A High-Level Data Decentralized Processing Platform for AIoT Applications

Kit-Lun Tong

Registration Number: 100328717

SUPERVISED BY

Primary: Edwin Ren

Secondary: Hane Aung

Department of Computing Sciences

University of East Anglia

July/2025

This dissertation is submitted for the degree of

Doctor of Philosophy

Declaration

I certify that the work contained in the thesis submitted by me for the degree of PhD is my original work except where due reference is made to other authors, and has not been previously submitted by me for a degree at this or any other university.

The following research papers, related to this work, were developed as part of the following projects:

Royal Society International Exchanges 2021 – An Intelligent Data Processing Platform for Smart Manufacturing – An AIoT Platform (2022-2024):

- Y.-C. Liang, K.-R. Wu, **K.-L. Tong**, Y. Ren, and Y.-C. Tseng (2023) “An Exchange-based AIoT Platform for Fast AI Application Development,” Q2SWinet ’23.
- (Chapter 3) **K.-L. Tong**, H.-C. Lin, K.-R. Wu, Y. Ren, G. Parr ,and Y.-C. Tseng (2025) "DAIoTtalk: A Data-Decentralized Pub-Sub AIoT Platform", VTC2025-Spring.

Jack Industrial Sewing Machine Company - Smart Sewing Machine Manufacturing Development (2022-2024):

- (Chapter 4) **K. L. Tong**, and Y. Ren (2024) "A Product Completion Estimation System with Unsupervised Learning for Smart Sewing Machines," internal paper for Jack Industrial Sewing Machine Company.

CHC Tech Limited Norwich - GNSS Error Source Recognition (2024-2025):

- (Chapter 5) **K.-L. Tong**, Y. Ren, X. Shi, Z. Chen, and X. Zhang (2025) "A Novel AI Temporal-Spatial Analysis Approach for GNSS Localization Propagation Error Source Recognition", accepted by VTC2025-Fall.

Moreover, the following research papers were also published during the PhD period (2021-2025):

- **K.-L. Tong**, K.-R. Wu, and Y.-C. Tseng (2021) "The Device–Object Pairing Problem: Matching IoT Devices with Video Objects in a Multi-Camera Environment," Sensors '21.
- R. Xiong, **K. L. Tong**, Y. Ren, W. Ren, and G. Parr (2023) "From 5G to 6G: It is time to sniff the communications between a base station and core networks," ACM MobiCom '23.

In accordance with the University's Generative AI Policy for Research and Innovation, this work's content is refined and proofread by artificial intelligence (AI) technologies, including ChatGPT 3.5 and Grammarly, piratically in synonyms query, grammar correction, and sentence reconstruction. No content generated from AI technologies without a reliable reference has been presented in the work.

Acknowledgements

I express my gratitude to the University of East Anglia for providing the studentship that supported my PhD journey.

I sincerely thank Dr. Edwin Ren, my primary supervisor, for the invaluable advice and mentorship he has provided me during my research.

I acknowledge Dr. Hane Aung, my secondary supervisor, for his insights and supervision during my PhD.

I am grateful to Prof. Yu-Chee Tseng at National Yang Ming Chiao Tung University for serving as a reference during my PhD application and for his unwavering support during my research.

I extend my appreciation to Prof. Hung-Lin Fu at National Yang Ming Chiao Tung University, who not only provided a reference for my PhD application but also be my first archery coach.

I also thank Prof. Lan-Da Van at National Yang Ming Chiao Tung University for supporting me as a reference during my PhD application process.

Finally, I deeply appreciate my family's financial support, which covered my living expenses and enabled me to focus on my research during this period.

Abstract

Artificial Intelligence of Things (AIoT), the fusion of Internet of Things (IoT) and Artificial Intelligence (AI), is changing manufacturing and navigation alongside many other industries. However, complexities in device scaling, data management, and lack of skilled personnel hinder the wide adoption of AIoT. A high-level IoT platform integrates communication protocols, databases, and application program interfaces (APIs). These centralize the management of an IoT solution to make it simpler to develop and deploy applications while also addressing device communication, data processing, and security factors.

On the other hand, traditional IoT platforms are often cloud-based or data-centralized, suffering inefficiencies in routing and process scaling limitations. To address these problems, we aim to develop DAIoTtalk, an AIoT platform with a data-decentralized architecture that builds upon IoTtalk. Additionally, DAIoTtalk supports flexible networking and low-code configuration by enabling gRPC-based Pub-Sub communications.

To showcase its applicability, we created case studies defined by different industries: SewingTalk and GNSS-EStalk. SewingTalk improves the productivity of textile manufacturing by analyzing logs of smart sewing machines using unsupervised learning to estimate daily completion. Meanwhile, GNSS-EStalk identifies sources of GNSS errors using an AI-driven temporal-spatial approach.

Through a series of experiments and case studies, we demonstrate the effectiveness of DAIoTtalk across multiple domains, addressing challenges related to communication efficiency, deployment versatility, and resource scalability.

