Giza Weather Game

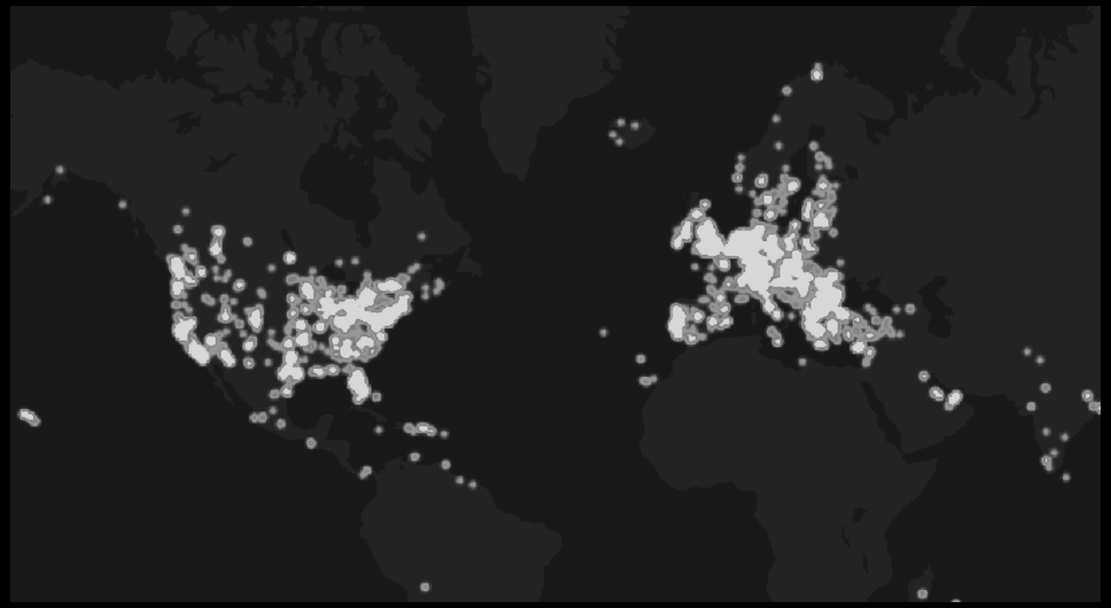
Powered By Giza&Starknet

WHAT WE BUILT

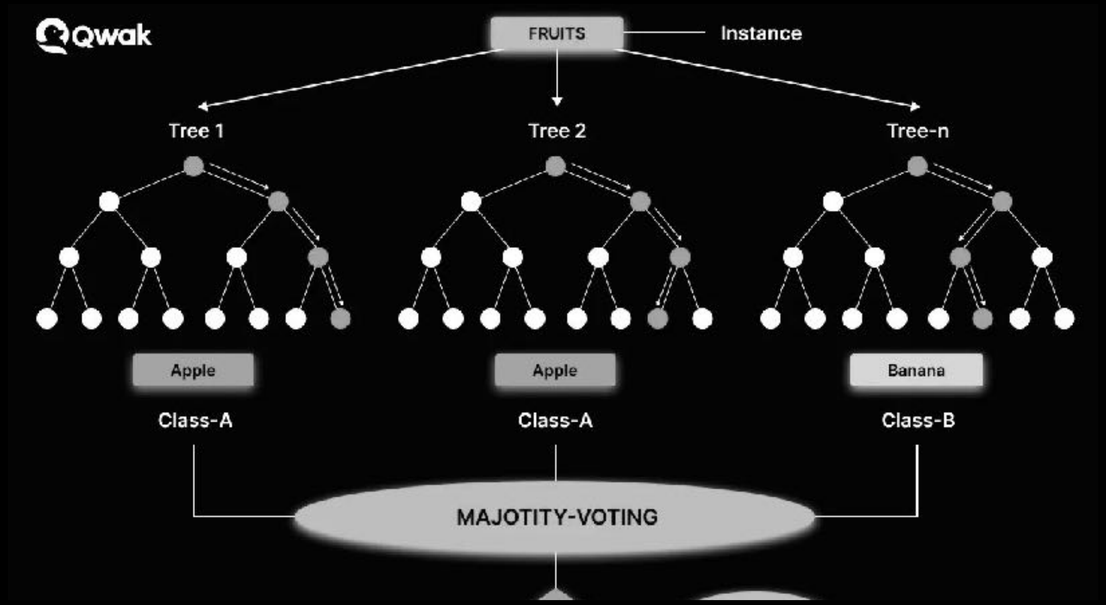
- A bet game platform for predict weather (using depin and zkml)

