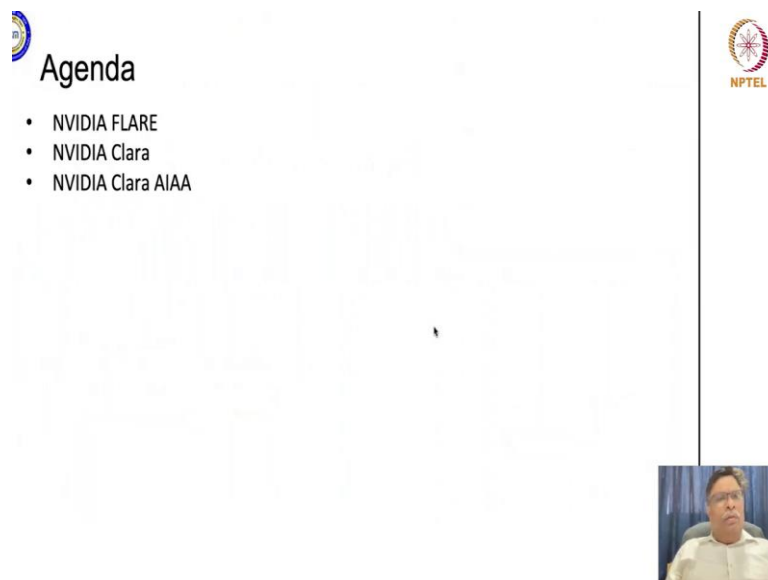


Applied Accelerated Artificial Intelligence
Prof. Satyadhyan Chickerur
School of Computer Science and Engineering
KLE Technological University
Indian Institute of Technology, Palakkad

Lecture - 60
Applied AI: Healthcare (Federated Learning, AI Assisted Annotation)
Session II - Part - 1

So, today we are trying to wind up this course with a session on Federated Learning and AI Assisted Annotation right. And in the previous session we had discussed about how federated learning could be done using TensorFlow federated right.

(Refer Slide Time: 00:40)



Agenda

- NVIDIA FLARE
- NVIDIA Clara
- NVIDIA Clara AIAA

NPTEL

And today, we will be concentrating on these three specific things right, one is NVIDIA FLARE, one is NVIDIA Clara, and the third one is NVIDIA Clara AIAA. So, I will try to show you and talk about as much as possible with some hands on, right.

You may be able to actually appreciate like to what level right annotation and other federated learning could be used ok. And maybe I will have to speed up a bit, but let us try to do it in a fashion at least as it covers everything right. So, let us start with NVIDIA FLARE.

(Refer Slide Time: 01:21)



NVIDIA FLARE :

- NVIDIA FLARE™ (NVIDIA Federated Learning Application Runtime Environment) is a domain-agnostic, open-source, and extensible SDK for Federated Learning.
- It allows researchers and data scientists to adapt existing ML/DL workflow to a federated paradigm and enables platform developers to build a secure, privacy-preserving offering for a distributed multi-party collaboration.



So, NVIDIA FLARE is basically NVIDIA Federated Learning Application Runtime Environment, which is domain agnostic its open source and extensible SDK for federated learning right.

So, the idea is you can use this runtime environment to actually convert your PyTorch program or any of your programs which you have already written to get ported into a federated learning applications framework right. So, that is when you would be using NVIDIA FLARE ok. So, it is basically going to allow you to develop a secure privacy preserving distributed multi party collaboration right. So, with your program which you have written, for training it centrally you can end up converting it into a federated learning application right, using this particular runtime environment.

(Refer Slide Time: 02:22)



So, if you see this you have got the runtime which is basically learner configuration in the sense you have PyTorch, TensorFlow program, NumPy MONAI. So, this MONAI thing we will discuss further as well and we will tell you what it is actually right. It is a open source network we will discuss that as well. So, you have your own models, you can train evaluate and update your models basically using all of this.

And then if you want to actually convert it into a federated type of a learning application right you have NVFlare runtime, which basically helps you to do all of this you can do training flows, evaluation flows, it helps you to actually change your learning algorithms and then you can implement some privacy preserving algorithm so on and so forth, right. So, the federated specification which NVIDIA FLARE gives you helps you to actually change or toggle all of this right.

(Refer Slide Time: 03:35)



The slide features the Nvidia logo on the left and the NPTEL logo on the right. The title "Nvidia NVFlare - Features :" is centered at the top. Below the title is a bulleted list of features. At the bottom right, there is a small video feed of a man speaking.

Nvidia NVFlare - Features :

- Privacy Preserving
- Training and Evaluation Workflows
- Extensible Management Tools
- Supports Popular ML/DL Frameworks
- Extensive API
- Reusable Building Blocks

So, let us understand what it is. When we say privacy preservation we are trying to preserve the privacy of the data, when you are talking of training and evaluation workflows. Technically there are two type of things which you should discuss. Training work flow in the sense, when you are trying to convert your PyTorch application into training workflow for a distributed type of environment.

So, you have got two types of workflows which you can implement. One is generally called as the scatter and gather which you all know which basically is done using in any of the distributed learning platforms right you have the scatter and gather mechanism.

So, you have the scatter and gather type of a concept wherein actually here there is something called as a central server right and this central server basically is going to broadcast tasks to the clients which you are having right. So, there will be a server then you will have various clients. So, these this particular centralized task is distributed into various smaller tasks and sent to various clients right.

And these clients will execute ok these tasks and then ultimately they will share this data back in the form of results. And then this server is going to actually gather and aggregate all of this result ok, all of these results and then come up with a federated weighting weighted average or something like that and give you a global evaluation or global training parameters or whatever right. So, that basically is something called as scatter


and gather type of a training mechanism you have a cyclic type of a mechanism also or a cyclic type of a workflow available.

Wherein a server gives something some task to the client the first client, gives certain specific task to the second client so on and so forth. And in the end when the final client completes its task right, that result comes back to the server and that is again consolidated. So, that is basically a cyclic type of a workflow right and when you talk of evaluation workflow, these are two important things other things; obviously, when you start working you will understand.

But the evaluation workflow is something like a cross site model what to say evaluation or validation for that matter. So, you can validate each client model against the server global model right and then you can analyze as to how this particular client ok is actually doing something related to the central global type of a model right. So, that basically is the evaluation. Then you have a global model evaluation which talks of basically a subset of the total global cross validated thing right.

So, these are certain things which you will be using and then you have management tools, you have got popular ML DL frameworks which you can use with it, then extensive APIs are there, reusable building blocks these are all very user friendly right.



(Refer Slide Time: 06:53)



NVIDIA NVFLARE : - Key Design Principles :

- Research Friendly
 - Ease of Experiments
 - Flexible for Innovation
 - Application Domain Agnostics
- Applicable to Real World Scenarios
 - Security and Privacy
 - System Failures and Unresponsive sites
 - Imperfect Dataset

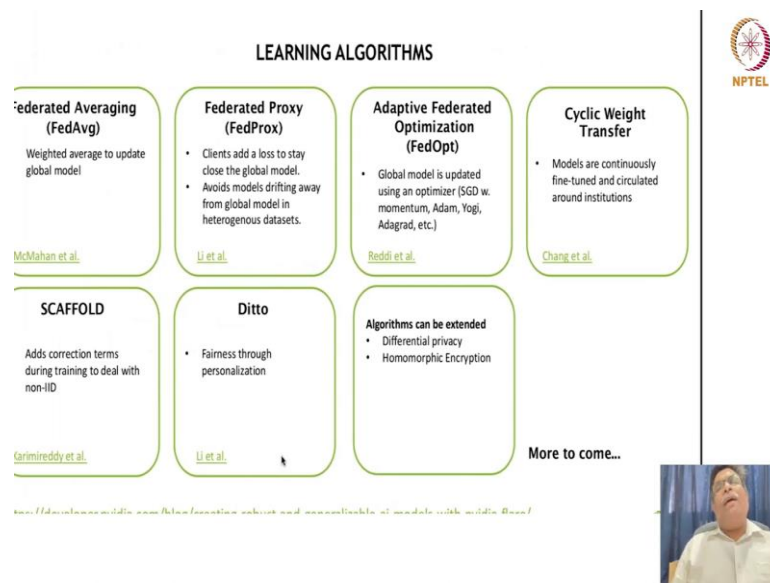
https://www.datacenter.net/Center/Client/Big-analytics-ask-nvidia-federated-learning-application-online-environment-for-developing-robust-ai-models/1001_actors.html



And then you have got a lot of domains which you can use. So, these are application domain agnostics and they are applicable to real world scenarios in a sense that you have the security and privacy in built into these runtimes.

And you can actually ok understand and deal with system failures and unresponsive sites and with imperfect data set right. So, these are the various design principles which NVIDIA NVFlare is going to be helping you with ok.

(Refer Slide Time: 07:34)



So, let us try to understand the learning algorithms, when you talk of learning algorithms what are the various learning algorithms available for you ok, when you use NVIDIA FLARE. You have got this federated averaging which basically means that you have weighted average to update global model right.

Then you have federated proxy, here each of these learning algorithms ok are basically trying to improve your model by applying federated learning 1 and then trying to optimize that federated learning ok. So, in this particular case of federated proxy, your client is going to add a loss to stay close to the global model. So, the idea is each client is going to have its own model which it is running right.

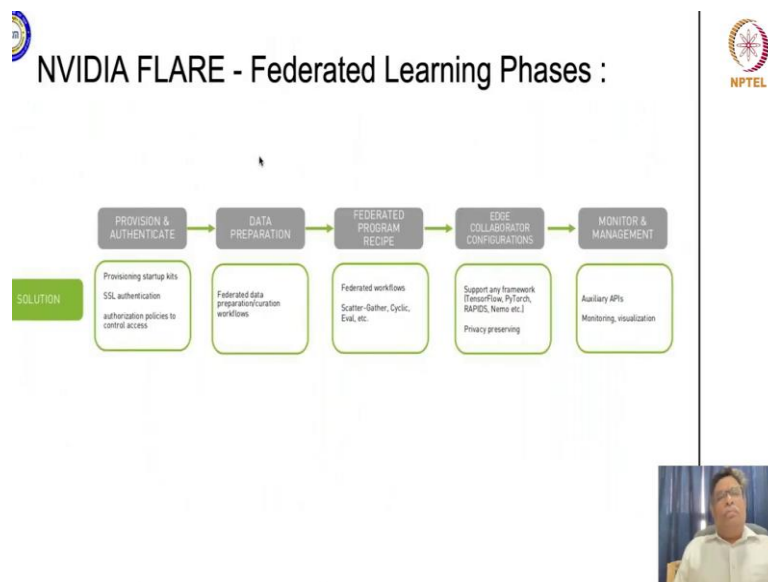
And then you try to reach as near to the global model ok and this basically avoids model drifting for heterogeneous data sets right. Since you are using heterogeneous data sets there is a lot of possibility that your model training drifts ok from one end to the other

because of the heterogeneity available in the data sets. So, you want your models not to drift away from the global model ok, and that is why you use something called as a federated proxy type of a model.

