**CBDC TokenAI – UTXO**

**Contents**

1. Introduction
2. Problem Statement
3. solution
4. POC (LSTM)
5. Architecture
6. Impact UTXO Components
7. Tech Stack
8. AI Model
9. Model Training
10. APIs
11. DB Schema – Tokens Value storation
12. Token Selection Logic

## **Introduction**

The project develops a Deep Learning time-series prediction model to estimate future UTXO transaction amounts, which are then used to determine the optimal set of token denominations. These predicted tokens are pre-created and stored in the **DTSP**'s **MariaDB** instance, ensuring that tokens are readily available when an Off-Us transfer request is initiated without facing token selection deadlock.

## **Problem Statement**

In an **UTXO Off-Us transfer request**, the system attempts to select high-value tokens from the available pool to satisfy the transaction amount. Once selected, these tokens are marked as **LOCKED** to prevent concurrent access and ensure atomicity of the transfer. However, this locking mechanism introduces a bottleneck during high-traffic times.

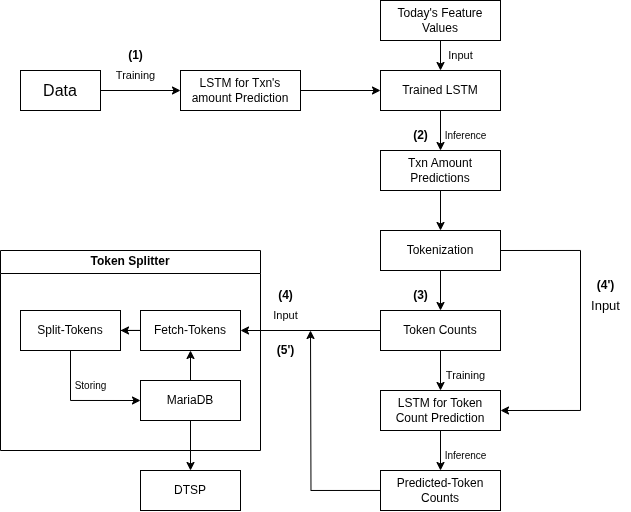
When multiple transactions simultaneously request high-value tokens—especially during peak loads tokens required by one transaction may already be locked by another. As a result, subsequent transactions are forced to wait until the locked tokens are released, leading to a **token deadlock**. This causes a **queue buildup**, where transactions are stalled, waiting for token availability.

## **Solution**

To address the challenges of token contention and transfer delays in CBDC UTXO’s Off-Us txns an AI model is employed to **predict upcoming transaction values** using historical transaction data. This model leverages temporal features to forecast the expected transactions for future transfers.

Based on these predictions, the system splits **tokens in advance** for better availability. Specifically, when the model anticipates high-value transactions, it splits large-value tokens into multiple smaller denominations. These smaller tokens are then stored in the **MariaDB**, ensuring they are readily available for future use.

## **POC (LSTM-based)**

The POC is a prediction pipeline for txn amount forecasting and token splitting using synthesized txn data. The POC begins with feature engineering on synthetically generated txn-al records using NumPy including temporal features such as hour\_of\_day, day\_of\_week, week\_of\_year, month\_of\_year, and transaction\_amount\_lag\_n to train an LSTM model for predicting future txn amounts. The predicted txn amount is then passed through a tokenization module, which determines the number of tokens required for the txn.

Two processing flows were explored:

#### **Flow (a): Direct Token Count Usage**

1. **Training**: An LSTM model is trained on historical transaction data to learn patterns in transaction amounts.
2. **Inference**: The trained model is used to predict the next transaction amount based on current feature values.
3. **Tokenization**: The predicted transaction amount is tokenized to compute token counts.

For example if the txn prediction LSTM predicts 489 rupees

We tokenize the txn amount into 400, 80 and 9 and their counts are computed which are then stored into a dictionary as key-value pairs respectively such as.

