frontiers in Blockchain

METHODS
published: 12 July 2021
doi: 10.3389/fbloc.2021.676159

iChain: Peer-To-Peer Machine Learning Powered by Blockchain Technology

*Caleb Johnson, Tao Lu, Pedro Rivera, Devon McDonald, Samuel Pritchett and Lu Peng**

Division of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA, United States

iChain is an application which was created to help meet the growing demand of machine learning. It allows users to pay those with powerful machines to run machine learning tasks for them, bypassing the need for a significant investment in a powerful computer to run it themselves. This is similar to services like a render farm. Our application functions using the Ethereum blockchain which ensures security and decentralization, as well as providing a platform for payment transactions. This article will discuss the background on machine learning and blockchain, the application, how it works, how the data moves through it, and how to use it. We hope our application will enable many without the funds to build or buy a powerful computer to experiment with and utilize complex machine learning tasks.

Keywords: blockchain, ethereum, peer-to-peer, machine learning, transactions

OPEN ACCESS

Edited by:
Joerg Osterrieder, Zurich University of Applied Sciences, Switzerland

Reviewed by:
Luca Di Persio, University of Verona, Italy
Nadia C. Fabrizio, CEFRIEL, Italy

****Correspondence:***
Lu Peng
lpeng@lsu.edu

Specialty section:
This article was submitted to Financial Blockchain, a section of the journal Frontiers in Blockchain

Received: *31 March 2021*
Accepted: *28 June 2021*
Published: *12 July 2021*

Citation:
Johnson C, Lu T, Rivera P, McDonald D, Pritchett S and Peng L (2021) iChain: Peer-To-Peer Machine Learning Powered by Blockchain Technology. Front. Blockchain 4:676159. doi: 10.3389/fbloc.2021.676159

THE DEMAND FOR MACHINE LEARNING AND BLOCKCHAIN

Blockchain is a term most have heard in relation to cryptocurrency, but there are many misconceptions on what blockchain actually is. As shown in **Figure 1**, blockchain is a "distributed ledger" or an append-only database which is stored on many different computers called nodes. Whenever one node wants to append some entry to this database, a certain percentage of nodes must agree it's valid. If they do, the change will be propagated across the network (Mearian, 2019).

In the public eye blockchain is largely associated with cryptocurrency, but blockchains with no cryptocurrency component exist, such as Hyperledger by IBM (Zand, 2019), and many blockchains do not have currency as their main purpose. Bitcoin is the first and most basic blockchain, which is also famously a cryptocurrency (Parkens, 2015) (Nakamoto, 2009). The distributed and trustless nature of a blockchain makes it well served for keeping track of transactions securely.

Ethereum is notable for being one of the first blockchains to implement smart contracts (Reiff, 2020). This is code that is stored on the blockchain and executed by nodes on the chain. To run the code, the nodes that perform it must be paid in Ethereum's cryptocurrency: ether. There are different languages, the most popular being Solidity, that are compiled to an assembly language which runs on the EVM (Ethereum Virtual Machine) (Buterin, 2013).

Machine learning (ML) has been widely used in many businesses, such as finance, healthcare, transportation etc. According to a recent industry report, the global machine learning market size is currently $1.58 billion in 2017 and will grow to $20.83 billion by 2024 (Columbus, 2020). As the market boosts, the demand for computation in machine learning is increasing sharply (Dinh et al., 2017; Di Persio and Honchar, 2018; Dugyala et al., 2018; Rahouti et al., 2018). Machine learning training is usually highly compute-intensive, time consuming and has a high requirement on pricy devices (Singh, 2019). In order to develop machine learning application in a productive pace, users need to acquire high performance devices such as high-end GPUs. This approach increases the

