AAE45 FYP Oral Presentation

FusionFly: A Scalable Open-Source Framework for AI-Powered Positioning Data Standardization

Ju Lin, Lingyao Zhu

Department of Aeronautical and Aviation Engineering
The Hong Kong Polytechnic University

ju.lin@connect.polyu.hk
ling-yao.zhu@connect.polyu.hk

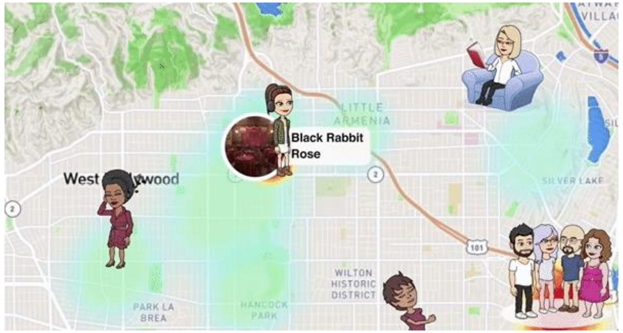
Supervisor: Prof. Li-Ta Hsu
lt.hsu@polyu.edu.hk

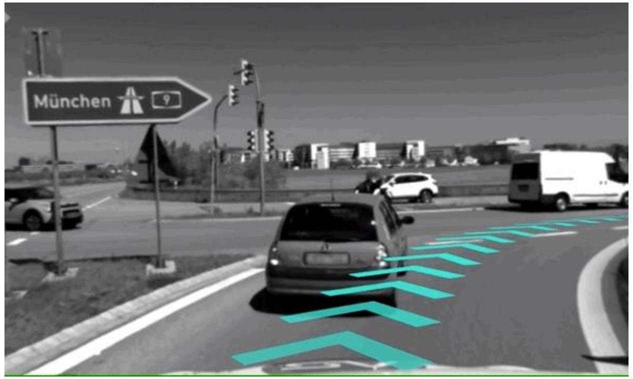
GitHub

Wiki

April 14th, 2025

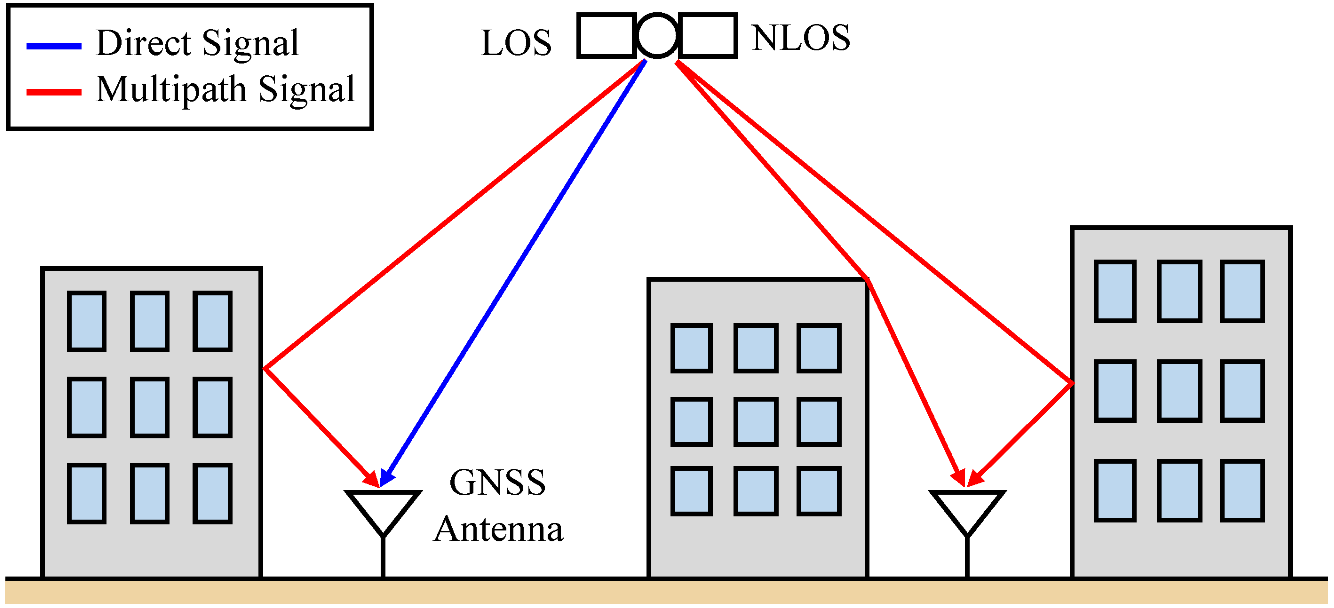
We need positioning & navigation everywhere





We live in Hong Kong, a highly dense city

GNSS is not always reliable



Multipath **NLOS Reception**

Satellite Positioning and Navigation

- Provided a foundational understanding of GNSS and sensor integration
- Explained how GNSS gives accurate outdoor positioning but struggles in urban areas due to multipath interference (Groves, 2013)
- GNSS performance drastically reduces in areas with tall buildings, where signals are often blocked (Xu et al., 2020)

- Explored combing GNSS with IMUs and visual odometry (Yang & Li, 2015)

↓ ↓

Track motion Estimate movement

Extended Kalman Filter

- Sensor fusion usually involves careful synchronization and calibration of sensor data
- Complex and time-consuming (Groves, 2013) (Zhao et al., 2019)

- Introduced an adaptive Kalman filter to reduce the manual effort involved in adjusting parameters
- Some manual tuning was still necessary (Yang et al., 2022)

Disadvantages

Limited Historical Data Utilization

Assumption of Gaussian Noise

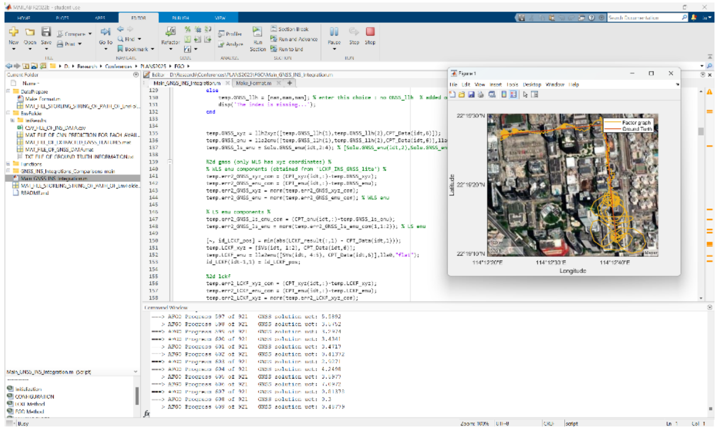
Sensitivity to Initial Estimates

Sensitivity to Outliers

Factor Graph Optimization

- FGO can handle complex sensor interactions better than EKF, especially in dynamic environments (Wang et al., 2021)

- Explored hybrid approaches combining EKF and FGO
- Hybrid methods could balance computational efficiency and accuracy effectively (Chen et al., 2022)



```
        else
            temp.GNSS_llh = [nan,nan,nan]; % enter this choice : no GNSS_llh  % added o
            disp('The index is missing...');
        end


        temp.GNSS_xyz = llh2xyz([temp.GNSS_llh(1),temp.GNSS_llh(2),CPT_Data(idt,6)]);
        temp.GNSS_enu = lla2enu([temp.GNSS_llh(1),temp.GNSS_llh(2),CPT_Data(idt,6)],lla
        temp.GNSS_ls_enu = Solu.GNSS_enu(idt,2:4); % [Solu.GNSS_enu(idt,2),Solu.GNSS_en

        %2d gnss (only WLS has xyz coordinates) %
        % WLS enu components (obtained from 'LCKF_INS_GNSS_lite') %
        temp.err2_GNSS_xyz_com = (CPT_xyz(idt,:)-temp.GNSS_xyz);
        temp.err2_GNSS_enu_com = (CPT_enu(idt,:)-temp.GNSS_enu);
        temp.err2_GNSS_xyz = norm(temp.err2_GNSS_xyz_com);
        temp.err2_GNSS_enu = norm(temp.err2_GNSS_enu_com); % WLS enu

        % LS enu components %
        temp.err2_GNSS_ls_enu_com = (CPT_enu(idt,:)-temp.GNSS_ls_enu);
        temp.err2_GNSS_ls_enu = norm(temp.err2_GNSS_ls_enu_com(1,1:2)); % LS enu

        [~, id_LCKF_pos] = min(abs(LCKF_result(:,1) - CPT_Data(idt,1)));
        temp.LCKF_xyz = [SVs(idt, 1:2), CPT_Data(idt,6)];
        temp.LCKF_enu = lla2enu([SVs(idt, 4:5), CPT_Data(idt,6)],lla0,"flat");
        id_LCKF(idt-1,1) = id_LCKF_pos;

        %2d lckf
        temp.err2_LCKF_xyz_com = (CPT_xyz(idt,:)-temp.LCKF_xyz);
        temp.err2_LCKF_enu_com = (CPT_enu(idt,:)-temp.LCKF_enu);
        temp.err2_LCKF_xyz = norm(temp.err2_LCKF_xyz_com);
```

```
---> AFGO Progress 597 of 921   GNSS solution uct: 5.5892
   > AFGO Progress 598 of 921   GNSS solution uct: 3.6752
===> AFGO Progress 599 of 921   GNSS solution uct: 3.2974
===> AFGO Progress 600 of 921   GNSS solution uct: 3.4341
   > AFGO Progress 601 of 921   GNSS solution uct: 3.4717
---> AFGO Progress 602 of 921   GNSS solution uct: 0.41372
===> AFGO Progress 603 of 921   GNSS solution uct: 2.9271
===> AFGO Progress 604 of 921   GNSS solution uct: 4.2498
   > AFGO Progress 605 of 921   GNSS solution uct: 3.5977
---> AFGO Progress 606 of 921   GNSS solution uct: 7.0972
===> AFGO Progress 607 of 921   GNSS solution uct: 0.81378
---> AFGO Progress 608 of 921   GNSS solution uct: 0.3
   > AFGO Progress 609 of 921   GNSS solution uct: 0.48779
```

Our FGO algorithm is based on Li et al.'s open-source package

https://github.com/ZhengdaoLI0602/GNSS_INS_Integrations_Comparisons

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?

Unit Discrepancies

- Deg vs Rad?
- Cartesian or Polar Coordinate?

Any Trouble?

Unit Discrepancies

- Deg vs Rad?
- Cartesian or Polar Coordinate?

Time Synchronization

- UNIX or YYYY-MM-DDTHH:mm:ss.sssZ?
- UTC time or Local time?
- Asynchronous sampling?

Any Trouble?

Unit Discrepancies

- Deg vs Rad?
- Cartesian or Polar Coordinate?

