IPIN 2024 PRESENTATION

Exploring the Feasibility of Automated Data Standardization using Large Language Models for Seamless Positioning

Max Jwo Lem Lee, **Ju Lin (Speaker)**, Li-Ta Hsu

Department of Aeronautical and Aviation Engineering
The Hong Kong Polytechnic University

maxjl.lee@connect.polyu.hk
ju.lin@connect.polyu.hk
lt.hsu@polyu.edu.hk

October 14, 2024 – October 17, 2024

Electronic Notes Available for Download through the IPIN Virtual Portal and ipin-conference.org

Sensor Fusion?

GSM/3/4/5G

Indoor GNSS

UWB

GNSS

Ultrasound

Image SLAM

Radio AM&FM

Accelerometer

BLE

Visual Positioning System (VPS)

Radar

NFC

WLAN

LiFi

WiFi

Theodolites

TV

Gyroscope

Imagemarkers

Contactless cards

Pressure

LiDAR

Bar codes

Cospas Sarsat, Argos

Magnetometer

Opportunity radio signals

Any Trouble?

> **Inconsistent Data Format**

- Variety of **formats**, **units**, or **conventions**
  - UTC time or Local time?
  - UNIX or YYYY-MM-DDTHH:mm:ss.sssZ?
  - Degree or Radian?
  - Cartesian or Polar Coordinate?

> **Which can lead to**

- Error during execution
- Incorrect state estimates

> **Current Solutions**

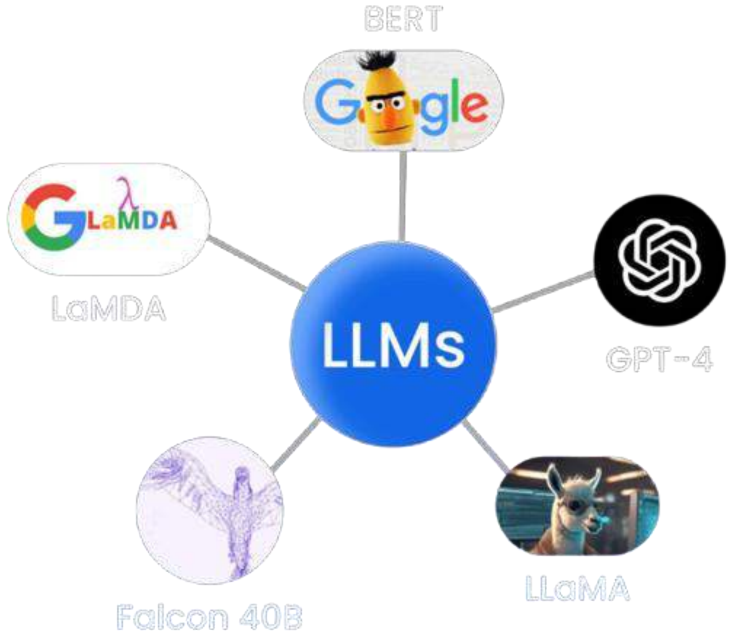
- Sensor Calibration and Preprocessing Pipelines
- Manual Unit Conversions
- Manual Coordinate Transformations

> **Drawbacks**

- High **complexity** and **maintenance** cost
- Manual **errors** -> error propagation
- Limited **flexibility** & **scalability**
- Compromised **real-time** performance

Can LLMs Help?

> **LLMs as a Solution**
- Contextual Understanding of Diverse Data
- Automatic Detection and Standardization
- Adaptability to New Sensors
- Reduction of Manual Effort and Error



Complexity

Flexibility

Errors

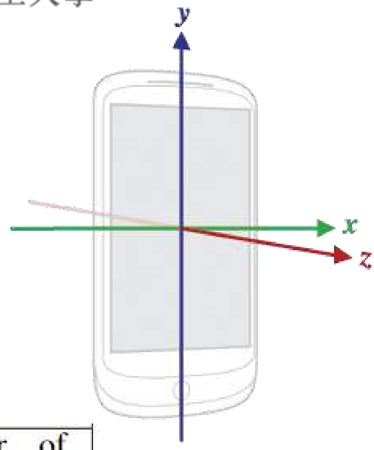
Scalability

Computational cost

Real-time performance

Schema

The Pre-defined Schema



- The schema is based on common inputs to the Kalman filter for sensor fusion
- Assumption: Coordinate System is Android or WGS84/HK1980 format

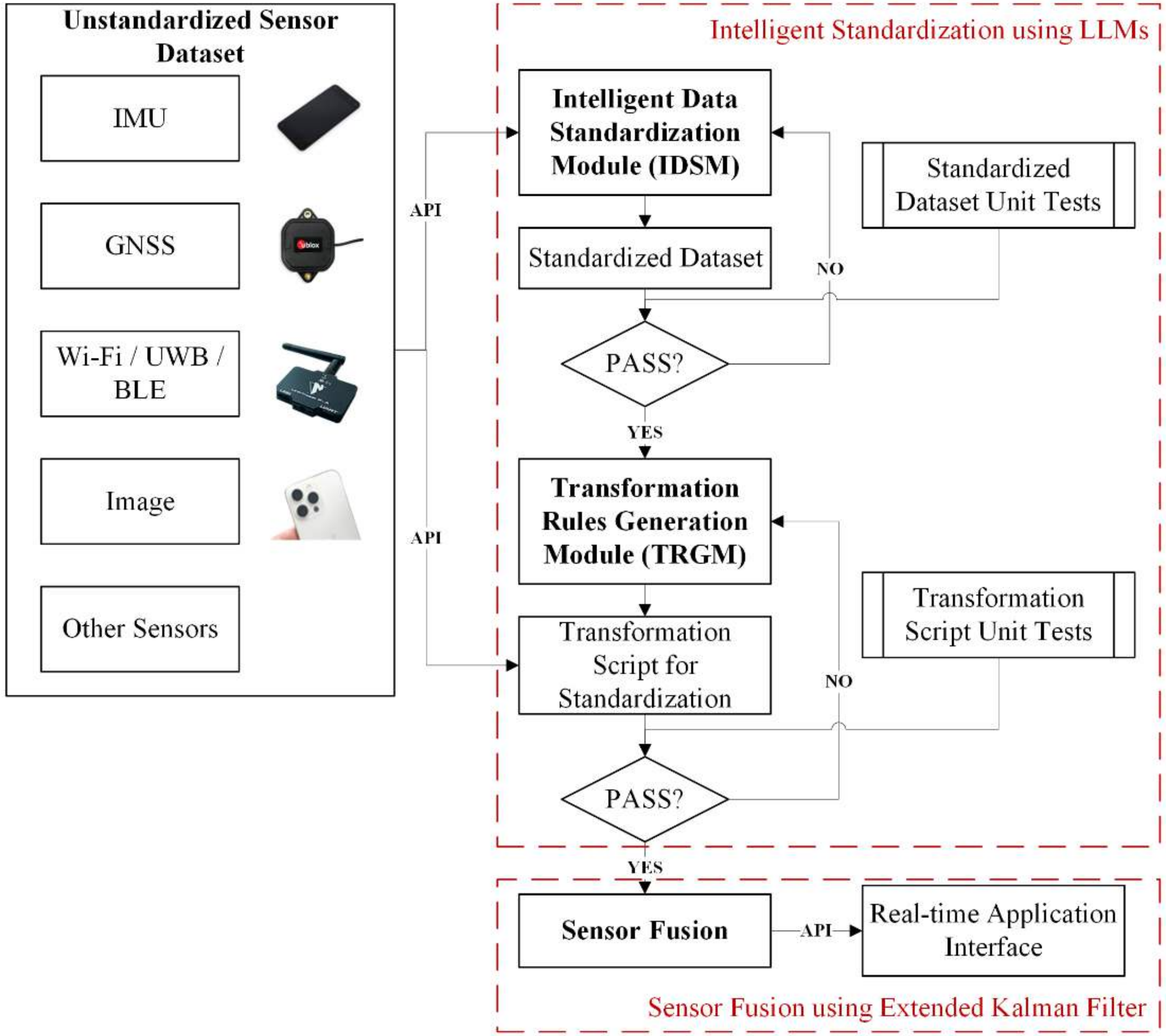
Sensor Type	Required Fields	Values Object Properties	Description
Magneto-meter	name, time, values	x (µT), y (µT), z (µT)	Measures magnetic field strength along x, y, and z axes in microteslas (µT).
Gyro-scope	name, time, values	x (rad/s), y (rad/s), z (rad/s)	Measures angular velocity along x, y, and z axes in radians per second (rad/s).
Accelero-meter	name, time, values	x (m/s²), y (m/s²), z (m/s²)	Measures acceleration along x, y, and z axes in meters per second squared (m/s²).
Gravity	name, time, values	x (m/s²), y (m/s²), z (m/s²)	Measures gravity effects along x, y, and z axes in meters per second squared (m/s²).
Ultra-Wideband (UWB)	name, time, values	x (m), y (m), z (m)	Determines spatial position in meters (m).
Bluetooth	name, time, values	x (m), y (m), z (m)	Determines spatial position in meters (m).
Pedometer	name, time, steps	steps (count)	Tracks the number of steps taken (count).
Ori-entation	name, time, values	qx, qy, qz, qw	Provides orientation details in quaternion format.
Baro-meter	name, time, values	relative altitude (m), pressure (mBar)	Measures relative altitude in meters (m) and atmospheric pressure in millibars (mBar).
Location	name, time, values	latitude (°), longitude (°), altitude (m), speed (m/s), speed accuracy (m/s), horizontal accuracy (m), vertical accuracy (m)	Provides comprehensive location data including coordinates (degrees), speed (meters per second), altitude (meters), and accuracies (meters).
Image	name, time, image	image (data)	Provides image data in binary format.

