GSoC Project Ideas List

Algorithm API Projects

A major theme in OpenMined is the development of APIs around privacy and machine learning algorithms to make them easy to use. These tend to be slightly more advanced than some other projects, because it requires a high level understanding of what one seeks to accomplish with an algorithm with the goal of minimizing the amount of knowledge the end user must have in order to use it.

For example, we have wrapped the Paillier Homomorphic Encryption system into a tensor abstraction. This allows users of PyTorch tensors to be able to leverage Paillier Homomorphic encryption without having to understand how the encryption system itself works.

More advanced projects in this category involve re-implementing the algorithm yourself, while less advanced projects in this category involve wrapping existing implementations.

Project: Combine Federated Learning & SplitNN APIs

Federated Learning and SpitNN are both methods for distributing the training of a neural network across multiple machines for the sake of privacy. However, they have different tradeoffs. At present, one can either choose to use one algorithm or the other, but not a combination of both. This project seeks to remedy this by creatively combining them in a clean, cohesive API.

Required Skills:

- Knowledge of PyTorch and Deep Learning
- Familiarity (or willingness to become familiar) with PySyft
- Ability to understand Federated Learning
- Ability to understand SplitNN

Difficulty: Medium
Github Issue Link: https://github.com/OpenMined/PySyft/issues/3111

While the project itself is not complex (neither Federated Learning or SplitNN are hard to understand), this project does require some significant API work within PySyft (which is, itself, a complex library).

Project: Develop an Incident Response Model for Privacy Preserving Machine Learning

OpenMined provides a vast array of tools to enable privacy-preserving machine-learning. However, in order to bridge the gap between industry and this new technology, we need to supply governance models which are relevant to Privacy-Preserving Machine Learning. This project in particular will be focusing on recommending an incident response model.

Required Skills:

- Knowledge of Information Security Management
- Familiarity with PySyft, PyGrid and distributed learning
- Ability to understand privacy-preserving ML workflows, the methods by which they can be subverted and how to mitigate these methods.
- (Desirable) Certified Information Security Auditor (ISACA etc.)

Difficulty: Medium
Github Issue Link: https://github.com/OpenMined/Roadmap/issues/11

While the project itself is not complex (the deliverables here will be mostly documentation), this project does require background knowledge in PPML, IS management, security audit and compliance.

Project: Implement FV Homomorphic Encryption from Scratch

FV (Fan-Vercauteren) Homomorphic Encryption is one of the leading approaches to homomorphic encryption. While current work is under way within our community to wrap the SEAL homomorphic encryption library, this will have limited portability. However, algorithms which are natively implemented in PySyft can be automatically run across Python, GPUs, Javascript, Android (Kotlin), and iOS (Swift). Thus, it is very advantageous for us to re-implement FV encryption from scratch.

The goal of this project will be to create a new tensor type which allows one to leverage FV to perform encrypted tensor operations.

Required Skills:

- Knowledge of PyTorch and Deep Learning
- Familiarity (or willingness to become familiar) with PySyft
- Ability to either implement the FV algorithm from the paper, or ability to port an existing C++ implementation (there are many)

Difficulty: Advanced
Github Issue Link: https://github.com/OpenMined/PySyft/issues/3097

While the API aspects of this project are not complex (we have straightforward instructions for how to implement new tensor types), the FV algorithm is non-trivial.

Project: Implement BGV Homomorphic Encryption from Scratch

BGV (Brakerski-Gentry-Vaikuntanathan) Homomorphic Encryption is one of the leading approaches to homomorphic encryption. While current work is under way within our community to wrap the SEAL homomorphic encryption library, this will have limited portability. However, algorithms which are natively implemented in PySyft can be automatically run across Python, GPUs, Javascript, Android (Kotlin), and iOS (Swift). Thus, it is very advantageous for us to re-implement BGV encryption from scratch.

The goal of this project will be to create a new tensor type which allows one to leverage BGV to perform encrypted tensor operations.

Required Skills:

- Knowledge of PyTorch and Deep Learning
- Familiarity (or willingness to become familiar) with PySyft
- Ability to either implement the BGV algorithm from the paper, or ability to port an existing C++ implementation (there are many)

Difficulty: Advanced
Github Issue Link: https://github.com/OpenMined/PySyft/issues/3098

While the API aspects of this project are not complex (we have straightforward instructions for how to implement new tensor types), the BGV algorithm is non-trivial.

