are just wrappers at the top of a chain (ex: wrapper>LoggingTensor>Torch tensor), so just forward the instruction to the next layer type in the chain (in the example above to LoggingTensor.handle_func_command), get the response and replace a wrapper on top of all tensors found in the response.

Syft Tensor: The arguments are syft tensors of same type: this can happen if at any node of the chain where some function is forwarded, the handle_func_command modify the function and make a new call but keeps the arguments "un-wrapped". Making a new call means that by default the command is treated here in the global router.

Parameters **command** – instruction of a function command: (command name,

<no self>, arguments[, kwargs]) :return: the response of the function command

_get_response(*cmd*, *args*, *kwargs*)
Return the evaluation of the cmd string parameter

_fix_torch_library(*cmd*)
Change the cmd string parameter to use nn.functional path to avoid erros.

send(*self*, **location*, *inplace: bool = False*, *user=None*, *local_autograd=False*, *preinitialize_grad=False*, *no_wrap=False*, *garbage_collect_data=True*)
Gets the pointer to a new remote object.

One of the most commonly used methods in PySyft, this method serializes the object upon which it is called (self), sends the object to a remote worker, creates a pointer to that worker, and then returns that pointer from this function.

Parameters

- **location** – The BaseWorker object which you want to send this object to. Note that this is never actually the BaseWorker but instead a class which instantiates the BaseWorker abstraction.

- **inplace** – if true, return the same object instance, else a new wrapper
- **local_autograd** – Use autograd system on the local machine instead of PyTorch's autograd on the workers.
- **preinitialize_grad** – Initialize gradient for AutogradTensors to a tensor
- **no_wrap** – If True, wrap() is called on the created pointer
- **garbage_collect_data** – argument passed down to create_pointer()

Returns A torch.Tensor[PointerTensor] pointer to self. Note that this object will likely be wrapped by a torch.Tensor wrapper.

Raises ***SendNotPermittedError*** – Raised if send is not permitted on this tensor.

send_(*self*, **location*, ***kwargs*)
Calls send() with inplace option, but only with a single location :param location: workers locations :return:

create_pointer(*self*, *location: BaseWorker = None*, *id_at_location: str or int = None*, *register: bool = False*, *owner: BaseWorker = None*, *ptr_id: str or int = None*, *garbage_collect_data: bool = True*, *shape=None*, ***kwargs*)
Creates a pointer to the "self" torch.Tensor object.

Returns A PointerTensor pointer to self. Note that this object will likely be wrapped by a torch.Tensor wrapper.

mid_get(*self*)
This method calls .get() on a child pointer and correctly registers the results

remote_get(*self*)
Assuming .child is a PointerTensor, this method calls .get() on the tensor that the .child is pointing to (which should also be a PointerTensor)

TODO: make this kind of message forwarding generic?

get(*self*, **args*, *inplace: bool = False*, *user=None*, *reason: str = ''*, ***kwargs*)
Requests the tensor/chain being pointed to, be serialized and return :param args: args to forward to worker :param inplace: if true, return the same object instance, else a new wrapper :param kwargs: kwargs to forward to worker

Raises ***GetNotPermittedError*** – Raised if get is not permitted on this tensor

get_(*self*, **args*, ***kwargs*)
Calls get() with inplace option set to True

allow(*self*, *user=None*)
This function returns will return True if it isn't a PrivateTensor, otherwise it will return the result of PrivateTensor's allow method.

Parameters **user** (*object, optional*) – User crendentials to be verified.

Returns If it is a public tensor/ allowed user, returns true, otherwise it returns false.

Return type boolean

move(*self*, *location*)

remote_send(*self*, *location*, *change_location=False*)

attr(*self*, *attr_name*)

clone(*self*, **args*, ***kwargs*)
Clone should keep ids unchanged, contrary to copy

float_prec(*self*)

float_prec_(*self*)

private_tensor(*self*, **args*, *allowed_users: Union[str] = []*, *no_wrap: bool = False*, ***kwargs*)
Convert a tensor or syft tensor to private tensor

Parameters

- ***args** (*tuple*) – args to transmit to the private tensor.
- **allowed_users** (*Union*) – Tuple of allowed users.
- **no_wrap** (*bool*) – if True, we don't add a wrapper on top of the private tensor
- ****kwargs** (*dict*) – kwargs to transmit to the private tensor

fix_prec(*self*, **args*, *storage='auto'*, *field_type='int100'*, *no_wrap: bool = False*, ***kwargs*)
Convert a tensor or syft tensor to fixed precision

Parameters

- ***args** (*tuple*) – args to transmit to the fixed precision tensor
- **storage** (*str*) – code to define the type of fixed precision tensor (values in (auto, crt, large))
- **field_type** (*str*) – code to define a storage type (only for CRTPrecisionTensor)
- **no_wrap** (*bool*) – if True, we don't add a wrapper on top of the fixed precision tensor
- ****kwargs** (*dict*) – kwargs to transmit to the fixed precision tensor

fix_prec_(*self*, **args*, ***kwargs*)
Performs an inplace transformation to fixed precision and change self to be a wrapper

Parameters

- ***args** – args to transmit to fix_prec
- ****kwargs** – kwargs to transmit to fix_prec

Returns self seen as a wrapper

_requires_large_precision(*self*, *max_precision*, *base*, *precision_fractional*)
Check if any of the elements in the tensor would require large precision.

share(*self, *owners: List[BaseWorker], field: Union[int, None] = None, crypto_provider: Union[BaseWorker, None] = None, requires_grad: bool = False, no_wrap: bool = False*)
This is a pass through method which calls .share on the child.

Parameters

- **owners** (*list*) – A list of BaseWorker objects determining who to send shares to.
- **field** (*int or None*) – The arithmetic field where live the shares.
- **crypto_provider** (BaseWorker *or None*) – The worker providing the crypto primitives.
- **requires_grad** (*bool*) – Should we add AutogradTensor to allow gradient computation, default is False.

share_(*self*, **args*, ***kwargs*)
Allows to call .share() as an inplace operation

combine(*self*, **pointers*)
This method will combine the child pointer with another list of pointers

Parameters **a list of pointers to be combined into a MultiPointerTensor** (**pointers*) –

keep (*self*, *obj*)
Call .keep() on self's child if the child is a Promise (otherwise an error is raised). .keep() is used to fulfill a promise with a value.

value (*self*)
Call .value() on self's child if the child is a Promise (otherwise an error is raised). .value() is used to retrieve the oldest unused value the promise was kept with.

torch_type (*self*)

encrypt (*self*, *public_key*)
This method will encrypt each value in the tensor using Paillier homomorphic encryption.

Parameters **a public key created using** (**public_key*) – syft.frameworks.torch.he.paillier.keygen()

decrypt (*self*, *private_key*)
This method will decrypt each value in the tensor, returning a normal torch tensor.

Parameters **a private key created using** (**private_key*) – syft.frameworks.torch.he.paillier.keygen()

numpy_tensor (*self*)
This method will cast the current tensor to one with numpy as the underlying representation. The tensor chain will be Wrapper > NumpyTensor > np.ndarray

1.1.1.2.1.92 syft.frameworks.torch.tensors.interpreters.numpy

1.1.1.2.1.93 Module Contents

class syft.frameworks.torch.tensors.interpreters.numpy.**NumpyTensor**(*numpy_tensor=None*, *owner=None*, *id=None*, *tags=None*, *description=None*, *verbose=False*)

Bases: *syft.frameworks.torch.tensors.interpreters.hook.HookedTensor*

NumpyTensor is a tensor which seeks to wrap the Numpy API with the PyTorch tensor API. This is useful because Numpy can offer a wide range of existing functionality ranging from large precision, custom scalar types, and polynomial arithmetic.

mm (*self*, *_self*, *other*)

transpose (*self*, *_self*, **dims*)

syft.frameworks.torch.tensors.interpreters.numpy.**create_numpy_tensor**(*numpy_tensor*)

1.1.1.2.1.94 syft.frameworks.torch.tensors.interpreters.paillier

1.1.1.2.1.95 Module Contents

class syft.frameworks.torch.tensors.interpreters.paillier.**PaillierTensor**(*owner=None*, *id=None*, *tags=None*, *description=None*)

Bases: *syft.generic.tensor.AbstractTensor*

encrypt(*self*, *public_key*)
This method will encrypt each value in the tensor using Paillier homomorphic encryption.

Parameters **a public key created using** (**public_key*) – syft.frameworks.torch.he.paillier.keygen()

encrypt_(*self*, *public_key*)
This method will encrypt each value in the tensor using Paillier homomorphic encryption.

Parameters **a public key created using** (**public_key*) – syft.frameworks.torch.he.paillier.keygen()

decrypt(*self*, *private_key*)
This method will decrypt each value in the tensor, returning a normal torch tensor.

=Args:

***private_key** **a private key created using** syft.frameworks.torch.he.paillier.keygen()

__add__(*self*, **args*, ***kwargs*)
Here is the version of the add method without the decorator: as you can see it is much more complicated. However you misght need sometimes to specify some particular behaviour: so here what to start from :)

__sub__(*self*, **args*, ***kwargs*)
Here is the version of the add method without the decorator: as you can see it is much more complicated. However you misght need sometimes to specify some particular behaviour: so here what to start from :)

__mul__(*self*, **args*, ***kwargs*)
Here is the version of the add method without the decorator: as you can see it is much more complicated. However you misght need sometimes to specify some particular behaviour: so here what to start from :)

mm(*self*, **args*, ***kwargs*)
Here is matrix multiplication between an encrypted and unencrypted tensor. Note that we cannot matrix multiply two encrypted tensors because Paillier does not support the multiplication of two encrypted values.

add(*self*, *_self*, **args*, ***kwargs*)
Here is an example of how to use the @overloaded.method decorator. To see what this decorator do, just look at the next method manual_add: it does exactly the same but without the decorator.

Note the subtlety between self and _self: you should use _self and NOT self.

sub(*self*, *_self*, **args*, ***kwargs*)
Here is an example of how to use the @overloaded.method decorator. To see what this decorator do, just look at the next method manual_add: it does exactly the same but without the decorator.

Note the subtlety between self and _self: you should use _self and NOT self.

mul (*self*, *_self*, **args*, ***kwargs*)
Here is an example of how to use the @overloaded.method decorator. To see what this decorator do, just look at the next method manual_add: it does exactly the same but without the decorator.

Note the subtlety between self and _self: you should use _self and NOT self.

static torch (*module*)
We use the @overloaded.module to specify we're writing here a function which should overload the function with the same name in the <torch> module :param module: object which stores the overloading functions

Note that we used the @staticmethod decorator as we're in a class

static simplify (*worker: AbstractWorker*, *tensor: PaillierTensor*)
This function takes the attributes of a LogTensor and saves them in a tuple :param tensor: a LogTensor :type tensor: PaillierTensor

Returns a tuple holding the unique attributes of the log tensor

Return type tuple

Examples

data = _simplify(tensor)

static detail (*worker: AbstractWorker*, *tensor_tuple: tuple*)
This function reconstructs a LogTensor given it's attributes in form of a tuple. :param worker: the worker doing the deserialization :param tensor_tuple: a tuple holding the attributes of the LogTensor

Returns a LogTensor

Return type *PaillierTensor*

Examples

logtensor = detail(data)

1.1.1.2.1.96 syft.frameworks.torch.tensors.interpreters.placeholder

1.1.1.2.1.97 Module Contents

class syft.frameworks.torch.tensors.interpreters.placeholder.**PlaceHolder**(*owner=None*, *id=None*, *tags: set = None*, *description: str = None*)

Bases: *syft.generic.tensor.AbstractTensor*

__repr__

instantiate(*self*, *tensor*)
Add a tensor as a child attribute. All operations on the placeholder will be also executed on this child tensor.

We remove wrappers is there are any.

__str__(*self*)
Compact representation of a Placeholder, including tags and optional child

copy(*self*)
Copying a placeholder doesn't duplicate the child attribute, because all copy operations happen locally where we want to keep reference to the same instantiated object. As the child doesn't get sent, this is not an issue.

static simplify(*worker: AbstractWorker*, *tensor: PlaceHolder*)
Takes the attributes of a PlaceHolder and saves them in a tuple.

Parameters

- **worker** – the worker doing the serialization
- **tensor** – a PlaceHolder.

Returns a tuple holding the unique attributes of the PlaceHolder.

Return type tuple

static detail(*worker: AbstractWorker*, *tensor_tuple: tuple*)
This function reconstructs a PlaceHolder given it's attributes in form of a tuple. :param worker: the worker doing the deserialization :param tensor_tuple: a tuple holding the attributes of the PlaceHolder

Returns a PlaceHolder

Return type *PlaceHolder*

static bufferize(*worker: AbstractWorker*, *tensor: PlaceHolder*)
Takes the attributes of a PlaceHolder and saves them in a Protobuf message.

Parameters

- **worker** – the worker doing the serialization
- **tensor** – a PlaceHolder.

Returns a Protobuf message holding the unique attributes of the PlaceHolder.

Return type PlaceholderPB

static unbufferize(*worker: AbstractWorker*, *protobuf_placeholder: PlaceholderPB*)
This function reconstructs a PlaceHolder given it's attributes in form of a Protobuf message. :param worker: the worker doing the deserialization :param protobuf_placeholder: a Protobuf message holding the attributes of the PlaceHolder

Returns a PlaceHolder

Return type *PlaceHolder*

1.1.1.2.1.98 syft.frameworks.torch.tensors.interpreters.plusisminus

1.1.1.2.1.99 syft.frameworks.torch.tensors.interpreters.polynomial

1.1.1.2.1.100 syft.frameworks.torch.tensors.interpreters.precision

1.1.1.2.1.101 Module Contents

class syft.frameworks.torch.tensors.interpreters.precision.**FixedPrecisionTensor**(*owner=None*, *id=None*, *field: int = 2**62*, *base: int = 10*, *precision_fractional: int = 3*, *kappa: int = 1*, *tags: set = None*, *description: str = None*)

Bases: *syft.generic.tensor.AbstractTensor*

__add__

__radd__

__sub__

__mul__

mul_

__truediv__

__pow__

__matmul__

mm

__eq__

get_class_attributes(*self*)
Specify all the attributes need to build a wrapper correctly when returning a response, for example precision_fractional is important when wrapping the result of a method on a self which is a fixed precision tensor with a non default precision_fractional.

property data(*self*)

property grad(*self*)
Gradient makes no sense for Fixed Precision Tensor, so we make it clear that if someone query .grad on a Fixed Precision Tensor it doesn't error but returns grad and can't be set

backward(*self*, **args*, ***kwargs*)
Calling backward on Precision Tensor doesn't make sense, but sometimes a call can be propagated downward the chain to an Precision Tensor (for example in create_grad_objects), so we just ignore the call.

attr(*self*, *attr_name*)

fix_precision(*self*, *check_range=True*)
This method encodes the .child object using fixed precision

float_precision(*self*)
this method returns a new tensor which has the same values as this one, encoded with floating point precision

truncate(*self*, *precision_fractional*, *check_sign=True*)

add(*self*, *_self*, *other*)
Add two fixed precision tensors together.

add_(*self*, *value_or_tensor*, *tensor=None*)

__iadd__(*self*, *other*)
Add two fixed precision tensors together.

sub(*self*, *_self*, *other*)
Subtracts a fixed precision tensor from another one.

__rsub__(*self*, *other*)

sub_(*self*, *value_or_tensor*, *tensor=None*)

__isub__(*self*, *other*)

t(*self*, *_self*, **args*, ***kwargs*)
Transpose a tensor. Hooked is handled by the decorator

mul_and_div(*self*, *other*, *cmd*)
Hook manually mul and div to add the truncation/rescaling part which is inherent to these operations in the fixed precision setting

mul(*self*, *other*)

__imul__(*self*, *other*)

div(*self*, *other*)

__itruediv__(*self*, *other*)

pow(*self*, *power*)
Compute integer power of a number by recursion using mul

This uses the following trick:

- Divide power by 2 and multiply base to itself (if the power is even)

- Decrement power by 1 to make it even and then follow the first step

Parameters **power** (*int*) – the exponent supposed to be an integer > 0

matmul (*self*, **args*, ***kwargs*)
Hook manually matmul to add the truncation part which is inherent to multiplication in the fixed precision setting

inverse (*self*, *iterations=8*)
Computes an approximation of the matrix inversion using Newton-Schulz iterations

exp (*self*, *iterations=8*)
Approximates the exponential function using a limit approximation: exp(x) = lim_{n -> infty} (1 + x / n) ^ n

Here we compute exp by choosing n = 2 ** d for some large d equal to iterations. We then compute (1 + x / n) once and square *d* times.

Parameters **iterations** (*int*) – number of iterations for limit approximation

Ref: https://github.com/LaRiffle/approximate-models

sigmoid (*self*, *method='exp'*)
Approximates the sigmoid function

Parameters

- **self** – the fixed precision tensor
- **method** (*str*) – (default = "exp") "exp": Use the exponential approximation and the sigmoid definition

  sigmoid(x) = 1 / (1 + exp(-x))

  "maclaurin": Use the Maclaurin / Taylor approximation, with polynomial interpolation of degree 5 over [-8,8] NOTE: This method is faster but not as precise as "exp" Ref: https://mortendahl.github.io/2017/04/17/private-deep-learning-with-mpc/#approximating-sigmoid

log (*self*, *iterations=2*, *exp_iterations=8*)
Approximates the natural logarithm using 8th order modified Householder iterations. Recall that Householder method is an algorithm to solve a non linear equation f(x) = 0. Here f: x -> 1 - C * exp(-x) with C = self

Iterations are computed by: y_0 = some constant h = 1 - self * exp(-y_n) y_{n+1} = y_n - h * (1 + h / 2 + h^2 / 3 + h^3 / 6 + h^4 / 5 + h^5 / 7)

Parameters

- **iterations** (*int*) – number of iterations for 6th order modified Householder approximation.
- **exp_iterations** (*int*) – number of iterations for limit approximation of exp

Ref: https://github.com/LaRiffle/approximate-models

__gt__ (*self*, *_self*, *other*)

__ge__ (*self*, *_self*, *other*)

__lt__ (*self*, *_self*, *other*)

__le__(*self*, *_self*, *other*)

eq(*self*, *_self*, *other*)

static torch(*module*)

classmethod handle_func_command(*cls*, *command*)
Receive an instruction for a function to be applied on a FixedPrecision Tensor, Perform some specific action (like logging) which depends of the instruction content, replace in the args all the FPTensors with their child attribute, forward the command instruction to the handle_function_command of the type of the child attributes, get the response and replace a FixedPrecision on top of all tensors found in the response. :param command: instruction of a function command: (command name, <no self>, arguments[, kwargs]) :return: the response of the function command

share(*self*, **owners*, *field=None*, *crypto_provider=None*)
Forward the .share() command to the child tensor, and reconstruct a new FixedPrecisionTensor since the command is not inplace and should return a new chain

Parameters

- ***owners** – the owners of the shares of the resulting AdditiveSharingTensor
- **field** – the field size in which the share values live
- **crypto_provider** – the worker used to provide the crypto primitives used to perform some computations on AdditiveSharingTensors

Returns A FixedPrecisionTensor whose child has been shared

share_(*self*, **args*, ***kwargs*)
Performs an inplace call to share. The FixedPrecisionTensor returned is therefore the same, contrary to the classic share version version

static simplify(*worker: AbstractWorker*, *tensor: FixedPrecisionTensor*)
Takes the attributes of a FixedPrecisionTensor and saves them in a tuple.

Parameters

- **worker** – the worker doing the serialization
- **tensor** – a FixedPrecisionTensor.

Returns a tuple holding the unique attributes of the fixed precision tensor.

Return type tuple

static detail(*worker: AbstractWorker*, *tensor_tuple: tuple*)
This function reconstructs a FixedPrecisionTensor given it's attributes in form of a tuple. :param worker: the worker doing the deserialization :param tensor_tuple: a tuple holding the attributes of the FixedPrecisionTensor

Returns a FixedPrecisionTensor

Return type *FixedPrecisionTensor*

Examples

shared_tensor = detail(data)

1.1.1.2.1.102 syft.frameworks.torch.tensors.interpreters.private

1.1.1.2.1.103 Module Contents

class syft.frameworks.torch.tensors.interpreters.private.**PrivateTensor**(*owner=None*, *id=None*, *tags: set = None*, *description: str = None*, *allowed_users=tuple()*, *parents=tuple()*, *command: str = None*)

Bases: *syft.generic.tensor.AbstractTensor*

get_class_attributes(*self*)
Specify all the attributes need to build a wrapper correctly when returning a response.

allow(*self*, *user*)
Overwrite native's allowed to verify if a specific user is allowed to get this tensor.

Parameters **user** (*object*) – user to be verified.

Returns A boolean value (True if the user is allowed and false if it isn't).

Return type bool

register_credentials(*self*, *users: Union[object, tuple] = []*)
Register a new user credential(s) into the list of allowed users to get this tensor.

Parameters **users** (*object or List*) – Credential(s) to be registered.

float_precision(*self*)
Forward float_precision method to next child on tensor stack.

static torch(*module*)

static simplify(*worker: AbstractWorker*, *tensor: PrivateTensor*)
Takes the attributes of a PrivateTensor and saves them in a tuple.

Parameters **tensor** (PrivateTensor) – a PrivateTensor.

Returns a tuple holding the unique attributes of the fixed private tensor.

Return type tuple

static detail(*worker: AbstractWorker*, *tensor_tuple: tuple*)

This function reconstructs a PrivateTensor given it's attributes in form of a tuple. :param worker: the worker doing the deserialization :type worker: AbstractWorker :param tensor_tuple: a tuple holding the attributes of the PrivateTensor :type tensor_tuple: tuple

Returns a PrivateTensor

Return type *PrivateTensor*

Examples

shared_tensor = detail(data)

1.1.1.2.1.104 Submodules

1.1.1.2.1.105 syft.frameworks.torch.functions

1.1.1.2.1.106 Module Contents

syft.frameworks.torch.functions.**combine_pointers**(**pointers: List[ObjectPointer]*) → MultiPointerTensor

Accepts a list of pointers and returns them as a MultiPointerTensor. See MultiPointerTensor docs for details.

Arg:

***pointers: a list of pointers to tensors (including** their wrappers like normal)

1.1.1.2.1.107 syft.frameworks.torch.torch_attributes

1.1.1.2.1.108 Module Contents

class syft.frameworks.torch.torch_attributes.**TorchAttributes**(*torch: ModuleType*, *hook: ModuleType*)

Bases: *syft.generic.frameworks.attributes.FrameworkAttributes*

Adds torch module related custom attributes.

TorchAttributes is a special class where all custom attributes related to the torch module can be added. Any global parameter, configuration, or reference relating to PyTorch should be stored here instead of attaching it directly to some other part of the global namespace.

The main reason we need this is because the hooking process occasionally needs to save global objects, notably including what methods to hook and what methods to NOT hook.

This will hold all necessary attributes PySyft needs.

Parameters

- **torch** – A ModuleType indicating the torch module
- **hook** – A TorchHook to stash

ALIAS = torch

Tensor

is_inplace_method(*self*, *method_name*)
Determine if a method is inplace or not.

Check if the method ends by _ and is not a __xx__, then stash for constant-time lookup.

Parameters **method_name** – The name for the method.

Returns Boolean denoting if the method is inplace or not.

1.1.1.3 syft.generic

1.1.1.3.1 Subpackages

1.1.1.3.1.1 syft.generic.frameworks

1.1.1.3.2 Subpackages

1.1.1.3.1.3 syft.generic.frameworks.hook

1.1.1.3.4 Submodules

1.1.1.3.1.5 syft.generic.frameworks.hook.hook

1.1.1.3.6 Module Contents

class syft.generic.frameworks.hook.hook.**FrameworkHook**(*framework_module*, *local_worker: BaseWorker = None*, *is_client: bool = True*)

Bases: abc.ABC

abstract classmethod create_shape(*cls*, *shape_dims*)
Factory method for creating a generic FrameworkShape.

abstract classmethod create_zeros(*cls*, *shape*, *dtype*, ***kwargs*)
Factory method for creating a generic zero FrameworkTensor.

classmethod create_wrapper(*cls*, *wrapper_type*, **args*, ***kwargs*)
Factory method for creating a generic wrapper of type wrapper_type.

abstract _hook_native_tensor(*self*, *tensor_type: type*, *syft_type: type*)
Add PySyft-specific tensor functionality to the given tensor type.

See framework-specific implementations for more details.

classmethod _transfer_methods_to_framework_class(*hook_cls*, *framework_cls: type*, *from_cls: type*, *exclude: List[str]*)
Adds methods from the from_cls class to the framework_cls class.

The class from_cls is a proxy class useful to avoid extending the native framework class directly.

Parameters

- **framework_cls** – The class to which we are adding methods, e.g. torch.Tensor or tf.Variable.
- **from_cls** – The class from which we are adding methods, e.g. TorchTensor, or TensorFlowVariable.

- **exclude** – A list of method names to exclude from the hooking process.

_hook_native_methods (*self*, *tensor_type: type*)
Add hooked version of all methods of to_auto_overload[tensor_type] to the tensor_type; instead of performing the native tensor method, the hooked version will be called

Parameters **tensor_type** – the tensor_type which holds the methods

_hook_properties (*hook_self*, *tensor_type: type*)
Overloads tensor_type properties.

If you're not sure how properties work, read: https://www.programiz.com/python-programming/property
:param tensor_type: The tensor class which is having properties

added to it.

_which_methods_should_we_auto_overload (*self*, *tensor_type: type*)
Creates a list of Torch methods to auto overload.

By default, it looks for the intersection between the methods of tensor_type and torch_type minus those in the exception list (syft.torch.exclude).

Parameters

- **tensor_type** – Iterate through the properties of this tensor type.
- **syft_type** – Iterate through all attributes in this type.

Returns A list of methods to be overloaded.

_hook_syft_tensor_methods (*self*, *tensor_type: type*, *syft_type: type*)
Add hooked version of all methods of to_auto_overload[tensor_type] to the syft_type, so that they act like regular tensors in terms of functionality, but instead of performing the native tensor method, it will be forwarded to each share when it is relevant

Parameters

- **tensor_type** – The tensor type to which we are adding methods.
- **syft_type** – the syft_type which holds the methods

_hook_private_tensor_methods (*self*, *tensor_type: type*, *syft_type: type*)
Add hooked version of all methods of the tensor_type to the Private Tensor: It'll add references to its parents and save command/operations history.

_hook_pointer_tensor_methods (*self*, *tensor_type*)
Add hooked version of all methods of the tensor_type to the Pointer tensor: instead of performing the native tensor method, it will be sent remotely to the location the pointer is pointing at.

_hook_object_pointer_methods (*self*, *framework_cls*)
Add hooked version of all methods of the framework_cls to the ObjectPointer: instead of performing the native object method, it will be sent remotely to the location the pointer is pointing at.

_hook_multi_pointer_tensor_methods (*self*, *tensor_type*)
Add hooked version of all methods of the torch Tensor to the Multi Pointer tensor: instead of performing the native tensor method, it will be sent remotely for each pointer to the location it is pointing at.

_hook_string_methods (*self*, *owner*)

_hook_string_pointer_methods (*self*)

_add_registration_to___init__ (*hook_self*, *tensor_type: type*, *is_tensor: bool = False*)
Adds several attributes to the tensor.

Overload tensor_type.__init__ to add several attributes to the tensor as well as (optionally) registering the tensor automatically. TODO: auto-registration is disabled at the moment, this might be bad.

Parameters

- **tensor_type** – The class of the tensor being hooked
- **torch_tensor** – An optional boolean parameter (default False) to specify whether to skip running the native initialization logic. TODO: this flag might never get used.

classmethod _perform_function_overloading(*cls*, *parent_module_name*, *parent_module*, *func_name*)

classmethod _get_hooked_syft_method(*cls*, *attr*)

Hook a method in order to replace all args/kwargs syft/torch tensors with their child attribute, forward this method with the new args and new self, get response and "rebuild" the syft tensor wrapper upon all tensors found

Parameters **attr** (*str*) – the method to hook

Returns the hooked method

classmethod _get_hooked_method(*cls*, *tensor_type*, *method_name*)

Hook a method in order to replace all args/kwargs syft/torch tensors with their child attribute if they exist If so, forward this method with the new args and new self, get response and "rebuild" the torch tensor wrapper upon all tensors found If not, just execute the native torch methodn

Parameters **attr** (*str*) – the method to hook

Returns the hooked method

classmethod _get_hooked_private_method(*cls*, *method_name*)

Hook a method in order to replace all args/kwargs syft/torch tensors with their child attribute if they exist If so, forward this method with the new args and new self, get response and "rebuild" the torch tensor wrapper upon all tensors found If not, just execute the native torch methodn

Parameters **attr** (*str*) – the method to hook

Returns the hooked method

classmethod _get_hooked_func(*cls*, *public_module_name*, *func_api_name*, *func*)

Hook a function in order to inspect its args and search for pointer or other syft tensors. - Calls to this function with normal tensors or numbers / string trigger

usual behaviour

- Calls with pointers send the command to the location of the pointer(s)
- Calls with syft tensor will in the future trigger specific behaviour

Parameters

- **public_module_name** (*str*) – the name of the public module you are hooking this function on (ie the same name that the user would import).
- **attr** (*str*) – the method to hook

Returns the hooked method

classmethod _get_hooked_pointer_method(*cls*, *attr*)

Hook a method to send it to remote worker

Parameters **attr** (*str*) – the method to hook

Returns the hooked method

classmethod _get_hooked_multi_pointer_method(*cls*, *attr*)
Hook a method to send it multiple remote workers

Parameters **attr** (*str*) – the method to hook

Returns the hooked method

classmethod _string_input_args_adaptor(*cls, args: Tuple[object]*)
This method is used when hooking String methods.

Some 'String' methods which are overriden from 'str' such as the magic '__add__' method expects an object of type 'str' as its first argument. However, since the '__add__' method here is hooked to a String type, it will receive arguments of type 'String' not 'str' in some cases. This won't worker for the underlying hooked method '__add__' of the 'str' type. That is why the 'String' argument to '__add__' should be peeled down to 'str'

Parameters **args** – A tuple or positional arguments of the method being hooked to the String class.

Returns A list of adapted positional arguments.

classmethod _wrap_str_return_value(*cls*, *_self*, *attr: str*, *value: object*)

classmethod _get_hooked_string_method(*cls*, *attr*)
Hook a *str* method to a corresponding method of *String* with the same name.

Args: attr (str): the method to hook

Return: the hooked method

classmethod _get_hooked_string_pointer_method(*cls*, *attr*)
Hook a *String* method to a corresponding method of *StringPointer* with the same name.

Args: attr (str): the method to hook

Return: the hooked method

1.1.1.3.1.7 syft.generic.frameworks.hook.hook_args

1.1.1.3.1.8 Module Contents

syft.generic.frameworks.hook.hook_args.**hook_method_args_functions**

syft.generic.frameworks.hook.hook_args.**hook_method_response_functions**

syft.generic.frameworks.hook.hook_args.**get_tensor_type_functions**

syft.generic.frameworks.hook.hook_args.**base_types**

syft.generic.frameworks.hook.hook_args.**one**

syft.generic.frameworks.hook.hook_args.**get_child**

syft.generic.frameworks.hook.hook_args.**type_rule**

syft.generic.frameworks.hook.hook_args.**forward_func**

syft.generic.frameworks.hook.hook_args.**backward_func**

syft.generic.frameworks.hook.hook_args.**ambiguous_methods**

syft.generic.frameworks.hook.hook_args.**ambiguous_functions**

syft.generic.frameworks.hook.hook_args.**register_type_rule**(*new_type_rules: Dict*)

syft.generic.frameworks.hook.hook_args.**register_forward_func**(*new_forward_rules: Dict*)

syft.generic.frameworks.hook.hook_args.**register_backward_func**(*new_backward_rules: Dict*)

syft.generic.frameworks.hook.hook_args.**register_ambiguous_method**(**method*)

syft.generic.frameworks.hook.hook_args.**register_ambiguous_function**(**function*)

syft.generic.frameworks.hook.hook_args.**default_backward_func**(*tensorcls*)

syft.generic.frameworks.hook.hook_args.**default_register_tensor**(**tensorcls*)

syft.generic.frameworks.hook.hook_args.**unwrap_args_from_method**(*attr*, *method_self*, *args*, *kwargs*)

Method arguments are sometimes simple types (such as strings or ints) but sometimes they are custom Syft tensors such as wrappers (i.e. FrameworkTensor), LoggingTensor or some other tensor type. Complex types (which have a .child attribute) need to have arguments converted from the arg to arg.child so that the types match as the method is being called down the chain. To make this efficient, we cache which args need to be replaced with their children in a dictionary called hook_method_args_functions. However, sometimes a method (an attr) has multiple different argument signatures, such that sometimes arguments have .child objects and other times they don't (such as x.div(), which can accept either a tensor or a float as an argument). This invalidates the cache, so we need to have a try/except which refreshes the cache if the signature triggers an error.

Parameters

- **attr** (*str*) – the name of the method being called
- **method_self** – the tensor on which the method is being called
- **args** (*list*) – the arguments being passed to the method
- **kwargs** (*dict*) – the keyword arguments being passed to the function (these are not hooked ie replace with their .child attr)

syft.generic.frameworks.hook.hook_args.**unwrap_args_from_function**(*attr*, *args*, *kwargs*, *return_args_type=False*)

See unwrap_args_from_method for details

Parameters

- **attr** (*str*) – the name of the function being called
- **args** (*list*) – the arguments being passed to the function
- **kwargs** (*dict*) – the keyword arguments being passed to the function (these are not hooked ie replace with their .child attr)
- **return_args_type** (*bool*) – return the type of the tensors in the
- **arguments** (*original*) –

Returns

- the arguments where all tensors are replaced with their child
- the type of this new child

(- the type of the tensors in the arguments)

syft.generic.frameworks.hook.hook_args.**build_unwrap_args_from_function**(*args*, *return_tuple=False*)

Build the function f that hook the arguments: f(args) = new_args

syft.generic.frameworks.hook.hook_args.**hook_response**(*attr*, *response*, *wrap_type*, *wrap_args={}*, *new_self=None*)

When executing a command, arguments are inspected and all tensors are replaced with their child attribute until a pointer or a framework tensor is found (for example an argument could be a framework wrapper with a child being a LoggingTensor, with a child being a framework tensor). When the result of the command is calculated, we need to rebuild this chain in the reverse order (in our example put back a LoggingTensor on top of the result and then a framework wrapper). To make this efficient, we cache which elements of the response (which can be more complicated with nested tuples for example) need to be wrapped in a dictionary called hook_method_response_functions. However, sometimes a method (an attr) has multiple different response signatures. This invalidates the cache, so we need to have a try/except which refreshes the cache if the signature triggers an error.

Parameters

- **attr** (*str*) – the name of the method being called
- **response** (*list or dict*) – the arguments being passed to the tensor
- **wrap_type** (*type*) – the type of wrapper we'd like to have
- **wrap_args** (*dict*) – options to give to the wrapper (for example the
- **for the precision tensor)** (*precision*) –
- **new_self** – used for the can just below of inplace ops

syft.generic.frameworks.hook.hook_args.**build_wrap_reponse_from_function**(*response*, *wrap_type*, *wrap_args*)

Build the function that hook the response.

Example

p is of type Pointer f is the hook_response_function then f(p) = (Wrapper)>Pointer

syft.generic.frameworks.hook.hook_args.**build_rule**(*args*)

Inspect the args object to find framework or syft tensor arguments and return a rule whose structure is the same as the args object, with 1 where there was (framework or syft) tensors and 0 when not (ex: number, str, . . .)

Example

in: ([tensor(1, 2), Pointer@bob], 42) out: ([1, 1], 0)

syft.generic.frameworks.hook.hook_args.**build_unwrap_args_with_rules**(*args*, *rules*, *return_tuple=False*)

Build a function given some rules to efficiently replace in the args object syft tensors with their child (but not pointer as they don't have .child), and do nothing for other type of object including framework tensors, str, numbers, bool, etc. Pointers trigger an error which can be caught to get the location for forwarding the call.

Parameters

- **args** (*tuple*) – the arguments given to the function / method

- **rules** (*tuple*) – the same structure but with boolean, true when there is a tensor
- **return_tuple** (*bool*) – force to return a tuple even with a single element

Returns a function that replace syft arg in args with arg.child

syft.generic.frameworks.hook.hook_args.**build_get_tensor_type**(*rules*, *layer=None*)

Build a function which uses some rules to find efficiently the first tensor in the args objects and return the type of its child.

Parameters

- **rules** (*tuple*) – a skeleton object with the same structure as args but each tensor is replaced with a 1 and other types (int, str) with a 0
- **layer** (*list or None*) – keep track of the path of inspection: each element in the list stand for one layer of deepness into the object, and its value for the index in the current layer. See example for details

Returns a function returning a type

Example

Understanding the layer parameter obj = (a, [b, (c, d)], e) the layer position is for: a: [0] b: [1, 0] c: [1, 1, 0] d: [1, 1, 1] e: [2]

Global behaviour example rules = (0, [1, (0, 0), 0) - First recursion level

0 found -> do nothing list found -> recursive call with layer = [1]

- Second recursion level 1 found -> update layer to [1, 0]

  build the function x: type(x[1][0]) break

- Back to first recursion level save the function returned in the lambdas list 0 found -> do nothing exit loop return the first (and here unique) function

syft.generic.frameworks.hook.hook_args.**one_layer**(*idx1*)

syft.generic.frameworks.hook.hook_args.**two_layers**(*idx1*, *idx2*)

syft.generic.frameworks.hook.hook_args.**three_layers**(*idx1*, **ids*)

syft.generic.frameworks.hook.hook_args.**four_layers**(*idx1*, **ids*)

syft.generic.frameworks.hook.hook_args.**get_element_at**

syft.generic.frameworks.hook.hook_args.**build_wrap_response_with_rules**(*response*, *rules*, *wrap_type*, *wrap_args*, *return_tuple=False*, *return_list=False*)

Build a function given some rules to efficiently replace in the response object syft or framework tensors with a wrapper, and do nothing for other types of object including , str, numbers, bool, etc.