Then you have this adopted adoptive federated optimization wherein, the global model is updated using a optimizer which you already have been working with like Adam or SGD, with momentum, Adagrad and so on and so forth, right. And then cyclic weight transfer as I told you models are continuously fine tuned and circulated around institutions. So, this basically means you are trying to transfer your weights ok from the first client to the second client to the third client so on and so forth, right.

So, they may go on improving it with their data sets right, then you have something called a scaffolding which basically adds the correction terms right during the training to deal with non-IID type of data sets and then you have ditto, and then you have algorithms which can be extended right. So, you can have homomorphic encryption and so on and so forth. So, something like this ok. So, these are various algorithms which are available to you.

(Refer Slide Time: 10:04)



Now, the federated learning phases if you see ok, if you want to develop a solution right. So, there are two things which you should concentrate on, one is the workflow which you are trying to develop and another one is the privacy. Because when you are talking

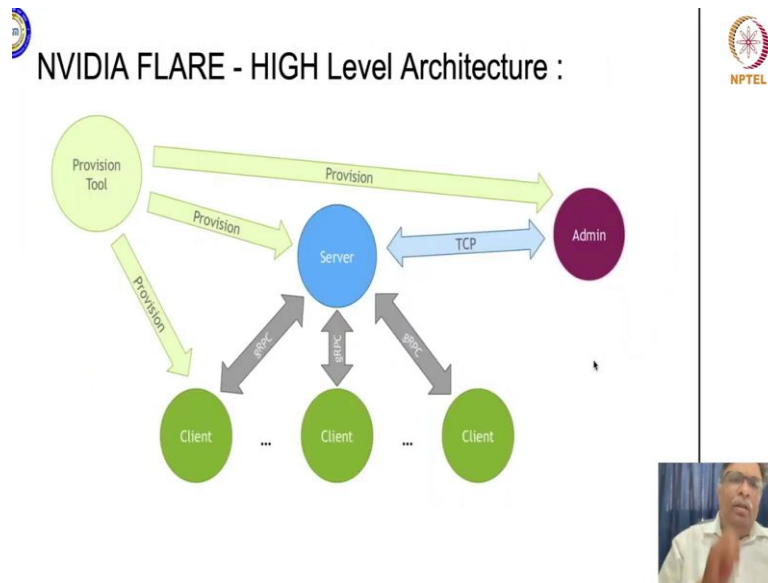
of federated learning right you will have to develop your workflow in such a manner that none of the client is able to see the other clients data ok.

So, you have got these five steps technically. So, first step is that you have to have a authentication and some provisioning. So, you basically have this provisioning startup kits, you have SSL authentication and then you have got appropriate authorization policies for access mechanisms. Then you need to prepare the data. So, you should have federated data, yesterday we showed you some mechanisms of course, they were in TensorFlow federated.

But you had to create your own federated data right and then here also you basically need to create your own federated data with certain curations need to be done for this workflow and then you have this federated workflow. You use scatter gather cyclic or eval type of workflow and then you basically are going to use any framework whether on TensorFlow, PyTorch, RAPIDS, at the edge right and you have this privacy preserving thing which will happen for inferencing.

And then at the same time you have got our APIs which will help you to monitor and visualize the whole federated learning solution right. So, the basic idea is you have to prepare the data. Before that you have to have a provisioning and authentication, then you come up with your the federated workflow and then you do that inferencing which you want to do it at the edge. And then you basically are in a position to monitor and manage whole of this right ok.

(Refer Slide Time: 12:19)

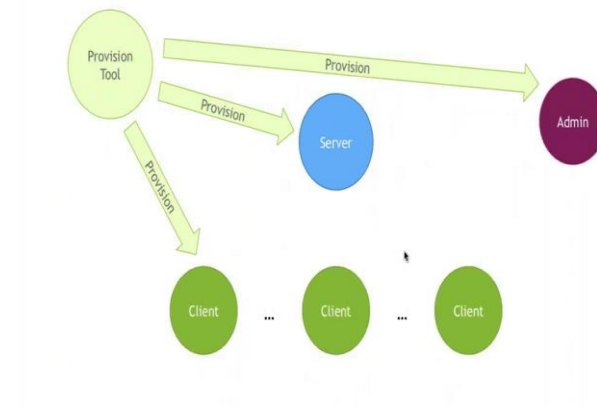


So, this is a very very high level architecture right, you have a provisioning tool which will help you to provision, this is the server, these are the clients, then there is a admin. So, you in our case today we will be showing you a server and the admin is the same and there is one client. So, we are going to show you with a minimal thing ok, but technically speaking, if you really want to implement it in a production environment in real life environment you will have a server, you will have admin, and you will have a number of clients right.

And then this is an open source RPC right Remote Procedure Call type of a communication and gRPC is the recent open source standards which you which this client server communication uses, right. So, there is this TCP IP which is required for communication between the server and the admin. And then the server and the client you have got this remote procedure call which is very very secure and which will maintain privacy so gRPC. And then you have a provision tool which will help you to provision all of this, right.

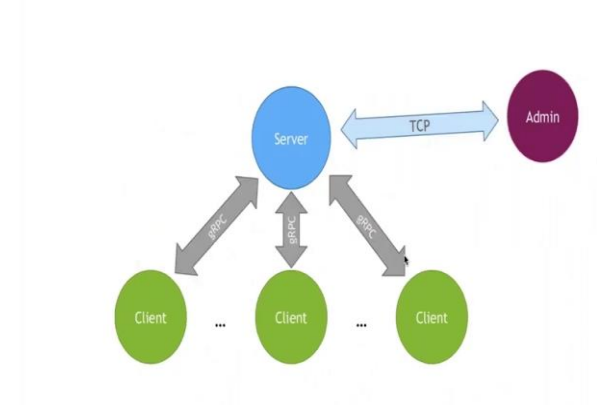
(Refer Slide Time: 13:25)

NVIDIA FLARE - HIGH Level Architecture :

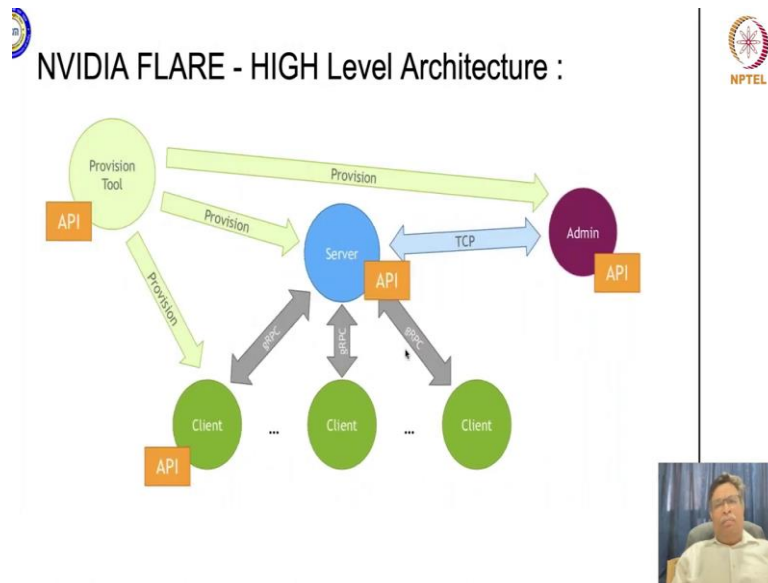


(Refer Slide Time: 13:27)

NVIDIA FLARE - HIGH Level Architecture :



(Refer Slide Time: 13:29)



So, this is how it is going to happen yeah. So, where are the various APIs which you are going to use ok? These are available to you with NVIDIA FLARE right ok.

(Refer Slide Time: 13:40)

The slide is titled 'NVIDIA FLARE - APIs' and lists four main API categories with their functions:

- Open Provision API
 - Defines overall project configuration and generates mutually-trusted configuration packages for server and clients using Provisioner and Builder modules.
- Server Controller API
 - The Controller is a python object that defines the global Federated Learning control flow via Tasks and Events.
- Client Worker API
 - The Worker API is used to define Executors that perform Tasks orchestrated by the Server Controller API
- Admin API
 - The Admin API provides a means to control the Federated Learning System and allows application developers to manage operation via external interfaces (e.g., Web UI).

The slide includes a small NVIDIA logo in the top left, an NPTEL logo in the top right, and a video feed of a speaker in the bottom right.

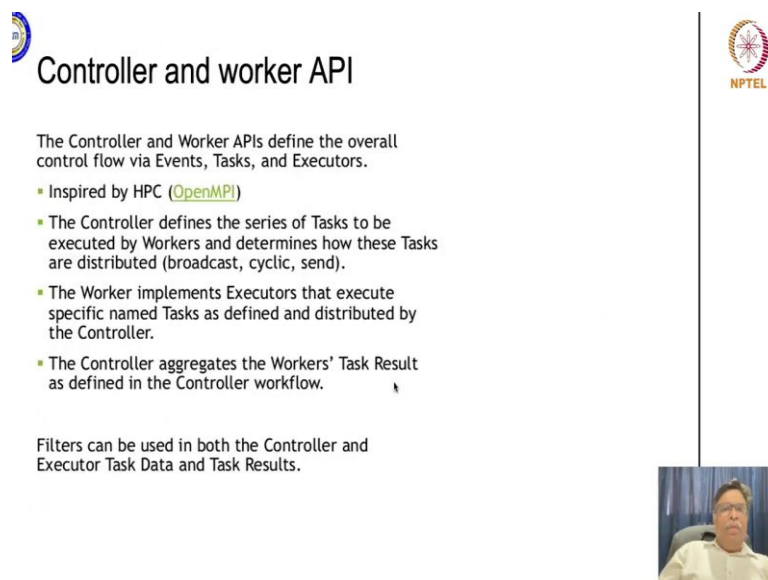
So, open provision API in the sense you have this provisioning API here, then you have a server controller API, then you have client worker API, and then you have admin API. So, at all these places right you have got this Application Programming Interfaces or API is available to you.

