



# Final-Term Presentation

Embedded AI - Time-Series Analysis

Chen, Yujie  
Hu, Jingyi  
Sadiq, Joshua  
Yuan, Hengzhen





## Background:

Time series data is a common and important information collected from embedded system sensors. Clustering and prediction are essential to learn the time series features. However, those intelligence methodologies are challenging on embedded platforms due to computational constraints. Moreover, in the current research landscape, There is a noticeable lack of research focusing on AI applications on limited resource processors. The scarcity of resources require a deliberate approach with lightweight design of models, code optimization and training data preprocessing for effective resolution.

## Motivation:

1. Importance of time series data
2. Fill the gap of AI studies on limited resource processors



## Technical Goals:

**Goal 1:** Implementation of **ARIMA** model on ESP32 for time-series prediction of dataset Global Land Temperature.

**Goal 2:** Implementation of **RNN** model on ESP32 for time-series prediction of dataset Global Land Temperature.

**Goal 3:** Implementation of **K-means** model on ESP32 for time-series clustering of dataset BME.

**Goal 4:** Implementation of **SOM** model on ESP32 for time-series clustering of dataset BME.

## Resource goals:

**Goal 1:** The time budget is 10.5 credits → 280 hours per person → 1120 hours in total.

**Goal 2:** The proposed submission date to deliver the final products is January 10th.



## Main Outcomes:

1. Training process of models on PC