Parameters

- **response** – a response used to build the hook function

- **rules** – the same structure objects but with boolean, at true when is replaces a tensor
- **return_tuple** – force to return a tuple even with a single element

Response: a function to "wrap" the response

syft.generic.frameworks.hook.hook_args.**zero_fold**(**a*, ***k*)

syft.generic.frameworks.hook.hook_args.**one_fold**(*return_tuple*, ***kwargs*)

syft.generic.frameworks.hook.hook_args.**two_fold**(*lambdas*, *args*, ***kwargs*)

syft.generic.frameworks.hook.hook_args.**three_fold**(*lambdas*, *args*, ***kwargs*)

syft.generic.frameworks.hook.hook_args.**four_fold**(*lambdas*, *args*, ***kwargs*)

syft.generic.frameworks.hook.hook_args.**five_fold**(*lambdas*, *args*, ***kwargs*)

syft.generic.frameworks.hook.hook_args.**six_fold**(*lambdas*, *args*, ***kwargs*)

syft.generic.frameworks.hook.hook_args.**seven_fold**(*lambdas*, *args*, ***kwargs*)

syft.generic.frameworks.hook.hook_args.**eight_fold**(*lambdas*, *args*, ***kwargs*)

syft.generic.frameworks.hook.hook_args.**many_fold**(*lambdas*, *args*, ***kwargs*)

syft.generic.frameworks.hook.hook_args.**typed_identity**(*a*)

We need to add typed identity for arguments which can be either number or tensors. If the argument changes from an int to a tensor, the assertion error triggered by typed_identity will be caught and a new signature will be computed for the command.

syft.generic.frameworks.hook.hook_args.**register_response_functions**

syft.generic.frameworks.hook.hook_args.**register_response**(*attr: str*, *response: object*, *response_ids: object*, *owner: AbstractWorker*) → object

When a remote worker execute a command sent by someone else, the response is inspected: all tensors are stored by this worker and a Pointer tensor is made for each of them.

To make this efficient, we cache which elements of the response (which can be more complicated with nested tuples for example) in the dict register_response_functions

However, sometimes a function (an attr) has multiple different response signatures. This invalidates the cache, so we need to have a try/except which refreshes the cache if the signature triggers an error.

Parameters

- **attr** (*str*) – the name of the function being called
- **response** (*object*) – the response of this function
- **owner** (BaseWorker) – the worker which registers the tensors

syft.generic.frameworks.hook.hook_args.**build_register_response_function**(*response: object*) → Callable

Build the function that registers the response and replaces tensors with pointers.

Example

(1, tensor([1, 2]) is the response f is the register_response_function then f(p) = (1, (Wrapper)>Pointer)

syft.generic.frameworks.hook.hook_args.**register_tensor**(*tensor: FrameworkTensorType*, *owner: AbstractWorker*, *response_ids: List = list()*)

Registers a tensor.

Parameters

- **tensor** – A tensor.
- **owner** – The owner that makes the registration.
- **response_ids** – List of ids where the tensor should be stored and each id is pop out when needed.

syft.generic.frameworks.hook.hook_args.**build_register_response**(*response: object*, *rules: Tuple*, *return_tuple: bool = False*) → Callable

Build a function given some rules to efficiently replace in the response object framework tensors with a pointer after they are registered, and do nothing for other types of object including , str, numbers, bool, etc.

Parameters

- **response** – the response
- **rules** – the rule specifying where the tensors are
- **return_tuple** – force to return a tuple even with a single element

Returns The function to apply on generic responses

1.1.1.3.1.9 syft.generic.frameworks.hook.trace

1.1.1.3.1.10 Module Contents

class syft.generic.frameworks.hook.trace.**Trace**

Manage logging operations on the fly as they run

active

if True, recording of operation is active

out_of_operation

it True, is ready to record the next operation, means that it's not inside the execution of an already recorded operation

logs

the list of operations recorded

clear(*self*)

enabled(*self*)

syft.generic.frameworks.hook.trace.**tracer**(*func_name=None*, *method_name=None*)

This is a decorator which allows to record operations and their results when a function or a method is hooked

This decorator is applied on overloaded_native_method and overloaded_func in the generic hook

1.1.1.3.1.11 Submodules

1.1.1.3.1.12 syft.generic.frameworks.attributes

1.1.1.3.1.13 Module Contents

class syft.generic.frameworks.attributes.**FrameworkAttributes**(*framework: ModuleType*, *hook: FrameworkHook*)

Bases: abc.ABC

property ALIAS(*cls*)

property Tensor(*cls*)
Default Tensor wrapper.

abstract is_inplace_method(*self*, *method_name*)
Determine if a method is inplace or not.

Framework-dependent, see subclasses for details.

Parameters **method_name** – The name for the method.

Returns Boolean denoting if the method is inplace or not.

_command_guard(*self*, *command: str*, *get_native: bool = False*)
Check command can be safely used.

Parameters

- **command** – A string indicating command name.
- **get_native** – A boolean parameter (default False) to indicate whether to return the command name or the native torch function. If False, return command name else return the native torch function.

Returns The command name or a native framework function

_is_command_valid_guard(*self*, *command: str*)
Validate the command.

Indicates whether a command is valid with respect to the framework guard.

Parameters

- **command** – A string indicating command to test.
- **framework_domain** – A string indicating the framework domain or module in which the command is supposed to be, e.g. dir(torch), dir(torch.Tensor), dir(tensorflow), etc. (roughly)

Returns A boolean indicating whether the command is valid.

classmethod get_native_framework_name(*cls*, *attr: str*)
Return the name of the native command for the given hooked command.

Parameters **attr** – A string indicating the hooked command name (ex: torch.add)

Returns torch.native_add)

Return type The name of the native command (ex

1.1.1.3.1.14 syft.generic.frameworks.overload

1.1.1.3.1.15 Module Contents

class syft.generic.frameworks.overload.**Module**
Bases: object

class syft.generic.frameworks.overload.**Overloaded**

static overload_method(*attr*)
hook args and response for methods that hold the @overloaded.method decorator

static overload_function(*attr*)
hook args and response for functions that hold the @overloaded.function decorator

static overload_module(*attr*)

syft.generic.frameworks.overload.**overloaded**

1.1.1.3.1.16 syft.generic.frameworks.remote

1.1.1.3.1.17 Module Contents

class syft.generic.frameworks.remote.**Remote**(*worker*, *framework_name*)
Bases: object

frameworks

static register_framework(*cls*)

1.1.1.3.1.18 syft.generic.frameworks.types

1.1.1.3.1.19 Module Contents

syft.generic.frameworks.types.**framework_tensors = []**

syft.generic.frameworks.types.**framework_shapes = []**

syft.generic.frameworks.types.**framework_layer_modules = []**

syft.generic.frameworks.types.**framework_layer_module**

syft.generic.frameworks.types.**framework_tensors**

syft.generic.frameworks.types.**FrameworkTensorType**

syft.generic.frameworks.types.**FrameworkTensor**

syft.generic.frameworks.types.**framework_shapes**

syft.generic.frameworks.types.**FrameworkShapeType**

syft.generic.frameworks.types.**FrameworkShape**

syft.generic.frameworks.types.**framework_layer_modules**

syft.generic.frameworks.types.**FrameworkLayerModuleType**

syft.generic.frameworks.types.**FrameworkLayerModule**

1.1.1.3.1.20 syft.generic.pointers

1.1.1.3.1.21 Submodules

1.1.1.3.1.22 syft.generic.pointers.callable_pointer

1.1.1.3.1.23 Module Contents

class syft.generic.pointers.callable_pointer.**CallablePointer**(*location: BaseWorker = None*, *id_at_location: Union[str, int] = None*, *owner: BaseWorker = None*, *id: Union[str, int] = None*, *garbage_collect_data: bool = True*, *point_to_attr: str = None*, *tags: List[str] = None*, *description: str = None*)

Bases: *syft.generic.pointers.object_pointer.ObjectPointer*

A class of pointers that are callable

A CallablePointer is an ObjectPointer which implements the __call__ function. This lets you execute a command directly on the object to which it points.

__call__(*self*, **args*, ***kwargs*)

syft.generic.pointers.callable_pointer.**create_callable_pointer**(*location: BaseWorker*, *id: str or int*, *id_at_location: str or int*, *owner: BaseWorker*, *tags*, *description*, *garbage_collect_data: bool = True*, *register_pointer: bool = True*) → ObjectPointer

Creates a callable pointer to the object identified by the pair (location, id_at_location).

Note, that there is no check whether an object with this id exists at the location.

Parameters

- **location** –
- **id** –

- **id_at_location** –
- **owner** –
- **tags** –
- **description** –
- **garbage_collect_data** –
- **register_pointer** –

Returns:

1.1.1.3.1.24 syft.generic.pointers.multi_pointer

1.1.1.3.1.25 Module Contents

class syft.generic.pointers.multi_pointer.**MultiPointerTensor**(*owner: BaseWorker = None*, *id: Union[str, int] = None*, *tags: List[str] = None*, *description: str = None*, *children: List[AbstractTensor] = []*)

Bases: *syft.generic.tensor.AbstractTensor*

The MultiPointerTensor gathers together several pointers to different locations which can be operated all at once. The pointers can be referencing the same value or a different one depending on the usage.

The MultiPointerTensor has the same structure that the AdditiveSharedTensor: its child attribute is a dictionary {worker.id: Pointer}

MultiPointerTensor can be directly instantiated using x.send(worker1, worker2, etc) where x is a syft or framework tensor. In that case, the value of x will be sent and replicated to each workers.

__str__(*self*)

property grad(*self*, *self_*)

__eq__(*self*, *other*)

__add__(*self*, *other*)
Adding a MultiPointer (MPT) and an AdditiveShared Tensor (AST) should return an AdditiveShared Tensor, so if we have this configuration, we permute self and other to use the fact that other.__add__(...) return an object of type other

Else, we just redirect to .add which works well

__mul__(*self*, *other*)
See __add__ for details but, MPT * AST should return AST

property shape(*self*)
This method returns the shape of the data being pointed to. This shape information SHOULD be cached on self._shape, but occasionally this information may not be present. If this is the case, then it requests the shape information from the remote object directly (which is inefficient and should be avoided).

dim(*self*)
This method fixes the error that the result of dim was a list of ints stored inside a multipointer tensor

get (*self*, *sum_results: bool = False*)

virtual_get (*self*, *sum_results: bool = False*)
Get the value of the tensor without calling get - Only for VirtualWorkers

static dispatch (*args*, *worker*)
utility function for handle_func_command which help to select shares (seen as elements of dict) in an argument set. It could perhaps be put elsewhere

Parameters

- **args** – arguments to give to a functions
- **worker** – owner of the shares to select

Returns args where the MultiPointerTensor are replaced by the appropriate share

classmethod handle_func_command (*cls*, *command*)
Receive an instruction for a function to be applied on a Syft Tensor, Replace in the args all the LogTensors with their child attribute, forward the command instruction to the handle_function_command of the type of the child attributes, get the response and replace a Syft Tensor on top of all tensors found in the response.

Parameters

- **command** – instruction of a function command: (command name,
- **self>, arguments[, kwargs])** (*<no*) –

Returns the response of the function command

set_garbage_collect_data (*self*, *value*)

static simplify (*worker: AbstractWorker*, *tensor: MultiPointerTensor*)
This function takes the attributes of a MultiPointerTensor and saves them in a tuple :param tensor: a MultiPointerTensor :type tensor: MultiPointerTensor

Returns a tuple holding the unique attributes of the additive shared tensor

Return type tuple

Examples

data = simplify(tensor)

static detail (*worker: AbstractWorker*, *tensor_tuple: tuple*)
This function reconstructs a MultiPointerTensor given it's attributes in form of a tuple. :param worker: the worker doing the deserialization :param tensor_tuple: a tuple holding the attributes of the MultiPointerTensor

Returns a MultiPointerTensor

Return type *MultiPointerTensor*

Examples

multi_pointer_tensor = detail(data)

1.1.1.3.1.26 syft.generic.pointers.object_pointer

1.1.1.3.1.27 Module Contents

class syft.generic.pointers.object_pointer.**ObjectPointer**(*location: BaseWorker = None*, *id_at_location: Union[str, int] = None*, *owner: BaseWorker = None*, *id: Union[str, int] = None*, *garbage_collect_data: bool = True*, *point_to_attr: str = None*, *tags: List[str] = None*, *description: str = None*)

Bases: *syft.generic.object.AbstractObject*

A pointer to a remote object.

An ObjectPointer forwards all API calls to the remote. ObjectPointer objects point to objects. They exist to mimic the entire API of an object, but instead of computing a function locally (such as addition, subtraction, etc.) they forward the computation to a remote machine as specified by self.location. Specifically, every ObjectPointer has a object located somewhere that it points to (they should never exist by themselves). The objects being pointed to can be on the same machine or (more commonly) on a different one. Note further that a ObjectPointer does not know the nature how it sends messages to the object it points to (whether over socket, http, or some other protocol) as that functionality is abstracted in the BaseWorker object in self.location.

static create_pointer(*obj*, *location: AbstractWorker = None*, *id_at_location: str or int = None*, *register: bool = False*, *owner: AbstractWorker = None*, *ptr_id: str or int = None*, *garbage_collect_data=None*)

Creates a pointer to the "self" FrameworkTensor object.

This method is called on a FrameworkTensor object, returning a pointer to that object. This method is the CORRECT way to create a pointer, and the parameters of this method give all possible attributes that a pointer can be created with.

Parameters

- **location** – The AbstractWorker object which points to the worker on which this pointer's object can be found. In nearly all cases, this is self.owner and so this attribute can usually be left blank. Very rarely you may know that you are about to move the Tensor to another worker so you can pre-initialize the location attribute of the pointer to some other worker, but this is a rare exception.
- **id_at_location** – A string or integer id of the tensor being pointed to. Similar to location, this parameter is almost always self.id and so you can leave this parameter to None. The only exception is if you happen to know that the ID is going to be something different than self.id, but again this is very rare and most of the time, setting this means that you are probably doing something you shouldn't.
- **register** – A boolean parameter (default False) that determines whether to register the new pointer that gets created. This is set to false by default because most of the time a

pointer is initialized in this way so that it can be sent to someone else (i.e., "Oh you need to point to my tensor? let me create a pointer and send it to you"). Thus, when a pointer gets created, we want to skip being registered on the local worker because the pointer is about to be sent elsewhere. However, if you are initializing a pointer you intend to keep, then it is probably a good idea to register it, especially if there is any chance that someone else will initialize a pointer to your pointer.

- **owner** – A AbstractWorker parameter to specify the worker on which the pointer is located. It is also where the pointer is registered if register is set to True.
- **ptr_id** – A string or integer parameter to specify the id of the pointer in case you wish to set it manually for any special reason. Otherwise, it will be set randomly.
- **garbage_collect_data** – If true (default), delete the remote tensor when the pointer is deleted.
- **local_autograd** – Use autograd system on the local machine instead of PyTorch's autograd on the workers.
- **preinitialize_grad** – Initialize gradient for AutogradTensors to a tensor.

Returns A FrameworkTensor[ObjectPointer] pointer to self. Note that this object itself will likely be wrapped by a FrameworkTensor wrapper.

wrap(*self*, *register=True*, *type=None*, ***kwargs*)
Wraps the class inside framework tensor.

Because PyTorch/TF do not (yet) support functionality for creating arbitrary Tensor types (via subclassing torch.Tensor), in order for our new tensor types (such as PointerTensor) to be usable by the rest of PyTorch/TF (such as PyTorch's layers and loss functions), we need to wrap all of our new tensor types inside of a native PyTorch type.

This function adds a .wrap() function to all of our tensor types (by adding it to AbstractTensor), such that (on any custom tensor my_tensor), my_tensor.wrap() will return a tensor that is compatible with the rest of the PyTorch/TensorFlow API.

Returns A wrapper tensor of class *type*, or whatever is specified as default by the current syft.framework.Tensor.

classmethod handle_func_command(*cls*, *command*)
Receive an instruction for a function to be applied on a Pointer, Get the remote location to send the command, send it and get a pointer to the response, return. :param command: instruction of a function command: (command name, None, arguments[, kwargs]) :return: the response of the function command

classmethod find_a_pointer(*cls*, *command*)
Find and return the first pointer in the args object, using a trick with the raising error RemoteObjectFoundError

get(*self*, *user=None*, *reason: str = ''*, *deregister_ptr: bool = True*)
Requests the object being pointed to.

The object to which the pointer points will be requested, serialized and returned.

Note: This will typically mean that the remote object will be removed/destroyed.

Parameters

- **user** (*obj, optional*) – authenticate/allow user to perform get on remote private objects.

- **reason** (*str, optional*) – a description of why the data scientist wants to see it.
- **deregister_ptr** (*bool, optional*) – this determines whether to deregister this pointer from the pointer's owner during this method. This defaults to True because the main reason people use this method is to move the tensor from the location to the local one, at which time the pointer has no use.

Returns An AbstractObject object which is the tensor (or chain) that this object used to point to on a location.

TODO: add param get_copy which doesn't destroy remote if true.

__str__(*self*)
Returns a string version of this pointer.

This is primarily for end users to quickly see things about the object. This tostring shouldn't be used for anything else though as it's likely to change. (aka, don't try to parse it to extract information. Read the attribute you need directly). Also, don't use this to-string as a serialized form of the pointer.

__repr__(*self*)
Returns the to-string method.

When called using __repr__, most commonly seen when returned as cells in Jupyter notebooks.

__del__(*self*)
This method garbage collects the object this pointer is pointing to. By default, PySyft assumes that every object only has one pointer to it. Thus, if the pointer gets garbage collected, we want to automatically garbage collect the object being pointed to.

_create_attr_name_string(*self*, *attr_name*)

attr(*self*, *attr_name*)

setattr(*self*, *name*, *value*)

static simplify(*worker: AbstractWorker*, *ptr: ObjectPointer*)
This function takes the attributes of a ObjectPointer and saves them in a dictionary :param ptr: a ObjectPointer :type ptr: ObjectPointer

Returns a tuple holding the unique attributes of the pointer

Return type tuple

Examples

data = simplify(ptr)

static detail(*worker: AbstractWorker*, *object_tuple: tuple*)
This function reconstructs an ObjectPointer given it's attributes in form of a dictionary. We use the spread operator to pass the dict data as arguments to the init method of ObjectPointer :param worker: the worker doing the deserialization :param tensor_tuple: a tuple holding the attributes of the ObjectPointer

Returns an ObjectPointer

Return type *ObjectPointer*

Examples

ptr = detail(data)

1.1.1.3.1.28 syft.generic.pointers.object_wrapper

1.1.1.3.1.29 Module Contents

class syft.generic.pointers.object_wrapper.**ObjectWrapper**(*obj*, *id: int*, *tags: List[str] = None*, *description: str = None*)

A class that wraps an arbitrary object and provides it with an id, tags, and description

__call__(*self*, **args*, ***kwargs*)

__str__(*self*)

__repr__(*self*)

property obj(*self*)

static create_pointer(*object, owner: BaseWorker, location: BaseWorker, ptr_id: Union[int, str], id_at_location: Union[int, str] = None, garbage_collect_data=None, **kwargs*)

Creates a callable pointer to the object wrapper instance

Parameters

- **owner** – A BaseWorker parameter to specify the worker on which the pointer is located. It is also where the pointer is registered if register is set to True.
- **location** – The BaseWorker object which points to the worker on which this pointer's object can be found. In nearly all cases, this is self.owner and so this attribute can usually be left blank. Very rarely you may know that you are about to move the Tensor to another worker so you can pre-initialize the location attribute of the pointer to some other worker, but this is a rare exception.
- **ptr_id** – A string or integer parameter to specify the id of the pointer.
- **id_at_location** – A string or integer id of the object being pointed to. Similar to location, this parameter is almost always self.id and so you can leave this parameter to None. The only exception is if you happen to know that the ID is going to be something different than self.id, but again this is very rare and most of the time, setting this means that you are probably doing something you shouldn't.
- **garbage_collect_data** – If True, delete the remote object when the pointer is deleted.

Returns A pointers.CallablePointer pointer to self.

static simplify(*worker: AbstractWorker*, *obj: ObjectWrapper*)

static detail(*worker: AbstractWorker*, *obj_wrapper_tuple: str*)

1.1.1.3.1.30 syft.generic.pointers.pointer_plan

1.1.1.3.1.31 Module Contents

class syft.generic.pointers.pointer_plan.**PointerPlan**(*location: AbstractWorker = None*, *id_at_location: Union[str, int] = None*, *owner: AbstractWorker = None*, *garbage_collect_data: bool = True*, *id: Union[str, int] = None*, *tags: List[str] = None*, *description: str = None*)

Bases: *syft.generic.pointers.object_pointer.ObjectPointer*

The PointerPlan keeps a reference to a remote Plan.

It allows to: - __call__ an evaluation of the remote plan - get the remote plan

It's a simplification compared to the current hybrid state of Plans which can be seen as pointers, which is ambiguous.

property location(*self*)

property id_at_location(*self*)

__call__(*self*, **args*, ***kwargs*)
Transform the call on the pointer in a request to evaluate the remote plan

parameters(*self*)
Return a list of pointers to the plan parameters

request_run_plan(*self, location: sy.workers.BaseWorker, response_ids: List[Union[str, int]], *args, **kwargs*)
Requests plan execution.

Send a request to execute the plan on the remote location.

Parameters

- **location** – to which worker the request should be sent
- **response_ids** – where the result should be stored
- **args** – arguments used as input data for the plan
- **kwargs** – named arguments used as input data for the plan

Returns Execution response

get(*self*, *deregister_ptr: bool = True*)
This is an alias to fetch_plan, to behave like a pointer

static simplify(*worker: AbstractWorker*, *ptr: PointerPlan*)

static detail(*worker: AbstractWorker*, *tensor_tuple: tuple*)

wrap(*self*)

__str__(*self*)
Returns a string version of this pointer.

Example

For single pointers: > [PointerPlan | me:33873097403 -> dan:72165846784]

Or for multi pointers: > [PointerPlan | me:55894304374

-> alice:72165846784 -> bob:72165846784

]

__del__(*self*)
This method garbage collects the object this pointer is pointing to. By default, PySyft assumes that every object only has one pointer to it. Thus, if the pointer gets garbage collected, we want to automatically garbage collect the object being pointed to.

1.1.1.3.1.32 syft.generic.pointers.pointer_protocol

1.1.1.3.1.33 Module Contents

class syft.generic.pointers.pointer_protocol.**PointerProtocol**(*location: AbstractWorker = None*, *id_at_location: Union[str, int] = None*, *owner: AbstractWorker = None*, *garbage_collect_data: bool = True*, *id: Union[str, int] = None*, *tags: List[str] = None*, *description: str = None*)

Bases: *syft.generic.pointers.object_pointer.ObjectPointer*

The PointerProtocol keeps a reference to a remote Protocol.

It allows to: - run an evaluation of the remote protocol - get the remote protocol

run(*self*, **args*, ***kwargs*)
Call a run on the remote protocol

request_remote_run(*self*, *location: BaseWorker*, *args*, *kwargs*)
Requests remote protocol execution.

Send a request to execute the protocol on the remote location.

Parameters

- **location** – to which worker the request should be sent
- **args** – arguments used as input data for the protocol
- **kwargs** – named arguments used as input data for the protocol

Returns Execution response

get(*self*, *deregister_ptr: bool = True*)
This is an alias to fetch_protocol, to behave like a pointer

static simplify(*worker: AbstractWorker*, *ptr: PointerPlan*)

static detail(*worker: AbstractWorker*, *tensor_tuple: tuple*)

wrap(*self*)

1.1.1.3.1.34 syft.generic.pointers.pointer_tensor

1.1.1.3.1.35 Module Contents

class syft.generic.pointers.pointer_tensor.**PointerTensor**(*location: AbstractWorker = None*, *id_at_location: Union[str, int] = None*, *owner: AbstractWorker = None*, *id: Union[str, int] = None*, *garbage_collect_data: bool = True*, *shape: FrameworkShapeType = None*, *point_to_attr: str = None*, *tags: List[str] = None*, *description: str = None*)

Bases: *syft.generic.pointers.object_pointer.ObjectPointer*, *syft.generic.tensor.AbstractTensor*

A pointer to another tensor.

A PointerTensor forwards all API calls to the remote.PointerTensor objects point to tensors (as their name implies). They exist to mimic the entire API of a normal tensor, but instead of computing a tensor function locally (such as addition, subtraction, etc.) they forward the computation to a remote machine as specified by self.location. Specifically, every PointerTensor has a tensor located somewhere that it points to (they should never exist by themselves). Note that PointerTensor objects can point to both FrameworkTensor objects AND to other PointerTensor objects. Furthermore, the objects being pointed to can be on the same machine or (more commonly) on a different one. Note further that a PointerTensor does not know the nature how it sends messages to the tensor it points to (whether over socket, http, or some other protocol) as that functionality is abstracted in the AbstractWorker object in self.location.

Example

```
>>> import syft as sy
>>> hook = sy.TorchHook()
>>> bob = sy.VirtualWorker(id="bob")
>>> x = sy.Tensor([1,2,3,4,5])
>>> y = sy.Tensor([1,1,1,1,1])
>>> x_ptr = x.send(bob) # returns a PointerTensor, sends tensor to Bob
>>> y_ptr = y.send(bob) # returns a PointerTensor, sends tensor to Bob
>>> # executes command on Bob's machine
>>> z_ptr = x_ptr + y_ptr
```

fix_precision

float_precision

get_shape(*self*)

Request information about the shape to the remote worker

property shape(*self*)
This method returns the shape of the data being pointed to. This shape information SHOULD be cached on self._shape, but occasionally this information may not be present. If this is the case, then it requests the shape information from the remote object directly (which is inefficient and should be avoided).

property grad(*self*)

property data(*self*)

is_none(*self*)

clone(*self*)
Clone should keep ids unchanged, contrary to copy. We make the choice that a clone operation is local, and can't affect the remote tensors, so garbage_collect_data is always False, both for the tensor cloned and the clone.

get_class_attributes(*self*)
Used for cloning (see AbtractTensor)

static create_pointer(*tensor*, *location: AbstractWorker = None*, *id_at_location: str or int = None*, *register: bool = False*, *owner: AbstractWorker = None*, *ptr_id: str or int = None*, *garbage_collect_data=None*, *shape=None*)
Creates a pointer to the "self" FrameworkTensor object.

This method is called on a FrameworkTensor object, returning a pointer to that object. This method is the CORRECT way to create a pointer, and the parameters of this method give all possible attributes that a pointer can be created with.

Parameters

- **location** – The AbstractWorker object which points to the worker on which this pointer's object can be found. In nearly all cases, this is self.owner and so this attribute can usually be left blank. Very rarely you may know that you are about to move the Tensor to another worker so you can pre-initialize the location attribute of the pointer to some other worker, but this is a rare exception.
- **id_at_location** – A string or integer id of the tensor being pointed to. Similar to location, this parameter is almost always self.id and so you can leave this parameter to None. The only exception is if you happen to know that the ID is going to be something different than self.id, but again this is very rare and most of the time, setting this means that you are probably doing something you shouldn't.
- **register** – A boolean parameter (default False) that determines whether to register the new pointer that gets created. This is set to false by default because most of the time a pointer is initialized in this way so that it can be sent to someone else (i.e., "Oh you need to point to my tensor? let me create a pointer and send it to you"). Thus, when a pointer gets created, we want to skip being registered on the local worker because the pointer is about to be sent elsewhere. However, if you are initializing a pointer you intend to keep, then it is probably a good idea to register it, especially if there is any chance that someone else will initialize a pointer to your pointer.
- **owner** – A AbstractWorker parameter to specify the worker on which the pointer is located. It is also where the pointer is registered if register is set to True.
- **ptr_id** – A string or integer parameter to specify the id of the pointer in case you wish to set it manually for any special reason. Otherwise, it will be set randomly.
- **garbage_collect_data** – If true (default), delete the remote tensor when the pointer is deleted.

Returns A FrameworkTensor[PointerTensor] pointer to self. Note that this object itself will likely be wrapped by a FrameworkTensor wrapper.

move(*self*, *location*)

remote_send(*self*, *destination*, *change_location=False*)
Request the worker where the tensor being pointed to belongs to send it to destination. For instance, if C holds a pointer, ptr, to a tensor on A and calls ptr.remote_send(B), C will hold a pointer to a pointer on A which points to the tensor on B. If change_location is set to True, the original pointer will point to the moved object. Considering the same example as before with ptr.remote_send(B, change_location=True): C will hold a pointer to the tensor on B. We may need to be careful here because this pointer will have 2 references pointing to it.

remote_get(*self*)

get(*self*, *user=None*, *reason: str = ''*, *deregister_ptr: bool = True*)
Requests the tensor/chain being pointed to, be serialized and return

Since PointerTensor objects always point to a remote tensor (or chain of tensors, where a chain is simply a linked-list of tensors linked via their .child attributes), this method will request that the tensor/chain being pointed to be serialized and returned from this function.

Note: This will typically mean that the remote object will be removed/destroyed. To just bring a copy back to the local worker, call .copy() before calling .get().

Parameters

- **user** (*obj, optional*) – user credentials to perform authentication process.
- **reason** (*str, optional*) – a description of why the data scientist wants to see it.
- **deregister_ptr** (*bool, optional*) – this determines whether to deregister this pointer from the pointer's owner during this method. This defaults to True because the main reason people use this method is to move the tensor from the remote machine to the local one, at which time the pointer has no use.

Returns An AbstractTensor object which is the tensor (or chain) that this object used to point to #on a remote machine.

attr(*self*, *attr_name*)

dim(*self*)

fix_prec(*self*, **args*, ***kwargs*)
Send a command to remote worker to transform a tensor to fix_precision

Returns A pointer to an FixPrecisionTensor

float_prec(*self*, **args*, ***kwargs*)
Send a command to remote worker to transform a fix_precision tensor back to float_precision

Returns A pointer to a Tensor

share(*self*, **args*, ***kwargs*)
Send a command to remote worker to additively share a tensor

Returns A pointer to an AdditiveSharingTensor

keep(*self*, **args*, ***kwargs*)
Send a command to remote worker to keep a promise

Returns A pointer to a Tensor

value (*self*, **args*, ***kwargs*)
Send a command to remote worker to get the result generated by a promise.

Returns A pointer to a Tensor

share_ (*self*, **args*, ***kwargs*)
Send a command to remote worker to additively share inplace a tensor

Returns A pointer to an AdditiveSharingTensor

set_garbage_collect_data (*self*, *value*)

item (*self*)
Raising error with a message to be using .get instead of .item

__eq__ (*self*, *other*)

static simplify (*worker: AbstractWorker*, *ptr: PointerTensor*)
This function takes the attributes of a PointerTensor and saves them in a dictionary :param worker: the worker doing the serialization :type worker: AbstractWorker :param ptr: a PointerTensor :type ptr: PointerTensor

Returns a tuple holding the unique attributes of the pointer

Return type tuple

Examples

data = simplify(ptr)

static detail (*worker: AbstractWorker*, *tensor_tuple: tuple*)
This function reconstructs a PointerTensor given it's attributes in form of a dictionary. We use the spread operator to pass the dict data as arguments to the init method of PointerTensor :param worker: the worker doing the deserialization :param tensor_tuple: a tuple holding the attributes of the PointerTensor

Returns a PointerTensor

Return type *PointerTensor*

Examples

ptr = detail(data)

static bufferize (*worker: AbstractWorker*, *ptr: PointerTensor*)

static unbufferize (*worker: AbstractWorker*, *protobuf_tensor: PointerTensorPB*)

1.1.1.3.1.36 syft.generic.pointers.string_pointer

1.1.1.3.1.37 Module Contents

class syft.generic.pointers.string_pointer.**StringPointer**(*location: BaseWorker = None*, *id_at_location: Union[str, int] = None*, *owner: BaseWorker = None*, *id: Union[str, int] = None*, *garbage_collect_data: bool = True*, *tags: List[str] = None*, *description: str = None*)

Bases: *syft.generic.pointers.object_pointer.ObjectPointer*

This class defines a pointer to a 'String' object that might live on a remote machine. In other words, it holds a pointer to a 'String' object owned by a possibly different worker (although it can also point to a String owned by the same worker'.

All String method are hooked to objects of this class, and calls to such methods are forwarded to the pointed-to String object.

1.1.1.3.2 Submodules

1.1.1.3.2.1 syft.generic.id_provider

1.1.1.3.2.2 Module Contents

syft.generic.id_provider.**create_random_id**()

class syft.generic.id_provider.**IdProvider**(*given_ids=None*)

Provides Id to all syft objects.

Generate id and store the list of ids generated Can take a pre set list in input and will complete when it's empty.

An instance of IdProvider is accessible via sy.ID_PROVIDER.

pop(*self*, **args*)

Provides random ids and store them.

The syntax .pop() mimics the list syntax for convenience and not the generator syntax.

Returns Random Id.

set_next_ids(*self*, *given_ids: List*, *check_ids: bool = True*)

Sets the next ids returned by the id provider

Note that the ids are returned in reverse order of the list, as a pop() operation is applied.

Parameters

- **given_ids** – List, next ids returned by the id provider
- **check_ids** – bool, check whether these ids conflict with already generated ids

start_recording_ids(*self*)

Starts the recording in form of a list of the generated ids.

get_recorded_ids(*self*, *continue_recording=False*)
Returns the generated ids since the last call to start_recording_ids.

Parameters **continue_recording** – if False, the recording is stopped and the list of recorded ids is reset

Returns list of recorded ids

1.1.1.3.2.3 syft.generic.metrics

1.1.1.3.2.4 Module Contents

class syft.generic.metrics.**NetworkMonitor**
Provides utility for the monitoring of network traffic, measuring the packets sent through specific filters as passed by user.

static get_packets(*timeout=50*, *interface=None*, *bpf_filter=None*, *display_filter='tcp.port == 80'*, *tshark_path=None*, *output_file=None*)
Returns the captured packets of the transmitted data using Wireshark.

Args: timeout: An integer. Set for sniffing with tshark. Default to 50 seconds in this setup. interface: A string. Name of the interface to sniff on. bpf_filter: A string. The capture filter in bpf syntax 'tcp port 80'. Needs to be changed to match filter for the traffic sent. Not to be confused with the display filters (e.g. tcp.port == 80). The former are much more limited and is used to restrict the size of a raw packet capture, whereas the latter is used to hide some packets from the packet list. More info can be found at https://wiki.wireshark.org/CaptureFilters. display_filter: A string. Default to 'tcp.port == 80' (assuming this is the port of the 'WebsocketClientWorker'). Please see notes for 'bpf_filter' for details regarding differences. More info can be found at https://wiki.wireshark.org/DisplayFilters. tshark_path: Path to the tshark binary. E.g. '/usr/local/bin/tshark'. output_file: A string. Path including the output file name is to saved. E.g. '/tmp/mycapture.cap'

Returns: catpure: A 'pyshark.capture.live_capture.LiveCapture' object. Of packets sent over WebSockets. length: An integer. The number of packets captured at the network interface.

static read_packet(*index=None*, *capture_input=None*)
Reads the info of one packet returned by get_packets using pretty_print().

Args: index: An integer. The index of the packet to be examined.

Returns: pretty_print: The info of the packet chosen to be read.

1.1.1.3.2.5 syft.generic.object

1.1.1.3.2.6 Module Contents

class syft.generic.object.**AbstractObject**(*id: int = None*, *owner: sy.workers.AbstractWorker = None*, *tags: List[str] = None*, *description: str = None*, *child=None*)
Bases: abc.ABC

This is a generic object abstraction.

is_wrapper = False

__str__(*self*)

__repr__(*self*)

describe(*self*, *description: str*)

tag(*self*, **_tags: str*)

serialize(*self*)
Serializes the tensor on which it's called.

This is the high level convenience function for serializing torch tensors. It includes three steps, Simplify, Serialize, and Compress as described in serde.py. By default serde is compressing using LZ4

Returns

The serialized form of the tensor. For example:

x = torch.Tensor([1,2,3,4,5]) x.serialize() # returns a serialized object

ser(*self*, **args*, ***kwargs*)

get(*self*)
Just a pass through. This is most commonly used when calling .get() on a Syft tensor which has a child which is a pointer, an additive shared tensor, a multi-pointer, etc.

mid_get(*self*)
This method calls .get() on a child pointer and correctly registers the results

get_class_attributes(*self*)
Return all elements which defines an instance of a certain class. By default there is nothing so we return an empty dict, but for example for fixed precision tensor, the fractional precision is very important.

classmethod on_function_call(*cls*, **args*)
Override this to perform a specific action for each call of a torch function with arguments containing syft tensors of the class doing the overloading

classmethod handle_func_command(*cls*, *command*)
Receive an instruction for a function to be applied on a Syft Tensor, Replace in the args all the LogTensors with their child attribute, forward the command instruction to the handle_function_command of the type of the child attributes, get the response and replace a Syft Tensor on top of all tensors found in the response.

Parameters

- **command** – instruction of a function command: (command name,
- **self>, arguments[, kwargs])** (*<no*) –

Returns the response of the function command

classmethod rgetattr(*cls*, *obj*, *attr*, **args*)
Get an attribute recursively.

This is a core piece of functionality for the PySyft tensor chain.

Parameters

- **obj** – the object holding the attribute
- **attr** – nested attribute
- **args** – optional arguments to provide

Returns the attribute obj.attr

Example

```
>>> rgetattr(obj, 'attr1.attr2.attr3')
[Out] obj.attr1.attr2.attr3
```

syft.generic.object.**initialize_object**(*hook*, *obj*, *owner=None*, *reinitialize=True*, *id=None*, *init_args=tuple()*, *init_kwargs={}*)

Initializes the tensor.

Parameters

- **hook** – A reference to TorchHook class.
- **obj** – An object to keep track of id, owner and whether it is a native tensor or a wrapper over pytorch.
- **reinitialize** – A boolean parameter (default True) to indicate whether to re-execute __init__.
- **owner** – The owner of the tensor being initialised, leave it blank to if you have already provided a reference to TorchHook class.
- **id** – The id of tensor, a random id will be generated if there is no id specified.

syft.generic.object.**_apply_args**(*hook*, *obj_to_register*, *owner=None*, *id=None*)

1.1.1.3.2.7 syft.generic.object_storage

1.1.1.3.2.8 Module Contents

class syft.generic.object_storage.**ObjectStorage**

A storage of objects identifiable by their id.

A wrapper object to a collection of objects where all objects are stored using their IDs as keys.

register_obj(*self*, *obj: object*, *obj_id: Union[str, int] = None*)

Registers the specified object with the current worker node.

Selects an id for the object, assigns a list of owners, and establishes whether it's a pointer or not. This method is generally not used by the client and is instead used by internal processes (hooks and workers).