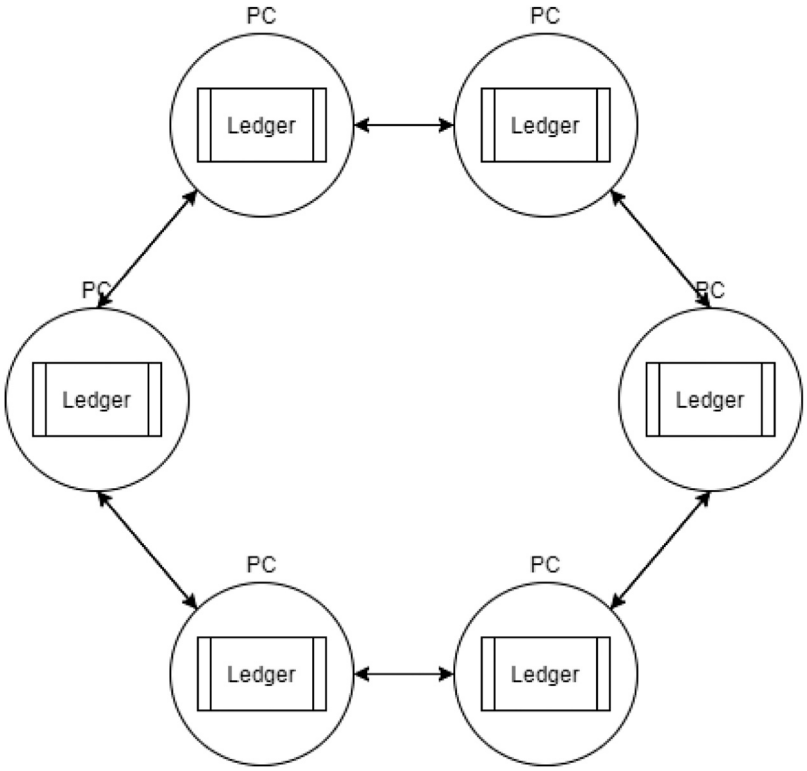

FIGURE 1 | Blockchain diagram.

financial burden of the project significantly. Alternatively, they can rent a node from cloud computing service provider such as Microsoft Azure, Amazon AWS, Google Cloud etc. However, these current solutions are all running in a centralized manner and facing the potential risk of data leakage and single point failure. Therefore, a reliable, cheap and private machine learning platform or infrastructure has been long desired by the industry.

Inspired by sharing economy, we propose a Peer-to-Peer (P2P) computation resource sharing system *iChain*, which connects the demand and supply of machine learning computing platforms. Anyone who is willing to share their computation devices can trade their computation power for cryptocurrencies. Meanwhile, anyone who has a machine learning task to do can post that task to the platform and get it done by using devices from others. The provider will then be paid in Ethereum cryptocurrency for their work. It was created for Ubuntu but it should run on any version of Linux, though the installation script may not work and installation will need to be done manually.

THE APPLICATION'S ARCHITECTURE AND DATAFLOW

First we will discuss our choice of Ethereum for this application. Ethereum was chosen primarily for its versatility and robustness allowed in the execution of smart contracts, its relative popularity to other smart-contract supporting blockchains in order to increase its audience and familiarity, and its ability to easily make payments between users (as opposed to a blockchain featuring smart contracts like Hyperledger which does not feature a currency component with which to make payments). The upcoming transition of Ethereum from Proof of Work to Proof of Stake, and with it the tokenization of the ETH currency, must also be considered. However, the tokenization should bring positive effects, making the currency more valuable and increasing its versatility even further (Ogilvie, 2020).

This application functions using a Node.js backend. This backend can be interacted with either through a CLI (Command Line Interface) or a web page. There is also a Python script, which runs for file transfer and task execution. The Node.js backend uses the popular web3 library to connect to the Ethereum blockchain in order to make transactions. This layout can be seen in **Figure 2**. **Figure 3** shows a different view which considers the distribution of the functional subsystems. This figure helps illustrate which actions each subsystem performs.

The mission statement of this project is to allow users who want to run complex machine learning tasks but who may not have a powerful computer to pay someone else to do it. This is

