LEDGER
ledgerjournal.org

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.20XX.XXX

RESEARCH ARTICLE

PredictChain: A Blockchain-based Predictive Marketplace

Abstract. One of the main issues with artificial intelligence training and development today is accessibility. Oftentimes, individuals or groups possess data that they would wish to be used in predictive analysis. However, these people may not have access to the compute capacity to train predictive models on this data. Additionally,yet other people have neither access to predictive training data, nor do they have access to adequate computational resources. Compounding this issue is the fact that the entities that *do* have these computational resources often keep the results of trained models hidden from the public. Due to this lack of accessibility, similar models are often trained on similar datasets, obtaining similar results. This results in both a notable inefficiency, and the withholding of useful predictions from the majority of people.

To help to aid with this issue, we propose PredictChain, a blockchain-based marketplace for predictive AI models. Through PredictChain, users are able to upload datasets for training predictive models, request to train models on any previously uploaded datasets, or submit queries to those trained models. These various models will be operated by a central node with computing resources available. A variety of archetype models can be chosen from, ranging from cheap, fast, and simple to more expensive, slower, and more powerful. This will allow for a large variety of predictive abilities for both simple and complex patterns. All the past predictions form these models will be stored on the blockchain for public viewing.

Given enough users, this project would have a notable impact on the state of machine learning usage. Currently, the problem of accessibility prevents many people from utilizing machine learning themselves, often leaving them to turn to highly private and centralized tech giants. This project would serve to open up this black-box of an industry and encourage the sharing of datasets and parameter configurations to create better models that are open to the public for usage.

KEY WORDS

1. Bockchain. 2. Decentralized. 3. Marketplace. 4. Oracle. 5. LSTM. 6. GRU. 7. RNN.

1. Introduction

1.1. Problem Solved—PredictChain helps to solve one of the main issues that involve AI models today: accessibility. Our project fulfils this need in two ways. Oftentimes, individuals or groups poses data that they would wish to be used in predictive analysis. However, these people may not have access to the compute capacity needed to train predictive models on this data. Additionally, yet other people have neither access to predictive training data, nor do they have access to computational resources. PredictChain solves both of these problems simultaneously.

When users upload their datasets, they allow a model to be trained on those datasets. Higher quality datasets will produce higher quality models. When users submit parameters for training, they allow the model that their parameters produce to be used publicly. Both of these users are rewarded for their work when a model is queried. The amount of this reward is based off of the correctness of the prediction. This encourages users to participate in contributing the resources needed for good predictions, while leaving a public record for other users to view.

1.2. Significance—As the trend of ever-growing machine learning models continues, this problem becomes increasingly relevant. As the scale and power of these models grow, the resources required to train them grow as well. This makes the training of useful machine learning models unattainable for most people. In order to get useful results from these models, users often have to pay large, centralized organizations, without any reward if they provide a good dataset or model parameters. PredictChain changes this paradigm by incentiveizing the thoughtful creation of useful models and datasets.

Additionally, after the recent mania and subsequent crash around blockchain adjacent technologies, it has become important to remind people that blockchains can be used for genuine utility in addition to investment. By primarily using Algos as a method of payment, instead of investment, it helps to, once again, show that cryptocurrencies can effectively be used as a pure form of payment for useful services. Of course, this is in addition to the many other services that use crypto in a similar manner, but adding one more project only helps the Algorand's notion of usefulness.

2. Background

Throughout our project, we have built upon, or have related to, the ideas of many other projects and papers. Some of these relations were explicit, such as our work with, and utilization of, various types of neural network. Others were implicit, like our project's relation to the many other implementations of blockchain-based predictive projects. It is important to acknowledge the influences that other works have had on our project, and it is equally important to analyze the similarities between our project and the other papers that we have referenced here. Through this, we hope to better understand the perspective of others on similar topics to our project and hopefully improve upon our work and techniques in the future.

2.1. Sharing Updatable Models—Decentralized & Collaborative AI on Blockchain[1] is a great example of a project that combines both artificial intelligence and blockchain technologies in a similar manner to our own. In this paper, Microsoft outlines a blockchain-based predictive marketplace that utilizes AI for predictive analysis. In this system, there exists an initial model that is trained on the data collaboratively provided by multiple users. To encourage the submission of high-quality data, they propose an incentive mechanism that rewards honest users, and punishes

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.20XX.XXX

dishonest ones. This model is trained off-chain on the client's machine. All the logic involving the data submission and incentives are done within an Ethereum smart contract.

