Federated Machine learning

Release 1.0.0

Ivan Brillo, Daniel Pipitone, Andrea Bochicchio

Oct 24, 2024

PYTHON MODULES:

1 Master module **1**
1.1 Functions . . . 1

2 Node Module **3**
2.1 Functions . . . 3

3 dataParser module **5**

4 federatedController module **7**

5 modelDefinition module **9**

6 networkModel module **11**

7 nodeController module **13**

8 UI Module **15**
8.1 Functions . . . 15

9 Node Erlang Module **17**
9.1 Functions . . . 17

10 Master Erlang Module **19**
10.1 Functions . . . 19
10.2 State Record . . . 20

11 Helper Erlang Module **21**
11.1 Functions . . . 21

Python Module Index **25**

Index **27**

CHAPTER

ONE

MASTER MODULE

This module serves as the master controller for the federated learning framework, facilitating communication between the Python-based federated learning model and the Erlang master process.

1.1 Functions

register_handler(*master_pid*)

Registers a message handler for communication with the Erlang master process. The handler processes incoming messages based on their code and executes corresponding actions. If an unrecognized message code is received, an unhandled message response is sent back.

Parameters
master_pid – The process identifier of the Erlang master process.

Returns
A confirmation string indicating successful registration of the handler.

encode_status_code(*code: str*) → Atom

Encodes a status code string into an Erlang Atom for communication with the Erlang system.

Parameters
code – The status code string.

Returns
An Erlang Atom representing the status code.

CHAPTER

TWO

NODE MODULE

This module serves as a node controller within the federated learning framework, handling communication between the Erlang master process and the local Python node.

2.1 Functions

register_handler(*master_pid*, *node_id*)

Registers a handler for Erlang messages directed to this Python process. The handler processes incoming messages and executes corresponding actions based on their code. Responses are sent back to the master process as acknowledgments or data.

Parameters

- **master_pid** – The Erlang process ID of the master process.
- **node_id** – The ID of this node as a string.

Returns

A string indicating whether the handler was registered correctly.

encode_status_code(*code: str*) → Atom

Encodes a status code string into an Erlang Atom for communication with the Erlang system.

Parameters

code – The status code string.

Returns

An Erlang Atom representing the status code.

CHAPTER

THREE

DATAPARSER MODULE

dataParser.**load_images_and_labels**(*file_path*, *image=False*, *label=False*)

dataParser.**preprocess_image**()

CHAPTER

FOUR

FEDERATEDCONTROLLER MODULE

class federatedController.**FederatedController**(*network_model:* NetworkModel)

Bases: object

static federated_weight_average(*parsed_outputs: list[dict]*) → list

Averages the weights from a list of node weights.

Args:
parsed_outputs: A list of dictionaries, where each dictionary contains the node weights and dataset size.

Returns:
A list of Numpy arrays, representing the averaged model weights.

get_definition() → str

Retrieve the network configuration as a JSON string.

Returns:
A JSON string containing the network configuration.

get_weights() → str

Retrieve the model weights as a JSON string.

Returns:
A JSON string containing the model weights.

update_weights(*node_outputs: list[list]*)

Update the model weights with the given node outputs.

The node outputs are given as a list of bytes, which are expected to be JSON strings containing dictionaries with the following keys:

- "weights": A list of Numpy arrays representing the Node weights.
- "size": The total size of the dataset for the specific Node weights.

The function will average the weights of all nodes based on their dataset size and set the averaged weights as the new model weights.

Args:
node_outputs: A list of JSONString, where each element contains the node weights and dataset size.

Returns:
None

CHAPTER

FIVE

MODELDEFINITION MODULE

CHAPTER

SIX

NETWORKMODEL MODULE

class networkModel.**NetworkModel**(**args*, ***kwargs*)

Bases: Model

call(*x*)

CHAPTER

SEVEN

NODECONTROLLER MODULE

class nodeController.**NodeController**

Bases: object

get_weights(*add_cardinality: bool = False*) → str

Returns the weights of the local model as a JSON string.

Args:
add_cardinality: If true, the dataset size of the node will be added to the JSON string.

Returns:
A JSON string containing the weights of the model and the optional dataset size.

initialize_model(*json_string: str*) → None

Initialize the local model with the provided configuration. The model is compiled with the Adam optimizer and categorical cross-entropy loss.

Args:
json_string: A JSON string containing the model configuration.

load_db() → str

Load the dataset and preprocess it.

Returns a JSON object containing the shapes of the training and test sets.

train_local() → None

Train the local model on the node's dataset for one epoch.

Returns:
A JSON object containing the training accuracy.

update_model(*json_string: str*) → None

Update the local model with the provided weights.

Args:
json_string: A JSON string containing the weights of the model.

CHAPTER

EIGHT

UI MODULE

This module manages the user interface (UI) interactions within the federated learning framework, specifically handling communication between the Erlang master process and the Python UI component.

8.1 Functions

ui.**register_handler**(*master_pid*)

Registers a handler for Erlang messages directed to this Python UI process. The handler processes incoming messages and casts the payload back to the master process.

Parameters
master_pid – The Erlang process ID of the master process.