HOW IT WORKS

- Get the raw data from WeatherXM Data Index, organize and clean data prepare for training.



- Create and Train an XGBoost Model with data. (‘temperature', 'humidity', 'wind_speed', 'pressure', ‘precipitation')



HOW IT WORKS

- Processed Data

WeatherXMData.csv

```csv
temperature,humidity,wind_speed,pressure,has_precipitation,next_day_precipitation
20.1,74.0,0.0,1005.37,False,False
18.4,61.0,0.39,1010.48,False,False
9.8,90.0,0.0,973.89,False,False
16.6,77.0,0.9,1002.43,False,False
25.2,67.0,0.39,1003.49,False,False
20.0,60.0,0.71,982.91,False,False
15.4,82.0,0.64,929.79,False,False
19.0,62.0,2.25,1013.75,False,False
6.7,78.0,0.32,1001.06,False,False
15.8,89.0,0.0,992.16,False,False
14.7,93.0,5.35,1007.83,False,False
15.9,54.0,3.8,989.48,False,False
12.7,38.0,0.97,785.47,False,False
20.8,58.0,0.64,995.45,False,False
29.4,57.0,0.64,908.41,False,False
9.8,87.0,0.0,862.89,False,True
14.4,95.0,0.0,980.72,True,False
15.1,77.0,0.0,988.0,False,False
14.2,74.0,0.0,1003.1,False,False
31.5,56.0,3.16,940.09,False,False
30.2,49.0,0.26,1016.31,False,False
12.5,95.0,0.19,1008.35,False,True
13.8,96.0,0.0,980.94,True,False
14.8,75.0,0.0,979.86,False,False
10.7,99.0,0.0,1003.65,False,False
16.7,61.0,0.19,1012.2,False,False
10.7,98.0,0.26,1016.59,False,False
15.9,85.0,0.0,985.06,False,False
14.3,99.0,0.71,1005.12,False,False
20.3,66.0,1.16,979.26,False,False
28.3,86.0,0.45,1016.54,False,False
22.4,58.0,0.0,981.05,False,True
12.5,98.0,0.0,981.97,True,False
18.2,86.0,0.0,991.22,False,True
16.8,87.0,3.35,1003.3,True,False
11.8,88.0,0.06,1013.37,False,False
14.9,84.0,0.64,975.21,False,True
13.4,99.0,0.71,990.41,True,False
19.6,70.0,1.22,1013.66,False,False
13.60000038,87.0,0.0,997.2025146,False,False
27.2,50.0,0.13,1013.94,False,False
20.6,69.0,0.0,995.17,False,False
```