The Model –This system leaves the exact definition of the model flexible between different clients running the system. They suggest that the specific class of model would be some type of predictive model, giving examples of CRF-RNNs and Support Vector Machines. Depending on the model that is used, the authors mention that the corresponding incentive mechanism would change from model to model.

Incentives –Similarly to the models, the authors leave the incentive mechanism of the project also without an exact definition. This is to give end users flexibility in their individual implementations. They do give several examples and specifications of this mechanism, however. These examples fall into two categories: gamification and monetary incentives. In the gamification strategy, they suggest that there should be no monetary incentive for contributing data, instead using tokens or badges. For the monetary route, they suggest that users who choose to upload data may have to also submit some stake into the contract as collateral.

2.2. The Price You Pay For Trust—Blockchain Enabled AI Marketplace: The Price You Pay For Trust[2] is another example of a marketplace that uses both AI and blockchain technologies effectively. One of the main focuses of this project is on privacy. It explicitly gives examples of health or insurance entities needing models trained on confidential data. The principles that this project is built around are *trust*, *fairness*, and *auditability*.

Datasets –The datasets in this system are unique since the bulk of the data is not tied directly to any one training request. In this project, a user uploads only a small chunk of data. This data will then be used as the validation set for the training. The majority of the training data for this query is pulled from private sources. This serves the purpose of only requiring users to upload a small portion of their data. However, this has the disadvantage of requiring external data to train models. To accommodate this, the authors implement several layers of security to protect private data sources.

Models –The models that this paper targets are much less integrated than those of other papers. Also unlike other implementations, these models are trained in a decentralized nature. This training strategy is reflective of federated learning. No one model is given all the data in bulk, it is only given sections with the confidential information stripped out. To add to their security, they also use hashing algorithms to further secure their data. Additionally, the models themselves are eventually shared with the requesting user along with the model results.

Blockchain Communication –Like other projects, this implementation uses a significant off-chain component. This component communicates with the private, permissioned blockchain that this system uses. This is to allow for more complexities in the training of the models as on-chain training has significant speed and memory constraints. By using Hyperledger Fabric for data distribution, data can be selectively distributed to only the nodes that need the data for training will have access to it.

2.3. Blockchain Prediction Markets for Renewable Energy—Towards the Use of Blockchain Prediction Markets on Forecasting Wind Power[3] is a paper that lays the groundwork for a use-case of blockchain empowered prediction markets. Like blockchain technology, renewable energy has recently ballooned in use. This paper combines these two fields in a way that utilizes their combined utility for aiding progress in the field of renewables.

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.20XX.XXX

Motives –A key problem with many forms of renewable energy sources are their unpredictable supply. Wind turbines need wind, solar panels need sunshine. Being able to better forecast the output of these energy production methods at a various time or place can help alleviate the pitfalls that prevent them from growing in market share.

Implementation –On their platform, anyone can propose an event and request predictions for that event with the promise of providing a reward. Reporters then propose several predictions over the course of the seven-day window that the market is active. In this implementation, the hosts of this predictive market do not take responsibility for the training and execution of the predictive models. This is all done on the side of the reporters. By using their framework, reporters who provide correct data are rewarded and those who provide inconsistent data are punished.

2.4. Stock Predictions—This next paper is closely aligned with the example dataset that we are using for PredictChain. They focus on using big data to train predictive machine learning models to forecast the short-term future behavior of stocks within the Chinese stock market.[4] For their dataset, they had gathered the past performance of 3558 stocks from the Chinese stock market over a period of two years. In order to gather this data, they utilized the *Tushare* API, along with web-scraping from *Sina Finance* and the *SWS Research* website. Within their paper, they utilize several techniques; designed to cut down on the noise that is inherent to the stock market to allow for better predictions. Their primary method for doing this was through feature engineering.

Feature Engineering –As part of their feature engineering, they utilized three primary techniques. First, they applied feature extension to their dataset. Through the use of this technique, they added additional meta-attributes to the entries, such as polarizing or calculating the fluctuation percentage. Adding this meta-data to the dataset helps to give the model a more complex picture of the stock and how it performs over time. Next, they eliminated some features, based on their influence, by using the Recursive Feature Elimination (RFE) algorithm. By eliminating unnecessary features, they allow their model to pay more attention to the most influential features, improving its predictive capabilities.

