DAI (Davinci AI) Functional implementation

1. **AI Block**: System Risk Managements.

   (1) Solutions for transaction rate problems

   @SET UP RAIDEN NETWORK

   - **By using Raiden network, we can implement Off Chain Scale and simplify transmission process of token (coin), so that we can conclude contract with faster and lower cost.**

   - In order to ensure that participants pay their debts, tokens have to be locked up as security in a smart contract for the lifetime of the payment channel. This deposit ensures that tokens can only be used to send and receive tokens to and from the channel partner until the channel is finally closed by either participant, preventing both from double-spending their tokens to other peers.

     Once a channel is created, participants may issue what can be considered certified checks freely back and forth. Instead of keeping track of all checks, however, each peer only keeps a copy of the latest one. The balance proof contains the final sum of all Raiden transfers sent to a participant up to a certain point, digitally signed by the sender. Since each channel has two participants, it always maintains two of those and together they are essentially the channel's bar tab if you will. Multiple credits are exchanged back and forth, changing the total amount owed between the participants, possibly even rebalancing the channel many times in the process.

     Finally, when one party decides to settle the balance on the blockchain, either to claim or pay their outstanding balance, they can close the channel at any time by presenting their balance proof of choice to the smart contract. The other participant the one that did not choose to close the channel -- must now present a balance proof of their own or do nothing if they received no transfers. After both parties have submitted their balance proofs, they may now withdraw their deposits. This withdrawal may be triggered by anyone, including addresses other than the two participants.

     If the second participant fails to present their balance proof in time, balances will be distributed according to the closing participant's proof, assuming that the other participant has not received any transfers. This way, Raiden asserts that each payment channel participant always has access to their funds.

@TRANSACT WITH SHARDED TYPE DATA

- **Sharding technique reduces data I / O load, enabling faster DApp communication.**

Reads / Writes

DataBase distributes the read and write workload across the shards in the sharded cluster, allowing each shard to process a subset of cluster operations. Both read and write workloads can be scaled horizontally across the cluster by adding more shards.

For queries that include the shard key or the prefix of a compound shard key, DBs can target the query at a specific shard or set of shards. These targeted operations are generally more efficient than broadcasting to every shard in the cluster.

Storage Capacity

Sharding distributes data across the shards in the cluster, allowing each shard to contain a subset of the total cluster data. As the data set grows, additional shards increase the storage capacity of the cluster.

High Availability

A sharded cluster can continue to perform partial read / write operations even if one or more shards are unavailable. While the subset of data on the unavailable shards cannot be accessed during the downtime, reads or writes directed at the available shards can still succeed.

You can deploy config servers as replica sets. A sharded cluster with a Config Server Replica Set (CSRS) can continue to process reads and writes as long as a majority of the replica set is available.

In production environments, individual shards should be deployed as replica sets, providing increased redundancy and availability.

Sharding Strategy

Database supports two sharding strategies for distributing data across sharded clusters.

Hashed Sharding

Hashed Sharding involves computing a hash of the shard key field's value. Each chunk is then assigned a range based on the hashed shard key values.

Database automatically computes the hashes when resolving queries using hashed indexes. Applications do not need to compute hashes.

Diagram of the hashed based segmentation.

While a range of shard keys may be "close", their hashed values are unlikely to be on the same chunk. Data distribution based on hashed values facilitates more even data distribution, especially in data sets where the shard key changes monotonically.

However, hashed distribution means that ranged-based queries on the shard key are less likely to target a single shard, resulting in more cluster wide broadcast operations

Ranged Sharding

Ranged sharding involves dividing data into ranges based on the shard key values. Each chunk is then assigned a range based on the shard key values.

Diagram of the shard key value space segmented into smaller ranges or chunks.

A range of shard keys whose values are "close" are more likely to reside on the same chunk. This allows for targeted operations as a DBs can route the operations to only the shards that contain the required data.

The efficiency of ranged sharding depends on the shard key chosen. Poorly considered shard keys can result in uneven distribution of data, which can negate some benefits of sharding or can cause performance bottlenecks. See shard key selection for ranged sharding.

Zones in Sharded Clusters

In sharded clusters, you can create zones of sharded data based on the shard key. You can associate each zone with one or more shards in the cluster. A shard can associate with any number of zones. In a balanced cluster, DataBase migrates chunks covered by a zone only to those shards associated with the zone.

Each zone covers one or more ranges of shard key values. Each range a zone covers is always inclusive of its lower boundary and exclusive of its upper boundary.

