دانشکده‌ی مهندسی کامپیوتر
فرزام کوهی رونقی - ۴۰۱۱۰۶۴۰۳
ثنا بابایان ونستان - ۴۰۱۱۰۵۶۸۹

مدارهای واسط

پاییز ۱۴۰۳

مستند پروژه مدارهای واسط

۱ مقدمه و تعریف مسئله

در این پروژه قصد داریم یک شبکه عصبی که قصد پیشبینی مقدار SOC را با ورودی‌های Current Voltage، و Temperature دارد، روی یک بورد آردوئینو پیاده کنیم. سپس با استفاده از پروتکل‌های مختلف ممکن، داده ورودی را برای inference به بورد انتقال داده و خروجی را از طریق همان پروتکل دریافت کنیم و زمان به طول انجامیده آنها را مقایسه کنیم. نکته قابل توجه این است که از آنجا که بورد ESP32 به جای بورد آردوئینو در دسترس بود، به جای آردوئینو از ESP32 استفاده کردیم.

۲ گام‌های پروژه

برای پیاده سازی پروژه، گام‌هایی باید طی شوند که به شکل زیر هستند

۱. پیاده سازی شبکه عصبی خواسته شده در مقاله State-of-Charge Estimation of Li-Ion Battery in Electric Vehicles: A Deep Neural Network Approach

۲. تبدیل مدل شبکه عصبی ساخته شده به فرمت h. برای استفاده در کد Arduino IDE

۳. پیاده سازی شبکه عصبی روی بورد ESP32 با استفاده از کتابخانه TensorFlowLite_ESP32

۴. پیاده سازی پروتکل UART (Serial) برای دریافت و ارسال داده از/به ESP32

۵. پیاده سازی پروتکل WiFi برای دریافت و ارسال داده از/به ESP32

۶. پیاده سازی پروتکل BLE برای دریافت و ارسال داده از/به ESP32

۷. مقایسه پروتکل‌های مختلف از نظر زمانی

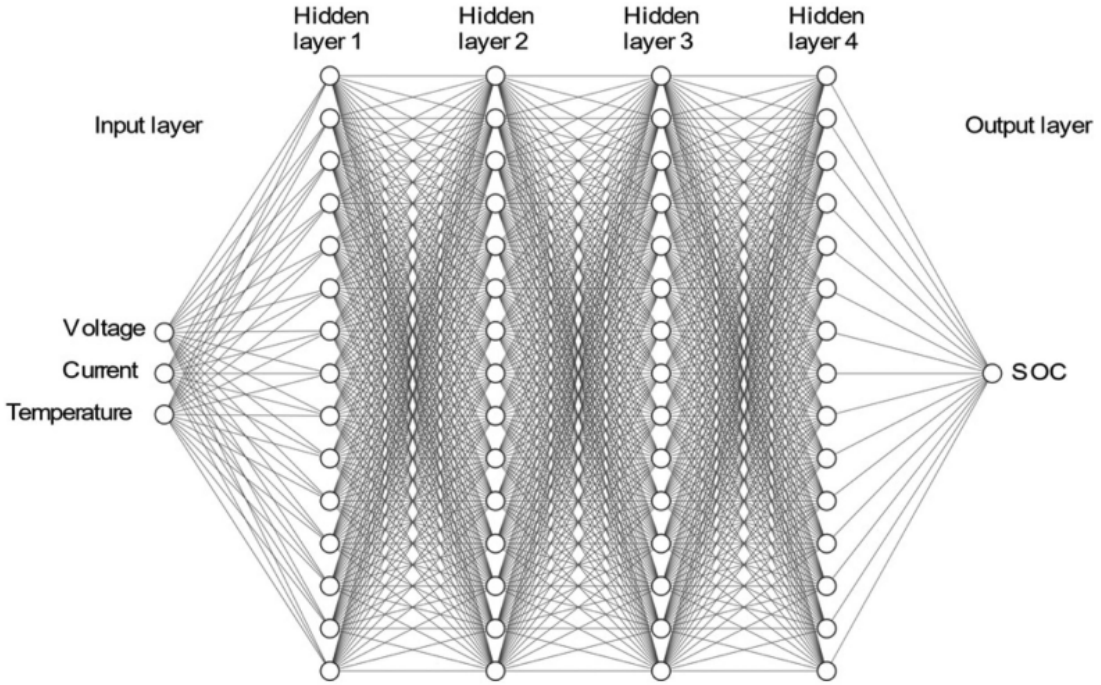
۸. تهیه گزارش

۳ گزارش پیاده سازی شبکه عصبی

۱.۳ پیاده سازی شبکه عصبی مقاله گفته شده

مقاله گفته شده، قصد دارد که SOC باتری را از روی مقادیر ولتاژ، جریان و دما پیش بینی کند. نکته حائز اهمیت این است که مقدار دقیق SOC به طور ویژه در فایل‌های دیتاست موجود نیست و باید مقدار آن را مطابق فرمول گفته شده در مقاله بدست آورد.