Predictive Models –For their testing, the authors used a two layered LSTM model, with the model only having an input layer and an output layer. As for their output, they kept the output as simple as possible to make sure the model was focussed only on predicting the movement of the stock. The model either outputted a 1 for the stock going up at a given time step, or a 0 if the model believed the stock would go down.

2.5. Evaluation and Analysis—Through this review, we examined several papers that demonstrate similar ideas and architectures to our project with slightly different implementations. Harris and Waggoner (2019)[1] had many similarities to our project, but was implemented using a more decentralized architecture. Sarpatwar, et al. (2019)[2] takes some of the decentralization ideas of Harris and Waggoner (2019)[1] and expands upon it by structuring their project with security as another main focus. Shamsi and Cuffe (2020)[3] demonstrates a market, similar to ours, but using a more decentralized architecture and a more specific scope. Finally, Shen and Shafiq (2020)[4] helped us to better understand some of the future improvements that we could make to both our dataset preprocessing and model training.

In our work, we have expanded upon these ideas and addressed some of their shortcomings. PredictChain features a more centralized architecture than these papers, Sarpatwar, et al. (2019)[2] and Shamsi and Cuffe (2020)[3] in particular. Our aim with this strategy is to focus on simplicity

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.20XX.XXX

and ease of use. Without the requirement for a large network of oracles, our implementation allows anyone to quickly set up an oracle instance and begin connecting to clients. Another design choice that we made was in our suite of built-in archetype models. By giving each client a preset list of models to choose from, it is no longer the client's responsibility to train and implement the models, as in Harris and Waggoner (2019).[1] This helps to make setup and usage much cleaner and more accessible. Overall, PredictChain aims to be more generalized and accessible to non-technical users at the cost of centralization.

3. Implementation Details

The structure of PredictChain is primarily broken up into two parts: the client and the oracle. Both of these parts interact with each other through the blockchain. The following diagram illustrates this relation:

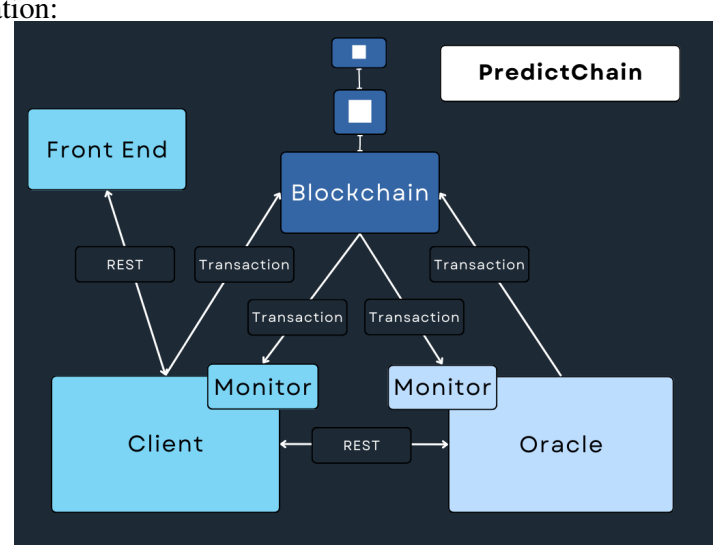

Fig. 1. The architecture of PredictChain

The above diagram illustrates the core components of the system, along with their methods of communication.

3.1. The Front End UI—Although not directly related to the AI or the blockchain parts of the product, it is still important to discuss the UI aspect of our project. The UI sets the user's impression of PredictChain by having multiple pages that create a more pleasant experience for the user. For example, the home page talks about our mission, how we differ from our competitors and example model sets we provide. The addition of the *FAQ* and *Meet the Team* pages allow users to understand who created PredictChain, as well as get answers about any privacy/security concerns they might have. Lastly, users can create an account or login to a pre-existing account. Then users are able to use PredictChain to its fullest functionality such as adding datasets, checking prices, or querying models with provided datasets. The UI talks to the client through a series of REST requests in order to convey this data, later receiving the result of the user's actions later through the same means.

3.2. The Client—The client serves as a middleman between the front end user interface and the blockchain. It is run as a server, serving UI content to the user, taking in requests

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.20XX.XXX

from the UI, and parsing those requests into a form suitable for both the blockchain and for the oracle. Additionally, the client constantly polls for updates coming from the oracle, through the blockchain, and then through its monitor. These updates are queued and sent to the front end upon request. This allows the user to both interact with the blockchain and to see the important updates that come from it.