2. Prediction process of models on ESP32
3. Key data collection on ESP32

## Milestones:

Task 1: Literature study and Implementation of models on PC

Oct 09 - Oct 20

Task 2: Hello world demo on ESP32

Oct 23 - Nov 03

Task 3: Implementation on ESP32

Nov 06 - Nov 17

Task 4: Data collection and analysis

Nov 20 - Dec 01

Task 5: Optimization of models

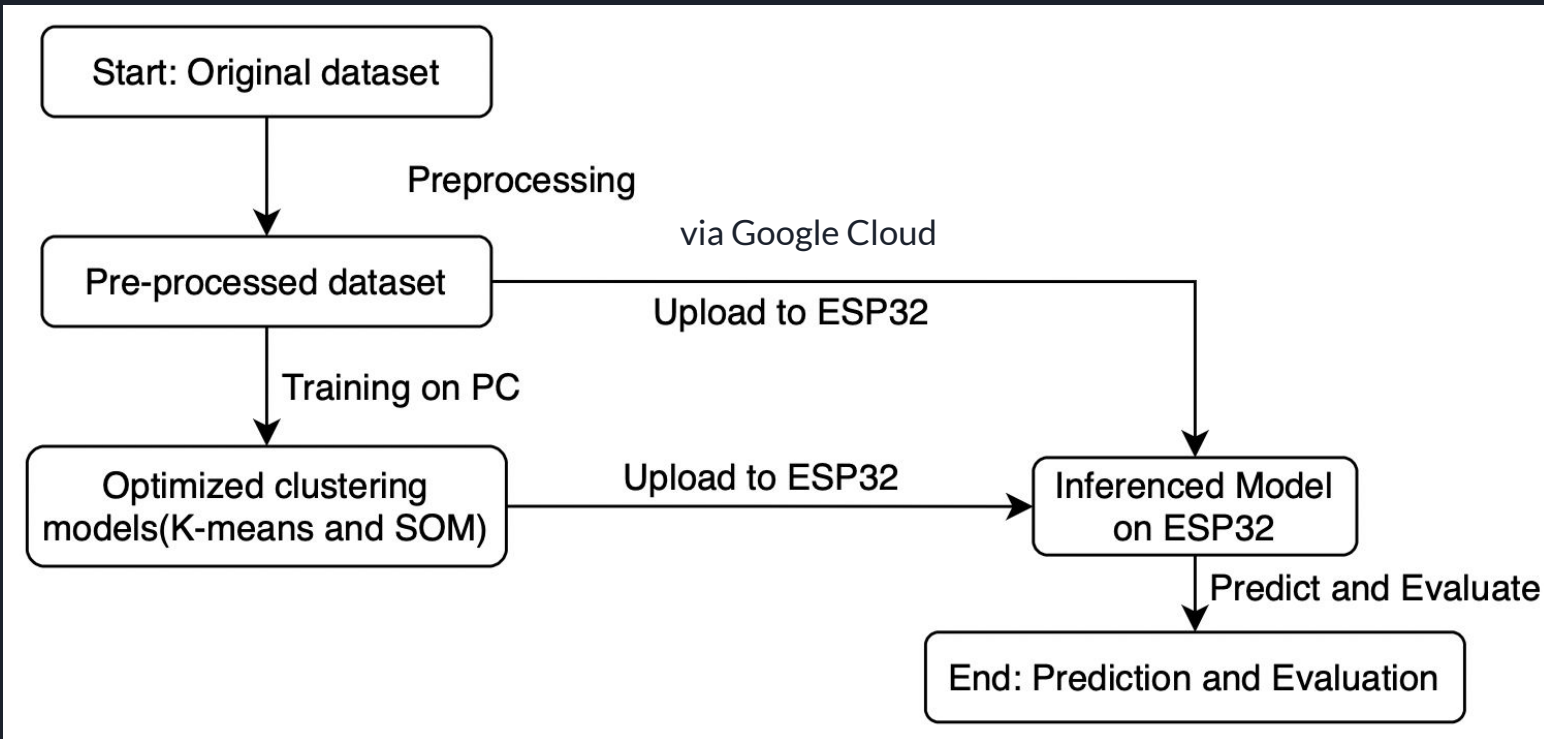
Dec 04 - Dec 15

Task 6: Summary and report preparation

Dec 18 - Jan 05

# Clustering models: Kmeans & SOM

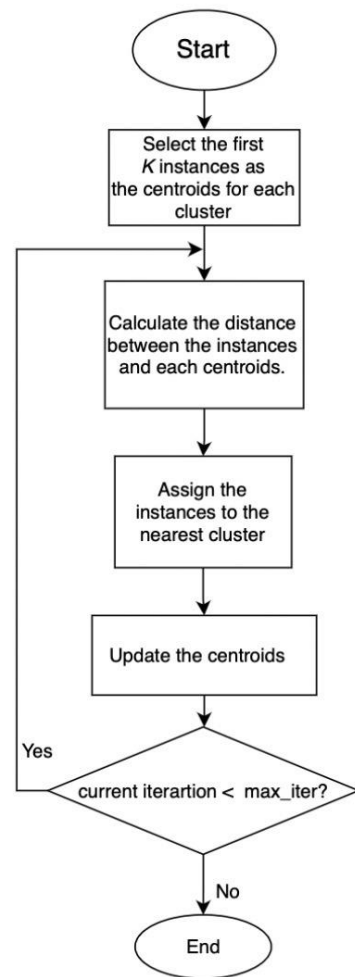
## Design Workflow



# Clustering models: Kmeans & SOM

## Algorithms - Kmeans

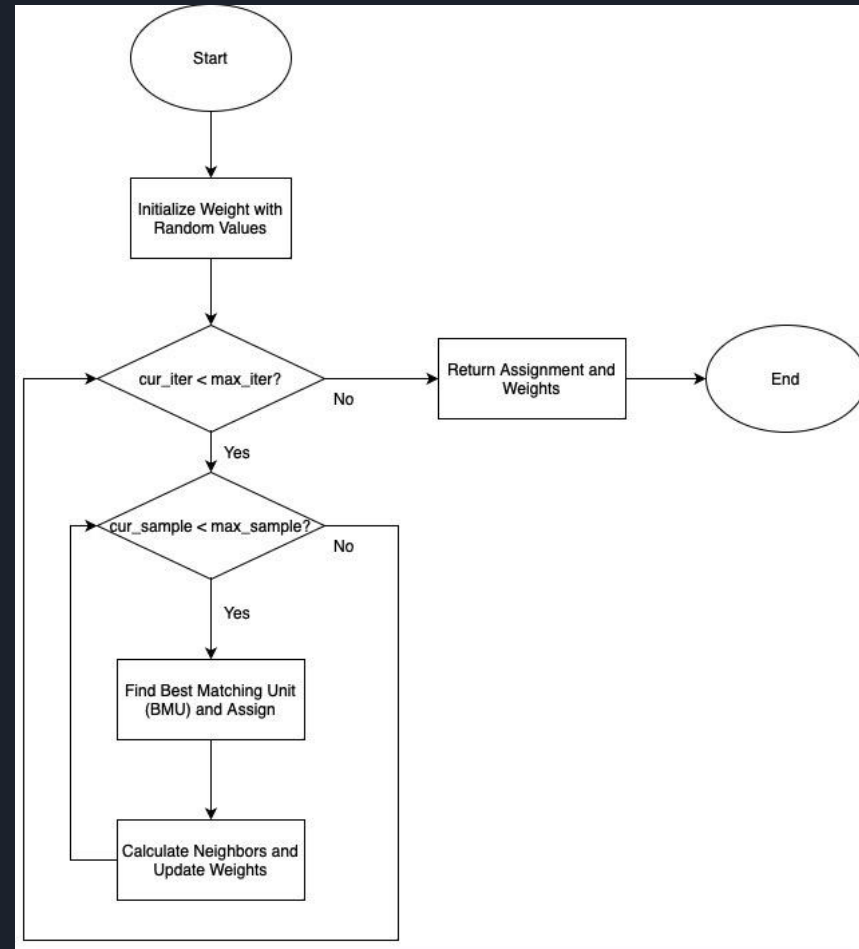
- Widely-used clustering algorithm.
- Objective is to partition a dataset into K distinct clusters.
- Operates iteratively: initializes centroids, assigns points to clusters, updates centroids.
- Assumes clusters are spherical and equally sized.
- Simple and computationally efficient, making it suitable for large datasets.
- May struggle with non-spherical or unevenly sized clusters.
- Sensitive to initial centroid selection.



# Clustering models: Kmeans & SOM

## Algorithms - SOM

- Artificial neural network for data visualization and organization.
- Represents clusters in a grid of neurons.
- Excels at capturing data topology and relationships.
- Trains by adjusting weights based on input vectors and neighboring neurons.
- Reduces dimensionality while preserving essential features.
- Effective for tasks like data visualization and pattern recognition.