Diagram of data distribution based on zones in a sharded cluster

You must use fields contained in the shard key when defining a new range for a zone to cover. If using a compound shard key, the range must include the prefix of the shard key. See shard keys in zones for more information.

When choosing a shard key, carefully consider the possibility of using zone sharding in the future, as you cannot change the shard key after sharding the collection.

Most commonly, zones serve to improve the locality of data for sharded clusters that span multiple data centers.

(2) Fake chain selection and double payment prevention

What exactly is a chain reorganization in Ethereum?

The client software running as the node participates in the consensus algorithm unique to Ethereum: Greedy Heaviest Observed Sub Tree (GHOST) protocol. It accepts a certain chain of blocks as the "truth" at a given point of time after participating with peer nodes in the consensus that communicate via the Ethereum wire protocol. The currently accepted truth is the above chain among the peers that have synchronized so far. It happens so that a few of the peers discover a different version of the chain from a different set of peers that complies with the GHOST protocol and is a stronger case of the truth.

There is a change at block number 623. The hash has changed and so have the subsequent children blocks.

Now the nodes are faced with a fork in the version of the truth. This is purely a high-level view, please note that the actual codebase of the node software doesn't exactly maintain a fork's data structure like this. After further synchronizing among the other peers who accepted the green version as the earlier truth, consensus is reached that since the red fork has _more computation_ done on it, it should be accepted as the current canonical truth.

Once a block or blocks are dropped, the Ethereum State Machine reverts the transactions that were applied so far from those blocks. The work specified in those transactions may or may not exist anymore in the newly accepted version of truth. In case they don't, the DApp is now left with a state inconsistent with the globally accepted version of the Ethereum State Machine and consequently, the final chain of blocks.

How does **'AI Block'** work for setting up the solution on these RISK POINTS?

- System Log(all about Contract transaction Data in Chain) Learning

  Machine learning Algorithm, Adjusting Visualization libraries

  - **Import** Scikit -learn, keras, matlab, numpy,. etc. **libraries**

```
# Load libraries
import numpy as np

# Prediction
from sklearn.linear_model import LogisticRegression
from sklearn.datasets.samples_generator import make_blobs
```

```
# Training a final classification model
from sklearn.linear_model import LogisticRegression
from sklearn.datasets.samples_generator import make_blobs

# Making multiple probability predictions
from sklearn.linear_model import LogisticRegression
from sklearn.datasets.samples_generator import make_blobs

# Training a final regression model
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_regression

# Importing Keras Sequential Model
from keras.models import Sequential
from keras.layers import Dense

# For visualization
import pandas
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

- **Check** Algorithm **& Evaluate** Each model

```
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))

# evaluate each model in turn
results = []
names = []


for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=seed)
```

```
    cv_results = model_selection.cross_val_score(model, X_train,
Y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)

.
.
.
```

- **Training** models

```
# Training a classification model
# generate 2d classification dataset
X, y = make_blobs(n_samples=100, centers=2, n_features=2,
random_state=1)
# fit final model
model = LogisticRegression()
model.fit(X, y)

# new instances where we do not know the answer
Xnew, _ = make_blobs(n_samples=3, centers=2, n_features=2,
random_state=1)


# Setting layer depth to Solving the Multi Layer Perceptron problem
log.debug("Enter the two values for input layers")
log.debug('a = ')
a = int(input())
log.debug('b = ')
b = int(input())

input_data = np.array([a, b])
weights = {
    'node_0': np.array([1, 1]),
    'node_1': np.array([-1, 1]),
    'output_node': np.array([2, -1])
}

node_0_value = (input_data * weights['node_0']).sum()
log.debug('node 0_hidden: {}'.format(node_0_value))

node_1_value = (input_data * weights['node_1']).sum()
log.debug('node_1_hidden: {}'.format(node_1_value))

hidden_layer_values = np.array([node_0_value, node_1_value])
```

```
output_layer = (hidden_layer_values * weights['output_node']).sum()
log.debug("output layer : {}".format(output_layer))

.
.
.
```

- **Risk** Prediction
  - **Prediction** result

```
#Prediction
# make a prediction
ynew = model.predict(Xnew)
# show the inputs and 1st predicted outputs
for i in range(len(Xnew)):
    print("X=%s, Predicted=%s" % (Xnew[i], ynew[i]))