3.3. The Oracle—The oracle accomplishes the majority of the other tasks that this project requires. It constantly polls for updates coming from the client, through the blockchain by using its own monitor.

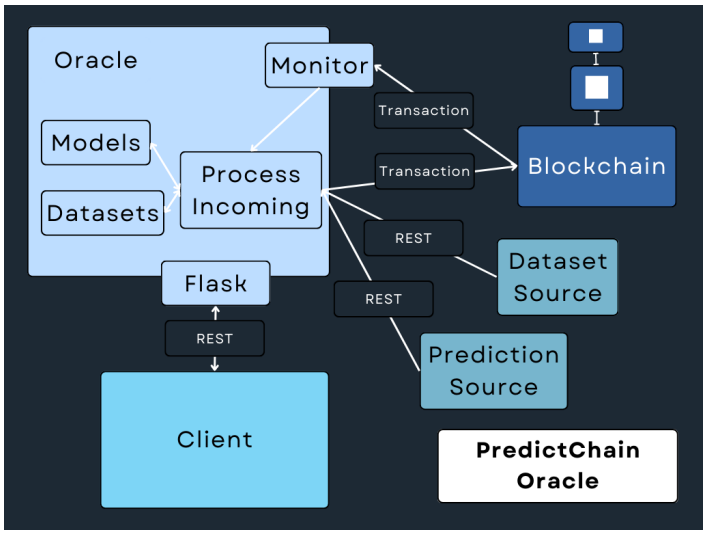

Fig. 2. The architecture of the Oracle

Upon receiving these updates, it begins the execution of one of its three main operations. These are:

- Downloading a user-specified dataset and saving it
- Training one of the raw models based on user inputted parameters
- Querying one of the trained models on user inputted data and comparing it to the real-world result

After each of these operations, the oracle sends out several blockchain transactions. These can be either rewards to contributors of a model or confirmations/results of the operation that has been performed.

When working with user-submitted datasets, the oracle uses a handler to manage the operations performed on that dataset. The handler can save datasets to a specified environment, load datasets from a specified environment, parse that dataset as a pandas dataframe, and split the dataset by the values of one of its attributes. The environments that the handler recognizes are *local* and *IPFS*. When using either of these environments, the handler abstracts away the complexities of working with either of them into a unified interface.

When working with user-trained models, the oracle uses a similar, common interface. This interface can create the model architecture, train the model on a selected dataset, query the trained model, evaluate its performance, save the model, and load it back from a specified environment. When creating and training a model, the interface chooses among a group of archetype or template

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.20XX.XXX

models. These models can be a:

- Multi-layered perceptron neural network[5]
- Recurrent neural network[6]
- Long short-term memory neural network[7]
- Gated recurrent unit neural network[8]

Each of these models has a *model_complexity* attribute. This is a simple float value, designed to give users a general idea of how performant a model can be once trained and serves as a method of calculating the cost of using or training that given model. The attribute itself is calculated using the size of the network and a linear multiplier to account for more complex model architectures. For models like GRUs or LSTMs, the complexity is higher as they are more complex and often better performing models.[9] For our RNN models, they have the ability to predict several time steps ahead, but their quality often degrades when a lag between evidence and prediction is introduced.[10] For these networks and our MLPs, the complexity is lower. This gives the desired effect of faster, simpler models being cheaper than the slower, more complex models without any heavy calculations. The interface abstracts most of the complexities of training, querying, and evaluating these models. The only difference between them is the inclusion of several optional parameters.

3.4. The Blockchain—In PredictChain, the blockchain serves as both a records keeper and a messenger between the client and the oracle. This is accomplished by using transactions as a form of direct communication. With every transaction sent, there is a note. This note is a json-encoded string (encoded in base64) that communicates information about the operation that the transaction is requesting and arguments for that operation. These operations are represented by a series of op codes. These codes are abbreviations of the operation name enclosed in angle brackets, for example $\langle QUERY\_MODEL \rangle$. The arguments to these operations are represented as a named dictionary, with each key being the name of the argument and each value being the argument itself. This named strategy allows the program to be very flexible without worrying about the exact ordering of the arguments. Blockchain is quite useful in its role due to its immutability and its transparency. Using a blockchain means that all requests are permanently stored and public, so other users can see what type of models are useful for specific datasets and what results those models have produced.