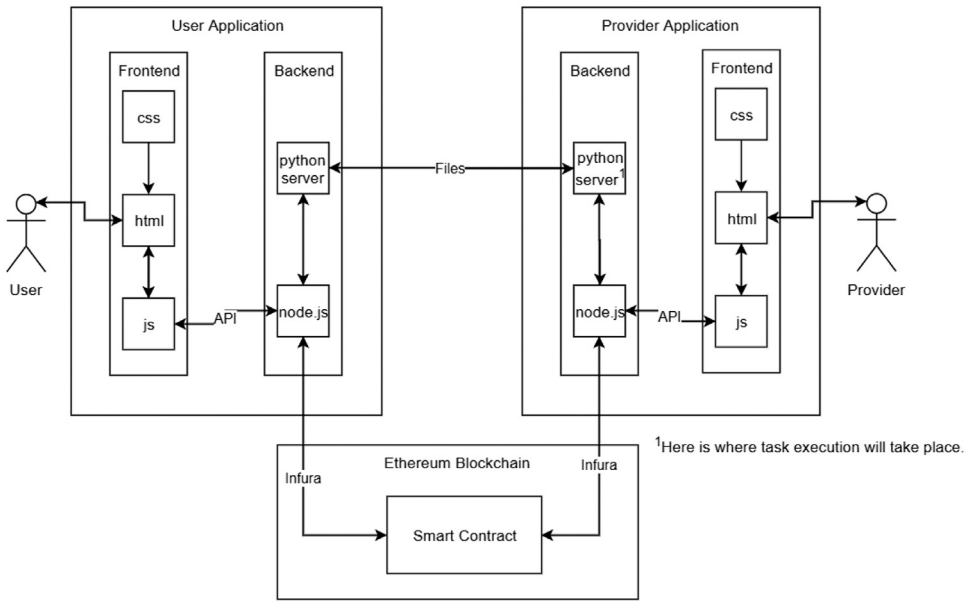

FIGURE 2 | Architectural overview.

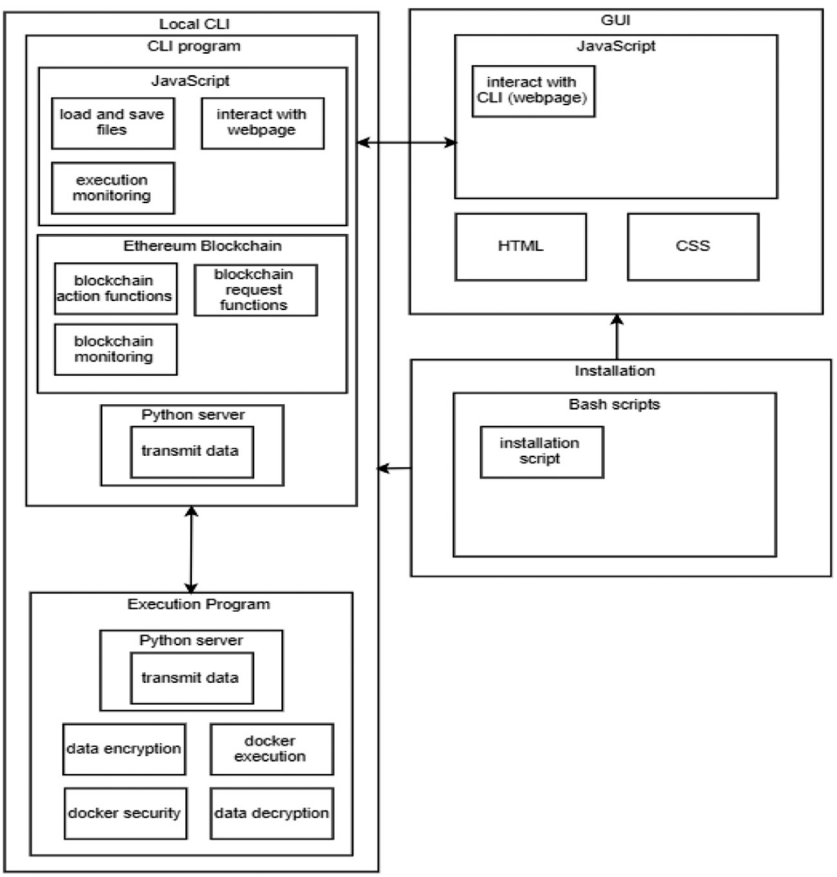

FIGURE 3 | Subsystem diagram of application.