Contents

List of Figures

List of Tables

Acronyms

ACK Acknowledgment

AI Artificial Intelligence

AIoT Artificial Intelligence of Things

AMQP Advanced Message Queuing Protocol

API Application Programming Interface

CB-IoT Cloud-Based Internet of Things

CFEC Categorical Focal Cross-Entropy

CNN Convolutional Neural Network

CoAP Constrained Application Protocol

DF Device Features

DFO Device Feature Object

DM Device Model

FPS Frames per Second

GNSS Global Navigation Satellite System

GPS Global Positioning System

GUI Graphical User Interface

HTTP Hypertext Transfer Protocol

IDF Input Device Feature

IDL Interface Definition Language

IMU Inertial Measurement Unit

IoT Internet of Things

IPC Inter-Process Communication

JF Join Function

JSON JavaScript Object Notation

KLD Kullback–Leibler Divergence

LSTM Long Short-Term Memory

MAE Mean Absolute Error

MITM Man-in-the-Middle

MLP Multi-Layer Perceptron

MQTT Message Queuing Telemetry Transport

MS-IoT Microservice Internet of Things

MSE Mean Squared Error

NB-IoT Narrowband Internet of Things

ODF Output device Feature

P2P Peer-to-Peer

PNT Positioning, Navigation, and Timing

Protobuf Protocol Buffers

Pub-Sub Publish-Subscribe

QoS Quality of Service

RDM Referral Distance Matrix

ReLU Rectified Linear Unit

Req-Resp Request-Response

RFI Radio Frequency Interference

RNN Recurrent Neural Network

RPC Remote Procedure Call

S/N Signal-to-Noise Ratio

SIP Session Initiation Protocol

SMEs Small and Medium-Sized Enterprises

TCP Transmission Control Protocol

TLS Transport Layer Security

VPN Virtual Private Network

WSNs Wireless Sensor Networks

ZFilter Z-Score Normalization Filtering

1

Introduction

1.1 The Evolution of Smart Technologies: From Smart Planet to AIoT

In 2008, the CEO of IBM, Sam Palmisano, proposed the Smart Planet idea [1] with three main aspects: Instrumented, Interconnected, and Intelligent. Instrumentation involves gathering real-time information from various sources operating differently, including sensors, personal devices, and appliances. Interconnected is the use of digital networking platforms like the Internet of Things (IoT) for seamless data exchange from service nodes worldwide. Intelligence involves the use of advanced technologies, including algorithms, modeling, cloud computing, data visualization, and artificial intelligence (AI), to improve decision-making, optimize services, etc.

Three years later, the German government introduced Industry 4.0 [2]. This revolution, ushered in by cyber-physical systems, allowed manufacturers to run "smart" factories by embedding advanced technologies within their equipment, creating further automationwhile providing more flexibility to respond to market needs.

Meanwhile, AI has advanced significantly over the past decade, enabling a wide range of highly efficient automated tasks and services. For example, the MNIST database [3], which serves as a benchmark dataset for handwritten digit

recognition, has been resolved to a level of accuracy that is astonishing for AI models, with test error rates less than 0.18%, compared to the 0.2% human error rate [4][5]. Likewise, the large-scale image classification ImageNet database [6] experienced an increase in accuracy from 50% in 2011 to more than 90% over a decade [7]. In this decade, artificial general intelligence (AGI) powered by generative AI is reshaping the landscape across various domains, including education, manufacturing, finance, social systems, healthcare, and service industries [8].

Today, the integration of AI and IoT, known as Artificial Intelligence of Things (AIoT), has become an essential part of modern life. AIoT applications are found in a wide variety of sectors, including smart factories, precision agriculture, intelligent buildings, and smart healthcare, helping to change industries and create innovation in ways we never thought possible.

1.2 Challenges in AIoT: Scaling Devices and Accessibility

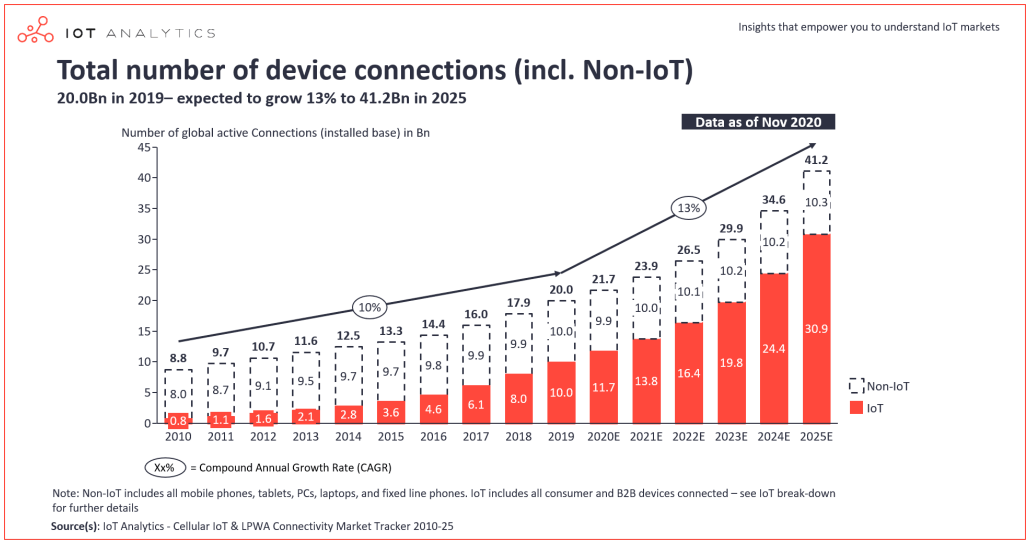


Figure 1.2.1: Number of devices connections in 2025 (from [9])

Despite its rapid advancements, AIoT development faces significant challenges. Fig. 1.2.1 illustrates the exponential growth of global device connections, as

reported by [9]. The number of IoT devices is projected to nearly triple, reaching 30.9 billion between 2020 and 2025. Simultaneously, [10] predicts that data generated by IoT devices will surge from 18.3 ZB in 2019 to 73.1 ZB by 2025. This explosive growth presents critical challenges in managing vast device networks, as well as transmitting, storing, securing, and processing massive data volumes efficiently.

Furthermore, the widespread adoption of AIoT applications is hindered by the specialized expertise required in both AI and IoT. Small and medium-sized enterprises (SMEs) and individual users often lack the necessary technical knowledge, making AIoT system development and deployment costly and resource-intensive. Additionally, many existing AIoT solutions are highly encapsulated, restricting user flexibility. As a result, modifying and redeploying these systems in new environments remains challenging, further limiting their accessibility and scalability.

1.3 High-level AIoT Platform

A high-level AIoT platform offers an integrated environment that supports scalable devices and cyber applications, simplifying the development and deployment of AIoT solutions. Key aspects include management, development, deployment, communication, data processing, and security.

- **Management** refers to seamless onboarding, monitoring, and control of AIoT nodes and requires frictionless registration and integration onto the platform. It allows for swift device provisioning, real-time visibility, and remote administration, streamlining the process for users to monitor node performance and status, at scale, with minimaleffort.
- **Development** enables the generation of cloud, edge and hybrid applications to be easily integrated and developed toward flexible, scalable and efficient

AIoT solutions in the platform. Its support for a wide range of use cases enables developers to deliver their goals,bringing an efficient performance.