The Pre-defined Schema

Sensor Type	Required Fields	Values Object Properties	Description
Magneto-meter	name, time, values	x (μT), y (μT), z (μT)	Measures magnetic field strength along x, y, and z axes in microteslas (μT).
Gyro-scope	name, time, values	x (rad/s), y (rad/s), z (rad/s)	Measures angular velocity along x, y, and z axes in radians per second (rad/s).
Accelero-meter	name, time, values	x (m/s^2), y (m/s^2), z (m/s^2)	Measures acceleration along x, y, and z axes in meters per second squared (m/s^2).
Gravity	name, time, values	x (m/s^2), y (m/s^2), z (m/s^2)	Measures gravity effects along x, y, and z axes in meters per second squared (m/s^2).
Ultra-Wideband (UWB)	name, time, values	x (m), y (m), z (m)	Determines spatial position in meters (m).
Bluetooth	name, time, values	x (m), y (m), z (m)	Determines spatial position in meters (m).
Pedometer	name, time, steps	steps (count)	Tracks the number of steps taken (count).
Ori-entation	name, time, values	qx, qy, qz, qw	Provides orientation details in quaternion format.
Baro-meter	name, time, values	relative altitude (m), pressure (mBar)	Measures relative altitude in meters (m) and atmospheric pressure in millibars (mBar).
Location	name, time, values	latitude (°), longitude (°), altitude (m), speed (m/s), speed accuracy (m/s), horizontal accuracy (m), vertical accuracy (m)	Provides comprehensive location data including coordinates (degrees), speed (meters per second), altitude (meters), and accuracies (meters).
Image	name, time, image	image (data)	Provides image data in binary format.

Framework

Proposed Framework



Intelligent Data Standardization Module (IDSM)

> **Overview**

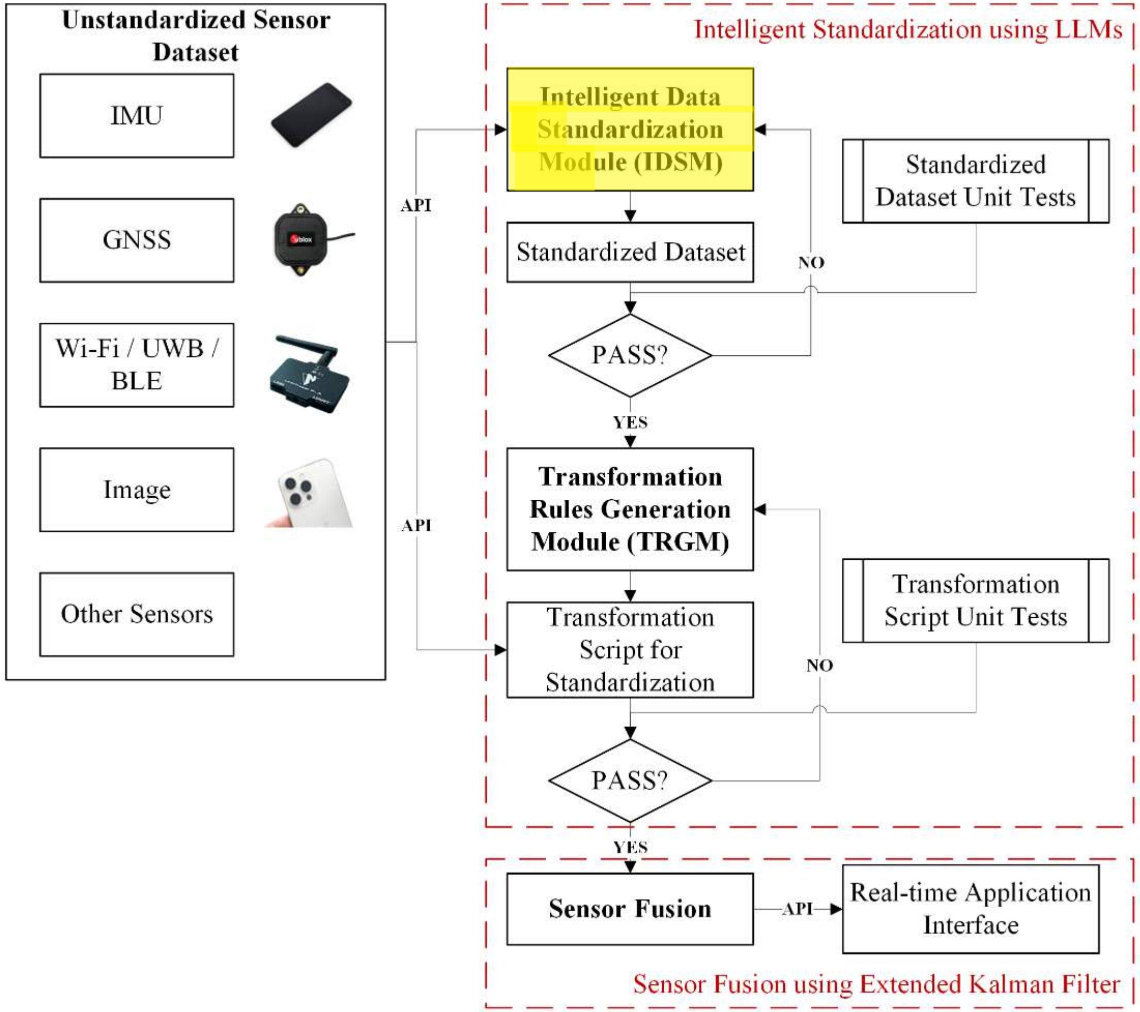
- IDSM leverages **GPT-4-0613** for standardizing sensor data.
- Trained Sensors: Pedometers, Magnetometers, Orientation Sensors, Gyroscopes, Accelerometers, Gravity Sensors, Barometers, GNSS Receivers, Bluetooth, and UWB

> **Data Standardization Process**

- Input Data: $\mathbf{D}_i = \{d_{i1}, d_{i2}, \ldots, d_{in}\}$.
- Standardized Data: $\mathbf{S}_i = \mathrm{IDSM}(\mathbf{D}_i)$

> **Fine-Tuning & Training**

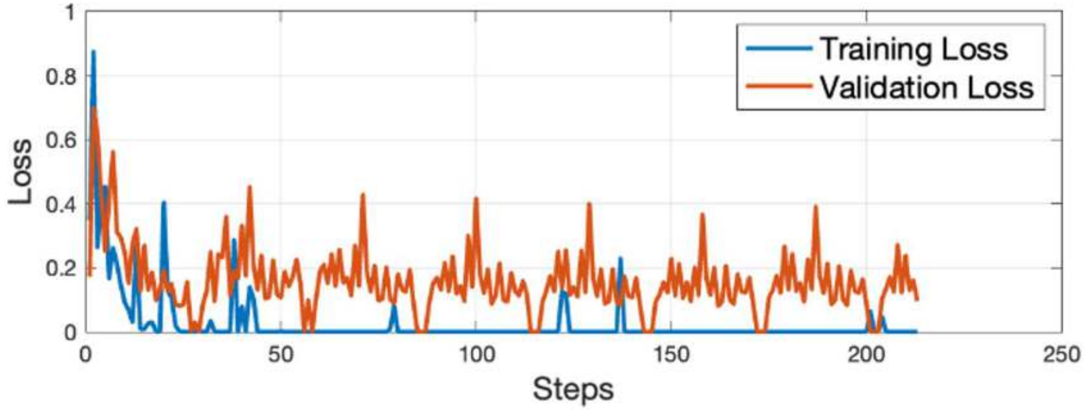
- Dataset: 100 training + 30 test pairs
- Structure: JSON-like format.



Intelligent Data Standardization Module (IDSM)

Data Input Issue Type

Issue Type	Description
Missing Values	Data entries were absent, requiring prediction or marking.
Irregular Data Formats	Sensor data appeared in non-uniform formats needing standardization.
Non-standard Unit Representations	Units of measurement varied, necessitating normalization.
Incomplete Information	Datasets were partially filled, reflecting real-world scenarios.

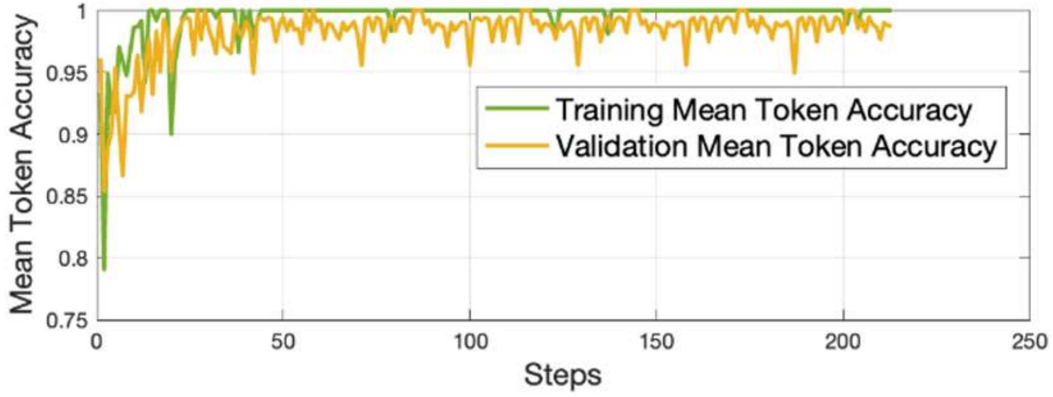


> **Training Loss**
- Decreased from **0.3484** loss to near zero by step **27**.