# training a regression model
# generate regression dataset
X, y = make_regression(n_samples=100, n_features=2, noise=0.1)
# fit final model
model = LinearRegression()
model.fit(X, y)
# new instances where we do not know the answer
Xnew, _ = make_regression(n_samples=3, n_features=2, noise=0.1,
random_state=1)

.
.
.

ynew = model.predict(Xnew)
# show the inputs and 2nd and final predicted outputs
for i in range(len(Xnew)):
    print("X=%s, Predicted=%s" % (Xnew[i], ynew[i]))
```

  - **Comparing** Algorithms and **Find** The best Suitable Algorithm

```
# Comparing Algorithms
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
```

```
ax.set_xticklabels(names)
plt.show()

plt.figure(figsize=(10,6), dpi=80)
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-",
label="cosine")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-",
label="sine")


# Make predictions on validation dataset
knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)
predictions = knn.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))


# making multiple probability predictions
# generate 2d classification dataset
X, y = make_blobs(n_samples=100, centers=2, n_features=2,
random_state=1)
# fit final model
model = LogisticRegression()
model.fit(X, y)

# new instances where we do not know the answer
Xnew, _ = make_blobs(n_samples=3, centers=2, n_features=2,
random_state=1)

.
.
.

ynew = model.predict_proba(Xnew)
# show the inputs and predicted probabilities
for i in range(len(Xnew)):
    print("X=%s, Predicted=%s" % (Xnew[i], ynew[i]))
```

- System **Enhancement**
  - **Enhancement** of Model

```
# Model Enhancement
# Initializing the seed value to a integer.
seed = 7
numpy.random.seed(seed)

# Loading the data set (SmartContract transaction log_Dataset)
dataset =
numpy.loadtxt('datasets/smartcontract_transactionData_20180704102558.
log', delimiter="\t")

# Loading the input values to X and Label values Y using slicing.
X = dataset[:, 0:8]
Y = dataset[:, 8]

# Initializing the Sequential model from KERAS.
model = Sequential()

# Creating a 16 neuron hidden layer with Linear Rectified activation
function.
model.add(Dense(16, input_dim=8, init='uniform', activation='relu'))

# Creating a 8 neuron hidden layer.
model.add(Dense(8, init='uniform', activation='relu'))

# Adding a output layer.
model.add(Dense(1, init='uniform', activation='sigmoid'))


# Compiling the model
model.compile(loss='binary_crossentropy',
              optimizer='adam', metrics=['accuracy'])
# Fitting the model
model.fit(X, Y, nb_epoch=150, batch_size=10)
scores = model.evaluate(X, Y)
log.debug("%s: %.2f%%" % (model.metrics_names[1], scores[1] * 100))
log.debug("Eccuracy %s/200 %.8f%%" % (model.metrics_names[2],
scores[2])

.
.
.
```

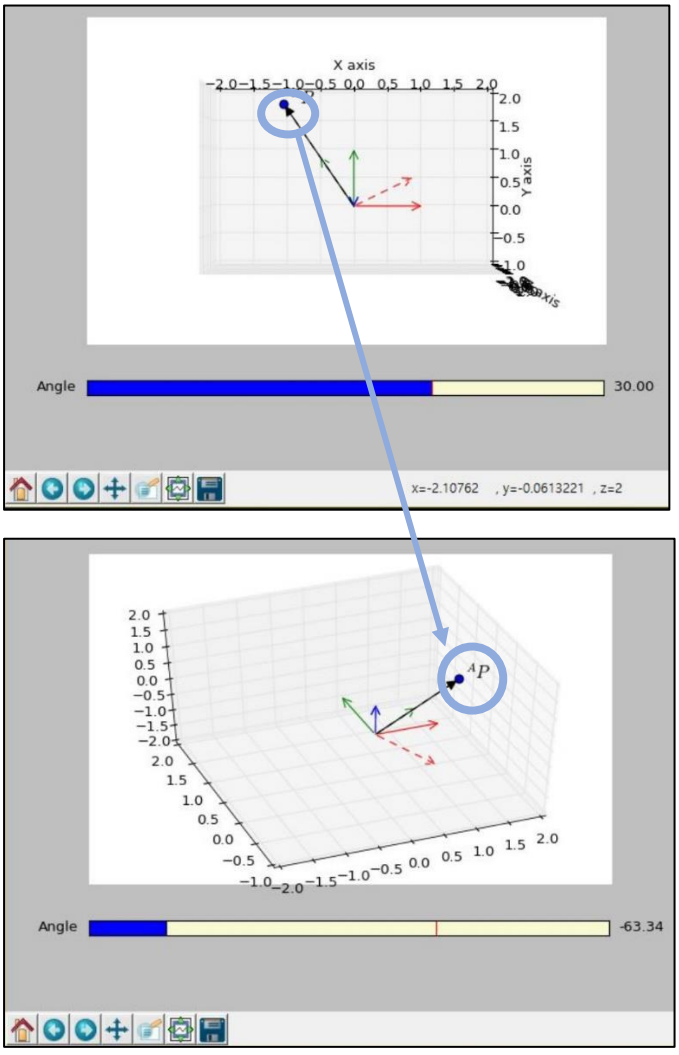
- **For Result** Visualization
  - **Visualization** of a Fined Data