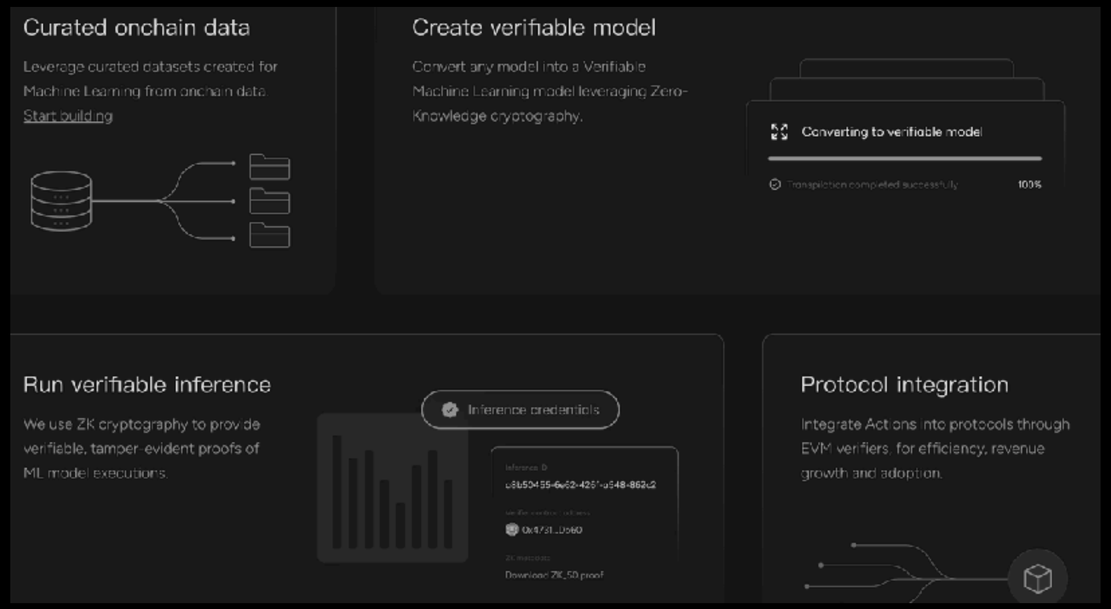
- XGBoost Model

train_xgboost.py > ...

```python
## Create and Train an XGBoost Model
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import numpy as np

df = pd.read_csv('WeatherXMData.csv')

features = ['temperature', 'humidity', 'wind_speed', 'pressure', 'has_precipitation']
target = 'next_day_precipitation'

X = df[features].values
y = df[target].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

n_estimators = 10
max_depth = 6

model = xgb.XGBRegressor(n_estimators=n_estimators, max_depth=max_depth)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')


## Save the model
from giza.zkcook import serialize_model
serialize_model(model, "xgb_weather.json")
```

HOW IT WORKS

- Transpile model to Orion Cairo with Giza(starknet) and deploy it on Giza Platform.



- Write a prediction game smart contract(Cairo / Solidity) and deploy on Sepolia.

Amount	1X	3X	9X	27X	81X	Fund	243X	729X	Emergency Fund
₹10	₹10	₹30	₹90	₹270	₹810	[illegible]	₹2,430	₹7,290	₹10,930
₹20	₹20	₹60	₹180	₹540	₹1,620	₹2,420	₹4,860	₹14,580	₹21,860
₹30	₹30	₹90	₹270	₹810	₹2,430	₹3,630	₹7,290	₹21,870	₹32,790
₹40	₹40	₹120	₹360	₹1,080	₹3,240	₹4,840	₹9,720	₹29,160	₹43,720
₹50	₹50	₹150	₹450	₹1,350	₹4,050	[illegible]	₹12,150	₹36,450	₹54,650
₹60	₹60	₹180	₹540	₹1,620	₹4,860	₹7,260	₹14,580	₹43,740	₹65,580
₹70	₹70	₹210	₹630	₹1,890	₹5,670	₹8,470	₹17,010	₹51,030	₹76,510
₹80	₹80	₹240	₹720	₹2,160	₹6,480	₹9,680	₹19,440	₹58,320	₹87,440
₹90	₹90	₹270	₹810	₹2,430	₹7,290	₹10,890	₹21,870	₹65,610	₹98,370
₹100	₹100	₹300	₹900	₹2,700	₹8,100	[illegible]	₹24,300	₹72,900	₹109,300
₹500	₹500	₹1,500	₹4,500	₹13,500	₹40,500	₹60,500	₹121,500	₹364,500	₹546,500