# Clustering Models: K-means & SOM

## 1. Dataset

Dataset: BME

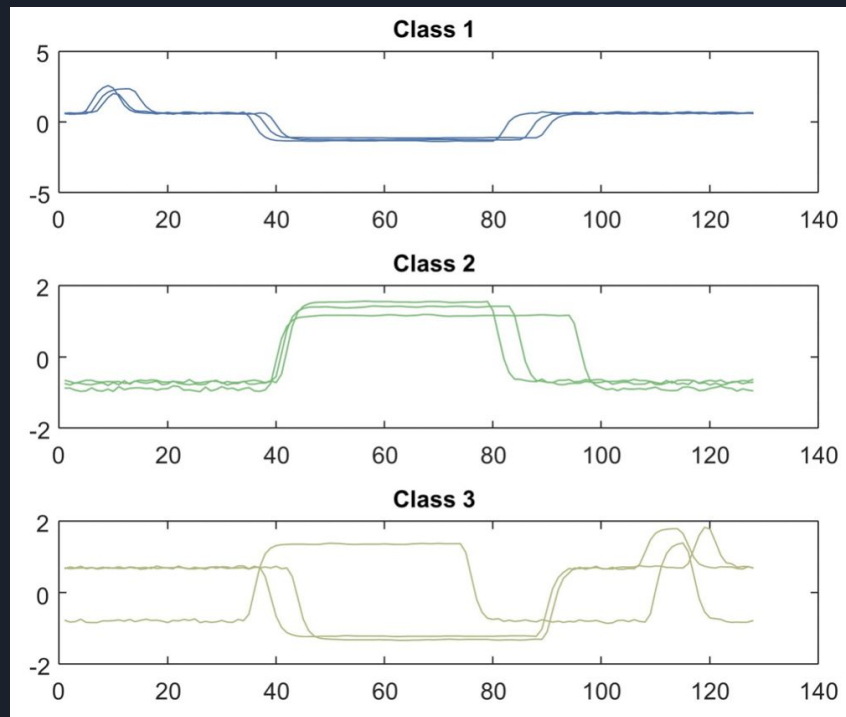
Time series length: 128

Number of classes: 3

Size: 180

\* *Limited memory of ESP32 → split into 6 batches, with 30 instances in each*

\* *Upload to Google Cloud*





# Clustering Models: K-means & SOM

## 2. Build & Train the models on PC

Both algorithms have been successfully implemented on PC.

```
// Set hyperparameters for SOM
int height = 3;
int width = 1;
int max_iter = 100;
float lr = 0.1;
float sigma = 1.0;
```

```
// Set hyperparameters for K-means
int K = 3;
int max_iter = 100;
```

Key operation: export the parameters to ESP32

**Weights(SOM) , Centroids(K-means)**



# Clustering Models: K-means & SOM

## 3. Transplant the code to ESP32

a. Access the dataset batches from Google Cloud

```
bool downloadFromGCS(const char* url)
```

b. Implement the inference function base on the parameters exported from well-trained PC-compatible models

```
void kMeansInference(double *input_data, double **centroids, int n_samples, int m_features, int *assignments)