Project: Wrap Open-License Zero-Knowledge Proof Library

Zero-knowledge proofs have an important role to play in the future of verified machine learning prediction. However, no deep learning framework has the ability to perform verified computation of neural networks using ZKPs. In this project, you will wrap an existing ZKP library into a new tensor type allowing users of PySyft to generate, evaluate, and verify tensor computations.

Required Skills:

- Knowledge of PyTorch and Deep Learning
- Familiarity (or willingness to become familiar) with PySyft
- Ability to wrap an existing Python, C++, Rust, or Javascript library within Python

Difficulty: Easy
Github Issue Link: https://github.com/OpenMined/PySyft/issues/3104

There are many libraries to choose from, some of which are already in Python.

Project: Implement ZKP From Scratch in PyTorch

Zero-knowledge proofs have an important role to play in the future of verified machine learning prediction. However, no deep learning framework has the ability to perform verified computation of neural networks using ZKPs.

Furthermore, algorithms which are natively implemented in PySyft can be automatically run across Python, GPUs, Javascript, Android (Kotlin), and iOS (Swift). Thus, it is very advantageous for us to re-implement ZKP encryption from scratch using PyTorch tensor operations.

In this project, you will implement from scratch (from the paper) or port an existing ZKP library into PyTorch operations. This will allow us to generate, evaluate, and verify ZKP graphs across all platforms supported by PySyft.

Required Skills:

- Knowledge of PyTorch and Deep Learning
- Familiarity (or willingness to become familiar) with PySyft
- Ability to understand the mathematics of ZKPs and/or current libraries implementing them

Difficulty: Advanced
Github Issue Link: https://github.com/OpenMined/PySyft/issues/3100

While the API shouldn't be complex, ZKP algorithms are challenging to understand and implement. There are, however, many great implementations to use as references.

Project: Transfer Learning

Transfer learning help in transferring knowledge from a source domain to a different target domain. In NLP, Transfer learning has become ubiquitous as a way of reusing pre-trained language models and fine tuning them to a target task like Text Classification or Named Entity Recognition.
The goal of this project is to apply Transfer Learning to a private dataset using SyferText. An example would be to train a BERT-based model for text auto-completion and personalized per user.

Required Skills:

- Experience with NLP (e.g. Spacy, NTLK)
- Experience with PyTorch
- Familiarity (or willingness to become familiar) with SyferText
- Familiarity (or willingness to become familiar) with PySyft

Difficulty: Medium
Github Issue Link: https://github.com/OpenMined/SyferText/issues/16

This project should not be too complicated, but requires some exploratory work.

Project: Implement Functional Encryption in PySyft

Functional Encryption is a technique which relates to Homomorphic Encryption since it performs computations over encrypted data, but it also provides automatic decryption of the ciphered result, which allows for interesting non-interactive scenarios. Put formally, given a function f and an encrypted value Enc(x), it outputs f(x). One classical example of Functional Encryption is spam filtering: Alice wants to send a confidential email to Bob, and the email server should analyse whether the email is spam without reading it before forwarding it or not to Bob.

This project will consist of re-implementing the Functional Encryption algorithm proposed in this Neurips 2019 paper. It relies on a crypto c++ based library named charm and introduces some crypto primitives that are hard to re-implement from scratch in Python. Therefore, the main focus of this project will be first to provide a user-friendly Tensor abstraction for Functional Encryption in PySyft, and make sure having charm as a dependency is not burdensome. Re-implementing those primitives from scratch will be a second (challenging) part for people with more crypto background.

Context
I'm the author of the paper linked above and have already provided elements to perform the implementation in this repo. I'm also leading the Crypto Team, whose roadmap is available here and I'm very keen to discuss alternative Function Encryption schemes if you have ideas!

Required Skills:

- Knowledge of Deep Learning (TensorFlow, PyTorch or JAX)
- Familiarity (or willingness to become familiar) with PySyft
- Familiarity with Linear Algebra, to understand the mathematics of Functional Encryption.
- Autonomy and willingness to take proactive steps

Difficulty: Intermediate to Advanced
Github Issue Link: https://github.com/OpenMined/PySyft/issues/3108

To be clear, this project is not about cleaning my first implementations! The main challenge is that FE currently allows only quadratic operations: for example x + 2, x * 3, x * y, 2(x-1)*y + 1 are valid operations but x*x*y is not. Therefore, the computations written by the end user should not be executed eagerly. Instead they should be buffered and organized to fit as a quadratic computation when possible. Using PySyft Plans is an excellent way of doing this. When high orders computations will be required, we will also consider the following trick which consists of encrypting higher orders monomials (for example encrypting xy alongside x and y allows to compute in a quadratic way x*(x*y))

Providing an easy-to-use, robust and flexible interface will be a guarantee of success for this project.