3.5. Software and Libraries—To properly implement our project, we build upon several libraries, resources and SDKs. These include:

3.5.1. Python Libraries—This project is primarily built in Python, using the Algorand SDK. The SDK makes interacting with the blockchain very straightforward. Through the tools provided by this library, we can easily read and write transactions from the Algorand blockchain.

Through Python, we also used data science and machine learning libraries such as Pandas and Torch. These libraries encapsulate many of the complexities of data preparation and model training for us. By using these libraries, we were able to concentrate on the higher-level functions of the project instead of worrying about the lower-level implementation.

Flask as also an important part of the project. We used Flask to allow both the client and oracle nodes to function as servers. The client would take in requests from the user and send out requests to the oracle. The oracle would then take in those requests and issue responses. We chose to use Flask in client-oracle communication to cut down on the amount of trivial transactions that would otherwise be made. For example, it would not benefit the accessibility or transparency of

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.20XX.XXX

the project greatly if the exchanged transactions were dominated by simple '*what is the price of XXX*' requests.

3.5.2. Node Libraries—Additionally, the front end utilizes the React framework and firebase. React was useful to us as it streamlined the process of making dynamic, modular code for the web interface. Firebase was invaluable for handling administrative tasks such as keeping track of registered users and their associated metadata.

3.5.3. Redis—For the recommended configuration of the project, we use Redis as well. Redis helps to provide a reliable store for our metadata about models and datasets in a simple, persistent manner.

3.6. Resource Links—The following table will provide links to the various resources relevant to our project and its evaluation:

Table 1. Project Resources

Resource	**Link**
GitHub	
Python Documentation	[Our GitHub]/docs/sphinx/index.html

4. Evaluation

While the most definitive evaluation would be to deploy our project and get feedback from actual users, we are limited in our time and in our scope. In place of this, we have devised several tests that are designed to evaluate each component of the project on its own and how the entire project functions as a whole.

4.1. Transactions—The usage of transactions is central to the communications protocol of PredictChain, so making sure the protocol functions correctly is critical to the evaluation of the project. Thanks to the SDK, we had no issue with encoding the notes and actually sending the transactions. Where problems can potentially arise is within the client and oracle monitors.

The monitors are classes inside the client and oracle that listen for any transactions that have their node address as the recipient. This listening is done using the Algorand indexer class and a constant polling for new transactions. We noticed that this monitor would sometimes skip or duplicate incoming transactions. As we developed fixes for these issues, we constantly evaluated the performance of the monitor. This was done through programmatically sending one or more transactions to the client or oracle address and checking to see how the monitor handled them. By comparing the unique transaction ids of the sent transactions to those processed, we were able to evaluate the monitor, identify issues, and create fixes. For example, we now constantly update the minimum timestamp the indexer can look for transactions after and we also keep a registry of the ids of all past transactions. This helps to eliminate the duplicate transactions that were getting through.

4.2. Models—Another critical component of PredictChain is its usage of a variety of models. At present, all of these models are different types of neural networks. However, these networks are not created equally, with each having different qualities, architectures, and outputs. In order to

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.20XX.XXX

evaluate the performance of these models, we ran several experiments on the models where their hyper-parameters and architectures were kept constant as they were tested. For this evaluation, we measured their performance on our sample dataset, The University of California, Irvine's *Dow Jones Index* data set.[11] This dataset has 16 different parameters, detailing the attributes of a range of stocks from the first half of 2011. For our usage, we do not eliminate any of these during training, although this may be an opportunity for future improvement. As for the models, they were all initialized with the following parameters:

Table 2. Model Evaluation Parameters

Parameter Name	**Value**	**Description**
epochs	70	The number of epochs that the model would train for
target_attrib	*close*	The attribute that the model was trying to predict, this is the daily closing price of the stock
hidden_dim	5	The number of neurons in each hidden layer
num_hidden_layers	1	The number of hidden layers
time_lag	0	The number of time steps that pass between the input window and the prediction
training_lookback	10	The number of time steps that recurrent models receive as input
sub_split_value	0	The integer id of the stock to predict, in this case its is *$AA* (Alcoa Corp)

We performed this test on all of our basic model structures, specifically our GRU, LSTM, RNN, and MLP models. In the following figures, we show the input data and predictions for each model. We also show the final loss and the final accuracy. The loss is calculated using the mean absolute error function $mae(x,y) = (\Sigma_{i=1}^{n}|y_i - x_i|)/n$ and our accuracy is calculated using $acc(x,y) = \sigma(-mae(x,y) + e^2)$ where σ is the sigmoid function. Our accuracy function is modified in this way so that very large losses are registered as somewhat accurate and lower losses are registered as very accurate. This helps to account for the very large losses generated by some of the less performant models.