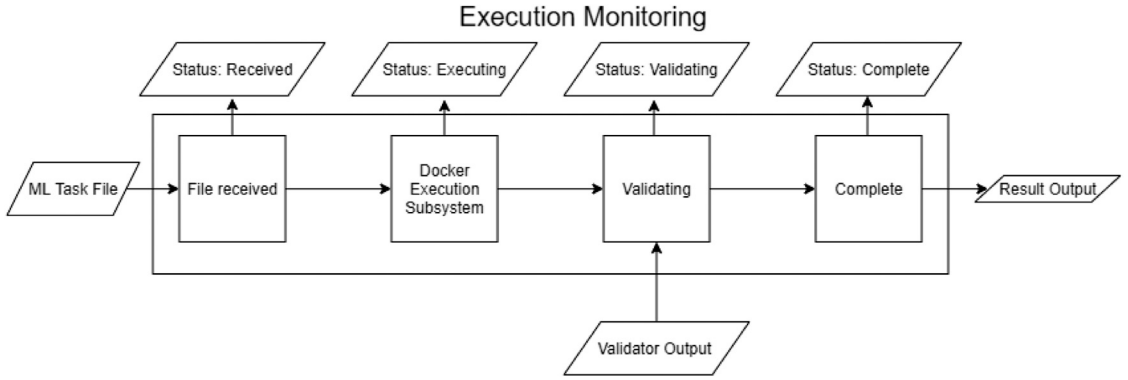

FIGURE 4 | Simplified execution status diagram.

similar to services such as a render farm, where one buys computing power to render three-dimensional animations.

This has many potential uses. Hobbyists, researchers, businesses, and those who want to experiment with Machine Learning but without a large investment will find it very useful.

From the user perspective, the one who uploads the task must take a few steps. After starting the application with their task file in a specified directory, they must start their request once their task file has been uploaded. After that, they must choose a provider. Once the provider has completed the task execution, they must also choose a validator to make sure the result is correct. Once validation is complete, they will receive their result file. They then must finalize their request and rate their provider.

The provider perspective is much simpler. They will simply start the application and start providing. After that, they will automatically be assigned to tasks, as either the main provider or the validator. They may leave the application running on their computer and accumulate profit from the execution they run.

Figure 4 is a very simplified progression of the task file as it goes through the program, showing the statuses outputted to the users as it goes.

Figure 5 is a much more in-depth diagram, which shows exactly what each of the three kinds of users do as the task goes through the steps to be successfully returned to the user in a completed state. Major progression is the user selects a provider, sends them the task, provider runs task, user selects validator, provider sends task to validator, validator validates, result is sent to user, then user finalizes the request.

As shown in **Figure 5**, the program works on a deposit system. When the user submits their task, they send a deposit to the contract. (In Ethereum, the smart contract itself can store ether) Once the user receives the result and finalizes the task, the provider will receive the payment from the contract. This ensures two things: 1) That the provider won't back out from the job after getting payment, and 2) The user won't back out from paying after getting the result. This makes sure that both user and provider remain honest.

Near the end of the user column in **Figure 5** is a step "give provider a rating." There is a rating system to assist the user in choosing their provider and validator. Once a user's task has been completed, they may rate their provider based on the quality and accuracy of the result. These ratings are then averaged and displayed to the user when they access the menu to choose the provider or validator. This helps assure that users can choose quality providers and validators who will complete the task efficiently and accurately. Due to constraints in smart contract file size, we actually have two smart contracts, one of them specifically for the rating system and aspects relating to it.

SAFETY AND SECURITY

We have many security considerations for this application. The most important one is in regard to the provider's computer, since they will be running arbitrary code written by a user. The application is written such that it will be "Dockerized" before being sent to the provider, who will then run it in a Docker sandbox. This ensures that if any malicious code is written, it will not be able to adversely affect the provider's computer. It also allows more robustness in the machine learning code.

The task file is also encrypted before being sent through the internet in case the user's code contains any sensitive or proprietary information. Furthermore, the files are transferred through Tor, which is a network whose main purpose is security and encryption. Both of these ensure user files will be very safe.

Finally, there are two systems in place to help ensure the result is correct. Both of them have been mentioned already. The first is the validation, where the validator runs the task and checks the result against the result given from the provider. The second is the rating system, so if someone has a low rating the user knows that they should not choose them.

