A YEAR IN APPFL SOFTWARE DEVELOPMENT

APPFL

AI4S

ZILINGHAN LI

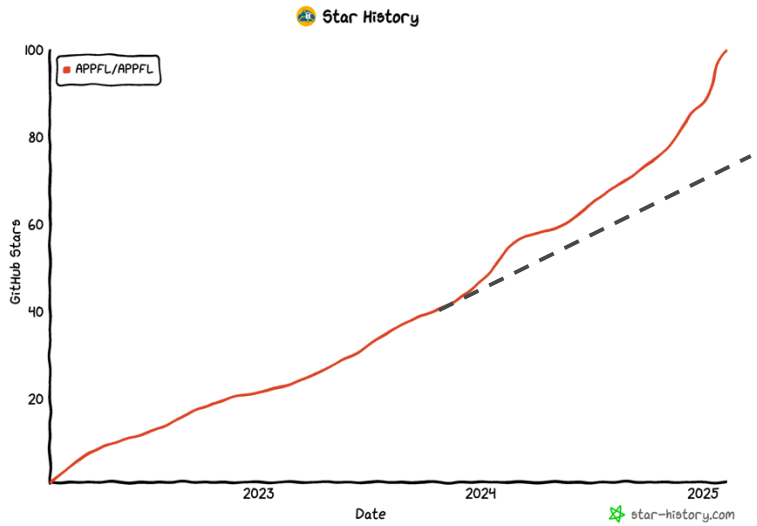
Machine Learning Engineer
Data Science and Learning Division,
Argonne National Laboratory
zilinghan.li@anl.gov

U.S. DEPARTMENT OF ENERGY Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

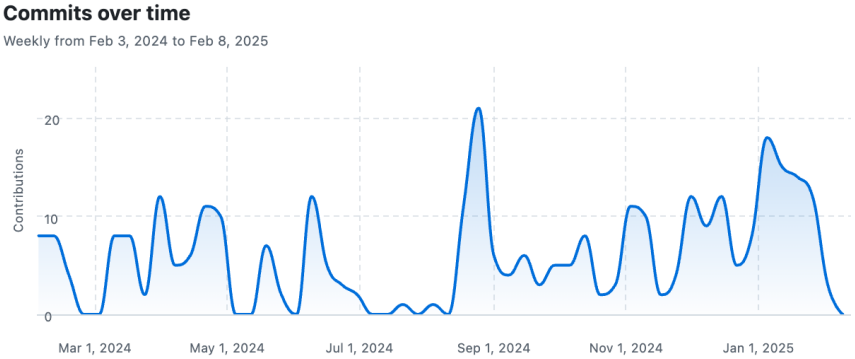
AI4S Quarterly Meeting,
Argonne National Laboratory, Lemont, IL
2025/02/10