```

```
void SOMInference(t_pos *assignment, double *data, int n_samples, int m_features, int height, int width, double **weights)
```



# Clustering Models: K-means & SOM

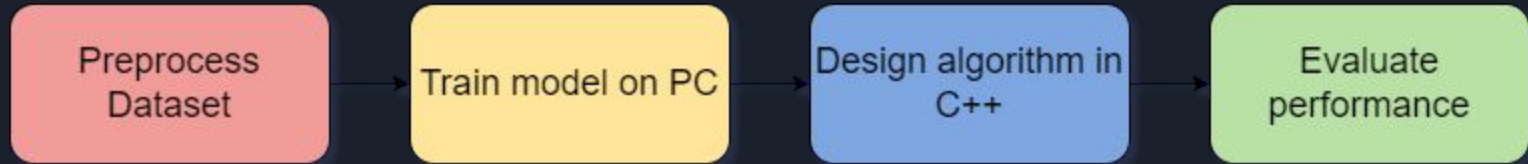
## 4. Performance evaluation

- Efficient & Accurate
- Quick Response Time
- High Rand Index

Model	Inference time	RI value	Memory consumption
K-means	78ms	0.6233	92212 bytes/batch
SOM	66ms	0.7577	87724 bytes/batch

# ARIMA

## Design Workflow



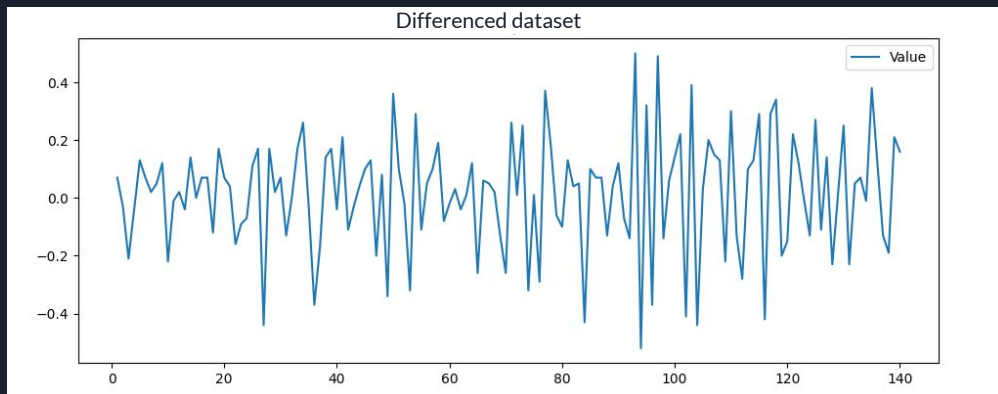
# ARIMA

## 1. Dataset

### Global Land Temperature Anomalies

Data points: 141

- Fits sufficiently on the ESP32
- <20% of Flash memory



# ARIMA

## 2. Training Model on PC

- EDA finds no stationarity —> Differencing

```
-----  
ADF Statistic: 0.9382926419378832  
p-value: 0.993570249862259  
Critical Values:  
1%, -3.479007355368944  
5%, -2.8828782366015093  
10%, -2.5781480587564603  
-----  
Critical values less than adf statistic: 3  
p-value is below the 5% significance level: False  
Stationary? False  
-----
```