Now we will discuss our file transfer system. This presented a unique challenge to us that took a lot longer to overcome than we thought it would. In the first versions of the project, connections were simply direct peer to peer connections using IP addresses. Not only is this insecure, but it posed a problem when peers could not connect to each other due to network architecture. Networks with rigid security and many NAT (Network Address Translation) layers could not connect to computers outside

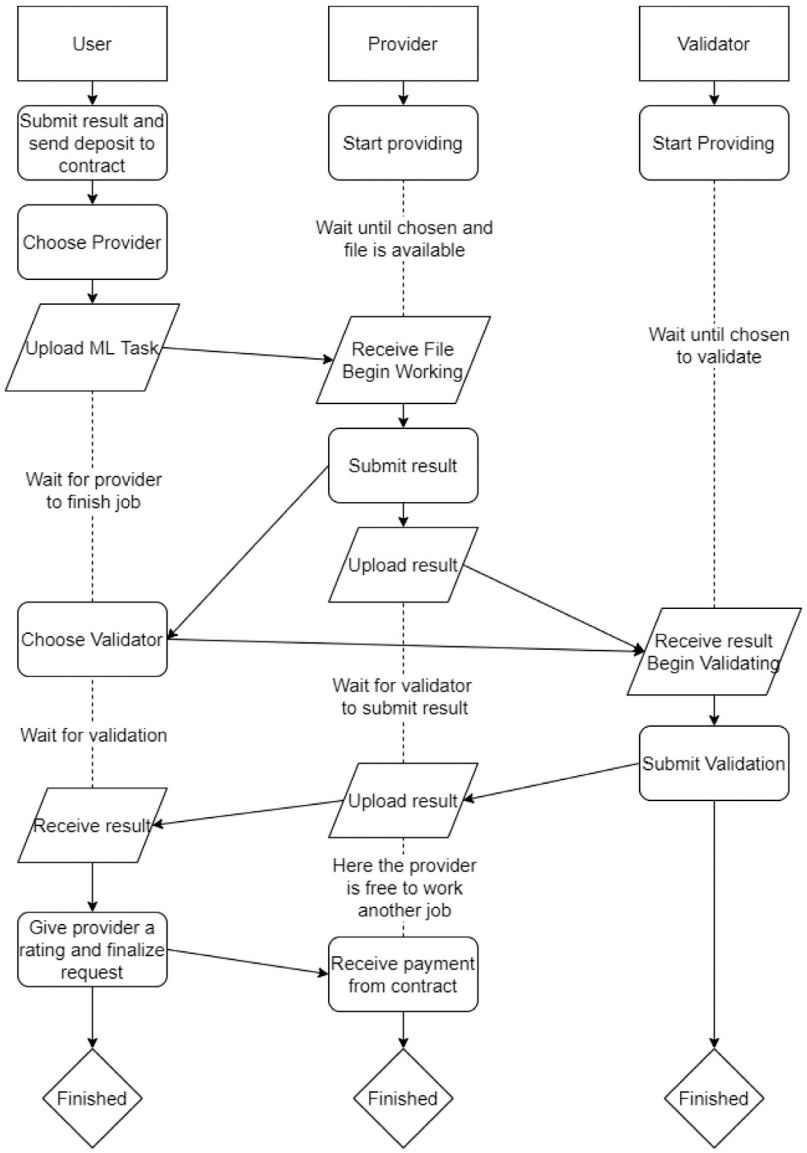

FIGURE 5 | Execution diagram to complete task.

that network. For example, we couldn't connect to a computer in LSU's network from outside of it.

We eventually found a solution in a program called Onionshare. This program allows a computer to host a file at a temporary address. A requester using Tor can then request the file and begin downloading it. This solution was very appealing to us. It allowed our requester to directly and securely download the file from a temporary address, making it difficult for anyone else to track the file location.

Unfortunately, Onionshare came with its own share of issues we had to spend a long time working out. The two primary ones are that it is slow, and that it is very error prone. To solve the problem of it being slow, we split the file into some amount of chunks, and send the chunks individually rather than sending the whole file at once. Once all the files are hosted, a final file containing the individual addresses is then hosted and sent to the requester. The requester can then read these addresses and make requests to each chunk. We discovered the optimal configuration is to split the file into as many chunks as the CPU has threads.

The side effect of splitting the file into x chunks is that there are now x times as many opportunities for errors to occur. A large chunk of development time was spent creating code to detect and correct these errors. For example, a very common problem is for the file to become stuck when it's starting to host. We had to build a system to detect when it would get stuck, and restart the hosting when this happens. Sometimes, it can happen several times in a row, requiring repeated restarts.