APPETIZERS

APPFL Repo Statistics



Gained 100 stars ⭐



>300 commits & 100,000 lines of changes & 11 releases

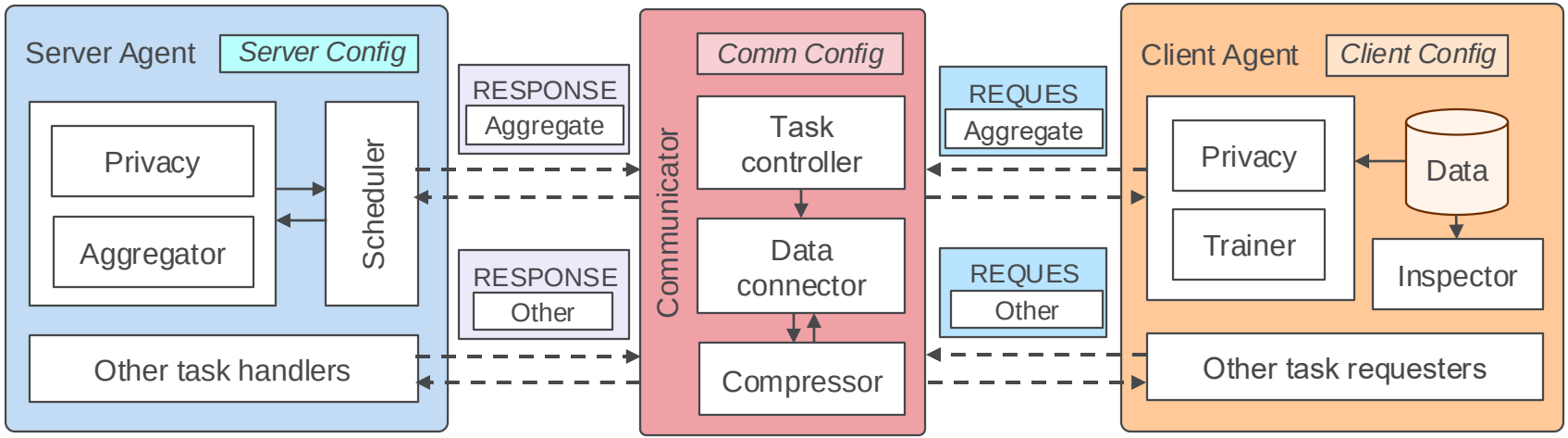
31 Issues Closed

65 Pull Requests Merged

>10 Code Contributors

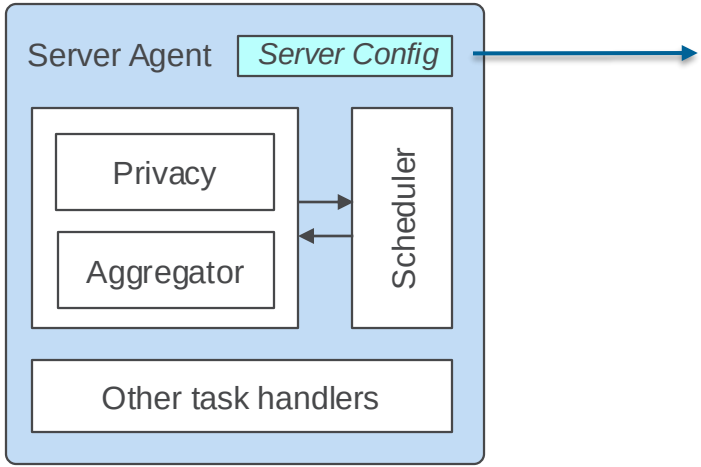
IMPORTANT MILESTONE

Release of a stable version 1.x with a new design



COMPONENT BREAKDOWN

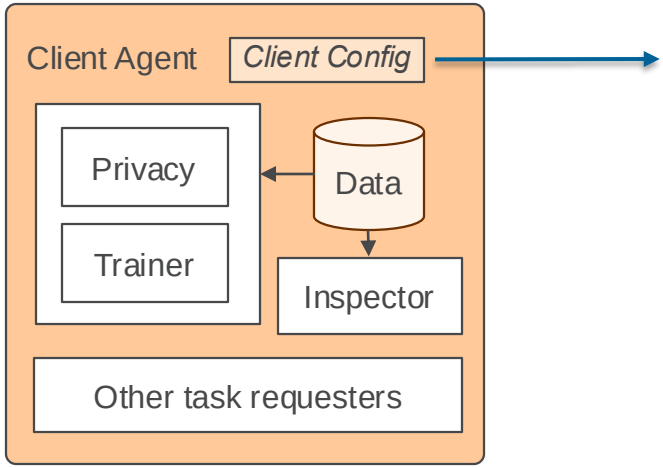
Server Agent




```yaml
server_configs:
  num_clients: 2
  scheduler: "SyncScheduler"
  scheduler_kwargs:
    same_init_model: True
  aggregator: "FedAvgAggregator"
  aggregator_kwargs:
    client_weights_mode: "equal"
  device: "cpu"
  num_global_epochs: 10
  logging_output_dirname: "./output"
  logging_output_filename: "result"
```

COMPONENT BREAKDOWN

Client Agent

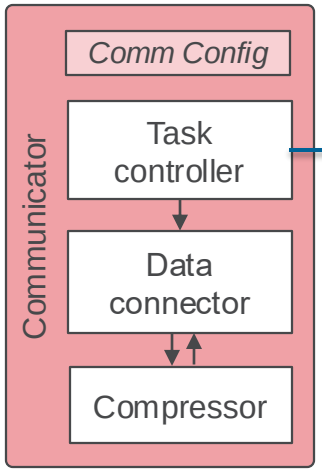



```yaml
client_configs:
  train_configs:
    # Local trainer
    trainer: "VanillaTrainer"
    optim: "Adam"
    num_local_steps: 100
    use_dp: True
    epsilon: 1
    device: "cuda:0"
    ...
  model_configs:
    ...
  data_readiness_configs:
    ...
  data_configs:
    ...
```

Most of the attributes in client configurations are supposed to be the same for all clients, so we set those shared attributes in the server configuration and shared them with all clients at the beginning of FL experiments to simplify the configuration process.