HOW IT WORKS

- Transpile Deploy Test Verify

verifiable_inference.py > ...

```python
import xgboost as xgb
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
import pandas as pd

from giza.agents.model import GizaModel


MODEL_ID = 675  # Update with your model ID
VERSION_ID = 4  # Update with your version ID

def prediction(input, model_id, version_id):
    model = GizaModel(id=model_id, version=version_id)

    (result, proof_id) = model.predict(
        input_feed={"input": input}, verifiable=True, model_category="XGB"
    )

    return result, proof_id


def execution():
    # The input data type should match the model's expected input
    input = X_test[1, :]

    print(input)
    print(X_test)

    (result, proof_id) = prediction(input, MODEL_ID, VERSION_ID)

    print(f"Predicted value for input {input} is {result}")

    return result, proof_id


if __name__ == "__main__":
    df = pd.read_csv('WeatherXMData.csv')

    features = ['temperature', 'humidity', 'wind_speed', 'pressure', 'has_precipitation']
    target = 'next_day_precipitation'

    X = df[features].values
    y = df[target].values
```

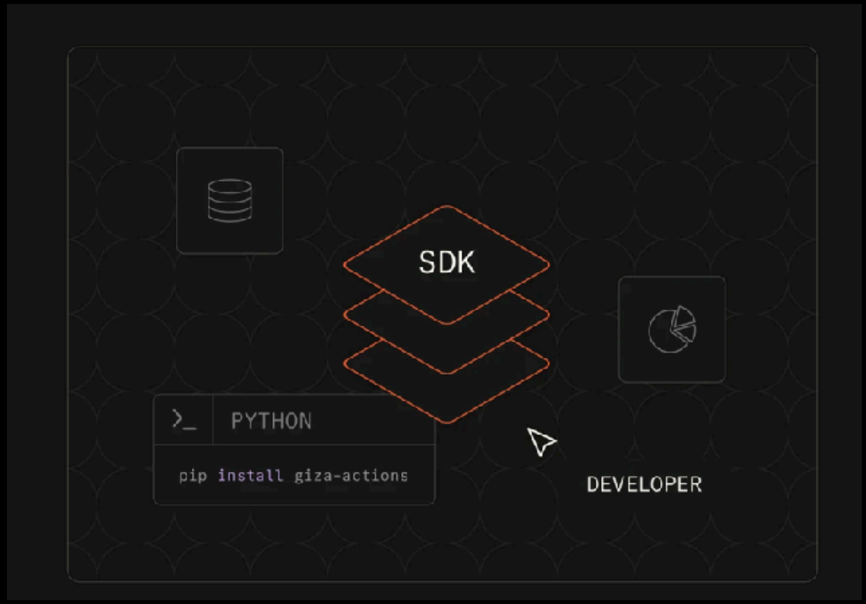
- Smart Contract(Cairo)

src > GizaWeatherGame.cairo

```cairo
use starknet::ContractAddress;

const OWNER_ROLE: felt252 = selector!("OWNER_ROLE");

#[derive(Serde, Drop, starknet::Store)]
struct Round {
    round_id: u64,
    start_timestamp: u64,
    duration_timestamp: u64,
    end_timestamp: u64,
    bet_award: u64,
    is_over: bool,
    is_rain: bool,
}

#[derive(Serde, Drop, starknet::Store)]
struct Bet {
    is_participated: bool,
    is_bet_rain: bool,
    is_over: bool,
}


#[starknet::interface]
trait IGWG<TContractState> {
    fn create_round(ref self: TContractState, start_timestamp: u64, probability: u64);
    fn place_bet(ref self: TContractState, round_id: u64, prediction: bool);
    fn over_round(ref self: TContractState, round_id: u64, is_rain: bool);
    fn claim_reward(ref self: TContractState, round_id: u64);
    fn grant_owner(ref self: TContractState, new_owner: ContractAddress);

    fn get_current_round_id(self: @TContractState) -> u64;
    fn get_duration_interval(self: @TContractState) -> u64;
    fn get_settlement_interval(self: @TContractState) -> u64;
    fn get_multi(self: @TContractState) -> u64;
    fn get_round(self: @TContractState, round_id: u64) -> Round;
    fn get_user_bet(self: @TContractState, user: ContractAddress, round_id: u64) -> Bet;

    fn get_rounds(self: @TContractState, round_ids: Array<u64>) -> Array<Round>;
    fn get_bets(self: @TContractState, user: ContractAddress, round_ids: Array<u64>) -> Array<Bet>;
}
```