- Box-Jenkins used to construct model

```
Best model: ARIMA(2,1,4)(0,0,0)[0]  
Total fit time: 3.901 seconds
```

- Internal parameters extracted from model

```
ar.L1      -1.086464  
ar.L2      -0.672785  
ma.L1      -0.537128  
ma.L2      -0.522477  
ma.L3      -0.619176  
ma.L4       0.723350  
sigma2     0.025877  
dtype: float64
```

# ARIMA

## 3. Algorithm Design for C++

- Implement equation in C++, parameters known from PC-model

$$y_t = \alpha_1 y_{t-1} + \alpha_2 y_{t-2} + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \theta_3 \varepsilon_{t-3} + \theta_4 \varepsilon_{t-4}$$

- Dataset stored and initialized as array

```
// Dataset: "Global land surface temperature anomalies" values
float A[] = {-0.50,-0.43,-0.46,-0.71,-0.58,-0.51,-0.49,-0.44,-0.32,-0.54,-0.55,-0.53,-0.57,-0.43,-0.43,-0.36,-0.29,-0.41,-0.24,-0.17,-0.13,-0.29,-0.38,-0.45,-0.34,-0.17,-0.61,-0.44,-0.42,-0.35,-0.48,-0.48,-0.31,-0.85,-0.88,-0.45,-0.62,-0.48,-0.31,-0.35,-0.14,-0.25,-0.26,-0.24,-0.14,-0.01,-0.21,-0.13,-0.47,-0.11,-0.01,-0.03,-0.35,-0.06,-0.17,-0.12,-0.02,0.17,0.09,0.07,0.10,0.06,0.07,0.19,-0.07,-0.01,0.04,0.06,-0.07,-0.33,-0.07,-0.06,0.19,-0.13,-0.12,-0.41,-0.04,0.14,0.08,-0.02,0.11,0.15,0.20,-0.23,-0.13,-0.06,0.01,-0.12,-0.08,0.04,-0.03,-0.17,0.33,-0.19,0.13,-0.24,0.25,0.11,0.17,0.31,0.53,0.12,0.51,0.07,0.10,0.30,0.45,0.58,0.36,0.66,0.53,0.25,0.35,0.48,0.77,0.35,0.64,0.98,0.78,0.63,0.85,0.97,0.96,0.83,1.10,0.99,1.13,0.90,0.91,1.16,0.93,0.98,1.05,1.04,1.42,1.54,1.41,1.22,1.43,1.59};
```

- Predicted values calculated for each iteration in for-loop based on eq. and stored in empty array
- Errors calculated alongside predicted value based on:  
 $\text{Error}[i] = \text{Dataset}[i] - \text{Predicted\_Value}[i]$