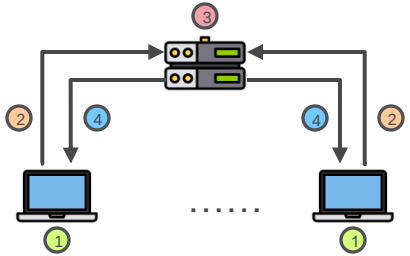
COMPONENT BREAKDOWN

Communicator – Task Controller



- Communication protocols for exchanging task signals

Client-driven (request-response model)



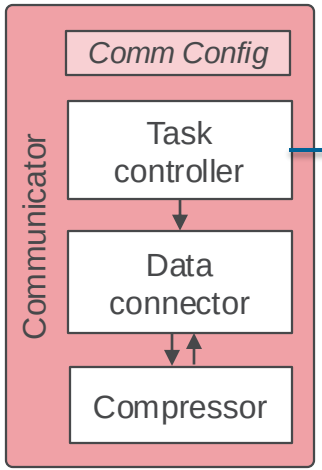
1. Perform local training
2. Request global aggregation
3. Perform global aggregation
4. Send aggregated model

Client-driven communication

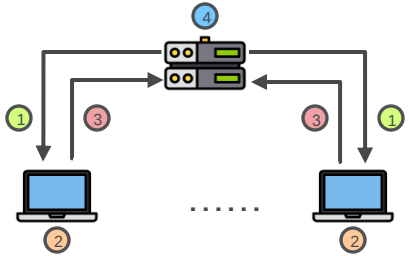
✅ Client has more autonomy for the FL experiments.

COMPONENT BREAKDOWN

Communicator – Task Controller



- Communication protocols for exchanging task signals
  - Client-driven (request-response model)
  - Server-driven (server-initiated communication)



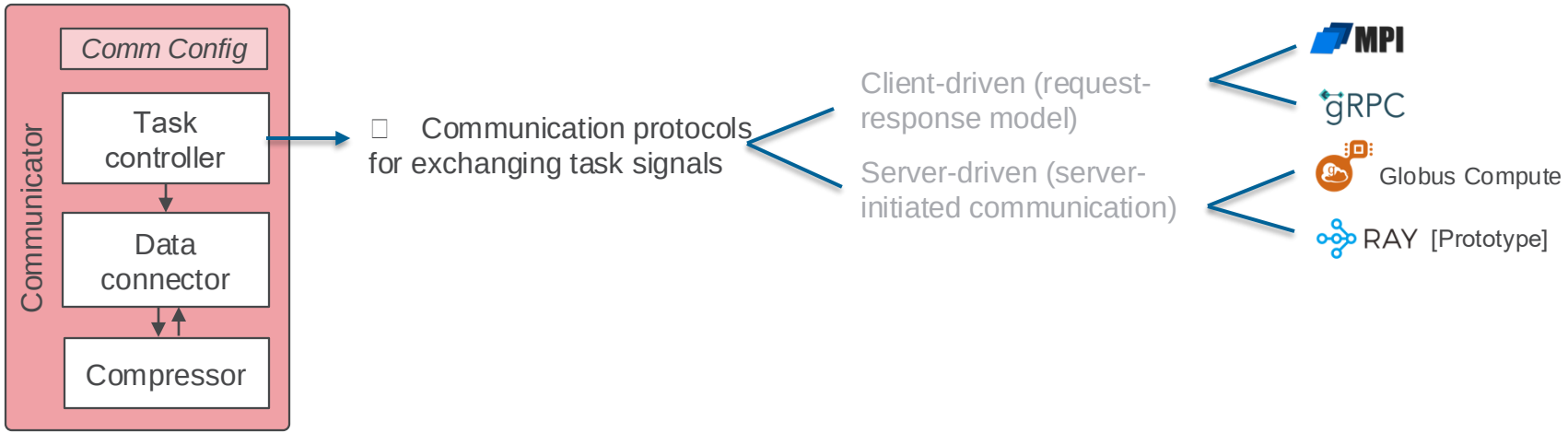
1. Send local training task
2. Perform local training
3. Send locally trained model
4. Perform global aggregation

Server-driven communication

✅ Easier Experiment coordination.