Parameters

- **obj** – A torch Tensor or Variable object to be registered.
- **obj_id** (*int or string*) – random integer between 0 and 1e10 or
- **uniquely identifying the object.** (*string*) –

de_register_obj(*self*, *obj: object*, *_recurse_torch_objs: bool = True*)

Deregisters the specified object.

Deregister and remove attributes which are indicative of registration.

Parameters

- **obj** – A torch Tensor or Variable object to be deregistered.
- **_recurse_torch_objs** – A boolean indicating whether the object is more complex and needs to be explored. Is not supported at the moment.

get_obj(*self, obj_id: Union[str, int]*)
Returns the object from registry.

Look up an object from the registry using its ID.

Parameters **obj_id** – A string or integer id of an object to look up.

Returns Object with id equals to *obj_id*.

set_obj(*self, obj: Union[FrameworkTensorType, AbstractTensor]*)
Adds an object to the registry of objects.

Parameters **obj** – A torch or syft tensor with an id.

rm_obj(*self, remote_key: Union[str, int]*)
Removes an object.

Remove the object from the permanent object registry if it exists.

Parameters **remote_key** – A string or integer representing id of the object to be removed.

force_rm_obj(*self, remote_key: Union[str, int]*)
Forces object removal.

Besides removing the object from the permanent object registry if it exists. Explicitly forces removal of the object modifying the *garbage_collect_data* attribute.

Parameters **remote_key** – A string or integer representing id of the object to be removed.

clear_objects(*self*, *return_self: bool = True*)
Removes all objects from the object storage.

Note: the "return self" statement is kept in order to avoid modifying the code shown in the udacity course.

Parameters **return_self** – flag, whether to return self as return value

Returns self, if return_self if True, else None

current_objects(*self*)
Returns a copy of the objects in the object storage.

1.1.1.3.2.9 syft.generic.string

1.1.1.3.2.10 Module Contents

class syft.generic.string.**String**(*object: object = None*, *encoding: str = None*, *errors: str = None*, *id: Union[int, str] = None*, *owner: BaseWorker = None*, *tags: List[str] = None*, *description: str = None*)
Bases: *syft.generic.object.AbstractObject*

This is a class that wraps the Python built-in *str* class. In addition to providing access to most of *str*'s method call API, it allows sending such wrapped string between workers the same way Syft tensors can be moved around among workers.

methods_to_hook

send(*self*, *location: BaseWorker*)
Sends this String object to the worker specified by 'location'. and returns a pointer to that string as a StringPointer object.

Parameters **location** – The BaseWorker object which you want to send this object to. Note that this is never actually the BaseWorker but instead a class which inherits the BaseWorker abstraction.

Returns A StringPointer objects to self.

get_class_attributes(*self*)
returns minimal necessary keyword arguments to create a String object

on(*self*, *object: str*, *wrap=False*)
Takes and object of type strings and assigns it to self.child

__add__(*self, other: Union[str, 'String']*)
[Important] overriding the *__add__* here is not yet activated. The real hooking happens in syft/generic/frameworks/hook/hook.py. Hooking as implemented here (using @overloaded.method) is to be activated when hook_args.py is adapted to wrapping reponses of *str* types into *String* types. This is not yet supported.

add_string(*self*, *_self: String*, *other: String*)
This method is created in a way adapted to the logic implemented in hook_args.py. That is, it can be wrapped with the decorator @oerloaded.method.

hook_args.py args hooking logic needs that the data types of argument be unchanged. For instance, 'other' should always be of a fixed type 'String' or 'str' but not alternating between both. This can cause unexpected behaviou due to caching in hook_args.py.

Parameters

- **_self** – a String object (as received by the decorator). It represents the objects on which we called the add method. It will always be of type *str* inside this method. Since the decorator methods strips the *str* out of the *String* object.
- **other** – a String object that we wish to concatenate to *_self*. Same as above, it is a String object as received by the decorator but here it will always be of type *str*.

Returns The concatentenated *str* object between *_self* and *other*. this *str* object will be wrapped by the decorator into a String object

static create_pointer(*obj*, *location: BaseWorker = None*, *id_at_location: str or int = None*, *register: bool = False*, *owner: BaseWorker = None*, *ptr_id: str or int = None*, *garbage_collect_data: bool = True*)
Creates a StringPointer object that points to a String object 'obj' after sending the latter to the worker 'location'.

Returns a StringPointer object

static simplify(*worker: BaseWorker*, *string: String*)
Breaks String object into a tuple of simpler objects, its constituting objects that are serializable.

Parameters

- **worker** – a BaseWorker object
- **string** – the String object to be simplified

Returns A tuple of simpler objects that are sufficient to recreate a String object that is a clone of *string*.

static detail(*worker: BaseWorker*, *simple_obj: Tuple*)
Create an object of type String from the reduced representation in *simple_obj*.

Parameters

- **worker** – BaseWorker The worker on which the new String object is to be created.
- **simple_obj** – tuple A tuple resulting from the serialized then deserialized returned tuple from the *simplify* static method above.

Returns A String object

1.1.1.3.2.11 syft.generic.tensor

1.1.1.3.2.12 Module Contents

class syft.generic.tensor.**AbstractTensor**(*id: int = None, owner: sy.workers.AbstractWorker = None, tags: List[str] = None, description: str = None, child=None*)

Bases: *syft.generic.object.AbstractObject*

wrap(*self*, *register=True*, *type=None*, ***kwargs*)

Wraps the class inside an empty object of class *type*.

Because PyTorch/TF do not (yet) support functionality for creating arbitrary Tensor types (via subclassing torch.Tensor), in order for our new tensor types (such as PointerTensor) to be usable by the rest of PyTorch/TF (such as PyTorch's layers and loss functions), we need to wrap all of our new tensor types inside of a native PyTorch type.

This function adds a .wrap() function to all of our tensor types (by adding it to AbstractTensor), such that (on any custom tensor my_tensor), my_tensor.wrap() will return a tensor that is compatible with the rest of the PyTorch/TensorFlow API.

Returns A wrapper tensor of class *type*, or whatever is specified as default by the current syft.framework.Tensor.

on(*self*, *tensor: AbstractTensor*, *wrap: bool = True*)

Add a syft(log) tensor on top of the tensor.

Parameters

- **tensor** – the tensor to extend
- **wrap** – if true, add the syft tensor between the wrapper
- **the rest of the chain. If false, just add it at the top**(*and*) –

Returns a syft/torch tensor

copy(*self*)

clone(*self*)

Clone should keep ids unchanged, contrary to copy

refresh(*self*)

Forward to Additive Shared Tensor the call to refresh shares

property shape(*self*)

__len__(*self*)

Alias .shape[0] with len(), helpful for pointers

property grad(*self*)

syft.generic.tensor.**initialize_tensor**(*hook*, *obj*, *owner=None*, *id=None*, *init_args=tuple()*, *init_kwargs={}*)

Initializes the tensor.

Parameters

- **hook** – A reference to TorchHook class.
- **cls** – An object to keep track of id, owner and whether it is a native tensor or a wrapper over pytorch.

- **is_tensor** – A boolean parameter (default False) to indicate whether it is torch tensor or not.
- **owner** – The owner of the tensor being initialised, leave it blank to if you have already provided a reference to TorchHook class.
- **id** – The id of tensor, a random id will be generated if there is no id specified.

1.1.1.4 syft.grid

1.1.1.4.1 Subpackages

1.1.1.4.1.1 syft.grid.authentication

1.1.1.4.1.2 Submodules

1.1.1.4.1.3 syft.grid.authentication.account

1.1.1.4.1.4 Module Contents

class syft.grid.authentication.account.**AccountCredential**(*username*, *password*)

Bases: *syft.grid.authentication.credential.AbstractCredential*

Parse/represent credentials based on username-password structure. Expected JSON Format: { "accounts": [{"user": "example1", "password": "pass_example"},

{"user": "user2", "password": "password2"},

]

}

USERNAME_FIELD = username

PASSWORD_FIELD = password

CREDENTIAL_FIELD = accounts

static parse(*path: str*, *file_name: str*)

Static method used to create new account authentication instances parsing a json file. :param path: Json file path. :type path: str :param file_name: File's name. :type file_name: str

Returns List of account objects.

Return type List

json(*self*)

Convert account instances into a JSON/Dictionary structure.

__str__(*self*)

1.1.1.4.1.5 syft.grid.authentication.credential

1.1.1.4.1.6 Module Contents

class syft.grid.authentication.credential.**AbstractCredential**
Bases: abc.ABC

AbstractCredential is an abstract class that defines generic methods used by all types of authentications defined in this module.

abstract parse(*self*)
Read, parse and load credential files.

abstract json(*self*)
Convert credential instances into a JSON structure.

1.1.1.4.2 Submodules

1.1.1.4.2.1 syft.grid.abstract_grid

1.1.1.4.2.2 Module Contents

class syft.grid.abstract_grid.**AbstractGrid**
Bases: abc.ABC

abstract search(*self, *query: Union[str]*)

abstract serve_model(*self*, *model*, *id: str*, *mpc: bool = False*, *allow_remote_inference: bool = False*, *allow_download: bool = False*, *n_replica: int = 1*)

abstract query_model_hosts(*self*, *id: str*, *mpc: bool = False*)

abstract run_remote_inference(*self*, *id: str*, *data: torch.Tensor*, *mpc: bool = False*)

_connect_all_nodes(*self, nodes: Tuple[Any], node_type: Any*)
Connect all nodes to each other. :param nodes: A tuple of grid clients.

_check_node_type(*self, grid_workers: List[Any], node_type: Any*)
Private method used to verify if workers used by grid network are exactly what we expect. :returns: Boolean result. :rtype: result

1.1.1.4.2.3 syft.grid.private_grid

1.1.1.4.2.4 Module Contents

class syft.grid.private_grid.**PrivateGridNetwork**(**workers*)
Bases: *syft.grid.abstract_grid.AbstractGrid*

search(*self*, **query*)
Searches over a collection of workers, returning pointers to the results grouped by worker. :param query: List of tags used to identify the desired tensor.

Returns list of pointers with pointers that matches with tags.

Return type results

serve_model(*self*, *model*, *id: str*, *mpc: bool = False*, *allow_remote_inference: bool = False*, *allow_download: bool = False*, *n_replica: int = 1*)

Choose some node(s) on grid network to host a unencrypted / encrypted model. :param model: Model to be hosted. :param id: Model's ID. :param mpc: Boolean flag to host a plain text / encrypted model :param allow_remote_inference: Allow to run inference remotely. :param allow_download: Allow to copy the model and run it locally. :param n_replica: Number of copies distributed through grid network.

Raises

- **RuntimeError** – If grid network doesn't have enough nodes to replicate the model.
- **NotImplementedError** – If workers used by grid network aren't grid nodes.

run_remote_inference(*self*, *id: str*, *data: torch.Tensor*, *mpc: bool = False*)

Search for a specific model registered on grid network, if found, It will run inference. :param id: Model's ID. :param dataset: Data used to run inference. :param mpc: Boolean flag to run a plain text / encrypted model

Returns Inference's result.

Return type Tensor

Raises

- **NotImplementedError** – If workers used by grid network aren't grid nodes.
- **RuntimeError** – If model id not found.

query_model_hosts(*self*, *id: str*, *mpc: bool = False*)

Search for node host from a specific model registered on grid network, if found, It will return the frist host/ set of hosts that contains the desired model. :param id: Model's ID. :param data: Data used to run inference. :param mpc: Boolean flag to search for a plain text / encrypted model

Returns First worker that contains the desired model.

Return type workers

Raises

- **NotImplementedError** – If workers used by grid network aren't grid nodes.
- **RuntimeError** – If model id not found.

_host_encrypted_model(*self*, *model*, *n_shares: int = 4*)

This method wiil choose some grid nodes at grid network to host an encrypted model.

Parameters

- **model** – Model to be hosted.
- **n_shares** – number of workers used by MPC protocol.

Raise: RuntimeError : If grid network doesn't have enough workers to host an encrypted model or if model is not a plan.

_query_encrypted_model_hosts(*self*, *id: str*)

Search for an encrypted model and return its mpc nodes.

Parameters **id** – Model's ID.

Returns Tuple structure containing Host, MPC Nodes and crypto provider.

Return type Tuple

Raises **RuntimeError** – If model id not found.

_run_unencrypted_inference(*self*, *id: str*, *data*)
Search for a plain-text model registered on grid network, if found, It will run inference. :param id: Model's ID. :param dataset: Data used to run inference.

Returns Inference's result.

Return type Tensor

Raises **RuntimeError** – If model id not found.

_run_encrypted_inference(*self*, *id: str*, *data*)
Search for an encrypted model and perform inference. :param model_id: Model's ID. :param data: Dataset to be shared/inferred. :param copy: Boolean flag to perform encrypted inference without lose plan.

Returns Inference's result.

Return type Tensor

Raises **RuntimeError** – If model id not found.

1.1.1.4.2.5 syft.grid.public_grid

1.1.1.4.2.6 Module Contents

class syft.grid.public_grid.**PublicGridNetwork**(*hook*, *gateway_url: str*, *credential: AbstractCredential = None*)
Bases: *syft.grid.abstract_grid.AbstractGrid*

search(*self, *query: Union[str]*)
Search a set of tags across the grid network.

Parameters **query** – A set of dataset tags.

Returns matrix of tensor pointers.

Return type tensor_results

serve_model(*self, model, id: Union[str, int], mpc: bool = False, allow_remote_inference: bool = False, allow_download: bool = False, n_replica: int = 1*)
Choose n (number of replicas defined at gateway) grid nodes registered in the grid network to host a model. :param model: Model to be hosted. :param id: Model's ID. :param mpc: Boolean flag to serve plan models in an encrypted/unencrypted format. :param allow_remote_inference: Allow workers to run inference in this model. :param allow_download: Allow workers to copy the model and run it locally.

query_model_hosts(*self*, *id: str*, *mpc: bool = False*)
This method will search for a specific model registered on grid network, if found, It will return all grid nodes that contains the desired model. :param id: Model's ID. :param mpc: Boolean flag to search plan models in an encrypted/unencrypted format.

Returns Worker / list of workers that contains the desired model.

Return type workers

Raises

- **RuntimeError** – If grid network doesn't have enough workers to host
- **an encrypted model, or if model isn't a plan.** –

run_remote_inference(*self*, *id: str*, *data: torch.Tensor*, *mpc: bool = False*)
This method will search for a specific model registered on the grid network, if found, It will run inference. :param id: Model's ID. :param dataset: Data used to run inference. :param mpc: Boolean flag to run encrypted/unencrypted inferences.

Returns Inference's result.

Return type Tensor

Raises **RuntimeError** – If model id not registered on the grid network.

_serve_unencrypted_model(*self*, *model*, *id*, *allow_remote_inference: bool*, *allow_download: bool*)
This method will choose one of grid nodes registered in the grid network to host a plain text model. :param model: Model to be hosted. :param id: Model's ID. :param allow_remote_inference: Allow workers to run inference in this model. :param allow_download: Allow workers to copy the model and run it locally.

_serve_encrypted_model(*self*, *model*)
This method wiil choose some grid nodes at grid network to host an encrypted model.

Parameters **model** – Model to be hosted.

Raises

- **RuntimeError** – If grid network doesn't have enough workers to host
- **an encrypted model, or if model isn't a plan.** –

_query_unencrypted_models(*self*, *id*)
Search for a specific model registered on grid network, if found, It will return the first node that contains the desired model. :param id: Model's ID.

Returns worker that contains the desired model.

Return type worker

_query_encrypted_models(*self*, *id*)
Search for a specific encrypted model registered on grid network, if found, It will return the first node that hosts the desired model and mpc shares. :param id: Model's ID.

Returns List of workers that contains the desired mpc model.

Return type workers

_run_unencrypted_inference(*self*, *id*, *data*)
Search for an unencrypted model and perform data inference. :param id: Model's ID. :param data: Dataset to be inferred.

Returns Inference's result.

Return type Tensor

Raises **RuntimeError** – If model if not found.

_run_encrypted_inference(*self*, *id*, *data*, *copy=True*)
Search for an encrypted model and perform inference.

Parameters

- **model_id** – Model's ID.
- **data** – Dataset to be shared/inferred.
- **copy** – Boolean flag to perform encrypted inference without lose plan.

Returns Inference's result.

Return type Tensor

Raises **RuntimeError** – If model id not found.

__connect_with_node(*self*, *node_id*, *node_url*)

_ask_gateway(*self*, *request_method*, *endpoint: str*, *body: Dict = {}*)

1.1.1.5 syft.messaging

1.1.1.5.1 Subpackages

1.1.1.5.1.1 syft.messaging.plan

1.1.1.5.1.2 Submodules

1.1.1.5.1.3 syft.messaging.plan.plan

1.1.1.5.1.4 Module Contents

class syft.messaging.plan.plan.**func2plan**(*args_shape=None*, *state=None*)
Bases: object

Decorator which converts a function to a plan.

Converts a function containing sequential pytorch code into a plan object which can be sent to any arbitrary worker.

This class should be used only as a decorator.

__call__(*self*, *plan_function*)

syft.messaging.plan.plan.**method2plan**(**args*, ***kwargs*)

class syft.messaging.plan.plan.**Plan**(*name: str = None, state: State = None, include_state: bool = False, is_built: bool = False, operations: List[Operation] = None, placeholders: Dict[Union[str, int], PlaceHolder] = None, forward_func=None, state_tensors=None, id: Union[str, int] = None, owner: sy.workers.BaseWorker = None, tags: List[str] = None, description: str = None*)
Bases: *syft.generic.object.AbstractObject*, *syft.generic.object_storage.ObjectStorage*

A Plan stores a sequence of torch operations, just like a function.

A Plan is intended to store a sequence of torch operations, just like a function, but it allows to send this sequence of operations to remote workers and to keep a reference to it. This way, to compute remotely this sequence of operations on some remote input referenced through pointers, instead of sending multiple messages you need now to send a single message with the references of the plan and the pointers.

All arguments are optional.

Parameters

- **name** – the name of the name
- **state** – store the plan tensors like model parameters
- **include_state** – if true, implies that the plan is a function, else a class. If true, the state is re-integrated in the args to be accessed within the function
- **is_built** – state if the plan has already been built.
- **placeholders** – dict of placeholders used in the plan

- **operations** – list of commands (called operations)
- **forward_func** – the function to be transformed into a plan
- **state_tensors** – a tuple of state elements. It can be used to populate a state
- **id** – plan id
- **owner** – plan owner
- **tags** – plan tags
- **description** – plan description

get

share

static _create_placeholders(*args_shape*)

property _known_workers(*self*)

property location(*self*)

property readable_plan(*self*)

parameters(*self*)
This is defined to match the torch api of nn.Module where .parameters() return the model tensors / parameters

send_msg(*self*, **args*, ***kwargs*)

request_obj(*self*, **args*, ***kwargs*)

respond_to_obj_req(*self, obj_id: Union[str, int]*)
Returns the deregistered object from registry.

Parameters **obj_id** – A string or integer id of an object to look up.

add_placeholder(*self*, *tensor*, *node_type=None*)
Create and register a new placeholder if not already existing (else return the existing one).

The placeholder is tagged by a unique and incremental index for a given plan.

Parameters

- **tensor** – the tensor to replace with a placeholder
- **node_type** – Should be "input" or "output", used to tag like this: #<type>-*

replace_with_placeholders(*self*, *obj*, ***kw*)
Replace in an object all FrameworkTensors with Placeholders

find_placeholders(*self*, **search_tags*)
Search method to retrieve placeholders used in the Plan using tag search. Retrieve all placeholders which have a tag containing at least one search_tag.

Parameters ***search_tags** – tuple of tags

Returns A list of placeholders found

build(*self*, **args*)
Builds the plan.

First, run the function to be converted in a plan in a context which activates the tracing and record the operations in trace.logs

Second, store the result ids temporarily to helper ordering the output placeholders at return time

Third, loop through the trace logs and replace the tensors found in the operations logged by PlaceHolders. Record those operations in plan.operations

Parameters **args** – Input arguments to run the plan

copy(*self*)
Creates a copy of a plan.

__setattr__(*self*, *name*, *value*)
Add new tensors or parameter attributes to the state and register them in the owner's registry

__call__(*self*, **args*, ***kwargs*)
Calls a plan execution with some arguments.

When possible, run the original function to improve efficiency. When it's not, for example if you fetched the plan from a remote worker, then run it from the tape of operations: - Instantiate input placeholders - for each recorded operation, run the operation on the placeholders

and use the result(s) to instantiate to appropriate placeholder.

- Return the instantiation of all the output placeholders.

run(*self, args: Tuple, result_ids: List[Union[str, int]]*)
Controls local or remote plan execution. If the plan doesn't have the plan built, first build it using the original function.

Parameters

- **args** – Arguments used to run plan.
- **result_ids** – List of ids where the results will be stored.

send(*self*, **locations: AbstractWorker*, *force=False*)
Send plan to locations.

If the plan was not built locally it will raise an exception. If *force* = true plan is going to be sent either way.

Parameters

- **locations** – List of workers.
- **force** – A boolean indicating if this operation should be forced.

get_(*self*)

get_pointers(*self*)

fix_precision_(*self*, **args*, ***kwargs*)

float_precision_(*self*)

share_(*self*, **args*, ***kwargs*)

create_pointer(*self*, *owner*, *garbage_collect_data*, *location=None*, *id_at_location=None*, ***kwargs*)

__str__(*self*)
Returns the string representation of Plan.

__repr__(*self*)

static simplify(*worker: AbstractWorker*, *plan: Plan*)
This function takes the attributes of a Plan and saves them in a tuple :param worker: the worker doing the serialization :type worker: AbstractWorker :param plan: a Plan object :type plan: Plan

Returns a tuple holding the unique attributes of the Plan object

Return type tuple

static detail(*worker: AbstractWorker*, *plan_tuple: tuple*)
This function reconstructs a Plan object given its attributes in the form of a tuple. :param worker: the worker doing the deserialization :param plan_tuple: a tuple holding the attributes of the Plan

Returns a Plan object

Return type plan

static bufferize(*worker: AbstractWorker*, *plan: Plan*)
This function takes the attributes of a Plan and saves them in a Protobuf message :param worker: the worker doing the serialization :type worker: AbstractWorker :param plan: a Plan object :type plan: Plan

Returns a Protobuf message holding the unique attributes of the Plan object

Return type PlanPB

static unbufferize(*worker: AbstractWorker*, *protobuf_plan: PlanPB*)
This function reconstructs a Plan object given its attributes in the form of a Protobuf message :param worker: the worker doing the deserialization :param protobuf_plan: a Protobuf message holding the attributes of the Plan

Returns a Plan object

Return type plan

syft.messaging.plan.plan.**tag_sort**(*keyword*)
Utility function to sort tensors by their (unique) tag including "keyword"

1.1.1.5.1.5 syft.messaging.plan.state

1.1.1.5.1.6 Module Contents

class syft.messaging.plan.state.**State**(*owner*, *state_placeholders=None*)
Bases: object

The State is a Plan attribute and is used to send tensors along functions.

It references Plan tensor or parameters attributes using their name, and make sure they are provided to remote workers who are sent the Plan.

__str__(*self*)
Returns the string representation of the State.

__repr__(*self*)

tensors(*self*)
Fetch and return all the state elements.

copy(*self*)

read(*self*)
Return state tensors that are from this plan specifically, but not those of plans including in this plan. If run while a plan is building, declare all the state tensors to the plan currently building.

static create_grad_if_missing(*tensor*)

fix_precision_(*self*, **args*, ***kwargs*)

float_precision_(*self*)

share_(*self*, **args*, ***kwargs*)

get_(*self*)
Get functionality that can only be used when getting back state elements converted to additive shared tensors. Other than this, you shouldn't need to the get the state separately.

static simplify(*worker: AbstractWorker*, *state: State*)
Simplify the plan's state when sending a plan

static detail(*worker: AbstractWorker*, *state_tuple: tuple*)
Reconstruct the plan's state from the state elements and supposed ids.

static bufferize(*worker: AbstractWorker*, *state: State*)
Serialize the State to Protobuf message

static unbufferize(*worker: AbstractWorker*, *protobuf_state: StatePB*)
Reconstruct the plan's state from the state elements and supposed ids.

1.1.1.5.1.7 Package Contents

class syft.messaging.plan.**func2plan**(*args_shape=None*, *state=None*)
Bases: object

Decorator which converts a function to a plan.

Converts a function containing sequential pytorch code into a plan object which can be sent to any arbitrary worker.

This class should be used only as a decorator.

__call__(*self*, *plan_function*)

syft.messaging.plan.**method2plan**(**args*, ***kwargs*)

class syft.messaging.plan.**Plan**(*name: str = None, state: State = None, include_state: bool = False, is_built: bool = False, operations: List[Operation] = None, placeholders: Dict[Union[str, int], PlaceHolder] = None, forward_func=None, state_tensors=None, id: Union[str, int] = None, owner: sy.workers.BaseWorker = None, tags: List[str] = None, description: str = None*)
Bases: *syft.generic.object.AbstractObject*, *syft.generic.object_storage.ObjectStorage*

A Plan stores a sequence of torch operations, just like a function.

A Plan is intended to store a sequence of torch operations, just like a function, but it allows to send this sequence of operations to remote workers and to keep a reference to it. This way, to compute remotely this sequence of operations on some remote input referenced through pointers, instead of sending multiple messages you need now to send a single message with the references of the plan and the pointers.

All arguments are optional.

Parameters

- **name** – the name of the name
- **state** – store the plan tensors like model parameters
- **include_state** – if true, implies that the plan is a function, else a class. If true, the state is re-integrated in the args to be accessed within the function
- **is_built** – state if the plan has already been built.
- **placeholders** – dict of placeholders used in the plan

- **operations** – list of commands (called operations)
- **forward_func** – the function to be transformed into a plan
- **state_tensors** – a tuple of state elements. It can be used to populate a state
- **id** – plan id
- **owner** – plan owner
- **tags** – plan tags
- **description** – plan description

get

share

static _create_placeholders(*args_shape*)

property _known_workers(*self*)

property location(*self*)

property readable_plan(*self*)

parameters(*self*)
This is defined to match the torch api of nn.Module where .parameters() return the model tensors / parameters

send_msg(*self*, **args*, ***kwargs*)

request_obj(*self*, **args*, ***kwargs*)

respond_to_obj_req(*self, obj_id: Union[str, int]*)
Returns the deregistered object from registry.

Parameters **obj_id** – A string or integer id of an object to look up.

add_placeholder(*self*, *tensor*, *node_type=None*)
Create and register a new placeholder if not already existing (else return the existing one).

The placeholder is tagged by a unique and incremental index for a given plan.

Parameters

- **tensor** – the tensor to replace with a placeholder
- **node_type** – Should be "input" or "output", used to tag like this: #<type>-*

replace_with_placeholders(*self*, *obj*, ***kw*)
Replace in an object all FrameworkTensors with Placeholders

find_placeholders(*self*, **search_tags*)
Search method to retrieve placeholders used in the Plan using tag search. Retrieve all placeholders which have a tag containing at least one search_tag.

Parameters ***search_tags** – tuple of tags

Returns A list of placeholders found

build(*self*, **args*)
Builds the plan.

First, run the function to be converted in a plan in a context which activates the tracing and record the operations in trace.logs

Second, store the result ids temporarily to helper ordering the output placeholders at return time

Third, loop through the trace logs and replace the tensors found in the operations logged by PlaceHolders. Record those operations in plan.operations

Parameters **args** – Input arguments to run the plan

copy(*self*)
Creates a copy of a plan.

__setattr__(*self*, *name*, *value*)
Add new tensors or parameter attributes to the state and register them in the owner's registry

__call__(*self*, **args*, ***kwargs*)
Calls a plan execution with some arguments.

When possible, run the original function to improve efficiency. When it's not, for example if you fetched the plan from a remote worker, then run it from the tape of operations: - Instantiate input placeholders - for each recorded operation, run the operation on the placeholders

and use the result(s) to instantiate to appropriate placeholder.

- Return the instantiation of all the output placeholders.

run(*self, args: Tuple, result_ids: List[Union[str, int]]*)
Controls local or remote plan execution. If the plan doesn't have the plan built, first build it using the original function.

Parameters

- **args** – Arguments used to run plan.
- **result_ids** – List of ids where the results will be stored.

send(*self*, **locations: AbstractWorker*, *force=False*)
Send plan to locations.

If the plan was not built locally it will raise an exception. If *force* = true plan is going to be sent either way.

Parameters

- **locations** – List of workers.
- **force** – A boolean indicating if this operation should be forced.

get_(*self*)

get_pointers(*self*)

fix_precision_(*self*, **args*, ***kwargs*)

float_precision_(*self*)

share_(*self*, **args*, ***kwargs*)

create_pointer(*self*, *owner*, *garbage_collect_data*, *location=None*, *id_at_location=None*, ***kwargs*)

__str__(*self*)
Returns the string representation of Plan.

__repr__(*self*)

static simplify(*worker: AbstractWorker*, *plan: Plan*)
This function takes the attributes of a Plan and saves them in a tuple :param worker: the worker doing the serialization :type worker: AbstractWorker :param plan: a Plan object :type plan: Plan

Returns a tuple holding the unique attributes of the Plan object

Return type tuple

static detail(*worker: AbstractWorker*, *plan_tuple: tuple*)
This function reconstructs a Plan object given its attributes in the form of a tuple. :param worker: the worker doing the deserialization :param plan_tuple: a tuple holding the attributes of the Plan

Returns a Plan object

Return type plan

static bufferize(*worker: AbstractWorker*, *plan: Plan*)
This function takes the attributes of a Plan and saves them in a Protobuf message :param worker: the worker doing the serialization :type worker: AbstractWorker :param plan: a Plan object :type plan: Plan

Returns a Protobuf message holding the unique attributes of the Plan object

Return type PlanPB

static unbufferize(*worker: AbstractWorker*, *protobuf_plan: PlanPB*)
This function reconstructs a Plan object given its attributes in the form of a Protobuf message :param worker: the worker doing the deserialization :param protobuf_plan: a Protobuf message holding the attributes of the Plan

Returns a Plan object

Return type plan

1.1.1.5.2 Submodules

1.1.1.5.2.1 syft.messaging.message

This file exists as the Python encoding of all Message types that Syft sends over the network. It is an important bottleneck in the system, impacting both security, performance, and cross-platform compatability. As such, message types should strive to not be framework specific (i.e., Torch, Tensorflow, etc.).

All Syft message types extend the Message class.

1.1.1.5.2.2 Module Contents

class syft.messaging.message.**Message**(*contents=None*)
All syft message types extend this class

All messages in the pysyft protocol extend this class. This abstraction requires that every message has an integer type, which is important because this integer is what determines how the message is handled when a BaseWorker receives it.

Additionally, this type supports a default simplifier and detailer, which are important parts of PySyft's serialization and deserialization functionality. You can read more abouty detailers and simplifiers in syft/serde/serde.py.

property contents(*self*)
Return a tuple with the contents of the message (backwards compatability)

Some of our codebase still assumes that all message types have a .contents attribute. However, the contents attribute is very opaque in that it doesn't put any constraints on what the contents might be. Some message types can be more efficient by storing their contents more explicitly (see Operation). They can override this property to return a tuple view on their other properties.

_simplify(*self*)

static simplify(*worker: AbstractWorker*, *ptr: Message*)

This function takes the attributes of a Message and saves them in a tuple. The detail() method runs the inverse of this method. :param worker: a reference to the worker doing the serialization :type worker: AbstractWorker :param ptr: a Message :type ptr: Message

Returns a tuple holding the unique attributes of the message

Return type tuple

Examples

data = simplify(ptr)

static detail(*worker: AbstractWorker*, *msg_tuple: tuple*)

This function takes the simplified tuple version of this message and converts it into a message. The simplify() method runs the inverse of this method.

This method shouldn't get called very often. It exists as a backup but in theory every message type should have its own detailer.

Parameters

- **worker** (AbstractWorker) – a reference to the worker necessary for detailing. Read syft/serde/serde.py for more information on why this is necessary.
- **msg_tuple** (*Tuple*) – the raw information being detailed.

Returns a Operation.

Return type ptr (*Message*)

Examples

message = detail(sy.local_worker, msg_tuple)

__str__(*self*)

Return a human readable version of this message

__repr__(*self*)

Return a human readable version of this message

class syft.messaging.message.**Operation**(*cmd_name*, *cmd_owner*, *cmd_args*, *cmd_kwargs*, *return_ids*)

Bases: *syft.messaging.message.Message*

All syft operations use this message type

In Syft, an operation is when one worker wishes to tell another worker to do something with objects contained in the worker._objects registry (or whatever the official object store is backed with in the case that it's been overridden). Semantically, one could view all Messages as a kind of operation, but when we say operation this is what we mean. For example, telling a worker to take two tensors and add them together is an operation. However, sending an object from one worker to another is not an operation (and would instead use the ObjectMessage type).

property contents(*self*)

Return a tuple with the contents of the operation (backwards compatability)

Some of our codebase still assumes that all message types have a .contents attribute. However, the contents attribute is very opaque in that it doesn't put any constraints on what the contents might be. Since we know this message is a operation, we instead choose to store contents in two pieces, self.message and

self.return_ids, which allows for more efficient simplification (we don't have to simplify return_ids because they are always a list of integers, meaning they're already simplified).

static simplify(*worker: AbstractWorker*, *ptr: Operation*)
This function takes the attributes of a Operation and saves them in a tuple :param worker: a reference to the worker doing the serialization :type worker: AbstractWorker :param ptr: a Message :type ptr: Operation

Returns a tuple holding the unique attributes of the message

Return type tuple

Examples

data = simplify(ptr)

static detail(*worker: AbstractWorker*, *msg_tuple: tuple*)
This function takes the simplified tuple version of this message and converts it into a Operation. The simplify() method runs the inverse of this method.

Parameters

- **worker** (*AbstractWorker*) – a reference to the worker necessary for detailing. Read syft/serde/serde.py for more information on why this is necessary.
- **msg_tuple** (*Tuple*) – the raw information being detailed.

Returns a Operation.

Return type ptr (*Operation*)

Examples

message = detail(sy.local_worker, msg_tuple)

static bufferize(*worker: AbstractWorker*, *operation: Operation*)
This function takes the attributes of a Operation and saves them in Protobuf :param worker: a reference to the worker doing the serialization :type worker: AbstractWorker :param ptr: a Message :type ptr: Operation

Returns a Protobuf message holding the unique attributes of the message

Return type protobuf_obj

Examples

data = bufferize(message)

static unbufferize(*worker: AbstractWorker*, *protobuf_obj: OperationMessagePB*)
This function takes the Protobuf version of this message and converts it into an Operation. The bufferize() method runs the inverse of this method.

Parameters

- **worker** (*AbstractWorker*) – a reference to the worker necessary for detailing. Read syft/serde/serde.py for more information on why this is necessary.
- **protobuf_obj** (*OperationPB*) – the Protobuf message

Returns an Operation

Return type obj (*Operation*)

Examples

message = unbufferize(sy.local_worker, protobuf_msg)

static _bufferize_args(*worker: AbstractWorker*, *args: list*)

static _bufferize_arg(*worker: AbstractWorker*, *arg: object*)

static _unbufferize_args(*worker: AbstractWorker*, *protobuf_args: list*)

static _unbufferize_arg(*worker: AbstractWorker*, *protobuf_arg: ArgPB*)

class syft.messaging.message.**ObjectMessage**(*contents*)
Bases: *syft.messaging.message.Message*

Send an object to another worker using this message type.

When a worker has an object in its local object repository (such as a tensor) and it wants to send that object to another worker (and delete its local copy), it uses this message type to do so.

static detail(*worker: AbstractWorker*, *msg_tuple: tuple*)
This function takes the simplified tuple version of this message and converts it into an ObjectMessage. The simplify() method runs the inverse of this method.

Parameters

- **worker** (*AbstractWorker*) – a reference to the worker necessary for detailing. Read syft/serde/serde.py for more information on why this is necessary.
- **msg_tuple** (*Tuple*) – the raw information being detailed.

Returns a ObjectMessage.

Return type ptr (*ObjectMessage*)

Examples

message = detail(sy.local_worker, msg_tuple)

static bufferize(*worker: AbstractWorker*, *message: ObjectMessage*)
This function takes the attributes of an Object Message and saves them in a protobuf object :param message: an ObjectMessage :type message: ObjectMessage

Returns a protobuf object holding the unique attributes of the object message

Return type protobuf

Examples

data = bufferize(object_message)

static unbufferize(*worker: AbstractWorker*, *protobuf_obj: ObjectMessagePB*)

class syft.messaging.message.**ObjectRequestMessage**(*contents*)
Bases: *syft.messaging.message.Message*

Request another worker to send one of its objects

If ObjectMessage pushes an object to another worker, this Message type pulls an object from another worker. It also assumes that the other worker will delete it's local copy of the object after sending it to you.

static detail(*worker: AbstractWorker*, *msg_tuple: tuple*)
This function takes the simplified tuple version of this message and converts it into an ObjectRequestMessage. The simplify() method runs the inverse of this method.

Parameters

- **worker** (AbstractWorker) – a reference to the worker necessary for detailing. Read syft/serde/serde.py for more information on why this is necessary.
- **msg_tuple** (*Tuple*) – the raw information being detailed.

Returns a ObjectRequestMessage.

Return type ptr (*ObjectRequestMessage*)

Examples

message = detail(sy.local_worker, msg_tuple)

class syft.messaging.message.**IsNoneMessage**(*contents*)
Bases: *syft.messaging.message.Message*

Check if a worker does not have an object with a specific id.

Occasionally we need to verify whether or not a remote worker has a specific object. To do so, we send an IsNoneMessage, which returns True if the object (such as a tensor) does NOT exist.

static detail(*worker: AbstractWorker*, *msg_tuple: tuple*)
This function takes the simplified tuple version of this message and converts it into an IsNoneMessage. The simplify() method runs the inverse of this method.

Parameters

- **worker** (AbstractWorker) – a reference to the worker necessary for detailing. Read syft/serde/serde.py for more information on why this is necessary.
- **msg_tuple** (*Tuple*) – the raw information being detailed.

Returns a IsNoneMessage.