Returns
A string indicating whether the handler was registered correctly.

ui.**encode_status_code**(*code: str*) → Atom

Encodes a status code string into an Erlang Atom for communication with the Erlang system.

Parameters
code – The status code string.

Returns
An Erlang Atom representing the status code.

CHAPTER

NINE

NODE ERLANG MODULE

9.1 Functions

node.**start_node**(*MasterPid*)

Starts the node process, initializes the Python process for handling node operations, and enters the loop to receive and process messages from the master node.

Parameters
MasterPid (*pid*) – The identifier of the master process.

Returns
none

Return type
none

node.**loop_node**(*MasterPid*, *PythonPid*)

Main loop function of the node. Handles incoming messages and interacts with the Python process based on the message type. The node can receive the following messages:

- **load_db**: Triggers the Python process to load the database and sends acknowledgment to the master.
- **initialize**: Initializes the node with a received model and sends acknowledgment back to the master process.
- **update_weights**: Updates the node's weights with the received values and sends acknowledgment to the master process.
- **train**: Starts training with the assigned dataset and sends the training result to the master process.
- **get_weights**: Requests the current weights from the node and returns them to the master process.
- **Other invalid messages**: Discards and logs an invalid message.

Parameters

- **MasterPid** (*pid*) – The identifier of the master process.
- **PythonPid** (*pid*) – The identifier of the Python process.

Returns
none

Return type
none

CHAPTER

TEN

MASTER ERLANG MODULE

10.1 Functions

master.**start_master**()

Starts the master node and initializes the Python processes for the model and the UI. This function spawns the master process, registers handlers for the Python processes, and returns the process identifier (PID) of the master.

Returns
The process identifier (PID) of the master.

Return type
pid

master.**notify_ui**(*State*, *Message*)

Sends a message to the Python UI process associated with the master. This function is used to notify the UI about the current state or other events happening in the master node.

Parameters

- **State** (*record(state)*) – The state record of the master process, containing the PID of the Python UI process.
- **Message** (*term*) – The message to send to the UI.

Returns
None.

master.**loop_master**(*State*)

Main loop function of the master node. Handles incoming messages and takes actions based on the message type. The following messages can be processed by the master node:

- *get_nodes*: Retrieves the current nodes in the cluster and notifies the UI.
- *load_db*: Distributes the command to load the database across initialized nodes.
- *initialize_nodes*: Initializes the nodes in the system and updates the state.
- *distribute_model*: Distributes the model across the initialized nodes.
- *distribute_weights*: Distributes the model weights across the distributed nodes.
- *train*: Initiates the training process on the distributed nodes and updates the weights.
- *[python_unhandled, Cause]*: Handles unhandled messages from the Python process.

- Other messages: Discards unrecognized messages and continues the loop.

Parameters
`State` (*record(state)*) – The state record of the master process, containing the PIDs of the Python processes and lists of nodes.

Returns
None.

10.2 State Record

The *state* record is used to keep track of the current state of the master process. It includes the PIDs of the Python model and UI processes, as well as lists of nodes involved in the distributed learning process.

Fields:

- **pythonModelPID**: PID of the Python process managing the model.
- **pythonUiPID**: PID of the Python process managing the UI.
- **initializedNodes**: List of PIDs of initialized nodes.
- **dbLoadedNodes**: List of PIDs of nodes that have loaded the database.
- **distributedNodes**: List of PIDs of nodes that have received the model.
- **weightsNodes**: List of PIDs of nodes that have updated weights.
- **trainNodes**: List of PIDs of nodes that have completed training.

CHAPTER

ELEVEN

HELPER ERLANG MODULE

11.1 Functions

helper.**get_cluster_nodes**()

Retrieves the list of cluster nodes, excluding the current master node.

Returns
A list of cluster nodes (excluding the master).

Return type
list

helper.**initialize_nodes**()

Initializes the active nodes in the cluster by spawning processes on the nodes and starting the *node* module.

Returns
A list of process identifiers (PIDs) of the spawned nodes.

Return type
list

helper.**distribute_object**(*PidList*, *Code*, *AckCode*, *Object*)

Sends a message with the given code and object to each PID in the list. It waits for the acknowledgment (AckCode) from all nodes and collects the responses.

Parameters

- **PidList** (*list*) – List of process identifiers (PIDs) to send the object to.
- **Code** (*term*) – The code to send with the object.
- **AckCode** (*term*) – The acknowledgment code expected from the nodes.
- **Object** (*term*) – The object to distribute to the nodes.

Returns
List of responses from the nodes.

Return type
list

helper.**wait_response**(*N*, *RespList*, *AckCode*)

Waits for responses from the nodes. It expects *N* responses with the given acknowledgment code.

Parameters

- **N** (*integer*) – Number of responses to wait for.
- **RespList** (*list*) – The list of responses collected so far.
- **AckCode** (*term*) – The acknowledgment code expected in the responses.

Returns
The complete list of responses.

Return type
list

helper.**init_python_process**()

Initializes a Python process by starting it and setting the Python environment path. Returns the process identifier (PID) of the Python process.