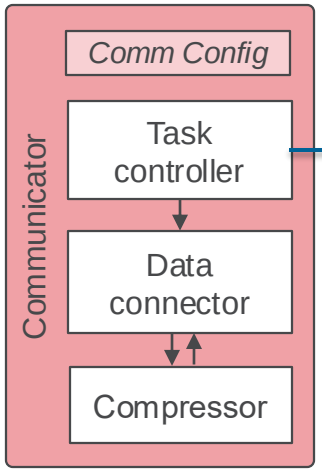
COMPONENT BREAKDOWN

Communicator – Task Controller



COMPONENT BREAKDOWN

Communicator – Task Controller



- Communication protocols for exchanging task signals
  - Client-driven (request-response model)
    - MPI
    - gRPC
  - Server-driven (server-initiated communication)
    - Globus Compute
    - RAY [Prototype]

Why do we support these many protocols?

Each protocol has its own pros and cons...

MPI: Great for simulation, but mostly just simulation

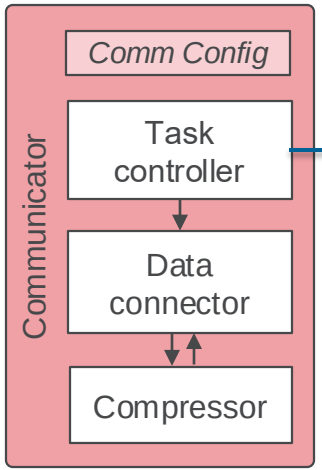
gRPC: Easy to use, but might not work on compute node with direct network access (e.g., does not work on Polaris)

Globus Compute: Easy coordination, does not require direct network access, but has size limitation, and weak support for cloud computes.

Ray: Similar pros as Globus Compute, good support for cloud, but also has size limitations

COMPONENT BREAKDOWN

Communicator – Task Controller



- Communication protocols for exchanging task signals
  - Client-driven (request-response model)
    - MPI
    - gRPC
  - Server-driven (server-initiated communication)
    - Globus Compute
    - RAY [Prototype]

Something in plan is:

Client-driven can be easily converted to server-driven… (as long as client is okay with it)

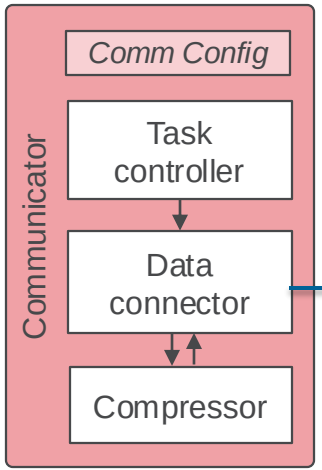
i.e., client send request: let me know what do you want me to do".

It is possible that different protocols can be integrated into one abstract communication (a server-driven one), and the server can interact with clients with hybrid communication protocols.

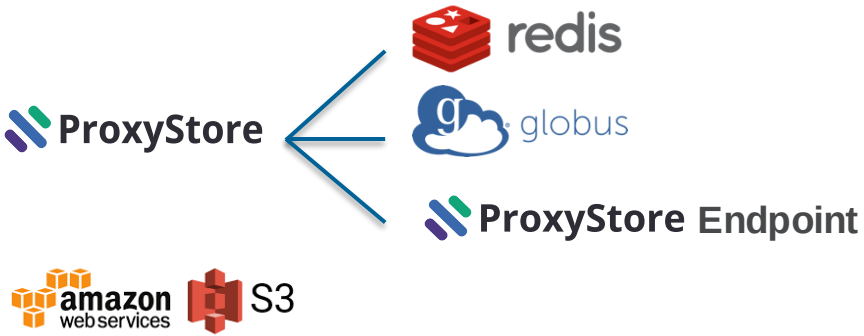
E.g.: Some local clients using MPI, some clients on PCs using gRPC, some clients on HPC using Globus Compute, and other clients on cloud using Ray…