Return type ptr (*IsNoneMessage*)

Examples

message = detail(sy.local_worker, msg_tuple)

class syft.messaging.message.**GetShapeMessage**(*contents*)
Bases: *syft.messaging.message.Message*

Get the shape property of a tensor in PyTorch

We needed to have a special message type for this because .shape had some constraints in the older version of PyTorch.

static detail(*worker: AbstractWorker*, *msg_tuple: tuple*)
This function takes the simplified tuple version of this message and converts it into an GetShapeMessage. The simplify() method runs the inverse of this method.

Parameters

- **worker** (AbstractWorker) – a reference to the worker necessary for detailing. Read syft/serde/serde.py for more information on why this is necessary.

- **msg_tuple** (*Tuple*) – the raw information being detailed.

Returns a GetShapeMessage.

Return type ptr (*GetShapeMessage*)

Examples

message = detail(sy.local_worker, msg_tuple)

class syft.messaging.message.**ForceObjectDeleteMessage**(*contents*)
Bases: *syft.messaging.message.Message*

Garbage collect a remote object

This is the dominant message for garbage collection of remote objects. When a pointer is deleted, this message is triggered by default to tell the object being pointed to to also delete itself.

static detail(*worker: AbstractWorker*, *msg_tuple: tuple*)
This function takes the simplified tuple version of this message and converts it into an ForceObjectDeleteMessage. The simplify() method runs the inverse of this method.

Parameters

- **worker** (AbstractWorker) – a reference to the worker necessary for detailing. Read syft/serde/serde.py for more information on why this is necessary.
- **msg_tuple** (*Tuple*) – the raw information being detailed.

Returns a ForceObjectDeleteMessage.

Return type ptr (*ForceObjectDeleteMessage*)

Examples

message = detail(sy.local_worker, msg_tuple)

class syft.messaging.message.**SearchMessage**(*contents*)
Bases: *syft.messaging.message.Message*

A client queries for a subset of the tensors on a remote worker using this type

For some workers like SocketWorker we split a worker into a client and a server. For this configuration, a client can request to search for a subset of tensors on the server using this message type (this could also be called a "QueryMessage").

static detail(*worker: AbstractWorker*, *msg_tuple: tuple*)
This function takes the simplified tuple version of this message and converts it into an SearchMessage. The simplify() method runs the inverse of this method.

Parameters

- **worker** (AbstractWorker) – a reference to the worker necessary for detailing. Read syft/serde/serde.py for more information on why this is necessary.
- **msg_tuple** (*Tuple*) – the raw information being detailed.

Returns a SearchMessage.

Return type ptr (*SearchMessage*)

Examples

message = detail(sy.local_worker, msg_tuple)

class syft.messaging.message.**PlanCommandMessage**(*command_name: str*, *message: tuple*)

Bases: *syft.messaging.message.Message*

Message used to execute a command related to plans.

property contents(*self*)

Returns a tuple with the contents of the operation (backwards compatability).

static simplify(*worker: AbstractWorker*, *ptr: PlanCommandMessage*)

This function takes the attributes of a PlanCommandMessage and saves them in a tuple

Parameters

- **worker** (AbstractWorker) – a reference to the worker doing the serialization
- **ptr** (PlanCommandMessage) – a Message

Returns a tuple holding the unique attributes of the message

Return type tuple

static detail(*worker: AbstractWorker*, *msg_tuple: tuple*)

This function takes the simplified tuple version of this message and converts it into a PlanCommandMessage. The simplify() method runs the inverse of this method.

Parameters

- **worker** (AbstractWorker) – a reference to the worker necessary for detailing. Read syft/serde/serde.py for more information on why this is necessary.
- **msg_tuple** (*Tuple*) – the raw information being detailed.

Returns a PlanCommandMessage.

Return type ptr (*PlanCommandMessage*)

1.1.1.5.2.3 syft.messaging.protocol

1.1.1.5.2.4 Module Contents

class syft.messaging.protocol.**Protocol**(*plans: List = None*, *id: int = None*, *owner: BaseWorker = None*, *tags: List[str] = None*, *description: str = None*)

Bases: *syft.generic.object.AbstractObject*

A Protocol coordinates a sequence of Plans, deploys them on distant workers and run them in a single pass.

It's a high level object which contains the logic of a complex computation distributed across several workers. The main feature of Protocol is the ability to be sent / searched / fetched back between workers, and finally deployed to identified workers. So a user can design a protocol, upload it to a cloud worker, and any other workers will be able to search for it, download it, and apply the computation program it contains on the workers that it is connected to.

Parameters

- **plans** – a list of pairs (worker, plan). "worker" can be either a real worker or a worker id or a string to represent a fictive worker. This last case can be used at creation to specify that

two plans should be owned (or not owned) by the same worker at deployment. "plan" can either be a Plan or a PointerPlan.

- **id** – the Protocol id
- **owner** – the Protocol owner
- **tags** – the Protocol tags (used for search)
- **description** – the Protocol description

deploy (*self*, **workers: BaseWorker*)
Calling .deploy() sends the plans to the designated workers.

This is done in 2 phases: first, we map the fictive workers provided at creation (named by strings) to the provided workers, and second, we send the corresponding plans to each of them. For the first phase, either there is exactly one real worker per plan or one real worker per fictive_worker. _resolve_workers replaces the fictive workers by the real ones.

Parameters **workers** – BaseWorker. The workers to which plans are to be sent

Returns self

Return type "Protocol"

Raises **RuntimeError** – If protocol is already deployed OR the number of workers provided does not equal the number of fictive workers and does not equal the number of plans

__call__ (*self*, **args*, ***kwargs*)

run (*self*, **args*, ***kwargs*)
Run the protocol by executing the plans sequentially

The input args provided are sent to the first plan location. This first plan is run and its output is moved to the second plan location, and so on. The final result is returned after all plans have run, and it is composed of pointers to the last plan location.

Raises **RuntimeError** – If the protocol has a location attribute and it is not the local worker

request_remote_run (*self*, *location: AbstractWorker*, *args*, *kwargs*)
Requests protocol execution.

Send a request to execute the protocol on the remote location.

Parameters

- **location** – to which worker the request should be sent
- **args** – Arguments used as input data for the protocol.
- **kwargs** – Named arguments used as input data for the protocol.

Returns

response from request to execute protocol

Return type *PointerTensor* or list of PointerTensors

static find_args_location (*args*)
Return location if args contain pointers else the local worker

Returns

The location of a pointer if in args, else local worker

Return type *BaseWorker*

send(*self*, *location: BaseWorker*)
Send a protocol to a worker, to be fetched by other workers

Parameters **location** – BaseWorker. The location to which a protocol is to be sent

static simplify(*worker: BaseWorker*, *protocol: Protocol*)
This function takes the attributes of a Protocol and saves them in a tuple

Parameters

- **worker** (BaseWorker) – the worker doing the serialization
- **protocol** (Protocol) – a Protocol object

Returns a tuple holding the unique attributes of the Protocol object

Return type tuple

Raises **TypeError** – if a plan is not sy.Plan or sy.PointerPlan

static create_from_attributes(*worker*, *id*, *tags*, *description*, *workers_resolved*, *plans_assignments*)
This function reconstructs a Protocol object given its attributes.

Parameters

- **worker** – the worker doing the deserialization
- **id** – Protocol id
- **tags** – Protocol tags
- **description** – Protocol description
- **workers_resolved** – Flag whether workers are resolved
- **plans_assignments** – List of workers/plans IDs

Returns a Protocol object

Return type protocol

static detail(*worker: BaseWorker*, *protocol_tuple: Tuple*)
This function reconstructs a Protocol object given its attributes in the form of a tuple.

Parameters

- **worker** – the worker doing the deserialization
- **protocol_tuple** – a tuple holding the attributes of the Protocol

Returns a Protocol object

Return type protocol

static bufferize(*worker: BaseWorker*, *protocol: Protocol*)
This function takes the attributes of a Protocol and saves them in protobuf ProtocolPB

Parameters

- **worker** (BaseWorker) – the worker doing the serialization
- **protocol** (Protocol) – a Protocol object

Returns a protobuf object holding the unique attributes of the Protocol object

Return type ProtocolPB

Raises **TypeError** – if a plan is not sy.Plan or sy.PointerPlan

static unbufferize(*worker: AbstractWorker*, *pb_protocol: ProtocolPB*)
This function reconstructs a Protocol object given protobuf object.

Parameters

- **worker** – the worker doing the deserialization
- **pb_protocol** – a ProtocolPB object

Returns a Protocol object

Return type protocol

create_pointer(*self*, *owner: AbstractWorker*, *garbage_collect_data: bool*)
Create a pointer to the protocol

Parameters

- **owner** – the owner of the pointer
- **garbage_collect_data** – bool

Returns pointer to the protocol

Return type *PointerProtocol*

__repr__(*self*)

__str__(*self*)

_assert_is_resolved(*self*)
Check if protocol has already been resolved

Raises **RuntimeError** – If protocol has already been resolved

_resolve_workers(*self, workers: Tuple[BaseWorker, ...]*)
Map the abstract workers (named by strings) to the provided workers and update the plans accordingly

Parameters **workers** – Iterable of workers. The workers to map to workers in the protocol

1.1.1.6 syft.serde

1.1.1.6.1 Subpackages

1.1.1.6.1.1 syft.serde.msgpack

1.1.1.6.1.2 Submodules

1.1.1.6.1.3 syft.serde.msgpack.native_serde

This file exists to provide one common place for all serialisation and **simplify_** and _detail for all native python objects.

1.1.1.6.1.4 Module Contents

syft.serde.msgpack.native_serde.**_simplify_collection**(*worker: AbstractWorker*, *my_collection: Collection*, *shallow: bool = False*) → Tuple

This function is designed to search a collection for any objects which may need to be simplified (i.e., torch tensors). It iterates through each object in the collection and calls _simplify on it. Finally, it returns the output as the tuple of simplified items of the input collection. This function is used to simplify list, set, and tuple. The reverse function, which undoes the functionality of this function is different for each of these types: _detail_collection_list, _detail_collection_set, _detail_collection_tuple.

Parameters **my_collection** (*Collection*) – a collection of python objects

Returns a tuple with simplified objects.

Return type Tuple

syft.serde.msgpack.native_serde.**_detail_collection_list**(*worker: AbstractWorker*, *my_collection: Tuple*, *shallow: bool = False*) → Collection

This function is designed to operate in the opposite direction of _simplify_collection. It takes a tuple of simple python objects and iterates through it to determine whether objects in the collection need to be converted into more advanced types. In particular, it converts binary objects into torch Tensors where appropriate.

Parameters

- **worker** – the worker doing the deserialization
- **my_collection** (*Tuple*) – a tuple of simple python objects (including binary).

Returns

a collection of the same type as the input where the objects in the collection have been detailed.

Return type Collection

syft.serde.msgpack.native_serde.**_detail_collection_set**(*worker: AbstractWorker*, *my_collection: Tuple*, *shallow: bool = False*) → Collection

This function is designed to operate in the opposite direction of _simplify_collection. It takes a tuple of simple python objects and iterates through it to determine whether objects in the collection need to be converted into more advanced types. In particular, it converts binary objects into torch Tensors where appropriate.

Parameters

- **worker** – the worker doing the deserialization
- **my_collection** (*Tuple*) – a tuple of simple python objects (including binary).

Returns

a collection of the same type as the input where the objects in the collection have been detailed.

Return type Collection

syft.serde.msgpack.native_serde.**_detail_collection_tuple**(*worker: AbstractWorker*, *my_tuple: Tuple*, *shallow: bool = False*) → Tuple

This function is designed to operate in the opposite direction of _simplify_collection. It takes a tuple of simple python objects and iterates through it to determine whether objects in the collection need to be converted into more advanced types. In particular, it converts binary objects into torch Tensors where appropriate. This is only applicable to tuples. They need special handling because *msgpack* is encoding a tuple as a list.

Parameters

- **worker** – the worker doing the deserialization
- **my_tuple** (*Tuple*) – a collection of simple python objects (including binary).

Returns

a collection of the same type as the input where the objects in the collection have been detailed.

Return type tuple

syft.serde.msgpack.native_serde.**_simplify_dictionary**(*worker: AbstractWorker*, *my_dict: Dict*, *shallow: bool = False*) → Tuple

This function is designed to search a dict for any objects which may need to be simplified (i.e., torch tensors). It iterates through each key, value in the dict and calls _simplify on it. Finally, it returns the output tuple of tuples containing key/value pairs. The reverse function to this function is _detail_dictionary, which undoes the functionality of this function.

Parameters **my_dict** – A dictionary of python objects.

Returns

Tuple containing tuples of simplified key/value pairs from the input dictionary.

Return type Tuple

syft.serde.msgpack.native_serde.**_detail_dictionary**(*worker: AbstractWorker*, *my_dict: Tuple*, *shallow: bool = False*) → Dict

This function is designed to operate in the opposite direction of _simplify_dictionary. It takes a dictionary of simple python objects and iterates through it to determine whether objects in the collection need to be converted into more advanced types. In particular, it converts binary objects into torch Tensors where appropriate.

Parameters

- **worker** – the worker doing the deserialization
- **my_dict** (*Tuple*) – a simplified dictionary of simple python objects (including binary).

Returns

a collection of the same type as the input where the objects in the collection have been detailed.

Return type Dict

syft.serde.msgpack.native_serde.**_simplify_str**(*worker: AbstractWorker*, *obj: str*) → tuple

syft.serde.msgpack.native_serde.**_detail_str**(*worker: AbstractWorker*, *str_tuple: tuple*) → str

syft.serde.msgpack.native_serde.**_simplify_range**(*worker: AbstractWorker*, *my_range: range*) → Tuple[int, int, int]

This function extracts the start, stop and step from the range.

Parameters **my_range** (*range*) – a range object

Returns a list defining the range parameters [start, stop, step]

Return type list

Examples

range_parameters = _simplify_range(range(1, 3, 4))

assert range_parameters == [1, 3, 4]

syft.serde.msgpack.native_serde.**_detail_range**(*worker: AbstractWorker, my_range_params: Tuple[int, int, int]*) → range

This function extracts the start, stop and step from a tuple.

Parameters

- **worker** – the worker doing the deserialization (only here to standardise signature with other _detail functions)
- **my_range_params** (*tuple*) – a tuple defining the range parameters [start, stop, step]

Returns a range object

Return type range

Examples

new_range = _detail_range([1, 3, 4])

assert new_range == range(1, 3, 4)

syft.serde.msgpack.native_serde.**_simplify_ellipsis**(*worker: AbstractWorker*, *e: Ellipsis*) → Tuple[bytes]

syft.serde.msgpack.native_serde.**_detail_ellipsis**(*worker: AbstractWorker*, *ellipsis: bytes*) → Ellipsis

syft.serde.msgpack.native_serde.**_simplify_slice**(*worker: AbstractWorker*, *my_slice: slice*) → Tuple[int, int, int]

This function creates a list that represents a slice.

Parameters **my_slice** (*slice*) – a python slice

Returns a list holding the start, stop and step values

Return type tuple

Examples

slice_representation = _simplify_slice(slice(1,2,3))

syft.serde.msgpack.native_serde.**_detail_slice**(*worker: AbstractWorker, my_slice: Tuple[int, int, int]*) → slice

This function extracts the start, stop and step from a list.

Parameters **my_slice** (*tuple*) – a list defining the slice parameters [start, stop, step]

Returns a range object

Return type range

Examples

new_range = _detail_range([1, 3, 4])

assert new_range == range(1, 3, 4)

syft.serde.msgpack.native_serde.**_simplify_ndarray**(*worker: AbstractWorker*, *my_array: numpy.ndarray*) → Tuple[bin, Tuple, Tuple]

This function gets the byte representation of the array and stores the dtype and shape for reconstruction

Parameters **my_array** (*numpy.ndarray*) – a numpy array

Returns a list holding the byte representation, shape and dtype of the array

Return type list

Examples

arr_representation = _simplify_ndarray(numpy.random.random([1000, 1000]))

syft.serde.msgpack.native_serde.**_detail_ndarray**(*worker: AbstractWorker*, *arr_representation: Tuple[bin, Tuple, str]*) → numpy.ndarray

This function reconstruct a numpy array from it's byte data, the shape and the dtype by first loading the byte data with the appropiate dtype and then reshaping it into the original shape

Parameters

- **worker** – the worker doing the deserialization
- **arr_representation** (*tuple*) – a tuple holding the byte representation, shape
- **dtype of the array** (*and*) –

Returns a numpy array

Return type numpy.ndarray

Examples

arr = _detail_ndarray(arr_representation)

syft.serde.msgpack.native_serde.**_simplify_numpy_number**(*worker: AbstractWorker*, *numpy_nb: Union[numpy.int32, numpy.int64, numpy.float32, numpy.float64]*) → Tuple[bin, Tuple]

This function gets the byte representation of the numpy number and stores the dtype for reconstruction

Parameters **numpy_nb** (*e.g numpy.float64*) – a numpy number

Returns a list holding the byte representation, dtype of the numpy number

Return type list

Examples

np_representation = _simplify_numpy_number(worker, numpy.float64(2.3)))

syft.serde.msgpack.native_serde.**_detail_numpy_number**(*worker: AbstractWorker, nb_representation: Tuple[bin, Tuple, str]*) → Union[numpy.int32, numpy.int64, numpy.float32, numpy.float64]

This function reconstruct a numpy number from it's byte data, dtype by first loading the byte data with the appropiate dtype

Parameters

- **worker** – the worker doing the deserialization
- **np_representation** (*tuple*) – a tuple holding the byte representation
- **dtype of the numpy number** (*and*) –

Returns a numpy number

Return type numpy.float or numpy.int

Examples

nb = _detail_numpy_number(nb_representation)

syft.serde.msgpack.native_serde.**MAP_NATIVE_SIMPLIFIERS_AND_DETAILERS**

1.1.1.6.1.5 syft.serde.msgpack.proto

This file exists to translate python classes to Serde type constants defined in *proto.json* file in https://github.com/OpenMined/proto. The reason for this is to have stable constants used in Serde serialization protocol and the definition file that can be used not only by PySyft but also in other languages.

https://github.com/OpenMined/proto (*pysyft_proto* module) is included as dependency in setup.py exposes contents of *proto.json* file in *proto_info* variable.

IMPORTANT: New types added in Serde need to be also defined in *proto.json*.

1.1.1.6.1.6 Module Contents

class syft.serde.msgpack.proto.**TypeInfo**(*name*, *obj*)

Convenience wrapper for type info defined in *proto_info*. Exposes type constants with error handling.

property code(*self*)

Returns *code* property (serialization constant) for class or throws an exception if it's not defined in *proto.json*.

property forced_code(*self*)

Returns *forced_code* property (serialization constant) for class or throws an exception if it's not defined in *proto.json*.

syft.serde.msgpack.proto.**fullname**(*cls*)
Returns full name of a given *class* (not instance of class). Source: https://stackoverflow.com/questions/2020014/get-fully-qualified-class-name-of-an-object-in-python.

syft.serde.msgpack.proto.**proto_type_info**(*cls*)
Returns *TypeInfo* instance for a given *class* identified by *cls* parameter. Throws an exception when such class does not exists in the *proto.json*.

1.1.1.6.1.7 syft.serde.msgpack.serde

This file exists to provide one common place for all msgpack serialization to occur. As msgpack only supports basic types and binary formats every type must be first be converted to one of these types. Thus, we've split our functionality into three steps. When converting from a PySyft object (or collection of objects) to an object to be sent over the wire (a message), those three steps are (in order):

1. Simplify - converts PyTorch objects to simple Python objects (using pickle)
2. Serialize - converts Python objects to binary
3. Compress - compresses the binary (Now we're ready send!)

Inversely, when converting from a message sent over the wire back to a PySyft object, the three steps are (in order):

1. Decompress - converts compressed binary back to decompressed binary
2. Deserialize - converts from binary to basic python objects
3. Detail - converts some basic python objects back to PyTorch objects (Tensors)

Furthermore, note that there is different simplification/serialization logic for objects of different types. Thus, instead of using if/else logic, we have global dictionaries which contain functions and Python types as keys. For simplification logic, this dictionary is called "simplifiers". The keys are the types and values are the simplification logic. For example, simplifiers[tuple] will return the function which knows how to simplify the tuple type. The same is true for all other simplifier/detailer functions.

By default, the simplification/detail operations expect Torch tensors. If the setup requires other serialization process, it can override the functions _serialize_tensor and _deserialize_tensor

By default, we serialize using msgpack and compress using lz4. If different compressions are required, the worker can override the function apply_compress_scheme

1.1.1.6.1.8 Module Contents

syft.serde.msgpack.serde.**MAP_TORCH_SIMPLIFIERS_AND_DETAILERS**

syft.serde.msgpack.serde.**MAP_TF_SIMPLIFIERS_AND_DETAILERS**

syft.serde.msgpack.serde.**MAP_TO_SIMPLIFIERS_AND_DETAILERS**

syft.serde.msgpack.serde.**OBJ_SIMPLIFIER_AND_DETAILERS**

syft.serde.msgpack.serde.**OBJ_FORCE_FULL_SIMPLIFIER_AND_DETAILERS**

syft.serde.msgpack.serde.**EXCEPTION_SIMPLIFIER_AND_DETAILERS**

syft.serde.msgpack.serde.**_force_full_simplify**(*worker: AbstractWorker*, *obj: object*) → object
To force a full simplify generally if the usual _simplify is not suitable.

If we can not full simplify a object we simplify it as usual instead.

Parameters **obj** – The object.

Returns The simplified object.

syft.serde.msgpack.serde.**_generate_simplifiers_and_detailers**()
Generate simplifiers, forced full simplifiers and detailers, by registering native and torch types, syft objects with custom simplify and detail methods, or syft objects with custom force_simplify and force_detail methods.

NOTE: this function uses *proto_type_info* that translates python class into Serde constant defined in https://github.com/OpenMined/proto. If the class used in *MAP_TO_SIMPLIFIERS_AND_DETAILERS*, *OBJ_SIMPLIFIER_AND_DETAILERS*, *EXCEPTION_SIMPLIFIER_AND_DETAILERS*, *OBJ_FORCE_FULL_SIMPLIFIER_AND_DETAILERS* is not defined in *proto.json* file in https://github.com/OpenMined/proto, this function will error. :returns: The simplifiers, forced_full_simplifiers, detailers

syft.serde.msgpack.serde.**inherited_simplifiers_found**

syft.serde.msgpack.serde.**_serialize_msgpack_simple**(*obj: object*, *worker: AbstractWorker = None*, *simplified: bool = False*, *force_full_simplification: bool = False*) → bin

syft.serde.msgpack.serde.**_serialize_msgpack_binary**(*simple_objects: object*, *worker: AbstractWorker = None*, *simplified: bool = False*, *force_full_simplification: bool = False*) → bin

syft.serde.msgpack.serde.**serialize**(*obj: object*, *worker: AbstractWorker = None*, *simplified: bool = False*, *force_full_simplification: bool = False*) → bin
This method can serialize any object PySyft needs to send or store.

This is the high level function for serializing any object or collection of objects which PySyft needs to send over the wire. It includes three steps, Simplify, Serialize, and Compress as described inline below.

Parameters

- **obj** (*object*) – the object to be serialized
- **simplified** (*bool*) – in some cases we want to pass in data which has already been simplified - in which case we must skip double simplification - which would be bad.... so bad... so... so bad
- **force_full_simplification** (*bool*) – Some objects are only partially serialized by default. For objects where this is the case, setting this flag to True will force the entire object to be serialized. For example, setting this flag to True will cause a VirtualWorker to be serialized WITH all of its tensors while by default VirtualWorker objects only serialize a small amount of metadata.

Returns the serialized form of the object.

Return type binary

syft.serde.msgpack.serde.**_deserialize_msgpack_binary**(*binary: bin*, *worker: AbstractWorker = None*) → object

syft.serde.msgpack.serde.**_deserialize_msgpack_simple**(*simple_objects: object*, *worker: AbstractWorker = None*) → object

syft.serde.msgpack.serde.**deserialize**(*binary: bin*, *worker: AbstractWorker = None*) → object

syft.serde.msgpack.serde.**_simplify**(*worker: AbstractWorker*, *obj: object*, ***kwargs*) → object
This function takes an object as input and returns a simple Python object which is supported by the chosen serialization method (such as JSON or msgpack). The reason we have this function is that some objects are

either NOT supported by high level (fast) serializers OR the high level serializers don't support the fastest form of serialization. For example, PyTorch tensors have custom pickle functionality thus its better to pre-serialize PyTorch tensors using pickle and then serialize the binary in with the rest of the message being sent.

Parameters **obj** – An object which may need to be simplified.

Returns An simple Python object which msgpack can serialize.

Raises

- **ValueError** – if *move_this* or *in_front_of_that* are not both single ASCII
- **characters.** –

syft.serde.msgpack.serde.**_detail**(*worker: AbstractWorker*, *obj: object*, ***kwargs*) → object

Reverses the functionality of _simplify. Where applicable, it converts simple objects into more complex objects such as converting binary objects into torch tensors. Read _simplify for more information on why _simplify and detail are needed.

Parameters

- **worker** – the worker which is acquiring the message content, for example
- **to specify the owner of a tensor received(not obvious for** (*used*) –
- **workers)** (*virtual*) –
- **obj** – a simple Python object which msgpack deserialized.

Returns

a more complex Python object which msgpack would have had trouble deserializing directly.

Return type obj

1.1.1.6.1.9 syft.serde.msgpack.torch_serde

This file exists to provide one common place for all serialisation and **simplify_** and _detail for all tensors (Torch and Numpy).

1.1.1.6.1.10 Module Contents

syft.serde.msgpack.torch_serde.**_serialize_tensor**(*worker: AbstractWorker*, *tensor*) → bin

Serialize the tensor using as default Torch serialization strategy This function can be overridden to provide different tensor serialization strategies

Args (torch.Tensor): an input tensor to be serialized

Returns A serialized version of the input tensor

syft.serde.msgpack.torch_serde.**_deserialize_tensor**(*worker: AbstractWorker*, *serializer: str*, *tensor_bin*) → torch.Tensor

Deserialize the input tensor passed as parameter into a Torch tensor. *serializer* parameter selects different deserialization strategies

Args worker: Worker serializer: Strategy used for tensor deserialization (e.g.: torch, numpy, all) tensor_bin: A simplified representation of a tensor

Returns a Torch tensor

syft.serde.msgpack.torch_serde.**simplified_tensor_serializer**(*worker: AbstractWorker*, *tensor: torch.Tensor*) → tuple

Strategy to serialize a tensor to native python types. If tensor requires to calculate gradients, it will be detached.

syft.serde.msgpack.torch_serde.**simplified_tensor_deserializer**(*worker: AbstractWorker*, *tensor_tuple: tuple*) → torch.Tensor

"Strategy to deserialize a simplified tensor into a Torch tensor

syft.serde.msgpack.torch_serde.**_simplify_torch_tensor**(*worker: AbstractWorker*, *tensor: torch.Tensor*) → bin

This function converts a torch tensor into a serliaized torch tensor using pickle. We choose to use this because PyTorch has a custom and very fast PyTorch pickler.

Parameters **tensor** (*torch.Tensor*) – an input tensor to be serialized

Returns serialized tuple of torch tensor. The first value is the id of the tensor and the second is the binary for the PyTorch object. The third is the chain of abstractions, and the fourth (optinally) is the chain of graident tensors (nested tuple)

Return type tuple

syft.serde.msgpack.torch_serde.**_detail_torch_tensor**(*worker: AbstractWorker*, *tensor_tuple: tuple*) → torch.Tensor

This function converts a serialized torch tensor into a torch tensor using pickle.

Parameters **tensor_tuple** (*bin*) – serialized obj of torch tensor. It's a tuple where the first value is the ID, the second vlaue is the binary for the PyTorch object, the third value is the chain of tensor abstractions, and the fourth object is the chain of gradients (.grad.grad, etc.)

Returns a torch tensor that was serialized

Return type torch.Tensor

syft.serde.msgpack.torch_serde.**_simplify_torch_parameter**(*worker: AbstractWorker*, *param: torch.nn.Parameter*) → bin

This function converts a torch Parameter into a serialized torch Parameter

Parameters **param** (*torch.nn.Parameter*) – an input Parameter to be serialized

Returns serialized tuple of torch Parameter. The first value is the id of the Parameter and the second is the binary for the PyTorch tensor data attribute and last is the requires_grad attr.

Return type tuple

syft.serde.msgpack.torch_serde.**_detail_torch_parameter**(*worker: AbstractWorker*, *param_tuple: tuple*) → torch.nn.Parameter

This function converts a serialized torch Parameter into a torch Parameter.

Parameters **param_tuple** (*tuple*) – serialized obj of torch tensor. It's a tuple where the first value is the ID and the second value is the binary for the PyTorch data attribute et and third value is the requires_grad attr.

Returns a torch Parameter that was serialized

Return type torch.Parameter

syft.serde.msgpack.torch_serde.**_simplify_torch_device**(*worker: AbstractWorker*, *device: torch.device*) → Tuple

syft.serde.msgpack.torch_serde.**_detail_torch_device**(*worker: AbstractWorker*, *device_type: tuple*) → torch.device

syft.serde.msgpack.torch_serde.**_simplify_script_module**(*worker: AbstractWorker*, *obj: torch.jit.ScriptModule*) → Tuple

Strategy to serialize a script module using Torch.jit

syft.serde.msgpack.torch_serde.**_detail_script_module**(*worker: AbstractWorker*, *script_module_bin: Tuple*) → torch.jit.ScriptModule

"Strategy to deserialize a binary input using Torch load

syft.serde.msgpack.torch_serde.**_simplify_torch_size**(*worker: AbstractWorker*, *size: torch.Size*) → Tuple[int]

syft.serde.msgpack.torch_serde.**_detail_torch_size**(*worker: AbstractWorker, size: Tuple[int]*) → torch.Size

syft.serde.msgpack.torch_serde.**_simplify_torch_mem_format**(*worker: AbstractWorker*, *mem_format: torch.memory_format*) → int

syft.serde.msgpack.torch_serde.**_detail_torch_mem_format**(*worker: AbstractWorker*, *mem_format: int*) → torch.memory_format

syft.serde.msgpack.torch_serde.**MAP_TORCH_SIMPLIFIERS_AND_DETAILERS**

1.1.1.6.1.11 Package Contents

syft.serde.msgpack.**proto_type_info**(*cls*)

Returns *TypeInfo* instance for a given *class* identified by *cls* parameter. Throws an exception when such class does not exists in the *proto.json*.

syft.serde.msgpack.**serialize**(*obj: object*, *worker: AbstractWorker = None*, *simplified: bool = False*, *force_full_simplification: bool = False*) → bin

This method can serialize any object PySyft needs to send or store.

This is the high level function for serializing any object or collection of objects which PySyft needs to send over the wire. It includes three steps, Simplify, Serialize, and Compress as described inline below.

Parameters

- **obj** (*object*) – the object to be serialized
- **simplified** (*bool*) – in some cases we want to pass in data which has already been simplified - in which case we must skip double simplification - which would be bad.... so bad... so... so bad
- **force_full_simplification** (*bool*) – Some objects are only partially serialized by default. For objects where this is the case, setting this flag to True will force the entire object to be serialized. For example, setting this flag to True will cause a VirtualWorker to be serialized WITH all of its tensors while by default VirtualWorker objects only serialize a small amount of metadata.

Returns the serialized form of the object.

Return type binary

syft.serde.msgpack.**deserialize**(*binary: bin*, *worker: AbstractWorker = None*) → object

1.1.1.6.1.12 syft.serde.protobuf

1.1.1.6.1.13 Submodules

1.1.1.6.1.14 syft.serde.protobuf.native_serde

This file exists to provide a common place for all Protobuf serialisation for native Python objects. If you're adding something here that isn't for *None*, think twice and either use an existing sub-class of Message or add a new one.

1.1.1.6.1.15 Module Contents

syft.serde.protobuf.native_serde.**_bufferize_none**(*worker: AbstractWorker*, *obj: type(None)*) → 'Empty'

This function converts None into an empty Protobuf message.

Parameters **obj** (*None*) – makes signature match other bufferize methods

Returns Empty Protobuf message

Return type protobuf_obj

syft.serde.protobuf.native_serde.**_unbufferize_none**(*worker: AbstractWorker*, *obj: Empty*) → 'type(None)'

This function converts an empty Protobuf message back into None.

Parameters **obj** (*Empty*) – Empty Protobuf message

Returns None

Return type obj

syft.serde.protobuf.native_serde.**MAP_NATIVE_PROTOBUF_TRANSLATORS**

1.1.1.6.1.16 syft.serde.protobuf.proto

This file exists to translate python classes to and from Protobuf messages. The reason for this is to have stable serialization protocol that can be used not only by PySyft but also in other languages.

https://github.com/OpenMined/syft-proto (*syft_proto* module) is included as a dependency in setup.py.

1.1.1.6.1.17 Module Contents

syft.serde.protobuf.proto.**MAP_PYTHON_TO_PROTOBUF_CLASSES**

syft.serde.protobuf.proto.**set_protobuf_id**(*field*, *id*)

syft.serde.protobuf.proto.**get_protobuf_id**(*field*)

1.1.1.6.1.18 syft.serde.protobuf.serde

1.1.1.6.1.19 Module Contents

syft.serde.protobuf.serde.**MAP_TORCH_PROTOBUF_TRANSLATORS**

syft.serde.protobuf.serde.**MAP_TO_PROTOBUF_TRANSLATORS**

syft.serde.protobuf.serde.**OBJ_PROTOBUF_TRANSLATORS**

syft.serde.protobuf.serde.**OBJ_FORCE_FULL_PROTOBUF_TRANSLATORS = []**

syft.serde.protobuf.serde.**EXCEPTION_PROTOBUF_TRANSLATORS = []**

syft.serde.protobuf.serde.**_force_full_bufferize**(*worker: AbstractWorker*, *obj: object*) → object

To force a full bufferize conversion generally if the usual _bufferize is not suitable.

If we can not full convert an object we convert it as usual instead.

Parameters **obj** – The object.

Returns The bufferize object.

syft.serde.protobuf.serde.**_generate_bufferizers_and_unbufferizers**()

Generate bufferizers, forced full bufferizers and unbufferizers, by registering native and torch types, syft objects with custom bufferize and unbufferize methods, or syft objects with custom force_bufferize and force_unbufferize methods.

NOTE: this function uses *proto_type_info* that translates python class into Serde constant defined in https://github.com/OpenMined/proto. If the class used in *MAP_TO_SIMPLIFIERS_AND_DETAILERS*, *OBJ_SIMPLIFIER_AND_DETAILERS*, *EXCEPTION_SIMPLIFIER_AND_DETAILERS*, *OBJ_FORCE_FULL_SIMPLIFIER_AND_DETAILERS* is not defined in *proto.json* file in https://github.com/OpenMined/proto, this function will error. :returns: The bufferizers, forced_full_bufferizers, unbufferizers

syft.serde.protobuf.serde.**inherited_bufferizers_found**

syft.serde.protobuf.serde.**serialize**(*obj: object*, *worker: AbstractWorker = None*, *simplified: bool = False*, *force_no_compression: bool = False*, *force_no_serialization: bool = False*, *force_full_simplification: bool = False*) → bin

This method can serialize any object PySyft needs to send or store.

This is the high level function for serializing any object or collection of objects which PySyft needs to send over the wire. It includes three steps, Simplify, Serialize, and Compress as described inline below.

Parameters

- **obj** (*object*) – the object to be serialized
- **simplified** (*bool*) – in some cases we want to pass in data which has already been simplified - in which case we must skip double simplification - which would be bad.... so bad... so... so bad
- **force_no_compression** (*bool*) – If true, this will override ANY module settings and not compress the objects being serialized. The primary expected use of this functionality is testing and/or experimentation.
- **force_no_serialization** (*bool*) – Primarily a testing tool, this will force this method to return human-readable Python objects which is very useful for testing and debugging (forceably overrides module compression, serialization, and the

'force_no_compression' override)). In other words, only simplification operations are performed.

- **force_full_simplification** (*bool*) – Some objects are only partially serialized by default. For objects where this is the case, setting this flag to True will force the entire object to be serialized. For example, setting this flag to True will cause a VirtualWorker to be serialized WITH all of its tensors while by default VirtualWorker objects only serialize a small amount of metadata.

Returns the serialized form of the object.

Return type binary

syft.serde.protobuf.serde.**deserialize**(*binary: bin*, *worker: AbstractWorker = None*, *unbufferizes=True*) → object

This method can deserialize any object PySyft needs to send or store.

This is the high level function for deserializing any object or collection of objects which PySyft has sent over the wire or stored. It includes three steps, Decompress, Deserialize, and Detail as described inline below.

Parameters

- **binary** (*bin*) – the serialized object to be deserialized.
- **worker** (AbstractWorker) – the worker which is acquiring the message content, for example used to specify the owner of a tensor received(not obvious for virtual workers)
- **unbufferizes** (*bool*) – there are some cases where we need to perform the decompression and deserialization part, but we don't need to unbufferize all the message. This is the case for Plan workers for instance

Returns the deserialized form of the binary input.

Return type object

syft.serde.protobuf.serde.**_bufferize**(*worker: AbstractWorker*, *obj: object*, ***kwargs*) → object

This function takes an object as input and returns a Protobuf object. The reason we have this function is that some objects are either NOT supported by high level (fast) serializers OR the high level serializers don't support the fastest form of serialization. For example, PyTorch tensors have custom pickle functionality thus its better to pre-serialize PyTorch tensors using pickle and then serialize the binary in with the rest of the message being sent.

Parameters **obj** – An object which needs to be converted to Protobuf.

Returns An Protobuf object which Protobuf can serialize.

syft.serde.protobuf.serde.**_unbufferize**(*worker: AbstractWorker*, *obj: object*, ***kwargs*) → object

Reverses the functionality of _bufferize. Where applicable, it converts simple objects into more complex objects such as converting binary objects into torch tensors. Read _bufferize for more information on why _bufferize and unbufferize are needed.

Parameters

- **worker** – the worker which is acquiring the message content, for example
- **to specify the owner of a tensor received(not obvious for** (*used*) –
- **workers)** (*virtual*) –
- **obj** – a simple Python object which msgpack deserialized.

Returns

a more complex Python object which msgpack would have had trouble deserializing directly.

Return type obj

1.1.1.6.1.20 syft.serde.protobuf.torch_serde

This file exists to provide one common place for all serialisation and **bufferize_** and _unbufferize for all tensors (Torch and Numpy).