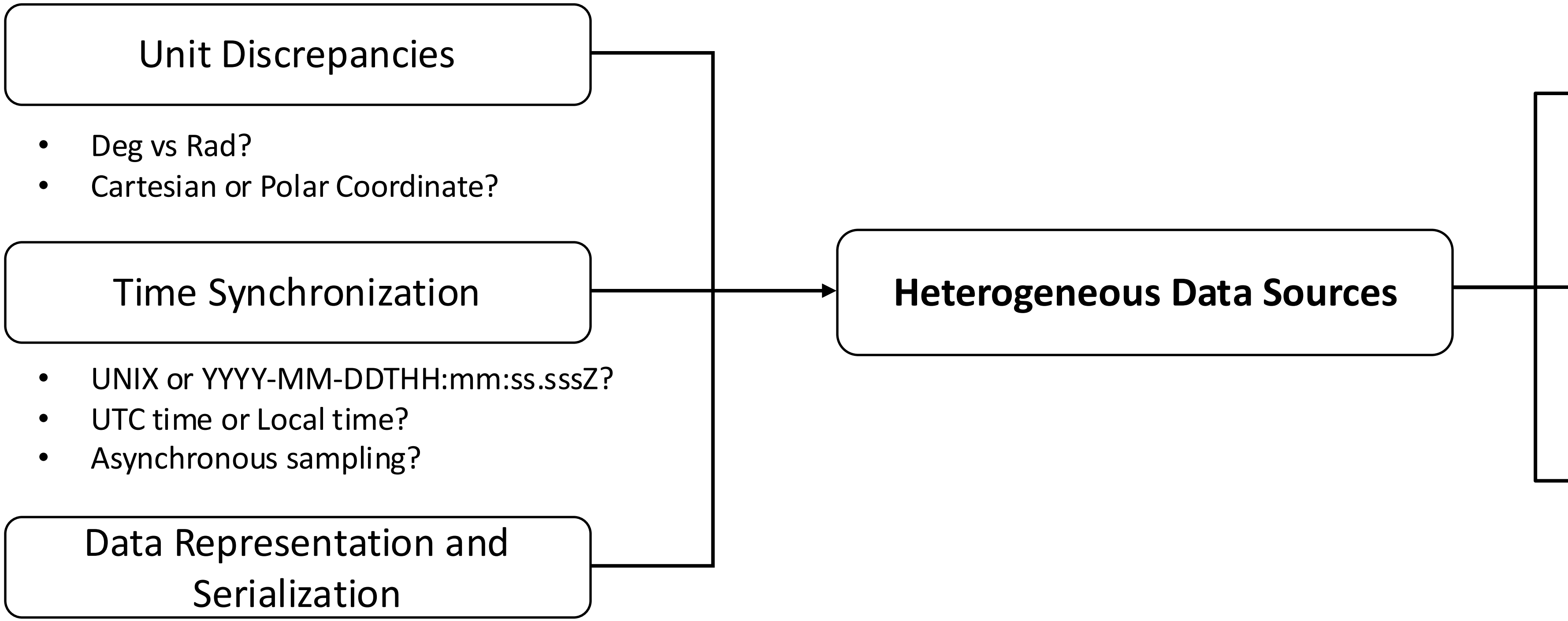
Time Synchronization

- UNIX or YYYY-MM-DDTHH:mm:ss.sssZ?
- UTC time or Local time?
- Asynchronous sampling?

Data Representation and Serialization

- File format in NMEA, CSV, or binary?
- Missing or inconsistent metadata?

Any Trouble?


Unit Discrepancies
Time Synchronization
Data Representation and Serialization
Heterogeneous Data Sources

Unit Discrepancies

- Deg vs Rad?
- Cartesian or Polar Coordinate?

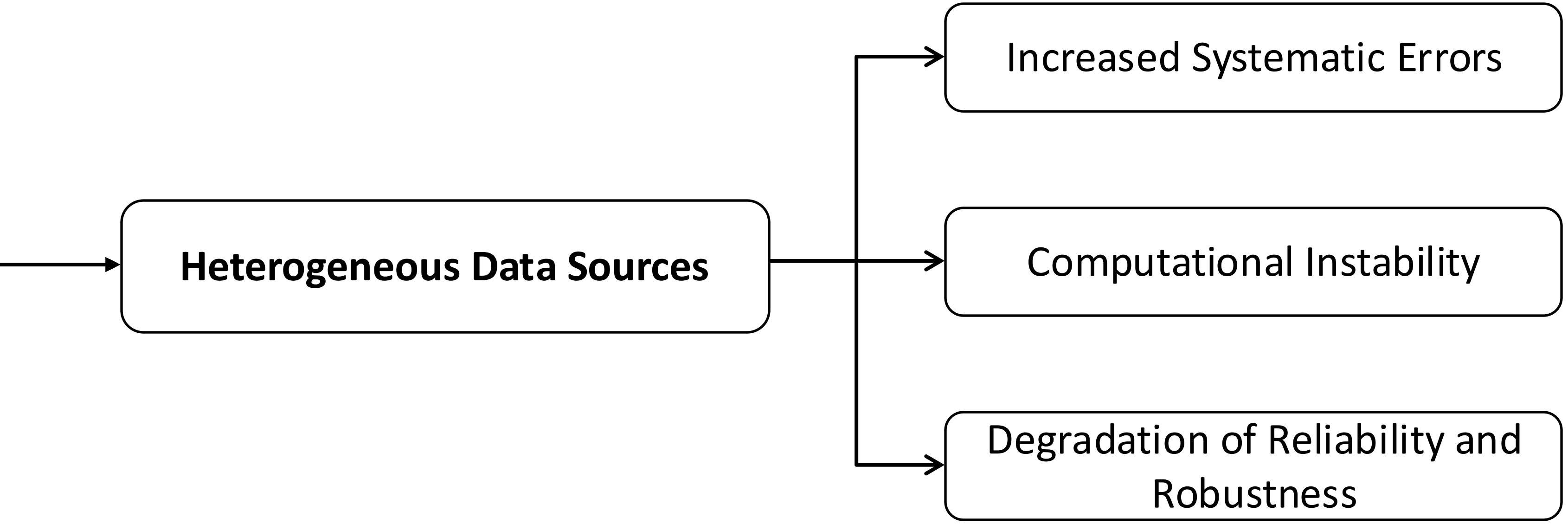
Time Synchronization

- UNIX or YYYY-MM-DDTHH:mm:ss.sssZ?
- UTC time or Local time?
- Asynchronous sampling?

Data Representation and Serialization

- File format in NMEA, CSV, or binary?
- Missing or inconsistent metadata?

Any Trouble?


Heterogeneous Data Sources
Increased Systematic Errors
Computational Instability
Degradation of Reliability and Robustness

Any Trouble?

Complexity

Flexibility

Errors

Computational cost

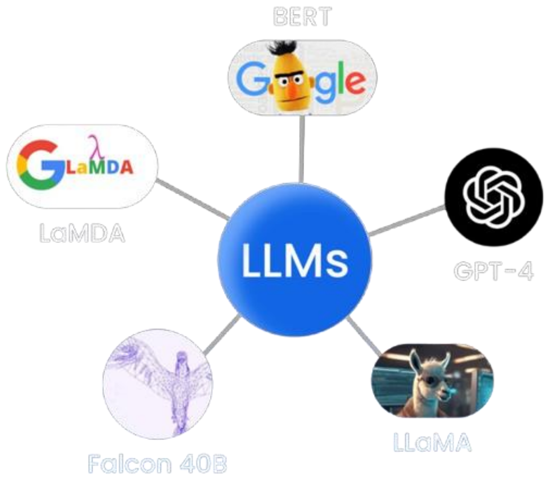
Scalability

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

LLMs look good!

But how to actually build it?

LLM as a Solution

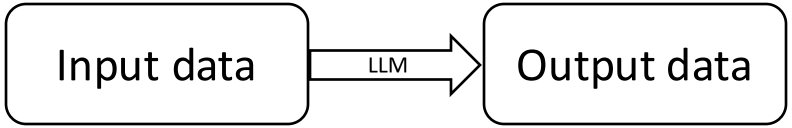
Non-agentic workflow (zero-shot)

Please transform the attached file into JSONL format, following the schema provided below:
{schema}

LLM as a Solution

Non-agentic workflow (zero-shot)

Please transform the attached file into JSONL format, following the schema provided below:
{schema}



LLM as a Solution

Non-agentic workflow (zero-shot)

Please transform the attached file into JSONL format, following the schema provided below:
{schema}

$$\begin{tikzcd}\boxed{\text{Input data}} \arrow[r, "\text{LLM}"] & \boxed{\text{Output data}}\end{tikzcd}$$

Challenges

- Hallucination
- Context windows
- Scalability issues
- No validation

LLM as a Solution

Challenges

Hallucination

Context windows

Scalability issues

No validation

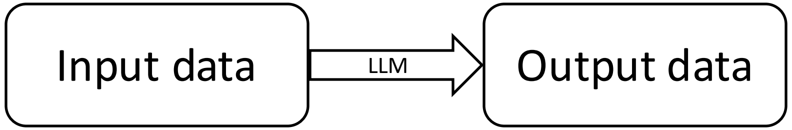
Agentic workflow

LLM as a Solution

Non-agentic workflow (zero-shot)

Agentic workflow

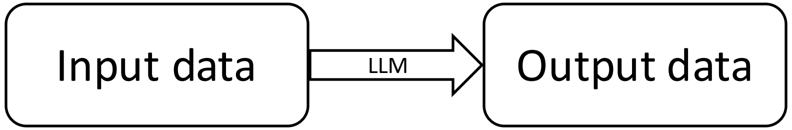
Please transform the attached file into JSONL format, following the schema provided below:
[schema]



LLM as a Solution

Non-agentic workflow (zero-shot)

Please transform the attached file into JSONL format, following the schema provided below:
[schema]



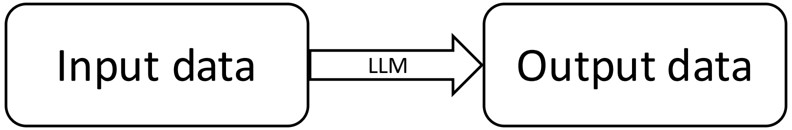
Agentic workflow

Input data type?

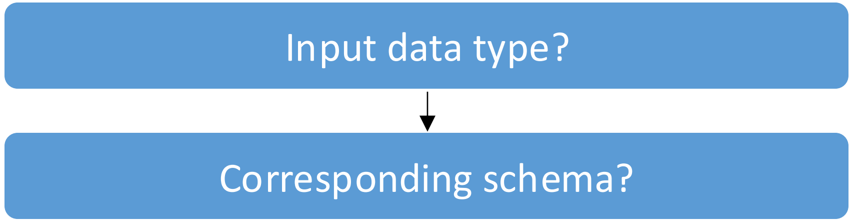
LLM as a Solution

Non-agentic workflow (zero-shot)

Please transform the attached file into JSONL format, following the schema provided below:
[schema]



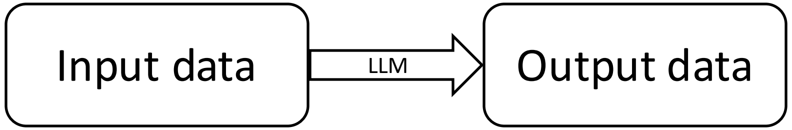
Agentic workflow



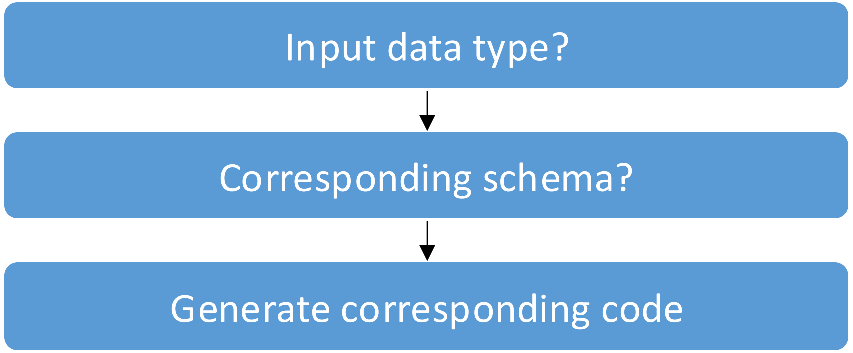
LLM as a Solution

Non-agentic workflow (zero-shot)