> **Training Accuracy**
- Increased from **93.38%** to **100%** by step **14**.

> **Validation Loss**
- Started at **0.1724**, briefly increased to **0.6984** at step **2**, then declined to nearly zero by step **27**.

> **Validation Accuracy**
- Increased from **96.14%** to near **100%** by step **27**.



Intelligent Data Standardization Module (IDSM)

Unstandardized Data


```
{
     "time": {
         "time": "Saturday, 04
May 2024 14:00:00 GMT"
     },
     "acc": {
         "x": 0.456,
         "x": 0.123,
         "z": 0.789,
     },
     "mag": {
         "x": -24.321,
         "y": -12.849,
         "z": -0.233,
     },
     "gyro": {
         "x": -0.654,
         "y": 1.234,
         "z": 2.931,
     }
}
```


Converted Data

- Timestamp Format
- Sensor Data Labels
- Data Structure
- Correction of Axis Labels
- Etc.

IDSM

Standardized Data


```
[
    {
        "name": "Accelerometer",
        "time": 1714831200000,
        "values": {
            "x": 0.456,
            "y": 0.123,
            "z": 0.789
        }
    },
    {
        "name": "Magnetometer",
        "time": 1714831200000,
        "values": {
            "x": -24.321,
            "y": -12.849,
            "z": -0.233
        }
    },
    {
        "name": "Gyroscope",
        "time": 1714831200000,
        "values": {
            "x": -0.654,
            "y": 1.234,
            "z": 2.931
        }
    }
]
```

Standardized Dataset **Unit Tests**

> **Overview**

- Validate compliance of sensor data with **predefined JSON schemas**.

> **Validation Function**

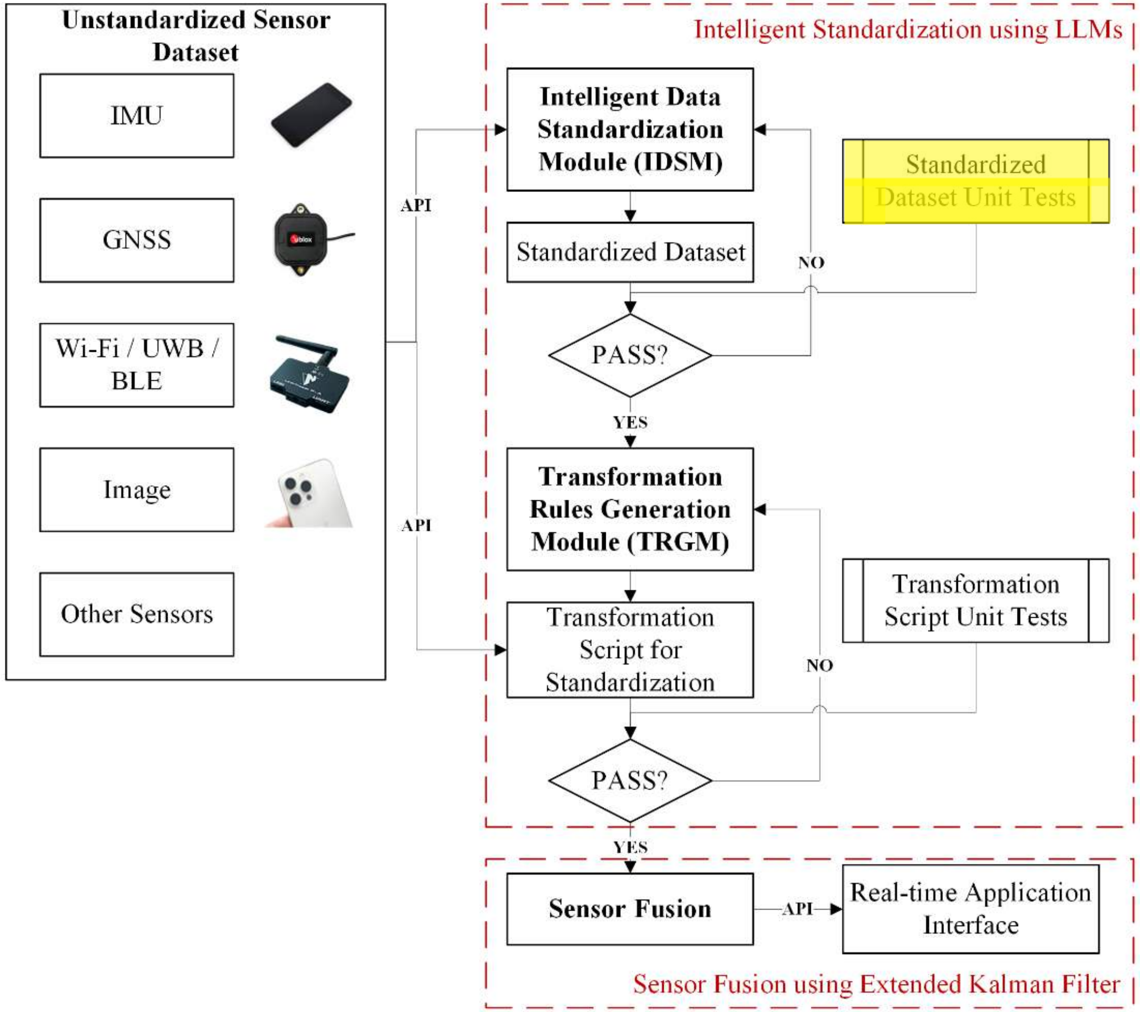
- $(\mathcal{v}, e) = \mathcal{V}_{\text{IDSM}}(\mathcal{S}, \sigma)$

> **Validation Process**

- Library Used:
  - JSON Schema
- Checks Performed:
  - Correct Data Types
  - Required Fields

> **If fail**

- Feed the error string back to LLMs to regenerate standardized JSON string



Most datasets (24 out of 30) required only **one iteration** for successful validation
The process typically completes in **5 iterations**; otherwise, it is regarded as a failure.

Transformation Rules Generation Module (TRGM)

> **Overview**

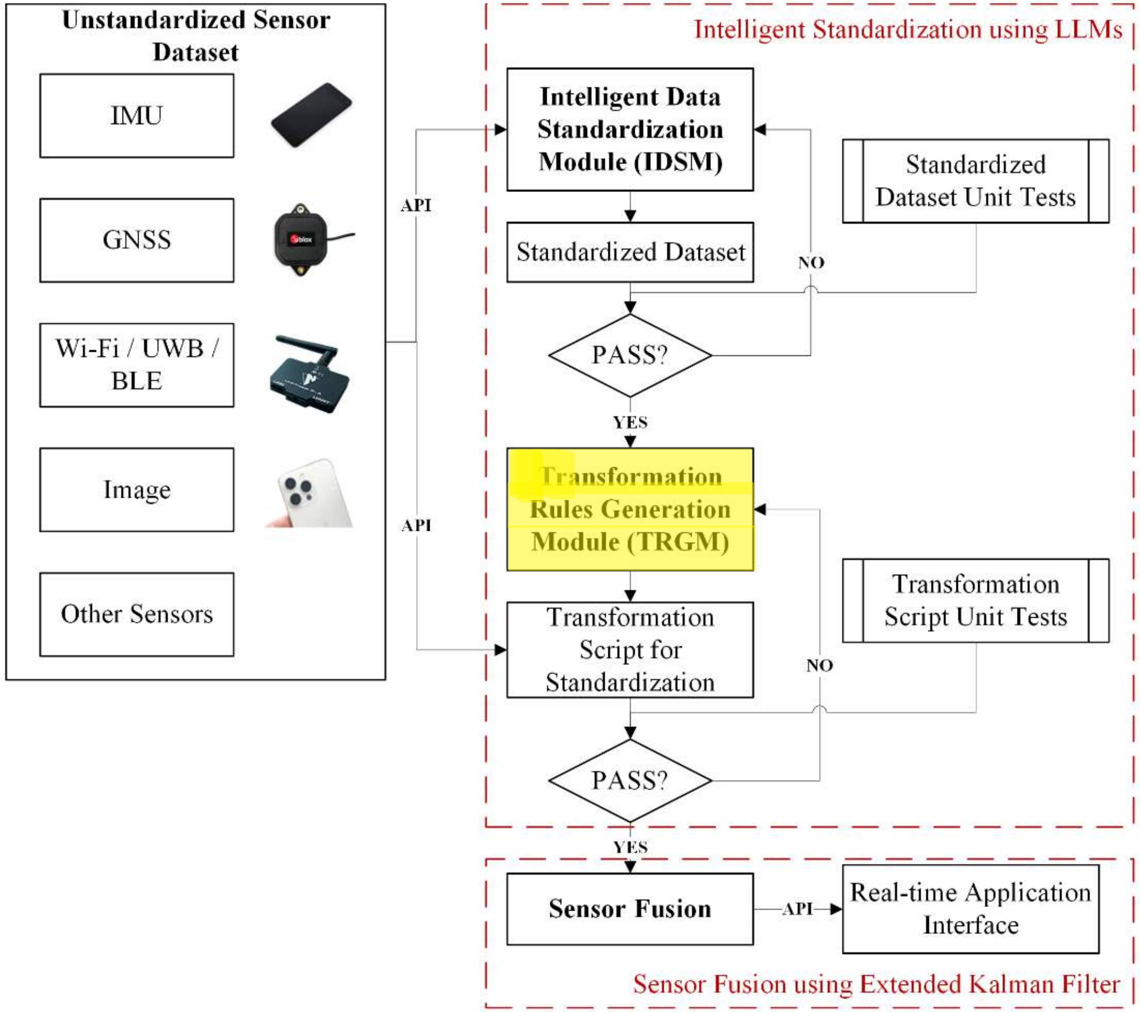
- TRGM leverages **GPT-4-0613** to generate transformation script for standardization