Returns
The process identifier (PID) of the Python process.

Return type
pid

helper.**python_register_handler**(*PythonPid*, *Module*, *MasterPid*)

Registers the Erlang master process as a handler for the Python process.

Parameters

- **PythonPid** (*pid*) – The process identifier (PID) of the Python process.
- **Module** (*atom*) – The module in Python where the handler function is located.
- **MasterPid** (*pid*) – The process identifier (PID) of the master Erlang process.

Returns
None

helper.**model_get_and_distribute**(*PidReq*, *CodeReq*, *MsgReq*, *CodeResp*, *SendCode*, *Ack*, *ListPid*)

Sends a request to the Python process to retrieve a model or weights, then distributes the received data to a list of nodes.

Parameters

- **PidReq** (*pid*) – The process identifier (PID) of the Python process making the request.
- **CodeReq** (*term*) – The code to request the model or weights.
- **MsgReq** (*term*) – The message accompanying the request.
- **CodeResp** (*term*) – The code to expect in the response.
- **SendCode** (*term*) – The code to send when distributing the received data.
- **Ack** (*term*) – The acknowledgment code expected from the nodes.

- **ListPid** (*list*) – List of process identifiers (PIDs) of the nodes to distribute the data to.

Returns
A list of responses from the nodes.

Return type
list

helper.**distribute_command**(*PidNodes*, *Request*, *ResponseCode*, *Payload*)

Distributes a command to a list of nodes and collects the responses. The payload is sent with the request.

Parameters

- **PidNodes** (*list*) – List of process identifiers (PIDs) of the nodes to send the request to.
- **Request** (*term*) – The request command to send.
- **ResponseCode** (*term*) – The response code expected in the acknowledgment.
- **Payload** (*term*) – The payload to send with the request.

Returns
A tuple containing a list of PIDs and the corresponding responses.

Return type
tuple

helper.**get_nodes_send_model**(*PidNodes*, *ModelPid*, *Request*, *ResponseCode*, *Payload*, *ModelCode*, *ModelAck*)

Retrieves model weights from the nodes, distributes the updated model, and sends it back to the Python process.

Parameters

- **PidNodes** (*list*) – List of process identifiers (PIDs) of the nodes to request the model from.
- **ModelPid** (*pid*) – The process identifier (PID) of the Python process holding the model.
- **Request** (*term*) – The request command to retrieve the model.
- **ResponseCode** (*term*) – The response code expected from the nodes.
- **Payload** (*term*) – The payload to send with the request.
- **ModelCode** (*term*) – The code to send when distributing the updated model.
- **ModelAck** (*term*) – The acknowledgment code expected from the nodes for the updated model.

Returns
A list of PIDs of the nodes that successfully received the updated model.

Return type
list

PYTHON MODULE INDEX

d

dataParser, 5

f

federatedController, 7

h

helper (*Erlang*), 21

m

modelDefinition, 9

n

networkModel, 11
nodeController, 13

u

ui (*Python*), 15

INDEX

B

built-in function
    encode_status_code(), 1
    register_handler(), 1

C

call() (*networkModel.NetworkModel method*), 11

D

dataParser
    module, 5
distribute_command() (*in module helper*), 23
distribute_object() (*in module helper*), 21

E

encode_status_code()
    built-in function, 1
encode_status_code() (*in module ui*), 15

F

federated_weight_average() (*federatedController.FederatedController static method*), 7
federatedController
    module, 7
FederatedController (*class in federatedController*), 7

G

get_cluster_nodes() (*in module helper*), 21
get_definition() (*federatedController.FederatedController method*), 7
get_nodes_send_model() (*in module helper*), 23
get_weights() (*federatedController.FederatedController method*), 7
get_weights() (*nodeController.NodeController method*), 13

H

helper
    module, 21

I

init_python_process() (*in module helper*), 22
initialize_model() (*nodeController.NodeController method*), 13
initialize_nodes() (*in module helper*), 21

L

load_db() (*nodeController.NodeController method*), 13
load_images_and_labels() (*in module dataParser*), 5
loop_master() (*in module master*), 19
loop_node() (*in module node*), 17

M

master
    module, 19
model_get_and_distribute() (*in module helper*), 22
modelDefinition
    module, 9
module
    dataParser, 5
    federatedController, 7
    helper, 21
    master, 19
    modelDefinition, 9
    networkModel, 11
    node, 17
    nodeController, 13
    ui, 15

N

networkModel
    module, 11
NetworkModel (*class in networkModel*), 11
node
    module, 17
nodeController
    module, 13
NodeController (*class in nodeController*), 13
notify_ui() (*in module master*), 19

P

preprocess_image() (*in module dataParser*), 5

python_register_handler() (*in module helper*), 22

R

register_handler()
    built-in function, 1
register_handler() (*in module ui*), 15

S

start_master() (*in module master*), 19
start_node() (*in module node*), 17

T

train_local() (*nodeController.NodeController method*), 13

U

ui
    module, 15
update_model() (*nodeController.NodeController method*), 13
update_weights() (*federatedController.FederatedController method*), 7

W

wait_response() (*in module helper*), 21