COMPONENT BREAKDOWN

Communicator – Data Connector



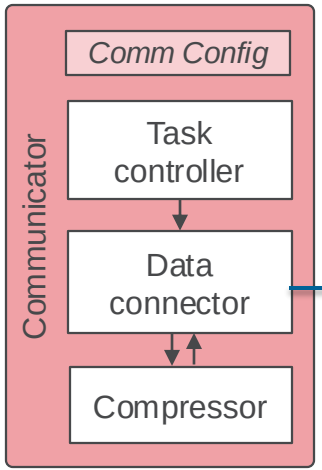
- Optional ways to communication large data




```yaml
proxystore_configs:
  enable_proxystore: True
  connector_type: "EndpointConnector"
  connector_configs:
    endpoints: [...] # endpoint ids
```

COMPONENT BREAKDOWN

Communicator – Data Connector



- Optional ways to communication large data

ProxyStore

redis

globus

ProxyStore Endpoint

amazon web services

S3

Potential benefits provided by these additional data connectors

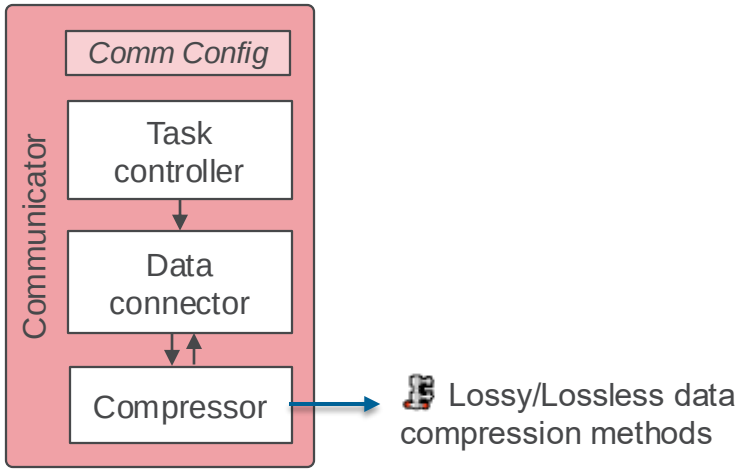
Bypass data transmission limits for certain protocols

Leverage faster data transmission methods

Checkpointing model parameters

COMPONENT BREAKDOWN

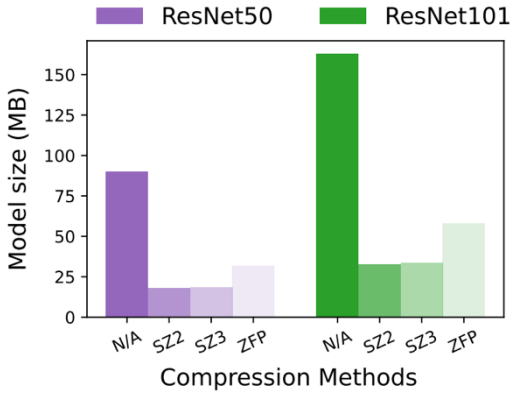
Communicator - Compressor

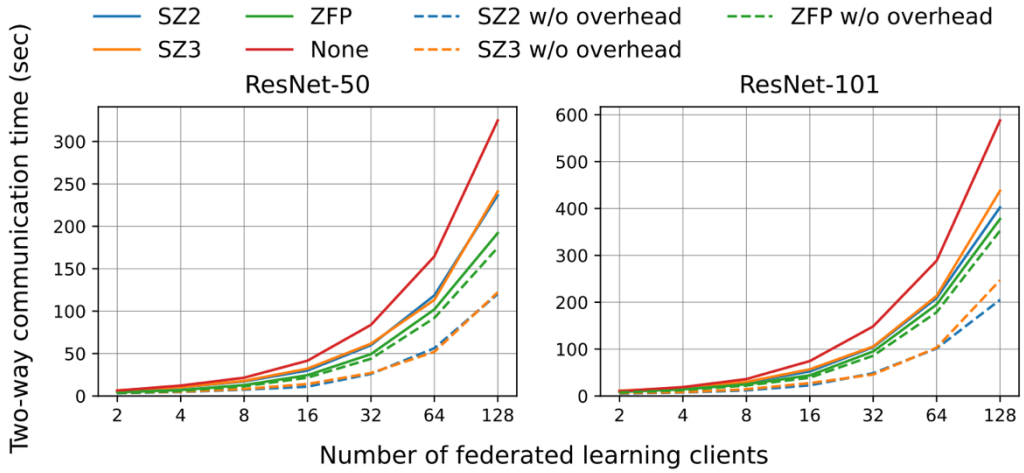



```yaml
comm_configs:
  compressor_configs:
    enable_compression: True
    lossy_compressor:  "SZ2Compressor"
    lossless_compressor: "blosc"
    error_bounding_mode: "REL"
    error_bound: 1e-3
    param_cutoff: 1024
```