So, let us just briefly see what all each of these APIs help you to do right. So, open provision API basically defines the overall project configuration and generates mutually trusted configuration packages for servers and clients using provisioner and builder modules. So, your whole project configuration which is basically a trusted configuration ok, between the server and the client has to be developed and you are using this provision and builder modules for doing it.

Then the server controller is a python object, that defines the global federated learning control flow via task and events. So, this open provisioning system helps us to configure everything, server controller helps us to control the federated learning flow, client API or the client worker API is used to define executors that perform tasks which are orchestrated by the server controller.

So, each client basically is going to actually define ok certain executors, which are going to perform certain specific task ok and these will be orchestrated obviously, by the server ok. And then the admin API basically controls the federated learning system and allows application developers to manage operation via external interfaces or web user interfaces or whatever, right. So, these are how various APIs could be used ok.

(Refer Slide Time: 15:47)




 **Controller and worker API**

The Controller and Worker APIs define the overall control flow via Events, Tasks, and Executors.

- Inspired by HPC ([OpenMPI](#))
- The Controller defines the series of Tasks to be executed by Workers and determines how these Tasks are distributed (broadcast, cyclic, send).
- The Worker implements Executors that execute specific named Tasks as defined and distributed by the Controller.
- The Controller aggregates the Workers' Task Result as defined in the Controller workflow.

Filters can be used in both the Controller and Executor Task Data and Task Results.





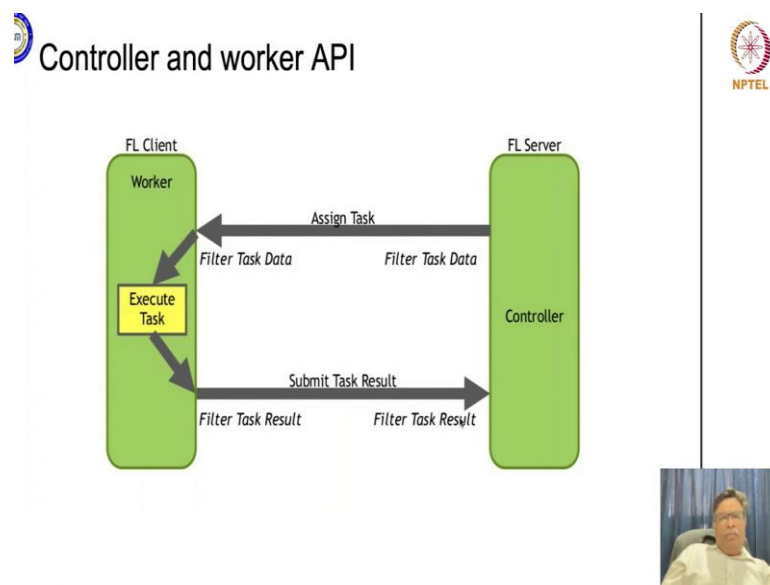
So, controller and worker API defines the overall control flow which basically is related to events, tasks and executors. So, the idea is you have got this open MPI right. So, this

controller and worker API actually is in context of developing a API, which basically revolves around the concept of open MPIs, open MPI right.

So, the controller defines the series of tasks to be executed by the worker and determines how these tasks are distributed. So, you can broadcast, you can do cyclic, you can do scatter gather and something of that sort, the worker implements executors that execute specific name tasks are defined and distributed by the controller.

So, this is basically a MPI type of a scatter gather mechanism. So, I would not go into the details of this. So, basic idea is you distribute, process and gather the results right ok.

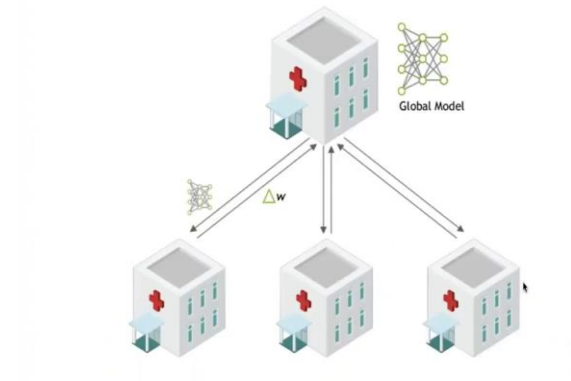
(Refer Slide Time: 16:54)



So, this is how it is going to happen basically it is something like this, you have a federated learning client you have a federated learning server. So, it is something like the controller is there and the worker has to execute the task. So, you assign the task, it executes the task and submits the task result back ok. So, this is how basically the controller and worker API behave ok, right.

(Refer Slide Time: 17:25)

Scatter Gather Controller Model for Training



So, let us go to the scatter gather controller model for training. So, you have a global model ok. So, for example, this is one client, this is another client, this is the third client. So, each of these clients will basically run their own program or model here and each will actually get us these weights, ok. So, these are the weights which are required to improve the model ok, for this local client here. So, again this client or this site will have its own model which is running.

So, it will have delta w 2 set of delta w 2, here also its set of delta w 3 and all of this will actually be used for training this global model, right. So, this is how it is scatter gather controller model.

(Refer Slide Time: 18:19)

Scatter Gather Controller Model for Training

1. Server initializes model
2. For number of rounds:
 1. Server broadcasts global model to workers
 2. Workers validate global model and train on their Data
 3. Workers keep track on their locally best model (Personalization)
 4. Workers send back updated model or updates
 5. Server Gathers (Aggregates) updates and updates the global model



So, you basically initialize the model. So, for a number of rounds, server broadcast the global model to the workers, workers validate global model and train on the data, then workers keep track of their locally best models and workers send back updated models and server gathers and aggregates it, right. So, this is how it is going to happen.

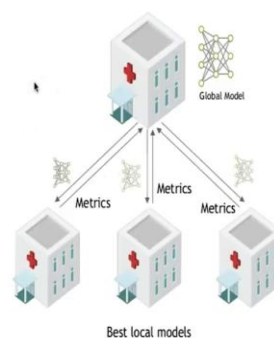
(Refer Slide Time: 18:39)

CONTROLLERS FOR MODEL EVALUATION:

FedEval (Global Model Validation/Cross-Site Validation)

1. Server sends models (e.g. global model registered best local models) to each worker for Evaluation
2. Server gathers the resulting metrics


Metrics	Evaluation sites			
	Site-1	Site-2	...	Site-N
Global (Final)
Global (Best)
Site-1
Site-2
...
Site-N



So, this is how basically the model evaluation is done. You have actually to send the information about your best local models, which is the metric which you are going to send. So, it can be a set of values for your delta w ok. So, this is a way of cross site


validation or a global model validation. So, these are your metrics, global final, global best, site-1, site-2, site-3 so many models. So, for each of these sites right how is this behaving? How is this behaving? What would be with regards to this? Ok all of this type of analysis is done.


(Refer Slide Time: 19:21)



EXAMPLES


brats18	Add brats example (#63)
cifar10	Add license header (#75)
hello-cyclic	add README files (#71)
hello-mnist	add README files (#71)
hello-numpy-cross-val	add README files (#71)
hello-numpy-sag	add README files (#71)
hello-pt	add README files (#71)
hello-tf2	add README files (#71)
prostate	Add prostate example (#65)
README.md	add README files (#71)






And then we will show you some example of cifar10, we will go to that of course and yeah.


(Refer Slide Time: 19:32)



Steps :

- Comprehensive example for researchers to compare algorithms
 1. Set up a virtual environment
 2. Create your FL workspace
 3. Run automated experiments
 1. Varying data heterogeneity of data splits
 2. Centralized training
 3. FedAvg on different data splits
 4. Advanced FL algorithms (FedProx and FedOpt)
 5. Secure aggregation using homomorphic encryption
 6. Differential privacy
 4. Results

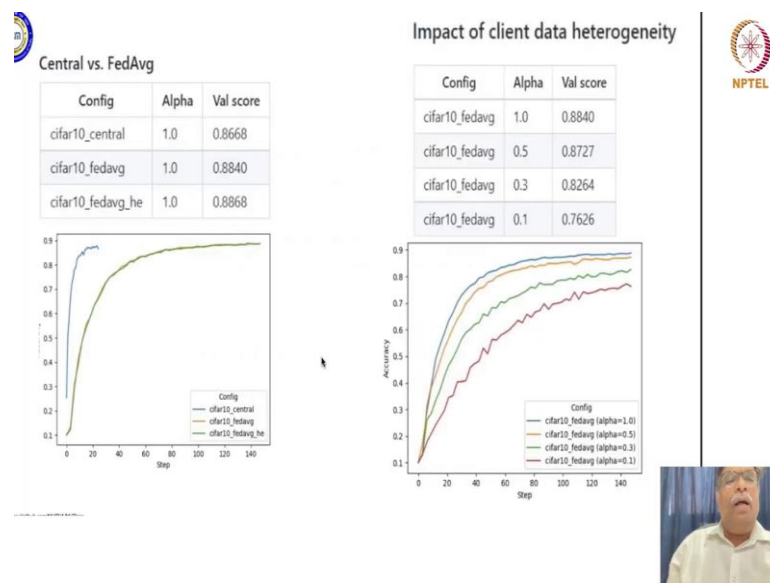




So, this is basically we will show you the example. So, but the basic idea is right, when you are trying to compare various algorithms right. So, the idea is that you set up your own virtual environment, create your own federated learning workspace, then you can run various experiments to basically understand right how basically everything works. And then you can basically develop your own models and then further on go on improving upon that, right.

So, this is how it is right. So, you can vary the data heterogeneity of the data splits then you can do centralized training, federated average on different data splits, then you have this federated algorithms which we talked about. And then how do you do homomorphic encryption and then privacy differential privacy so on and so forth. So, you have got a lot of things which you can work with ok so yeah.


(Refer Slide Time: 20:39)



So, see here for the cifar10 ok, the central cifar10 configuration what is the value of alpha and what is the score right validation score? So, federated average, this is cifar federated average and this is ok something like that, then this is again configuration how is that validation to score changes right with the alpha value getting changed right.

So, this is how is the client data heterogeneities impact ok, on the validation score so on and so forth. There are so many parameters which you can vary and see we are just showing you this.

(Refer Slide Time: 21:29)



CROSS-SITE VALIDATION AND GLOBAL MODEL EVALUATION

Performance of locally best models (selected by best validation score on local data) using
 (a) local training data alone and (b) after federated learning.