- **Deployment** facilitates fast and straightforward reconfiguration and customizing to specific application needs. This platform enables users, regardless of their experience level, to rapidly adapt AIoT nodes to a wide array of application scenarios and deploy in different environments.

- **Communication** maintains a scalable interconnectivity with AIoT nodes which efficiently manages throughput and the number of connections. It is designed to handle large amounts of data traffic with low latency and reliable connections across devices and heterogeneous networks by implementing advanced communication methods.

- **Data Processing** allows predictive analytics, anomaly detection, and intelligent optimizations, specific to use cases. It processes data in real time to deliver relevant insights, enhancing decision-making and operational efficiency according to the specific requirements of a given application.

- **Security** ensures any access for nodes and users is managed, authenticated, and authorized, protecting the system's integrity and confidentiality. It implements strict measuresto manage access control, identity authentication, and secure interactions across the AIoT spectrum, protecting sensitive information and services from unauthorized users and potential attacks.

1.4 Benefits of an AIoT platform

In the UK, nearly 60% of manufacturing companies are SMEs. AIoT technology offers multiple benefits, including automated operations, increased productivity, cost reduction, and enhanced competitiveness.

First, an AIoT system gathers data from various heterogeneous sources, analyzes information according to AI models, offers insights from the results, and makes automated decisions. This allows for more efficient production processes since it is possible to monitor and control machines from centralized and remote places through high-end devices, such as computers and smartphones. AI models can also be used to continuously test quality on manufacturing lines.

By leveraging AI-driven insights, SMEs can optimize resource allocation, time management, and labor distribution, resulting in significant cost savings. Enhanced product quality and reduced manufacturing cycles enable SMEs to make their products more competitively priced. In addition to this, if they leverage these AI-powered big data models, they can adjust based on customer interactions, making their product better suited to the market.

Conversely, a sustainable ecosystem can be developed for the platform to foster collaboration and innovation. By enabling users to share their creations, the ecosystem simplifies and accelerates AIoT application development. A comparable model can be seen in platforms like Unreal Engine [11] and Unity [12], which provide digital game development tools and assets. Similarly, IFTTT [13] exemplifies an IoT ecosystem where users can create automation workflows using an "IF This Then That" logic. These ecosystems empower both professional developers and independent creators, offering opportunities for freelancing and entrepreneurial ventures.

1.5 Aim and Objectives of the Research

With the advancement of IoT applications, utilizing an IoT platform to facilitate data exchange and application deployment is essential. In general, the primary data flows in AIoT platforms fall into three categories as illustrated in Fig. 1.5.1: data-cloud-based, data-centralized, and data-decentralized. In a data-cloud-based platform, most of the resources and services are hosted and

managed by a cloud service provider. Data is uploaded and processed within the cloud, providing access to users/devices via its cloud interfaces. A data-centralized platform, on the other hand, can support a microservice-IoT (MS-IoT)[14] by functioning as a server broker, facilitating the collection, management, and redistribution of data to resources or services located in multiple discrete servers or end devices through a publish-subscribe (Pub-Sub) architecture. An entity can publish a piece of data to the broker, and multiple entities can subscribe to the data broadcast from the broker, thus enabling more complicated multicasting scenarios among multiple entities, enabling the adaptability of building more sophisticated AIoT solutions. Nevertheless, most of the implementations of these two categories mainly rely on common IoT communication protocols like HTTP/REST, MQTT, or CoAP, which are designed for massive connectivity to enable transmitting small data volumes. However, in many AI application scenarios that require multimedia streaming, high-bandwidth, multi-hop, and continuous data flows, there are deficiencies in such designs [15][16].

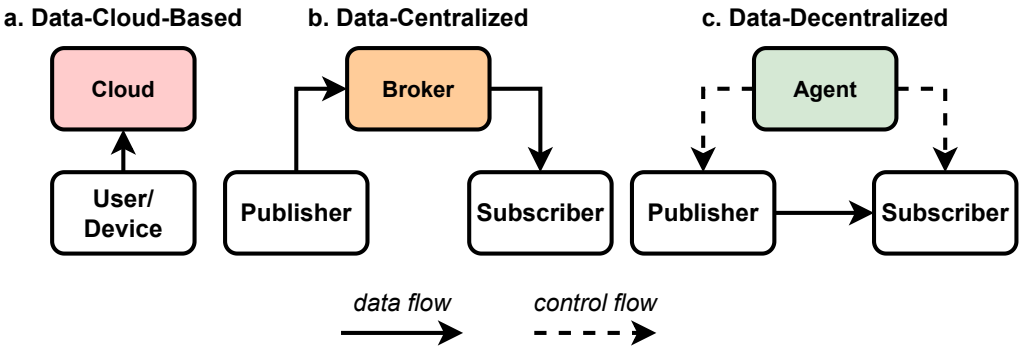


Figure 1.5.1: Categorization of IoT platforms based on data flows.

These challenges can be significantly mitigated through *data-decentralized* direct sender-to-receiver exchanges facilitated by a remote "Agent" used solely to establish connectivity. This work presents a prototype data-decentralized AIoT platform featuring peer-to-peer (P2P) communications powered by customized gRPC remote procedure calls based on the publish-subscribe (Pub-Sub) paradigm. The proposed framework enhances communication efficiency. As an

extension of IoTtalk [17], a high-level IoT platform that ensures device management with more adaptable node networking and provides a testbed for low-code development, thereby fulfilling deployment versatility. Moreover, this work introduces enhancements to improve resource scalability. Specifically, the Join Function (JF), originally responsible for handling pre-processing and post-processing during data transmission among nodes in the IoTtalk server, is redesigned to allocate customized functions directly to edge nodes. This offloading strategy enhances scalability by distributing computational tasks closer to the data sources. We refer to this version with data-decentralized communications as DAIoTtalk.

To demonstrate DAIoTtalk's AIoT capabilities, we have partnered with manufacturing collaborators and showcased two case studies: Sewingtalk and GNSS-EStalk.