1.1.1.6.1.21 Module Contents

syft.serde.protobuf.torch_serde.**SERIALIZERS_SYFT_TO_PROTOBUF**

syft.serde.protobuf.torch_serde.**SERIALIZERS_PROTOBUF_TO_SYFT**

syft.serde.protobuf.torch_serde.**_serialize_tensor**(*worker: AbstractWorker*, *tensor*) → bin

Serialize the tensor using as default Torch serialization strategy This function can be overridden to provide different tensor serialization strategies

Args (torch.Tensor): an input tensor to be serialized

Returns A serialized version of the input tensor

syft.serde.protobuf.torch_serde.**_deserialize_tensor**(*worker: AbstractWorker*, *serializer: str*, *tensor_bin*) → torch.Tensor

Deserialize the input tensor passed as parameter into a Torch tensor. *serializer* parameter selects different deserialization strategies

Args worker: Worker serializer: Strategy used for tensor deserialization (e.g.: torch, numpy, all) tensor_bin: A simplified representation of a tensor

Returns a Torch tensor

syft.serde.protobuf.torch_serde.**protobuf_tensor_serializer**(*worker: AbstractWorker*, *tensor: torch.Tensor*) → TensorDataPB

Strategy to serialize a tensor using Protobuf

syft.serde.protobuf.torch_serde.**protobuf_tensor_deserializer**(*worker: AbstractWorker*, *protobuf_tensor: TensorDataPB*) → torch.Tensor

"Strategy to deserialize a binary input using Protobuf

syft.serde.protobuf.torch_serde.**_bufferize_torch_tensor**(*worker: AbstractWorker*, *tensor: torch.Tensor*) → bin

This function converts a Torch tensor into a serialized tensor using Protobuf. Depending on the worker's serializer, the tensor contents may be serialized to binary representations using Torch or Numpy, or to a generic inner Protobuf message for cross-platform communication.

Parameters **tensor** (*torch.Tensor*) – an input tensor to be serialized

Returns Protobuf version of torch tensor.

Return type protobuf_obj

syft.serde.protobuf.torch_serde.**_unbufferize_torch_tensor**(*worker: AbstractWorker*, *protobuf_tensor: TorchTensorPB*) → torch.Tensor

This function converts a Protobuf torch tensor back into a Torch tensor. The tensor contents can be deserialized from binary representations produced by Torch or Numpy, or from the generic Protobuf message format for cross-platform communication.

Parameters **protobuf_tensor** (*bin*) – Protobuf message of torch tensor.

Returns a torch tensor converted from Protobuf

Return type torch.Tensor

syft.serde.protobuf.torch_serde.**_bufferize_torch_device**(*worker: AbstractWorker*, *device: torch.device*) → DevicePB

syft.serde.protobuf.torch_serde.**_unbufferize_torch_device**(*worker: AbstractWorker*, *protobuf_device: DevicePB*) → torch.device

syft.serde.protobuf.torch_serde.**_bufferize_torch_parameter**(*worker: AbstractWorker*, *param: torch.nn.Parameter*) → ParameterPB

syft.serde.protobuf.torch_serde.**_unbufferize_torch_parameter**(*worker: AbstractWorker*, *protobuf_param: ParameterPB*) → torch.nn.Parameter

syft.serde.protobuf.torch_serde.**_bufferize_script_module**(*worker: AbstractWorker*, *script_module: torch.jit.ScriptModule*) → ScriptModulePB

syft.serde.protobuf.torch_serde.**_unbufferize_script_module**(*worker: AbstractWorker*, *protobuf_script: ScriptModulePB*) → torch.jit.ScriptModule

syft.serde.protobuf.torch_serde.**_bufferize_script_function**(*worker: AbstractWorker*, *script_module: torch.jit.ScriptFunction*) → ScriptFunctionPB

syft.serde.protobuf.torch_serde.**_unbufferize_script_function**(*worker: AbstractWorker*, *protobuf_script: ScriptFunctionPB*) → torch.jit.ScriptFunction

syft.serde.protobuf.torch_serde.**_bufferize_traced_module**(*worker: AbstractWorker*, *script_module: torch.jit.TopLevelTracedModule*) → TracedModulePB

syft.serde.protobuf.torch_serde.**_unbufferize_traced_module**(*worker: AbstractWorker*, *protobuf_script: TracedModulePB*) → torch.jit.TopLevelTracedModule

syft.serde.protobuf.torch_serde.**_bufferize_torch_size**(*worker: AbstractWorker*, *size: torch.Size*) → SizePB

syft.serde.protobuf.torch_serde.**_unbufferize_torch_size**(*worker: AbstractWorker*, *protobuf_size: SizePB*) → torch.Size

syft.serde.protobuf.torch_serde.**MAP_TORCH_PROTOBUF_TRANSLATORS**

1.1.1.6.1.22 Package Contents

syft.serde.protobuf.**serialize**(*obj: object*, *worker: AbstractWorker = None*, *simplified: bool = False*, *force_no_compression: bool = False*, *force_no_serialization: bool = False*, *force_full_simplification: bool = False*) → bin

This method can serialize any object PySyft needs to send or store.

This is the high level function for serializing any object or collection of objects which PySyft needs to send over the wire. It includes three steps, Simplify, Serialize, and Compress as described inline below.

Parameters

- **obj** (*object*) – the object to be serialized
- **simplified** (*bool*) – in some cases we want to pass in data which has already been simplified - in which case we must skip double simplification - which would be bad.... so bad... so... so bad
- **force_no_compression** (*bool*) – If true, this will override ANY module settings and not compress the objects being serialized. The primary expected use of this functionality is testing and/or experimentation.
- **force_no_serialization** (*bool*) – Primarily a testing tool, this will force this method to return human-readable Python objects which is very useful for testing and debugging (forceably overrides module compression, serialization, and the 'force_no_compression' override)). In other words, only simplification operations are performed.
- **force_full_simplification** (*bool*) – Some objects are only partially serialized by default. For objects where this is the case, setting this flag to True will force the entire object to be serialized. For example, setting this flag to True will cause a VirtualWorker to be serialized WITH all of its tensors while by default VirtualWorker objects only serialize a small amount of metadata.

Returns the serialized form of the object.

Return type binary

syft.serde.protobuf.**deserialize**(*binary: bin*, *worker: AbstractWorker = None*, *unbufferizes=True*) → object

This method can deserialize any object PySyft needs to send or store.

This is the high level function for deserializing any object or collection of objects which PySyft has sent over the wire or stored. It includes three steps, Decompress, Deserialize, and Detail as described inline below.

Parameters

- **binary** (*bin*) – the serialized object to be deserialized.
- **worker** (AbstractWorker) – the worker which is acquiring the message content, for example used to specify the owner of a tensor received(not obvious for virtual workers)
- **unbufferizes** (*bool*) – there are some cases where we need to perform the decompression and deserialization part, but we don't need to unbufferize all the message. This is the case for Plan workers for instance

Returns the deserialized form of the binary input.

Return type object

1.1.1.6.1.23 syft.serde.torch

1.1.1.6.1.24 Submodules

1.1.1.6.1.25 syft.serde.torch.serde

1.1.1.6.1.26 Module Contents

syft.serde.torch.serde.**TORCH_DTYPE_STR**

syft.serde.torch.serde.**TORCH_STR_DTYPE**

syft.serde.torch.serde.**TORCH_MFORMAT_ID**

syft.serde.torch.serde.**TORCH_ID_MFORMAT**

syft.serde.torch.serde.**torch_tensor_serializer**(*worker: AbstractWorker*, *tensor*) → bin
Strategy to serialize a tensor using Torch saver

syft.serde.torch.serde.**torch_tensor_deserializer**(*worker: AbstractWorker*, *tensor_bin*) → torch.Tensor
"Strategy to deserialize a binary input using Torch load

syft.serde.torch.serde.**numpy_tensor_serializer**(*worker: AbstractWorker*, *tensor: torch.Tensor*) → bin
Strategy to serialize a tensor using numpy npy format. If tensor requires to calculate gradients, it will be detached.

Args (torch.Tensor): an input tensor to be serialized

Returns A serialized version of the input tensor

syft.serde.torch.serde.**numpy_tensor_deserializer**(*tensor_bin*) → torch.Tensor
Strategy to deserialize a binary input in npy format into Torch tensor

Args tensor_bin: A binary representation of a tensor

Returns a Torch tensor

1.1.1.6.2 Submodules

1.1.1.6.2.1 syft.serde.compression

This file exists to provide one common place for all compression methods used in simplifying and serializing PySyft objects.

1.1.1.6.2.2 Module Contents

syft.serde.compression.**NO_COMPRESSION = 40**

syft.serde.compression.**LZ4 = 41**

syft.serde.compression.**ZSTD = 42**

syft.serde.compression.**scheme_to_bytes**

syft.serde.compression.**_apply_compress_scheme**(*decompressed_input_bin*) → tuple
Apply the selected compression scheme. By default is used LZ4

Parameters **decompressed_input_bin** – the binary to be compressed

syft.serde.compression.**apply_lz4_compression**(*decompressed_input_bin*) → tuple
Apply LZ4 compression to the input

Parameters **decompressed_input_bin** – the binary to be compressed

Returns a tuple (compressed_result, LZ4)

syft.serde.compression.**apply_zstd_compression**(*decompressed_input_bin*) → tuple
Apply ZSTD compression to the input

Parameters **decompressed_input_bin** – the binary to be compressed

Returns a tuple (compressed_result, ZSTD)

syft.serde.compression.**apply_no_compression**(*decompressed_input_bin*) → tuple
No compression is applied to the input

Parameters **decompressed_input_bin** – the binary

Returns a tuple (the binary, LZ4)

syft.serde.compression.**_compress**(*decompressed_input_bin: bin*) → bin
This function compresses a binary using the function _apply_compress_scheme if the input has been already compressed in some step, it will return it as it is

Parameters **decompressed_input_bin** (*bin*) – binary to be compressed

Returns a compressed binary

Return type bin

syft.serde.compression.**_decompress**(*binary: bin*) → bin
This function decompresses a binary using the scheme defined in the first byte of the input

Parameters **binary** (*bin*) – a compressed binary

Returns decompressed binary

Return type bin

1.1.1.6.2.3 syft.serde.serde

This file exists to provide one common place for all serialization to occur regardless of framework. By default, we serialize using msgpack and compress using lz4. If different compressions are required, the worker can override the function apply_compress_scheme.

1.1.1.6.2.4 Module Contents

syft.serde.serde.**serialize**(*obj: object, worker: AbstractWorker = None, simplified: bool = False, force_full_simplification: bool = False, strategy: Callable[[object, AbstractWorker], bin] = msgpack.serialize*) → bin

This method can serialize any object PySyft needs to send or store.

This is the high level function for serializing any object or collection of objects which PySyft needs to send over the wire. It includes three steps, Simplify, Serialize, and Compress as described inline below.

Parameters

- **obj** (*object*) – the object to be serialized
- **simplified** (*bool*) – in some cases we want to pass in data which has already been simplified - in which case we must skip double simplification - which would be bad.... so bad... so... so bad
- **force_full_simplification** (*bool*) – Some objects are only partially serialized by default. For objects where this is the case, setting this flag to True will force the entire object to be serialized. For example, setting this flag to True will cause a VirtualWorker to be serialized WITH all of its tensors while by default VirtualWorker objects only serialize a small amount of metadata.

Returns the serialized form of the object.

Return type binary

syft.serde.serde.**deserialize**(*binary: bin, worker: AbstractWorker = None, strategy: Callable[[bin, AbstractWorker], object] = msgpack.deserialize*) → object

This method can deserialize any object PySyft needs to send or store.

This is the high level function for deserializing any object or collection of objects which PySyft has sent over the wire or stored. It includes three steps, Decompress, Deserialize, and Detail as described inline below.

Parameters

- **binary** (*bin*) – the serialized object to be deserialized.
- **worker** (AbstractWorker) – the worker which is acquiring the message content, for example used to specify the owner of a tensor received(not obvious for virtual workers)
- **details** (*bool*) – there are some cases where we need to perform the decompression and deserialization part, but we don't need to detail all the message. This is the case for Plan workers for instance

Returns the deserialized form of the binary input.

Return type object

1.1.1.6.3 Package Contents

class syft.serde.**AbstractWorker**
Bases: abc.ABC

syft.serde.**serialize**(*obj: object, worker: AbstractWorker = None, simplified: bool = False, force_full_simplification: bool = False, strategy: Callable[[object, AbstractWorker], bin] = msgpack.serialize*) → bin
This method can serialize any object PySyft needs to send or store.

This is the high level function for serializing any object or collection of objects which PySyft needs to send over the wire. It includes three steps, Simplify, Serialize, and Compress as described inline below.

Parameters

- **obj** (*object*) – the object to be serialized
- **simplified** (*bool*) – in some cases we want to pass in data which has already been simplified - in which case we must skip double simplification - which would be bad.... so bad... so... so bad
- **force_full_simplification** (*bool*) – Some objects are only partially serialized by default. For objects where this is the case, setting this flag to True will force the entire object to be serialized. For example, setting this flag to True will cause a VirtualWorker to be serialized WITH all of its tensors while by default VirtualWorker objects only serialize a small amount of metadata.

Returns the serialized form of the object.

Return type binary

syft.serde.**deserialize**(*binary: bin, worker: AbstractWorker = None, strategy: Callable[[bin, AbstractWorker], object] = msgpack.deserialize*) → object
This method can deserialize any object PySyft needs to send or store.

This is the high level function for deserializing any object or collection of objects which PySyft has sent over the wire or stored. It includes three steps, Decompress, Deserialize, and Detail as described inline below.

Parameters

- **binary** (*bin*) – the serialized object to be deserialized.
- **worker** (AbstractWorker) – the worker which is acquiring the message content, for example used to specify the owner of a tensor received(not obvious for virtual workers)
- **details** (*bool*) – there are some cases where we need to perform the decompression and deserialization part, but we don't need to detail all the message. This is the case for Plan workers for instance

Returns the deserialized form of the binary input.

Return type object

1.1.1.7 syft.workers

1.1.1.7.1 Submodules

1.1.1.7.1.1 syft.workers.abstract

1.1.1.7.1.2 Module Contents

class syft.workers.abstract.**AbstractWorker**
Bases: abc.ABC

1.1.1.7.1.3 syft.workers.base

1.1.1.7.1.4 Module Contents

syft.workers.base.**logger**

class syft.workers.base.**BaseWorker**(*hook: FrameworkHook*, *id: Union[int, str] = 0*, *data: Union[List, tuple] = None*, *is_client_worker: bool = False*, *log_msgs: bool = False*, *verbose: bool = False*, *auto_add: bool = True*)
Bases: *syft.workers.abstract.AbstractWorker*, *syft.generic.object_storage.ObjectStorage*

Contains functionality to all workers.

Other workers will extend this class to inherit all functionality necessary for PySyft's protocol. Extensions of this class overrides two key methods _send_msg() and _recv_msg() which are responsible for defining the procedure for sending a binary message to another worker.

At it's core, BaseWorker (and all workers) is a collection of objects owned by a certain machine. Each worker defines how it interacts with objects on other workers as well as how other workers interact with objects owned by itself. Objects are either tensors or of any type supported by the PySyft protocol.

Parameters

- **hook** – A reference to the TorchHook object which is used to modify PyTorch with PySyft's functionality.
- **id** – An optional string or integer unique id of the worker.
- **known_workers** – An optional dictionary of all known workers on a network which this worker may need to communicate with in the future. The key of each should be each worker's unique ID and the value should be a worker class which extends BaseWorker. Extensions of BaseWorker will include advanced functionality for adding to this dictionary(node discovery). In some cases, one can initialize this with known workers to help bootstrap the network.
- **data** – Initialize workers with data on creating worker object
- **is_client_worker** – An optional boolean parameter to indicate whether this worker is associated with an end user client. If so, it assumes that the client will maintain control over when variables are instantiated or deleted as opposed to handling tensor/variable/model lifecycle internally. Set to True if this object is not where the objects will be stored, but is instead a pointer to a worker that exists elsewhere.

- **log_msgs** – An optional boolean parameter to indicate whether all messages should be saved into a log for later review. This is primarily a development/testing feature.
- **auto_add** – Determines whether to automatically add this worker to the list of known workers.

abstract _send_msg(*self*, *message: bin*, *location: BaseWorker*)
Sends message from one worker to another.

As BaseWorker implies, you should never instantiate this class by itself. Instead, you should extend BaseWorker in a new class which instantiates _send_msg and _recv_msg, each of which should specify the exact way in which two workers communicate with each other. The easiest example to study is VirtualWorker.

Parameters

- **message** – A binary message to be sent from one worker to another.
- **location** – A BaseWorker instance that lets you provide the destination to send the message.

Raises **NotImplementedError** – Method not implemented error.

abstract _recv_msg(*self*, *message: bin*)
Receives the message.

As BaseWorker implies, you should never instantiate this class by itself. Instead, you should extend BaseWorker in a new class which instantiates _send_msg and _recv_msg, each of which should specify the exact way in which two workers communicate with each other. The easiest example to study is VirtualWorker.

Parameters **message** – The binary message being received.

Raises **NotImplementedError** – Method not implemented error.

registration_enabled(*self*)

remove_worker_from_registry(*self*, *worker_id*)
Removes a worker from the dictionary of known workers. :param worker_id: id to be removed

remove_worker_from_local_worker_registry(*self*)
Removes itself from the registry of hook.local_worker.

load_data(*self, data: List[Union[FrameworkTensorType, AbstractTensor]]*)
Allows workers to be initialized with data when created

The method registers the tensor individual tensor objects.

Parameters **data** – A list of tensors

send_msg(*self*, *message: Message*, *location: BaseWorker*)
Implements the logic to send messages.

The message is serialized and sent to the specified location. The response from the location (remote worker) is deserialized and returned back.

Every message uses this method.

Parameters

- **msg_type** – A integer representing the message type.
- **message** – A Message object

- **location** – A BaseWorker instance that lets you provide the destination to send the message.

Returns The deserialized form of message from the worker at specified location.

recv_msg (*self*, *bin_message: bin*)
Implements the logic to receive messages.

The binary message is deserialized and routed to the appropriate function. And, the response serialized the returned back.

Every message uses this method.

Parameters **bin_message** – A binary serialized message.

Returns A binary message response.

send (*self, obj: Union[FrameworkTensorType, AbstractTensor], workers: BaseWorker, ptr_id: Union[str, int] = None, garbage_collect_data=None, **kwargs*)
Sends tensor to the worker(s).

Send a syft or torch tensor/object and its child, sub-child, etc (all the syft chain of children) to a worker, or a list of workers, with a given remote storage address.

Parameters

- **tensor** – A syft/framework tensor/object to send.
- **workers** – A BaseWorker object representing the worker(s) that will receive the object.
- **ptr_id** – An optional string or integer indicating the remote id of the object on the remote worker(s).
- **local_autograd** – Use autograd system on the local machine instead of PyTorch's autograd on the workers.
- **preinitialize_grad** – Initialize gradient for AutogradTensors to a tensor
- **garbage_collect_data** – argument passed down to create_pointer()

Example

```
>>> import torch
>>> import syft as sy
>>> hook = sy.TorchHook(torch)
>>> bob = sy.VirtualWorker(hook)
>>> x = torch.Tensor([1, 2, 3, 4])
>>> x.send(bob, 1000)
Will result in bob having the tensor x with id 1000
```

Returns A PointerTensor object representing the pointer to the remote worker(s).

execute_command (*self*, *message: tuple*)
Executes commands received from other workers.

Parameters **message** – A tuple specifying the command and the args.

Returns A pointer to the result.

execute_plan_command (*self*, *message: tuple*)
Executes commands related to plans.

This method is intended to execute all commands related to plans and avoiding having several new message types specific to plans.

Parameters **message** – A tuple specifying the command and args.

send_command(*self*, *recipient: BaseWorker*, *message: tuple*, *return_ids: str = None*)
Sends a command through a message to a recipient worker.

Parameters

- **recipient** – A recipient worker.
- **message** – A tuple representing the message being sent.
- **return_ids** – A list of strings indicating the ids of the tensors that should be returned as response to the command execution.

Returns A list of PointerTensors or a single PointerTensor if just one response is expected.

get_obj(*self, obj_id: Union[str, int]*)
Returns the object from registry.

Look up an object from the registry using its ID.

Parameters **obj_id** – A string or integer id of an object to look up.

respond_to_obj_req(*self*, *request_msg: tuple*)
Returns the deregistered object from registry.

Parameters **request_msg** (*tuple*) – Tuple containing object id, user credentials and reason.

register_obj(*self*, *obj: object*, *obj_id: Union[str*, *int] = None*)
Registers the specified object with the current worker node.

Selects an id for the object, assigns a list of owners, and establishes whether it's a pointer or not. This method is generally not used by the client and is instead used by internal processes (hooks and workers).

Parameters

- **obj** – A torch Tensor or Variable object to be registered.
- **obj_id** (*int or string*) – random integer between 0 and 1e10 or string uniquely identifying the object.

de_register_obj(*self*, *obj: object*, *_recurse_torch_objs: bool = True*)
De-registers the specified object with the current worker node.

Parameters

- **obj** – the object to deregister
- **_recurse_torch_objs** – A boolean indicating whether the object is more complex and needs to be explored.

send_obj(*self*, *obj: object*, *location: BaseWorker*)
Send a torch object to a worker.

Parameters

- **obj** – A torch Tensor or Variable object to be sent.
- **location** – A BaseWorker instance indicating the worker which should receive the object.

request_obj(*self, obj_id: Union[str, int], location: BaseWorker, user=None, reason: str = ''*)
Returns the requested object from specified location.

Parameters

- **obj_id** (*int or string*) – A string or integer id of an object to look up.
- **location** (BaseWorker) – A BaseWorker instance that lets you provide the lookup location.
- **user** (*object, optional*) – user credentials to perform user authentication.
- **reason** (*string, optional*) – a description of why the data scientist wants to see it.

Returns A torch Tensor or Variable object.

get_worker (*self, id_or_worker: Union[str, int, 'BaseWorker'], fail_hard: bool = False*)
Returns the worker id or instance.

Allows for resolution of worker ids to workers to happen automatically while also making the current worker aware of new ones when discovered through other processes.

If you pass in an ID, it will try to find the worker object reference within self._known_workers. If you instead pass in a reference, it will save that as a known_worker if it does not exist as one.

This method is useful because often tensors have to store only the ID to a foreign worker which may or may not be known by the worker that is de-serializing it at the time of deserialization.

Parameters

- **id_or_worker** – A string or integer id of the object to be returned or the BaseWorker object itself.
- **fail_hard** (*bool*) – A boolean parameter indicating whether we want to throw an exception when a worker is not registered at this worker or we just want to log it.

Returns A string or integer id of the worker or the BaseWorker instance representing the worker.

Example

```
>>> import syft as sy
>>> hook = sy.TorchHook(verbose=False)
>>> me = hook.local_worker
>>> bob = sy.VirtualWorker(id="bob",hook=hook, is_client_worker=False)
>>> me.add_worker([bob])
>>> bob
<syft.core.workers.virtual.VirtualWorker id:bob>
>>> # we can get the worker using it's id (1)
>>> me.get_worker('bob')
<syft.core.workers.virtual.VirtualWorker id:bob>
>>> # or we can get the worker by passing in the worker
>>> me.get_worker(bob)
<syft.core.workers.virtual.VirtualWorker id:bob>
```

_get_worker (*self, worker: AbstractWorker*)

_get_worker_based_on_id (*self, worker_id: Union[str, int], fail_hard: bool = False*)

add_worker (*self, worker: BaseWorker*)
Adds a single worker.

Adds a worker to the list of _known_workers internal to the BaseWorker. Endows this class with the ability to communicate with the remote worker being added, such as sending and receiving objects, commands, or information about the network.

Parameters **worker** (*BaseWorker*) – A BaseWorker object representing the pointer to a remote worker, which must have a unique id.

Example

```
>>> import torch
>>> import syft as sy
>>> hook = sy.TorchHook(verbose=False)
>>> me = hook.local_worker
>>> bob = sy.VirtualWorker(id="bob",hook=hook, is_client_worker=False)
>>> me.add_worker([bob])
>>> x = torch.Tensor([1,2,3,4,5])
>>> x
1
2
3
4
5
[syft.core.frameworks.torch.tensor.FloatTensor of size 5]
>>> x.send(bob)
FloatTensor[_PointerTensor - id:9121428371 owner:0 loc:bob
            id@loc:47416674672]
>>> x.get()
1
2
3
4
5
[syft.core.frameworks.torch.tensor.FloatTensor of size 5]
```

add_workers (*self, workers: List['BaseWorker']*)
Adds several workers in a single call.

Parameters **workers** – A list of BaseWorker representing the workers to add.

__str__ (*self*)
Returns the string representation of BaseWorker.

A to-string method for all classes that extend BaseWorker.

Returns The Type and ID of the worker

Example

A VirtualWorker instance with id 'bob' would return a string value of. >>> import syft as sy >>> bob = sy.VirtualWorker(id="bob") >>> bob <syft.workers.virtual.VirtualWorker id:bob>

Note: __repr__ calls this method by default.

__repr__ (*self*)
Returns the official string representation of BaseWorker.

__getitem__ (*self, idx*)

static is_tensor_none (*obj*)

request_is_remote_tensor_none(*self*, *pointer: PointerTensor*)
Sends a request to the remote worker that holds the target a pointer if the value of the remote tensor is None or not. Note that the pointer must be valid: if there is no target (which is different from having a target equal to None), it will return an error.

Parameters **pointer** – The pointer on which we can to get information.

Returns A boolean stating if the remote value is None.

static get_tensor_shape(*tensor: FrameworkTensorType*)
Returns the shape of a tensor casted into a list, to bypass the serialization of a torch.Size object.

Parameters **tensor** – A torch.Tensor.

Returns A list containing the tensor shape.

request_remote_tensor_shape(*self*, *pointer: PointerTensor*)
Sends a request to the remote worker that holds the target a pointer to have its shape.

Parameters **pointer** – A pointer on which we want to get the shape.

Returns A torch.Size object for the shape.

fetch_plan(*self, plan_id: Union[str, int], location: BaseWorker, copy: bool = False*)
Fetchs a copy of a the plan with the given *plan_id* from the worker registry.

This method is executed for local execution.

Parameters **plan_id** – A string indicating the plan id.

Returns A plan if a plan with the given *plan_id* exists. Returns None otherwise.

_fetch_plan_remote(*self, plan_id: Union[str, int], copy: bool*)
Fetches a copy of a the plan with the given *plan_id* from the worker registry.

This method is executed for remote execution.

Parameters **plan_id** – A string indicating the plan id.

Returns A plan if a plan with the given *plan_id* exists. Returns None otherwise.

fetch_protocol(*self, protocol_id: Union[str, int], location: BaseWorker, copy: bool = False*)
Fetch a copy of a the protocol with the given *protocol_id* from the worker registry.

This method is executed for local execution.

Parameters **protocol_id** – A string indicating the protocol id.

Returns A protocol if a protocol with the given *protocol_id* exists. Returns None otherwise.

_fetch_protocol_remote(*self, protocol_id: Union[str, int], copy: bool*)
Target function of fetch_protocol, find and return a protocol

search(*self, query: Union[List[Union[str, int]], str, int]*)
Search for a match between the query terms and a tensor's Id, Tag, or Description.

Note that the query is an AND query meaning that every item in the list of strings (query*) must be found somewhere on the tensor in order for it to be included in the results.

Parameters

- **query** – A list of strings to match against.
- **me** – A reference to the worker calling the search.

Returns A list of PointerTensors.

request_search(*self, query: List[str], location: BaseWorker*)

_get_msg(*self*, *index*)
Returns a decrypted message from msg_history. Mostly useful for testing.

Parameters **index** – the index of the message you'd like to receive.

Returns A decrypted messaging.Message object.

static create_message_execute_command(*command_name: str*, *command_owner=None*, *return_ids=None*, **args*, ***kwargs*)
helper function creating a message tuple for the execute_command call

Parameters

- **command_name** – name of the command that shall be called
- **command_owner** – owner of the function (None for torch functions, "self" for classes derived from workers.base or ptr_id for remote objects
- **return_ids** – optionally set the ids of the return values (for remote objects)
- ***args** – will be passed to the call of command_name
- ****kwargs** – will be passed to the call of command_name

Returns (command_name, command_owner, args, kwargs), return_ids

Return type tuple

property serializer(*self*, *workers=None*)
Define the serialization strategy to adopt depending on the workers it's connected to. This is relevant in particular for Tensors which can be serialized in an efficient way between workers which share the same Deep Learning framework, but must be converted to lists or json-like objects in other cases.

Parameters **workers** – (Optional) the list of workers involved in the serialization. If not provided, self._known_workers is used.

Returns 'all': serialization must be compatible with all kinds of workers 'torch': serialization will only work between workers that support PyTorch (more to come: 'tensorflow', 'numpy', etc)

Return type A str code

static simplify(*_worker: AbstractWorker*, *worker: AbstractWorker*)

static detail(*worker: AbstractWorker*, *worker_tuple: tuple*)
This function reconstructs a PlanPointer given it's attributes in form of a tuple.

Parameters

- **worker** – the worker doing the deserialization
- **plan_pointer_tuple** – a tuple holding the attributes of the PlanPointer

Returns A worker id or worker instance.

static force_simplify(*_worker: AbstractWorker*, *worker: AbstractWorker*)

static force_detail(*worker: AbstractWorker*, *worker_tuple: tuple*)

1.1.1.7.1.5 syft.workers.node_client

1.1.1.7.1.6 Module Contents

class syft.workers.node_client.**NodeClient**(*hook*, *address*, *credential: AbstractCredential = None*, *id: Union[int, str] = 0*, *is_client_worker: bool = False*, *log_msgs: bool = False*, *verbose: bool = False*, *encoding: str = 'ISO-8859-1'*)

Bases: *syft.workers.websocket_client.WebsocketClientWorker*, *syft.federated.federated_client.FederatedClient*

Federated Node Client.

property url(*self*)

Get Node URL Address. :returns: Node's address. :rtype: address (str)

property models(*self*)

Get models stored at remote node.

Returns List of models stored in this node.

Return type models (List)

_authenticate(*self*)

Perform Authentication Process using credentials grid credentials. :raises RuntimeError: If authentication process fail.

_update_node_reference(*self*, *new_id: str*)

Update worker references changing node id references at hook structure. :param new_id: New worker ID. :type new_id: str

_parse_address(*self*, *address: str*)

Parse Address string to define secure flag and split into host and port. :param address: Adress of remote worker. :type address: str

_get_node_id(*self*)

Get Node ID from remote node worker :returns: node id used by remote worker. :rtype: node_id (str)

_forward_json_to_websocket_server_worker(*self*, *message: dict*)

Prepare/send a JSON message to a remote node and receive the response. :param message: message payload. :type message: dict

Returns response payload.

Return type node_response (dict)

_forward_to_websocket_server_worker(*self*, *message: bin*)

Send a bin message to a remote node and receive the response. :param message: message payload. :type message: bytes

Returns response payload.

Return type node_response (bytes)

_return_bool_result(*self*, *result*, *return_key=None*)

connect_nodes(*self*, *node*)

Connect two remote workers between each other. :param node: Node that will be connected with this remote worker. :type node: WebsocketFederatedClient

Returns node response.

Return type node_response (dict)

serve_model(*self*, *model*, *model_id: str = None*, *mpc: bool = False*, *allow_download: bool = False*, *allow_remote_inference: bool = False*)

Hosts the model and optionally serve it using a Socket / Rest API. :param model: A jit model or Syft Plan. :param model_id: An integer/string representing the model id. :type model_id: str :param If it isn't provided and the model is a Plan we use model.id,: :param if the model is a jit model we raise an exception.: :param allow_download: Allow to copy the model to run it locally. :type allow_download: bool :param allow_remote_inference: Allow to run remote inferences. :type allow_remote_inference: bool

Returns True if model was served sucessfully.

Return type result (bool)

Raises

- **ValueError** – model_id isn't provided and model is a jit model.
- **RunTimeError** – if there was a problem during model serving.

run_remote_inference(*self*, *model_id*, *data*)

Run a dataset inference using a remote model.

Parameters

- **model_id** (*str*) – Model ID.
- **data** (*Tensor*) – dataset to be inferred.

Returns Inference result

Return type inference (Tensor)

Raises **RuntimeError** – If an unexpected behavior happen.

delete_model(*self*, *model_id: str*)

Delete a model previously registered.

Parameters **model_id** (String) – ID of the model that will be deleted.

Returns If succeeded, return True.

Return type result (bool)

__str__(*self*)

1.1.1.7.1.7 syft.workers.tfe

To be extended in the near future.

1.1.1.7.1.8 Module Contents

syft.workers.tfe.**logger**

syft.workers.tfe.**_TMP_DIR**

class syft.workers.tfe.**TFEWorker**(*host=None*, *auto_managed=True*)

start(*self*, *player_name*, *cluster*)

Start the worker as a player in the given cluster. Depending on whether the worker was constructed with a host or not this may launch a subprocess running a TensorFlow server.

stop(*self*)

Stop the worker. This will shutdown any TensorFlow server launched in *start()*.

connect_to_model(*self*, *input_shape*, *output_shape*, *cluster*, *sess=None*)
Connect to a TF Encrypted model being served by the given cluster.

This must be done before querying the model.

query_model(*self*, *data*)
Encrypt data and sent it as input to the model being served.

This will block until a result is ready, and requires that a connection to the model has already been established via *connect_to_model()*.

query_model_async(*self*, *data*)
Asynchronous version of *query_model* that will not block until a result is ready. Call *query_model_join* to retrive result.

This requires that a connection to the model has already been established via *connect_to_model()*.

query_model_join(*self*)
Retrives the result from calling *query_model_async*, blocking until ready.

class syft.workers.tfe.**TFECluster**(**workers*)
A TFECluster represents a group of TFEWorkers that are aware about each other and collectively perform an encrypted computation.

property workers(*self*)

start(*self*)
Start all workers in the cluster.

stop(*self*)
Stop all workers in the cluster.

_build_cluster(*self*, *workers*)

1.1.1.7.1.9 syft.workers.virtual

1.1.1.7.1.10 Module Contents

class syft.workers.virtual.**VirtualWorker**
Bases: *syft.workers.base.BaseWorker*, *syft.federated.federated_client.FederatedClient*

_send_msg(*self*, *message: bin*, *location: BaseWorker*)

_recv_msg(*self*, *message: bin*)

1.1.1.7.1.11 syft.workers.websocket_client

1.1.1.7.1.12 Module Contents

syft.workers.websocket_client.**logger**

syft.workers.websocket_client.**TIMEOUT_INTERVAL = 999999**

class syft.workers.websocket_client.**WebsocketClientWorker**(*hook*, *host: str*, *port: int*, *secure: bool = False*, *id: Union[int, str] = 0*, *is_client_worker: bool = False*, *log_msgs: bool = False*, *verbose: bool = False*, *data: List[Union[torch.Tensor, AbstractTensor]] = None*)

Bases: *syft.workers.base.BaseWorker*

property url(*self*)

connect(*self*)

close(*self*)

search(*self*, *query*)

_send_msg(*self*, *message: bin*, *location=None*)

_forward_to_websocket_server_worker(*self*, *message: bin*)

_recv_msg(*self*, *message: bin*)
Forwards a message to the WebsocketServerWorker

_send_msg_and_deserialize(*self*, *command_name: str*, **args*, ***kwargs*)

list_objects_remote(*self*)

objects_count_remote(*self*)

clear_objects_remote(*self*)

async async_fit(*self*, *dataset_key: str*, *return_ids: List[int] = None*)
Asynchronous call to fit function on the remote location.

Parameters

- **dataset_key** – Identifier of the dataset which shall be used for the training.
- **return_ids** – List of return ids.

Returns See return value of the FederatedClient.fit() method.

fit(*self*, *dataset_key: str*, ***kwargs*)
Call the fit() method on the remote worker (WebsocketServerWorker instance).

Note: The argument return_ids is provided as kwargs as otherwise there is a miss-match with the signature in VirtualWorker.fit() method. This is important to be able to switch between virtual and websocket workers.

Parameters

- **dataset_key** – Identifier of the dataset which shall be used for the training.
- ****kwargs** – return_ids: List[str]

evaluate(*self*, *dataset_key: str*, *return_histograms: bool = False*, *nr_bins: int = -1*, *return_loss=True*, *return_raw_accuracy: bool = True*)
Call the evaluate() method on the remote worker (WebsocketServerWorker instance).

Parameters

- **dataset_key** – Identifier of the local dataset that shall be used for training.
- **return_histograms** – If True, calculate the histograms of predicted classes.
- **nr_bins** – Used together with calculate_histograms. Provide the number of classes/bins.
- **return_loss** – If True, loss is calculated additionally.
- **return_raw_accuracy** – If True, return nr_correct_predictions and nr_predictions

Returns

- loss: avg loss on data set, None if not calculated.
- nr_correct_predictions: number of correct predictions.
- nr_predictions: total number of predictions.
- histogram_predictions: histogram of predictions.
- histogram_target: histogram of target values in the dataset.

Return type Dictionary containing depending on the provided flags

__str__(*self*)

Returns the string representation of a Websocket worker.

A to-string method for websocket workers that includes information from the websocket server

Returns The Type and ID of the worker

1.1.1.7.1.13 syft.workers.websocket_server

1.1.1.7.1.14 Module Contents

class syft.workers.websocket_server.**WebsocketServerWorker**(*hook*, *host: str*, *port: int*, *id: Union[int, str] = 0*, *log_msgs: bool = False*, *verbose: bool = False*, *data: List[Union[torch.Tensor, AbstractTensor]] = None*, *loop=None*, *cert_path: str = None*, *key_path: str = None*)

Bases: *syft.workers.virtual.VirtualWorker*, *syft.federated.federated_client.FederatedClient*

async _consumer_handler(*self*, *websocket: websockets.WebSocketCommonProtocol*)

This handler listens for messages from WebsocketClientWorker objects.

Parameters **websocket** – the connection object to receive messages from and add them into the queue.

async _producer_handler(*self*, *websocket: websockets.WebSocketCommonProtocol*)

This handler listens to the queue and processes messages as they arrive.

Parameters **websocket** – the connection object we use to send responses back to the client.

_recv_msg(*self*, *message: bin*)

async _handler(*self*, *websocket: websockets.WebSocketCommonProtocol*, **unused_args*)

Setup the consumer and producer response handlers with asyncio.