> **Transformation Process**

- $(\mathcal{v}, e) = \mathcal{V}_{\mathrm{TRGM}}(\mathcal{T}, \mathcal{S})$

> **Transformation Rules**

Field Name	Data Type	Description
inputPath	String	JSONPath expression pointing to the source field in the input JSON structure.
outputPath	String	JSONPath expression pointing to the target field in the output JSON structure.
transformation	Function	A function or expression used to transform the input data before mapping it to the output path.
Example		
inputPath	String	$.sensor_data.Accelerometer.timestamp
outputPath	String	$[?(@.name == 'Accelerometer')].time
transformation	Function	float(re.search(r'[-+]?*?+', value).group(0))



Transformation Script **Unit Tests**

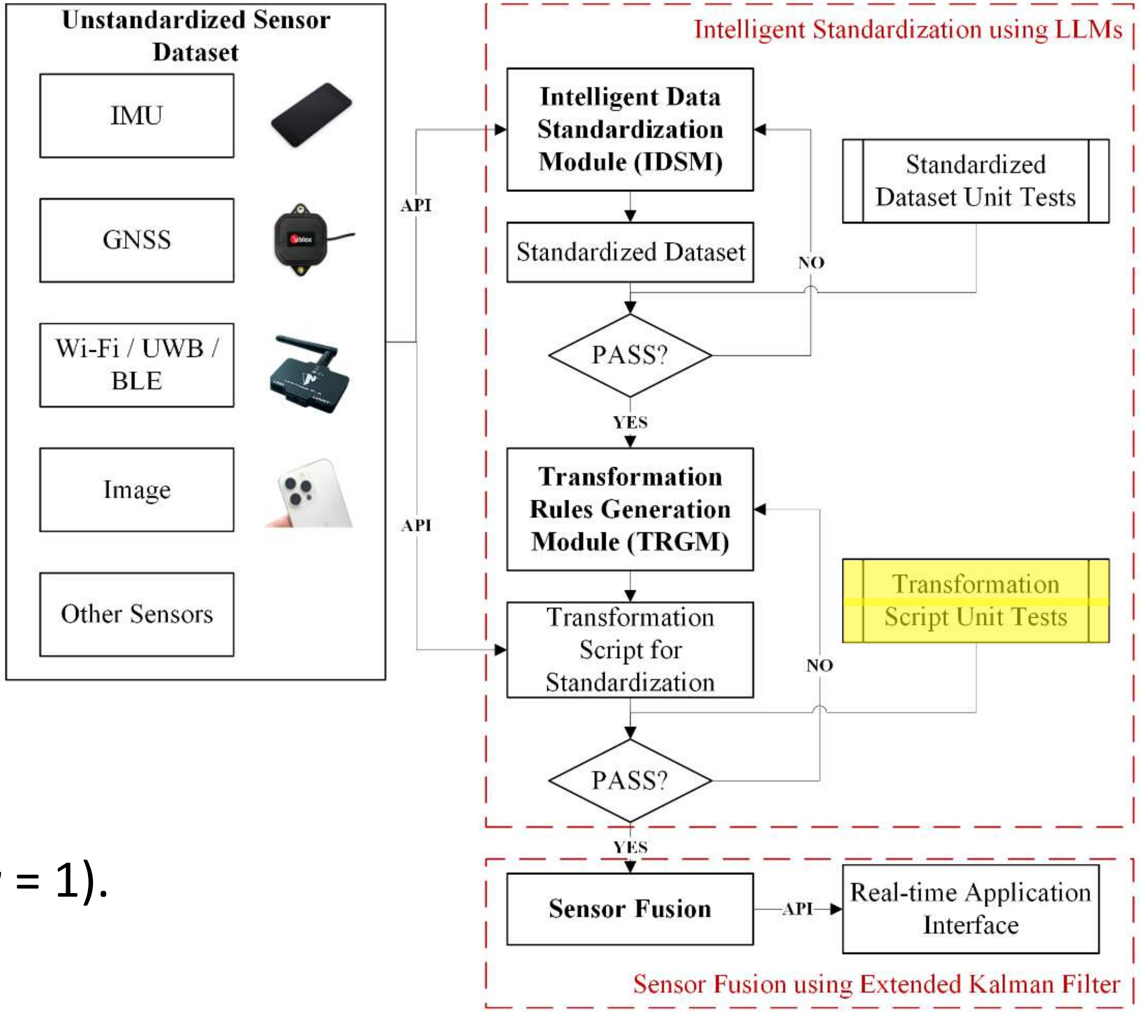
> **Overview**

- Ensure accuracy and functionality of transformation scripts generated by LLMs.
- Scripts convert input JSON data (I) into the standardized format (S).

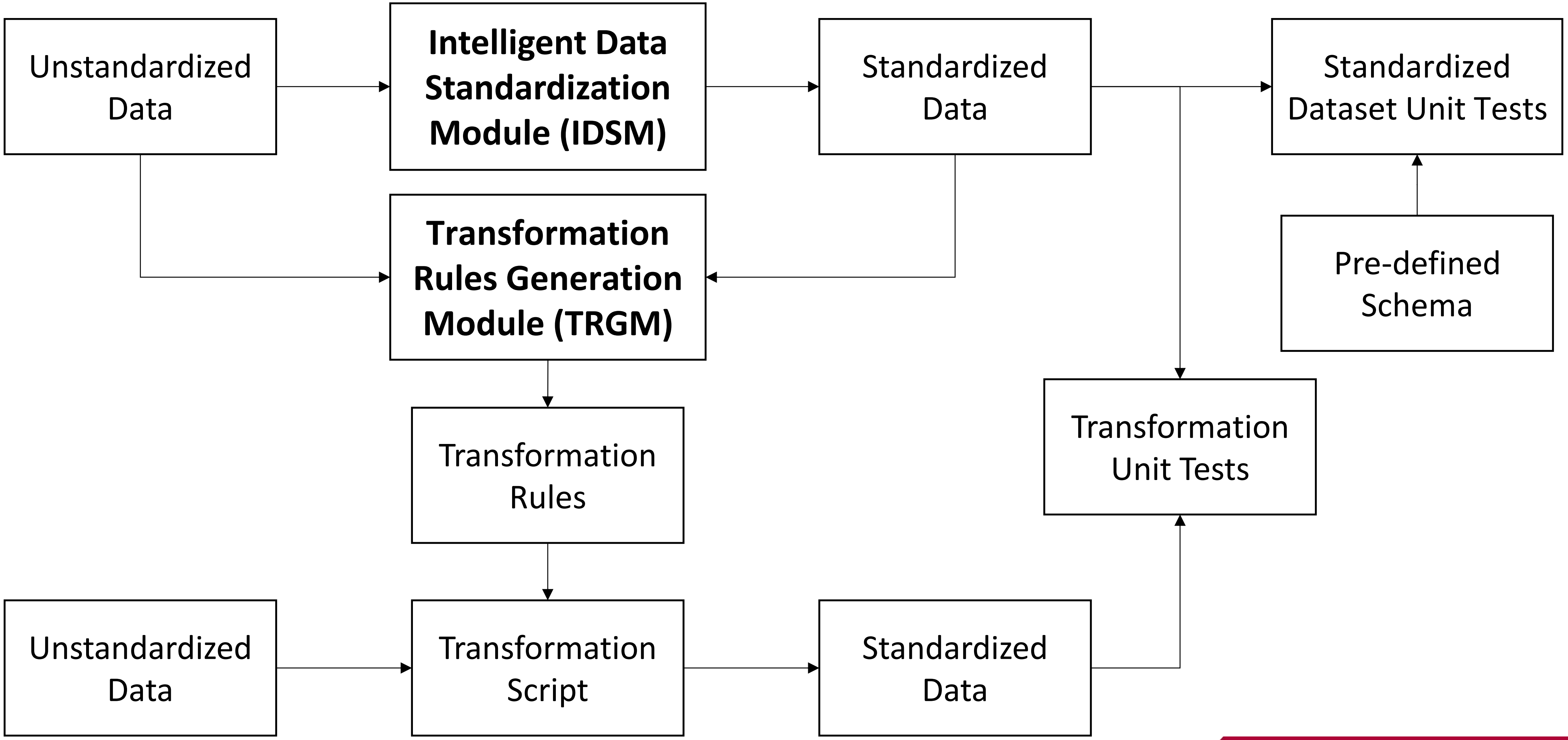
> **Validation Function**

- $(\nu, e) = \mathcal{V}_{TRGM}(\mathcal{T}(\mathrm{I}), \mathcal{S})$
- $\mathcal{T} = \boldsymbol{F}_{TRGM}(\boldsymbol{S}, \boldsymbol{I}, e)$

Process repeats until no errors are detected (ν = 1).