SewingTalk is a system developed to estimate product completion on textile production lines by analyzing smart sewing machine log data. In many textile-exporting countries, the sewing manufacturing industry remains predominantly traditional, with manual planning and production management being the norm. To address this, we collaborated with one of the world's largest sewing machine manufacturers to conduct experiments using their latest smart sewing machines. These machines capture worker inputs and transmit operation logs to the cloud. By analyzing these logs using unsupervised learning techniques, we aim to identify patterns in production processes and estimate the number of finished products, leading to more accurate evaluations of individual worker performance and overall pipeline utilization.

GNSS-EStalk is a system that supports global navigation satellite system (GNSS) error source analysis using an AI-driven temporal-spatial approach. GNSS error source analysis is essential for identifying factors that impact the accuracy of positioning, navigation, and timing (PNT) services. Detecting and correcting these factors is crucial for improving overall service accuracy. Traditional methods

mainly focus on surface-level receiver output data, which may overlook deeper, underlying factors. Moreover, analyzing daily data can be costly and requires advanced expertise. GNSS-EStalk addresses these challenges by identifying highly consistent noise segments in daily data, which helps uncover potential causes. It also utilizes a multi-model deep learning approach to classify these noise segments and determine the sources of errors.

In summary, this research has the following aim, objective, research question, and novelty:

1.5.1 Aim

This work aims to develop DAIoTtalk, a data-decentralized AIoT platform based on IoTtalk. It enhances communication efficiency, deployment versatility, and resource scalability. These overcome the limitations of existing centralized IoT architectures in AIoT applications.

1.5.2 Objective

- To design and implement a novel data-decentralized pub-sub communication framework based on the gRPC protocol.
- To extend the existing IoTtalk platform to support P2P communication through the proposed gRPC pub-sub framework.
- To evaluate the communication efficiency and resource scalability of the proposed framework.
- To demonstrate the capability of the platform to support customized AI-driven analytics and classification.
- To validate the deployment versatility of the proposed DAIoTtalk through real-world case studies in sewing manufacturing by SewingTalk and GNSS error source analysis by GNSS-EStalk.

1.5.3 Research Question

- How can a gRPC-based publish-subscribe architecture be designed to support efficient and scalable data-decentralized communication in AIoT systems?
- What are the performance benefits of using a gRPC-based data-decentralized pub-sub framework over traditional centralized communication models in AIoT platforms?
- What architectural modifications are required to integrate a gRPC-based pub-sub framework into IoTtalk to support peer-to-peer communication?
- How does the proposed framework compare to traditional centralized architectures in terms of communication efficiency and resource scalability?
- To what extent can the platform enable domain-specific AI-driven analytics and classification in real-world AIoT use cases?
- How adaptable is the platform in supporting various AI models and classification tasks tailored to different AIoT scenarios?
- How versatile is the proposed DAIoTtalk platform in supporting diverse real-world AIoT applications, such as sewing manufacturing and GNSS error source analysis?
- Can DAIoTtalk effectively support deployment across heterogeneous AIoT domains with distinct data, communication, and processing requirements?

1.5.4 Novelty

- Supports direct P2P communication between nodes, eliminating reliance on traditional centralized brokers or cloud-based infrastructure for data routing.

- Develops a topic-based gRPC publish-subscribe framework, which is uncommon in P2P AIoT communication architectures.
- Extends the IoTtalk platform by integrating a gRPC-based publish-subscribe framework, providing a more scalable and efficient communication model tailored for modern AI-driven applications.
- Redesigns the JF to remotely allocate customized functions directly to edge nodes, enabling distributed processing at the network edge. This capability is rarely found in existing IoT platforms.
- Develops the SewingTalks, a system developed to estimate product completion on textile production lines by analyzing smart sewing machine log data.
- Develops the GNSS-EStalk, a system that supports GNSS error source analysis using a novel AI-driven temporal-spatial approach.

1.6 Chapter Description

We begin by presenting the background of this work in Chapter 2, which includes a survey of common IoT protocols, an overview of existing high-level IoT platforms, the applications of IoTtalk, and a critical analysis comparing these with the DAIoTtalk platform.

Next, we introduce DAIoTtalk in Chapter 3, detailing the gRPC data-decentralized framework and the implementation of DAIoTtalk.

Then, we present SewingTalk in Chapter 4, the first case study, to showcase the deployment versatility of DAIoTtalk in sewing industry. This chapter discusses related works, model developments, how the models are deployed on DAIoTtalk to support customization, and model evaluations.

Following that, we introduce GNSS-EStalk in Chapter 5, the second case study, to

showcase the deployment versatility of DAIoTtalk in PNT service. This chapter discusses related work, model developments, how the components are deployed in different DAIoTtalk projects for different purposes, and model evaluations.

After that, we provide an evaluation of the DAIoTtalk platform in Chapter 6, discussing how it enhances communication efficiency, deployment versatility, and resource scalability.

Finally, we provide conclusions in Chapter 7 to summarize this work, and suggest some future works.

2

Background and Survey

2.1 Survey on IoT Communication Protocol

Application layer communication protocols manage communication among IoT devices, enabling developers to control data flows and build IoT applications more efficiently. Table 2.1 lists common IoT application protocols, based on aspects from [21] and [18].

The Transport column indicates support for the Transmission Control Protocol (TCP), where the receiver sends an acknowledgment (ACK) upon receiving a packet, and the User Datagram Protocol (UDP), where the sender does not verify packet receipt. Quality of Service (QoS) refers to traffic control mechanisms that ensure the delivery of messages. Request-Response (Req-Resp) is a communication method where a device sends a request and waits for a reply to complete the data exchange. Publish-Subscribe (Pub-Sub) involves a data publisher sending a message to a topic, while data subscribers receive the

Protocol	Transport	QoS	req-resp	pub-sub	broker	Binary encoding	Ref.
REST/HTTP1.x	TCP	-	✓	-	-	-	[18], [19], [20]
MQTT	TCP	3 levels	-	✓	✓	✓	[18], [21], [22]
CoAP	UDP	2 levels	✓	✓	-	✓	[18], [23]
AMQP	TCP/UDP	3 levels	✓	✓	✓	✓	[18], [21], [24]
ZeroMQ	TCP/UDP	-	-	✓	-	✓	[21].[25]
gRPC	TCP	-	✓	-	-	✓	[26]

Table 2.1: Common IoT application layer protocol

message by subscribing to that topic. A broker facilitates message broadcasting to other devices upon receiving a message. Protocols that do not require a broker can perform Peep-to-Peep (P2P) communication. Binary encoding reduces the size of the data compared to string encoding when transmitting binary data, such as images or files, but also reduces the readability of the payload for humans.