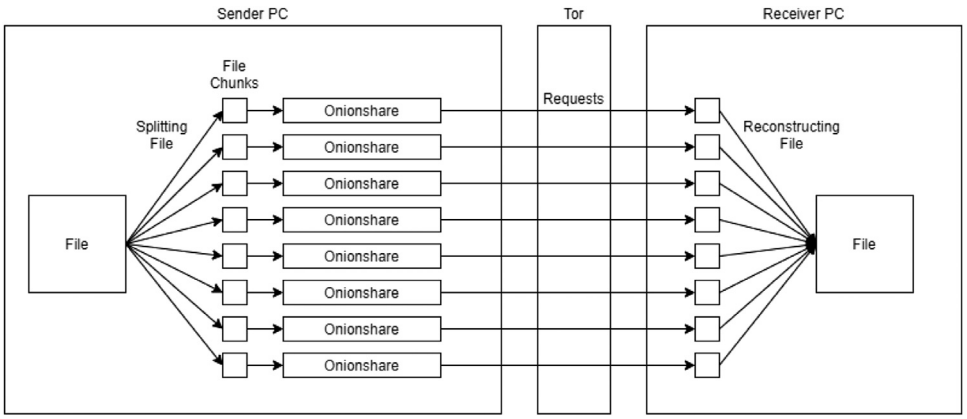

FIGURE 6 | File transfer process.

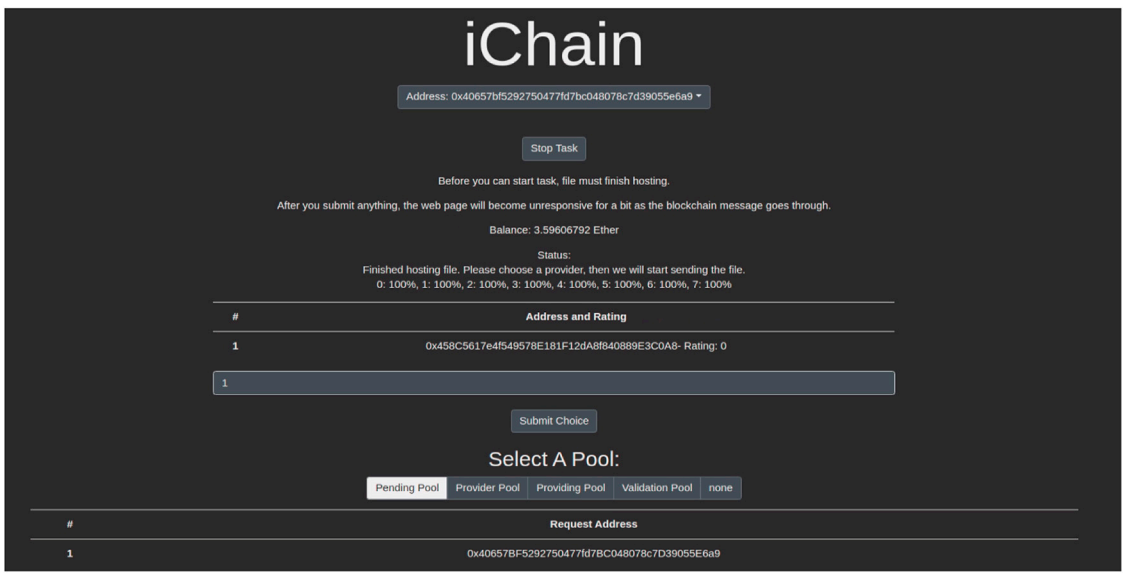

FIGURE 7 | User web application screenshot.

Figure 6 illustrates the process of sending the individual file chunks from the host to the requester using our novel file transfer. As far as speed goes, unfortunately the speed of transfer can sometimes just depend on Tor itself and varies by time and day. By our measurements, an average transfer speed is roughly 10 Mb/s using our system.

USER INTERFACE

Now that we've spent so much time discussing the application, how it works, and its systems, let's actually look at it. **Figure 7** is a screenshot of the web page application. This is for the user who uploads the machine learning task. Displayed on this figure from top to bottom:

1) The selected Ethereum address of the user
2) An option to stop the task
3) Some information for the user to help them
4) The user's balance at their selected Ethereum address
5) The status of their task execution. In this screenshot, their status is that they are currently waiting to choose a provider. Each chunk of their file is 100% finished hosting.
6) A form with which to select a provider. There is an Ethereum address displayed along with its rating (currently 0 since it has completed no tasks.)
7) A pool viewer. Pools are how we sort users and providers in different stages of task execution. Here, it can be seen that the user's address is in the "pending" pool, as it is still waiting to be assigned a provider.

```python
import tensorflow as tf
from tensorflow import keras
import numpy as np
import sys

#sys.argv[1] definition
#0->providor
#1->validator


#data section
(trainData, trainLabel), (testData,testLabel) = keras.datasets.mnist.load_data()
trainData=trainData/ 255.0
testData= testData/255.0

#create model
if sys.argv[1] == '0':
  mod = keras.Sequential([
    keras.layers.Conv1D(64, 2, padding="same", activation=tf.nn.relu, input_shape=(28,28)),
    keras.layers.Conv1D(64, 2, padding="same", activation=tf.nn.relu),
    keras.layers.MaxPooling1D(2),
    keras.layers.Dropout(.25),
    keras.layers.BatchNormalization(),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
    ])
else:
  mod =  keras.models.load_model("result.h5")


mod.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

if sys.argv[1] == '0':
  mod.fit(trainData, trainLabel, epochs=2)
  mod.save('result.h5')
loss, acc = mod.evaluate(testData, testLabel)
print('acc: ', acc)
print('loss: ',loss)
```

FIGURE 8 | Example machine learning script.

The application is distributed using a Bash script which, when run, will install all necessary software and libraries as well as downloading the application's executable files from a GitHub repository. This repository includes instructions for running as both user and provider, as well as how to create a proper machine learning task. The Onionshare repository is also downloaded, as the user must have it to perform the file transfer.

In order for the program to work, there are a couple restrictions in how the machine learning script must be written. Luckily, these restrictions don't constrain the actual machine learning, just how it must be written. The main restrictions on this file are that it must be written in python3 and it must accept a command line argument to determine whether it will be executed fully or simply check the result. The former option is for the provider, the latter is for the validator. **Figure 8** shows such a script.

To use the program, a user will need an Ethereum address and some amount of ether cryptocurrency. They must then add their keystore file to the programs user or worker directory, depending on which they are using. Helpfully, our program has a built in function to create an Ethereum address and the keystore file in the correct directory for a user so they don't have to figure out how to do it themselves. Users will still need to procure their own ether.

Currently, the application resides on the Ropsten test network. This is a network for Ethereum where the ether is "fake," just meaning it has no value and one can get it freely. As the name implies this network is for testing ideas without needing to risk spending real money. Eventually it will be placed on the main Ethereum network so that providers may be paid properly for their work.

DISCUSSION

There are some imperfections with the proposed approach to iChain. To ensure security and accuracy, many different calls

must be made to the smart contract. Since every call to the smart contract requires the mining of on Ethereum block, and it can take from 10 to 20 s for an Ethereum block to be mined, every run of the application can have the user spend 1–2 min just waiting for a block to be mined. The file transfer system, also ensuring security and connectability across networks, can be quite slow, even with the large speedup achieved by our method of splitting the file into chunks, taking several minutes to transfer a 1 GB file.

Similar applications exist for P2P energy trading markets over blockchain, where users with connected power grids trade energy using a blockchain market [Infinite Energy, (2020). W, 2020]. An example of one such market is DeTrade (Oprea and Bâra, 2020). Compared to our application, these markets use much more complex mechanisms for selection of buyers and sellers of energy. While this does automate the process of selecting a seller, it seems much too complex for our purposes, where the process of selecting a seller to compute your ML task is simple and allows the user more control. DeTrade also uses tokens backed by a third party bank, which is done to prevent price fluctuation, but causes the platform to not achieve full decentralization. Said (2020) proposes another such market specifically for electric vehicles within parking lots to trade electricity, utilizing machine learning to maximize profits.

Huckle et al. (2016) describes some applications of blockchain in various decentralized Internet of Things (IoT) scenarios. Although our application is not aimed at IoT, this paper's focus on DApps and their implementation in future transactions is similar to our goal of decentralized trading of services.

Hawlitschek et al. (2018) discuss the limitations of trust-free technology in blockchain. They assert that trust is essential to functioning society and that as blockchain becomes more ubiquitous it will rely more on trust. Our application does require a certain level of trust to believe that the result returned from the provider will be correct, but the validators and rating system allow the user to have good reason for this trust.

CONCLUSION

In summary, we have created a project that significantly lowers the barrier of entry to running high-complexity machine learning code. We hope that its use will allow more experimentation from those curious about machine learning, whether they are researchers, businesses, just trying it out, or anything in between.