		(a) Local: Test									(b) Federated: Test								
		client	1	2	3	4	5	6	7			client	1	2	3	4	5	6	7
T _{train}	1	0.62	0.59	0.44	0.02	0.02	-0.01	0.04	T _{train}	1	0.62	0.62	0.48	0.15	0.23	0.24	0.11		
	2	0.15	0.56	0.02	-0.01	-0.00	0.00	-0.01		2	0.22	0.65	0.11	0.04	0.00	0.00	-0.01		
	3	0.19	0.01	0.64	0.02	0.07	0.00	0.05		3	0.41	0.17	0.63	0.07	-0.00	0.01	-0.01		
	4	0.11	0.02	-0.00	0.63	0.52	0.61	0.50		4	0.06	0.48	-0.02	0.69	0.57	0.65	0.52		
	5	-0.00	-0.01	-0.03	0.54	0.62	0.65	0.31		5	0.24	0.13	0.02	0.64	0.62	0.69	0.52		
	6	0.01	0.11	-0.02	0.49	0.59	0.71	0.32		6	0.23	0.01	-0.00	0.53	0.68	0.76	0.31		
	7	0.03	0.05	-0.05	0.40	0.37	0.46	0.69		7	0.10	0.21	0.13	0.55	0.44	0.52	0.77		
									Global										
diag. mean									0.64	diag. mean									0.68
off-diag. mean									0.18	off-diag. mean									0.26

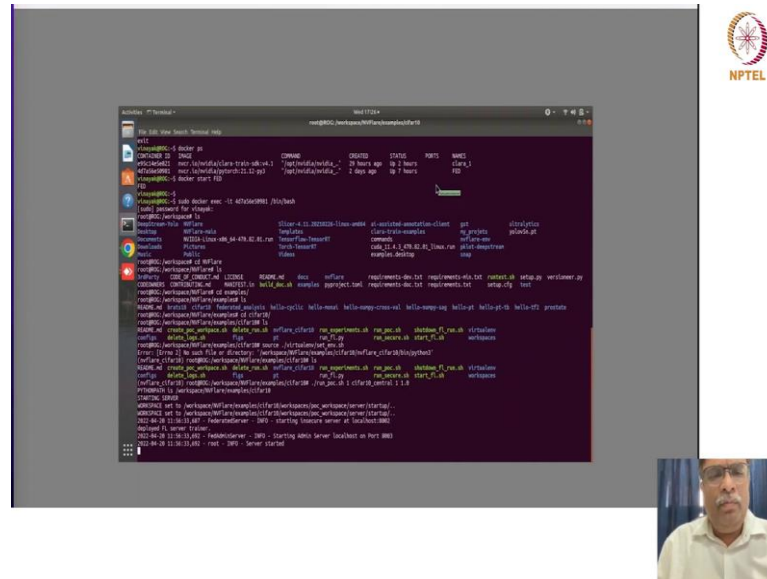
Federated Learning for Breast Density Classification: A Real World Implementation



So, this is cross site validation, this is just an example to show you like how people use this ok for selecting the best validation score. Because you are trying to work with the local data, but you are trying to improve upon some global data right. So, if you send something which is not good ok, then how is that global data is also going to be affected all of that needs to have some correlation right.

And then you need to do some cross site validation, and then based on that you can actually see and send in that value ok. So, yeah so before that let us try to do this NVIDIA FLARE hands on demo first and then we will go to Clara ok.

(Refer Slide Time: 22:33)



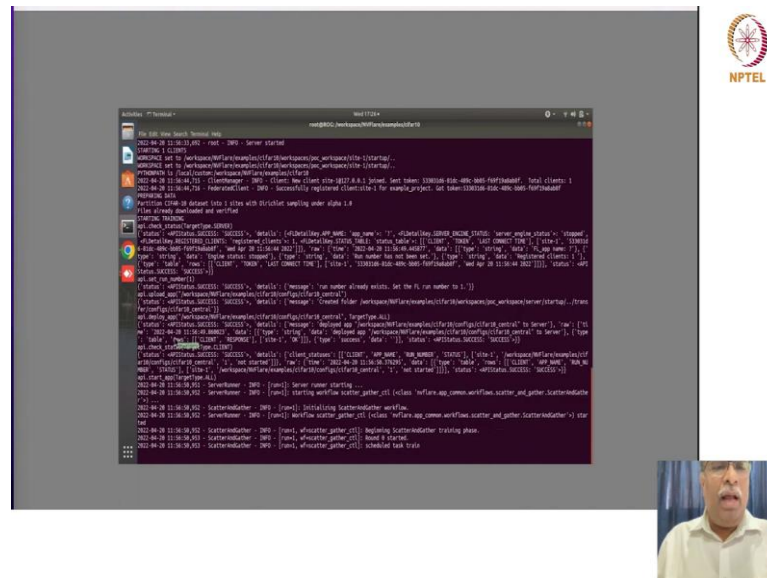
So, let me just share yes. So, 1 minute let me just increase in the size of it, yes. So, we are going to show you cipher central ok. So, we are trying to use a NVIDIA docker container which basically is pulled from NVIDIA cloud, which is NGC ok and the name is this fed ok. So, we are trying to download a docker container which is fed ok.

And this basically is a docker container, which is going to run federated training or federated learning example with ok 1 server, 1 client ok for training. So, the server itself behaves as a admin here. So, let us start it and then we will try to see ok. So, we have come to the workspace and then in this workspace, if you see we have got this NVIDIA FLARE. So, we will open this and see what all is available.

So, if you go to the CD sorry, if you go to the examples folder. So, you have got a lot of examples you have got cifar10, federated analysis then you have this hello world which is cyclic you can use hello with MONAI, then you have got cross validation so many of them, right. So, we are trying to show you this cifar10 example, right. So, if you see here, you have got proof of concept you can run secure you can run certain specific experiments and then you have got this virtual environment.

So, let us try to basically go to this virtual environment, yes. Why is this error ok? Yeah we have come to the NVFlare cifar10. So, that is ok so that should not be of concern ok, yes. So, we are going to run this proof of concept now, with this condition which tells us that we are going to use ok a central algorithm with 1 server and 1 client.

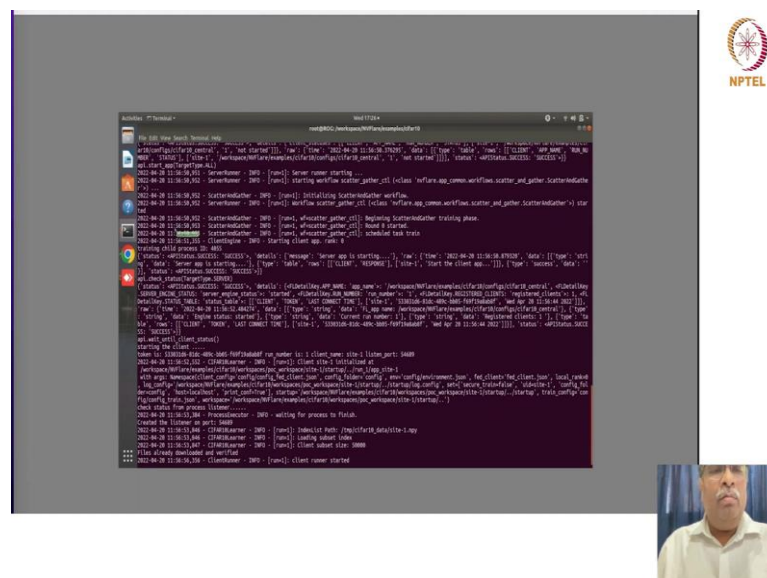
(Refer Slide Time: 25:56)



The terminal window displays the execution of a script that sets up a server and clients for a machine learning project. The script includes commands for setting environment variables, installing dependencies, and starting the server and clients. The output shows the successful execution of these commands and the initialization of the server and clients.

NPTEL

(Refer Slide Time: 26:09)

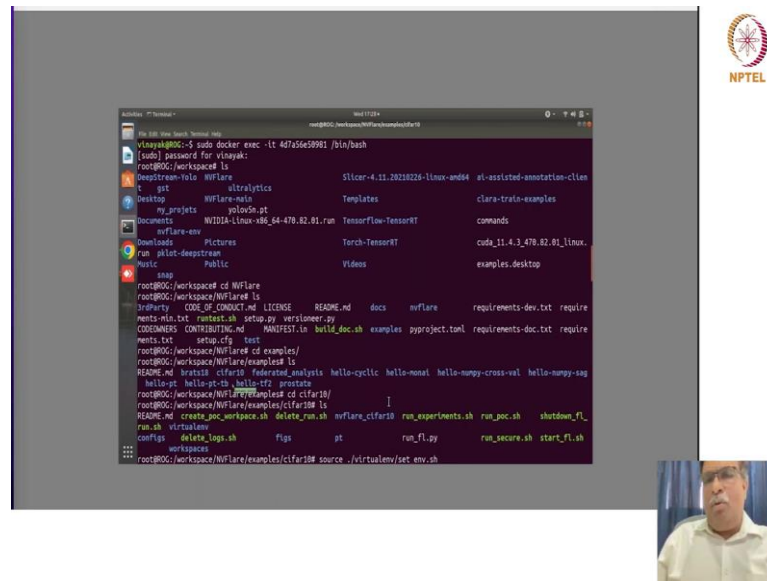


The terminal window displays the execution of a script that sets up a server and clients for a machine learning project. The script includes commands for setting environment variables, installing dependencies, and starting the server and clients. The output shows the successful execution of these commands and the initialization of the server and clients.

NPTEL

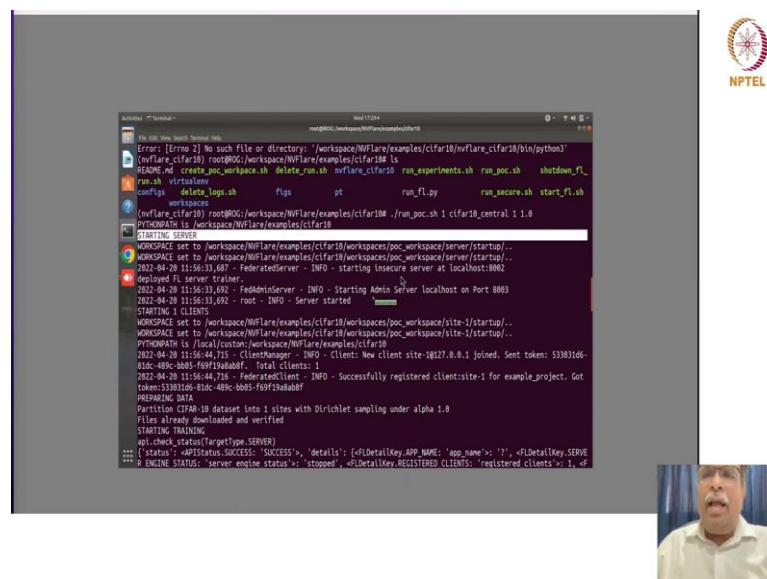
So, it is now starting the server, see here we have started the server, we have started the admin server and we are starting one of the clients. Now, we are able to prepare the data it has already prepared.