The results of our evaluations are as follows:

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.20XX.XXX

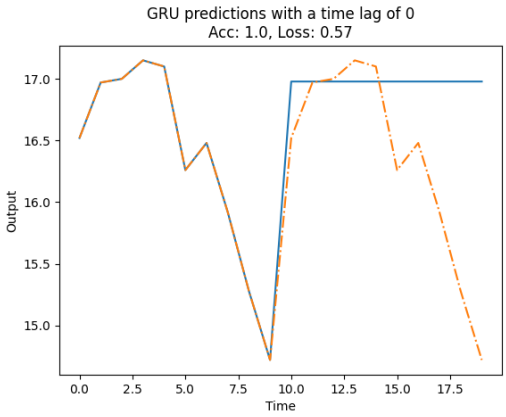

Fig. 3. The results from our GRU model

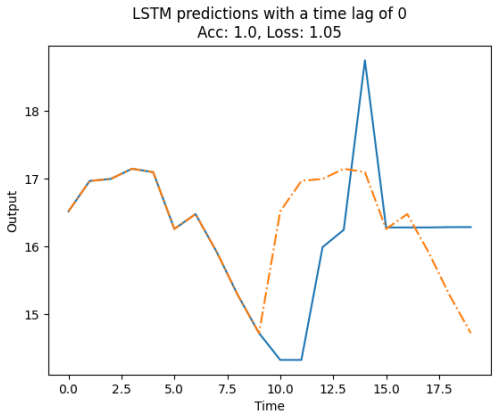

Fig. 4. The results from our LSTM model

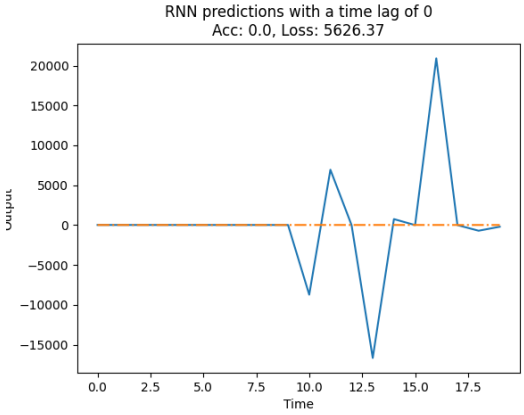

Fig. 5. The results from our RNN model

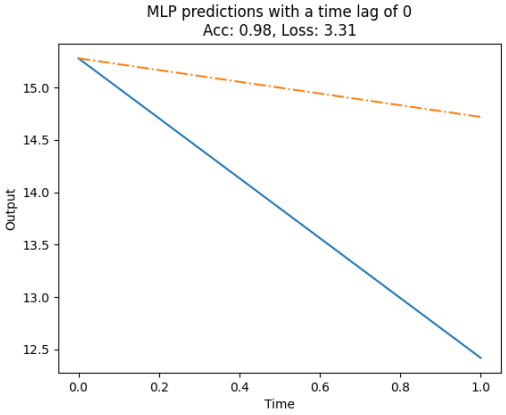

Fig. 6. The results from our MLP model

Each of these results reflects the individual strengths and weaknesses of each model. As is reflected in our *model_complexity* calculations, the GRU model is the most performant, with a loss of only 0.57. This is closely followed by the LSTM model, which still offers a good prediction, but is slightly less consistent overall. In our evaluation of our RNNs, we noticed a good example of the exploding gradients problem. Without the limits provided by the GRU and LSTM models, the predictions of the RNN become increasingly erratic. Finally, our MLP model does not suffer from an exploding gradient, but it does lack the recurrence of the other three models, only being able to output one day at a time.

4.3. End-To-End Testing—The final component of our testing was the end-to-end series of tests we performed. The goal of these tests was to give us a holistic picture of how well our project functioned and how well that functioning met our stated goals. For all of these tests, we started with front end user interaction and ended with the response to those actions.