2.1.1 REST/HTTP (HTTP/1.x)

Hypertext Transfer Protocol (HTTP)/1.x [20] is the most widely used client-server protocol for web applications. It operates over TCP to ensure reliable data delivery and follows a request-response communication model between the client and server. REST/HTTP is associated with REST [19], a messaging architecture that defines interaction methods such as POST and GET. Messages are typically encoded in UTF-8 plain text and structured using JavaScript Object Notation (JSON) [27], enhancing human readability during application development.

2.1.2 MQTT

Message Queuing Telemetry Transport (MQTT) [22] is a TCP-based messaging protocol. It is designed for large-scale IoT deployments, particularly for resource-constrained sensor nodes operating with low bandwidth, unstable networks, and limited power. MQTT follows a broker-client architecture within a Pub-Sub model. Although it is TCP-based, message loss can still occur due to factors such as wireless interference or sudden disconnections. To optimize transmission reliability and device efficiency, MQTT defines three QoS levels: 0 (no guarantee), 1 (at least once), and 2 (exactly once), with higher QoS levels requiring additional broker resources.

2.1.3 CoAP

The Constrained Application Protocol (CoAP) [23] is a UDP-based messaging protocol designed for devices with limited processing power and complex network conditions. Like REST/HTTP, it supports RESTful request-response interactions but uses a binary-encoded header, making it more lightweight than the string-encoded REST/HTTP header. CoAP matches requests and responses over UDP using a token value in the header. Additionally, Its observe function enhances GET interactions by enabling the server to push updates to clients, similar to a publish-subscribe model. CoAP defines two QoS levels: Non-Confirmable and Confirmable.

2.1.4 AMQP

The Advanced Message Queuing Protocol (AMQP) [24] is a messaging protocol that supports both reliable asynchronous and synchronous communication. It is designed to enable inter-process communication while efficiently handling high message volumes. AMQP facilitates a publish-subscribe architecture with two types of brokers: the Exchange and the Queue. The Exchange receives messages from publishers and routes them to one or more subscriber Queues, where messages remain until they are consumed. Starting from AMQP 1.0, the protocol also supports direct P2P communication without a broker. Its QoS mechanism is similar to MQTT, offering three levels of message delivery assurance.

2.1.5 ZeroMQ

ZeroMQ [25] is a high-level asynchronous messaging library that integrates multiple communication protocols, designed for distributed and concurrent system communication. Unlike AMQP, it provides message queuing functionality without requiring a broker. ZeroMQ natively supports both

Pub-Sub and Req-Resp communication models across various transports, including TCP and UDP for network communication, as well as in-process and inter-process communication for system-level messaging. An evaluation by [21] found that ZeroMQ maintains stable performance across different data volumes, particularly achieving high throughput with lower latency compared to MQTT and AMQP.

2.1.6 gRPC (HTTP/2)

gRPC [26] is an HTTP/2-based Remote Procedure Call (RPC) framework. Compared to HTTP/1.x, HTTP/2 utilizes a binary framing layer to encode messages in binary format, making it significantly more efficient than the plain text format of HTTP/1.x. gRPC performs request-response interactions by serializing messages with Protocol Buffers (Protobuf) [28], an Interface Definition Language (IDL) that enables cross-platform serialization and deserialization while being more efficient than JSON for data transmission [29]. In addition to the request-response model, HTTP/2 supports bidirectional streaming, making gRPC particularly effective for transmitting large volumes of data, such as camera frames for AI applications.

2.2 Analysis of Existing High-level IoT Platforms

The IoT platform enables remote management of large-scale IoT devices. Table 2.2 compare the platforms under study, summarizing key aspects of a high-level platform, identified in [36], [37], and [38]. The first column of each table lists these aspects, including Management, Development, Low-Code Configuration, Communication, Data Processing, and Security.

Notable IoT platforms include ThingsBoard [30], SensorCloud [32], and OpenRemote [34], which provide multi-application solutions. Temboo [31] specializes in environmental surveillance, while Fiware [33] serves as both an IoT

Table 2.2: Comparison of IoT Platforms

Platform		**Thingsboard**[30]	**Temboo**[31]	**SensorCloud**[32]	**Fiware**[33]	**OpenRemote**[34]	**IoTtalk**[17]
Management	Status	✓	✓	✓	✓	✓	✓
	Group	✓	-	-	✓	✓	-
	Application	✓	Location Only	-	✓	-	✓
Development	Ecosystem	-	-	-	✓	-	-
	API	Data exchange	Data exchange	Data exchange	Data exchange, Gateway, add-ons	Data exchange	Data exchange
	Customization	nodes	limited nodes	nodes	nodes, add-ons, IoT Agents	nodes, protocol, UI, apps	nodes
Low-Code Configuration	Flow Graph	For Rule	-	-	For data flow	For Rule	Separated pair For data flow
	Console	✓	✓	✓	✓	✓	✓
Communication	Protocol Integrated	HTTP1.1, MQTT, CoAP	HTTP1.1	HTTP1.1	MQTT, HTTP, websocket, etc	HTTP1.1, MQTT	HTTP1.1, MQTT
	Gateway	MQTT, CoAP, LoRaWAN, etc	MQTT, CoAP, etc	HTTP	LoRaWAN, self-define	-	NB-IoT, self-define
	High Throughout	-	-	-	By add-ons	-	-
Data Processing	Filter	✓	✓	✓	✓	✓	✓
	Maths	✓	-	✓	✓	✓	✓
	AI	Linear Prediction	-	-	Big Data analysis	-	[35]
Security	Accessing	✓	-	✓	✓	-	-
	Authentication	✓	✓	✓	✓	✓	✓
	Authorization	✓	✓	✓	✓	✓	✓

platform and a customizable framework for user-specific needs. IoTtalk [17] stands out as a simple, low-cost platform with strong integration capabilities, which we are evolving into an AIoT platform. The following sections will examine each platform in detail.