یک نگاه به ستون‌های فایل دیتاست می‌اندازیم. دیتاست‌ها از این لینک بدست آمده است.



شکل ۱: شبکه عصبی بیان شده در مقاله

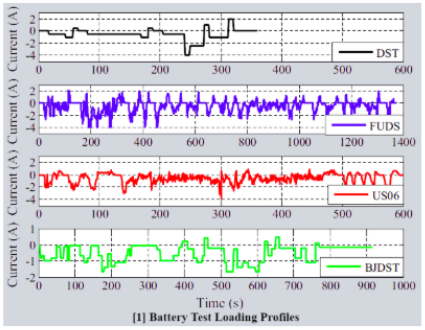
$$\mathrm{SOC}\,(t) = \mathrm{SOC}\,(t-1) + \frac{I\,(t)}{Q_n}\delta t$$

شکل ۲: فرمول محاسبه SOC در مقاله

Dynamic Test Profile

This test consists of different dynamic current profiles like Dynamic stress test DST , Federal Urban Driving Schedule FUDS, US06 Highway Driving Schedule and Beijing Dynamic Stress Test BJDST.

All tests were performed for 80% battery level and 50% battery level at 0°C, 25°C and 45°C.



[1] Battery Test Loading Profiles

Initial Capacity			
DST	Data for 0°C	Data for 25°C	Data for 45°C
FUDS	Data for 0°C	Data for 25°C	Data for 45°C
US06	Data for 0°C	Data for 25°C	Data for 45°C
BJDST	Data for 0°C	Data for 25°C	Data for 45°C

شکل ۳: دیتاست‌های مرتبط در لینک

ما از دیتاست BJDST استفاده کردیم و برای اینکه بتوانیم تاثیر دما را در جدول داده‌ها بیاوریم، ۳ فایل ذکر شده با دمای ۰، ۲۵ و ۴۵ درجه سانتیگراد را با یکدیگر merge کردیم و یک ستون دما به جدول اضافه کردیم. فایل merged.csv در میان فایل‌های پروژه نمایانگر این موضوع است.

سپس برای اینکه ستون SOC را به داده‌ها اضافه کنیم و در نهایت train را انجام دهیم، یک فایل نوت بوک ساختیم که در فایل‌های ضمیمه پروژه قرار گرفته است.

حال به اختصار کدهای این نوتبوک را توضیح می‌دهم.

```
import tensorflow as tf
from tensorflow import keras
import pandas as pd
from pandas import read_csv
from sklearn.model_selection import train_test_split


file_name = "merged.csv"
df = pd.read_csv(file_name)

columns_to_keep = ['Current(A)', 'Voltage(V)', 'temperature', 'Step_Time(s)
    ', 'Step_Index']
df = df[columns_to_keep]

df = df.dropna()

Qn = 2.0
initial_SOC = 0.8

```

```
df['SOC'] = 0.0


for i in range(len(df)):
    if i == 0 or df.loc[i, 'Step_Index'] != df.loc[i - 1, 'Step_Index']:
        df.loc[i, 'SOC'] = initial_SOC
    else:

        dt = (df.loc[i, 'Step_Time(s)'] - df.loc[i - 1, 'Step_Time(s)']) /
            3600

        df.loc[i, 'SOC'] = (df.loc[i - 1, 'SOC'] + (df.loc[i, 'Current(A)']
             / Qn) * dt).clip(0, 1)

output_file_name = "preprocessed_data.csv"
df.to_csv(output_file_name, index=False)

print(f"Preprocessing complete. Data saved to {output_file_name}.")
```