Please transform the attached file into JSONL format, following the schema provided below:
[schema]



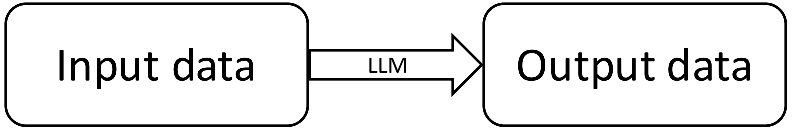
Agentic workflow



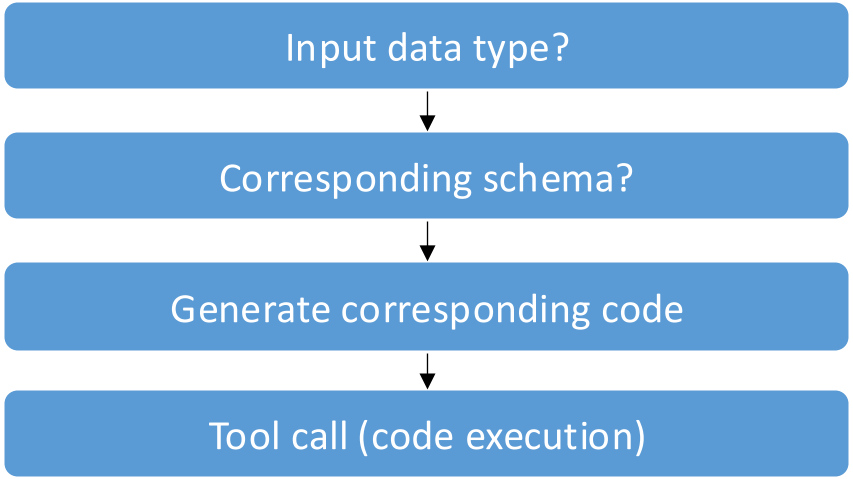
LLM as a Solution

Non-agentic workflow (zero-shot)

Please transform the attached file into JSONL format, following the schema provided below:
[schema]



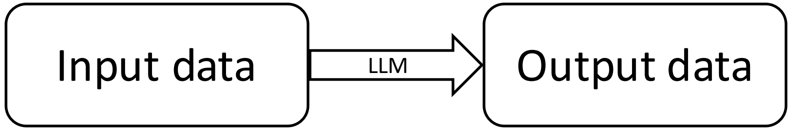
Agentic workflow



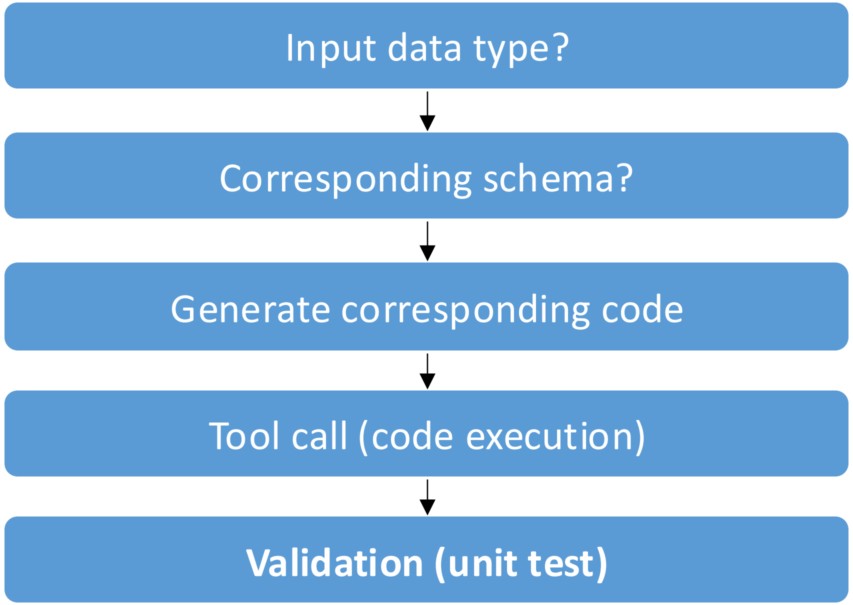
LLM as a Solution

Non-agentic workflow (zero-shot)

Please transform the attached file into JSONL format, following the schema provided below:
[schema]



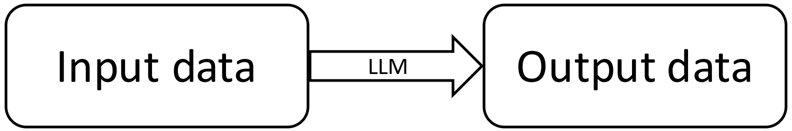
Agentic workflow



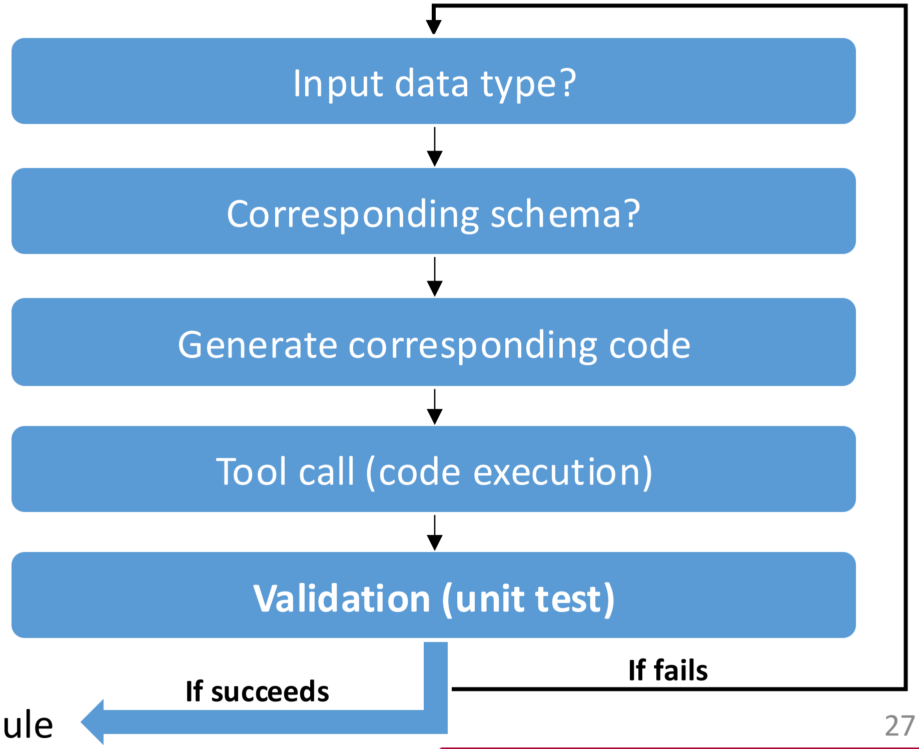
LLM as a Solution

Non-agentic workflow (zero-shot)

Please transform the attached file into JSONL format, following the schema provided below:
[schema]



Agentic workflow



Agentic Reasoning

Definition:

- AI Agents are autonomous software systems designed to perform complex tasks with minimal human intervention.
- They **decompose high-level goals into sub-tasks**, self-generate prompts, and use available **tools** (e.g., web browsing, code execution) to complete tasks.

Why They Matter:

- Shift from interactive chatbots (e.g., ChatGPT) to self-directed systems that proactively execute projects.
- Potential to revolutionize industries such as software development, business process automation, and personal assistance.

AutoGPT: Open-Source Agent Pioneer

Developed as an open-source AI agent that leverages OpenAI's APIs.

Key Features:

- Autonomous Task Planning: Automatically breaks down a given goal into sequential sub-tasks without continuous user input.
- Internet Connectivity: Uses web access and plug-ins to fetch real-time data and incorporate external information.
- Memory Management: Can manage short- and long-term memory through file and database storage to maintain context during multi-step tasks.
- Applications: From software prototyping to market research and content creation; it demonstrates a new level of task automation.

Significance:

- Sets a benchmark for integrating large language models (LLMs) with task automation.
- Inspires further development in multi-agent systems and highlights both potential and limitations (e.g., challenges with accuracy and potential infinite loops).

Devin AI: The Autonomous Software Engineer

Positioned as the "world's first AI software engineer"

Key Features:

- End-to-End Code Handling: Plans, writes, debugs, tests, and deploys code based on natural language prompts.
- Built-In Developer Tools: Equipped with its own command line, code editor, and web browser to access documentation and execute code in a sandboxed environment.
- Learning and Adaptation: Capable of learning from its errors and refining its approach over time.

Manus AI: The Chinese Contender

Positioned as the world's first fully autonomous general AI agent

Key Features:

- Fully Autonomous Operation: Completes complex, multi-step tasks from a single prompt.
- Multi-Agent Architecture: Utilizes specialized sub-agents that collaborate asynchronously in the cloud—allowing it to work continuously even after user disconnection.
- Advanced Tool Integration: Capable of interacting with web browsers, executing code, and integrating with external systems to generate actionable outputs.
- Performance and Benchmarks: Claims state-of-the-art performance on the GAIA benchmark; some reports suggest it outperforms models like OpenAI's DeepResearch.