Price Queries –The first, and most straightforward, set of operations that we tested were the three price query operations. We performed this test by submitting query requests on the front

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.20XX.XXX

end, then tracing the functions called by that operation from the client to the oracle, then to the oracle getting the price, then finally back to the client. We performed this test for all three of our queries, testing different sets of inputs, both valid and invalid, seeing if they were handled properly.

Major Transactions –The next set of tests we performed focused on our major transactions: *Upload Dataset*, *Train Model*, and *Query Model*. These tests were similar to those for the price queries. As we performed these tests, we noted that, while the operations did complete successfully, the user feedback and results were ambiguous. Additionally, we realized that it may be confusing for the user to remember the exact names of the models and datasets in order to use them. To address these issues, we made several additions. To address the issue of users having to input the exact names, we added a dropdown feature instead of a test input. We now store a list of all current datasets and models in the system. This is updated upon the reloading of the page or when a user submits a transaction. This ensures the list is always updated for better ease of use. To address the feedback issue, we added a feedback section that periodically pinged the client to see if any new response transactions came in from the oracle. If these transactions did come in, we would display both the operation, the model or dataset name, and any extra data (like the result of a query) to the user. This way, the user would not have to manually check the transaction note on the Algorand block explorer.

By performing these tests, we were able to both evaluate the quality of the project as a whole and make important additions to improve the user experience. Our transaction tests helped us to identify the issues that had previously existed and to verify that the current communication protocol was working properly. Our model tests helped us to confirm our previous assumptions about the nature of our various models and which scenarios they are most useful in. Finally, our end-to-end testing confirmed the overall functioning of the project and inspired us to make some valuable improvements to the user experience.

5. Conclusion

5.1. Summary—Over the course of our work, we have made PredictChain into a working blockchain-based marketplace for predictive AI models. Through PredictChain, users are now able to upload datasets for training predictive models, request that basic models be trained on any previously uploaded datasets, or submit queries to those trained models. These various models will be operated by a central node with computing resources available. A variety of models are available, ranging from cheap, fast, and simple to more expensive, slower, and more powerful. This will allow for a large variety of predictive abilities for both simple and complex patterns. All the past predictions form these models will be stored on the blockchain for public viewing.

5.2. Future Work—As mentioned above, we have several opportunities for improvement that could be accomplished with future work into this project. One major improvement that we could make to the project is giving users who upload datasets greater flexibility in how their data is preprocessed. This may come in the form of some feature engineering, similar to that found in Shen and Shafiq(2020).[4] Another potential improvement that could be introduced with future work is the addition of a greater variety of models or more example datasets. Currently, we only use neural networks to use for predictions. In future work, we would like to add a more diverse set of models, such as decision trees or more statistical models based off of Bayesian

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.20XX.XXX

inference. Finally, we could make improvements to how our models are trained, for instance, allowing users to prune off attributes from a dataset that they did not deem useful to the trained model. Adding these improvements may make PredictChain a more fleshed-out version of itself without changing the core principles of our project.

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.20XX.XXX

Notes and References

[1] Harris, Waggoner "Decentralized & Collaborative AI on Blockchain." *IEEE* .

[2] Sarpatwar, Ganapavarapu, Shanmugam, Rahman, Vaculin "Blockchain Enabled AI Marketplace: The Price You Pay for Trust." .

[3] Shamsi, Cuffe "Towards the Use of Blockchain Prediction Markets for Forecasting Wind Power." *IEEE* .

[4] Shen, Shafiq "Short-term stock market price trend prediction using a comprehensive deep learning system." *Journal of Big Data* .

[5] Rosenblatt "The Perceptron: A Perceiving and Recognizing Automoton." *Cornell Aeronautical Labratory* .

[6] Rumelhard, Hinton, Williams "Learning Internal Representations by Error Propagation." *Paralell Distributed Processing: Exploration in the Microstructure of Cognition* **1**.

[7] Hochreiter, Schmidhuber "Long Short-Term Memory." *Neural Computation* .

[8] Cho, K., *et al.* "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation." `1406.1078`.

[9] Chung, Gulcehre, Cho, Bengio "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling." *Neural and Evolutionary Computing* .

[10] Hochreiter, Schmidhuber "Bridging Long Time Lags by Weight Guessing and 'Long Short Term Memory'." .

[11] of California Irvine, U. "Dow Jones Index Data Set." archive.ics.uci.edu/ml/datasets/Dow+Jones+Index.

SSN 2379-5980 (online)
DOI 10.5195/LEDGER.20XX.XXX