2.2.1 Management

For management, we focus on how to remotely oversee large-scale IoT nodes through an interface. Key aspects include status management, group management, and application management.

Status management enables users to track node status such as connectivity, battery level, and operational state remotely. Since this is one of the basic functionalities of the IoT management platform, it was found that all studied platforms offer this functionality.

Group management is necessary for efficiently operating large-scale IoT deployments. It enables users to categorize nodes based on their function, allowing a single configuration to be applied across multiple devices. ThingsBoard, Fiware, and OpenRemote natively support this feature, making it easier to manage homogeneous sensor nodes. In contrast, Temboo, SensorCloud, and IoTtalk require an additional gateway to group sensors, which is not a built-in feature and makes large-scale management more cumbersome.

Application management enables users to organize their devices into separate projects to support multiple applications. This is provided by ThingsBoard, Fiware, and IoTtalk. Temboo organizes all devices according to where they are located, and this is not particularly flexible. SensorCloud and OpenRemote do not natively support the concept of multi-projects. Users of those platforms have to create multiple accounts to cover different use cases.

2.2.2 Development

For Development, we evaluate the support provided by each platform for creating IoT applications. Key aspects include Ecosystem, Application Programming Interface (API), and Customization.

Ecosystem refers to the availability of templates or add-ons shared by developers or third parties, making application development more accessible. Among the studied platforms, only Fiware offers this feature, allowing users to enhance their platform with shared add-ons to tailor their IoT solutions to specific needs.

API is a toolkit for developers to create standard nodes. Most platform facilitates data exchange between devices and the cloud or other nodes by API. Additionally, Fiware also offers APIs that enable gateway and add-on development, strengthening its ecosystem.

Customization defines the level of adaptability a platform offers. Most platforms allow users to customize end nodes to send and receive specific data. However, Temboo is limited to predefined node types based on its specifications. Fiware extends customization capabilities through add-ons and IoT Agents, enabling the transmission of heterogeneous data. OpenRemote allows UI modifications for improved user experience and provides templates for mobile app development.

2.2.3 Low-Code Configuration

Low-Code Configuration provides a user-friendly graphical interface to create IoT applications. The low-code interface allows even users with minimal background knowledge to configure, adjust, and deploy nodes with little effort. There are 2 common types of low-code interfaces: Flow Graph and Console.

Flow Graph is a logic-based UI that connects components visually. It not only configures data flow but also enables users to structure their AI applications, which is clearer than explaining using text alone. ThingsBoard employs a flow-

based architecture to define data rules, such as calculations and if-else filters. Fiware uses a flow graph to configure communication between node parameters. IoTtalk, on the other hand, relies on a separate IDF/ODF pair design to represent data flow, which can be confusing for users.

Console typically consists of checkboxes, dropdown lists, and short text fields for variable preset configuration. As shown in Table 2.2, this is a fundamental low-code GUI component, and all the studied platforms provide this feature.

2.2.4 Communication

Communication support plays a crucial role in handling node volume and heterogeneous data transmission. Key factors include protocol integration, gateway support, and high-throughput capabilities for large-scale data processing.

Protocol Integration determines platform compatibility. All studied platforms support REST HTTP, the most common web protocol, but it is inefficient for handling massive or large-volume data due to UTF-8 encoding overhead. Additionally, REST HTTP requires an extra implementation to support a Pub-Sub architecture. ThingsBoard, Fiware, and OpenRemote natively support MQTT, a lightweight protocol designed for managing large-scale IoT deployments. Fiware also supports WebSocket and other protocols for high-volume data streaming.

Gateway support is essential for integrating non-native protocols and enabling communication between heterogeneous networks, such as Wireless Sensor Networks (WSNs). Platforms like ThingsBoard, Temboo, and Fiware support multiple IoT protocols, including CoAP and LoRaWAN. Additionally, Fiware allows users to customize gateways for specific needs. SensorCloud leverages HTTP gateways to aggregate data from multiple nodes, addressing scalability challenges. IoTtalk enables flexible connectivity for cyber-IoT devices, allowing

them to function as adaptive network gateways. [39] showcases an example using Narrowband Internet of Things (NB-IoT).

High Throughput is a key distinction between traditional IoT platforms and AIoT platforms. AI models require the transmission of large datasets, such as model weights, camera frames, and 3D point clouds. Most existing platforms do not support high-throughput communication, as it significantly increases maintenance costs in cloud-based or centralized systems. Fiware offers add-ons for streaming large data, such as video.

2.2.5 Data processing

When data is sent to the platform, it needs to be processed to extract insights that can be analyzed for decision-making. There are three main approaches to data processing: Filtering, Mathematical Operations, and AI Processing.

Filtering is a fundamental function for separating data, typically implemented using if-else logic. All the studied platforms support this feature.

Mathematical operations are used to apply a formula that transforms raw sensor data into numerical features. This is a common feature of centralized or cloud-based platforms, as applying functions to process data on a single server is easier. All platforms support this except Temboo, which is designed for ambient monitoring (temperature and humidity) and does not require a complex calculation.

Interestingly, different platforms implement filtering and mathematical operations in different ways. ThingsBoard and OpenRemote offer a flow-based GUI to create filtering and calculation rules, making it more user-friendly. Meanwhile, ThingsBoard, SensorCloud, Fiware, and IoTtalk integrate these operations with programming using IPython [40], providing greater flexibility for developers.

AI Processing is becoming essential in IoT, making it a key component of an AIoT platform. ThingsBoard supports basic AI functionality, such as linear prediction for forecasting input trends. Fiware offers add-ons for big data analysis. IoTtalk does not natively support AI processing, but there is work [35] to implement AI models in cyber-IoT devices.