Auto gpt

AI Agents

manus
The general AI agent

Devin

manus
The general AI agent

AI Agents

Devin

Auto gpt

Devin

AI Agents

manus
The general AI agent

Auto gpt

Devin

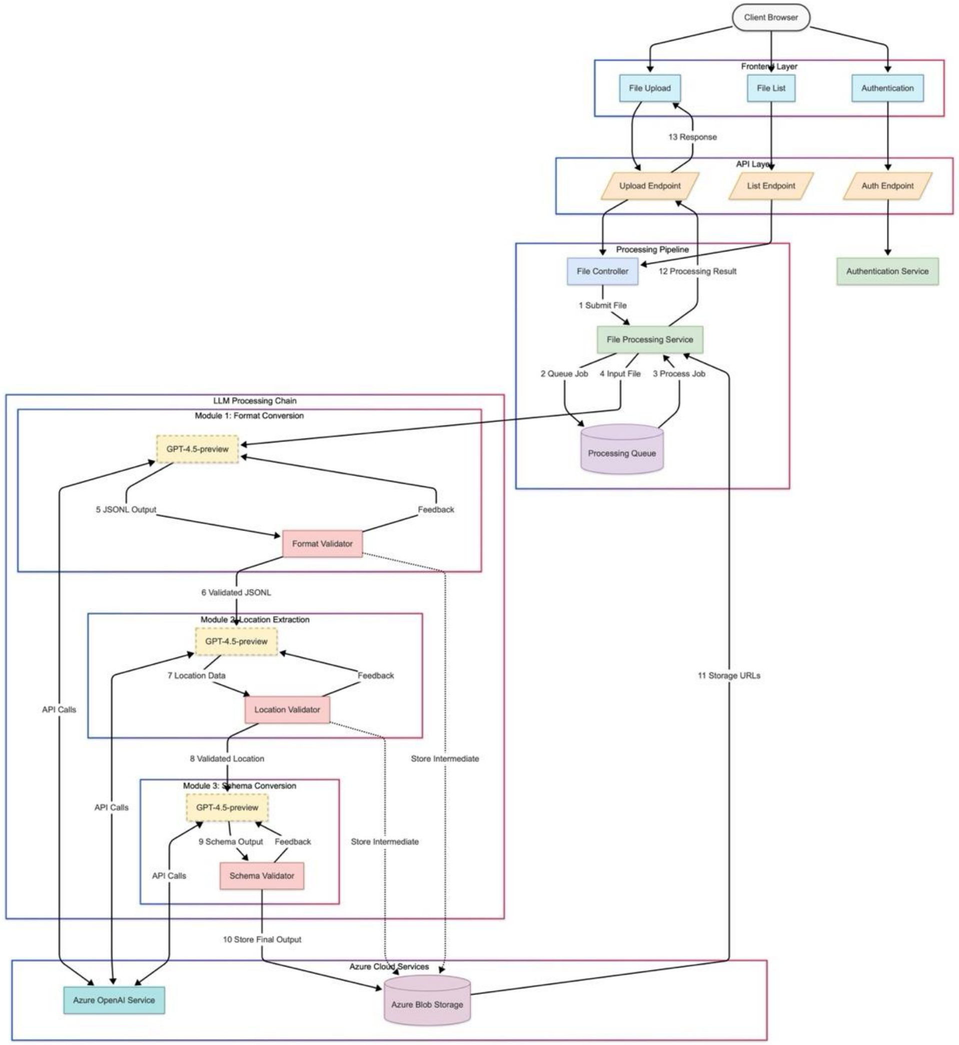
Autogpt

manus The general AI agent

inspire

FusionFly

FusionFly

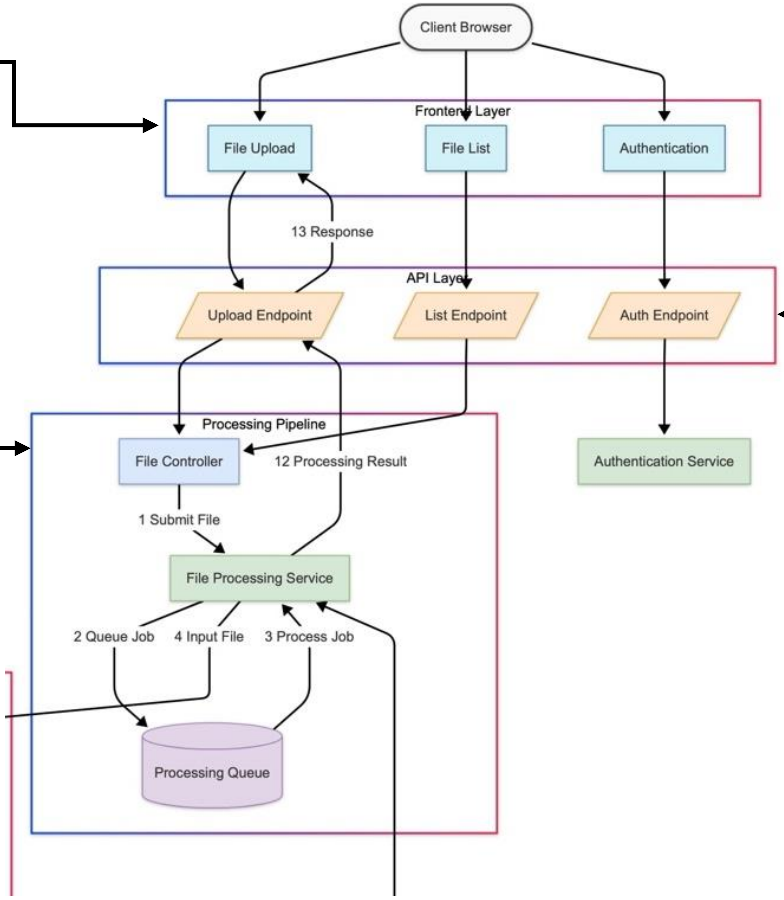


Client Browser
Frontend Layer
File Upload
File List
Authentication
13 Response
API Layer
Upload Endpoint
List Endpoint
Auth Endpoint
Processing Pipeline
File Controller
12 Processing Result
Authentication Service
1 Submit File
File Processing Service
2 Queue Job
4 Input File
3 Process Job
Processing Queue
LLM Processing Chain
Module 1: Format Conversion
GPT-4.5-preview
5 JSONL Output
Feedback
Format Validator
6 Validated JSONL
Module 2: Location Extraction
GPT-4.5-preview
7 Location Data
Feedback
Location Validator
API Calls
11 Storage URLs
8 Validated Location
Store Intermediate
Module 3: Schema Conversion
GPT-4.5-preview
API Calls
9 Schema Output
Feedback
Store Intermediate
API Calls
Schema Validator
10 Store Final Output
Azure Cloud Services
Azure OpenAI Service
Azure Blob Storage

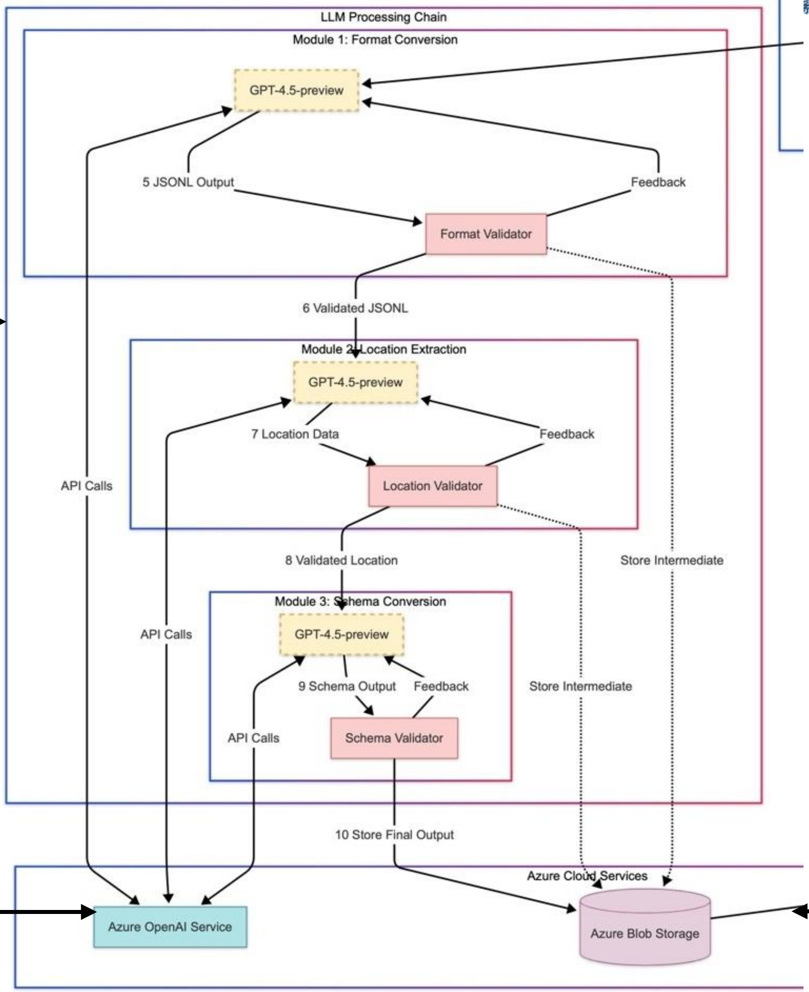
Client-side components

Processing pipeline

API Layer



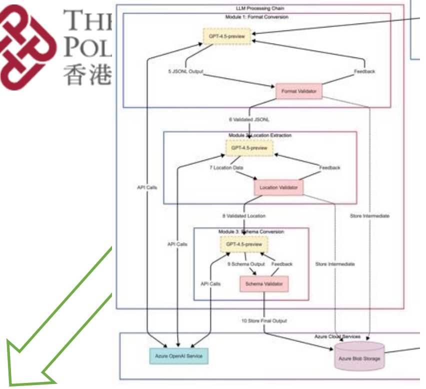
LLM process chain

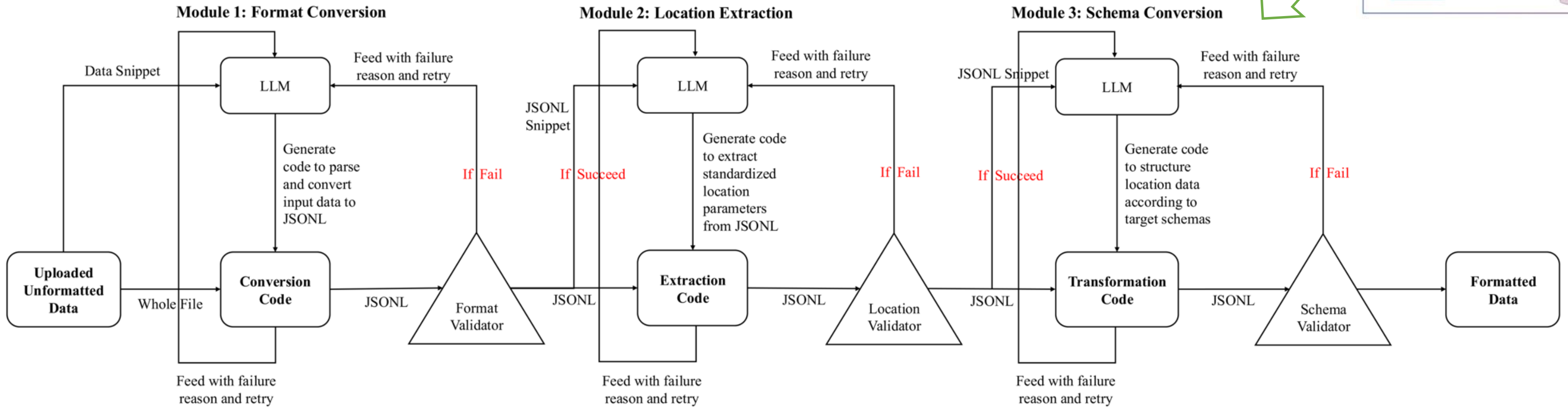


Cloud storage

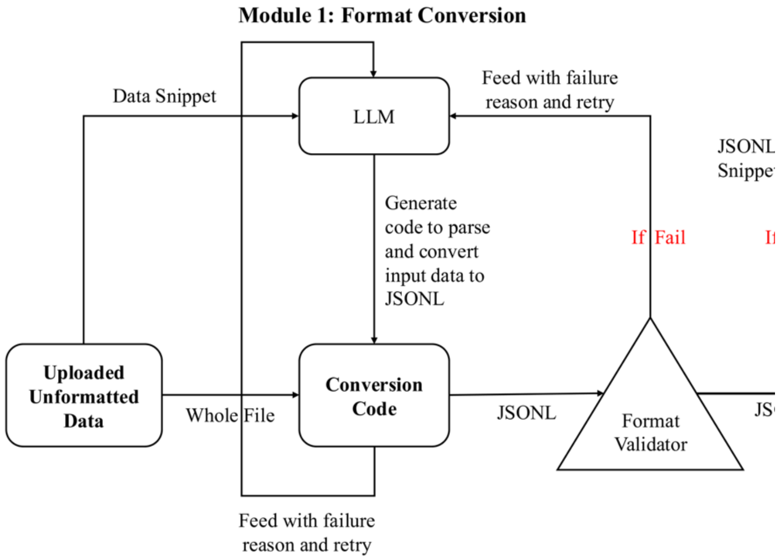
Azure OpenAI service

LLM Process Chain




Module 1: Format Conversion
Data Snippet
LLM
Feed with failure reason and retry
Generate code to parse and convert input data to JSONL
If Fail
Uploaded Unformatted Data
Whole File
Conversion Code
JSONL
Format Validator
Feed with failure reason and retry
Module 2: Location Extraction
JSONL Snippet
If Succeed
LLM
Feed with failure reason and retry
Generate code to extract standardized location parameters from JSONL
If Fail
JSONL
Extraction Code
JSONL
Location Validator
Feed with failure reason and retry
Module 3: Schema Conversion
JSONL Snippet
If Succeed
LLM
Feed with failure reason and retry
Generate code to structure location data according to target schemas
If Fail
JSONL
Transformation Code
JSONL
Schema Validator
Formatted Data
Feed with failure reason and retry

Module 1: Format Conversion


Module 1: Format Conversion
Data Snippet
LLM
Feed with failure reason and retry
JSONL Snippe
Generate code to parse and convert input data to JSONL
If Fail
If
Uploaded Unformatted Data
Whole File
Conversion Code
JSONL
Format Validator
JS
Feed with failure reason and retry

Module 1: Format Conversion

System Message:
You are a JavaScript developer tasked with writing NODE.JS CODE to convert data files.
Write a complete, executable Node.js script that will:
1. Read the input file from "[inputPath]"
2. Parse the data based on the format
3. Convert to JSONL format (one JSON object per line)
4. Write the result to "[outputPath]"

YOUR RESPONSE MUST BE *ONLY* THE JAVASCRIPT CODE WITH NO EXPLANATIONS OR DESCRIPTIONS OUTSIDE THE CODE.
NO markdown formatting or code blocks.