AUTHOR CONTRIBUTIONS

LP created the idea, supervised, and gave advice. TL managed and helped development. CJ and PR worked on the website and the file transfer system. DM and SP worked on the Node.js backend and blockchain connections.

REFERENCES

Buterin, V. (2013). *Ethereum White Paper*. Retrieved from ethereum.org: https://ethereum.org/en/whitepaper/.

Columbus, L. (2020). *Roundup of Machine Learning Forecasts and Market Estimates*. Retrieved from Forbes: https://www.forbes.com/sites/louiscolumbus/2020/01/19/roundup-of-machine-learning-forecasts-and-market-estimates-2020/.

Di Persio, L., and Honchar, O. (2018). Multitask Machine Learning for Financial Forecasting. *Int. J. Circuits, Syst. Signal Process.* 12, 444–451.

Dinh, T. T. A., Wang, J., Chen, G., Liu, R., Ooi, B. C., and Tan, K. L. (2017). "BLOCKBENCH: A Framework for Analyzing Private Blockchains," Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago IL, May 14–19, 2017, PartF127746, 1085–1100.

Dugyala, R., Hanuman Reddy, N., Raghuram, G., and Lakshminarayana, J. (2018). Decentralized Secure Online Digital Data Registrations. *Int. J. Eng. Technology(UAE)* 7(4), 42–44. doi:10.14419/ijet.v7i4.6.20231

Hawlitschek, F., Notheisen, B., and Teubner, T. (2018). The Limits of Trust-free Systems: A Literature Review on Blockchain Technology and Trust in the Sharing Economy. *Electron. Commerce Res. Appl.* 29, 50–63. doi:10.1016/j.elerap.2018.03.005

Huckle, S., Bhattacharya, R., White, M., and Beloff, N. (2016). Internet of Things, Blockchain and Shared Economy Applications. *Proced. Computer Sci.* 98, 461–466. doi:10.1016/j.procs.2016.09.074

Infinite Energy (2020). *What Is Peer-To-Peer Energy Trading?*. Retrieved from Infinite Energy: https://www.infiniteenergy.com.au/peer-to-peer-energy-trading/.

Mearian, L. (2019). *What Is Blockchain? the Complete Guide*. Retrieved from Computerworld: https://www.computerworld.com/article/3191077/what-is-blockchain-the-complete-guide.html.

Nakamoto, S. (2009). *Bitcoin: A Peer-To-Peer Electronic Cash System*. Cryptography Mailing list at https://metzdowd.com.

Ogilvie, T. (2020). *Tokenized Staked ETH Will Replace ETH – and That's a Good Thing*. Retrieved from Coindesk: https://www.coindesk.com/staked-eth-will-replace-traded-eth.

Oprea, S. V., and Bâra, A. (2020). Local Market Mechanisms Survey for Peer-To-Peer Electricity Trading on Blockchain Platform. *Scientific Bull. Naval Acad.* 23 (1), 186–191.

Parkens, D. (2015). *The Great Chain of Being Sure about Things*. Retrieved from The Economist: https://www.economist.com/briefing/2015/10/31/the-great-chain-of-being-sure-about-things.

Rahouti, M., Xiong, K., and Ghani, N. (2018). Bitcoin Concepts, Threats, and Machine-Learning Security Solutions. *IEEE Access* 6 (8528406), 67189–67205. doi:10.1109/access.2018.2874539

Reiff, N. (2020). *Bitcoin vs. Ethereum: What's the Difference?*. Retrieved from Investopedia: https://www.investopedia.com/articles/investing/031416/bitcoin-vs-ethereum-driven-different-purposes.asp.

Said, D. (2020). A Decentralized Electricity Trading Framework (DETF) for Connected EVs: a Blockchain and Machine Learning for Profit Margin Optimization. *IEEE Trans. Ind. Inform.*, 1. doi:10.1109/tii.2020.3045011

Singh, H. (2019). *Everything You Need to Know about Hardware Requirements for Machine Learning*. Retrieved from eInfochips: https://www.einfochips.com/blog/everything-you-need-to-know-about-hardware-requirements-for-machine-learning/.

Zand, M. (2019). *An Introduction to Hyperledger Fabric*. Retrieved from opensource.com: https://opensource.com/article/19/9/introduction-hyperledger-fabric.

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.