Proposed Framework



Unstandardized Data


```json
{
  "sensor_data": {
    "Accelerometer": {
      "timestamp": 1683302400000,
      "readings": {
        "x": 9.81,
        "y": 0.02,
        "z": -0.03
      }
    },
    "Magnetometer": {
      "timestamp": 1683302400000,
      "accuracy_level": 2,
      "coordinates": {
        "x": 0.012,
        "y": -0.030,
        "z": 0.022
      }
    },
    "Gyroscope": {
      "timestamp": 1683302400000,
      "axis": {
        "x": 1.23,
        "y": 0.45,
        "z": -0.67
      }
    }
  }
}
```


IDSM

Standardized Data


```json
[
  {
    "name": "Accelerometer",
    "time": 1683302400000,
    "values": {
      "x": 9.81,
      "y": 0.02,
      "z": -0.03
    }
  },
  {
    "name": "Magnetometer",
    "time": 1683302400000,
    "accuracy": 2,
    "values": {
      "x": 0.012,
      "y": -0.030,
      "z": 0.022
    }
  },
  {
    "name": "Gyroscope",
    "time": 1683302400000,
    "values": {
      "x": 1.23,
      "y": 0.45,
      "z": -0.67
    }
  }
]
```


Transformation Rules


```json
{
  "rules": [
    {
      "inputPath": "$.sensor_data.Accelerometer.timestamp",
      "outputPath": "$[?(@.name == 'Accelerometer')].time"
    },
    {
      "inputPath": "$.sensor_data.Accelerometer.readings.x",
      "outputPath": "$[?(@.name == 'Accelerometer')].values.x"
    },
    {
      "inputPath": "$.sensor_data.Magnetometer.timestamp",
      "outputPath": "$[?(@.name == 'Magnetometer')].time"
    },
    {
      "inputPath": "$.sensor_data.Magnetometer.coordinates.x",
      "outputPath": "$[?(@.name == 'Magnetometer')].values.x"
    },
    {
      "inputPath": "$.sensor_data.Gyroscope.timestamp",
      "outputPath": "$[?(@.name == 'Gyroscope')].time"
    },
    {
      "inputPath": "$.sensor_data.Gyroscope.axis.x",
      "outputPath": "$[?(@.name == 'Gyroscope')].values.x"
    }
  ]
}
```


TRGM

Transformation Rules

Generic Transformation Script

```
def apply_transformation(input_JSON, transformation_rules):
    """Applies transformation rules to input JSON and produces output JSON."""
    output_JSON = {}

    for rule in transformation_rules["rules"]:
        # Convert inputPath to list of keys for dictionary access
        input_keys = rule['inputPath'].replace('$.', '').split('.')
        value = get_from_dict(input_JSON, input_keys)

        # Check if a transformation is needed and apply it
        if "transformation" in rule:
            transformation_code = rule["transformation"]
            value = eval(transformation_code)

        # Convert outputPath to list of keys and set value in output JSON
        output_keys = rule['outputPath'].replace('$.', '').split('.')
        set_in_dict(output_JSON, output_keys, value)

    return output_JSON
```

"speed": "1.5 m/s"

Transformation Rules

```
{
"rules": [
{"inputPath": "$.name", "outputPath": "$.name"},
{"inputPath": "$.time", "outputPath": "$.time"},
{"inputPath": "$.values.latitude", "outputPath": "$.values.latitude"},
{"inputPath": "$.values.longitude", "outputPath": "$.values.longitude"},
{"inputPath": "$.values.altitude", "outputPath": "$.values.altitude"},
{
"inputPath": "$.values.speed",
"outputPath": "$.values.speed",
"transformation": "float(re.search(r'[-+]?\\d*\\.?\\d+', value).group(0))"
},
{"inputPath": "$.values.speedAccuracy", "outputPath": "$.values.speedAccuracy"},
{"inputPath": "$.values.bearingAccuracy", "outputPath": "$.values.bearingAccuracy"},
{"inputPath": "$.values.horizontalAccuracy", "outputPath": "$.values.horizontalAccuracy"},
{"inputPath": "$.values.verticalAccuracy", "outputPath": "$.values.verticalAccuracy"},
{"inputPath": "$.values.bearing", "outputPath": "$.values.bearing"}
]
}
```

Change in Location

Change in Value

Extended Kalman Filter (EKF)

> **Method:**

- The standardized data is passed to the Extended Kalman Filter (EKF) for sensor fusion.

> **Outcome:**

- The EKF integrates data from multiple sensors (GNSS, UWB, IMU, BLE…) to provide real-time positional and velocity estimates.

> **State Transition Matrix & State Vector**

$$\mathbf{F}_k = \begin{bmatrix} \mathbf{I}_{3\times3} & \Delta t \mathbf{I}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \end{bmatrix}$$

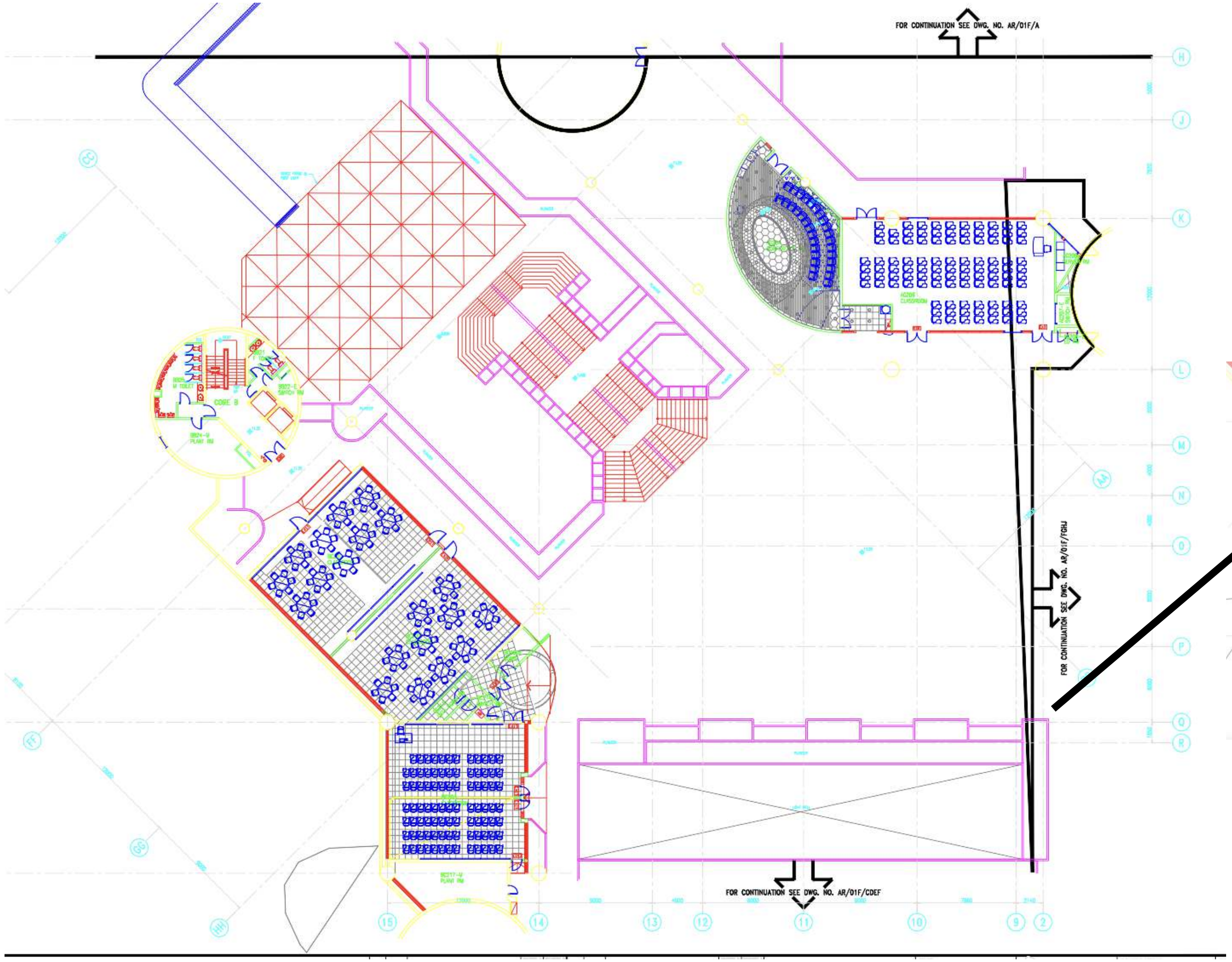
Sensor Type	Measurement Vector	Measurement Matrix Covariance
GNSS Receiver	$\mathbf{z}_{\text{GNSS}} = \begin{bmatrix} \phi_{\text{GNSS}} \\ \lambda_{\text{GNSS}} \\ h_{\text{GNSS}} \end{bmatrix}$	$\mathbf{R}_{\text{GNSS}} = \begin{bmatrix} 655.00 & 0 & 0 \\ 0 & 655.00 & 0 \\ 0 & 0 & 655.00 \end{bmatrix}$
UWB Sensor	$\mathbf{z}_{\text{UWB}} = \begin{bmatrix} x_{\text{UWB}} \\ y_{\text{UWB}} \\ z_{\text{UWB}} \end{bmatrix}$	$\mathbf{R}_{\text{UWB}} = \begin{bmatrix} 1.00 & 0 & 0 \\ 0 & 1.00 & 0 \\ 0 & 0 & 1.00 \end{bmatrix}$
Camera	$\mathbf{z}_{\text{cam}} = img_{\text{cam}}$	$\mathbf{R}_{\text{cam}} = \begin{bmatrix} 0.15 & 0 & 0 \\ 0 & 0.15 & 0 \\ 0 & 0 & 0.15 \end{bmatrix}$