[For NMEA data, additional instructions:]
NMEA Parsing Rules:
1. Parse each NMEA sentence based on type (GGA, RMC, GSV, etc.)
2. Extract timestamps from data
3. Convert coordinates to decimal degrees
4. Extract all available data from each sentence

Module 1: Format Conversion

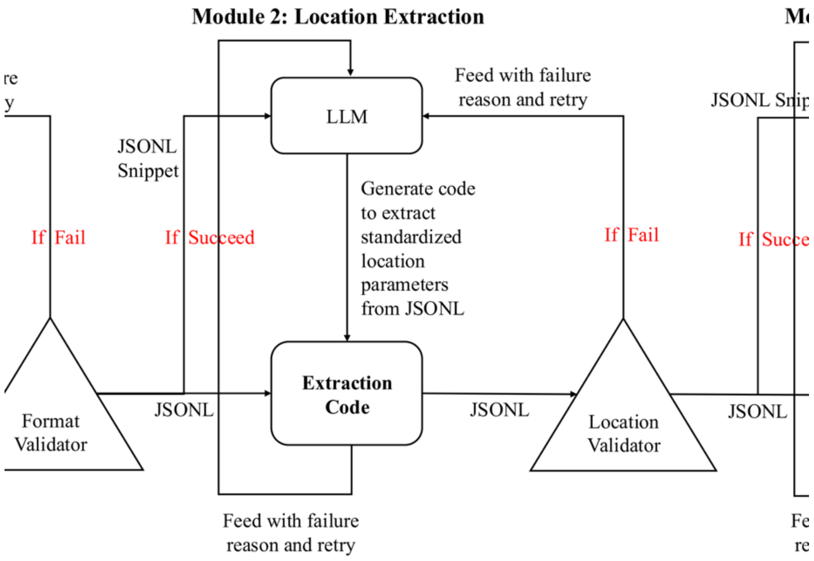
User Message:
JAVASCRIPT CODE GENERATION TASK:
Write a Node.js script to:
1. Read the file: "[inputPath]"
2. Parse each line of the [FORMAT] format data
3. Convert each line to a JSON object
4. Write the resulting JSONL to: "[outputPath]"

Requirements:
- Use Node.js fs module
- Handle the entire file processing
- Include error handling
- BE COMPLETELY SELF-CONTAINED AND EXECUTABLE

INPUT SAMPLES:
[sample data from input file]

[For NMEA, additional instructions:]
Parsing guidelines:
- Parse each valid NMEA line to a JSON object
- Extract all relevant data
- Convert coordinates to decimal degrees
- Include timestamp information when available

Module 2: Location Extraction


Module 2: Location Extraction
Mo
re
y
LLM
Feed with failure reason and retry
JSONL Snip
JSONL Snippet
Generate code to extract standardized location parameters from JSONL
If Fail
If Succeed
If Fail
If Succe
Extraction Code
Format Validator
JSONL
JSONL
Location Validator
JSONL
Feed with failure reason and retry
Fe
re

Module 2: Location Extraction

System Message:
You are a JavaScript developer tasked with writing NODE.JS CODE to extract location data.
Write a complete, executable Node.js script that will:
1. Read the ENTIRE input JSONL file from "[inputPath]"
2. Extract ONLY location-related information from each JSON object
3. Convert to JSONL format (one JSON object per line)
4. Write the result to "[outputPath]"

YOUR RESPONSE MUST BE *ONLY* THE JAVASCRIPT CODE WITH NO EXPLANATIONS OR DESCRIPTIONS OUTSIDE THE CODE.
NO markdown formatting or code blocks.

Module 2: Location Extraction

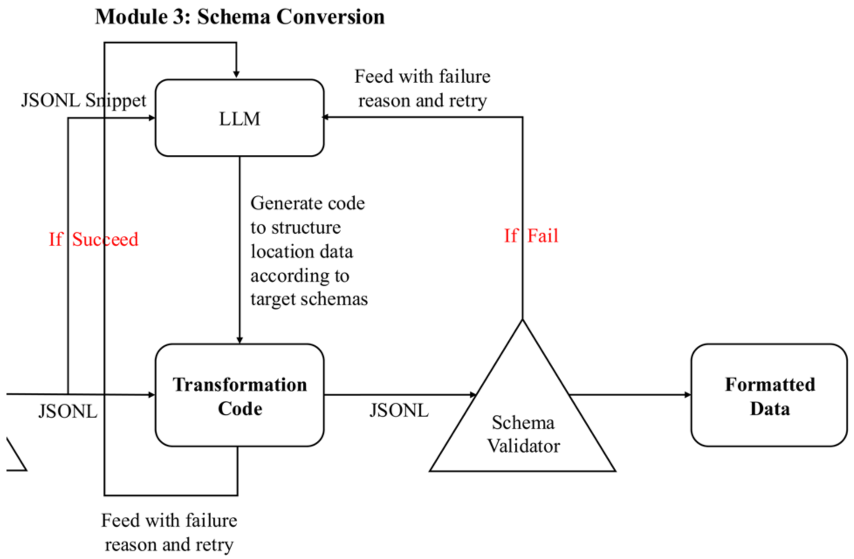
User Message:
JAVASCRIPT CODE GENERATION TASK:
Write a Node.js script to:
1. Read the ENTIRE file: "[inputPath]"
2. Extract location data from each JSON object
3. Transform each line to a new JSON object with location data
4. Write the resulting JSONL to: "[outputPath]"

Requirements:
- Use Node.js fs module
- Handle the entire file processing (potentially thousands of records)
- Include error handling
- BE COMPLETELY SELF-CONTAINED AND EXECUTABLE

CRITICAL INSTRUCTION: I am showing you only SAMPLES from the input file below. Your script must process the ENTIRE FILE, not just these samples.

INPUT SAMPLE:
[sample data from second conversion input]
REQUIRED OUTPUT FIELDS PER LINE:
- timestamp_ms: number (Unix timestamp in milliseconds)
- latitude: number (decimal degrees, between -90 and 90)
- longitude: number (decimal degrees, between -180 and 180)
- altitude: number (meters, if available, otherwise null)
- hdop: number (horizontal dilution of precision, if available, otherwise null)

Module 3: Schema Conversion


Module 3: Schema Conversion
JSONL Snippet
LLM
Feed with failure reason and retry
If Succeed
Generate code to structure location data according to target schemas
If Fail
JSONL
Transformation Code
JSONL
Schema Validator
Formatted Data
Feed with failure reason and retry

Property	GNSS Schema	IMU Schema
Top-Level Structure	Object with position data	Object with motion data
Required Fields	time_unix, position_lla	time_unix, linear_acceleration, angular_velocity, orientation
Time Field	time_unix (number)	time_unix (number)
Position Data	position_lla object with: – latitude_deg (-90 to 90) – longitude_deg (-180 to 180) – altitude_m	Not applicable
Motion Data	Not applicable	Three motion components:1. linear_acceleration (x, y, z)2. angular_velocity (x, y, z)3. orientation (w, x, y, z quaternion)
Quality Metrics	– clock_error_estimate (number) – dop (Dilution of Precision)	Not applicable
Coordinate System	Geographic (lat/lon/alt)	Cartesian (x, y, z)
Units	– Latitude/Longitude: degrees– Altitude: meters	– Acceleration: m/s^2 – Angular velocity: rad/s – Orientation: quaternion (unitless)
Data Source	GNSS/GPS receivers, RINEX, NMEA	IMU sensors, accelerometers, gyroscopes

Module 3: Schema Conversion

Submodule 1:
Schema Conversion LLM

+

Submodule 2:
Conversion Script LLM

Submodule 1: Schema Conversion LLM

System Message:
You are a data format conversion specialist.
Your task is to directly convert the input JSON data to match the target schema.
DO NOT write code or explanations.
ONLY return the converted JSON objects, one per line.
Each converted object must strictly conform to the target schema.

User Message:
Convert these input JSON objects to match the target schema:

TARGET SCHEMA:
```json
[GNSS or IMU schema]
```

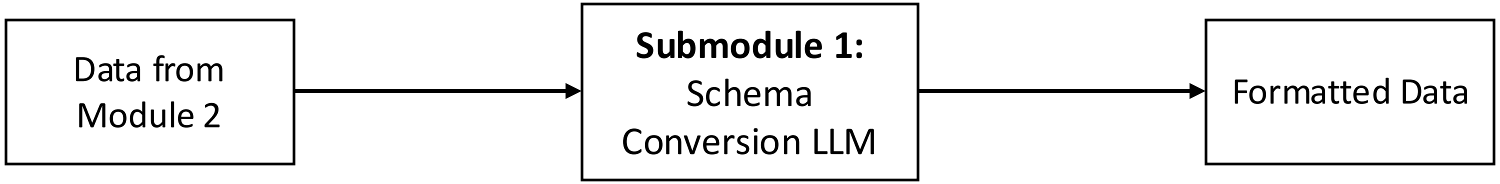
INPUT JSON (one object per line):

[sample data from Module 2]