(Refer Slide Time: 26:28)



So, if you see this show it once again from what we have done till now. Maybe for people to understand ok. So, yeah so this is this NVFlare folder and if you go to that folder you have got a lot of examples. So, you have got examples which are cifar10, you can do federated analysis, you can do cyclic type of hello program, you have got MONAI hello then there are so many of the examples right.

(Refer Slide Time: 26:44)



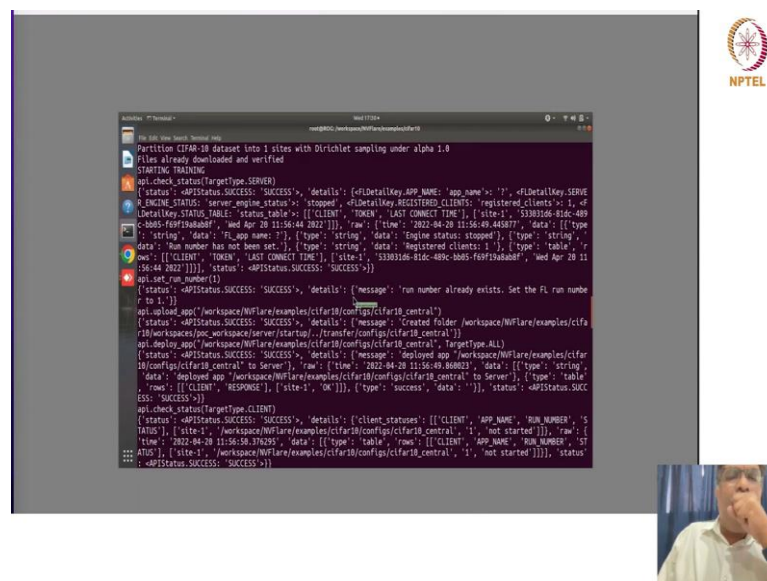
So, we will try to work with this cifar10 example right. So, first thing is you have to set up your virtual space right and once you set up your virtual space, you can go to this

proof of concept creation right. So, one minute workspace ok we are first starting the server here and then you are in a position to see that we have deployed the server trainer. We are starting the admin server, which is running on this local host port 8003.

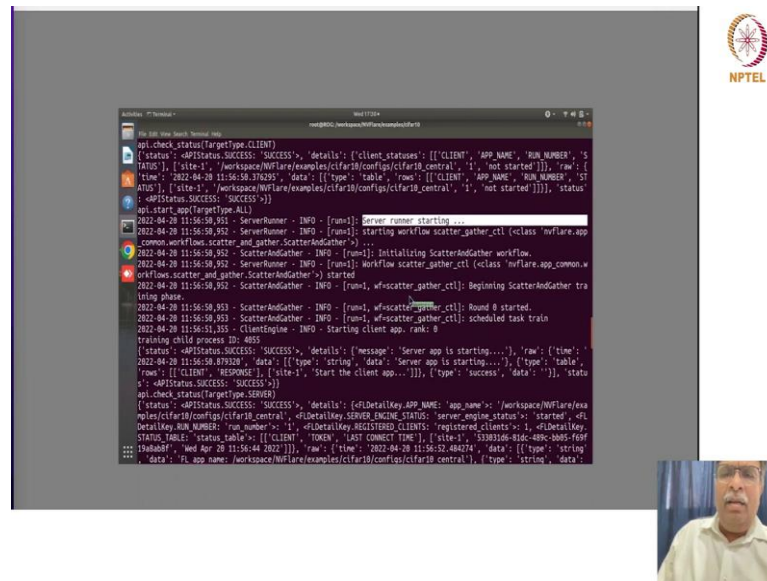
So, we have started the server, we have started one client also ok and that particular client basically is now a client which is going to send local information to the global server, right. So, this is how it is. You can do more than one clients also right you could have done 2 or 3, but the point is when I am showing this it gets struck. So, that is the reason why I showed 1, otherwise we have tested it up till 6, that works fine ok in a good environment. And we are now preparing data also.

So, here we are partitioning the CIFAR-10 data ok, with one site with this sampling with alpha 1 dot 0 right. So, this these are certain parameters ok, for sampling and now trying to understand how it is partitioned right. So, we are partitioning it assuming that there is one global and one local server now at present ok.

(Refer Slide Time: 28:36)



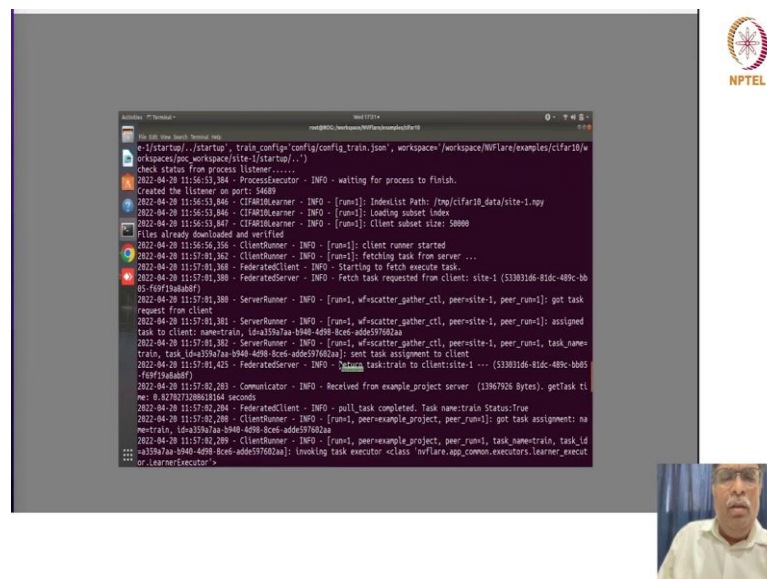
(Refer Slide Time: 28:43)



So, you are starting the training. So, you have got various this thing. So, you are talking of APIs. So, if you see here, you have got this scatter gather workflow which you are using ok, you are initializing it and then you are running it ok.

So, success, then the target is server then you get successes so many of this thing is happening ok.

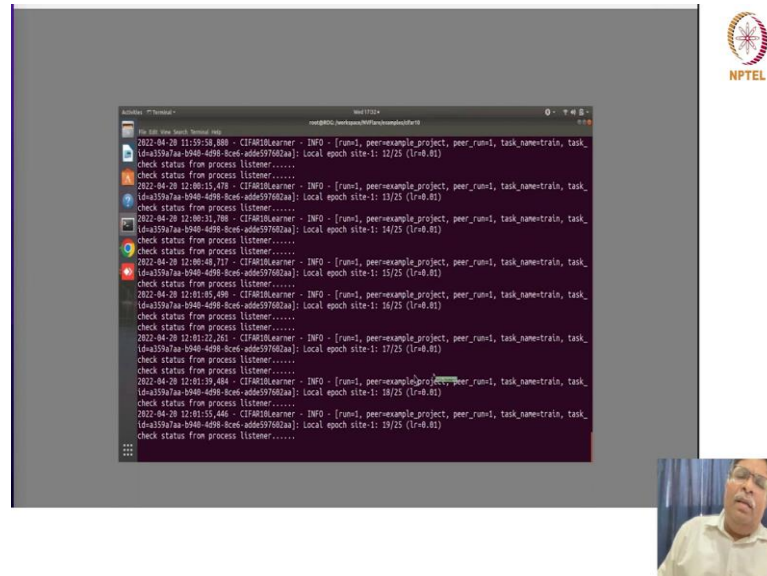
(Refer Slide Time: 29:01)



And then there is a process executed which talks of waiting for the process to finish. You have got all of these federated client information, executing the task federated server

request fetching the task requested from the client so on and so forth. So, all of this is happening correctly then yeah.

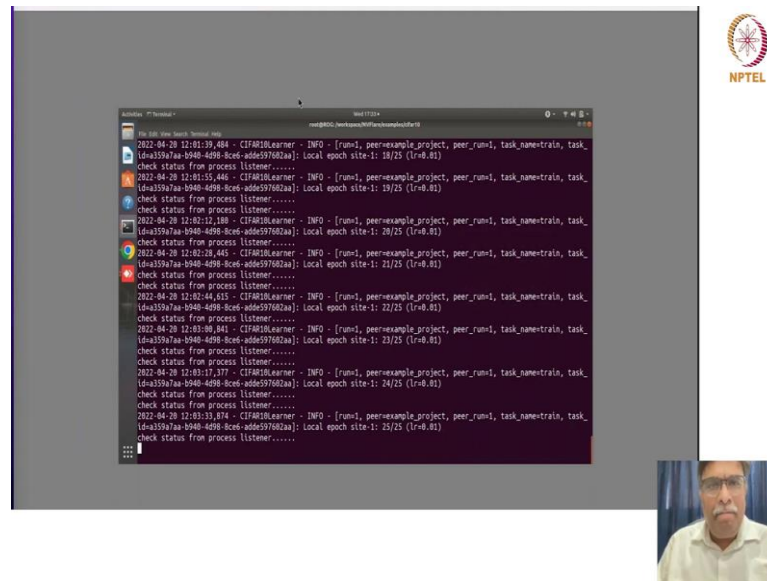
(Refer Slide Time: 29:34)



So, it is basically trying to see tell you that local epoch right is now 17 out of 25 epochs which we have finalized ok. This is 16 epochs this is a local thing. So, if you have got about 2 or 5 clients like this for each of these clients right you will have you to you will have these local epochs for each of the sites.