(TokenValue, Count)

(50, 3)

(100, 2)

(400, 1)

(80, 1)

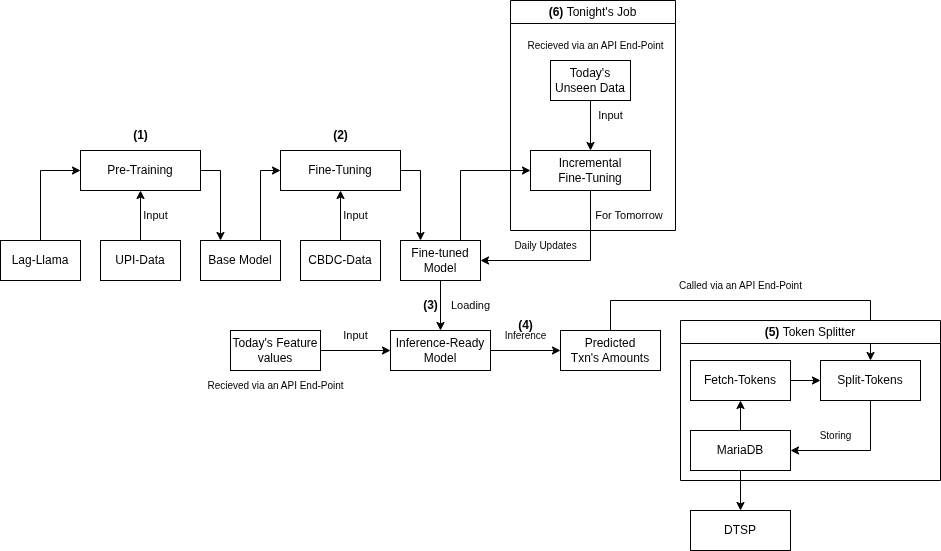
(9, 1)

1. **Token Splitting**: These token counts are directly passed to a token splitter, which fetches tokens from MariaDB, splits them accordingly, and stores the newly created tokens back into the database.

#### **Flow (b): Token Count Prediction via LSTM**

1. **Training**: Same as Flow (a), an LSTM is trained on transaction amounts.
2. **Inference**: Predict the transaction amount using the trained model.
3. **Tokenization**: Token counts are derived from the predicted transaction amount.
4. **Token Count Prediction**: These token amounts and token counts from the dictionary are used as input to a second LSTM model trained to predict token counts.
5. **Token Splitting**: The predicted token counts are then passed to the token splitter, which interacts with MariaDB to fetch, split, and persist the tokens.

## **Architecture**



### **Model Architecture and Inference Pipeline**

The time series prediction system is built on **Lag-LLama,** which is a general-purpose open-source foundation model for univariate probabilistic time series forecasting based on a decoder-only transformer architecture that uses lags as covariates

1. **Pre-training**:  
   The foundational Lag-LLama model is pre-trained on large-scale feature engineered UPI transaction data to capture the temporal payment patterns of India. This stage gives us a base-model which has learned the foundational representations relevant to UPI transaction sequences.
2. **Fine-Tuning**:  
   The base-model pre-trained on UPI is then fine-tuned on feature engineered CBDC UTXO txns data specifically the bank-wise Off-Us txns data to adapt the model to the underlying temporal patterns of each bank’s CBDC UTXO. This newly fine-tuned model will now be able to predict the future day’s txns
3. **Model Deployment**:  
   The fine-tuned model is deployed into the production environment as an inference-ready service. It is exposed via an API endpoint that accepts daily-time feature inputs and returns predicted transaction amounts.
4. **Inference Workflow**:  
   The deployed model inputs daily feature vectors including:
   1. hour\_of\_day
   2. day\_of\_week
   3. week\_of\_year
   4. month\_of\_year
   5. transaction\_amount\_lag\_n (lagged transaction values)  
      These features are used to generate **predicted transaction amounts**, which are then passed to the **token splitter** module.
5. **Token Splitter**:  
   The token splitter fetches tokens from **MariaDB**, splits the tokens into smaller tokens denominations based on the predicted transaction amount that the fine-tuned model outputs and then writes back the new tokens into the DB.
6. **Scheduled Incremental Fine-Tuning**:  
   A nightly **cron job** is scheduled to perform incremental fine-tuning of the model using the latest unseen transaction data from the current day. This daily update loop ensures that the model remains adaptive to the short-term behaviour without forgetting the long-term trends. This new model will then have to be re-deployed into the production environment after every incremental fine-tuning job.

**API Integration**:  
Both the **predicted transaction amounts** and the **resulting token counts** are managed through a dedicated API layer.