Platform / Cloud Integration Projects

Critical to our roadmap is the ability to easily spin up/down cloud machines running PyGrid servers. These projects implement such functionality.

Project: Implement Auto-Scaling on Google Cloud

In this project, you will implement functionality necessary to automatically spin-up Google cloud machines, load a PyGrid instance, run a training job, and tear down the instance upon completion (depositing the results into another long-running instance). The primary feature will be the ability to run a “hyperparameter sweep”, as exemplified below.

```
Parameters = {“alpha” : [0, 0.01, 0.02], “batch_size” : [32,64]}

gcloud = gr.GoogleCloud(credentials)

cluster = gcloud.LazyCluster(n_machines=10,
                             type="n-series",
                             priority="low",
                            eviction_policy="Delete",
                            max_price_usd=0.55,
                            reboot_max_price_usd=0.65)

cluster.sweep(model=model,
              parameters=parameters,
              optim=optim.SGD,
              results_node=grid['my node']
              )
```

Required Skills:

- Knowledge of PyTorch and Deep Learning
- Familiarity (or willingness to become familiar) with PySyft
- Familiarity (or willingness to become familiar) with Google Cloud’s Infrastructure

Level: Beginner
Github Issue Link: https://github.com/OpenMined/PySyft/issues/3099

While this is a sizeable project, the core functionality is relatively straightforward (provision machines, install and start PyGrid servers, and run training scripts). Furthermore, there are plenty of tutorials on how to run PyGrid servers and how to run AWS Cloud machines.

Project: Implement Auto-Scaling on Azure Cloud

In this project, you will implement functionality necessary to automatically spin-up Azure cloud machines, load a PyGrid instance, run a training job, and tear down the instance upon completion (depositing the results into another long-running instance). The primary feature will be the ability to run a "hyperparameter sweep", as exemplified below.

```
Parameters = {"alpha" : [0, 0.01, 0.02], "batch_size" : [32,64]}

azure = gr.AzureCloud(credentials)

cluster = azure.LazyCluster(n_machines=10,
                            type="n-series",
                            priority="low",
                           eviction_policy="Delete",
                           max_price_usd=0.55,
                           reboot_max_price_usd=0.65)

cluster.sweep(model=model,
              parameters=parameters,
              optim=optim.SGD,
              results_node=grid['my node']
              )
```

Required Skills:

- Knowledge of PyTorch and Deep Learning
- Familiarity (or willingness to become familiar) with PySyft
- Familiarity (or willingness to become familiar) with Azure's Cloud Infrastructure

Level: Beginner
Github Issue Link: Coming soon!

While this is a sizeable project, the core functionality is relatively straightforward (provision machines, install and start PyGrid servers, and run training scripts). Furthermore, there are plenty of tutorials on how to run PyGrid servers and how to run Azure Cloud machines.

Project: Implement Auto-Scaling on Amazon Web Services

In this project, you will implement functionality necessary to automatically spin-up AWS cloud machines, load a PyGrid instance, run a training job, and tear down the instance upon

completion (depositing the results into another long-running instance). The primary feature will be the ability to run a “hyperparameter sweep”, as exemplified below.

```
Parameters = {“alpha” : [0, 0.01, 0.02], “batch_size” : [32,64]}

aws = gr.AWSCloud(credentials)

cluster = aws.LazyCluster(n_machines=10,
                          type="n-series",
                          priority="low",
                         eviction_policy="Delete",
                         max_price_usd=0.55,
                         reboot_max_price_usd=0.65)

cluster.sweep(model=model,
              parameters=parameters,
              optim=optim.SGD,
              results_node=grid['my node']
              )
```

Required Skills:

- Knowledge of PyTorch and Deep Learning
- Familiarity (or willingness to become familiar) with PySyft
- Familiarity (or willingness to become familiar) with AWS’s Cloud Infrastructure

Level: Beginner
Github Issue Link: Coming soon!

While this is a sizeable project, the core functionality is relatively straightforward (provision machines, install and start PyGrid servers, and run training scripts). Furthermore, there are plenty of tutorials on how to run PyGrid servers and how to run AWS Cloud machines.