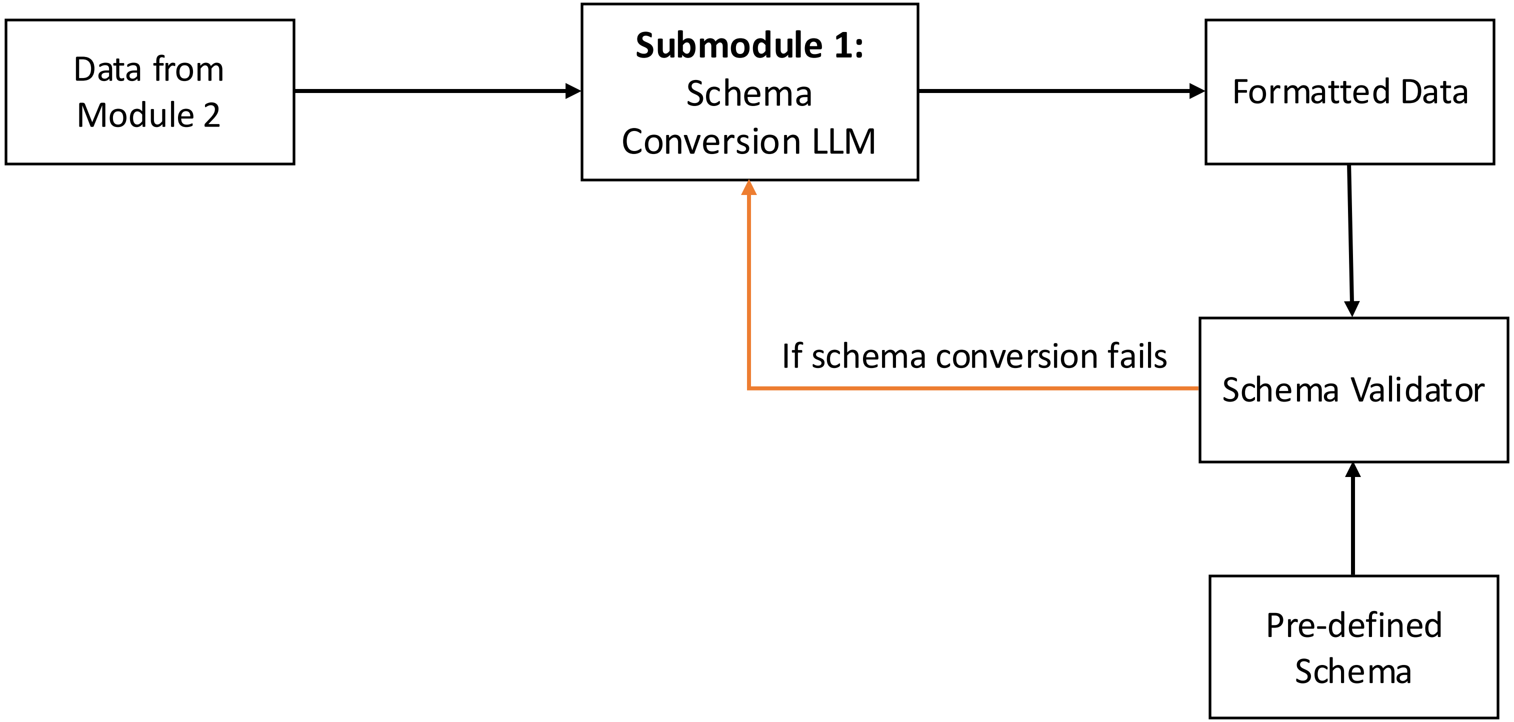
FIELD MAPPING:
[GNSS or IMU specific field mapping]

RESPOND ONLY with the converted JSON objects, one per line. No explanations or code.

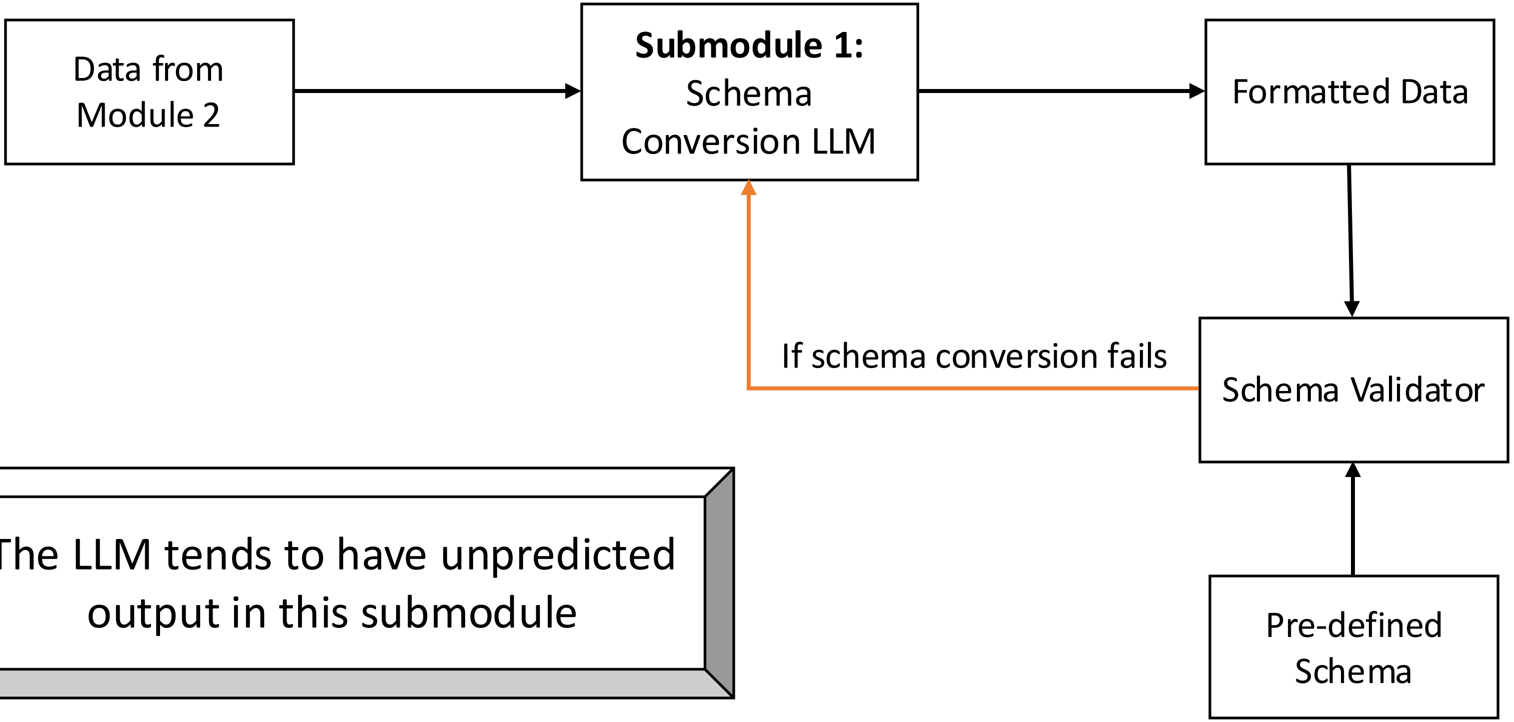
Submodule 1: Schema Conversion LLM



Submodule 1: Schema Conversion LLM



Submodule 1: Schema Conversion LLM



The LLM tends to have unpredicted output in this submodule

Department of Aeronautical and Aviation Engineering 航空及民航工程學系

THE HONG KONG POLYTECHNIC UNIVERSITY 香港理工大學

Submodule 1: Supervised Fine-tuning (SFT)

Fine-Tuning & Training

- Dataset: 800 training + 200 validation pairs
- Structure: JSONL.

Training Loss

- The training loss curve shows a consistent decrease, indicating effective learning.

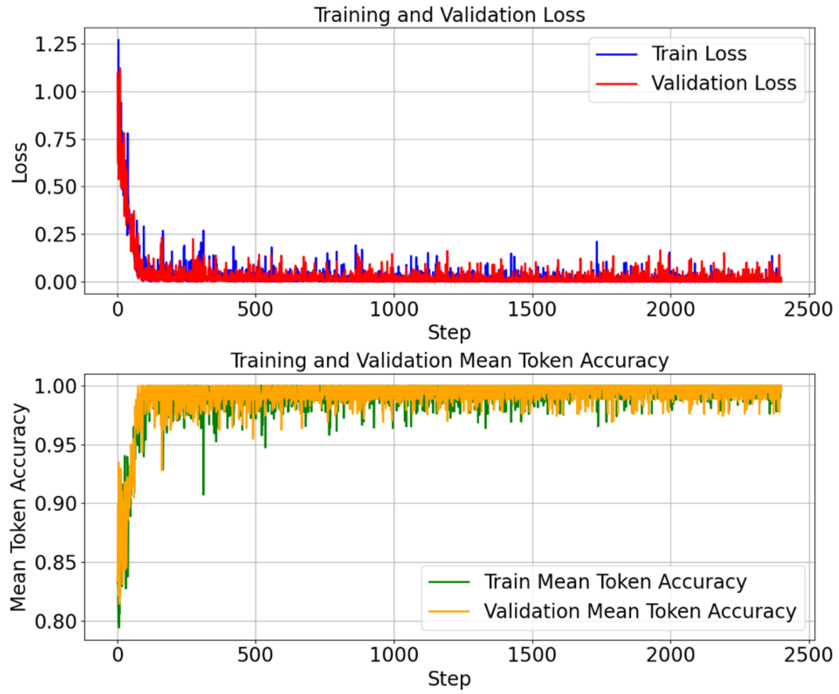
Validation Loss

- The validation loss curve stabilizes after an initial decrease around 200 steps.

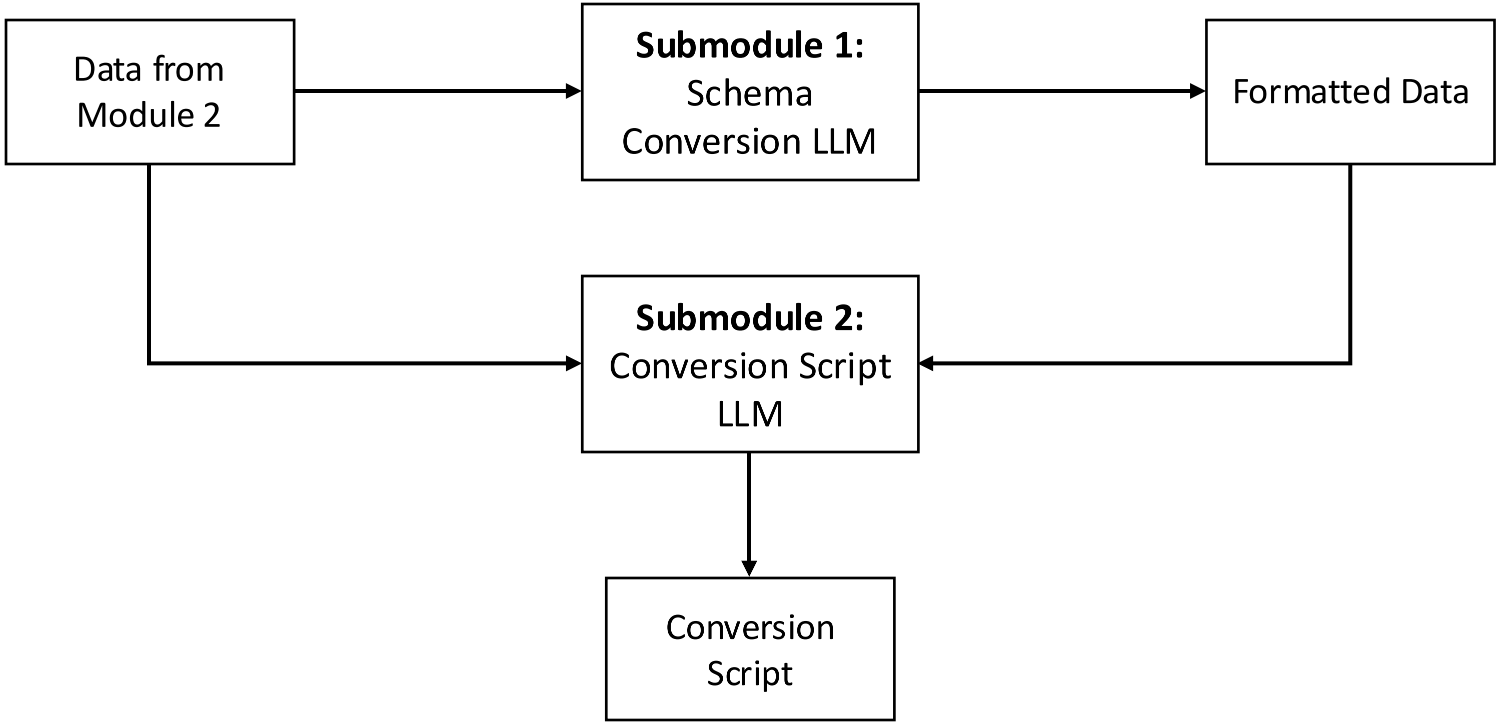
Mean Token Accuracy

- Training accuracy rises rapidly and stabilizes at a high level, while validation accuracy remains consistently above 98%, confirming effective generalization.