HOW IT WORKS

- Automatically obtain weather data (with WeatherXM api) and feed it to the model to get the rain probability, and then upload it to the sepolia to create predictions round with AI agent(Giza).



- User has fun with this via frontend(next+ starknet-react)

HOW IT WORKS

- AI Agent

create_prediction_round_starknet.py > get_current_data_from_WeatherXM

```python
# main ai agent flow
async def main():
    account = Account(
        address=address,
        client=client,
        key_pair=KeyPair.from_private_key(private_key),
        chain=StarknetChainId.SEPOLIA,
    )

    contract = await Contract.from_address(provider=account, address=contract_address)
    agent = GizaAgent.from_id(
        id=agent_id
    )

    prediction = agent.predict(input_feed={"input": get_current_data_from_WeatherXM()}, verifiable=True, mod

    p = int(prediction.value * 1000)
    ts = int(time.time())

    print('prediction', p, "timestamp", ts)

    [current_round_id,] = await contract.functions["get_current_round_id"].call()
    print("before create round current round id", current_round_id)

    try:
        invocation = await contract.functions["create_round"].invoke_v1(
            start_timestamp= ts, probability=p, max_fee=int(1e15)
        )
        await invocation.wait_for_acceptance()
    except:
        print('---')

    [current_round_id,] = await contract.functions["get_current_round_id"].call()
    print("after create round current round id", current_round_id)



# schedule task
while True:
    asyncio.run(main())
    time.sleep(60 * 60 * 24)
```

(parameter) provider: BaseAccount | Client

provider: BaseAccount or Client.

- Frontend

packages > nextjs > app > prediction > page.tsx > Prediction

```typescript
const Prediction: NextPage = () => {

  const [queryRoundIds, setQueryRoundIds] = useState([6, 5, 2, 3, 4]);

  const { data: roundsData } = useContractRead({
    address: gwgContractAddress,
    abi: GWGABI,
    functionName: "get_rounds",
    args: [queryRoundIds],
    watch: true,
    blockIdentifier: "pending" as BlockNumber,
  });

  const { data: betsData } = useContractRead({
    address: gwgContractAddress,
    abi: GWGABI,
    functionName: "get_bets",
    args: [owner as string, queryRoundIds],
    watch: true,
    blockIdentifier: "pending" as BlockNumber,
  });

  useEffect(() => {
    if (roundsData && Array.isArray(roundsData)) {
      const parseData = (roundsData as any[]).map((round) => ({
        roundId: Number(round.round_id),
        startTimestamp: Number(round.start_timestamp),
        durationTimestamp: Number(round.duration_timestamp),
        endTimestamp: Number(round.end_timestamp),
        betAward: Number(round.bet_award),
        isOver: round.is_over,
        isRain: round.is_rain,
      }));
      setRounds(parseData as Round[]);
      console.log(parseData);
    }
  }, [roundsData]);

  useEffect(() => {
    if (betsData && Array.isArray(betsData)) {
      const parseData = (betsData as any[]).map((bet) => ({
        isParticipated: bet.is_participated,
        isBetRain: bet.is_bet_rain,
```

Demo