Project: Replacing PyGrid Transport with DIDComm using Hyperledger Aries

Hyperledger Aries is an open-source reference architecture for developing self-sovereign identity agents of which there are many implementations under development, although the python one is probably the most relevant. As a first step towards understanding how these agents, could be combined with the OM stack to provide secure communication channels for

distributed ML, it would be valuable to develop a docker image combining PyGrid and a Hyperledger Aries agent. Then attempting to replace the transport protocol, currently a flask http server, with the agent to agent communication protocol called DIDComm. To test this works and develop understanding of how Aries and OM might be combined, this PoC could be reimplemented using the new docker image.

Required Skills:

- Experience with Docker
- Experience with Dev Ops
- Familiarity (or willingness to become familiar) with Hyperledger Aries
- Familiarity (or willingness to become familiar) with PyGrid

Difficulty: Medium
Github Issue Link: https://github.com/OpenMined/PyGrid/issues/493/

This project should not be too complicated, although it may include some work to get compatible dependencies across both projects. Exploring the correct flow to replace the current messaging in PyGrid with DIDComm may require some thought.

Project: Threepio Support Dashboard

Threepio supports translation between a number of different languages and libraries. It will be some time before we have total support between all API commands. In order for users to confidently use our worker libraries, they will have to understand what commands are supported.

Project Scope:

- Building a simple OpenMined branded web application to display statistics & API endpoint support
- Expanding any needed documentation crawler functionality to get additional relevant information on API endpoints.
- Building integrations into our existing translation libraries to take into account for custom translations.
- Integrating dashboard with CI to allow for continuous deployment.

Required Skills:

- Front end development skills
- Javascript
- CSS
- Python & scrapy

- Basic layout design

Difficulty: Beginner
Github Issue Link: https://github.com/OpenMined/threepio/issues/33

This project will be perfect for someone with a web development background to get to touch a lot of different machine learning libraries without needing significant data science experience. Furthermore it will greatly enable future contributors to both OpenMined as well as a number of major machine learning libraries.

Project: Defining a OM/ML Message Type within Hyperledger Aries

Hyperledger Aries is an open-source reference architecture for developing self-sovereign identity agents of which there are many implementations under development, although the python one is probably the most relevant. Aries agents send asynchronous messages using a protocol called DIDComm, these messages have a type which defines the structure of the message. Each message type has it's own handler for implementing agent specific responses to these messages. Currently the message type Basic Message, is implemented in most agent implementations. We developed a PoC which hijacked this basic messaging feature to send ML messages in a federated learning example. A better implementation of this would be to define a specific message type for distributed learning communication - some exploration would be needed to understand the exact requirements of this message type. For example should there be one message type for each distributed learning implementation, or should type be in the data structure of the message?

Required Skills:
- Willing to explore a complex code base
- Familiarity (or willingness to become familiar) with Hyperledger Aries
- Familiarity (or willingness to become familiar) with PySyft
- Familiarity (or willingness to become familiar) with PyGrid

Difficulty: Medium
Github Issue Link: https://github.com/OpenMined/PyGrid/issues/495

This project should be fairly straightforward, although an initial learning curve to get to grips with the aries architecture will be required. A best case scenario would be a ML Message Type ratified by the Hyperledger Aries community and included as an RFC similar to the Basic Message and implemented as a reference in one of the aries implementations - or a fork of one of these projects.

Project: OpenMined-Android

We are developing a Community App that will have the feature and information about the OpenMined Community and the It's Projects. In this project, you will be implementing from scratch and will be working on the project adding more features.

Required Skills:

- Knowledge of Android Development with Kotlin
- Knowledge of Android Basics, Android UI, Data management, Firebase etc
- Familiarity (or willingness to become familiar) with OpenMined Projects and OpenMined Community
- You are highly motivated and want to get involved in the project on a regular basis during the GSoC program

Difficulty: Medium
Github Issue Link: https://github.com/OpenMined/Roadmap/issues/10

The main goal of this project is to make even more accessible to the community members to navigate through the organization & to get updates from the OpenMined Android App

---

Project: Framework agnostic support for PySyft - TensorFlow

PySyft functionalities are mostly built on top of Pytorch, but ideally, PySyft should be completely framework agnostic. This meaning that all operations executed with PySyft should not depend on the framework that the actual low-level operations are being executed with.

In this project, you will be improving/implementing support for PySyft ops within TensorFlow.