As expected, lower validation loss corresponds to higher validation accuracy.



Submodule 2: Conversion Script LLM


Data from Module 2
Submodule 1: Schema Conversion LLM
Formatted Data
Submodule 2: Conversion Script LLM
Conversion Script

Submodule 2: Conversion Script LLM

System Message:
You are a JavaScript developer tasked with writing a NODE.JS TRANSFORMATION SCRIPT to convert data according to a specific schema.
Write a complete, executable Node.js script that will:
1. Read the ENTIRE input JSONL file from "[inputPath]"
2. Transform each JSON object to match the required schema structure
3. Convert to JSONL format (one JSON object per line)
4. Write the result to "[outputPath]"

YOUR RESPONSE MUST BE *ONLY* THE JAVASCRIPT CODE WITH NO EXPLANATIONS OR DESCRIPTIONS OUTSIDE THE CODE.
NO markdown formatting or code blocks.

You have been provided with both raw input data and an example of how it should be transformed.
Analyze both to understand the transformation pattern, then create a script to apply this pattern to all records.

Submodule 2: Conversion Script LLM

User Message:
JAVASCRIPT CODE GENERATION TASK:
Write a Node.js transformation script to:
1. Read the ENTIRE file: "[inputPath]"
2. Transform each JSON object to match the target schema
3. Write the resulting JSONL to: "[outputPath]"

Requirements:
- Use Node.js fs module
- Handle the entire file processing (potentially thousands of records)
- Include error handling
- BE COMPLETELY SELF-CONTAINED AND EXECUTABLE

TARGET SCHEMA:
```json
[GNSS or IMU schema]
```

CRITICAL INSTRUCTION: I am showing you sample data below. Your script must process the ENTIRE FILE, not just these samples.

INPUT SAMPLE (raw data):

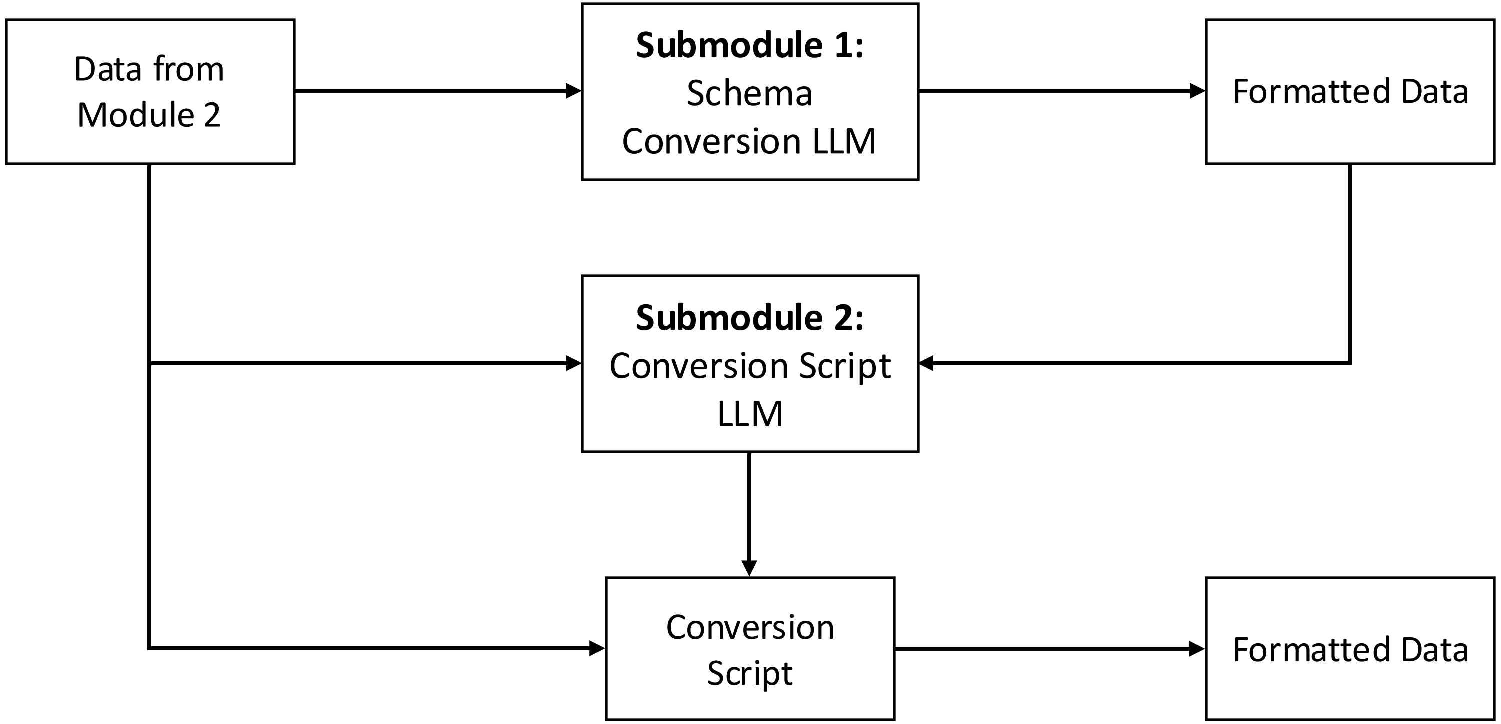
[sample data from Module 2]

CONVERTED SAMPLE (how the data should look after transformation):

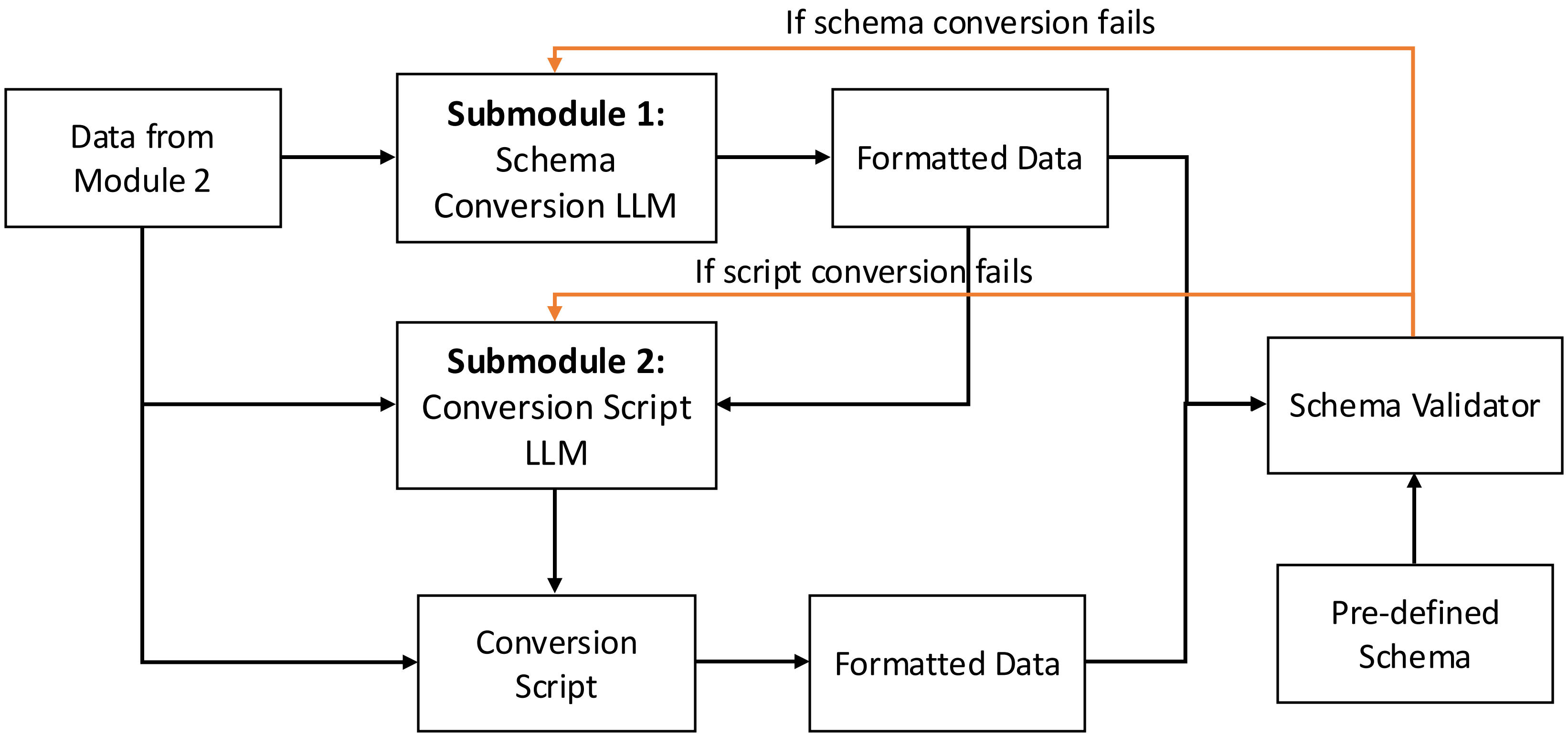
[output from Submodule 1]

FIELD MAPPING:
[GNSS or IMU specific field mapping]

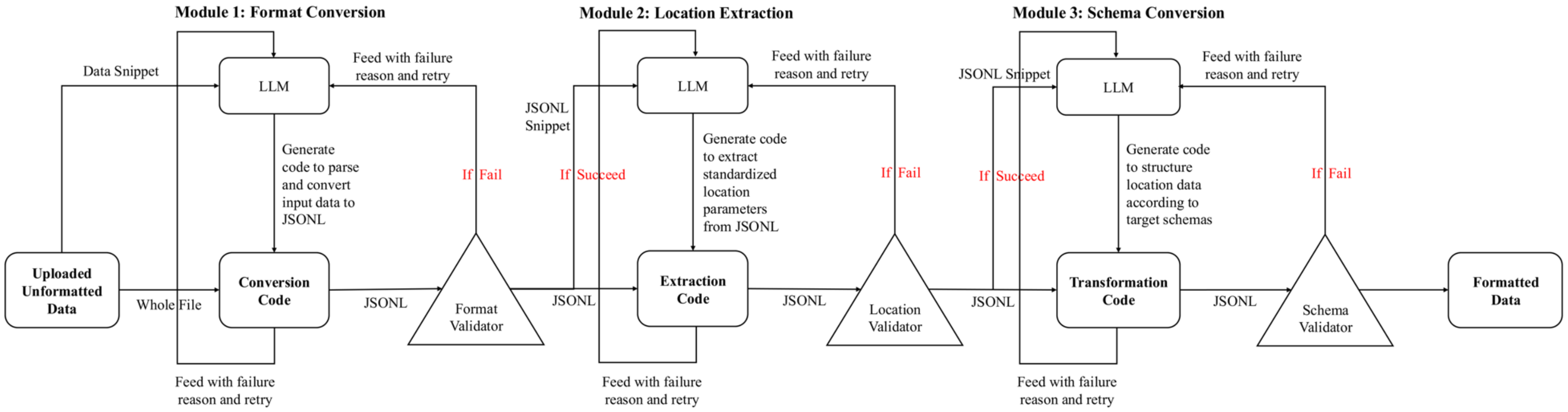
Submodule 2: Conversion Script LLM


Data from Module 2
Submodule 1: Schema Conversion LLM
Formatted Data
Submodule 2: Conversion Script LLM
Conversion Script
Formatted Data