## **Impact UTXO components**

API

CHAINCODE

MARIADB

DLT

## **Tech Stack**

Lag Llama

Python

Fast API

Actix web

Sqlx

Angular

Mariadb

## **APIs**

Token Accept api

------------------------

Description - Predicted token is going to submit to RUST Api to store into DB

Purpose – Store token detail into DB

Type - POST

Req Body - {

tokens: ["10","23","1","0.5","34"] // no empty

}

Get Token For UI Display

---------------------------------

Description - Display the predicted tokens in the Angular Dashboard

Purpose – Display Tokens

Type – GET

Resp Body -

[{

"token\_id":"72426d00-3a54-41f7-8dcc-5db0d8b42a96", // no empty

"tokens\_frequency":"100", // no empty

"value":"10", // no empty

"Remarks":"Model predicted/ utxo submitted", // no empty

"status": "predicted/pending\_creation/created/failed" // no empty

},

{

"token\_id":"72426d00-3a54-41f7-8dcc-5db0d8b42a96", // no empty

"tokens\_frequency":"90", // no empty

"value":"78", // no empty

"Remarks":"Model predicted/ utxo submitted", // no empty

"status": "predicted/pending\_creation/created/failed"// no empty

}

]

Save Token Creation Request’s Meta Information

-----------------------------------------------------------------

Description – Token creation request from UI’s meta information is store into DB

Purpose – Token Creation Request Store

Type – POST

Req Body - {

"token\_ids":["72426d00-3a54-41f7-8dcc-5db0d8b42a96","72426d00-3a54-41f7-8dcc-5db0d8b42a96","72426d00-3a54-41f7-8dcc-5db0d8b42a96"] // no empty,

"username":"rajeev", // no empty

"creation\_time": "2017-07-23 13:10:11"// no empty

}

Post Request to CBDC API

---------------------------------------------

Description - UI is triggering a event to CBDC api to create token

Purpose – Initiate Token creation Event

Type - POST

Req Body - {

token\_ids: "[{72426d00-3a54-41f7-8dcc-5db0d8b42a96}, {72426d00-3a54-41f7-8dcc-5db0d8b42a96}, {72426d00-3a54-41f7-8dcc-5db0d8b42a96}, {72426d00-3a54-41f7-8dcc-5db0d8b42a96}, {72426d00-3a54-41f7-8dcc-5db0d8b42a96}]" // no empty

timestamp: "2017-07-23 13:10:11", // no empty

total\_value: "123456789", //no empty

total\_token\_to\_be\_create: "1347", // no empty

}

ACK : - {

"Ack":"Token Creation Initiated", // no empty

}

Once got the ACK, RUST Api change the status of the tokens to pending creation and waiting for the final Response CREATED/FAILED.

Also change the Remarks to utxo\_submitted.

Fetch Tokens by CBDC API from MODEL Storage DB

---------------------------------------------------------------------

Description - A db connection to fetch the tokens value to be create and Stored into DTSP Mariadb and DLT.

Purpose – Fetch Token Values

Type - DB Connection

SELECT \* from vtokens\_values where creation\_id=”**72426d00-3a54-41f7-8dcc-5db0d8b42a96**” ;

User Management

----------------------------

Description - Users who are going to login into the ui to create/read tokens.

Purpose – User Management

Type - CRUD User Management APIs

Roles – Admin/Read/Others, these roles will be hardcoded as of now.

## **DB Schema**

Vtoken\_values

|  |  |  |
| --- | --- | --- |
| **creation\_id** | VAR CHAR (primary KEY) | Not NULL |
| **tokens\_frequency** | int | Not NULL |
| **value** | int | Not NULL |
| **predict\_time** | timestamp | Not NULL |
| **remarks** | text | Not NULL |
| **status** | text | Not NULL |

Predifined “Status” Value -

Predicted – Initial Time when model predicted value

Pending\_creation – token creation request successfully submitted to UTXO API.

Created - Once token created and stored in MARIADB & DLT

Failed – If API is not able to create/store and middle stop to creation.

Predifined “Remarks” Value -

Model Predicted – Initial When tokens are predicted by model

Utxo Submitted – Whem RUST API call to UTXO API for token creation

Meta\_information (Ui Trigger)

|  |  |  |
| --- | --- | --- |
| **id** | varchar(primary KEY) | Not NULL |
| **username** | varchar | Not NULL |
| **creation\_time** | timestamp | Not NULL |
| **creation\_ids** | json | Not NULL |

Users

|  |  |  |
| --- | --- | --- |
| **id** | varchar(primary) | Not NULL |
| **username** | varchar | Not NULL |
| **password** | varchar | Not NULL |
| **role** | text | Not NULL |

Validation in ui that correct value from ui and after select from db’s tokens value are same

Rust Actix Web Server.

Responsibilities.

* Retrieve data from the CBDC transactions database.
* Endpoint to process and send today transactions to Inference Model endpoint.
* Model predictions are stored in dedicated database and will be pushed to pre-token VP-UTXO-UI.
* CronJob will pull data from this server to fine-tune the model.
* Maintain security in each stage.
* Maintain metrics of the system.