```
#Visualization
# Set limits
plt.xlim(X.min()*1.1, X.max()*1.1)
plt.ylim(C.min()*1.1, C.max()*1.1)

# Setting tick labels
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi], [r'$-\pi$', r'$-
\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
plt.yticks([-1, 0, +1], [r'$-1$', r'$0$', r'$+1$'])

# Moving spines
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))
.
.
.

# Adding a legend
plt.legend(loc='upper left', frameon=False)

# Annotate some points
t = 2*np.pi/3
plt.plot([t,t],[0,np.cos(t)], color ='blue', linewidth=2.5,
linestyle="--")
plt.scatter([t,],[np.cos(t),], 50, color ='blue')

plt.annotate(r'$\sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
             xy=(t, np.sin(t)), xycoords='data',
             xytext=(+10, +30), textcoords='offset points',
fontsize=16,
             arrowprops=dict(arrowstyle="->",
connectionstyle="arc3,rad=.2"))

plt.plot([t,t],[0,np.sin(t)], color ='red', linewidth=2.5,
linestyle="--")
plt.scatter([t,],[np.sin(t),], 50, color ='red')
```

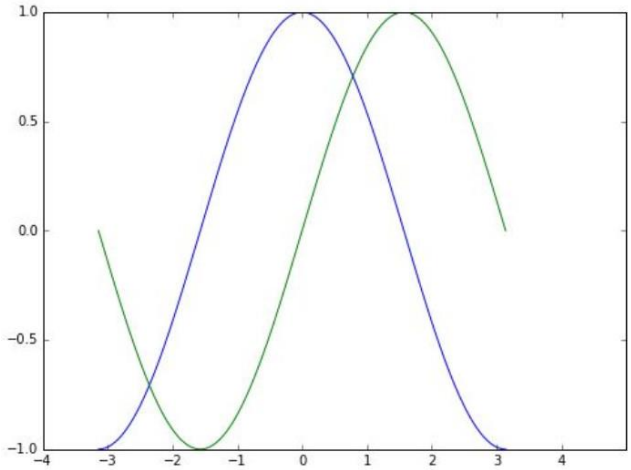
```
plt.annotate(r'$\cos(\frac{2\pi}{3})=-\frac{1}{2}$',
             xy=(t, np.cos(t)), xycoords='data',
             xytext=(-90, -50), textcoords='offset points',
fontsize=16,
             arrowprops=dict(arrowstyle="->",
connectionstyle="arc3,rad=.2"))
.
.
.
# keypoints (for accuracy of Visualization)
for label in ax.get_xticklabels() + ax.get_yticklabels():
    label.set_fontsize(16)
    label.set_bbox(dict(facecolor='white', edgecolor='None',
alpha=0.65 ))
```

- **Visualized Output Data(1)**

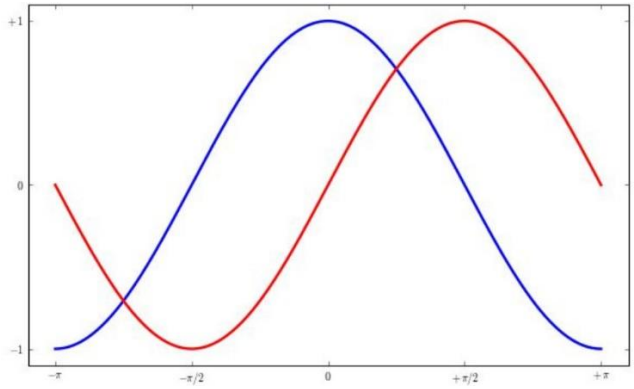
  (with MatplotLib Slider widget)