COMPONENT BREAKDOWN

Communicator - Compressor





HOW TO USE APPFL

Short Answer: appfl.ai

Longer Answer: https://appfl.ai/en/latest/tutorials/firstrun.html

MPI simulation

If we want to run FL experiment in parallel using MPI, we can run the example using the following command, which runs the FedCompass algorithm with five clients.

```
mpiexec -n 6 python ./mpi/run_mpi.py --server_config ./resources/configs/mnist/server_fedcom
    --client_config ./resources/configs/mnist/client_1.yaml
```

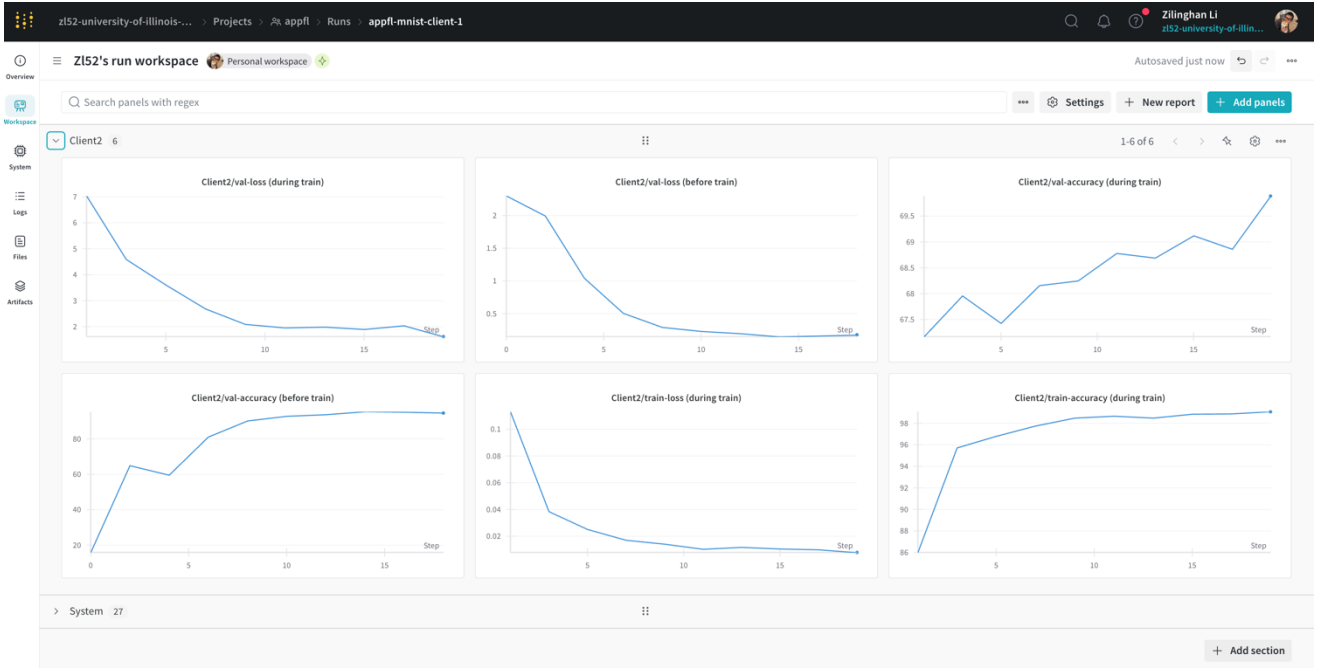
User needs to prepare a server configuration file, and client configuration files.

We also provider example runners for different communication protocols in examples folder, and we are planning providing launchers for all communication protocols.

ANOTHER USEFUL FEATURE

wanb Weights & Biases

```
client_configs:
  ...
  wandb_configs:
    enable_wandb: True
    entity: appfl-group
    project: appfl
    exp_name: appfl-mnist
```



SOME USEFUL QR CODES

THANK YOU