Predict Chain

Matthew Pisano, Connor Patterson, William Hawkins

Spring 2023

1 Project Description

PredictChain is a marketplace for predictive AI models. Users are able to upload datasets for training predictive models, request that train 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 or nodes with computing resources available. A variety of models will be made 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.

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 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 solved both of these problems simultaneously.

When users upload their datasets, they allow 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, and it produces a correct 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 Background

As the trend of ever-growing machine learning morels continues, this problem becomes increasingly relevant. As the scale and power of models steadily grows, the resources required to train them grows as well. This makes training 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 to the Algorand's notion of usefulness.

1.3 Use Cases and User Stories

User Story #1

Scenario: As a data analyst, I want to have access to trends of various stock markets so that I can create predictive models that will inform my investment strategies

User Story #2

Scenario: As a stockbroker, I want to compare my dataset to another dataset against the same model so that I can get an idea as to which is better

Use Cases

For the sake of space, we will only go over two simple Use Cases, one that allows the user to add a dataset, and another that checks a price request for a dataset

Table 1: Add Dataset

Identifier:	UC1
Description:	The user logs in to PredictChain and adds a dataset to their account
Actor(s):	Site User
Precondition(s):	PredictChain is set up properly (connected to the client which is connect to Oracle)
Event Flow:	1. The user logs in properly into their account (or creates a new account) 2. The user inputs a link to the dataset file 3. The user inputs a name corresponding to that dataset 4. The user enter in the amount of bytes the dataset file is 5. The user presses "Submit" 6. The user then either sees the corresponding Oracle transaction ID or an error occurred
Postcondition(s):	The website displays the information onto the dashboard corresponding to the outcome

Table 2: Dataset Price Request

Identifier:	UC2
Description:	The user logs in to PredictChain and checks the price for a dataset that contains a certain amount of bytes
Actor(s):	Site User
Precondition(s):	PredictChain is set up properly (connected to the client which is connect to Oracle)
Event Flow:	1. The user logs in properly into their account (or creates a new account) 2. The user inputs the corresponding amount of bytes that a sample file would have to check the price for 3. The user is then shown the price that it would cost for a user to upload a file that size in bytes
Postcondition(s):	The website displays the information onto the dashboard corresponding to the input

2 Implementation Details

The following table will provide links to the various resources relevant to our project and its evaluation:

Table 3: Project Resources

Resource	Link
GitHub	github.com/AI-and-Blockchain/S23_PredictChain
Python Documentation	github.com/AI-and-Blockchain/S23_PredictChain/docs/sphinx/index.html
Demo Video	www.youtube.com/watch?v=icWc1qvhsgY

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.

2.1 The UI

Although not directly related to the AI nor the blockchain parts of the product, it is still important to discuss the UI aspect of our project. The UI creates the face 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 FAQ pages 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 in order to get this data and then provides it to the user in a nice manner.

2.2 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 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. 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 important updates that come from it.

2.3 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. Upon receiving these updates, it begins the execution of one of its 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 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's recognize 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 models. These models can be a:

- Multi-layered perceptron neural network

- Recurrent neural network

- Long short-term memory neural networks

- Gated recurrent unit neural networks

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 linter multiplier to account for more complex model architectures. For models like GRUs or LSTMs, the complexity is higher as they are mode complex, and often better performing models. For models like 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.

2.4 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 is a note. This note is a json-encoded string (encoded in base64) that communicate information about the operation that the transaction is requesting be performed and arguments relevant to that operation. The operations in the note are represented by a series of op codes. These codes are abbreviations of the operation name enclosed in angle brackets, for example ¡QUERY_MODEL¿. 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.

2.5 Software and 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 encapsulated 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 of 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 the project greatly if the exchanged transactions were dominated by simple 'what is the price of ...' requests.

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.

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

3 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.

3.1 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. There are few issues with encoding the notes and actually sending the transactions, thanks to the SDK. 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 mode transactions to the client or oracle addressand 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 duplicate transactions getting through.

3.2 Other

4 Conclusion

4.1 Summary

PredictChain is a blockchain-based marketplace for predictive AI models. Through PredictChain, users are able to upload datasets for training predictive models, request that train 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 or nodes with computing resources available. A variety of models will be made 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.

4.2 Limitations

A notable limitation that we experienced during this project was constraints on both our time and the amount of hours we could each put into the project. If this project was worked on for several months longer, the final product would be significantly more fleshed out with more features. If this were more akin to a multi-semester project, we would have had the opportunity to add more starting datasets and adding a more diverse set of models.

Another limitation that we had experienced was that we each had other commitments to other classes or activities. This limited the amount of hours per week we could each work on improving the project.

Despite, or perhaps aided by, these limitations, were able to budget our time and distribute the work in such a way that out final project is in a functioning state that fulfils our initial objective: to create a prototype for a accessible and transparent marketplace for machine learning training and predictions.

4.3 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 the overhaul of the architecture of the project. As mentioned previously, we had briefly considered using many model training nodes instead of a centralized node. Adding this to the project would make the project more decentralized and open up another opportunity for community contribution. This would likely come in the form of users hosting training nodes and being rewarded for the quality of models that it produces. 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 inference.