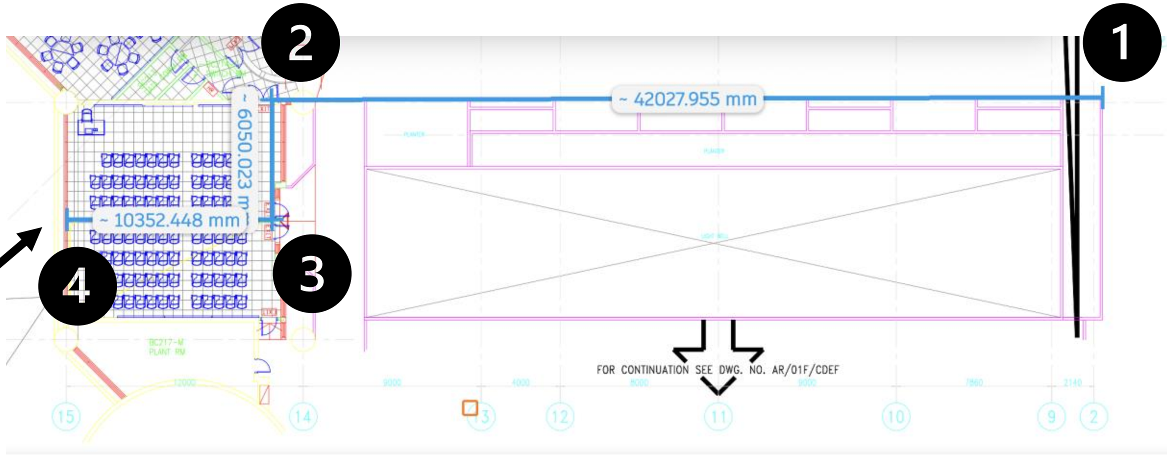
Sensor Type	Control Input Vector	Control Input Matrix	Process Noise Covariance
IMU	$\mathbf{u}_{\text{IMU}} = \begin{bmatrix} a_x \\ a_y \\ a_z \\ \omega_x \\ \omega_y \\ \omega_z \\ m_x \\ m_y \\ m_z \end{bmatrix}$	$\mathbf{B}_{\text{IMU}} = \begin{bmatrix} \frac{\Delta t^2}{2} \mathbf{I}_{3\times3} \\ \Delta t \mathbf{I}_{3\times3} \end{bmatrix}$	$\mathbf{Q} = \begin{bmatrix} \frac{\sigma_a^2 \Delta t^4}{4} \mathbf{I}_{3\times3} & \frac{\sigma_a^2 \Delta t^3}{2} \mathbf{I}_{3\times3} \\ \frac{\sigma_a^2 \Delta t^3}{2} \mathbf{I}_{3\times3} & \sigma_a^2 \Delta t^2 \mathbf{I}_{3\times3} \end{bmatrix}$

Experiment

Experiment Setup

The experiment took place in a seamless 60-meter environment





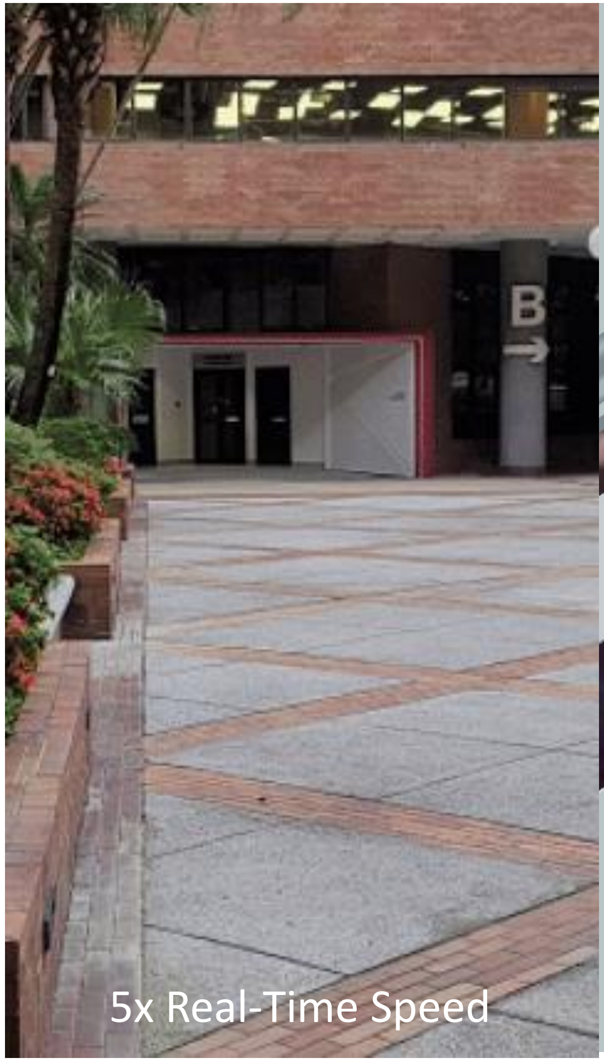
Experiment Setup

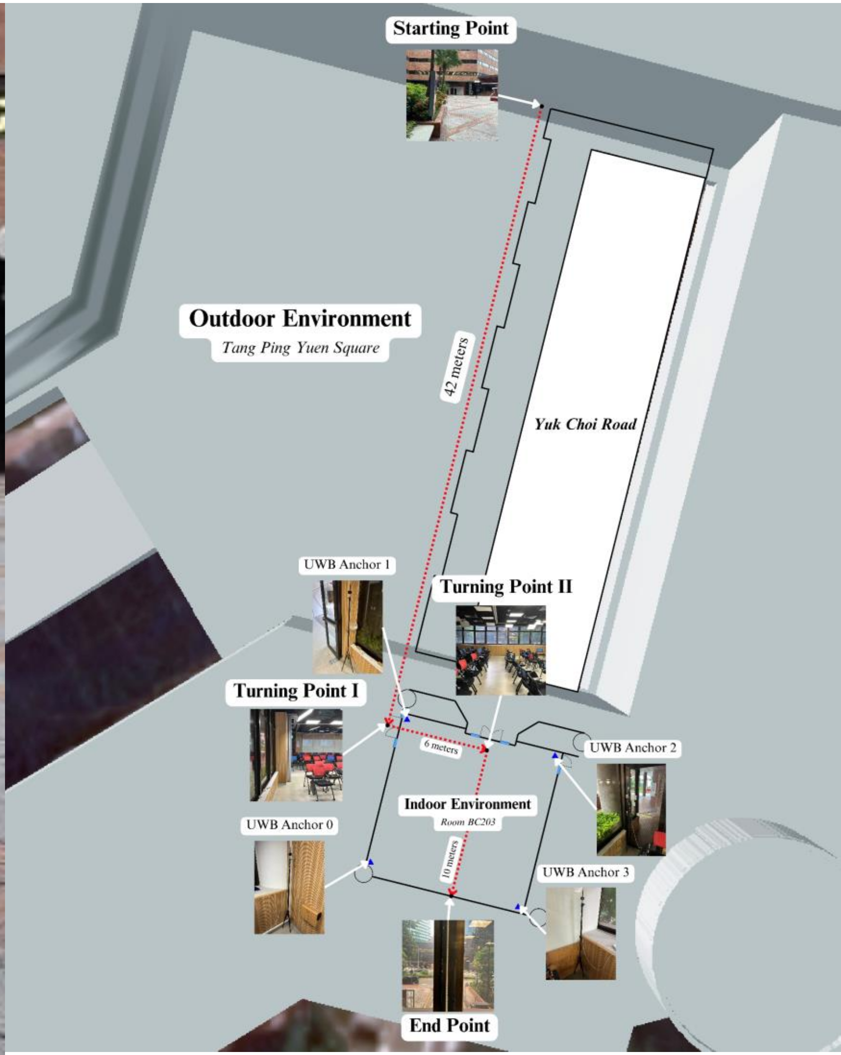
> **Streaming Sensors:**

- GNSS [11]
  - U-blox F9P
  - iPhone 14 Pro
- UWB [12]
  - Nooploop LinkTrack P-A Series
- VPS [13 – 14]
  - Samsung Galaxy Note 20 Ultra
- IMU [15]
  - iPhone 14 Pro