فرضی که ما برای استفاده از این دیتاست merged.csv در نظر گرفتیم، این بود که به ازای هر Step_Index، در ابتدای Step_Index، مقدار اولیه SOC را برابر 0.8 قرار می‌دهیم و بعد از آن، در آن Step_Index، با مقادیر Step_Time کار می‌کنیم و تغییرات آن را به عنوان δt در نظر می‌گیریم و مطابق رابطه گفته شده در شکل ۲، مقدار SOC را به ازای هر سطر بدست می‌آوریم. همچنین سطرهای دارای NA را نیز با dropna حذف می‌کنیم. مقدار اولیه Q_n را نیز برابر 2Ah (مطابق مقاله) در نظر می‌گیریم. نکته قابل توجه این است که در برخی موارد، مقدار SOC از یک بیشتر می‌شد که ما آن را به مقدار یک Cut off کردیم. در نهایت نیز داده‌های جدید را در فایل preprocessed_data.csv ذخیره می‌کنیم.

```
features_columns = ["Voltage(V)", "Current(A)", "temperature"]
target_column = ["SOC"]

dataset = df


features = dataset[features_columns]
targets = dataset[target_column]


X_train, X_test, y_train, y_test = train_test_split(features, targets,
    test_size=0.2, random_state=42)


initial_learning_rate = 0.01
lr_schedule = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate, decay_steps=1000, decay_rate=0.96, staircase=
        True
)
```

```
18
19
20  model = keras.Sequential([
21      keras.layers.InputLayer(input_shape=(features.shape[1],)),
22
23
24      keras.layers.BatchNormalization(),
25
26          keras.layers.Dense(16, activation='relu', kernel_initializer='
                he_normal'),
27      keras.layers.BatchNormalization(),
28
29          keras.layers.Dense(8, activation='relu', kernel_initializer='
                he_normal'),
30      keras.layers.BatchNormalization(),
31
32          keras.layers.Dense(4, activation='relu', kernel_initializer='
                he_normal'),
33      keras.layers.BatchNormalization(),
34
35
36
37
38      keras.layers.Dense(1, activation='sigmoid')
39  ])
40
41
42  model.compile(optimizer=keras.optimizers.Adam(learning_rate=lr_schedule),
43                loss='mean_squared_error', metrics=['mae'])
44
45
46  model.fit(X_train, y_train, batch_size=256, epochs=100, verbose=1)
47
48
49  test_loss, test_mae = model.evaluate(X_test, y_test, verbose=0)
50  print(f"Test MAE: {test_mae:.4f}")
51
52
53  model.save("battery_soc_model.h5")
54
55  model = tf.keras.models.load_model("battery_soc_model.h5")
56
57  converter = tf.lite.TFLiteConverter.from_keras_model(model)
58  tflite_model = converter.convert()
59
60  with open("battery_soc_model.tflite", "wb") as f:
61      f.write(tflite_model)
```

شبکه عصبی بالا را برای این داده‌ها می‌زنیم. لایه‌های مخفی شامل ۱۶، ۸ و ۴ نورون و لایه نهایی شامل یک نورون است. تابع activation لایه‌های مخفی relu و لایه خروجی نیز sigmoid است. بعد از هر لایه نیز از BatchNormalization استفاده کردیم. و همجنین از Adam به عنوان Optimizer استفاده کردیم. در واقع سعی کردیم تا حد خوبی مدل ما شبیه به جدول زیر شود (به دلیل محدودیت حافظه ESP32، تطابق کامل با این جدول ممکن نبود.)

TABLE II
HYPERPARAMETER SELECTION FOR THE PROPOSED DNN

Hyperparameter	Selection Options
Number of hidden neurons	64 per hidden layer
Number of hidden layers	Variable
Learning rate	Adaptive learning rate
Neuron initialization	He. Initialization
Optimization algorithm	Adam
Dropouts	No
Batch size	256
Batch normalization	After non-linearity
Non-linearity	ReLU for hidden layer neurons. Sigmoid for output layer neuron.

شکل ۴: پارامترهای مقاله اصلی

۲.۳ تبدیل مدل به فایل h.

اینکار به سادگی با استفاده از دستور زیر میسر است:

xxd -i [tflite filename] [.h filename]

و پس از آن، مقادیر باینری مدل tflite ما در فایل h. موجود خواهد بود.

۳.۳ پیاده سازی شبکه عصبی بر روی بورد ESP32

بدین منظور، از یک نسخه قدیمی از TensorFlowLite_ESP32 استفاده می‌کنیم، زیرا نسخه جدید و قابل اطمینانی از آن موجود نیست. همچنین نسخه ESP32 را نیز در Arduino IDE به مقدار 2.0.17 تقلیل می‌دهیم.