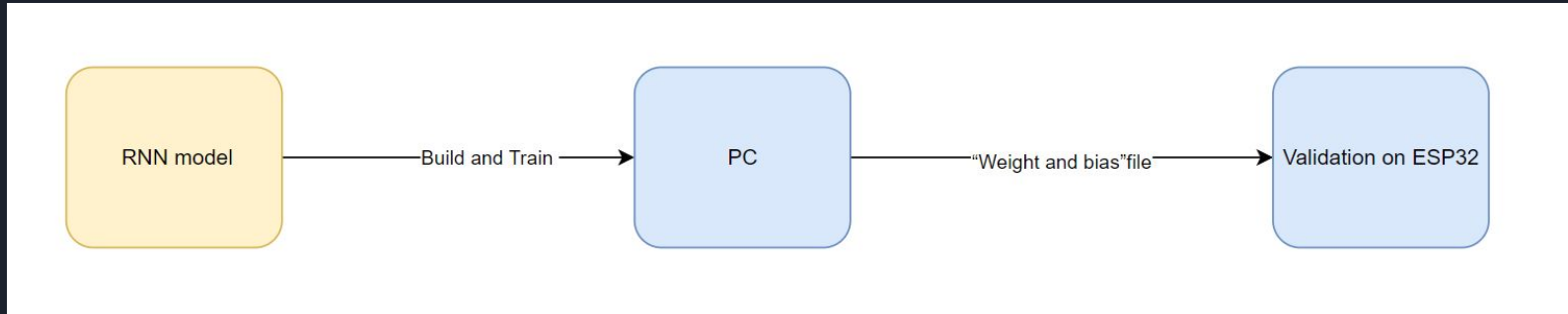
# ARIMA

## 4. Performance evaluation

- For ESP32's model:
  - C++'s Chrono-module allows high-resolution time-taking
  - PlatformIO measures utilized storage space
- For Python model:
  - `time.time()` allows time-taking
  - `sys.getsizeof(model)` returns the size of an object
- Comparison:

Model	Inference time	Dataset memory usage
Python	997 $\mu s$	2256 Bytes
ESP32	134 $\mu s$	576 Bytes

- Results: ESP32 bare-metal implementation superior!



**Data set :** A deterministic series sampled from a sinusoidal wave with period 20 seconds, with a sample rate of 100 Hz.

- Completely implement the inference process of the RNN model on esp32
- Optimize memory usage of the weights and bias
- Test and analyze the performance of the whole inference process

# Performance test

RNN

**Platform:** Ardunio IDE


**Memory footprint:** library function

```
size_t freeHeap = ESP.getFreeHeap();
```

**Inference time:** mark the beginning and the ending

```
unsigned long executionTime = endTime - startTime;
```

(The average of 20 results)



Hidden Size	Memory (float) (KB)
16	1.344
32	4.736
64	17.664
128	68.028
158	102.96
256	-
318	-

- The limitation of DRAM

**Float -> Fixed point**

Float 4 bytes

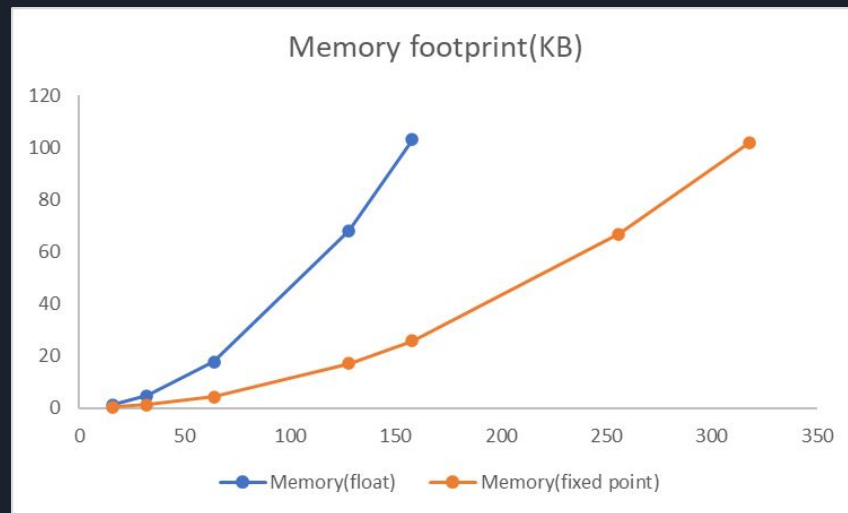
Fixed point 2 bytes

# Results and Analysis

RNN

TABLE III  
MEMORY CONSUMPTION FOR DIFFERENT HIDDEN SIZES

Hidden Size	Memory (float) (KB)	Memory (fixed point) (KB)
16	1.344	0.34
32	4.736	1.18
64	17.664	4.41
128	68.028	17.02
158	102.96	25.76
256	-	66.84
318	-	101.96