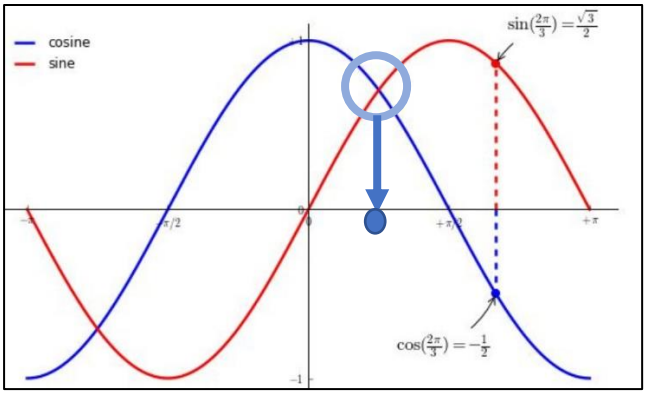
- **Visualized Output Data(2)**



➔ **First training for Model Classification**



➔ **Repetitive training for Model Classification *(near the Result)***



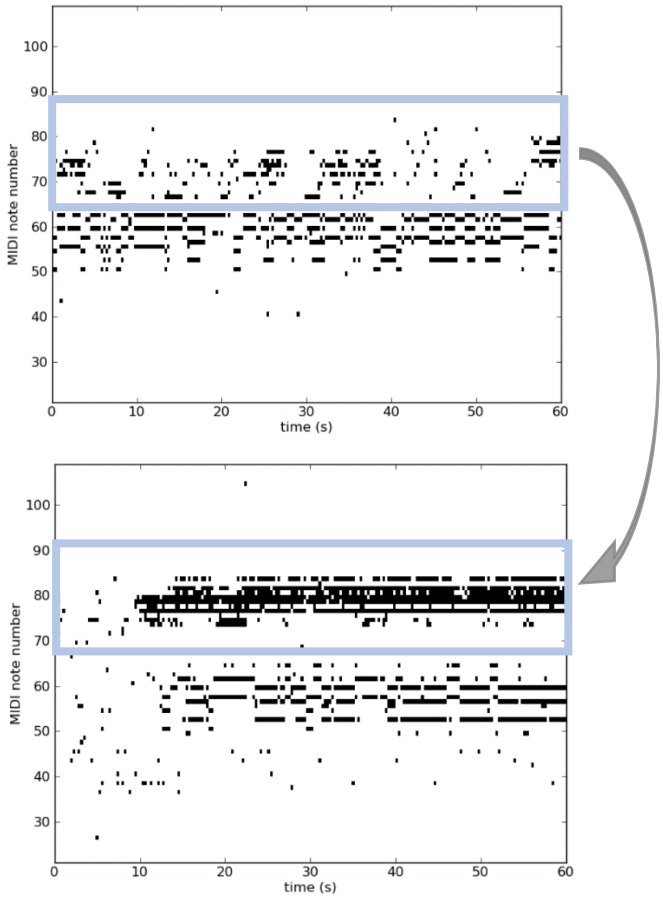
➔ **Adjust graph to find the Accurate Point**

- **The Output Data(3)**

Eccuracy *1/200* ***-15.0308940028***
Eccuracy 2/200 -10.4892606673
Eccuracy 3/200 -10.2394696138
Eccuracy 4/200 -10.1431669994
Eccuracy 5/200 -9.7005382843
Eccuracy 6/200 -8.5985647524
Eccuracy 7/200 -8.35115428534
Eccuracy 8/200 -8.26453580552
Eccuracy 9/200 -8.21208991542
Eccuracy 10/200 -8.16847274143
.
..
... truncated ...
.
..
Eccuracy 190/200 -4.74799179994
Eccuracy 191/200 -4.73488515216
Eccuracy 192/200 -4.7326138489
Eccuracy 193/200 -4.73841636884
Eccuracy 194/200 -4.70255511452
Eccuracy 195/200 -4.71872634914
Eccuracy 196/200 -4.7276415885
Eccuracy 197/200 -4.73497644728
Eccuracy 198/200 -inf
Eccuracy 199/200 -4.75554987143
Eccuracy *200/200* ***-4.72591935412***

- **The Output Data(4)**

(Finding Accurate Points of Data using MIDI packages_Graph Licence)



2. Identify user computing power and classify appropriate node creation, for providing service according to each classified node.

   DAI learns user's computing power in advance and groups users' node creation ability. For example, a computing power user who can create a node with a large capacity and a computing power user who can generate a small node have a classification and storage. Based on this, a differentiated business model can be constructed according to each node capability, and a profitable service can be provided.