> **Location**

- Tang Ping Yuen Square + BC203

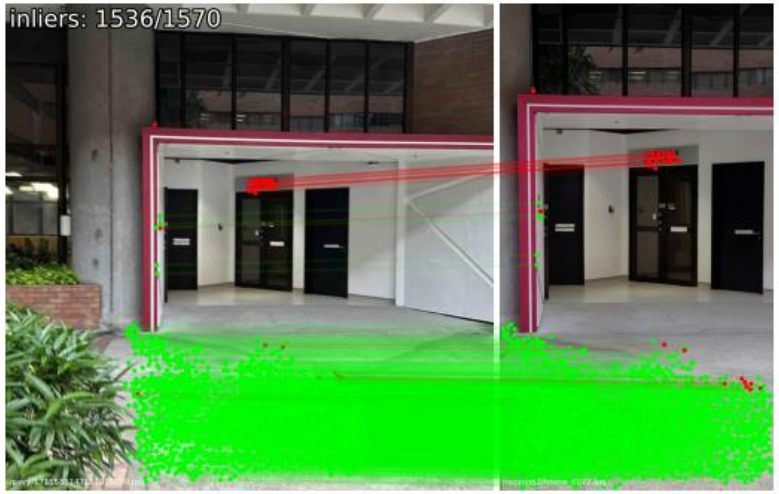


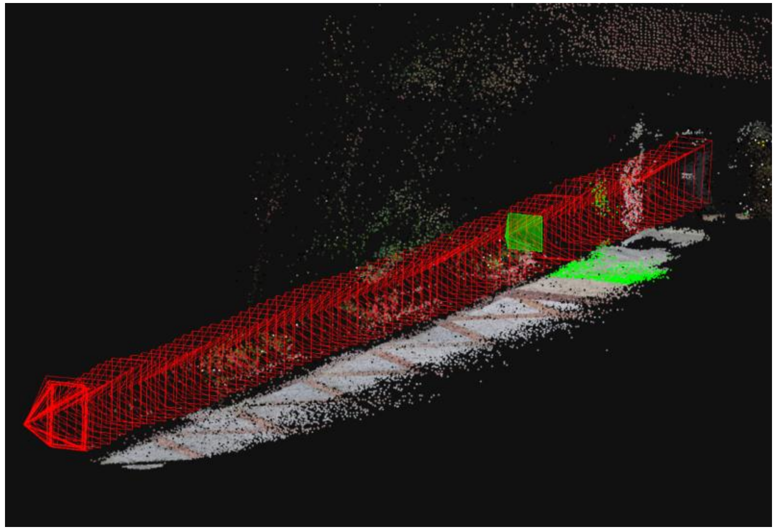


[11] "ZED-F9P module," U-blox. https://www.u-blox.com/en/product/zed-f9p-module (accessed May 21, 2024).
[12] "UWB High-Precision Positioning: LinkTrack P-A Series," Nooploop.https://www.nooploop.com/en/linktrack/ (accessed May 21, 2024).
[13] P. -E. Sarlin, C. Cadena, R. Siegwart and M. Dymczyk, "FromCoarse to Fine: Robust Hierarchical Localization at Large Scale,"2019 IEEE/CVF Conference on Computer Vision and Pattern Recog-nition (CVPR), Long Beach, CA, USA, 2019, pp. 12708-12717, doi:10.1109/CVPR.2019.01300.
[14] P. -E. Sarlin, D. DeTone, T. Malisiewicz and A. Rabinovich,"SuperGlue: Learning Feature Matching With Graph Neural Net-works," 2020 IEEE/CVF Conference on Computer Vision and PatternRecognition (CVPR), Seattle, WA, USA, 2020, pp. 4937-4946, doi:10.1109/CVPR42600.2020.00499.
[15] K. T. H. Choi, "tszheichoi / awesome-sensor-logger," GitHub.https://github.com/tszheichoi/awesome-sensor-logger/.

Experiment Setup

Visual Positioning System*





*Compromised performance in indoor environments.

Ultra-wideband

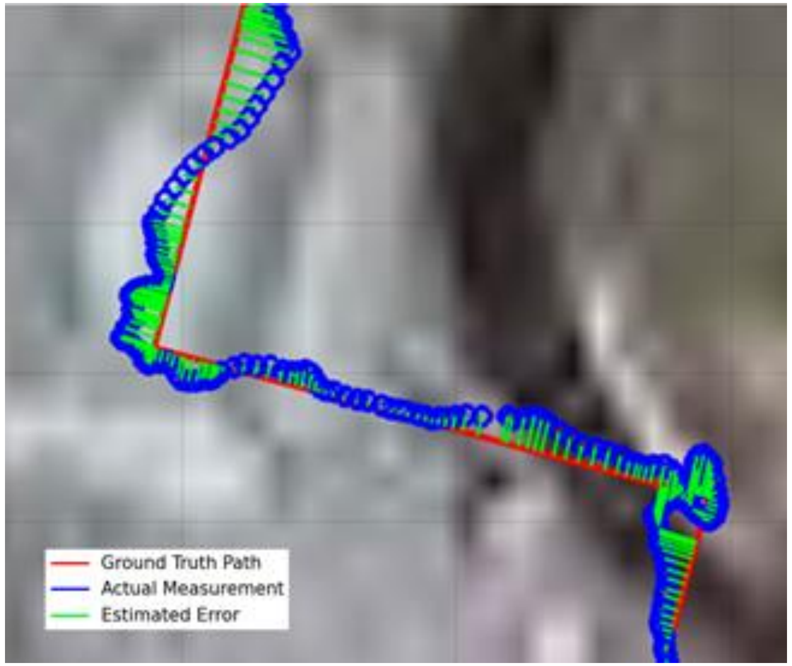
Experiment Setup

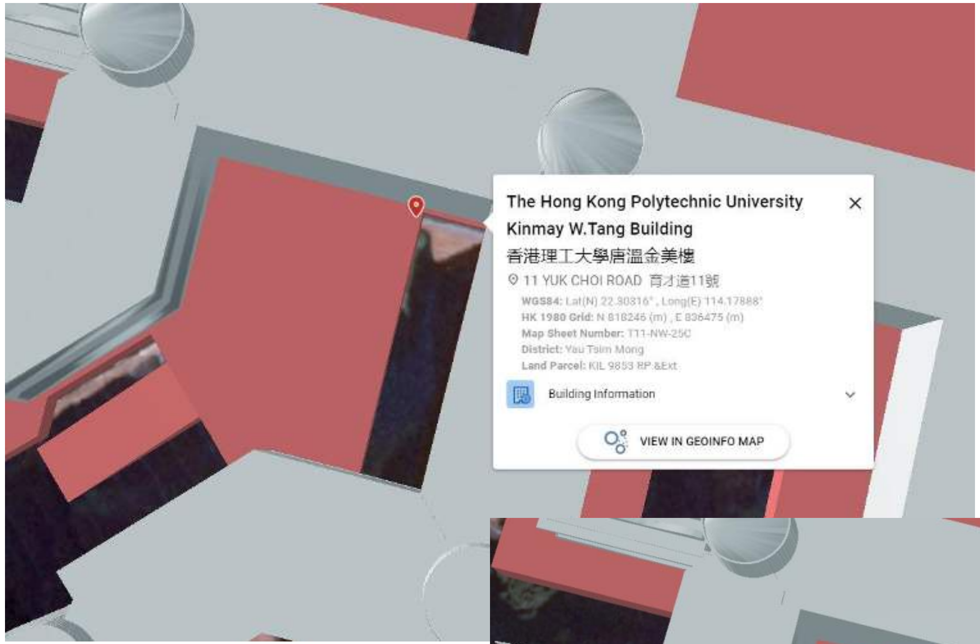
> **Ground Truth**

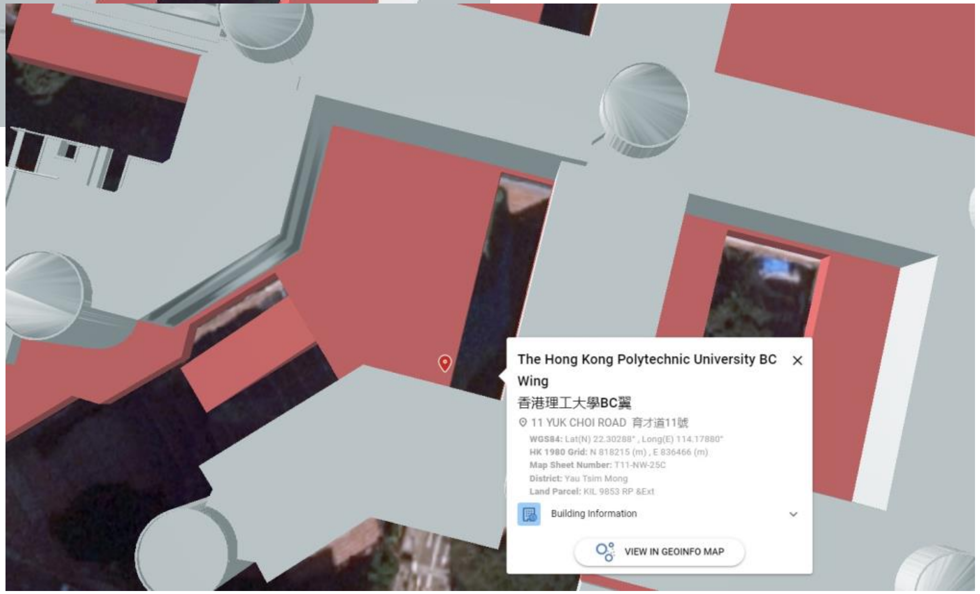
- HKSAR Gov's GeoInfo Map
- Detailed Floor Plan