این کد پیاده سازی را به صورت یک فایل cpp. و h. در کنار پیاده سازی پروتکل‌ها قرار دادیم که نام آن را inference گذاشتیم.

```cpp
#include "inference.h"
#include "model_data.h"

Inference::Inference()
    : error_reporter(nullptr), model(nullptr), interpreter(nullptr),
      input(nullptr), output(nullptr) {}

void Inference::begin() {
    Serial.begin(115200);
    while (!Serial);
    Serial.println("\nStarting Regression Model Setup...");

    static tflite::MicroErrorReporter micro_error_reporter;
    error_reporter = &micro_error_reporter;

    model = tflite::GetModel(battery_soc_model_tflite);
    if (model->version() != TFLITE_SCHEMA_VERSION) {
        Serial.println("Model version mismatch!");
        while (1);
    }

    static tflite::MicroMutableOpResolver<4> resolver;
    resolver.AddFullyConnected();
    resolver.AddRelu();
    resolver.AddLogistic();
    resolver.AddReshape();

    static tflite::MicroInterpreter static_interpreter(
        model, resolver, tensor_arena, kTensorArenaSize, error_reporter);
    interpreter = &static_interpreter;

    if (interpreter->AllocateTensors() != kTfLiteOk) {
        Serial.println("AllocateTensors failed!");
        while (1);
    }

    input = interpreter->input(0);
    output = interpreter->output(0);

    Serial.println("\nModel initialized!");
}

float Inference::predict(float input_features[], int size) {
    if (size != input->dims->data[1]) {
        Serial.println("Input size mismatch!");
        return -1;
```

```
    }

    for (int i = 0; i < size; i++) {
        input->data.f[i] = input_features[i];
    }

    if (interpreter->Invoke() != kTfLiteOk) {
        Serial.println("Invoke failed!");
        return -1;
    }

    return output->data.f[0];
}
```

این کد بدین گونه کار می‌کند که مدل را از روی فایل باینری model_data.h لود کرده و سپس با استفاده از تابع predict، می‌توانیم ورودی داده و خروجی شبکه عصبی را دریافت کنیم. ما از این تابع برای استفاده شبکه عصبی‌مان در پروتکل‌های مختلف استفاده خواهیم کرد.

۴ گزارش پیاده سازی پروتکل‌های ارتباطی

ما از سه پروتکل ارتباطی WiFi, BLE, UART استفاده کرده‌ایم که نحوه استفاده از آنها را در ادامه توضیح خواهیم داد.

۱.۴ Wi-Fi

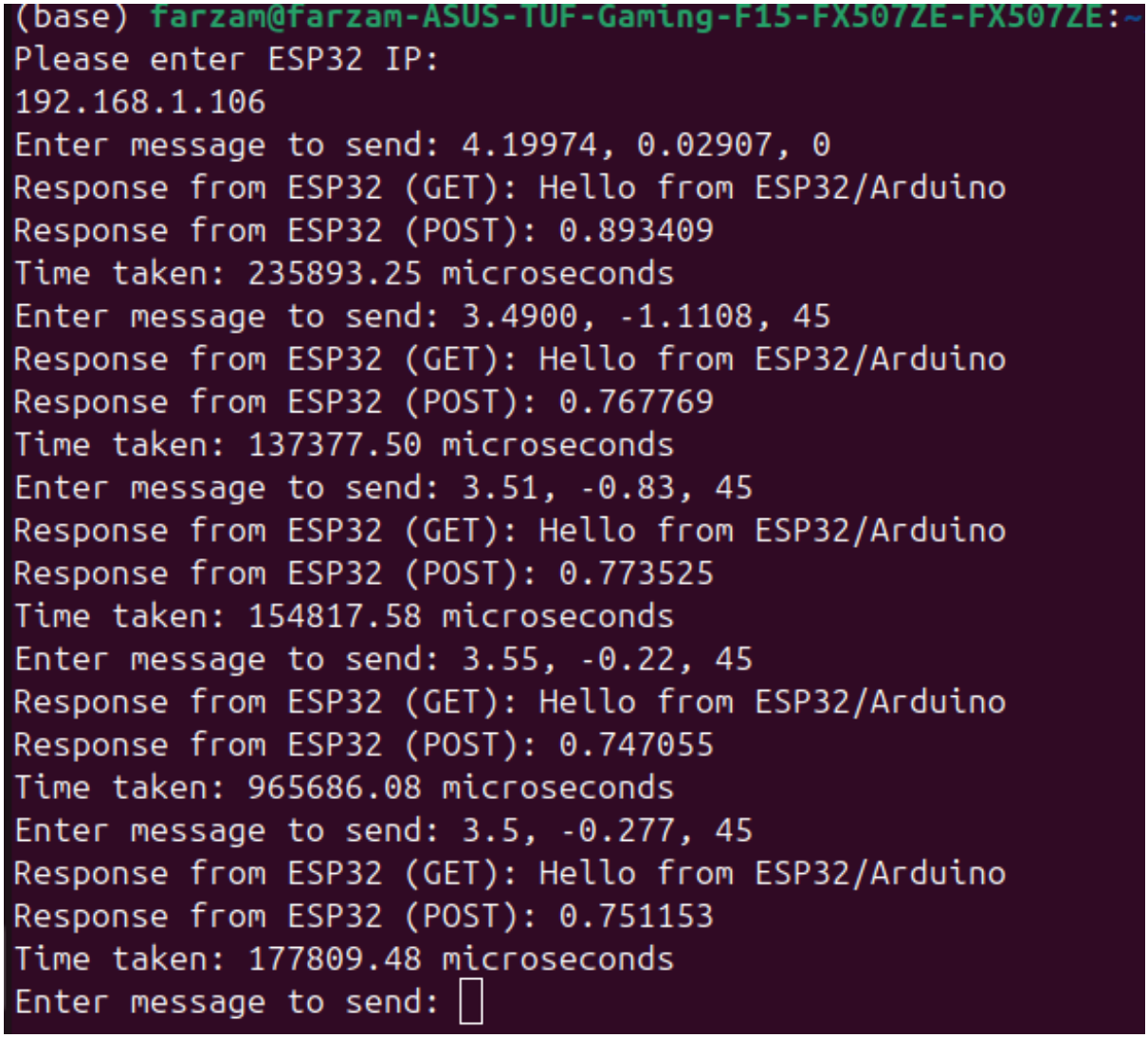
Wi-Fi یک فناوری ارتباطی بی‌سیم است که برای انتقال داده‌ها در شبکه‌های محلی (LAN) استفاده می‌شود. این پروتکل:

- در باندهای 2.4GHz و 5GHz کار می‌کند.
- سرعت بالایی دارد و برای اتصال دستگاه‌ها به اینترنت یا ایجاد شبکه‌های محلی مناسب است.
- در بردهای ESP32 و ESP8266 به‌صورت داخلی پشتیبانی می‌شود و برای ارسال داده‌ها به سرور، کنترل از راه دور و اینترنت اشیا (IoT) کاربرد دارد.

در ادامه، توضیحاتی درباره نحوه پیاده سازی پروتکل WiFi برای ESP32 صحبت می‌کنیم.

بدین منظور، ما یک کد ino. برای سمت ESP32 و یک کد پایتون برای سمت pc داریم تا این دو بتوانند با هم ارتباط بگیرند. به این صورت که داده‌های ورودی شبکه عصبی، از pc به ESP32 از طریق کد پایتون و WiFi منتقل شده و پس از infernce روی ESP32، از طریق کد ino. به سمت pc می‌رود و خروجی توسط کد پایتون دریافت شده و نمایش داده می‌شود.

برای مثال در زیر می‌توانیم یک مثال از عملکرد پروتکل WiFi را ببینیم.



شکل ۵: Wifi

پس از تعدادی اجرا، مشاهده می‌شود که به طور میانگین، کل مدت زمان inference و انتقال داده‌ها از طریق پروتکل، تقریبا برابر 150000 میکروثانیه است.

۲.۴ Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE) یک نسخه کم‌مصرف از بلوتوث معمولی است که برای ارتباط بین دستگاه‌های کم‌مصرف مانند سنسورها، گجت‌های پوشیدنی و دستگاه‌های IoT استفاده می‌شود. ویژگی‌های کلیدی آن:

- مصرف انرژی بسیار پایین، مناسب برای دستگاه‌های باتری‌خور.
- سرعت کمتر نسبت به Wi-Fi اما کافی برای انتقال داده‌های کم‌حجم.
- برد تقریبی 10 تا 100 متر بسته به قدرت فرستنده و محیط.