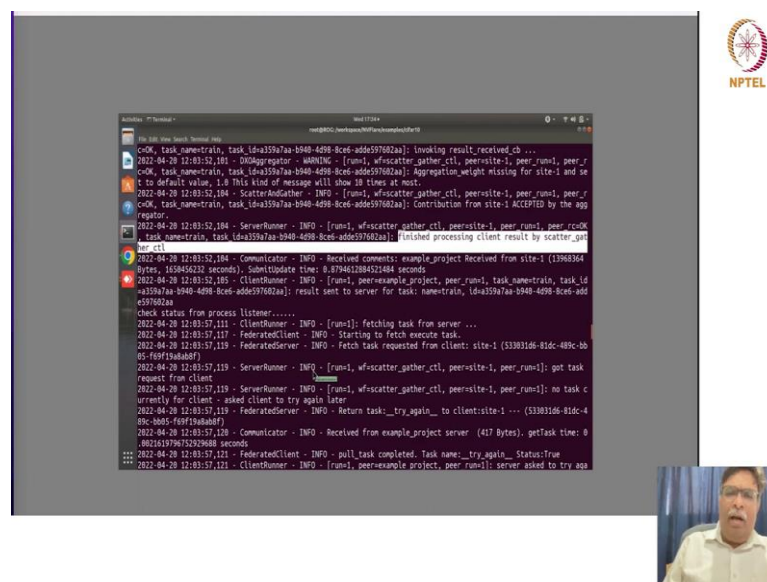
So, local epoch site-2, local epoch site-3 local epoch site-4 so on and so forth. And since each of these is to be showing you a maximum epoch of 25 now. So, each of this site will have like this ok, at the server end.

(Refer Slide Time: 30:48)



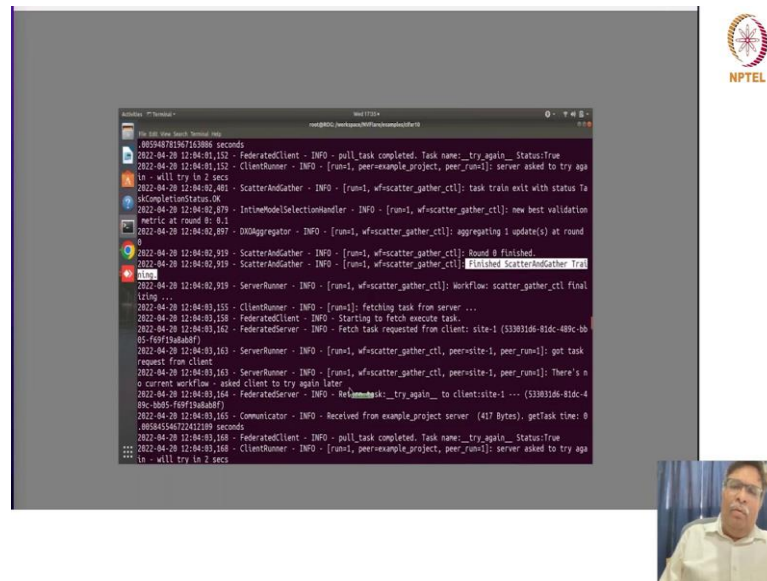
So, we will wait for all of this to complete the idea was to just show you how it happens, ok. So, if you see the learning rate it is point o 1. So, another two epochs. So, it got stuck again ok.

(Refer Slide Time: 32:36)



So, if you see here, that you have now completed everything ok and the through the communicator you have received ok from site-1 ok so many bytes of information in so much seconds and you basically have to update it now, ok. So, all of this happens ok.

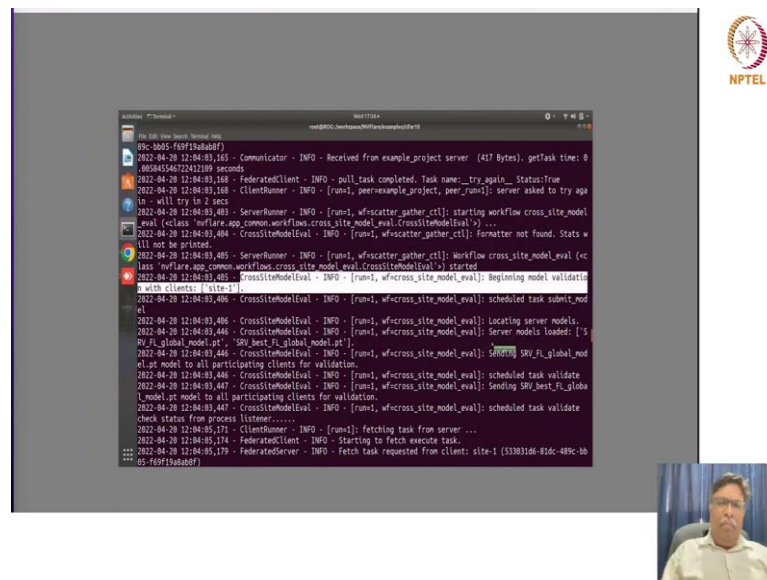
(Refer Slide Time: 33:05)



The terminal window displays a series of log messages. Key messages include: "pull task completed. Task name: try_again Status: True", "server asked to try again - will try in 2 secs", "task train exit with status TaskCompletionStatus.OK", "new best validation metric at round 8: 8.1", "aggregating 1 update(s) at round", "Round 8 finished", "Finished ScatterAndGather Train", "Workflow: scatter_gather_ctl finalizing ...", "Fetching task from server ...", "Starting to fetch execute task.", "Fetch task requested from client: site-1 (53303166-81dc-489c-bb05-f69f15a0d8f)", "get task request from client", "There's no current workflow - asked client to try again later", "Releasing try_again to client: site-1 ... (53303166-81dc-489c-bb05-f69f15a0d8f)", "Received from example_project server (417 Bytes). getTask time: 0.005645546722412109 seconds", "pull task completed. Task name: try_again Status: True", "server asked to try again - will try in 2 secs".

So, this will go on happening. So, run 1, run 2 ok. So, now, here if you see this once you have updated and done things. So, now, the idea is after run 1 ok, and after this scatter gather control gives you the final information, that the round is finished and you have finished the scatter and gather training right.

(Refer Slide Time: 33:41)

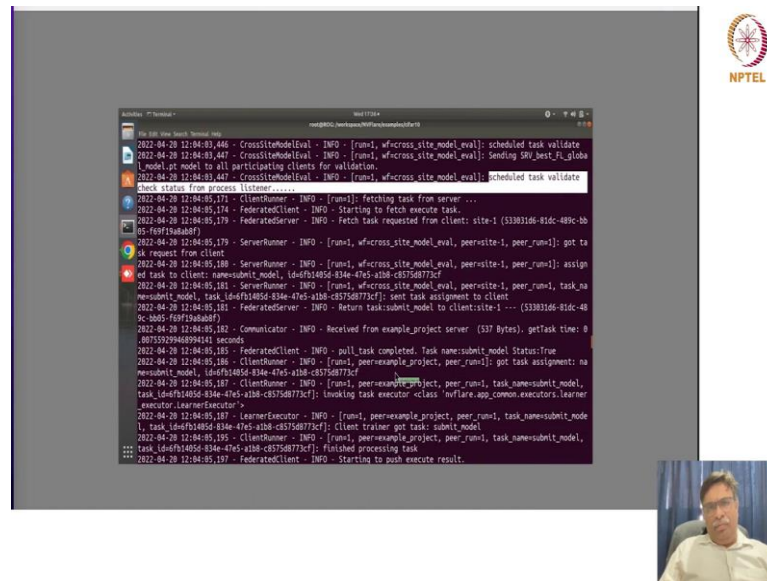


The terminal window displays a series of log messages. Key messages include: "Received from example_project server (417 Bytes). getTask time: 0.005645546722412109 seconds", "pull task completed. Task name: try_again Status: True", "server asked to try again - will try in 2 secs", "starting workflow cross_site_model_eval (class 'nvflare.app_common.workflow_cross_site_model_eval.CrossSiteModelEval') ...", "Formatter not found. Stats will not be printed.", "Workflow cross_site_model_eval (<class 'nvflare.app_common.workflow_cross_site_model_eval.CrossSiteModelEval') started", "Beginning model validation", "scheduled task submit_model", "Locating server models.", "Server models loaded: ['SRV_FI_global_model.pt', 'SRV_best_FI_global_model.pt']", "scheduled task validate", "Sending SRV_best_FI_global_model.pt model to all participating clients for validation.", "scheduled task validate", "check status from process listener.....", "Fetching task from server ...", "Starting to fetch execute task.", "Fetch task requested from client: site-1 (53303166-81dc-489c-bb05-f69f15a0d8f)".

So, now with respect to site-1 ok, there has to be this cross site model evaluation which will happen.

So, this model validation thing ok, with the clients is going to happen. So, it is a cross site validation which is going to happen. So, you have the server model which is there, which is basically a standard model which you would be using right. And then you are basically going to compare that models performance with the local models performance and that is called as cross site model evaluation ok.

(Refer Slide Time: 34:27)