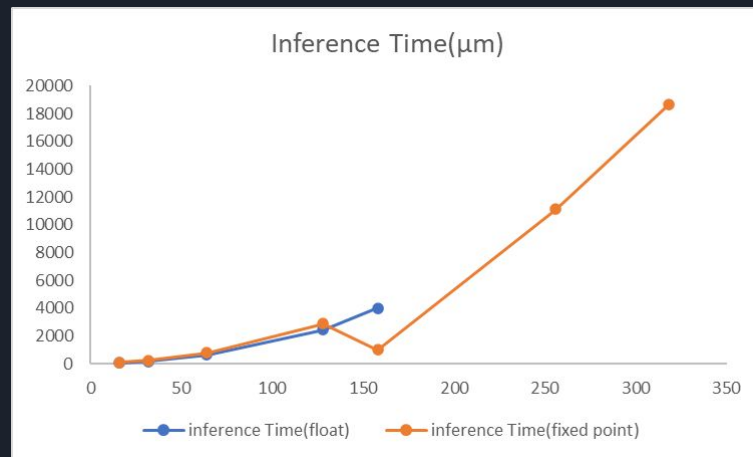


# Results and Analysis

RNN

TABLE II  
INFERENCE TIME FOR DIFFERENT HIDDEN SIZES

Hidden Size	Inference Time (float) ( $\mu$ s)	Inference Time (fixed point) ( $\mu$ s)
16	71	110.2
32	186	246.4
64	633	791.1
128	2452	2900.8
158	4022	1012.1
256	-	11133.6
318	-	18647.1



## Technical goals:

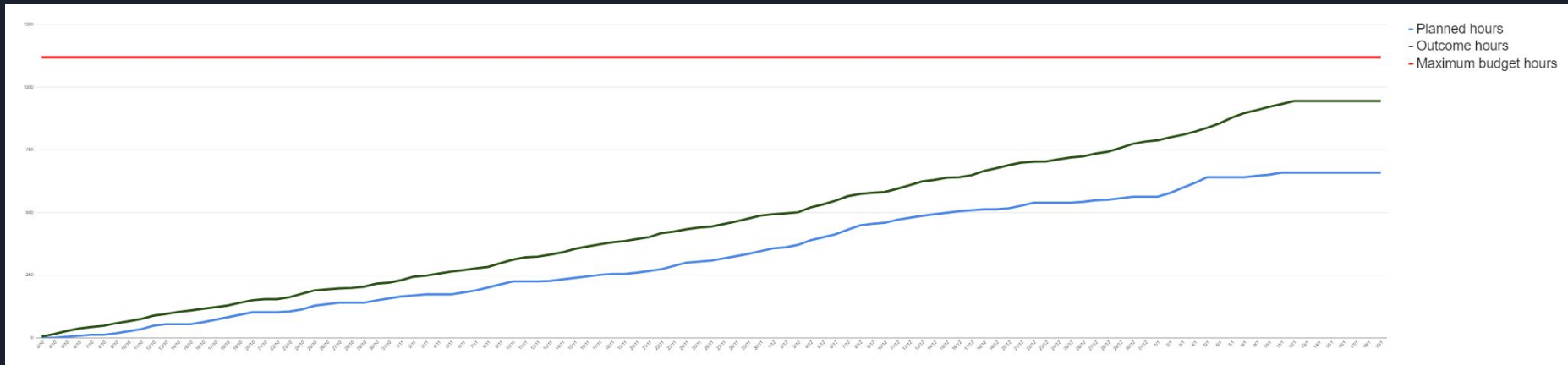
	Design?	Implemented?	Metrics measured?
ARIMA	✓	✓	✓
RNN	✓	✓	✓
K-means	✓	✓	✓
SOM	✓	✓	✓



## Business goals:

	Below max budget?	Delivered?
Yujie	✓	✓
Jingyi	✓	✓
Joshua	✓	✓
Hengzhen	✓	✓

# Time resource allocation





Thank you for your patience!



# Q&A