> **Absolute Trajectory Error (ATE)**

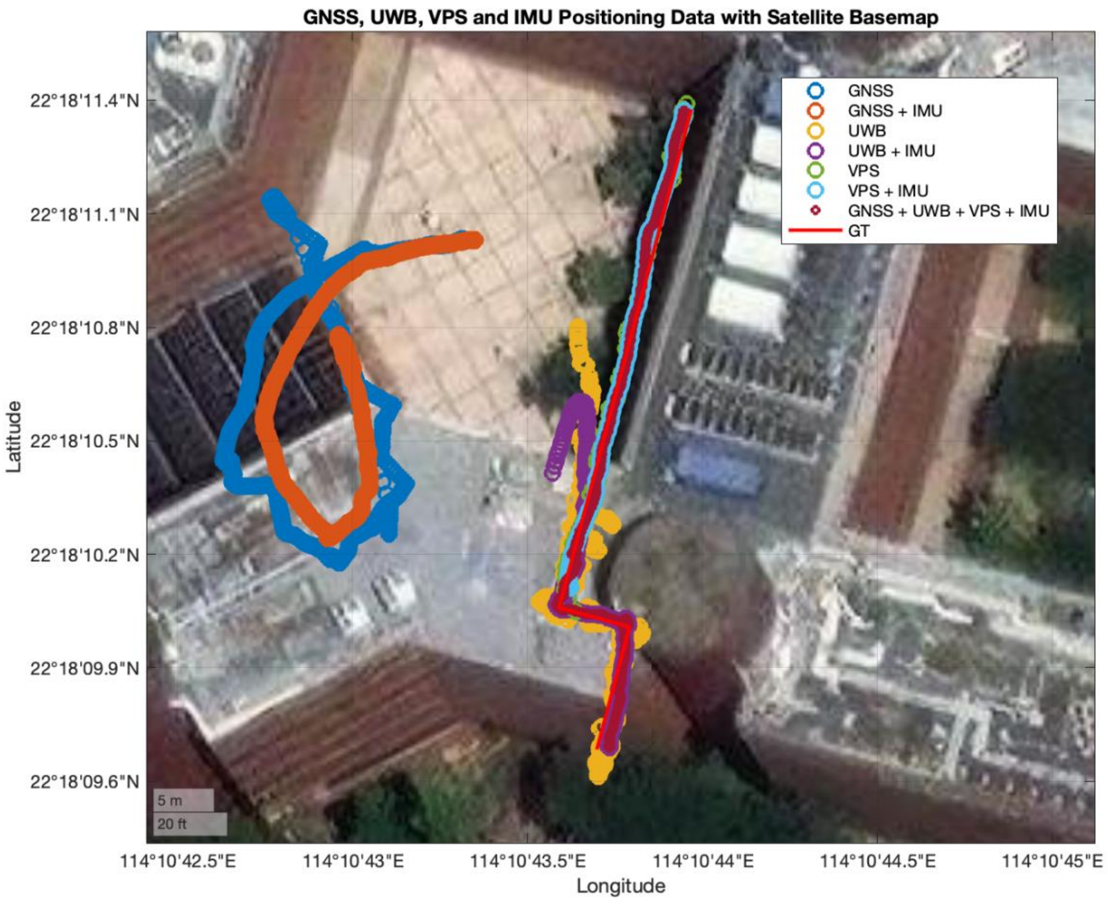
- $E = \sqrt{(x_m - x_t)^2 + (y_m - y_t)^2}$



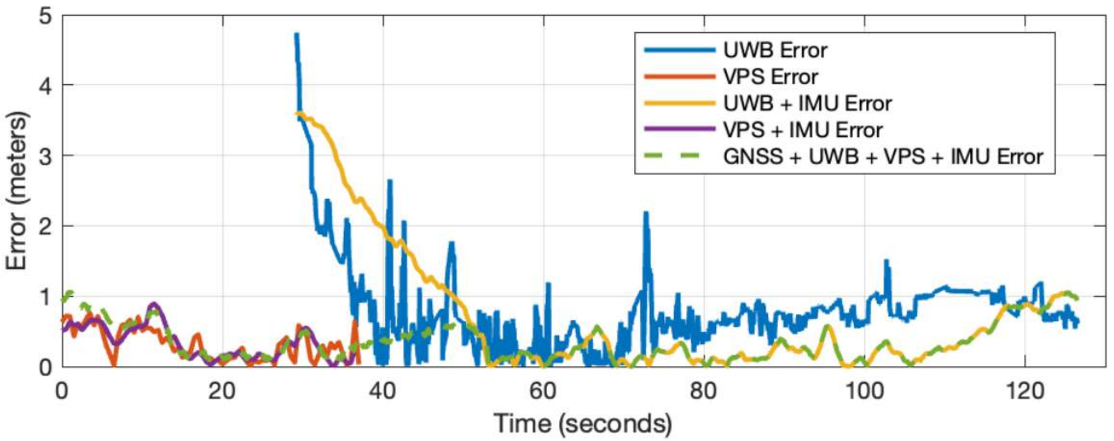




Experiment Results



Method	Mean Error (m)	Std. Dev. (m)	RMSE (m)	Median Error (m)	Max Error (m)
GNSS	25.02	5.47	25.61	25.21	33.75
GNSS + IMU	23.24	4.22	23.62	23.38	29.40
UWB	0.79	0.62	1.00	0.71	4.75
UWB + IMU	0.69	0.89	1.13	0.31	3.92
VPS	0.32	0.23	0.39	0.30	0.76
VPS + IMU	0.33	0.23	0.41	0.31	0.91
GNSS + VPS + UWB + IMU	**0.33**	**0.24**	**0.41**	**0.27**	**0.95**



Sensor data from multiple devices can be seamlessly fused under our proposed framework to achieve optimized positioning results.

Conclusion

> **Automated Data Standardization**
- Demonstrated the feasibility of real-time standardization with large language models.

> **IDSM Performance**
- Achieved near-zero loss and nearly full accuracy in training/validation.
- Standardized data across diverse sensor inputs.

> **TRGM Efficiency**
- Significantly reduced manual effort in script generation.
- Enhanced productivity, minimized human intervention.

> **Limitations**
- Reliant on predefined schemas and manual covariance matrices.
- Controlled settings may not fully reflect real-world complexities.

Future Vision

> **Broad Future & Spectrum of Application**

- **Expanding Potential:** LLMs demonstrate vast potential across diverse fields.
- **Sensor Integration:** LLMs can be integrated with various sensor technologies, opening new possibilities for real-time applications in seamless positioning.
- **Contextual Capabilities:** LLMs enhance the understanding of their environment, unlocking new possibilities for interaction and application.

> **Rising Performance of Large Language Models.**

- **Latency Optimization:** Efforts are ongoing to reduce response times, enhancing the efficiency of LLMs in real-world applications.
- **Improved Accuracy:** LLMs are becoming more precise, providing increasingly reliable outputs in various use cases.

Acknowledgement

This research is supported by the University Grants Committee (UGC) of Hong Kong under the Research Impact Fund scheme for the project R5009-21, titled *Reliable Multiagent Collaborative Global Navigation Satellite System Positioning for Intelligent Transportation Systems*.

Thank you for your attention!

Questions, Comments and Collaboration are welcome.

Max Jwo Lem Lee / Ju Lin / Li-Ta Hsu

max.jl.lee@connect.polyu.hk / ju.lin@connect.polyu.hk / lt.hsu@polyu.edu.hk

Department of Aeronautical and Aviation Engineering,

The Hong Kong Polytechnic University

Appendix - Intelligent Data Standardization Module

> **Data Standardization Process**

- The IDSM's primary objective is to transform heterogeneous sensor data D into a standardized format S. The raw data collected from various sensors is represented as
  - Input Data: $\mathbf{D}_i = \{d_{i1}, d_{i2}, \ldots, d_{in}\}$.
- where di denotes the data from the i-th sensor. The standardization process is expressed as:
  - Standardized Data: $\mathbf{S}_i = \mathrm{IDSM}(\mathbf{D}_i)$

> **Unit Tests Validation Function**

- The unit tests validate the standardized dataset S against predefined JSON schemas for various sensor types. The validation function is expressed as:
  - $(\nu, e) = \mathcal{V}_{\mathrm{IDSM}}(\mathcal{S}, \sigma)$
- where ν indicates whether S conforms to schema σ, ande contains details of validation errors. The iterative process ensures S conforms to the schema through multiple cycles if necessary, up to a maximum of five iterations, based on empirical evidence and practical considerations.

Appendix - Transformation Rules Generation Module

> **Transformation Process**

- It converts input JSON files into a specified output format, reducing manual intervention. The process is represented as:
  - $(v, e) = \mathcal{V}_{\mathrm{TRGM}}(\mathcal{T}, \mathcal{S})$
- where S is the standardized data, and I is the input JSON structure. The transformation rules are then used to create a ”Transformation Script for Standardization” T , ensuring consistency in data transformation tasks.

> **Unit Tests Validation Function**

- The unit tests validate the accuracy and functionality of thetransformation scripts generated by the LLM. These scriptsare essential for converting the input JSON data (I) intothe desired standardized format (S). The validation function,which assesses whether the transformation scripts perform asexpected, is formally expressed as:
  - $(v, e) = \mathcal{V}_{TRGM}(\mathcal{T}(\mathrm{I}), \mathcal{S})$
- Here, T (I) represents the output generated by applyingthe transformation script T to the input JSON structure I,v indicates the success of the transformation in matchingthe standardized data schema S, and e details any errorsencountered during the process. If discrepancies or errors($e \neq 0$) are identified, these errors are fed back into thetransformation function:
  - $\mathcal{T} = \boldsymbol{F}_{TRGM}(\boldsymbol{S}, \boldsymbol{I}, e)$