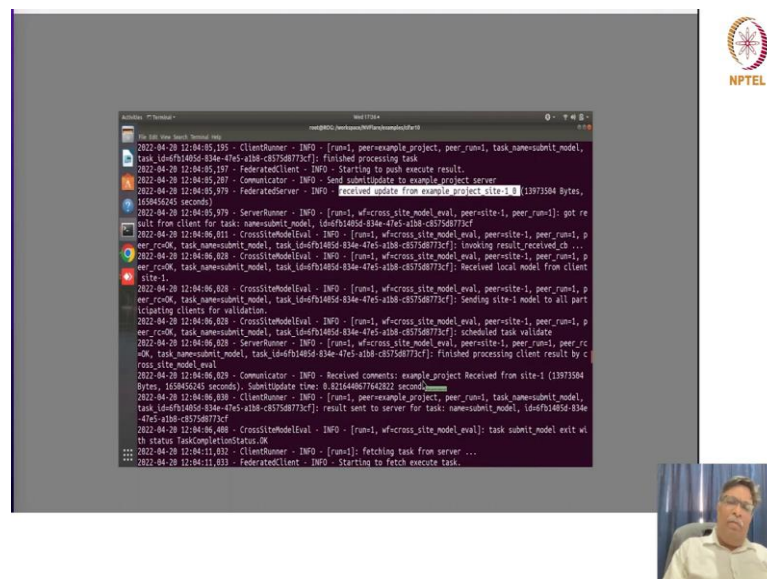


```

2022-04-20 12:04:43.446 - CrossSiteModelEval - INFO - [run], wfcross_site_model_eval: scheduled task validate
2022-04-20 12:04:43.447 - CrossSiteModelEval - INFO - [run], wfcross_site_model_eval: Sending SV_test_F1_global_model.pt model to all participating clients for validation.
2022-04-20 12:04:43.447 - CrossSiteModelEval - INFO - [run], wfcross_site_model_eval: Scheduled task validate
Send status from process listener
2022-04-20 12:04:45.171 - ClientRunner - INFO - [run]: fetching task from server ...
2022-04-20 12:04:45.174 - FederatedClient - INFO - Starting to fetch execute task.
2022-04-20 12:04:45.179 - FederatedServer - INFO - Fetch task requested from client: site-1 (53303166-81dc-489c-bb05-f69f19abab8f)
2022-04-20 12:04:45.179 - ServerRunner - INFO - [run], wfcross_site_model_eval, peersite-1, peer_run1: got task request from client
2022-04-20 12:04:45.180 - ServerRunner - INFO - [run], wfcross_site_model_eval, peersite-1, peer_run1: assigned task to client: namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf
2022-04-20 12:04:45.181 - ServerRunner - INFO - [run], wfcross_site_model_eval, peersite-1, peer_run1, task_namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf: sent task assignment to client
2022-04-20 12:04:45.181 - FederatedServer - INFO - Return task:submit_model to client:site-1 --- (53303166-81dc-489c-bb05-f69f19abab8f)
2022-04-20 12:04:45.182 - Communicator - INFO - Received from example_project server (537 Bytes), getTask time: 0.0075929468994141 seconds
2022-04-20 12:04:45.185 - FederatedClient - INFO - pull task completed. Task namesubmit_model Status:True
2022-04-20 12:04:45.186 - ClientRunner - INFO - [run], peerexample_project, peer_run1: got task assignment: namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf
2022-04-20 12:04:45.187 - ClientRunner - INFO - [run], peerexample_project, peer_run1, task_namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf: Invoking task executor: class, nvflare.app_common.executors.Learner_executor.LearnerExecutor
2022-04-20 12:04:45.187 - LearnerExecutor - INFO - [run], peerexample_project, peer_run1, task_namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf: Client trainer got task: submit_model
2022-04-20 12:04:45.195 - ClientRunner - INFO - [run], peerexample_project, peer_run1, task_namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf: finished processing task
2022-04-20 12:04:45.197 - FederatedClient - INFO - Starting to push execute result.
2022-04-20 12:04:45.197 - Communicator - INFO - Send submitUpdate to example project server
2022-04-20 12:04:45.197 - FederatedServer - INFO - Received update from example project site-1 (13973504 Bytes, 1630456245 seconds)
2022-04-20 12:04:45.199 - ServerRunner - INFO - [run], wfcross_site_model_eval, peersite-1, peer_run1: got result from client for task: namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf
2022-04-20 12:04:46.018 - CrossSiteModelEval - INFO - [run], wfcross_site_model_eval, peersite-1, peer_run1, peer_run1, task_namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf: finished task validate
2022-04-20 12:04:46.018 - CrossSiteModelEval - INFO - [run], wfcross_site_model_eval, peersite-1, peer_run1, peer_run1, task_namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf: Invoking result_received_callback ...
2022-04-20 12:04:46.028 - ServerRunner - INFO - [run], wfcross_site_model_eval, peersite-1, peer_run1, peer_run1, task_namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf: Received local model from client site-1.
2022-04-20 12:04:46.028 - CrossSiteModelEval - INFO - [run], wfcross_site_model_eval, peersite-1, peer_run1, peer_run1, task_namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf: Sending site-1 model to all participating clients for validation.
2022-04-20 12:04:46.028 - CrossSiteModelEval - INFO - [run], wfcross_site_model_eval, peersite-1, peer_run1, peer_run1, task_namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf: Scheduled task validate
2022-04-20 12:04:46.028 - ServerRunner - INFO - [run], wfcross_site_model_eval, peersite-1, peer_run1, peer_run1, task_namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf: finished processing client result by cross_site_model_eval
2022-04-20 12:04:46.029 - Communicator - INFO - Received comments: example project Received from site-1 (13973504 Bytes, 1630456245 seconds). SubmitUpdate time: 0.8216440677642822 seconds
2022-04-20 12:04:46.029 - ClientRunner - INFO - [run], peerexample_project, peer_run1, task_namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf: result sent to server for task: namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf
2022-04-20 12:04:46.029 - CrossSiteModelEval - INFO - [run], wfcross_site_model_eval: task submit_model exit with status TaskCompletionException
2022-04-20 12:04:11.032 - ClientRunner - INFO - [run]: fetching task from server ...
2022-04-20 12:04:11.033 - FederatedClient - INFO - Starting to fetch execute task.
  
```

So, so we are trying to schedule a task for validation and then we are waiting for it. So, here if you see you can see that received from example project server some information.

(Refer Slide Time: 34:45)



```

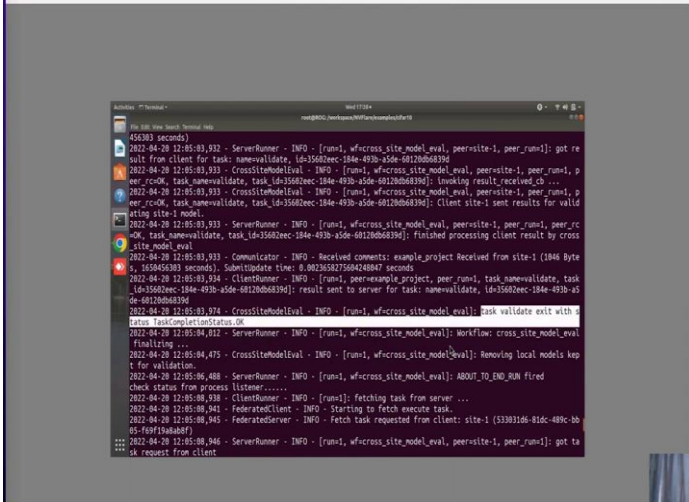
2022-04-20 12:04:45.195 - ClientRunner - INFO - [run], peerexample_project, peer_run1, task_namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf: finished processing task
2022-04-20 12:04:45.197 - FederatedClient - INFO - Starting to push execute result.
2022-04-20 12:04:45.197 - Communicator - INFO - Send submitUpdate to example project server
2022-04-20 12:04:45.197 - FederatedServer - INFO - Received update from example project site-1 (13973504 Bytes, 1630456245 seconds)
2022-04-20 12:04:45.199 - ServerRunner - INFO - [run], wfcross_site_model_eval, peersite-1, peer_run1: got result from client for task: namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf
2022-04-20 12:04:46.018 - CrossSiteModelEval - INFO - [run], wfcross_site_model_eval, peersite-1, peer_run1, peer_run1, task_namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf: finished task validate
2022-04-20 12:04:46.018 - CrossSiteModelEval - INFO - [run], wfcross_site_model_eval, peersite-1, peer_run1, peer_run1, task_namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf: Invoking result_received_callback ...
2022-04-20 12:04:46.028 - ServerRunner - INFO - [run], wfcross_site_model_eval, peersite-1, peer_run1, peer_run1, task_namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf: Received local model from client site-1.
2022-04-20 12:04:46.028 - CrossSiteModelEval - INFO - [run], wfcross_site_model_eval, peersite-1, peer_run1, peer_run1, task_namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf: Sending site-1 model to all participating clients for validation.
2022-04-20 12:04:46.028 - CrossSiteModelEval - INFO - [run], wfcross_site_model_eval, peersite-1, peer_run1, peer_run1, task_namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf: Scheduled task validate
2022-04-20 12:04:46.028 - ServerRunner - INFO - [run], wfcross_site_model_eval, peersite-1, peer_run1, peer_run1, task_namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf: finished processing client result by cross_site_model_eval
2022-04-20 12:04:46.029 - Communicator - INFO - Received comments: example project Received from site-1 (13973504 Bytes, 1630456245 seconds). SubmitUpdate time: 0.8216440677642822 seconds
2022-04-20 12:04:46.029 - ClientRunner - INFO - [run], peerexample_project, peer_run1, task_namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf: result sent to server for task: namesubmit_model, task_id=f1405d-834e-47e5-a1b8-c8575d8773cf
2022-04-20 12:04:46.029 - CrossSiteModelEval - INFO - [run], wfcross_site_model_eval: task submit_model exit with status TaskCompletionException
2022-04-20 12:04:11.032 - ClientRunner - INFO - [run]: fetching task from server ...
2022-04-20 12:04:11.033 - FederatedClient - INFO - Starting to fetch execute task.
  
```

And then you can see here starting to push the execute result right from the federated client, through the communicator and then your server responds that you have received an update from this particular site right.

So, the idea about cross site model evaluation is that everything might be a very confusing thing as to how it is automatically doing everything and all, but that should not be a concern at present. You will understand it when you basically see the documentation, but the basic idea is that since you are trying to have a global assessment of how each client is going to change your global accuracy, right.

You should ensure that the data is not of a low quality. In the sense the information which you are getting should not be such that it pulls down your global accuracy down right. So, you will have to ensure that. So, doing that requires something like cross site model evaluation right, something like this is going to happen. Then I suppose I hope you have understood this and then once that is done, task validate exited with status task completion which is ok right.