Parameters **websocket** – the websocket connection to the client

start (*self*)
Start the server

list_objects (*self*, **args*)

objects_count (*self*, **args*)

1.1.2 Submodules

1.1.2.1 syft.codes

1.1.2.1.1 Module Contents

class syft.codes.**PLAN_CMDS**
Bases: object

FETCH_PLAN = fetch_plan

FETCH_PROTOCOL = fetch_protocol

class syft.codes.**TENSOR_SERIALIZATION**
Bases: object

TORCH = torch

NUMPY = numpy

TF = tf

ALL = all

class syft.codes.**GATEWAY_ENDPOINTS**
Bases: object

SEARCH_TAGS = /search

SEARCH_MODEL = /search-model

SEARCH_ENCRYPTED_MODEL = /search-encrypted-model

SELECT_MODEL_HOST = /choose-model-host

SELECT_ENCRYPTED_MODEL_HOSTS = /choose-encrypted-model-host

class syft.codes.**REQUEST_MSG**
Bases: object

TYPE_FIELD = type

GET_ID = get-id

CONNECT_NODE = connect-node

HOST_MODEL = host-model

RUN_INFERENCE = run-inference

LIST_MODELS = list-models

DELETE_MODEL = delete-model

RUN_INFERENCE = run-inference

AUTHENTICATE = authentication

class syft.codes.**RESPONSE_MSG**
Bases: object

NODE_ID = id

ERROR = error

SUCCESS = success

MODELS = models

INFERENCE_RESULT = prediction

1.1.2.2 syft.dependency_check

1.1.2.2.1 Module Contents

syft.dependency_check.**logger**

syft.dependency_check.**pstf_spec**

syft.dependency_check.**tfe_spec**

syft.dependency_check.**tfe_available**

syft.dependency_check.**torch_spec**

syft.dependency_check.**torch_available**

1.1.2.3 syft.exceptions

Specific Pysyft exceptions.

1.1.2.3.1 Module Contents

exception syft.exceptions.**DependencyError**(*package*, *pypi_alias=None*)
Bases: Exception

exception syft.exceptions.**PureFrameworkTensorFoundError**
Bases: BaseException

Exception raised for errors in the input. This error is used in a recursive analysis of the args provided as an input of a function, to break the recursion if a FrameworkTensor is found as it means that _probably_ all the tensors are pure torch/tensorflow and the function can be applied natively on this input.

expression -- input expression in which the error occurred

message -- explanation of the error

exception syft.exceptions.**RemoteObjectFoundError**(*pointer*)
Bases: BaseException

Exception raised for errors in the input. This error is used in a context similar to PureFrameworkTensorFoundError but to indicate that a Pointer to a remote tensor was found in the input and thus that the command should be send elsewhere. The pointer retrieved by the error gives the location where the command should be sent.

expression -- input expression in which the error occurred

message -- explanation of the error

exception syft.exceptions.**InvalidTensorForRemoteGet**(*tensor: object*)
Bases: Exception

Raised when a chain of pointer tensors is not provided for *remote_get*.

exception syft.exceptions.**WorkerNotFoundException**
Bases: Exception

Raised when a non-existent worker is requested.

exception syft.exceptions.**CompressionNotFoundException**
Bases: Exception

Raised when a non existent compression/decompression scheme is requested.

exception syft.exceptions.**CannotRequestObjectAttribute**
Bases: Exception

Raised when .get() is called on a pointer which points to an attribute of another object.

exception syft.exceptions.**TensorsNotCollocatedException**(*tensor_a*, *tensor_b*, *attr='a method'*)
Bases: Exception

Raised when a command is executed on two tensors which are not on the same machine. The goal is to provide as useful input as possible to help the user identify which tensors are where so that they can debug which one needs to be moved.

exception syft.exceptions.**ResponseSignatureError**(*ids_generated=None*)
Bases: Exception

Raised when the return of a hooked function is not correctly predicted (when defining in advance ids for results)

get_attributes(*self*)
Specify all the attributes need to report an error correctly.

static simplify(*worker: sy.workers.AbstractWorker*, *e*)
Serialize information about an Exception which was raised to forward it

static detail(*worker: sy.workers.AbstractWorker, error_tuple: Tuple[str, str, dict]*)
Detail and re-raise an Exception forwarded by another worker

exception syft.exceptions.**SendNotPermittedError**
Bases: Exception

Raised when calling send on a tensor which does not allow send to be called on it. This can happen do to sensitivity being too high

static simplify(*worker: sy.workers.AbstractWorker*, *e*)
Serialize information about an Exception which was raised to forward it

static detail(*worker: sy.workers.AbstractWorker, error_tuple: Tuple[str, str, dict]*)
Detail and re-raise an Exception forwarded by another worker

exception syft.exceptions.**GetNotPermittedError**
Bases: Exception

Raised when calling get on a pointer to a tensor which does not allow get to be called on it. This can happen do to sensitivity being too high

static simplify(*worker: sy.workers.AbstractWorker*, *e*)
Serialize information about an Exception which was raised to forward it

static detail(*worker: sy.workers.AbstractWorker, error_tuple: Tuple[str, str, dict]*)
Detail and re-raise an Exception forwarded by another worker

exception syft.exceptions.**IdNotUniqueError**
Bases: Exception

Raised by the ID Provider when setting ids that have already been generated

exception syft.exceptions.**PlanCommandUnknownError**(*command_name: object*)
Bases: Exception

Raised when an unknown plan command execution is requested.

exception syft.exceptions.**ObjectNotFoundError**(*obj_id*, *worker*)
Bases: Exception

Raised when object with given object id is not found on worker

obj_id -- id of the object with which the interaction is attempted

worker -- virtual worker on which the interaction is attempted

exception syft.exceptions.**InvalidProtocolFileError**
Bases: Exception

Raised when PySyft protocol file cannot be loaded.

exception syft.exceptions.**UndefinedProtocolTypeError**
Bases: Exception

Raised when trying to serialize type that is not defined in protocol file.

exception syft.exceptions.**UndefinedProtocolTypePropertyError**
Bases: Exception

Raised when trying to get protocol type property that is not defined in protocol file.

syft.exceptions.**route_method_exception**(*exception*, *self*, *args*, *kwargs*)

1.1.2.4 syft.sandbox

1.1.2.4.1 Module Contents

syft.sandbox.**create_sandbox**(*gbs*, *verbose=True*, *download_data=True*)
There's some boilerplate stuff that most people who are just playing around would like to have. This will create that for you

syft.sandbox.**hook**(*gbs*)

1.1.2.5 syft.version

1.1.2.5.1 Module Contents

syft.version.**__version__ = 0.2.3a1**

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

syft, 1
syft.codes, 148
syft.dependency_check, 149
syft.exceptions, 149
syft.federated, 1
syft.federated.federated_client, 1
syft.federated.train_config, 2
syft.frameworks, 4
syft.frameworks.keras, 4
syft.frameworks.keras.hook, 5
syft.frameworks.keras.layers, 4
syft.frameworks.keras.layers.constructor, 4
syft.frameworks.keras.model, 4
syft.frameworks.keras.model.sequential, 4
syft.frameworks.tensorflow, 6
syft.frameworks.torch, 6
syft.frameworks.torch.dp, 6
syft.frameworks.torch.dp.pate, 6
syft.frameworks.torch.fl, 9
syft.frameworks.torch.fl.dataloader, 9
syft.frameworks.torch.fl.dataset, 10
syft.frameworks.torch.fl.utils, 11
syft.frameworks.torch.functions, 61
syft.frameworks.torch.he, 14
syft.frameworks.torch.he.paillier, 14
syft.frameworks.torch.hook, 15
syft.frameworks.torch.hook.hook, 15
syft.frameworks.torch.hook.hook_args, 17
syft.frameworks.torch.linalg, 18
syft.frameworks.torch.linalg.lr, 18
syft.frameworks.torch.linalg.operations, 20
syft.frameworks.torch.mpc, 25
syft.frameworks.torch.mpc.beaver, 25
syft.frameworks.torch.mpc.securenn, 25
syft.frameworks.torch.mpc.spdz, 27
syft.frameworks.torch.nn, 28
syft.frameworks.torch.nn.conv, 28
syft.frameworks.torch.nn.pool, 28
syft.frameworks.torch.nn.rnn, 29
syft.frameworks.torch.tensors, 32
syft.frameworks.torch.tensors.decorators, 32
syft.frameworks.torch.tensors.decorators.logging, 32
syft.frameworks.torch.tensors.interpreters, 33
syft.frameworks.torch.tensors.interpreters.additive 33
syft.frameworks.torch.tensors.interpreters.autograd 38
syft.frameworks.torch.tensors.interpreters.build_gr 40
syft.frameworks.torch.tensors.interpreters.crt_prec 41
syft.frameworks.torch.tensors.interpreters.gradient 43
syft.frameworks.torch.tensors.interpreters.gradient 44
syft.frameworks.torch.tensors.interpreters.hook, 45
syft.frameworks.torch.tensors.interpreters.large_pr 46
syft.frameworks.torch.tensors.interpreters.native, 48
syft.frameworks.torch.tensors.interpreters.numpy, 52
syft.frameworks.torch.tensors.interpreters.paillier 53
syft.frameworks.torch.tensors.interpreters.placehol 54
syft.frameworks.torch.tensors.interpreters.plusismi 56
syft.frameworks.torch.tensors.interpreters.polynomi 56
syft.frameworks.torch.tensors.interpreters.precisio 56
syft.frameworks.torch.tensors.interpreters.private, 60
syft.frameworks.torch.torch_attributes,

61
syft.generic, 62
syft.generic.frameworks, 62
syft.generic.frameworks.attributes, 71
syft.generic.frameworks.hook, 62
syft.generic.frameworks.hook.hook, 62
syft.generic.frameworks.hook.hook_args, 65
syft.generic.frameworks.hook.trace, 70
syft.generic.frameworks.overload, 72
syft.generic.frameworks.remote, 72
syft.generic.frameworks.types, 72
syft.generic.id_provider, 86
syft.generic.metrics, 87
syft.generic.object, 87
syft.generic.object_storage, 89
syft.generic.pointers, 73
syft.generic.pointers.callable_pointer, 73
syft.generic.pointers.multi_pointer, 74
syft.generic.pointers.object_pointer, 76
syft.generic.pointers.object_wrapper, 79
syft.generic.pointers.pointer_plan, 80
syft.generic.pointers.pointer_protocol, 81
syft.generic.pointers.pointer_tensor, 82
syft.generic.pointers.string_pointer, 86
syft.generic.string, 90
syft.generic.tensor, 92
syft.grid, 93
syft.grid.abstract_grid, 94
syft.grid.authentication, 93
syft.grid.authentication.account, 93
syft.grid.authentication.credential, 94
syft.grid.private_grid, 94
syft.grid.public_grid, 96
syft.messaging, 98
syft.messaging.message, 105
syft.messaging.plan, 98
syft.messaging.plan.plan, 98
syft.messaging.plan.state, 101
syft.messaging.protocol, 111
syft.sandbox, 151
syft.serde, 114
syft.serde.compression, 132
syft.serde.msgpack, 114
syft.serde.msgpack.native_serde, 114
syft.serde.msgpack.proto, 119
syft.serde.msgpack.serde, 120
syft.serde.msgpack.torch_serde, 122
syft.serde.protobuf, 125
syft.serde.protobuf.native_serde, 125
syft.serde.protobuf.proto, 125
syft.serde.protobuf.serde, 126
syft.serde.protobuf.torch_serde, 128
syft.serde.serde, 133
syft.serde.torch, 131
syft.serde.torch.serde, 131
syft.version, 151
syft.workers, 135
syft.workers.abstract, 135
syft.workers.base, 135
syft.workers.node_client, 143
syft.workers.tfe, 144
syft.workers.virtual, 145
syft.workers.websocket_client, 145
syft.workers.websocket_server, 147

INDEX

Symbols

_DataLoaderIter (*class in syft.frameworks.torch.fl.dataloader*), 9
_DataLoaderOneWorkerIter (*class in syft.frameworks.torch.fl.dataloader*), 9
_TMP_DIR (*in module syft.workers.tfe*), 144
__add__ (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor attribute*), 33
__add__ (*syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionTensor attribute*), 41
__add__ (*syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor attribute*), 46
__add__ (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor attribute*), 56
__add__() (*syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor method*), 39
__add__() (*syft.frameworks.torch.tensors.interpreters.paillier.PaillierTensor method*), 53
__add__() (*syft.generic.pointers.multi_pointer.MultiPointerTensor method*), 74
__add__() (*syft.generic.string.String method*), 91
__bool__() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 33
__call__() (*syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc method*), 44
__call__() (*syft.generic.pointers.callable_pointer.CallablePointer method*), 73
__call__() (*syft.generic.pointers.object_wrapper.ObjectWrapper method*), 79
__call__() (*syft.generic.pointers.pointer_plan.PointerPlan method*), 80
__call__() (*syft.messaging.plan.Plan method*), 104
__call__() (*syft.messaging.plan.func2plan method*), 102
__call__() (*syft.messaging.plan.plan.Plan method*), 100
__call__() (*syft.messaging.plan.plan.func2plan method*), 98
__call__() (*syft.messaging.protocol.Protocol method*), 112
__connect_with_node() (*syft.grid.public_grid.PublicGridNetwork method*), 97
__del__() (*syft.generic.pointers.object_pointer.ObjectPointer method*), 78
__del__() (*syft.generic.pointers.pointer_plan.PointerPlan method*), 81
__eq__ (*syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionTensor attribute*), 41
__eq__ (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor attribute*), 56
__eq__() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 37
__eq__() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 49
__eq__() (*syft.generic.pointers.multi_pointer.MultiPointerTensor method*), 74
__eq__() (*syft.generic.pointers.pointer_tensor.PointerTensor method*), 85
__ge__() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 37
__ge__() (*syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor method*), 39
__ge__() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 58
__getattribute__() (*syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor method*), 39
__getitem__() (*syft.frameworks.torch.fl.BaseDataset method*), 13
__getitem__() (*syft.frameworks.torch.fl.FederatedDataset method*), 13
__getitem__() (*syft.frameworks.torch.fl.dataset.BaseDataset method*), 10
__getitem__() (*syft.frameworks.torch.fl.dataset.FederatedDataset method*), 11
__getitem__() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 35
__getitem__() (*syft.workers.base.BaseWorker method*), 140
__gt__ (*syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor attribute*), 47
__gt__() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 36

__gt__() (syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor method), 39
__gt__() (syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method), 58
__hook_properties() (syft.frameworks.torch.hook.hook.TorchHook method), 16
__iadd__() (syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor method), 39
__iadd__() (syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor method), 47
__iadd__() (syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method), 57
__imul__() (syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method), 36
__imul__() (syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor method), 47
__imul__() (syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method), 57
__initialized (syft.frameworks.torch.fl.FederatedDataLoader attribute), 14
__initialized (syft.frameworks.torch.fl.dataloader.FederatedDataLoader attribute), 10
__isub__() (syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor method), 39
__isub__() (syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor method), 47
__isub__() (syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method), 57
__iter__() (syft.frameworks.torch.fl.FederatedDataLoader method), 14
__iter__() (syft.frameworks.torch.fl.dataloader.FederatedDataLoader method), 10
__iter__() (syft.frameworks.torch.fl.dataloader._DataLoaderIter method), 9
__iter__() (syft.frameworks.torch.fl.dataloader._DataLoaderOneWorkerIter method), 9
__itruediv__() (syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method), 36
__itruediv__() (syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method), 57
__le__() (syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method), 37
__le__() (syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor method), 39
__le__() (syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method), 58
__len__() (syft.frameworks.torch.fl.BaseDataset method), 13
__len__() (syft.frameworks.torch.fl.FederatedDataLoader method), 14
__len__() (syft.frameworks.torch.fl.FederatedDataset method), 14
__len__() (syft.frameworks.torch.fl.dataloader.FederatedDataLoader method), 10
__len__() (syft.frameworks.torch.fl.dataloader._DataLoaderIter method), 9
__len__() (syft.frameworks.torch.fl.dataset.BaseDataset method), 10
__len__() (syft.frameworks.torch.fl.dataset.FederatedDataset method), 11
__long() (syft.generic.tensor.AbstractTensor method), 92
__lshift__() (syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor attribute), 47
__lshift__() (syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method), 37
__lt__() (syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor method), 39
__lt__() (syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method), 58
__matmul__() (syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor attribute), 56
__matmul__() (syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method), 36
__matmul__() (syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor method), 39
__matmul__() (syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor attribute), 46
__mod__() (syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method), 36
__mod__() (syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionTensor attribute), 41
__mul__() (syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor attribute), 46
__mul__() (syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor attribute), 56
__mul__() (syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method), 36
__mul__() (syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor method), 39
__mul__() (syft.frameworks.torch.tensors.interpreters.paillier.PaillierTensor method), 53
__multi_pointer() (syft.frameworks.torch.tensors.interpreters.multi_pointer.MultiPointerTensor method), 74
__ne__() (syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionTensor method), 42
__new_args() (syft.frameworks.torch.fl.FederatedDataLoader method), 14
__new_args() (syft.frameworks.torch.fl.dataloader.FederatedDataLoader method), 10
__next__() (syft.frameworks.torch.fl.dataloader._DataLoaderIter method), 9
__next__() (syft.frameworks.torch.fl.dataloader._DataLoaderOneWorkerIter method), 9
__pow__() (syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor attribute), 33
__pow__() (syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor

attribute), 56
__pow__() (syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor method), 39
__radd__ (syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor attribute), 33
__radd__ (syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionTensor attribute), 41
__radd__ (syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor attribute), 56
__repr__ (syft.frameworks.torch.tensors.interpreters.placeholder.PlaceHolder attribute), 54
__repr__() (syft.frameworks.torch.fl.FederatedDataset method), 14
__repr__() (syft.frameworks.torch.fl.dataset.FederatedDataset method), 11
__repr__() (syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method), 33
__repr__() (syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionTensor method), 42
__repr__() (syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc method), 44
__repr__() (syft.frameworks.torch.tensors.interpreters.native.TorchTensor method), 49
__repr__() (syft.generic.object.AbstractObject method), 87
__repr__() (syft.generic.pointers.object_pointer.ObjectPointer method), 78
__repr__() (syft.generic.pointers.object_wrapper.ObjectWrapper method), 79
__repr__() (syft.messaging.message.Message method), 106
__repr__() (syft.messaging.plan.Plan method), 104
__repr__() (syft.messaging.plan.plan.Plan method), 100
__repr__() (syft.messaging.plan.state.State method), 101
__repr__() (syft.messaging.protocol.Protocol method), 114
__repr__() (syft.workers.base.BaseWorker method), 140
__rmul__ (syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionTensor attribute), 41
__rsub__() (syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method), 35
__rsub__() (syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionTensor method), 42
__rsub__() (syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method), 57
__setattr__() (syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc method), 44
__setattr__() (syft.messaging.plan.Plan method), 104
__setattr__() (syft.messaging.plan.plan.Plan method), 100
__str__() (syft.federated.train_config.TrainConfig method), 2
__str__() (syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method), 35
__str__() (syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionTensor method), 42
__str__() (syft.frameworks.torch.tensors.interpreters.native.TorchTensor method), 49
__str__() (syft.frameworks.torch.tensors.interpreters.placeholder.PlaceHolder method), 55
__str__() (syft.generic.object.AbstractObject method), 87
__str__() (syft.generic.pointers.multi_pointer.MultiPointerTensor method), 74
__str__() (syft.generic.pointers.object_pointer.ObjectPointer method), 78
__str__() (syft.generic.pointers.object_wrapper.ObjectWrapper method), 79
__str__() (syft.generic.pointers.pointer_plan.PointerPlan method), 80
__str__() (syft.grid.authentication.account.AccountCredential method), 93
__str__() (syft.messaging.message.Message method), 106
__str__() (syft.messaging.plan.Plan method), 104
__str__() (syft.messaging.plan.plan.Plan method), 100
__str__() (syft.messaging.plan.state.State method), 101
__str__() (syft.messaging.protocol.Protocol method), 114
__str__() (syft.workers.base.BaseWorker method), 140
__str__() (syft.workers.node_client.NodeClient method), 144
__str__() (syft.workers.websocket_client.WebsocketClientWorker method), 147
__sub__ (syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor attribute), 33
__sub__ (syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionTensor attribute), 41
__sub__ (syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor attribute), 46
__sub__ (syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor attribute), 55
__sub__() (syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor method), 39
__sub__() (syft.frameworks.torch.tensors.interpreters.paillier.PaillierTensor method), 52
__truediv__ (syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor attribute), 33
__truediv__ (syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionTensor attribute), 41
__truediv__ (syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor

attribute), 56
__truediv__() (*syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor method*), 39
__version__ (*in module syft.version*), 151
_add_intercept() (*syft.frameworks.torch.linalg.EncryptedLinearRegression static method*), 23
_add_intercept() (*syft.frameworks.torch.linalg.lr.EncryptedLinearRegression static method*), 19
_add_registration_to___init__() (*syft.generic.frameworks.hook.hook.FrameworkHook method*), 63
_apply_args() (*in module syft.generic.object*), 89
_apply_compress_scheme() (*in module syft.serde.compression*), 132
_apply_time_step() (*syft.frameworks.torch.nn.rnn.RNNBase method*), 29
_args_not_supported_by_tfe (*in module syft.frameworks.keras.model.sequential*), 4
_ask_gateway() (*syft.grid.public_grid.PublicGridNetwork method*), 97
_assert_is_resolved() (*syft.messaging.protocol.Protocol method*), 114
_authenticate() (*syft.workers.node_client.NodeClient method*), 143
_backward_func() (*syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor static method*), 48
_bufferize() (*in module syft.serde.protobuf.serde*), 127
_bufferize_arg() (*syft.messaging.message.Operation static method*), 108
_bufferize_args() (*syft.messaging.message.Operation static method*), 108
_bufferize_none() (*in module syft.serde.protobuf.native_serde*), 125
_bufferize_script_function() (*in module syft.serde.protobuf.torch_serde*), 129
_bufferize_script_module() (*in module syft.serde.protobuf.torch_serde*), 129
_bufferize_torch_device() (*in module syft.serde.protobuf.torch_serde*), 129
_bufferize_torch_parameter() (*in module syft.serde.protobuf.torch_serde*), 129
_bufferize_torch_size() (*in module syft.serde.protobuf.torch_serde*), 130
_bufferize_torch_tensor() (*in module syft.serde.protobuf.torch_serde*), 128
_bufferize_traced_module() (*in module syft.serde.protobuf.torch_serde*), 129
_build_cluster() (*syft.workers.tfe.TFECluster method*), 145
_build_optimizer() (*syft.federated.federated_client.FederatedClient method*), 1
_check_node_type() (*syft.grid.abstract_grid.AbstractGrid method*), 98
_check_ptrs() (*syft.frameworks.torch.linalg.DASH method*), 24
_check_ptrs() (*syft.frameworks.torch.linalg.EncryptedLinearRegression method*), 23
_check_ptrs() (*syft.frameworks.torch.linalg.lr.DASH method*), 20
_check_ptrs() (*syft.frameworks.torch.linalg.lr.EncryptedLinearRegression method*), 19
_check_train_config() (*syft.federated.federated_client.FederatedClient method*), 1
_command_guard() (*syft.generic.frameworks.attributes.FrameworkAttributes method*), 71
_compress() (*in module syft.serde.compression*), 132
_compute_pvalues() (*syft.frameworks.torch.linalg.DASH method*), 24
_compute_pvalues() (*syft.frameworks.torch.linalg.EncryptedLinearRegression method*), 23
_compute_pvalues() (*syft.frameworks.torch.linalg.lr.DASH method*), 20
_compute_pvalues() (*syft.frameworks.torch.linalg.lr.EncryptedLinearRegression method*), 19
_configure_tfe() (*in module syft.frameworks.keras.model.sequential*), 5
_connect_all_nodes() (*syft.grid.abstract_grid.AbstractGrid method*), 94
_consumer_handler() (*syft.workers.websocket_server.WebsocketServerWorker method*), 147
_create_attr_name_string() (*syft.generic.pointers.object_pointer.ObjectPointer method*), 78
_create_data_loader() (*syft.federated.federated_client.FederatedClient method*), 2
_create_internal_representation() (*syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor method*), 47
_create_placeholders() (*syft.messaging.plan.Plan static method*), 103
_create_placeholders() (*syft.messaging.plan.plan.Plan static method*),

99
_decompress() (*in module syft.serde.compression*), 132
_deserialize_msgpack_binary() (*in module syft.serde.msgpack.serde*), 121
_deserialize_msgpack_simple() (*in module syft.serde.msgpack.serde*), 121
_deserialize_tensor() (*in module syft.serde.msgpack.torch_serde*), 122
_deserialize_tensor() (*in module syft.serde.protobuf.torch_serde*), 128
_detail() (*in module syft.serde.msgpack.serde*), 122
_detail_collection_list() (*in module syft.serde.msgpack.native_serde*), 115
_detail_collection_set() (*in module syft.serde.msgpack.native_serde*), 115
_detail_collection_tuple() (*in module syft.serde.msgpack.native_serde*), 115
_detail_dictionary() (*in module syft.serde.msgpack.native_serde*), 116
_detail_ellipsis() (*in module syft.serde.msgpack.native_serde*), 117
_detail_ndarray() (*in module syft.serde.msgpack.native_serde*), 118
_detail_numpy_number() (*in module syft.serde.msgpack.native_serde*), 119
_detail_range() (*in module syft.serde.msgpack.native_serde*), 117
_detail_script_module() (*in module syft.serde.msgpack.torch_serde*), 124
_detail_slice() (*in module syft.serde.msgpack.native_serde*), 117
_detail_str() (*in module syft.serde.msgpack.native_serde*), 116
_detail_torch_device() (*in module syft.serde.msgpack.torch_serde*), 124
_detail_torch_mem_format() (*in module syft.serde.msgpack.torch_serde*), 124
_detail_torch_parameter() (*in module syft.serde.msgpack.torch_serde*), 123
_detail_torch_size() (*in module syft.serde.msgpack.torch_serde*), 124
_detail_torch_tensor() (*in module syft.serde.msgpack.torch_serde*), 123
_expand_item() (*syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor static method*), 47
_fetch_plan_remote() (*syft.workers.base.BaseWorker method*), 141
_fetch_protocol_remote() (*syft.workers.base.BaseWorker method*), 141
_fit() (*syft.federated.federated_client.FederatedClient method*), 2
_fix_torch_library() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 49
_force_full_bufferize() (*in module syft.serde.protobuf.serde*), 126
_force_full_simplify() (*in module syft.serde.msgpack.serde*), 120
_forward_func() (*syft.frameworks.torch.tensors.interpreters.large_pre static method*), 48
_forward_json_to_websocket_server_worker() (*syft.workers.node_client.NodeClient method*), 143
_forward_to_websocket_server_worker() (*syft.workers.node_client.NodeClient method*), 143
_forward_to_websocket_server_worker() (*syft.workers.websocket_client.WebsocketClientWorker method*), 146
_generate_bufferizers_and_unbufferizers() (*in module syft.serde.protobuf.serde*), 126
_generate_simplifiers_and_detailers() (*in module syft.serde.msgpack.serde*), 121
_get_batch() (*syft.frameworks.torch.fl.dataloader._DataLoaderIter method*), 9
_get_batch() (*syft.frameworks.torch.fl.dataloader._DataLoaderOneWo method*), 9
_get_hooked_additive_shared_method() (*syft.frameworks.torch.hook.hook.TorchHook method*), 17
_get_hooked_base_worker_method() (*syft.frameworks.torch.hook.hook.TorchHook method*), 16
_get_hooked_func() (*syft.frameworks.torch.hook.hook.TorchHook class method*), 17
_get_hooked_func() (*syft.generic.frameworks.hook.hook.FrameworkHook class method*), 64
_get_hooked_method() (*syft.generic.frameworks.hook.hook.FrameworkHook class method*), 64
_get_hooked_multi_pointer_method() (*syft.generic.frameworks.hook.hook.FrameworkHook class method*), 64
_get_hooked_private_method() (*syft.generic.frameworks.hook.hook.FrameworkHook class method*), 64
_get_hooked_string_method() (*syft.generic.frameworks.hook.hook.FrameworkHook class method*), 65
_get_hooked_string_pointer_method()

(*syft.generic.frameworks.hook.hook.FrameworkHook class method*), 65
_get_hooked_syft_method() (*syft.generic.frameworks.hook.hook.FrameworkHook class method*), 64
_get_layer_type() (*in module syft.frameworks.keras.model.sequential*), 5
_get_maximum_precision() (*in module syft.frameworks.torch.tensors.interpreters.native*), 48
_get_msg() (*syft.workers.base.BaseWorker method*), 141
_get_node_id() (*syft.workers.node_client.NodeClient method*), 143
_get_response() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 49
_get_worker() (*syft.workers.base.BaseWorker method*), 139
_get_worker_based_on_id() (*syft.workers.base.BaseWorker method*), 139
_get_workers() (*syft.frameworks.torch.linalg.DASH static method*), 24
_get_workers() (*syft.frameworks.torch.linalg.EncryptedLinearRegression static method*), 23
_get_workers() (*syft.frameworks.torch.linalg.lr.DASH static method*), 20
_get_workers() (*syft.frameworks.torch.linalg.lr.EncryptedLinearRegression static method*), 19
_getitem_multipointer() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 34
_getitem_public() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 35
_handler() (*syft.workers.websocket_server.WebsocketServerWorker method*), 147
_hook_additive_shared_tensor_methods() (*syft.frameworks.torch.hook.hook.TorchHook method*), 16
_hook_layers() (*syft.frameworks.keras.hook.KerasHook method*), 5
_hook_module() (*syft.frameworks.torch.hook.hook.TorchHook method*), 17
_hook_multi_pointer_tensor_methods() (*syft.generic.frameworks.hook.hook.FrameworkHook method*), 63
_hook_native_methods() (*syft.generic.frameworks.hook.hook.FrameworkHook method*), 63
_hook_native_tensor() (*syft.frameworks.torch.hook.hook.TorchHook method*), 16
_hook_native_tensor() (*syft.generic.frameworks.hook.hook.FrameworkHook method*), 62
_hook_object_pointer_methods() (*syft.generic.frameworks.hook.hook.FrameworkHook method*), 63
_hook_optim() (*syft.frameworks.torch.hook.hook.TorchHook method*), 17
_hook_parameters() (*syft.frameworks.torch.hook.hook.TorchHook method*), 16
_hook_pointer_tensor_methods() (*syft.generic.frameworks.hook.hook.FrameworkHook method*), 63
_hook_private_tensor_methods() (*syft.frameworks.torch.hook.hook.TorchHook method*), 16
_hook_private_tensor_methods() (*syft.generic.frameworks.hook.hook.FrameworkHook method*), 63
_hook_properties() (*syft.generic.frameworks.hook.hook.FrameworkHook method*), 63
_hook_sequential() (*syft.frameworks.keras.hook.KerasHook method*), 5
_hook_string_methods() (*syft.generic.frameworks.hook.hook.FrameworkHook method*), 63
_hook_string_pointer_methods() (*syft.generic.frameworks.hook.hook.FrameworkHook method*), 63
_hook_syft_tensor_methods() (*syft.frameworks.torch.hook.hook.TorchHook method*), 16
_hook_syft_tensor_methods() (*syft.generic.frameworks.hook.hook.FrameworkHook method*), 63
_hook_tensor() (*syft.frameworks.torch.hook.hook.TorchHook method*), 17
_hook_torch_module() (*syft.frameworks.torch.hook.hook.TorchHook method*), 16
_hook_worker_methods() (*syft.frameworks.torch.hook.hook.TorchHook method*), 16
_host_encrypted_model() (*syft.grid.private_grid.PrivateGridNetwork method*), 95
_init_hidden() (*syft.frameworks.torch.nn.rnn.RNNBase method*), 29
_instantiate_tfe_layer() (*in module syft.frameworks.keras.model.sequential*), 5

_internal_representation_to_large_ints() (syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor method), 47
_inv_upper() (syft.frameworks.torch.linalg.DASH static method), 24
_inv_upper() (syft.frameworks.torch.linalg.lr.DASH static method), 20
_is_command_valid_guard() (syft.generic.frameworks.attributes.FrameworkAttributes method), 71
_is_parameter() (syft.frameworks.torch.tensors.interpreters.native.TorchTensor method), 49
_known_workers() (syft.messaging.plan.Plan property), 103
_known_workers() (syft.messaging.plan.plan.Plan property), 99
_ldl() (in module syft.frameworks.torch.linalg.operations), 20
_moduli_for_fields (in module syft.frameworks.torch.tensors.interpreters.crt_precision), 42
_norm_mpc() (in module syft.frameworks.torch.linalg.operations), 21
_parse_address() (syft.workers.node_client.NodeClient method), 143
_perform_function_overloading() (syft.generic.frameworks.hook.hook.FrameworkHook class method), 64
_private_div() (syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method), 36
_private_mul() (syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method), 35
_producer_handler() (syft.workers.websocket_server.WebsocketServerWorker method), 147
_public_div() (syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method), 36
_public_mul() (syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method), 35
_query_encrypted_model_hosts() (syft.grid.private_grid.PrivateGridNetwork method), 95
_query_encrypted_models() (syft.grid.public_grid.PublicGridNetwork method), 97
_query_unencrypted_models() (syft.grid.public_grid.PublicGridNetwork method), 97
_random_common_bit() (in module syft.frameworks.torch.mpc.securenn), 25
_random_common_value() (in module syft.frameworks.torch.mpc.securenn), 25
_rebuild_tfe_model() (in module syft.frameworks.keras.model.sequential), 5
_recv_msg() (syft.workers.base.BaseWorker method), 136
_recv_msg() (syft.workers.virtual.VirtualWorker method), 145
_recv_msg() (syft.workers.websocket_client.WebsocketClientWorker method), 146
_recv_msg() (syft.workers.websocket_server.WebsocketServerWorker method), 147
_remote_dot_products() (syft.frameworks.torch.linalg.DASH static method), 24
_remote_dot_products() (syft.frameworks.torch.linalg.EncryptedLinearRegression static method), 23
_remote_dot_products() (syft.frameworks.torch.linalg.lr.DASH static method), 20
_remote_dot_products() (syft.frameworks.torch.linalg.lr.EncryptedLinearRegression static method), 19
_remote_qr() (syft.frameworks.torch.linalg.DASH static method), 24
_remote_qr() (syft.frameworks.torch.linalg.lr.DASH static method), 20
_requires_large_precision() (syft.frameworks.torch.tensors.interpreters.native.TorchTensor method), 51
_resolve_shares() (syft.messaging.protocol.Protocol method), 114
_restore_large_number() (syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor static method), 47
_return_bool_result() (syft.workers.node_client.NodeClient method), 143
_run_encrypted_computation() (syft.grid.private_grid.PrivateGridNetwork method), 96
_run_encrypted_inference() (syft.grid.public_grid.PublicGridNetwork method), 97
_run_unencrypted_inference() (syft.grid.private_grid.PrivateGridNetwork method), 95
_run_unencrypted_inference() (syft.grid.public_grid.PublicGridNetwork method), 97
_send_msg() (syft.workers.base.BaseWorker method), 136
_send_msg() (syft.workers.virtual.VirtualWorker method), 145

_send_msg() (*syft.workers.websocket_client.WebsocketClientWorker method*), 146
_send_msg_and_deserialize() (*syft.workers.websocket_client.WebsocketClientWorker method*), 146
_serialize_msgpack_binary() (*in module syft.serde.msgpack.serde*), 121
_serialize_msgpack_simple() (*in module syft.serde.msgpack.serde*), 121
_serialize_tensor() (*in module syft.serde.msgpack.torch_serde*), 122
_serialize_tensor() (*in module syft.serde.protobuf.torch_serde*), 128
_serve_encrypted_model() (*syft.grid.public_grid.PublicGridNetwork method*), 97
_serve_unencrypted_model() (*syft.grid.public_grid.PublicGridNetwork method*), 97
_share_ptrs() (*syft.frameworks.torch.linalg.DASH method*), 24
_share_ptrs() (*syft.frameworks.torch.linalg.EncryptedLinearRegression method*), 23
_share_ptrs() (*syft.frameworks.torch.linalg.lr.DASH method*), 20
_share_ptrs() (*syft.frameworks.torch.linalg.lr.EncryptedLinearRegression method*), 19
_shares_of_zero() (*in module syft.frameworks.torch.mpc.securenn*), 26
_simplify() (*in module syft.serde.msgpack.serde*), 121
_simplify() (*syft.messaging.message.Message method*), 105
_simplify_collection() (*in module syft.serde.msgpack.native_serde*), 115
_simplify_dictionary() (*in module syft.serde.msgpack.native_serde*), 116
_simplify_ellipsis() (*in module syft.serde.msgpack.native_serde*), 117
_simplify_ndarray() (*in module syft.serde.msgpack.native_serde*), 118
_simplify_numpy_number() (*in module syft.serde.msgpack.native_serde*), 118
_simplify_range() (*in module syft.serde.msgpack.native_serde*), 116
_simplify_script_module() (*in module syft.serde.msgpack.torch_serde*), 124
_simplify_slice() (*in module syft.serde.msgpack.native_serde*), 117
_simplify_str() (*in module syft.serde.msgpack.native_serde*), 116
_simplify_torch_device() (*in module syft.serde.msgpack.torch_serde*), 123
_simplify_torch_mem_format() (*in module syft.serde.msgpack.torch_serde*), 124
_simplify_torch_parameter() (*in module syft.serde.msgpack.torch_serde*), 123
_simplify_torch_size() (*in module syft.serde.msgpack.torch_serde*), 124
_simplify_torch_tensor() (*in module syft.serde.msgpack.torch_serde*), 123
_sizes_for_fields() (*in module syft.frameworks.torch.tensors.interpreters.crt_precision*), 42
_split_numbers() (*syft.frameworks.torch.tensors.interpreters.large_p static method*), 47
_string_input_args_adaptor() (*syft.generic.frameworks.hook.hook.FrameworkHook class method*), 65
_swap_axis() (*syft.frameworks.torch.nn.rnn.RNNBase method*), 29
_transfer_methods_to_framework_class() (*syft.generic.frameworks.hook.hook.FrameworkHook class method*), 62
_transfer_methods_to_native_tensor() (*syft.frameworks.torch.hook.hook.TorchHook class method*), 17
_trim_params() (*in module syft.frameworks.keras.model.sequential*), 5
_unbufferize() (*in module syft.serde.protobuf.serde*), 127
_unbufferize_arg() (*syft.messaging.message.Operation static method*), 108
_unbufferize_args() (*syft.messaging.message.Operation static method*), 108
_unbufferize_none() (*in module syft.serde.protobuf.native_serde*), 125
_unbufferize_script_function() (*in module syft.serde.protobuf.torch_serde*), 129
_unbufferize_script_module() (*in module syft.serde.protobuf.torch_serde*), 129
_unbufferize_torch_device() (*in module syft.serde.protobuf.torch_serde*), 129
_unbufferize_torch_parameter() (*in module syft.serde.protobuf.torch_serde*), 129
_unbufferize_torch_size() (*in module syft.serde.protobuf.torch_serde*), 130
_unbufferize_torch_tensor() (*in module syft.serde.protobuf.torch_serde*), 129
_unbufferize_traced_module() (*in module syft.serde.protobuf.torch_serde*), 129
_update_node_reference() (*syft.workers.node_client.NodeClient method*), 143
_which_methods_should_we_auto_overload()