These can be summarised in 3 main areas based on how PySyft is structured:

- **Support TensorFlow ops and tensors:** this is where most of the work is needed and consists of wrapping TensorFlow so it can be used with PySyft tensors and operations.
- **Remove execution with TensorFlow:** supporting remote operations of executing on different workers (Virtual or not). This should be more or less straightforward since this should be the same for all frameworks, maybe some work is needed around TF Tensor serialization
- **Privacy related ops:** making any changes in the codebase needed in order to have SMPC, Differential Privacy, SplitNN, etc. working with TensorFlow. Not clear right now how much work is needed in practice to get this working, but in theory not a lot.

Completion criteria:
PySyft Tutorials 1 - 4 running seamlessly on both TensorFlow and Pytorch. If more time available continue supporting operations for other tutorials.

Development timeline

This is a summarized draft to help you with your proposal, you're free to change this as you'd like!

1. Support simple TensorFlow operations (e.g. tf.add, tf.matmul, tf.nn.relu, …) and basic Tensor support.
2. Support basic remote execution ops with **pointers** (**send, move**, **get**). At this point the tutorial 1 , tutorial 3 should be executing successfully with TensorFlow.
3. Support training logic with TensorFlow (i.e. backprop, optimizers). At this point the tutorial 2 and tutorial 4 should be ready, notice that this will potentially be very different from the existing Pytorch tutorial due to framework differences (we can/should discuss this when implementing).
4. You did it :D!!!

Extra

5. Federated Dataset support: should be more or less straightforward :). tutorial 6 should be ready.
6. Grid and dataset support. tutorial 5 and tutorial 7 should be ready.
7. Plans and protocols support: this is potentially non-trivial, so we can/should discuss this when implementing. tutorial 8 and tutorial 8 bis ready.
8. Implement SMPC support. tutorial 9 ready.
9. SMPC support during training / inference. tutorials 10, 11 and 12 are ready.
10. You did it again :D!!!

Required Skills:

- Knowledge of TensorFlow & Python
- Familiarity (or willingness to become familiar) with PySyft
- Familiarity (or willingness to become familiar) with OpenMined Projects and OpenMined Community

Difficulty: Medium
Github Issue Link: https://github.com/OpenMined/PySyft/issues/3160

Optional Projects

Optional Project: Implement the command translation layer inside of PySyft which will allow for PySyft Tensorflow plans to be converted to PyTorch

It's worth noting that this project is considered optional "extra credit" for the Federated Learning roadmap:
https://github.com/OpenMined/Roadmap/blob/master/web_and_mobile_team/projects/web_and_mobile_fl.md

Problem:
Essentially we'll already have a translation function that exists for Javascript - this project simply brings the same logic to Python. This may not end up serving a permanent purpose in PySyft, but we think that this would allow for a developer to create a plan using PySyft for TensorFlow and still have training available for mobile phones. This would be an incredible achievement since currently we're scoped to only allow the training of PyTorch models on mobile devices (since they require the use of TorchScript). If a developer wanted to write their PySyft model using TensorFlow, they would then need to convert this plan to use PyTorch, before ultimately hosting the plan on PyGrid as a TorchScript.

Solution:
We are building a command translation library for the syft.js web worker library called "Threepio". This project will need to be extended to support TensorFlow (Python) bindings. By this point, it should already have bindings for PyTorch and TensorFlow.js.

Requirements:

- Willing to work with the Web & Mobile team and be mentored by @Nolski, the author of Threepio
- Familiarity with multiple deep learning frameworks: PyTorch, TensorFlow, and TensorFlow.js (NumPy is a bonus!)
- Good knowledge and understanding of the problems and challenges described in the Federated Learning roadmap (questions can be directed to @cereallarceny)

Difficulty: Medium
Github Issue Link: https://github.com/OpenMined/PySyft/issues/2997

Optional Project: Implement changes in PySyft client for federated learning

Based on the Federated Learning roadmap we need to build a worker library for static federated learning in PySyft. The worker will be based on the other 3 worker libraries that already exist: syft.js (web), SwiftSyft (iOS), and KotlinSyft (Android). The API will be a mirror image of the other federated learning worker libraries, allowing for environments other than a web browser or mobile device to be supported: IoT, Raspberry Pi, desktop application, Windows phone, etc.