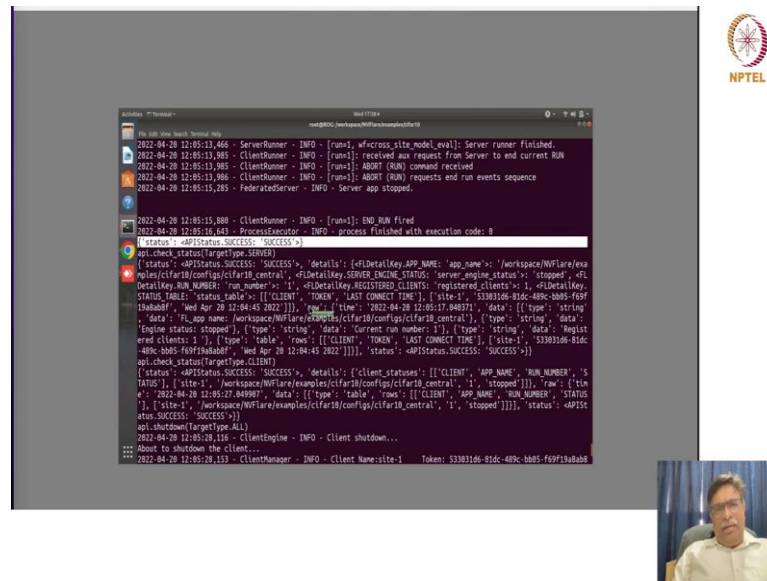
(Refer Slide Time: 36:14)



```
450.83 seconds)
2022-04-28 12:05:03.932 - ServerRunner - INFO - [runs, wf:cross_site_model_eval, peersite-1, peer_run1]: got re
sult from client for task: namevalidate, [id:35602ec-184e-493b-a5de-60120d06839d]
2022-04-28 12:05:03.933 - CrossSiteModelEval - INFO - [runs, wf:cross_site_model_eval, peersite-1, peer_run1, p
eer_rc=OK, task_namevalidate, task_id=35602ec-184e-493b-a5de-60120d06839d]: Invoking result_received() ...
2022-04-28 12:05:03.933 - CrossSiteModelEval - INFO - [runs, wf:cross_site_model_eval, peersite-1, peer_run1, p
eer_rc=OK, task_namevalidate, task_id=35602ec-184e-493b-a5de-60120d06839d]: Client site-1 sent results for valid
ating site's model
2022-04-28 12:05:03.933 - ServerRunner - INFO - [runs, wf:cross_site_model_eval, peersite-1, peer_run1, peer_rc
=OK, task_namevalidate, task_id=35602ec-184e-493b-a5de-60120d06839d]: finished processing client result by cross
site_model_eval
2022-04-28 12:05:03.933 - Communicator - INFO - Received comments: example_project Received from site-1 (1846 Byte
s, 16504560.83 seconds). SubmitUpdate time: 0.0023658275604240847 seconds
2022-04-28 12:05:03.934 - ClientRunner - INFO - [runs, peer-example_project, peer_run1, task_namevalidate, task
_id=35602ec-184e-493b-a5de-60120d06839d]: result sent to server for task: namevalidate, [id:35602ec-184e-493b-a
5de-60120d06839d]
2022-04-28 12:05:03.934 - CrossSiteModelEval - INFO - [runs, wf:cross_site_model_eval]: task validate exit with s
tatus taskCompletionStatus.OK
2022-04-28 12:05:04.012 - ServerRunner - INFO - [runs, wf:cross_site_model_eval]: Workflow: cross_site_model_eval
finalizing ...
2022-04-28 12:05:04.475 - CrossSiteModelEval - INFO - [runs, wf:cross_site_model_eval]: Removing local models kep
t for validation.
2022-04-28 12:05:05.488 - ServerRunner - INFO - [runs, wf:cross_site_model_eval]: ABOUT_TO_END_RUN fired
check status from process listener.....
2022-04-28 12:05:08.938 - ClientRunner - INFO - [runs]: fetching task from server ...
2022-04-28 12:05:08.941 - FederatedClient - INFO - Starting to fetch execute task
2022-04-28 12:05:08.945 - FederatedServer - INFO - Fetch task requested from client: site-1 (53031146-81dc-489c-bb
05-69ff19ab88f)
...
2022-04-28 12:05:08.946 - ServerRunner - INFO - [runs, wf:cross_site_model_eval, peersite-1, peer_run1]: got t
ask request from client
```

So, you have got this information about the task validation or the validation is completed right. So, yeah so this is basically like about to end run.

(Refer Slide Time: 36:45)



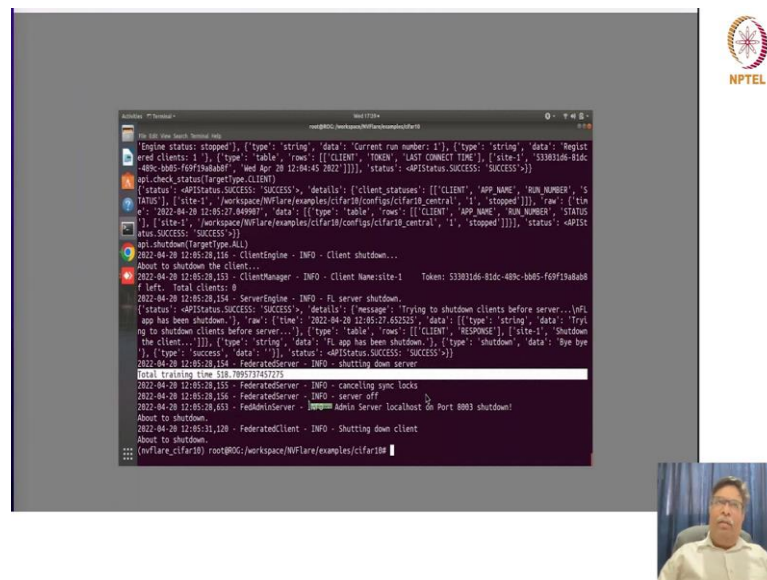
```
2022-04-20 12:05:13.466 - ServerRunner - INFO - [runs]: w/cross_site_model_eval: Server runner finished.
2022-04-20 12:05:13.985 - ClientRunner - INFO - [runs]: received aux request from Server to end current RUN
2022-04-20 12:05:13.985 - ClientRunner - INFO - [runs]: ABORT (RUN) command received
2022-04-20 12:05:13.986 - ClientRunner - INFO - [runs]: ABORT (RUN) requests end events sequence
2022-04-20 12:05:15.285 - FederatedServer - INFO - Server app stopped.

2022-04-20 12:05:15.880 - ClientRunner - INFO - [runs]: END RUN Fired
2022-04-20 12:05:16.643 - ProcessExecutor - INFO - process finished with execution code: 0

[status]: <APISuccess: SUCCESS>
api.check_status(target_type=SERVER)
[status]: <APISuccess: SUCCESS>, details: {'detailKey.APP_NAME': 'workspace/NFLare/ena
mies/cifar10/configs/cifar10_central', 'detailKey.SERVER_ENGINE_STATUS': 'server_engine_status': 'stopped', 'cl
DetailKey.RUN_NUMBER': 'run_number': '1', 'detailKey.REGISTERED_CLIENTS': 'registered_clients': '1', 'detailKey.
STATUS_TABLE': 'status_table': '[{"CLIENT": "TOKEN", "LAST_CONNECT_TIME": [{"site": "53303106-81dc-489c-bb05-f69f19abab
19abab"}], "time": "2022-04-20 12:05:17.040271", "data": [{"type": "string",
"data": "FL app name: workspace/NFLare/cifar10/configs/cifar10_central"}, {"type": "string", "data":
"Engine status: stopped"}, {"type": "string", "data": "Current run number: 1"}, {"type": "string", "data": "Regist
ered clients: 1"}, {"type": "table", "rows": [{"CLIENT": "TOKEN", "LAST_CONNECT_TIME": [{"site": "53303106-81dc-
489c-bb05-f69f19abab"}], "time": "2022-04-20 12:05:17.040271"}]}], 'status': <APISuccess: SUCCESS>}}
api.check_status(target_type=CLIENT)
[status]: <APISuccess: SUCCESS>, details: {'client_statuses': [{"CLIENT": "APP_NAME", "RUN_NUMBER", "S
TATUS"}], 'site': 'workspace/NFLare/examples/cifar10/configs/cifar10_central', '1', 'stopped'}], 'raw': {'tim
e': '2022-04-20 12:05:27.049987', 'data': [{"type": "table", "rows": [{"CLIENT": "APP_NAME", "RUN_NUMBER", "STATUS
"}, {"site": "workspace/NFLare/examples/cifar10/configs/cifar10_central", "1", "stopped"}]}], 'status': <APIS
tatus: SUCCESS>}}
api.shutdown(target_type=ALL)
About to shutdown the client.
2022-04-20 12:05:28.116 - ClientEngine - INFO - Client shutdown...
2022-04-20 12:05:28.153 - ClientManager - INFO - Client Name:site-1 Token: 53303106-81dc-489c-bb05-f69f19abab
Left: Total Clients: 0
2022-04-20 12:05:28.154 - ServerEngine - INFO - FL server shutdown.
[status]: <APISuccess: SUCCESS>, details: {'message': 'Trying to shutdown clients before server...[n]
app has been shutdown.', 'raw': {'time': '2022-04-20 12:05:27.652525', 'data': [{"type": "string", "data": "Tryi
ng to shutdown clients before server..."}, {"type": "table", "rows": [{"CLIENT": "RESPONSE"}, {"site": "1", "shutdow
n the client..."}], "type": "string", "data": "FL app has been shutdown."}, {"type": "shutdown", "data": "Bye bye
"}, {"type": "success", "data": ""}], 'status': <APISuccess: SUCCESS>}}
2022-04-20 12:05:28.154 - FederatedServer - INFO - Shutting down server
Total training time 518.709573457225
2022-04-20 12:05:28.155 - FederatedServer - INFO - canceling sync locks
2022-04-20 12:05:28.156 - FederatedServer - INFO - server off
2022-04-20 12:05:28.653 - FedMiniserver - Admin Server Localhost On Port 8003 shutdown!
About to shutdown.
2022-04-20 12:05:28.120 - FederatedClient - INFO - Shutting down client
About to shutdown.
(workspace/NFLare/examples/cifar10) root@BGC/workspace/NFLare/examples/cifar10
```

So, you are going to do this, so end run fire yeah. So, process finished with execution code 0. So, we are having a status which shows that it is a success right.

(Refer Slide Time: 36:58)



```
2022-04-20 12:05:28.154 - ServerEngine - INFO - FL server shutdown.
[status]: <APISuccess: SUCCESS>, details: {'message': 'Trying to shutdown clients before server...[n]
app has been shutdown.', 'raw': {'time': '2022-04-20 12:05:27.652525', 'data': [{"type": "string", "data": "Tryi
ng to shutdown clients before server..."}, {"type": "table", "rows": [{"CLIENT": "RESPONSE"}, {"site": "1", "shutdow
n the client..."}], "type": "string", "data": "FL app has been shutdown."}, {"type": "shutdown", "data": "Bye bye
"}, {"type": "success", "data": ""}], 'status': <APISuccess: SUCCESS>}}
2022-04-20 12:05:28.154 - FederatedServer - INFO - Shutting down server
Total training time 518.709573457225
2022-04-20 12:05:28.155 - FederatedServer - INFO - canceling sync locks
2022-04-20 12:05:28.156 - FederatedServer - INFO - server off
2022-04-20 12:05:28.653 - FedMiniserver - Admin Server Localhost On Port 8003 shutdown!
About to shutdown.
2022-04-20 12:05:28.120 - FederatedClient - INFO - Shutting down client
About to shutdown.
(workspace/NFLare/examples/cifar10) root@BGC/workspace/NFLare/examples/cifar10
```

So, once it is success, we can actually shut down our local what to say our server's right and then try to analyze this particular thing now ok yeah.

So, if you see this analysis here, when you are trying to shut down the server your total training time is 518.709 ok seconds. So, this is basically a model which is trained using federated learning. So, we wanted to show you this, if the data is split there is 1 client

and 1 server and then you train this using this ok. So, this is what we wanted to show you in this example, this is a proof of concept which basically tells you that federated learning ok can train your models effectively.