2.2.6 Security

Cybersecurity is a major concern, especially for AIoT applications that rely on the Internet for data exchange. For keeping data and applications safe, security is a critical aspect of AIoT platform. Key aspects include access control, authentication, and authorization.

Access Control manages node and data permissions across users and projects because a single node may be involved in multiple projects. ThingsBoard, SensorCloud, and Fiware implement this by allowing users and nodes to be grouped within projects, simplifying access management.

Authentication indicates whether a user or node is authorized to access the system. The studied platforms commonly use username-password authentication or token-based authentication to validate access.

Authorization protects transmission from man-in-the-middle (MITM) attacks, where an unauthorized entity intercepts, accesses, or modifies data. Encryption and certification protocols, such as Transport Layer Security (TLS) [41] and OAuth2 [42], are widely adopted by the studied platforms to mitigate these risks.

2.3 IoTtalk Application

IoTtalk [17] is a key milestone in our AIoT platform. It is an IoT management platform with a low-cost, reconfigurable architecture designed to develop

interactive science experiments and provide IoT solutions for small- to medium-sized manufacturers. It has been widely adopted across various applications, including development testbed, smart city, agriculture, and AI. Here, we present some examples.

2.3.1 Development Testbed with IoTtalk

The low-cost and reconfigurable architecture enables IoTtalk to be used as a development testbed like ArduTalk [43], SimTalk [44], and NB-IoTtalk [39].

ArduTalk is a graphical programming environment for IoT development with Arduino. It provides a user-friendly interface for managing multiple Arduino boards connected to the cloud. Beginners can easily link and re-link connections between sensors and actuators in a low-code environment, enabling the rapid creation of various IoT projects.

SimTalk provides a simulation mechanism for ensuring the correct implementation and behavior analysis of IoT applications. It features animated simulations for both input (sensor) and output (actuator) IoT devices. To showcase the verification of IoT application configurations through simulations, it demonstrates examples such as a smart farm application, interactive art, and a pendulum physics experiment.

NB-IoTtalk proposes a service platform for rapidly developing NB-IoT applications. It employs a tag mechanism to offer an intuitive GUI for managing a large number of NB-IoT devices. To showcase large-scale IoT application deployment, a smart parking lot application with event-triggered reporting in NB-IoT is developed.

2.3.2 Smart City Application with IoTtalk

Smart city application is a typical IoT topic that brings convenience to daily lives. Many studies use IoTtalk to implement Smart City projects, including CampusTalk [45], HouseTalk [46], and FusionTalk [47].

CampusTalk introduces a collection of IoT applications deployed across a school campus, showcasing cyber-physical interactions. Examples include using smartphones as musical glow sticks for large campus concerts, virtual sports teaching and analysis of table tennis for physics lectures, and a cyber-physical artwork controlled via remote devices in the applied art institute.

HouseTalk presents a smart house based on the passive building design concept. It incorporates a 'green core', a plant wall integrated with IoT devices, to effectively reduce the house's energy consumption. Non-thermodynamic cycle systems, such as mechanical ventilation and evaporative cooling, purify the air and cool the space without relying on active Heating, Ventilation, and Air Conditioning systems.

FusionTalk introduces an object identification system for video surveillance, combining object recognition in camera frames with Bluetooth low-energy beacons. This integration enables precise localization, identification, and tracking of target objects within the monitored area. Administrators benefit from a GUI that not only enhances the visualization of object movement and behaviour, but also offers reconfigurable controls for managing cameras, IoT devices, and network applications.

2.3.3 Agriculture Application with IoTtalk

IoTtalk has been applied to the agriculture industry for crop and livestock management with sensors and actuators with network applications. AgriTalk[48], FishTalk[49], and PigTalk[50] are the example.

AgriTalk is designed for low-cost precision farming, which focuses on soil

cultivation. In precision farming, using sensors enables monitoring of data such as soil quality, weather conditions, and crop growth. Meanwhile, this data helps control actuators with precision, including spraying, drip irrigation, and repellent lights. With the IoTtalk platform, farmers can remotely and automatically manage crops or operate devices semi-automatically. A use case demonstrates curcumin farming in mountainous regions over long distances (more than 30 km), achieving a five times increase in production.

FishTalk introduces a novel fish-care system that utilizes aquarium sensors to control actuators in real time. The sensors measure key parameters such as temperature, pH, and water levels. The parameters regulate actuators like food dispensers, pumps, and ambient lighting. An intelligent feeding mechanism prevents overfeeding, while an integrated camera enables remote monitoring. The system also has the potential for large-scale aquaculture applications.

PigTalk leverages AIoT solutions to detect and mitigate piglet crushing in farrowing houses. The system monitors vocalizations, using machine learning to detect piglet distress calls. In emergencies, it automatically activates sow-alert actuators to prevent crushing. Integrated cameras enable real-time monitoring. Validated in a commercial farrowing house, the system saved piglets within 0.05 seconds with a 99.93% success rate.

2.3.4 AI Application with IoTtalk

There are other applications with AI models deployed on IoTtalk. [51], TrafficTalk [52] and MusicTalk [53] are the typical example.

[51] implements AI models as cyber devices for house valuation applications. It provides a non-physical IoT example of AItalk [35], a platform integrating AI and big data with IoT using IoTtalk, to estimate house values with models like Decision Tree, K-nearest neighbors, and Support Vector machine. Real datasets with factors like location, house ages, and parking space are experimented with

the system.

TrafficTalk estimates traffic queue dissipation using deep learning models. When a video clip of mixed traffic flow is captured, it is processed by a cyber IoT device equipped with image recognition models to identify vehicle types in the frames. The system then converts the video into vehicle density maps, which are analyzed by a convolutional neural network (CNN) to predict dissipation time.

MusicTalk utilizes IoTtalk to identify and classify musical instruments from audio recordings. It processes audio clips through a microservices node to extract features and predict instrument types by ensembling Vision Transformer and CNN models. The system also supports training and integrating new instrument models, expanding its detection capabilities.

2.4 Critical Analysis of Related Work

In general, IoT platforms can be categorized into three types: cloud-based, centralized, and decentralized. In this section, we provide an overview of each category, followed by a critical analysis of this work and the others.