(*syft.generic.frameworks.hook.hook.FrameworkHook method*), 63
_wrap_and_send_obj() (*syft.federated.train_config.TrainConfig method*), 3
_wrap_str_return_value() (*syft.generic.frameworks.hook.hook.FrameworkHook class method*), 65

A

AbstractCredential (*class in syft.grid.authentication.credential*), 94
AbstractGrid (*class in syft.grid.abstract_grid*), 94
AbstractObject (*class in syft.generic.object*), 87
AbstractTensor (*class in syft.generic.tensor*), 92
AbstractWorker (*class in syft.serde*), 134
AbstractWorker (*class in syft.workers.abstract*), 135
AccountCredential (*class in syft.grid.authentication.account*), 93
accuracy() (*in module syft.frameworks.torch.fl.utils*), 12
active (*syft.generic.frameworks.hook.trace.Trace attribute*), 70
add() (*syft.frameworks.torch.tensors.decorators.logging.LoggingTensor method*), 32
add() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 35
add() (*syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionTensor method*), 42
add() (*syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor method*), 47
add() (*syft.frameworks.torch.tensors.interpreters.paillier.PaillierTensor method*), 53
add() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 57
add_ (*syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor attribute*), 46
add_() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 57
add_constructor_registration() (*in module syft.frameworks.keras.layers*), 4
add_constructor_registration() (*in module syft.frameworks.keras.layers.constructor*), 4
add_dataset() (*syft.federated.federated_client.FederatedClient method*), 1
add_model() (*in module syft.frameworks.torch.fl.utils*), 11
add_placeholder() (*syft.messaging.plan.Plan method*), 103
add_placeholder() (*syft.messaging.plan.plan.Plan method*), 99
add_string() (*syft.generic.string.String method*), 91
add_worker() (*syft.workers.base.BaseWorker method*), 139
add_workers() (*syft.workers.base.BaseWorker method*), 140
AddBackward (*class in syft.frameworks.torch.tensors.interpreters.gradients*), 43
AdditiveSharingTensor (*class in syft.frameworks.torch.tensors.interpreters.additive_shared*), 33
ALIAS (*syft.frameworks.torch.torch_attributes.TorchAttributes attribute*), 61
ALIAS() (*syft.generic.frameworks.attributes.FrameworkAttributes property*), 71
ALL (*syft.codes.TENSOR_SERIALIZATION attribute*), 148
allow() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 50
allow() (*syft.frameworks.torch.tensors.interpreters.private.PrivateTensor method*), 60
ambiguous_functions (*in module syft.frameworks.torch.hook.hook_args*), 17
ambiguous_functions (*in module syft.generic.frameworks.hook.hook_args*), 65
ambiguous_methods (*in module syft.frameworks.torch.hook.hook_args*), 17
ambiguous_methods (*in module syft.generic.frameworks.hook.hook_args*), 65
apply_dim_transformations() (*in module syft.frameworks.torch.tensors.interpreters.gradients_core*), 44
apply_lz4_compression() (*in module syft.serde.compression*), 132
apply_no_compression() (*in module syft.serde.compression*), 132
apply_zstd_compression() (*in module syft.serde.compression*), 132
arrmap() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 37
AsinBackward (*class in syft.frameworks.torch.tensors.interpreters.gradients*), 43
async_fit() (*syft.workers.websocket_client.WebsocketClientWorker method*), 146
attr() (*syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor method*), 39
attr() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 50
attr() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 57
attr() (*syft.generic.pointers.object_pointer.ObjectPointer method*), 78
attr() (*syft.generic.pointers.pointer_tensor.PointerTensor method*), 84

AUTHENTICATE (*syft.codes.REQUEST_MSG attribute*), 148
AutogradTensor (*class in syft.frameworks.torch.tensors.interpreters.autograd*), 38
AvgPool2d (*class in syft.frameworks.torch.nn*), 31
AvgPool2d (*class in syft.frameworks.torch.nn.pool*), 28

B

backward() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 34
backward() (*syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor method*), 39
backward() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 57
backward_func (*in module syft.frameworks.torch.hook.hook_args*), 17
backward_func (*in module syft.generic.frameworks.hook.hook_args*), 65
backwards_grad() (*in module syft.frameworks.torch.tensors.interpreters.autograd*), 38
base_types (*in module syft.generic.frameworks.hook.hook_args*), 65
BaseDataset (*class in syft.frameworks.torch.fl*), 13
BaseDataset (*class in syft.frameworks.torch.fl.dataset*), 10
BaseWorker (*class in syft.workers.base*), 135
bufferize() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor static method*), 38
bufferize() (*syft.frameworks.torch.tensors.interpreters.placeholder.PlaceHolder static method*), 55
bufferize() (*syft.generic.pointers.pointer_tensor.PointerTensor static method*), 85
bufferize() (*syft.messaging.message.ObjectMessage static method*), 108
bufferize() (*syft.messaging.message.Operation static method*), 107
bufferize() (*syft.messaging.plan.Plan static method*), 105
bufferize() (*syft.messaging.plan.plan.Plan static method*), 101
bufferize() (*syft.messaging.plan.state.State static method*), 102
bufferize() (*syft.messaging.protocol.Protocol static method*), 113
build() (*syft.messaging.plan.Plan method*), 103
build() (*syft.messaging.plan.plan.Plan method*), 99
build_get_tensor_type() (*in module syft.generic.frameworks.hook.hook_args*), 68
build_register_response() (*in module syft.generic.frameworks.hook.hook_args*), 70
build_register_response_function() (*in module syft.generic.frameworks.hook.hook_args*), 69
build_rule() (*in module syft.generic.frameworks.hook.hook_args*), 67
build_unwrap_args_from_function() (*in module syft.generic.frameworks.hook.hook_args*), 66
build_unwrap_args_with_rules() (*in module syft.generic.frameworks.hook.hook_args*), 67
build_wrap_reponse_from_function() (*in module syft.generic.frameworks.hook.hook_args*), 67
build_wrap_response_with_rules() (*in module syft.generic.frameworks.hook.hook_args*), 68

C

CallablePointer (*class in syft.generic.pointers.callable_pointer*), 73
CannotRequestObjectAttribute, 150
chunk() (*syft.frameworks.torch.tensors.interpreters.additive_shared.Addi method*), 36
clear() (*syft.generic.frameworks.hook.trace.Trace method*), 70
clear_objects() (*syft.generic.object_storage.ObjectStorage method*), 90
clear_objects() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 38
clear_objects_remote() (*syft.workers.websocket_client.WebsocketClientWorker method*), 146
clone() (*syft.frameworks.torch.tensors.interpreters.additive_shared.Addi method*), 34
clone() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 50
clone() (*syft.generic.pointers.pointer_tensor.PointerTensor method*), 83
clone() (*syft.generic.tensor.AbstractTensor method*), 92
close() (*syft.workers.websocket_client.WebsocketClientWorker method*), 146
code() (*syft.serde.msgpack.proto.TypeInfo property*), 119
coef (*syft.frameworks.torch.linalg.DASH attribute*), 24
coef (*syft.frameworks.torch.linalg.EncryptedLinearRegression attribute*), 23
coef (*syft.frameworks.torch.linalg.lr.DASH attribute*), 19
coef (*syft.frameworks.torch.linalg.lr.EncryptedLinearRegression attribute*), 18

combine() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 51
combine_pointers() (*in module syft.frameworks.torch.functions*), 61
CompressionNotFoundException, 150
compute_q_noisy_max() (*in module syft.frameworks.torch.dp.pate*), 6
compute_q_noisy_max_approx() (*in module syft.frameworks.torch.dp.pate*), 6
compute_q_noisy_max_torch() (*in module syft.frameworks.torch.dp.pate*), 8
connect() (*syft.workers.websocket_client.WebsocketClientWorker method*), 146
CONNECT_NODE (*syft.codes.REQUEST_MSG attribute*), 148
connect_nodes() (*syft.workers.node_client.NodeClient method*), 143
connect_to_model() (*syft.workers.tfe.TFEWorker method*), 145
construct_grad_fn_class() (*in module syft.frameworks.torch.tensors.interpreters.build_gradients*), 40
contents() (*syft.messaging.message.Message property*), 105
contents() (*syft.messaging.message.Operation property*), 106
contents() (*syft.messaging.message.PlanCommandMessage property*), 111
Conv2d (*class in syft.frameworks.torch.nn*), 31
Conv2d (*class in syft.frameworks.torch.nn.conv*), 28
copy() (*syft.frameworks.torch.tensors.interpreters.placeholder.PlaceHolder method*), 55
copy() (*syft.generic.tensor.AbstractTensor method*), 92
copy() (*syft.messaging.plan.Plan method*), 104
copy() (*syft.messaging.plan.plan.Plan method*), 100
copy() (*syft.messaging.plan.state.State method*), 101
create_callable_pointer() (*in module syft.generic.pointers.callable_pointer*), 73
create_from_attributes() (*syft.messaging.protocol.Protocol static method*), 113
create_gaussian_mixture_toy_data() (*in module syft.frameworks.torch.fl.utils*), 12
create_grad_if_missing() (*syft.messaging.plan.state.State static method*), 101
create_message_execute_command() (*syft.workers.base.BaseWorker static method*), 142
create_numpy_tensor() (*in module syft.frameworks.torch.tensors.interpreters.numpy*), 52
create_pointer() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 50
create_pointer() (*syft.generic.pointers.object_pointer.ObjectPointer static method*), 76
create_pointer() (*syft.generic.pointers.object_wrapper.ObjectWrapp static method*), 79
create_pointer() (*syft.generic.pointers.pointer_tensor.PointerTensor static method*), 83
create_pointer() (*syft.generic.string.String static method*), 91
create_pointer() (*syft.messaging.plan.Plan method*), 104
create_pointer() (*syft.messaging.plan.plan.Plan method*), 100
create_pointer() (*syft.messaging.protocol.Protocol method*), 114
create_random_id() (*in module syft.generic.id_provider*), 86
create_sandbox() (*in module syft.sandbox*), 151
create_shape() (*syft.frameworks.torch.hook.hook.TorchHook method*), 16
create_shape() (*syft.generic.frameworks.hook.hook.FrameworkHook class method*), 62
create_tensor_from_numpy() (*syft.frameworks.torch.tensors.interpreters.large_precision.Large static method*), 47
create_wrapper() (*syft.frameworks.torch.hook.hook.TorchHook method*), 16
create_wrapper() (*syft.generic.frameworks.hook.hook.FrameworkHo class method*), 62
create_zeros() (*syft.frameworks.torch.hook.hook.TorchHook method*), 16
create_zeros() (*syft.generic.frameworks.hook.hook.FrameworkHook class method*), 62
CREDENTIAL_FIELD (*syft.grid.authentication.account.AccountCredentia attribute*), 93
CRTPrecisionTensor (*class in syft.frameworks.torch.tensors.interpreters.crt_precision*), 41
current_objects() (*syft.generic.object_storage.ObjectStorage method*), 90

D

DASH (*class in syft.frameworks.torch.linalg*), 23
DASH (*class in syft.frameworks.torch.linalg.lr*), 19
data() (*syft.frameworks.torch.tensors.interpreters.autograd.AutogradTen property*), 39
data() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor property*), 49
data() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisi property*), 57
data() (*syft.generic.pointers.pointer_tensor.PointerTensor property*), 83
dataset_federate() (*in module syft.frameworks.torch.fl.dataset*), 11

de_register_obj() (syft.generic.object_storage.ObjectStorage method), 89
de_register_obj() (syft.workers.base.BaseWorker method), 138
decompose() (in module syft.frameworks.torch.mpc.securenn), 25
decrypt() (syft.frameworks.torch.tensors.interpreters.native.TorchTensor method), 52
decrypt() (syft.frameworks.torch.tensors.interpreters.paillier.PaillierTensor method), 53
default_backward_func() (in module syft.generic.frameworks.hook.hook_args), 66
default_collate() (in module syft.frameworks.torch.fl.dataloader), 9
default_pytorch_maximum_precision() (in module syft.frameworks.torch.tensors.interpreters.native), 48
default_register_tensor() (in module syft.generic.frameworks.hook.hook_args), 66
DELETE_MODEL (syft.codes.REQUEST_MSG attribute), 148
delete_model() (syft.workers.node_client.NodeClient method), 144
DependencyError, 149
deploy() (syft.messaging.protocol.Protocol method), 112
derivatives (in module syft.frameworks.torch.tensors.interpreters.build_gradients), 40
describe() (syft.generic.object.AbstractObject method), 87
description() (syft.frameworks.torch.tensors.interpreters.native.TorchTensor property), 48
deserialize() (in module syft.serde), 134
deserialize() (in module syft.serde.msgpack), 125
deserialize() (in module syft.serde.msgpack.serde), 121
deserialize() (in module syft.serde.protobuf), 130
deserialize() (in module syft.serde.protobuf.serde), 127
deserialize() (in module syft.serde.serde), 133
detail() (syft.exceptions.GetNotPermittedError static method), 150
detail() (syft.exceptions.ResponseSignatureError static method), 150
detail() (syft.exceptions.SendNotPermittedError static method), 150
detail() (syft.federated.train_config.TrainConfig static method), 3
detail() (syft.frameworks.torch.tensors.decorators.logging.LoggingTensor static method), 33
detail() (syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor static method), 38
detail() (syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor static method), 40
detail() (syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionInfo static method), 42
detail() (syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc static method), 44
detail() (syft.frameworks.torch.tensors.interpreters.paillier.PaillierTensor static method), 54
detail() (syft.frameworks.torch.tensors.interpreters.placeholder.PlaceHolder static method), 55
detail() (syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor static method), 59
detail() (syft.frameworks.torch.tensors.interpreters.private.PrivateTensor static method), 61
detail() (syft.generic.pointers.multi_pointer.MultiPointerTensor static method), 75
detail() (syft.generic.pointers.object_pointer.ObjectPointer static method), 78
detail() (syft.generic.pointers.object_wrapper.ObjectWrapper static method), 79
detail() (syft.generic.pointers.pointer_plan.PointerPlan static method), 80
detail() (syft.generic.pointers.pointer_protocol.PointerProtocol static method), 81
detail() (syft.generic.pointers.pointer_tensor.PointerTensor static method), 85
detail() (syft.generic.string.String static method), 91
detail() (syft.messaging.message.ForceObjectDeleteMessage static method), 110
detail() (syft.messaging.message.GetShapeMessage static method), 109
detail() (syft.messaging.message.IsNoneMessage static method), 109
detail() (syft.messaging.message.Message static method), 106
detail() (syft.messaging.message.ObjectMessage static method), 108
detail() (syft.messaging.message.ObjectRequestMessage static method), 108
detail() (syft.messaging.message.Operation static method), 107
detail() (syft.messaging.message.PlanCommandMessage static method), 111
detail() (syft.messaging.message.SearchMessage static method), 110
detail() (syft.messaging.plan.Plan static method), 105
detail() (syft.messaging.plan.plan.Plan static method), 101
detail() (syft.messaging.plan.state.State static method), 102
detail() (syft.messaging.protocol.Protocol static

method), 113
detail() (*syft.workers.base.BaseWorker static method*), 142
dim() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 34
dim() (*syft.generic.pointers.multi_pointer.MultiPointerTensor method*), 74
dim() (*syft.generic.pointers.pointer_tensor.PointerTensor method*), 84
disable_gc() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor property*), 49
dispatch() (*syft.generic.pointers.multi_pointer.MultiPointerTensor static method*), 75
div() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 36
div() (*syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionTensor method*), 42
div() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 57
DivBackward (*class in syft.frameworks.torch.tensors.interpreters.gradients*), 43
division() (*in module syft.frameworks.torch.mpc.securenn*), 26

E

eight_fold() (*in module syft.generic.frameworks.hook.hook_args*), 69
enabled() (*syft.generic.frameworks.hook.trace.Trace method*), 70
encrypt() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 52
encrypt() (*syft.frameworks.torch.tensors.interpreters.paillier.PaillierTensor method*), 53
encrypt_() (*syft.frameworks.torch.tensors.interpreters.paillier.PaillierTensor method*), 53
EncryptedLinearRegression (*class in syft.frameworks.torch.linalg*), 22
EncryptedLinearRegression (*class in syft.frameworks.torch.linalg.lr*), 18
eq() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 37
eq() (*syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor method*), 39
eq() (*syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionTensor method*), 42
eq() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 59
ERROR (*syft.codes.RESPONSE_MSG attribute*), 149
evaluate() (*syft.federated.federated_client.FederatedClient method*), 2
evaluate() (*syft.workers.websocket_client.WebsocketClientWorker method*), 146
EXCEPTION_PROTOBUF_TRANSLATORS (*in module syft.serde.protobuf.serde*), 126
EXCEPTION_SIMPLIFIER_AND_DETAILERS (*in module syft.serde.msgpack.serde*), 120
execute_command() (*syft.workers.base.BaseWorker method*), 137
execute_plan_command() (*syft.workers.base.BaseWorker method*), 137
exp() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 58
extract_batches_per_worker() (*in module syft.frameworks.torch.fl.utils*), 11

F

federate() (*in module syft.frameworks.torch.fl.dataset*), 11
federated_avg() (*in module syft.frameworks.torch.fl.utils*), 12
FederatedClient (*class in syft.federated.federated_client*), 1
FederatedDataLoader (*class in syft.frameworks.torch.fl*), 14
FederatedDataLoader (*class in syft.frameworks.torch.fl.dataloader*), 9
FederatedDataset (*class in syft.frameworks.torch.fl*), 13
FederatedDataset (*class in syft.frameworks.torch.fl.dataset*), 11
FETCH_PLAN (*syft.codes.PLAN_CMDS attribute*), 148
fetch_plan() (*syft.workers.base.BaseWorker method*), 141
FETCH_PROTOCOL (*syft.codes.PLAN_CMDS attribute*), 148
fetch_protocol() (*syft.workers.base.BaseWorker method*), 141
filter_layers() (*in module syft.frameworks.keras.layers*), 4
filter_layers() (*in module syft.frameworks.keras.layers.constructor*), 4
find() (*syft.generic.pointers.object_pointer.ObjectPointer class method*), 77
find_location() (*syft.messaging.protocol.Protocol static method*), 112
find_placeholders() (*syft.messaging.plan.Plan method*), 103
find_placeholders() (*syft.messaging.plan.plan.Plan method*), 99
fit() (*syft.federated.federated_client.FederatedClient method*), 2
fit() (*syft.frameworks.torch.linalg.DASH method*), 24

fit() (*syft.frameworks.torch.linalg.EncryptedLinearRegression method*), 23
fit() (*syft.frameworks.torch.linalg.lr.DASH method*), 20
fit() (*syft.frameworks.torch.linalg.lr.EncryptedLinearRegression method*), 18
fit() (*syft.workers.websocket_client.WebsocketClientWorker method*), 146
five_fold() (*in module syft.generic.frameworks.hook.hook_args*), 69
fix_large_precision() (*syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor method*), 47
fix_prec() (*syft.frameworks.torch.fl.BaseDataset method*), 13
fix_prec() (*syft.frameworks.torch.fl.dataset.BaseDataset method*), 11
fix_prec() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 51
fix_prec() (*syft.generic.pointers.pointer_tensor.PointerTensor method*), 84
fix_prec_() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 51
fix_precision (*syft.frameworks.torch.fl.BaseDataset attribute*), 13
fix_precision (*syft.frameworks.torch.fl.dataset.BaseDataset attribute*), 10
fix_precision (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor attribute*), 48
fix_precision (*syft.generic.pointers.pointer_tensor.PointerTensor attribute*), 82
fix_precision() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 57
fix_precision_ (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor attribute*), 48
fix_precision_() (*syft.messaging.plan.Plan method*), 104
fix_precision_() (*syft.messaging.plan.plan.Plan method*), 100
fix_precision_() (*syft.messaging.plan.state.State method*), 101
FixedPrecisionTensor (*class in syft.frameworks.torch.tensors.interpreters.precision*), 56
flip() (*in module syft.frameworks.torch.mpc.securenn*), 25
float_prec() (*syft.frameworks.torch.fl.BaseDataset method*), 13
float_prec() (*syft.frameworks.torch.fl.dataset.BaseDataset method*), 11
float_prec() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 50
float_prec() (*syft.generic.pointers.pointer_tensor.PointerTensor method*), 84
float_prec_() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 50
float_precision (*syft.frameworks.torch.fl.BaseDataset attribute*), 13
float_precision (*syft.frameworks.torch.fl.dataset.BaseDataset attribute*), 10
float_precision (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor attribute*), 48
float_precision (*syft.generic.pointers.pointer_tensor.PointerTensor attribute*), 82
float_precision() (*syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor method*), 39
float_precision() (*syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionTensor method*), 41
float_precision() (*syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor method*), 47
float_precision() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 57
float_precision() (*syft.frameworks.torch.tensors.interpreters.private.PrivateTensor method*), 60
float_precision_ (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor attribute*), 48
float_precision_() (*syft.messaging.plan.Plan method*), 104
float_precision_() (*syft.messaging.plan.plan.Plan method*), 100
float_precision_() (*syft.messaging.plan.state.State method*), 101
force_detail() (*syft.workers.base.BaseWorker static method*), 142
force_rm_obj() (*syft.generic.object_storage.ObjectStorage method*), 90
force_simplify() (*syft.workers.base.BaseWorker static method*), 142
forced_code() (*syft.serde.msgpack.proto.TypeInfo property*), 119
ForceObjectDeleteMessage (*class in syft.messaging.message*), 110
forward() (*syft.frameworks.torch.nn.AvgPool2d method*), 31
forward() (*syft.frameworks.torch.nn.conv.Conv2d method*), 28
forward() (*syft.frameworks.torch.nn.Conv2d method*), 31
forward() (*syft.frameworks.torch.nn.GRUCell method*), 30

forward() (*syft.frameworks.torch.nn.LSTMCell method*), 31
forward() (*syft.frameworks.torch.nn.pool.AvgPool2d method*), 28
forward() (*syft.frameworks.torch.nn.rnn.GRUCell method*), 29
forward() (*syft.frameworks.torch.nn.rnn.LSTMCell method*), 29
forward() (*syft.frameworks.torch.nn.rnn.RNNBase method*), 29
forward() (*syft.frameworks.torch.nn.rnn.RNNCell method*), 29
forward() (*syft.frameworks.torch.nn.RNNCell method*), 30
forward_func (*in module syft.frameworks.torch.hook.hook_args*), 17
forward_func (*in module syft.generic.frameworks.hook.hook_args*), 65
four_fold() (*in module syft.generic.frameworks.hook.hook_args*), 69
four_layers() (*in module syft.generic.frameworks.hook.hook_args*), 68
framework_layer_module (*in module syft.generic.frameworks.types*), 72
framework_layer_modules (*in module syft.generic.frameworks.types*), 72
framework_shapes (*in module syft.generic.frameworks.types*), 72
framework_tensors (*in module syft.generic.frameworks.types*), 72
FrameworkAttributes (*class in syft.generic.frameworks.attributes*), 71
FrameworkHook (*class in syft.generic.frameworks.hook.hook*), 62
FrameworkLayerModule (*in module syft.generic.frameworks.types*), 72
FrameworkLayerModuleType (*in module syft.generic.frameworks.types*), 72
frameworks (*syft.generic.frameworks.remote.Remote attribute*), 72
FrameworkShape (*in module syft.generic.frameworks.types*), 72
FrameworkShapeType (*in module syft.generic.frameworks.types*), 72
FrameworkTensor (*in module syft.generic.frameworks.types*), 72
FrameworkTensorType (*in module syft.generic.frameworks.types*), 72
fullname() (*in module syft.serde.msgpack.proto*), 119
func2plan (*class in syft.messaging.plan*), 102
func2plan (*class in syft.messaging.plan.plan*), 98

G

garbage_collection() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor property*), 49
GATEWAY_ENDPOINTS (*class in syft.codes*), 148
gc() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor property*), 49
ge() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveS method*), 37
generate_shares() (*syft.frameworks.torch.tensors.interpreters.additive_shared.Addit static method*), 34
get (*syft.messaging.plan.Plan attribute*), 103
get (*syft.messaging.plan.plan.Plan attribute*), 99
get() (*syft.federated.train_config.TrainConfig method*), 3
get() (*syft.frameworks.torch.fl.BaseDataset method*), 13
get() (*syft.frameworks.torch.fl.dataset.BaseDataset method*), 11
get() (*syft.frameworks.torch.tensors.interpreters.additive_shared.Additive method*), 34
get() (*syft.frameworks.torch.tensors.interpreters.autograd.AutogradTenso method*), 39
get() (*syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecis method*), 42
get() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 50
get() (*syft.generic.object.AbstractObject method*), 88
get() (*syft.generic.pointers.multi_pointer.MultiPointerTensor method*), 74
get() (*syft.generic.pointers.object_pointer.ObjectPointer method*), 77
get() (*syft.generic.pointers.pointer_plan.PointerPlan method*), 80
get() (*syft.generic.pointers.pointer_protocol.PointerProtocol method*), 81
get() (*syft.generic.pointers.pointer_tensor.PointerTensor method*), 84
get_() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 50
get_() (*syft.messaging.plan.Plan method*), 104
get_() (*syft.messaging.plan.plan.Plan method*), 100
get_() (*syft.messaging.plan.state.State method*), 101
get_attributes() (*syft.exceptions.ResponseSignatureError method*), 150
get_child (*in module syft.generic.frameworks.hook.hook_args*), 65
get_class_attributes() (*syft.frameworks.torch.tensors.interpreters.additive_shared.Addit method*), 34
get_class_attributes() (*syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPre*

method), 42
get_class_attributes() (*syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor method*), 47
get_class_attributes() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 57
get_class_attributes() (*syft.frameworks.torch.tensors.interpreters.private.PrivateTensor method*), 60
get_class_attributes() (*syft.generic.object.AbstractObject method*), 88
get_class_attributes() (*syft.generic.pointers.pointer_tensor.PointerTensor method*), 83
get_class_attributes() (*syft.generic.string.String method*), 91
get_coeff() (*syft.frameworks.torch.linalg.DASH method*), 24
get_coeff() (*syft.frameworks.torch.linalg.lr.DASH method*), 20
get_element_at (*in module syft.generic.frameworks.hook.hook_args*), 68
get_garbage_collect_data() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 37
GET_ID (*syft.codes.REQUEST_MSG attribute*), 148
get_loss_fn() (*syft.federated.train_config.TrainConfig method*), 3
get_model() (*syft.federated.train_config.TrainConfig method*), 3
get_native_framework_name() (*syft.generic.frameworks.attributes.FrameworkAttributes class method*), 71
get_obj() (*syft.generic.object_storage.ObjectStorage method*), 89
get_obj() (*syft.workers.base.BaseWorker method*), 138
get_p_values() (*syft.frameworks.torch.linalg.DASH method*), 24
get_p_values() (*syft.frameworks.torch.linalg.lr.DASH method*), 20
get_packets() (*syft.generic.metrics.NetworkMonitor static method*), 87
get_pointers() (*syft.messaging.plan.Plan method*), 104
get_pointers() (*syft.messaging.plan.plan.Plan method*), 100
get_protobuf_id() (*in module syft.serde.protobuf.proto*), 125
get_recorded_ids() (*syft.generic.id_provider.IdProvider method*), 86
get_shape() (*syft.generic.pointers.pointer_tensor.PointerTensor method*), 82
get_shape() (*syft.frameworks.torch.linalg.DASH method*), 24
get_shape() (*syft.frameworks.torch.linalg.lr.DASH method*), 20
get_shape() (*syft.workers.base.BaseWorker static method*), 141
get_tensor_type_functions (*in module syft.generic.frameworks.hook.hook_args*), 65
get_worker() (*syft.workers.base.BaseWorker method*), 139
GetNotPermittedError, 150
GetShapeMessage (*class in syft.messaging.message*), 109
grad() (*syft.frameworks.torch.tensors.interpreters.additive_shared.Additi property*), 34
grad() (*syft.frameworks.torch.tensors.interpreters.autograd.AutogradTens property*), 39
grad() (*syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrec property*), 41
grad() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor property*), 49
grad() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisi property*), 57
grad() (*syft.generic.pointers.multi_pointer.MultiPointerTensor property*), 74
grad() (*syft.generic.pointers.pointer_tensor.PointerTensor property*), 83
grad() (*syft.generic.tensor.AbstractTensor property*), 92
GradFunc (*class in syft.frameworks.torch.tensors.interpreters.gradients_c* 44
gradient() (*syft.frameworks.torch.tensors.interpreters.gradients.AddBa method*), 43
gradient() (*syft.frameworks.torch.tensors.interpreters.gradients.AsinBa method*), 43
gradient() (*syft.frameworks.torch.tensors.interpreters.gradients.DivBac method*), 43
gradient() (*syft.frameworks.torch.tensors.interpreters.gradients.Matmu method*), 43
gradient() (*syft.frameworks.torch.tensors.interpreters.gradients.MulBa method*), 43
gradient() (*syft.frameworks.torch.tensors.interpreters.gradients.PowBa method*), 43
gradient() (*syft.frameworks.torch.tensors.interpreters.gradients.ReluBa method*), 44
gradient() (*syft.frameworks.torch.tensors.interpreters.gradients.Sigmoi method*), 43
gradient() (*syft.frameworks.torch.tensors.interpreters.gradients.SinBac

method), 43
gradient() (*syft.frameworks.torch.tensors.interpreters.gradients.SubBackward method*), 44
gradient() (*syft.frameworks.torch.tensors.interpreters.gradients.SubBackward method*), 44
gradient() (*syft.frameworks.torch.tensors.interpreters.gradients.SumBackward method*), 43
gradient() (*syft.frameworks.torch.tensors.interpreters.gradients.SumBackward method*), 43
gradient() (*syft.frameworks.torch.tensors.interpreters.gradients.TanhBackward method*), 44
gradient() (*syft.frameworks.torch.tensors.interpreters.gradients.TBackward method*), 43
gradient() (*syft.frameworks.torch.tensors.interpreters.gradients_core.GradFunc method*), 44
GRU (*class in syft.frameworks.torch.nn*), 30
GRU (*class in syft.frameworks.torch.nn.rnn*), 30
GRUCell (*class in syft.frameworks.torch.nn*), 30
GRUCell (*class in syft.frameworks.torch.nn.rnn*), 29
gt() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 36
gt() (*syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor method*), 47

H

handle_func_command() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor class method*), 37
handle_func_command() (*syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor class method*), 39
handle_func_command() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor class method*), 49
handle_func_command() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor class method*), 59
handle_func_command() (*syft.generic.object.AbstractObject class method*), 88
handle_func_command() (*syft.generic.pointers.multi_pointer.MultiPointerTensor class method*), 75
handle_func_command() (*syft.generic.pointers.object_pointer.ObjectPointer class method*), 77
has_child() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 48
hook() (*in module syft.sandbox*), 151
hook_method_args_functions (*in module syft.generic.frameworks.hook.hook_args*), 65
hook_method_response_functions (*in module syft.generic.frameworks.hook.hook_args*), 65
hook_response() (*in module syft.generic.frameworks.hook.hook_args*), 67
HookedTensor (*class in syft.frameworks.torch.tensors.interpreters.hook*), 45
HOST_USER_BLACKLIST (*syft.codes.REQUEST_MSG attribute*), 148

I

id (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor property*), 49
id_at_location() (*syft.generic.pointers.pointer_plan.PointerPlan property*), 80
IdenticalQueueError, 151
IdProvider (*class in syft.generic.id_provider*), 86
INFERENCE_RESULT (*syft.codes.RESPONSE_MSG attribute*), 149
inherited_bufferizers_found (*in module syft.serde.protobuf.serde*), 126
inherited_simplifiers_found (*in module syft.serde.msgpack.serde*), 121
init_hidden() (*syft.frameworks.torch.nn.rnn.RNNCellBase method*), 29
init_shares() (*syft.frameworks.torch.tensors.interpreters.additive_sha method*), 34
initialize_object() (*in module syft.generic.object*), 89
initialize_tensor() (*in module syft.generic.tensor*), 92
input_tensor() (*syft.frameworks.torch.tensors.interpreters.placeholder. method*), 54
intercept (*syft.frameworks.torch.linalg.EncryptedLinearRegression attribute*), 23
intercept (*syft.frameworks.torch.linalg.lr.EncryptedLinearRegression attribute*), 18
inverse_permutation() (*syft.frameworks.torch.tensors.interpreters.large_precision.Large property*), 47
inv_sym() (*in module syft.frameworks.torch.linalg*), 22
inv_sym() (*in module syft.frameworks.torch.linalg.operations*), 20
InvalidProtocolFileError, 151
InvalidTensorForRemoteGet, 149
inverse() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPre method*), 58
iris_data_partial() (*in module syft.frameworks.torch.fl.utils*), 12
is_inplace_method() (*syft.frameworks.torch.torch_attributes.TorchAttributes method*), 61
is_inplace_method() (*syft.generic.frameworks.attributes.FrameworkAttributes*

method), 71
is_none() (*syft.generic.pointers.pointer_tensor.PointerTensor method*), 83
is_tensor_none() (*syft.workers.base.BaseWorker static method*), 140
is_wrapper (*syft.generic.object.AbstractObject attribute*), 87
IsNoneMessage (*class in syft.messaging.message*), 109
item() (*syft.generic.pointers.pointer_tensor.PointerTensor method*), 85

J

json() (*syft.grid.authentication.account.AccountCredential method*), 93
json() (*syft.grid.authentication.credential.AbstractCredential method*), 94

K

keep() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 52
keep() (*syft.generic.pointers.pointer_tensor.PointerTensor method*), 84
KerasHook (*class in syft.frameworks.keras.hook*), 5
keygen (*in module syft.frameworks.torch.he.paillier*), 14

L

LargePrecisionTensor (*class in syft.frameworks.torch.tensors.interpreters.large_precision*), 46
le() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 37
LIST_MODELS (*syft.codes.REQUEST_MSG attribute*), 148
list_objects() (*syft.workers.websocket_server.WebsocketServerWorker method*), 148
list_objects_remote() (*syft.workers.websocket_client.WebsocketClientWorker method*), 146
load_data() (*syft.workers.base.BaseWorker method*), 136
location() (*syft.frameworks.torch.fl.BaseDataset property*), 13
location() (*syft.frameworks.torch.fl.dataset.BaseDataset property*), 11
location() (*syft.generic.pointers.pointer_plan.PointerPlan property*), 80
location() (*syft.messaging.plan.Plan property*), 103
location() (*syft.messaging.plan.plan.Plan property*), 99
locations() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor property*), 33
log() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecision method*), 58
logger (*in module syft.dependency_check*), 149
logger (*in module syft.frameworks.torch.fl.dataset*), 10
logger (*in module syft.frameworks.torch.fl.utils*), 11
logger (*in module syft.workers.base*), 135
logger (*in module syft.workers.tfe*), 144
logger (*in module syft.workers.websocket_client*), 145
LoggingTensor (*class in syft.frameworks.torch.tensors.decorators.logging*), 32
logmgf_exact() (*in module syft.frameworks.torch.dp.pate*), 6
logmgf_exact_torch() (*in module syft.frameworks.torch.dp.pate*), 8
logmgf_from_counts() (*in module syft.frameworks.torch.dp.pate*), 6
logmgf_from_counts_torch() (*in module syft.frameworks.torch.dp.pate*), 8
logs (*syft.generic.frameworks.hook.trace.Trace attribute*), 70
LSTM (*class in syft.frameworks.torch.nn*), 30
LSTM (*class in syft.frameworks.torch.nn.rnn*), 30
LSTMCell (*class in syft.frameworks.torch.nn*), 30
LSTMCell (*class in syft.frameworks.torch.nn.rnn*), 29
lt() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 37
lt() (*syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor method*), 47
LZ4 (*in module syft.serde.compression*), 132

M

manual_add() (*syft.frameworks.torch.tensors.decorators.logging.LoggingTensor method*), 32
many_fold() (*in module syft.generic.frameworks.hook.hook_args*), 69
MAP_NATIVE_PROTOBUF_TRANSLATORS (*in module syft.serde.protobuf.native_serde*), 125
MAP_NATIVE_SIMPLIFIERS_AND_DETAILERS (*in module syft.serde.msgpack.native_serde*), 119
MAP_PYTHON_TO_PROTOBUF_CLASSES (*in module syft.serde.protobuf.proto*), 125
MAP_TF_SIMPLIFIERS_AND_DETAILERS (*in module syft.serde.msgpack.serde*), 120
MAP_TO_PROTOBUF_TRANSLATORS (*in module syft.serde.protobuf.serde*), 126
MAP_TO_SIMPLIFIERS_AND_DETAILERS (*in module syft.serde.msgpack.serde*), 120
MAP_TORCH_PROTOBUF_TRANSLATORS (*in module syft.serde.protobuf.serde*), 126
MAP_TORCH_PROTOBUF_TRANSLATORS (*in module syft.serde.protobuf.torch_serde*), 130