Requirements:

- Willing to work with the Web & Mobile team and be mentored by @vkkhare, the author of KotlinSyft
- Familiarity with Websockets
- Familiarity with WebRTC
- Good knowledge of PySyft
- Intimate knowledge of Python
- History of developing libraries (in any language)
- Good knowledge and understanding of the problems and challenges described in the Federated Learning roadmap (questions can be directed to @cereallarceny)

Difficulty: High
Github Issue Link: https://github.com/OpenMined/PySyft/issues/2996

This is quite a large project and would more or less allow for static federated learning to be performed in any environment. A federated learning system of this breadth has not previously been developed, anywhere. Make no mistake, this is a big project, but it also has very far-reaching ramifications for federated learning as a whole. Fortunately, the web and mobile team has already built 3 federated learning worker libraries in the past, so you'll be inheriting the wisdom, patterns, and roadmap that has basically been finalized by your new colleagues. If you're not afraid of hard projects with a pretty large payoff, this is the one for you.

Optional Project: Add "test_federated_training" to PySyft

The `test_federated_training` function will allow a user to spin up a series of VirtualWorkers with which they run against their FL model. This allows them to locally simulate the process of deploying their model to the edge. The developer will also want to define their initial model parameters (e.g. model name, version, hyperparameters, etc.) and federated learning configuration (e.g. the number of cycles, maximum number of workers, etc.).

This process will create a network of VirtualWorkers in PySyft, send the plans to execute to each of them, execute the model on each of the VirtualWorkers, run any protocols for secure aggregation, push the results back to PySyft, run the averaging plan to update the global model, and finally return the model back to the developer for inspection.

```
# These attributes should correspond to inputs in the training_plan
client_config = {
  name: "my-federated-model",
  version: "0.1.0",
  batch_size: 32,
  lr: 0.01,
  optimizer: "SGD"
}

server_config = {
  max_workers: 100,
  pool_selection: "random",  # or "iterate"
  num_cycles: 5,
  do_not_reuse_workers_until_cycle: 4,
  cycle_length: 8 * 60 * 60,  # 8 hours
  minimum_upload_speed: 2000,  # 2 mbps
  minimum_download_speed: 4000  # 4 mbps
}

sy.test_federated_training(
  model=model,
  client_plans={ "training_plan": training_plan },
  client_protocols={ "secure_agg_protocol":
secure_aggregation_protocol },
  client_config=client_config,
  server_averaging_plan=averaging_plan,
  server_config=server_config
)
```

Requirements:

- Willing to work with the Web & Mobile team and be mentored by @mccorby, the author of KotlinSyft
- Very good knowledge and understanding of PySyft
- Intimate knowledge of Python

- Good knowledge and understanding of the problems and challenges described in the Federated Learning roadmap (questions can be directed to @cereallarceny)

Difficulty: Medium

Github Issue Link: https://github.com/OpenMined/PySyft/issues/2995

Optional Project: Develop end-to-end testing suite for all full system testing in Federated Learning

This issue doesn't need to live in syft-proto, but it must exist as some sort of a Github issue somewhere. This project is probably the best central location for it for now.

This issue describes how we will need to develop a testing suite that can run span multiple repositories, languages, and environments. We want to ensure that our worker libraries (syft.js, KotlinSyft, and SwiftSyft) are always working properly with PySyft and PyGrid. Likewise, we'll need to test this on various versions of syft-proto and Threepio to ensure that our assisting libraries pair nicely with the other various projects. We've created a ticket for setting up a CI system - this is a ticket for actually getting some sort of end-to-end testing framework working between all 7 repositories in question.

We have already converted all repositories inside of the OpenMined ecosystem (at least all the ones involved with federated learning) to use Github Actions instead of TravisCI. Working with Github Actions and workflows will be the basis of this issue.

Requirements:

- Willing to work with the Web & Mobile team and be mentored by @vvmnnnkv, the author of syft.js
- Intimate knowledge and understanding of the problems and challenges described in the Federated Learning roadmap (questions can be directed to @cereallarceny)
- Good knowledge of syft.js, KotlinSyft, SwiftSyft, syft-proto, PySyft, PyGrid, and Threepio... primarily as an understanding of knowing how these systems relate and work together to accomplish federated learning
- Intimate knowledge of GitOps: actions, workflows, and projects

Difficulty: Medium

Github Issue Link: https://github.com/OpenMined/syft-proto/issues/37

Optional Project: Allow for periodic status updates of current FL cycles on PyGrid

A model developer may want to inspect the progress of the current federated learning cycle in progress, as well as potentially see how previous cycles have gone. We should allow for the creation of an inspect_federated_training() method (or whatever name is appropriate) to inspect the current or former progress of FL cycles in PyGrid.

Github Issue Link: https://github.com/OpenMined/PyGrid/issues/441