از حیث نحوه ارتباط، پیاده سازی BLE مشابه WiFi است و صرفا از پروتکل بلوتوث استفاده می‌کند. در تصویر زیر، نمونه‌ای از کار با بلوتوث برای این شبکه عصبی را قرار می‌دهیم. همانطور که مشاهده می‌شود، مدت زمان به طول انجامیده (مجموع inference و انتقال داده‌ها) تا حد خوبی شبیه WiFi است. تقریبا می‌توان گفت ۱۷۰۰۰۰ میکروثانیه است.

```
Connected to ESP32!
Enter message: 3.5516, -1.1114, 0
Sent: 3.5516, -1.1114, 0
Received: Processed: 0.759369
Round-trip time: 0.143491 seconds
Enter message: 3.5516, -1.1114, 0
Sent: 3.5516, -1.1114, 0
Received: Processed: 0.759369
Round-trip time: 0.179914 seconds
Enter message: 3.5516, -1.1114, 0
Sent: 3.5516, -1.1114, 0
Received: Processed: 0.759369
Round-trip time: 0.171526 seconds
Enter message: 3.5516, -1.1114, 0
Sent: 3.5516, -1.1114, 0
Received: Processed: 0.759369
Round-trip time: 0.153265 seconds
Enter message: 3.5516, -1.1114, 0
Sent: 3.5516, -1.1114, 0
Received: Processed: 0.759369
Round-trip time: 0.210646 seconds
Enter message: 3.5516, -1.1114, 0
Sent: 3.5516, -1.1114, 0
Received: Processed: 0.759369
Round-trip time: 0.143157 seconds
Enter message: 3.5516, -1.1114, 0
Sent: 3.5516, -1.1114, 0
Received: Processed: 0.759369
Round-trip time: 0.211356 seconds
Enter message: 3.5516, -1.1114, 0
Sent: 3.5516, -1.1114, 0
Received: Processed: 0.759369
Round-trip time: 0.144201 seconds
Enter message:
```

شکل ۶: BLE

۳.۴ UART (Serial)

UART (Universal Asynchronous Receiver/Transmitter) یک پروتکل ارتباطی سریال است که برای ارتباط بین دو دستگاه (مانند میکروکنترلرها و سنسورها) استفاده می‌شود.

- از دو سیم اصلی TX (ارسال) و RX (دریافت) استفاده می‌کند.

- ارتباط آن آسنکرون است، یعنی به سیگنال کلاک جداگانه نیاز ندارد.
- در بردهایی مانند Arduino و ESP32 برای ارتباط با سنسورها، ماژول‌های GPS، ماژول‌های GSM و ... استفاده می‌شود.

برای پیاده سازی این بخش، صرفا استفاده از توابع Serial خود ESP32 کافی است، زیرا خود سریال این بورد و بوردهای آردوئینو، از UART استفاده می‌کنند. مشاهده می‌شود که زمان کل (هم infernce و هم انتقال داده‌ها با UART، حدودا ۲۵۰ میکروثانیه زمان خواهد برد.

```
Total Time in UART: 261 µs
You sent: 3.4164, 0.4440, 25
The Value is "0.719401".
Total Time in UART: 251 µs
You sent: 3.4164, 0.4440, 25
The Value is "0.719401".
Total Time in UART: 260 µs
You sent: 3.4164, 0.4440, 25
The Value is "0.719401".
Total Time in UART: 252 µs
You sent: 3.4164, 0.4440, 25
The Value is "0.719401".
Total Time in UART: 260 µs
You sent: 3.4164, 0.4440, 25
The Value is "0.719401".
Total Time in UART: 261 µs
You sent: 3.4164, 0.4440, 25
The Value is "0.719401".
Total Time in UART: 260 µs
You sent: 3.4164, 0.4440, 25
The Value is "0.719401".
Total Time in UART: 252 µs
You sent: 3.4164, 0.4440, 25
The Value is "0.719401".
Total Time in UART: 253 µs
```

شکل ۷: UART (Serial)

۵ بررسی نتایج

مشاهده شد که مدت زمان انتقال داده و infernce در پروتکل BLE و WiFi نزدیک به هم است و هر دو از این مقدار در UART بیشتر هستند.

۶ نتیجه‌گیری

بدیهتا می‌توان نتیجه گرفت که پروتکل سیمی UART۷ مدت زمان کمتری برای انتقال داده نیاز دارد تا دو پروتکل بیسیم WiFi و .BLE اما بنابر شرایط مسئله، استفاده از WiFi و BLE می‌تواند در برخی موارد inference روی بورد آردوئینو مثمر ثمر واقع شود. برای مثال دستگاه‌های IoT و خانه‌های هوشمند که باید با استفاده از پروتکل‌های بیسیم میان بخش‌های مختلف آنها ارتباط صورت گیرد که اغلب دارای وظایف هوش مصنوعی نیز هستند.