MAP_TORCH_SIMPLIFIERS_AND_DETAILERS (*in module syft.serde.msgpack.serde*), 120
MAP_TORCH_SIMPLIFIERS_AND_DETAILERS (*in module syft.serde.msgpack.torch_serde*), 124
matmul() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 36
matmul() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 58
MatmulBackward (*class in syft.frameworks.torch.tensors.interpreters.gradients*), 43
max() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 37
maxpool() (*in module syft.frameworks.torch.mpc.securenn*), 27
maxpool2d() (*in module syft.frameworks.torch.mpc.securenn*), 27
maxpool_deriv() (*in module syft.frameworks.torch.mpc.securenn*), 27
Message (*class in syft.messaging.message*), 105
method2plan() (*in module syft.messaging.plan*), 102
method2plan() (*in module syft.messaging.plan.plan*), 98
methods_to_hook (*syft.generic.string.String attribute*), 90
mid_get() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 50
mid_get() (*syft.generic.object.AbstractObject method*), 88
mm (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor attribute*), 56
mm() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 36
mm() (*syft.frameworks.torch.tensors.interpreters.numpy.NumpyTensor method*), 52
mm() (*syft.frameworks.torch.tensors.interpreters.paillier.PaillierTensor method*), 53
mod() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 36
mod() (*syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor method*), 47
MODELS (*syft.codes.RESPONSE_MSG attribute*), 149
models() (*syft.workers.node_client.NodeClient property*), 143
Module (*class in syft.generic.frameworks.overload*), 72
move() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 50
move() (*syft.generic.pointers.pointer_tensor.PointerTensor method*), 84
msb() (*in module syft.frameworks.torch.mpc.securenn*), 26
mul() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 36
mul() (*syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionTensor method*), 42
mul() (*syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor method*), 47
mul() (*syft.frameworks.torch.tensors.interpreters.paillier.PaillierTensor method*), 53
mul() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 57
mul_ (*syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor attribute*), 46
mul_ (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor attribute*), 56
mul_() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 57
MulBackward (*class in syft.frameworks.torch.tensors.interpreters.gradients*), 43
MultiPointerTensor (*class in syft.generic.pointers.multi_pointer*), 74

N

NetworkMonitor (*class in syft.generic.metrics*), 87
NO_COMPRESSION (*in module syft.serde.compression*), 132
no_wrap (*in module syft.frameworks.torch.mpc.securenn*), 25
no_wrap (*in module syft.frameworks.torch.mpc.spdz*), 27
no_wrap (*in module syft.frameworks.torch.tensors.interpreters.additive_shared*), 33
NODE_ID (*syft.codes.RESPONSE_MSG attribute*), 149
NodeClient (*class in syft.workers.node_client*), 143
NUMPY (*syft.codes.TENSOR_SERIALIZATION attribute*), 148
numpy_tensor() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 52
numpy_tensor_deserializer() (*in module syft.serde.torch.serde*), 131
numpy_tensor_serializer() (*in module syft.serde.torch.serde*), 131
numpy_type_map (*in module syft.frameworks.torch.fl.dataloader*), 9
NumpyTensor (*class in syft.frameworks.torch.tensors.interpreters.numpy*), 52

O

obj() (*syft.generic.pointers.object_wrapper.ObjectWrapper property*), 79
OBJ_FORCE_FULL_PROTOBUF_TRANSLATORS (*in module syft.serde.protobuf.serde*), 126
OBJ_FORCE_FULL_SIMPLIFIER_AND_DETAILERS (*in module syft.serde.msgpack.serde*), 120

OBJ_PROTOBUF_TRANSLATORS (*in module syft.serde.protobuf.serde*), 126
OBJ_SIMPLIFIER_AND_DETAILERS (*in module syft.serde.msgpack.serde*), 120
ObjectMessage (*class in syft.messaging.message*), 108
ObjectNotFoundError, 151
ObjectPointer (*class in syft.generic.pointers.object_pointer*), 76
ObjectRequestMessage (*class in syft.messaging.message*), 108
objects_count() (*syft.workers.websocket_server.WebsocketServerWorker method*), 148
objects_count_remote() (*syft.workers.websocket_client.WebsocketClientWorker method*), 146
ObjectStorage (*class in syft.generic.object_storage*), 89
ObjectWrapper (*class in syft.generic.pointers.object_wrapper*), 79
on() (*syft.generic.string.String method*), 91
on() (*syft.generic.tensor.AbstractTensor method*), 92
on_function_call() (*syft.frameworks.torch.tensors.decorators.logging.LoggingTensor class method*), 32
on_function_call() (*syft.generic.object.AbstractObject class method*), 88
one (*in module syft.generic.frameworks.hook.hook_args*), 65
one_fold() (*in module syft.generic.frameworks.hook.hook_args*), 69
one_layer() (*in module syft.generic.frameworks.hook.hook_args*), 68
Operation (*class in syft.messaging.message*), 106
out_of_operation (*syft.generic.frameworks.hook.trace.Trace attribute*), 70
overload_function() (*syft.generic.frameworks.overload.Overloaded static method*), 72
overload_method() (*syft.generic.frameworks.overload.Overloaded static method*), 72
overload_module() (*syft.generic.frameworks.overload.Overloaded static method*), 72
Overloaded (*class in syft.generic.frameworks.overload*), 72
overloaded (*in module syft.generic.frameworks.overload*), 72

P

p (*in module syft.frameworks.torch.mpc.securenn*), 25
PaillierTensor (*class in syft.frameworks.torch.tensors.interpreters.paillier*), 53
parameters() (*syft.generic.pointers.pointer_plan.PointerPlan method*), 80
parameters() (*syft.messaging.plan.Plan method*), 103
parameters() (*syft.messaging.plan.plan.Plan method*), 99
parse() (*syft.grid.authentication.account.AccountCredential static method*), 93
parse() (*syft.grid.authentication.credential.AbstractCredential method*), 94
PASSWORD_FIELD (*syft.grid.authentication.account.AccountCredential attribute*), 93
perform_analysis() (*in module syft.frameworks.torch.dp.pate*), 7
perform_analysis_torch() (*in module syft.frameworks.torch.dp.pate*), 8
PlaceHolder (*class in syft.frameworks.torch.tensors.interpreters.placeholder*), 54
Plan (*class in syft.messaging.plan*), 102
Plan (*class in syft.messaging.plan.plan*), 98
PLAN_CMDS (*class in syft.codes*), 148
PlanCommandMessage (*class in syft.messaging.message*), 111
PlanCommandUnknownError, 151
PointerPlan (*class in syft.generic.pointers.pointer_plan*), 80
PointerProtocol (*class in syft.generic.pointers.pointer_protocol*), 81
PointerTensor (*class in syft.generic.pointers.pointer_tensor*), 82
pop() (*syft.generic.id_provider.IdProvider method*), 86
positive() (*syft.frameworks.torch.tensors.interpreters.additive_shared.A method*), 36
pow() (*syft.frameworks.torch.tensors.interpreters.additive_shared.Additive method*), 36
pow() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecision method*), 57
PowBackward (*class in syft.frameworks.torch.tensors.interpreters.gradients*), 43
predict() (*syft.frameworks.torch.linalg.EncryptedLinearRegression method*), 23
predict() (*syft.frameworks.torch.linalg.lr.EncryptedLinearRegression method*), 19
private_compare() (*in module syft.frameworks.torch.mpc.securenn*), 26
private_tensor() (*syft.frameworks.torch.tensors.interpreters.native.T method*), 51

PrivateGridNetwork (*class in syft.grid.private_grid*), 94
PrivateTensor (*class in syft.frameworks.torch.tensors.interpreters.private*), 60
proto_type_info() (*in module syft.serde.msgpack*), 124
proto_type_info() (*in module syft.serde.msgpack.proto*), 120
protobuf_tensor_deserializer() (*in module syft.serde.protobuf.torch_serde*), 128
protobuf_tensor_serializer() (*in module syft.serde.protobuf.torch_serde*), 128
Protocol (*class in syft.messaging.protocol*), 111
pstf_spec (*in module syft.dependency_check*), 149
PublicGridNetwork (*class in syft.grid.public_grid*), 96
PureFrameworkTensorFoundError, 149
pvalue (*syft.frameworks.torch.linalg.DASH attribute*), 24
pvalue (*syft.frameworks.torch.linalg.lr.DASH attribute*), 20
pvalue_coef (*syft.frameworks.torch.linalg.EncryptedLinearRegression attribute*), 23
pvalue_coef (*syft.frameworks.torch.linalg.lr.EncryptedLinearRegression attribute*), 18
pvalue_intercept (*syft.frameworks.torch.linalg.EncryptedLinearRegression attribute*), 23
pvalue_intercept (*syft.frameworks.torch.linalg.lr.EncryptedLinearRegression attribute*), 18

Q

Q_BITS (*in module syft.frameworks.torch.mpc.securenn*), 25
qr() (*in module syft.frameworks.torch.linalg*), 22
qr() (*in module syft.frameworks.torch.linalg.operations*), 21
query_model() (*syft.workers.tfe.TFEWorker method*), 145
query_model_async() (*syft.workers.tfe.TFEWorker method*), 145
query_model_hosts() (*syft.grid.abstract_grid.AbstractGrid method*), 94
query_model_hosts() (*syft.grid.private_grid.PrivateGridNetwork method*), 95
query_model_hosts() (*syft.grid.public_grid.PublicGridNetwork method*), 96
query_model_join() (*syft.workers.tfe.TFEWorker method*), 145

R

read() (*syft.messaging.plan.state.State method*), 101
read_packet() (*syft.generic.metrics.NetworkMonitor static method*), 87
readable_plan() (*syft.messaging.plan.Plan property*), 103
readable_plan() (*syft.messaging.plan.plan.Plan property*), 99
reconstruct() (*syft.frameworks.torch.tensors.interpreters.additive_sha method*), 34
reconstruct() (*syft.frameworks.torch.tensors.interpreters.crt_precision method*), 42
recv_msg() (*syft.workers.base.BaseWorker method*), 137
refresh() (*syft.frameworks.torch.tensors.interpreters.additive_shared.A method*), 34
refresh() (*syft.generic.tensor.AbstractTensor method*), 92
register_ambiguous_function() (*in module syft.generic.frameworks.hook.hook_args*), 66
register_ambiguous_method() (*in module syft.generic.frameworks.hook.hook_args*), 66
register_backward_func() (*in module syft.generic.frameworks.hook.hook_args*), 66
register_credentials() (*syft.frameworks.torch.tensors.interpreters.private.PrivateTensor method*), 60
register_forward_func() (*in module syft.generic.frameworks.hook.hook_args*), 65
register_framework() (*syft.generic.frameworks.remote.Remote static method*), 72
register_obj() (*syft.generic.object_storage.ObjectStorage method*), 89
register_obj() (*syft.workers.base.BaseWorker method*), 138
register_response() (*in module syft.generic.frameworks.hook.hook_args*), 69
register_response_functions (*in module syft.generic.frameworks.hook.hook_args*), 69
register_tensor() (*in module syft.generic.frameworks.hook.hook_args*), 70
register_type_rule() (*in module syft.generic.frameworks.hook.hook_args*), 65
registration_enabled() (*syft.workers.base.BaseWorker method*), 136
relu() (*in module syft.frameworks.torch.mpc.securenn*), 26

relu() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 36
relu() (*syft.frameworks.torch.tensors.interpreters.autograd.AutogradTensor method*), 39
relu_deriv() (*in module syft.frameworks.torch.mpc.securenn*), 26
ReluBackward (*class in syft.frameworks.torch.tensors.interpreters.gradients*), 44
Remote (*class in syft.generic.frameworks.remote*), 72
remote_get() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 50
remote_get() (*syft.generic.pointers.pointer_tensor.PointerTensor method*), 84
remote_send() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 50
remote_send() (*syft.generic.pointers.pointer_tensor.PointerTensor method*), 84
RemoteObjectFoundError, 149
remove_dataset() (*syft.federated.federated_client.FederatedClient method*), 1
remove_worker_from_local_worker_registry() (*syft.workers.base.BaseWorker method*), 136
remove_worker_from_registry() (*syft.workers.base.BaseWorker method*), 136
replace_with_placeholders() (*syft.messaging.plan.Plan method*), 103
replace_with_placeholders() (*syft.messaging.plan.plan.Plan method*), 99
request_is_remote_tensor_none() (*syft.workers.base.BaseWorker method*), 140
REQUEST_MSG (*class in syft.codes*), 148
request_obj() (*syft.messaging.plan.Plan method*), 103
request_obj() (*syft.messaging.plan.plan.Plan method*), 99
request_obj() (*syft.workers.base.BaseWorker method*), 138
request_remote_run() (*syft.generic.pointers.pointer_protocol.PointerProtocol method*), 81
request_remote_run() (*syft.messaging.protocol.Protocol method*), 112
request_remote_tensor_shape() (*syft.workers.base.BaseWorker method*), 141
request_run_plan() (*syft.generic.pointers.pointer_plan.PointerPlan method*), 80
request_search() (*syft.workers.base.BaseWorker method*), 141
request_triple() (*in module syft.frameworks.torch.mpc.beaver*), 25
reset_parameters() (*syft.frameworks.torch.nn.LSTMCell method*), 31
reset_parameters() (*syft.frameworks.torch.nn.rnn.LSTMCell method*), 29
reset_parameters() (*syft.frameworks.torch.nn.rnn.RNNCellBase method*), 29
respond_to_obj_req() (*syft.messaging.plan.Plan method*), 103
respond_to_obj_req() (*syft.messaging.plan.plan.Plan method*), 99
respond_to_obj_req() (*syft.workers.base.BaseWorker method*), 138
RESPONSE_MSG (*class in syft.codes*), 149
ResponseSignatureError, 150
rgetattr() (*syft.generic.object.AbstractObject class method*), 88
rm_obj() (*syft.generic.object_storage.ObjectStorage method*), 90
RNN (*class in syft.frameworks.torch.nn*), 30
RNN (*class in syft.frameworks.torch.nn.rnn*), 30
RNNBase (*class in syft.frameworks.torch.nn.rnn*), 29
RNNCell (*class in syft.frameworks.torch.nn*), 30
RNNCell (*class in syft.frameworks.torch.nn.rnn*), 29
RNNCellBase (*class in syft.frameworks.torch.nn.rnn*), 29
route_method_exception() (*in module syft.exceptions*), 151
run() (*syft.generic.pointers.pointer_protocol.PointerProtocol method*), 81
run() (*syft.messaging.plan.Plan method*), 104
run() (*syft.messaging.plan.plan.Plan method*), 100
run() (*syft.messaging.protocol.Protocol method*), 112
RUN_INFERENCE (*syft.codes.REQUEST_MSG attribute*), 148
run_remote_inference() (*syft.grid.abstract_grid.AbstractGrid method*), 94
run_remote_inference() (*syft.grid.private_grid.PrivateGridNetwork method*), 95
run_remote_inference() (*syft.grid.public_grid.PublicGridNetwork method*), 96
run_remote_inference() (*syft.workers.node_client.NodeClient method*), 144

S

scale_model() (*in module syft.frameworks.torch.fl.utils*), 12
scheme_to_bytes (*in module syft.serde.compression*), 132
search() (*syft.grid.abstract_grid.AbstractGrid method*), 94
search() (*syft.grid.private_grid.PrivateGridNetwork method*), 94
search() (*syft.grid.public_grid.PublicGridNetwork method*), 96
search() (*syft.workers.base.BaseWorker method*), 141
search() (*syft.workers.websocket_client.WebsocketClientWorker method*), 146
SEARCH_ENCRYPTED_MODEL (*syft.codes.GATEWAY_ENDPOINTS attribute*), 148
SEARCH_MODEL (*syft.codes.GATEWAY_ENDPOINTS attribute*), 148
SEARCH_TAGS (*syft.codes.GATEWAY_ENDPOINTS attribute*), 148
SearchMessage (*class in syft.messaging.message*), 110
SELECT_ENCRYPTED_MODEL_HOSTS (*syft.codes.GATEWAY_ENDPOINTS attribute*), 148
SELECT_MODEL_HOST (*syft.codes.GATEWAY_ENDPOINTS attribute*), 148
select_share() (*in module syft.frameworks.torch.mpc.securenn*), 26
select_worker() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor static method*), 37
send() (*syft.federated.train_config.TrainConfig method*), 3
send() (*syft.frameworks.torch.fl.BaseDataset method*), 13
send() (*syft.frameworks.torch.fl.dataset.BaseDataset method*), 11
send() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 49
send() (*syft.generic.string.String method*), 90
send() (*syft.messaging.plan.Plan method*), 104
send() (*syft.messaging.plan.plan.Plan method*), 100
send() (*syft.messaging.protocol.Protocol method*), 112
send() (*syft.workers.base.BaseWorker method*), 137
send_() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 50
send_command() (*syft.workers.base.BaseWorker method*), 138
send_msg() (*syft.messaging.plan.Plan method*), 103
send_msg() (*syft.messaging.plan.plan.Plan method*), 99
send_msg() (*syft.workers.base.BaseWorker method*), 136
send_obj() (*syft.workers.base.BaseWorker method*), 138
SendNotPermittedError, 150
sens_at_k() (*in module syft.frameworks.torch.dp.pate*), 7
sens_at_k_torch() (*in module syft.frameworks.torch.dp.pate*), 8
ser() (*syft.generic.object.AbstractObject method*), 88
serialize() (*in module syft.serde*), 134
serialize() (*in module syft.serde.msgpack*), 124
serialize() (*in module syft.serde.msgpack.serde*), 121
serialize() (*in module syft.serde.protobuf*), 130
serialize() (*in module syft.serde.protobuf.serde*), 126
serialize() (*in module syft.serde.serde*), 133
serialize() (*syft.generic.object.AbstractObject method*), 88
serializer() (*syft.workers.base.BaseWorker property*), 142
SERIALIZERS_PROTOBUF_TO_SYFT (*in module syft.serde.protobuf.torch_serde*), 128
SERIALIZERS_SYFT_TO_PROTOBUF (*in module syft.serde.protobuf.torch_serde*), 128
serve() (*in module syft.frameworks.keras.model*), 5
serve() (*in module syft.frameworks.keras.model.sequential*), 4
serve_model() (*syft.grid.abstract_grid.AbstractGrid method*), 94
serve_model() (*syft.grid.private_grid.PrivateGridNetwork method*), 94
serve_model() (*syft.grid.public_grid.PublicGridNetwork method*), 96
serve_model() (*syft.workers.node_client.NodeClient method*), 143
set_garbage_collect_data() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 37
set_garbage_collect_data() (*syft.generic.pointers.multi_pointer.MultiPointerTensor method*), 75
set_garbage_collect_data() (*syft.generic.pointers.pointer_tensor.PointerTensor method*), 85
set_next_ids() (*syft.generic.id_provider.IdProvider method*), 86
set_obj() (*syft.federated.federated_client.FederatedClient method*), 1
set_obj() (*syft.generic.object_storage.ObjectStorage method*), 90
set_protobuf_id() (*in module syft.serde.protobuf.proto*), 125

setattr() (*syft.generic.pointers.object_pointer.ObjectPointer method*), 78
seven_fold() (*in module syft.generic.frameworks.hook.hook_args*), 69
shape() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor property*), 34
shape() (*syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionTensor property*), 42
shape() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor property*), 48
shape() (*syft.generic.pointers.multi_pointer.MultiPointerTensor property*), 74
shape() (*syft.generic.pointers.pointer_tensor.PointerTensor property*), 82
shape() (*syft.generic.tensor.AbstractTensor property*), 92
share (*syft.messaging.plan.Plan attribute*), 103
share (*syft.messaging.plan.plan.Plan attribute*), 99
share() (*in module syft.frameworks.keras.model*), 5
share() (*in module syft.frameworks.keras.model.sequential*), 4
share() (*syft.frameworks.torch.fl.BaseDataset method*), 13
share() (*syft.frameworks.torch.fl.dataset.BaseDataset method*), 11
share() (*syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionTensor method*), 42
share() (*syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor method*), 48
share() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 51
share() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 59
share() (*syft.generic.pointers.pointer_tensor.PointerTensor method*), 84
share_() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 51
share_() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 59
share_() (*syft.generic.pointers.pointer_tensor.PointerTensor method*), 85
share_() (*syft.messaging.plan.Plan method*), 104
share_() (*syft.messaging.plan.plan.Plan method*), 100
share_() (*syft.messaging.plan.state.State method*), 101
share_convert() (*in module syft.frameworks.torch.mpc.securenn*), 26
sigmoid() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 58
SigmoidBackward (*class in syft.frameworks.torch.tensors.interpreters.gradients*), 43
simplified_tensor_deserializer() (*in module syft.serde.msgpack.torch_serde*), 123
simplified_tensor_serializer() (*in module syft.serde.msgpack.torch_serde*), 123
simplify() (*syft.exceptions.GetNotPermittedError static method*), 150
simplify() (*syft.exceptions.ResponseSignatureError static method*), 150
simplify() (*syft.exceptions.SendNotPermittedError static method*), 150
simplify() (*syft.federated.train_config.TrainConfig static method*), 3
simplify() (*syft.frameworks.torch.tensors.decorators.logging.LoggingT static method*), 32
simplify() (*syft.frameworks.torch.tensors.interpreters.additive_shared.A static method*), 37
simplify() (*syft.frameworks.torch.tensors.interpreters.autograd.Autogra static method*), 40
simplify() (*syft.frameworks.torch.tensors.interpreters.crt_precision.CR static method*), 42
simplify() (*syft.frameworks.torch.tensors.interpreters.gradients_core.G static method*), 44
simplify() (*syft.frameworks.torch.tensors.interpreters.paillier.PaillierTe static method*), 54
simplify() (*syft.frameworks.torch.tensors.interpreters.placeholder.Plac static method*), 55
simplify() (*syft.frameworks.torch.tensors.interpreters.precision.FixedP static method*), 59
simplify() (*syft.frameworks.torch.tensors.interpreters.private.PrivateTe static method*), 70
simplify() (*syft.generic.pointers.multi_pointer.MultiPointerTensor static method*), 75
simplify() (*syft.generic.pointers.object_pointer.ObjectPointer static method*), 78
simplify() (*syft.generic.pointers.object_wrapper.ObjectWrapper static method*), 79
simplify() (*syft.generic.pointers.pointer_plan.PointerPlan static method*), 80
simplify() (*syft.generic.pointers.pointer_protocol.PointerProtocol static method*), 81
simplify() (*syft.generic.pointers.pointer_tensor.PointerTensor static method*), 85
simplify() (*syft.generic.string.String static method*), 91
simplify() (*syft.messaging.message.Message static method*), 105
simplify() (*syft.messaging.message.Operation static method*), 107
simplify() (*syft.messaging.message.PlanCommandMessage static method*), 111
simplify() (*syft.messaging.plan.Plan static method*), 104
simplify() (*syft.messaging.plan.plan.Plan static method*), 100

simplify() (*syft.messaging.plan.state.State static method*), 102
simplify() (*syft.messaging.protocol.Protocol static method*), 113
simplify() (*syft.workers.base.BaseWorker static method*), 142
SinBackward (*class in syft.frameworks.torch.tensors.interpreters.gradients*), 43
SinhBackward (*class in syft.frameworks.torch.tensors.interpreters.gradients*), 43
six_fold() (*in module syft.generic.frameworks.hook.hook_args*), 69
smooth_sens_torch() (*in module syft.frameworks.torch.dp.pate*), 8
smoothed_sens() (*in module syft.frameworks.torch.dp.pate*), 7
spdz_mul() (*in module syft.frameworks.torch.mpc.spdz*), 27
split_signature() (*in module syft.frameworks.torch.tensors.interpreters.build_gradients*), 40
SqrtBackward (*class in syft.frameworks.torch.tensors.interpreters.gradients*), 44
start() (*syft.workers.tfe.TFECluster method*), 145
start() (*syft.workers.tfe.TFEWorker method*), 144
start() (*syft.workers.websocket_server.WebsocketServerWorker method*), 148
start_recording_ids() (*syft.generic.id_provider.IdProvider method*), 86
State (*class in syft.messaging.plan.state*), 101
stop() (*in module syft.frameworks.keras.model*), 5
stop() (*in module syft.frameworks.keras.model.sequential*), 4
stop() (*syft.frameworks.torch.fl.dataloader._DataLoaderIter method*), 9
stop() (*syft.frameworks.torch.fl.dataloader._DataLoaderOneWorkerIter method*), 9
stop() (*syft.workers.tfe.TFECluster method*), 145
stop() (*syft.workers.tfe.TFEWorker method*), 144
String (*class in syft.generic.string*), 90
StringPointer (*class in syft.generic.pointers.string_pointer*), 86
sub() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 35
sub() (*syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPrecisionTensor method*), 42
sub() (*syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor method*), 47
sub() (*syft.frameworks.torch.tensors.interpreters.paillier.PaillierTensor method*), 53
sub() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 57
sub_() (*syft.frameworks.torch.tensors.interpreters.large_precision.LargePrecisionTensor attribute*), 46
sub_() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 57
SubBackward (*class in syft.frameworks.torch.tensors.interpreters.gradients*), 43
SUCCESS (*syft.codes.RESPONSE_MSG attribute*), 149
SumBackward (*class in syft.frameworks.torch.tensors.interpreters.gradients*), 43
summarize() (*syft.frameworks.torch.linalg.EncryptedLinearRegression method*), 23
summarize() (*syft.frameworks.torch.linalg.lr.EncryptedLinearRegression method*), 19
syft (*module*), 1
syft.codes (*module*), 148
syft.dependency_check (*module*), 149
syft.exceptions (*module*), 149
syft.federated (*module*), 1
syft.federated.federated_client (*module*), 1
syft.federated.train_config (*module*), 2
syft.frameworks (*module*), 4
syft.frameworks.keras (*module*), 4
syft.frameworks.keras.hook (*module*), 5
syft.frameworks.keras.layers (*module*), 4
syft.frameworks.keras.layers.constructor (*module*), 4
syft.frameworks.keras.model (*module*), 4
syft.frameworks.keras.model.sequential (*module*), 4
syft.frameworks.tensorflow (*module*), 6
syft.frameworks.torch (*module*), 6
syft.frameworks.torch.dp (*module*), 6
syft.frameworks.torch.dp.pate (*module*), 6
syft.frameworks.torch.fl (*module*), 9
syft.frameworks.torch.fl.dataloader (*module*), 9
syft.frameworks.torch.fl.dataset (*module*), 10
syft.frameworks.torch.fl.utils (*module*), 11
syft.frameworks.torch.functions (*module*), 14
syft.frameworks.torch.he (*module*), 14
syft.frameworks.torch.he.paillier (*module*), 14
syft.frameworks.torch.hook (*module*), 15
syft.frameworks.torch.hook.hook (*module*), 15

syft.frameworks.torch.hook.hook_args (*module*), 17
syft.frameworks.torch.linalg (*module*), 18
syft.frameworks.torch.linalg.lr (*module*), 18
syft.frameworks.torch.linalg.operations (*module*), 20
syft.frameworks.torch.mpc (*module*), 25
syft.frameworks.torch.mpc.beaver (*module*), 25
syft.frameworks.torch.mpc.securenn (*module*), 25
syft.frameworks.torch.mpc.spdz (*module*), 27
syft.frameworks.torch.nn (*module*), 28
syft.frameworks.torch.nn.conv (*module*), 28
syft.frameworks.torch.nn.pool (*module*), 28
syft.frameworks.torch.nn.rnn (*module*), 29
syft.frameworks.torch.tensors (*module*), 32
syft.frameworks.torch.tensors.decorators (*module*), 32
syft.frameworks.torch.tensors.decorators.logging (*module*), 32
syft.frameworks.torch.tensors.interpreters (*module*), 33
syft.frameworks.torch.tensors.interpreters.abstract (*module*), 33
syft.frameworks.torch.tensors.interpreters.autograd (*module*), 38
syft.frameworks.torch.tensors.interpreters.build_rnn (*module*), 40
syft.frameworks.torch.tensors.interpreters.crt_precision (*module*), 41
syft.frameworks.torch.tensors.interpreters.gradients (*module*), 43
syft.frameworks.torch.tensors.interpreters.gradients_core (*module*), 44
syft.frameworks.torch.tensors.interpreters.hook (*module*), 45
syft.frameworks.torch.tensors.interpreters.large_precision (*module*), 46
syft.frameworks.torch.tensors.interpreters.native (*module*), 48
syft.frameworks.torch.tensors.interpreters.numpy (*module*), 52
syft.frameworks.torch.tensors.interpreters.paillier (*module*), 53
syft.frameworks.torch.tensors.interpreters.placeholder (*module*), 54
syft.frameworks.torch.tensors.interpreters.private (*module*), 56
syft.frameworks.torch.tensors.interpreters.private_tensor (*module*), 56
syft.frameworks.torch.tensors.interpreters.private_tensor (*module*), 60
syft.frameworks.torch.torch_attributes (*module*), 61
syft.generic (*module*), 62
syft.generic.frameworks (*module*), 62
syft.generic.frameworks.attributes (*module*), 71
syft.generic.frameworks.hook (*module*), 62
syft.generic.frameworks.hook.hook (*module*), 62
syft.generic.frameworks.hook.hook_args (*module*), 65
syft.generic.frameworks.hook.trace (*module*), 70
syft.generic.frameworks.overload (*module*), 72
syft.generic.frameworks.remote (*module*), 72
syft.generic.frameworks.types (*module*), 72
syft.generic.id_provider (*module*), 86
syft.generic.metrics (*module*), 87
syft.generic.object (*module*), 87
syft.generic.object_storage (*module*), 89
syft.generic.pointers (*module*), 73
syft.generic.pointers.callable_pointer (*module*), 73
syft.generic.pointers.multi_pointer (*module*), 74
syft.generic.pointers.object_pointer (*module*), 76
syft.generic.pointers.object_wrapper (*module*), 79
syft.generic.pointers.pointer_plan (*module*), 80
syft.generic.pointers.pointer_protocol (*module*), 81
syft.generic.pointers.pointer_tensor (*module*), 82
syft.generic.pointers.string_pointer (*module*), 86
syft.generic.string (*module*), 90
syft.generic.tensor (*module*), 92
syft.grid (*module*), 93
syft.grid.abstract_grid (*module*), 94
syft.grid.authentication (*module*), 93
syft.grid.authentication.account (*module*), 93
syft.grid.authentication.credential (*module*), 94
syft.grid.private_grid (*module*), 94
syft.grid.public_grid (*module*), 96
syft.messaging (*module*), 98

syft.messaging.message (*module*), 105
syft.messaging.plan (*module*), 98
syft.messaging.plan.plan (*module*), 98
syft.messaging.plan.state (*module*), 101
syft.messaging.protocol (*module*), 111
syft.sandbox (*module*), 151
syft.serde (*module*), 114
syft.serde.compression (*module*), 132
syft.serde.msgpack (*module*), 114
syft.serde.msgpack.native_serde (*module*), 114
syft.serde.msgpack.proto (*module*), 119
syft.serde.msgpack.serde (*module*), 120
syft.serde.msgpack.torch_serde (*module*), 122
syft.serde.protobuf (*module*), 125
syft.serde.protobuf.native_serde (*module*), 125
syft.serde.protobuf.proto (*module*), 125
syft.serde.protobuf.serde (*module*), 126
syft.serde.protobuf.torch_serde (*module*), 128
syft.serde.serde (*module*), 133
syft.serde.torch (*module*), 131
syft.serde.torch.serde (*module*), 131
syft.version (*module*), 151
syft.workers (*module*), 135
syft.workers.abstract (*module*), 135
syft.workers.base (*module*), 135
syft.workers.node_client (*module*), 143
syft.workers.tfe (*module*), 144
syft.workers.virtual (*module*), 145
syft.workers.websocket_client (*module*), 145
syft.workers.websocket_server (*module*), 147

T

t() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 57
tab (*in module syft.frameworks.torch.tensors.interpreters.build_gradients*), 40
tag() (*syft.generic.object.AbstractObject method*), 88
tag_sort() (*in module syft.messaging.plan.plan*), 101
tags() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor property*), 48
TanhBackward (*class in syft.frameworks.torch.tensors.interpreters.gradients*), 44
TBackward (*class in syft.frameworks.torch.tensors.interpreters.gradients*), 43
Tensor (*syft.frameworks.torch.torch_attributes.TorchAttributes attribute*), 61
Tensor() (*syft.generic.frameworks.attributes.FrameworkAttributes property*), 71
TENSOR_SERIALIZATION (*class in syft.codes*), 148
tensors() (*syft.messaging.plan.state.State method*), 101
tensors_to_literals() (*in module syft.frameworks.torch.dp.pate*), 7
TensorsNotCollocatedException, 150
TF (*syft.codes.TENSOR_SERIALIZATION attribute*), 148
tfe_available (*in module syft.dependency_check*), 149
tfe_spec (*in module syft.dependency_check*), 149
TFECluster (*class in syft.workers.tfe*), 145
TFEWorker (*class in syft.workers.tfe*), 144
three_fold() (*in module syft.generic.frameworks.hook.hook_args*), 69
three_layers() (*in module syft.generic.frameworks.hook.hook_args*), 68
TIMEOUT_INTERVAL (*in module syft.workers.websocket_client*), 145
TORCH (*syft.codes.TENSOR_SERIALIZATION attribute*), 148
torch() (*syft.frameworks.torch.tensors.decorators.logging.LoggingTenso static method*), 32
torch() (*syft.frameworks.torch.tensors.interpreters.additive_shared.Addi static method*), 36
torch() (*syft.frameworks.torch.tensors.interpreters.autograd.AutogradTe static method*), 39
torch() (*syft.frameworks.torch.tensors.interpreters.crt_precision.CRTPre static method*), 42
torch() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor static method*), 49
torch() (*syft.frameworks.torch.tensors.interpreters.paillier.PaillierTenso static method*), 54
torch() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecis static method*), 59
torch() (*syft.frameworks.torch.tensors.interpreters.private.PrivateTenso static method*), 60
torch_available (*in module syft.dependency_check*), 149
TORCH_DTYPE_STR (*in module syft.serde.torch.serde*), 131
TORCH_MFORMAT (*in module syft.serde.torch.serde*), 131
TORCH_MFORMAT_ID (*in module syft.serde.torch.serde*), 131
torch_spec (*in module syft.dependency_check*), 149
TORCH_STR_DTYPE (*in module syft.serde.torch.serde*), 131
torch_tensor_deserializer() (*in module syft.serde.torch.serde*), 131
torch_tensor_serializer() (*in module*

syft.serde.torch.serde), 131
torch_type() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 52
TorchAttributes (*class in syft.frameworks.torch.torch_attributes*), 61
TorchHook (*class in syft.frameworks.torch.hook.hook*), 15
TorchTensor (*class in syft.frameworks.torch.tensors.interpreters.native*), 48
Trace (*class in syft.generic.frameworks.hook.trace*), 70
tracer() (*in module syft.generic.frameworks.hook.trace*), 70
TrainConfig (*class in syft.federated.train_config*), 2
transform() (*syft.frameworks.torch.fl.BaseDataset method*), 13
transform() (*syft.frameworks.torch.fl.dataset.BaseDataset method*), 10
transpose() (*syft.frameworks.torch.tensors.interpreters.numpy.NumpyTensor method*), 52
truncate() (*syft.frameworks.torch.tensors.interpreters.precision.FixedPrecisionTensor method*), 57
two_fold() (*in module syft.generic.frameworks.hook.hook_args*), 69
two_layers() (*in module syft.generic.frameworks.hook.hook_args*), 68
TYPE_FIELD (*syft.codes.REQUEST_MSG attribute*), 148
type_precision (*in module syft.frameworks.torch.tensors.interpreters.large_precision*), 48
type_rule (*in module syft.frameworks.torch.hook.hook_args*), 17
type_rule (*in module syft.generic.frameworks.hook.hook_args*), 65
typed_identity() (*in module syft.generic.frameworks.hook.hook_args*), 69
TypeInfo (*class in syft.serde.msgpack.proto*), 119

U

unbufferize() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor static method*), 38
unbufferize() (*syft.frameworks.torch.tensors.interpreters.placeholder.PlaceHolder static method*), 55
unbufferize() (*syft.generic.pointers.pointer_tensor.PointerTensor static method*), 85
unbufferize() (*syft.messaging.message.ObjectMessage static method*), 108
unbufferize() (*syft.messaging.message.Operation static method*), 107
unbufferize() (*syft.messaging.plan.Plan static method*), 105
unbufferize() (*syft.messaging.plan.plan.Plan static method*), 101
unbufferize() (*syft.messaging.plan.state.State static method*), 102
unbufferize() (*syft.messaging.protocol.Protocol static method*), 113
UndefinedProtocolTypeError, 151
UndefinedProtocolTypePropertyError, 151
unwrap_args_from_function() (*in module syft.generic.frameworks.hook.hook_args*), 66
unwrap_args_from_method() (*in module syft.generic.frameworks.hook.hook_args*), 66
url() (*syft.workers.node_client.NodeClient property*), 143
url() (*syft.workers.websocket_client.WebsocketClientWorker property*), 146
USERNAME_FIELD (*syft.grid.authentication.account.AccountCredential attribute*), 83

V

value() (*syft.frameworks.torch.tensors.interpreters.native.TorchTensor method*), 52
value() (*syft.generic.pointers.pointer_tensor.PointerTensor method*), 84
virtual_get() (*syft.frameworks.torch.tensors.interpreters.additive_shared.AdditiveSharingTensor method*), 34
virtual_get() (*syft.generic.pointers.multi_pointer.MultiPointerTensor method*), 75
VirtualWorker (*class in syft.workers.virtual*), 145

W

WebsocketClientWorker (*class in syft.workers.websocket_client*), 145
WebsocketServerWorker (*class in syft.workers.websocket_server*), 147
WorkerNotFoundException, 150
workers() (*syft.frameworks.torch.fl.dataset.FederatedDataset property*), 11
workers() (*syft.frameworks.torch.fl.FederatedDataset property*), 13
workers() (*syft.workers.tfe.TFECluster property*), 145
wrap() (*syft.generic.pointers.object_pointer.ObjectPointer method*), 77
wrap() (*syft.generic.pointers.pointer_plan.PointerPlan method*), 80
wrap() (*syft.generic.pointers.pointer_protocol.PointerProtocol method*), 82
wrap() (*syft.generic.tensor.AbstractTensor method*), 92

Z

zero() (*syft.frameworks.torch.tensors.interpreters.additive_shared.Additi*

method), 34
zero_fold() (*in module syft.generic.frameworks.hook.hook_args*), 69
ZSTD (*in module syft.serde.compression*), 132