Module 3: Schema Conversion


If schema conversion fails
Data from Module 2
Submodule 1: Schema Conversion LLM
Formatted Data
If script conversion fails
Submodule 2: Conversion Script LLM
Schema Validator
Conversion Script
Formatted Data
Pre-defined Schema

LLM Process Chain


Module 1: Format Conversion
Module 2: Location Extraction
Module 3: Schema Conversion
Data Snippet
LLM
Feed with failure reason and retry
Generate code to parse and convert input data to JSONL
If Fail
Uploaded Unformatted Data
Whole File
Conversion Code
JSONL
Format Validator
Feed with failure reason and retry
JSONL Snippet
If Succeed
LLM
Feed with failure reason and retry
Generate code to extract standardized location parameters from JSONL
If Fail
JSONL
Extraction Code
JSONL
Location Validator
Feed with failure reason and retry
JSONL Snippet
If Succeed
LLM
Feed with failure reason and retry
Generate code to structure location data according to target schemas
If Fail
JSONL
Transformation Code
JSONL
Schema Validator
Formatted Data
Feed with failure reason and retry

Backend Architecture

Language & Framework:

- Node.js with Express and TypeScript

Structure:

- Follows MVC pattern with dedicated routes, controllers, and services

API Routes:

- Authentication (/api/auth/*) -
  User registration, login, profile management
- File Processing (/api/files/*) - File upload, processing, listing, and download

Error Handling:

- Comprehensive error handling with fallback mechanisms

Deployment:

- Azure-ready with configuration for production environments

File Processing Pipeline:

- Multer for file uploads with custom storage and validation
- Bull queue system for asynchronous processing
- Multi-step conversion process

Frontend Stack

Framework: React with React Router for navigation

State Management: Local state and React hooks

Authentication: Token-based auth with localStorage

API Communication: Axios with interceptors for auth token handling

Components:

- Protected routes requiring authentication
- File upload with multi-file handling
- File listing and downloading
- User authentication (login/register) forms

UI: Modern styling with Tailwind CSS

Database Integration

Primary Database: Azure Cosmos DB

- Container: users with email as partition key
- Database: fusionfly-db

Local Development Fallback: File-based JSON storage system

- Automatic directory creation for local storage
- Structured fallback mechanism for development without Azure

Storage: Azure Blob Storage

- Organized containers for uploads, processed data, and results

Local Fallback: Directory-based file storage

- Automatic creation of needed directories

Demo Video

https://youtu.be/oGyyChs1mSo

Benchmark

Comprehensive Test Cases

- The benchmark directory includes various test cases for normal scenarios (Medium Urban Environment with NMEA/OBS) and edge cases (missing data, corrupted data, format variations).

Scientific Evaluation Approach

- Rigorous scientific framework for evaluating transformation quality rather than just positioning performance.

```
benchmark_dataset/
├── raw/              # Raw, unformatted navigation data
│   ├── gnss/
│   │   ├── nmea/     # NMEA format GNSS data
│   │   └── obs/      # RINEX observation format GNSS data
│   └── imu/          # Raw IMU data in CSV format
├── standardized/     # Standardized data following the schema
│   ├── gnss_data/    # Standardized GNSS data
│   └── imu_data/     # Standardized IMU data
├── test_cases/       # Test cases for benchmarking
│   ├── normal/       # Normal scenarios
│   │   ├── case1/    # Medium Urban Environment (GNSS NMEA + IMU)
│   │   ├── case2/    # Medium Urban Environment (GNSS OBS + IMU)
│   │   └── case3/    # Tunnel Environment (IMU only)
│   └── edge_cases/   # Edge cases for robustness testing
│       ├── missing_data/   # Data with missing fields
│       ├── corrupted_data/ # Data with corrupted values
│       └── format_variations/# Variations in data format
├── metadata/         # Dataset metadata
│   ├── dataset_info.json          # General information about the dataset
│   ├── sensor_specifications.json # Specifications of sensors used
│   └── schema_documentation.json  # Documentation of the standardized schema
└── evaluation/       # Evaluation tools
    ├── metrics.py    # Script for evaluating conversion accuracy
    ├── benchmark.py  # Script for benchmarking conversion speed
    └── results_template/ # Templates for evaluation results
```

https://github.com/Thorkee/FusionFly/tree/main/benchmark

Dataset – Normal Scenario

Medium Urban Environment with NMEA (case1)

- Complete GNSS (NMEA format) and IMU data
- Typical urban canyon with high-rising buildings
- Real data from UrbanNav Dataset

Medium Urban Environment with OBS (case2)

- Complete GNSS (RINEX OBS format) and IMU data
- Same environment as case1 but with different GNSS format
- Real data from UrbanNav Dataset

Tunnel Environment (case3)

- IMU data only (no GNSS)
- Simulates tunnel environment with no GNSS reception
- Uses real IMU data from UrbanNav Dataset

Dataset – Edge Cases

Missing Data

- GNSS data with missing position
- GNSS data with missing time
- GNSS data with missing DOP
- IMU data with missing linear acceleration
- IMU data with missing angular velocity
- IMU data with missing orientation

Corrupted Data

- GNSS data with invalid coordinates
- GNSS data with extreme DOP values
- GNSS data with inconsistent timestamps
- IMU data with outliers in acceleration
- IMU data with NaN values
- IMU data with inconsistent timestamps

Format Variation

- NMEA data with different sentence types
- RINEX observation data with varying satellite counts
- IMU data with different field ordering
- IMU data with extra fields
- IMU data with different units
- GNSS data with different position formats (ECEF vs. LLA)

Evaluation Metrics

Data Field Accuracy Metrics

Robustness Metrics

Data Field Accuracy Metrics

Core Principle:

- Point-by-point comparison between ground truth and transformed data

Matching Algorithm:

- Pairs data points by closest timestamps (within 0.1s threshold)
- Uses dot notation to access nested fields (e.g., position_lla.latitude_deg)
- Evaluates numerical differences across all relevant navigation parameters

Implementation

```
# Field comparison logic
for gt_entry in ground_truth_data:
   # Find closest timestamp in converted data
   closest_entry =
find_closest_by_timestamp(converted_data, gt_entry)

   # If match found within threshold
   if closest_entry and time_difference < 0.1:
      # Calculate absolute error for each field
      error = abs(gt_value - conv_value)
      errors.append(error)
```

Data Field Accuracy Metrics – Scoring

Mean Absolute Error (MAE):

- mae = np.mean(errors) - Average absolute difference
- Simple, interpretable measure of error magnitude

Root Mean Square Error (RMSE):

- rmse = sqrt(mean(errors^2)) - Penalizes larger errors
- Key metric for assessing overall accuracy

Normalized RMSE (NRMSE):

- nrmse = rmse / (max_original - min_original)
- Makes errors comparable across different measurement types

Department of Aeronautical and Aviation Engineering
航空及民航工程學系

THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

Robustness Metrics

Core Principle:

- Systematic testing with intentionally degraded data

Quantifies resilience to real-world data issues including:

- Missing data points
- Corrupted values (extreme/invalid readings)
- Inconsistent sampling rates
- Special values (NaN, infinite)

Test Case Design:

- Controlled corruption of specific data points
- Various degrees of data degradation (5%, 10%, 20%, 30%)
- Multiple error types tested independently

Implementation

```
{
"time_unix": 1621218775.5489783,
"linear_acceleration": {
"x": 100.0, // Extreme value (corrupted)
"y": -100.0, // Extreme value (corrupted)
"z": 100.0 // Extreme value (corrupted)
},
"angular_velocity": {
"x": -0.0562, "y": 0.0114, "z": -0.0098
}
}
```

Data Field Accuracy Metrics – Scoring

Success Rate:

- success_rate = successful_conversions / total_test_cases
- Measures basic ability to process problematic inputs

Error Recovery Rate:

- recovery_rate = properly_handled / total_corrupted_points
- Assesses how well the system identifies and handles anomalies

Accuracy Under Stress:

- Combines field accuracy metrics on edge case data
- Measures degradation compared to normal condition

Discussions

Make sure the three AI-powered modules-Format Conversion, Location Extraction, and Schema Conversion work smoothly together

Small mistakes or misunderstandings at one stage could affect the next stage

Discussions

Make sure the three AI-powered modules-Format Conversion, Location Extraction, and Schema Conversion work smoothly together

We had to spend time carefully designing how these modules connect and communicate

Small mistakes or misunderstandings at one stage could affect the next stage

Discussions

Evaluate the accuracy and improvements clearly and effectively

AI-driven systems often produce slightly different results each time

Although we created a detailed benchmark test based on the UrbanNav dataset, we did not have enough time to fully test it

Discussions

Evaluate the accuracy and improvements clearly and effectively

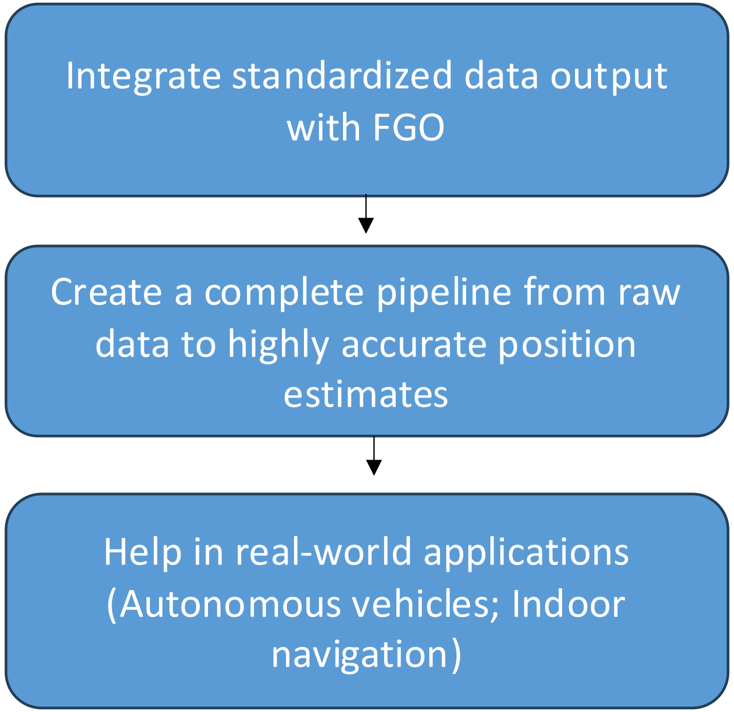
AI-driven systems often produce slightly different results each time

Although we created a detailed benchmark test based on the UrbanNav dataset, we did not have enough time to fully test it

More testing are needed

Can we introduce a supervising agent to make the pipeline more robust?

Future Work



Integrate standardized data output with FGO

Create a complete pipeline from raw data to highly accurate position estimates

Help in real-world applications (Autonomous vehicles; Indoor navigation)

Future Work

Expanding FusionFly to support more types of sensors and data formats

Positioning systems often use additional sensors (Cameras; UWB)

Allow FusionFly to be used more widely in various applications (Smart city solutions)

Thank you for your attention!

Questions, Comments and Collaboration are welcome.

Ju Lin / Lingyao Zhu

ju.lin@connect.polyu.hk / ling-yao.zhu@connect.polyu.hk

Department of Aeronautical and Aviation Engineering,

The Hong Kong Polytechnic University

GitHub

Wiki