2.4.1 Data-cloud-based

The data-cloud-based platform joins data into a cloud server operated by a service provider. [30], [34], and [32] are typical cloud-based IoT (CB-IoT) [54], providing service for data management and device monitoring. They often offer limited customization and control for testing and deploying AI models. [55] and [56] offer service-oriented IoT (SOA-IoT)[57] features. They treat IoT devices, sensors, and actuators as services that can be accessed or controlled through an interface. Nonetheless, they are typically designed for static or semi-static environments, which may not sufficiently satisfy the dynamic nature of AIoT development.

2.4.2 Data-centralized

The data-centralized platform establishes a broker or server for data gathering and redistribution, enabling flexible data coordination. [17] and [33] are the microservices-IoT (MS-IoT) [14] platforms that break down the IoT system into small, independently operating services with distinct functionalities, featuring flexibility for application development. However, they rely on common IoT application protocols like HTTP/REST, MQTT, or CoAP, which enable the connection of a massive device with small data volumes but are not well-suited for multimedia streaming. [58] and [59] utilize session initiation protocol (SIP) and advanced message queuing protocol (AMQP) broker respectively to overcome the limitation. Nevertheless, a single point of overload or failure is a potential concern for this approach, particularly considering that AIoT solutions often involve substantial data flow requirements.

2.4.3 Data-decentralized

The data-centralized platform allows devices to engage in P2P communication with each other, mitigating the risk of single-point failure and enhancing flexibility and resilience in device connectivity, which are crucial for AIoT applications. [60] is an MS-IoT platform designed to ensure data privacy and security through P2P connections among nodes. However, it primarily focuses on device management and data storage rather than comprehensive AIoT development. Our approach is data-decentralized with MS-IoT design but leans more towards AIoT than [60]. In our framework, we allow the deployment of AI models or algorithms as service nodes, enabling high-throughput streaming empowered by gRPC.

2.4.4 Summary of Research Gap

Table 2.3 summarizes the differences between this Work and existing IoT Platforms in terms of communication efficiency, deployment versatility, and

Table 2.3: Summary of Comparison Between This Work and Existing IoT Platforms

	Data-cloud-based	**Data-centralized**	**Data-decentralized**	
Platforms	[30], [34], [32], [55], [56]	[17], [33], [58], [59]	[60]	This Work
Communication Efficiency	Limited	Medium	High	High
Deployment Versatility	Low	Medium	Low	High
Resource Scalability	Medium	Limited	High	High

resource scalability.

For communication efficiency, a platform is expected to provide low-latency, high-throughput, and protocol-efficient communication. The data-cloud-based approach offers limited support in this regard, as it typically relies on SOA-IoT interfaces that are not optimized for dynamic or multimedia data. The data-centralized approach performs moderately, using traditional protocols (e.g., HTTP/REST, MQTT, CoAP) that are inefficient for high-throughput or multimedia scenarios. Although alternatives such as SIP and AMQP offer improvements, they still introduce risks of single-point failure or overload, which undermines scalability in AIoT deployments. In contrast, the data-decentralized approach excels in communication efficiency, as P2P communication reduces latency and enables direct device interaction. Our proposed gRPC-based pub-sub framework further enhances this by supporting efficient, real-time data streaming.

For deployment versatility, a platform should offer flexible support for various IoT nodes, AI models, network topologies, and deployment environments. The data-cloud-based approach provides limited versatility, as it typically restricts customization and relies on fixed service models, making it more suitable for static or semi-static environments. The data-centralized approach performs moderately. It features a modular design that allows flexible node deployment, but dynamic AI model integration remains limited. In the data-decentralized approach, [60] offers

low support in this area, focusing primarily on device connectivity while lacking AI model deployment capabilities. In contrast, this work provides high deployment versatility by supporting modular service nodes that can dynamically deploy both IoT nodes and AI models. This enables customized, adaptive deployments across diverse scenarios, from edge to cloud, tailored to specific application needs.

For resource scalability, a platform should be capable of handling increasing numbers of devices, larger data volumes, and heavier computational loads without performance bottlenecks. The data-cloud-based approach offers moderate scalability, as it generally scales well in the cloud but has limited adaptability at the edge. The data-centralized approach scores low in this regard, as it is prone to bottlenecks and single-point overload at the central server or broker, particularly under AI-intensive workloads. In contrast, the data-decentralized approach offers high scalability due to its distributed architecture, which enhances resilience and fault tolerance. This work adopts a data-decentralized, MS-IoT based design that distributes computation and data flow across nodes, potentially supporting horizontal scaling, load balancing, and efficient resource utilization across edge, fog, and cloud layers.

All in all, this work addresses the research gap by integrating a microservices-based IoT (MS-IoT) design with a data-decentralized architecture. It employs gRPC for efficient, high-throughput communication and enables IoT devices, AI models, and algorithms to be deployed as modular service nodes, supporting full-stack AIoT development. As a result, the proposed framework offers significant advantages in communication efficiency, deployment versatility, and resource scalability compared to existing approaches.

3

DAIoTtalk - A Data-Decentralized Pub-Sub AIoT Platform

3.1 Chapter Introduction

To address the flexibility and scalability requirements of AIoT communications, we develop a gRPC-based publish-subscribe framework in Section 3.2. This framework is then integrated into IoTtalk to form DAIoTtalk, as described in Section 3.3. A case study is also presented in this chapter to demonstrate the DAIoTtalk platform in practice.

3.2 gRPC Pub-Sub Framework

Fig. 3.2.1 shows the proposed Pub-Sub architecture using gRPC. In general, each publisher-subscriber pair should consist of a gRPC server and a gRPC client. The Agent is to resolve a client connection to a server according to the user-defined rules. The transmission mode can be either push or pull, indicating whether the client is a publisher or a subscriber respectively.

There are 4 'Re' steps in the frameworks, which are Register (ReG), Resolve (ReS), Request (ReQ), and Response (ReP). First, the node registers itself with the Agent. Next, the Agent resolves the connection between nodes based on the configuration. Then, the publisher requests data from the subscriber. Finally, the

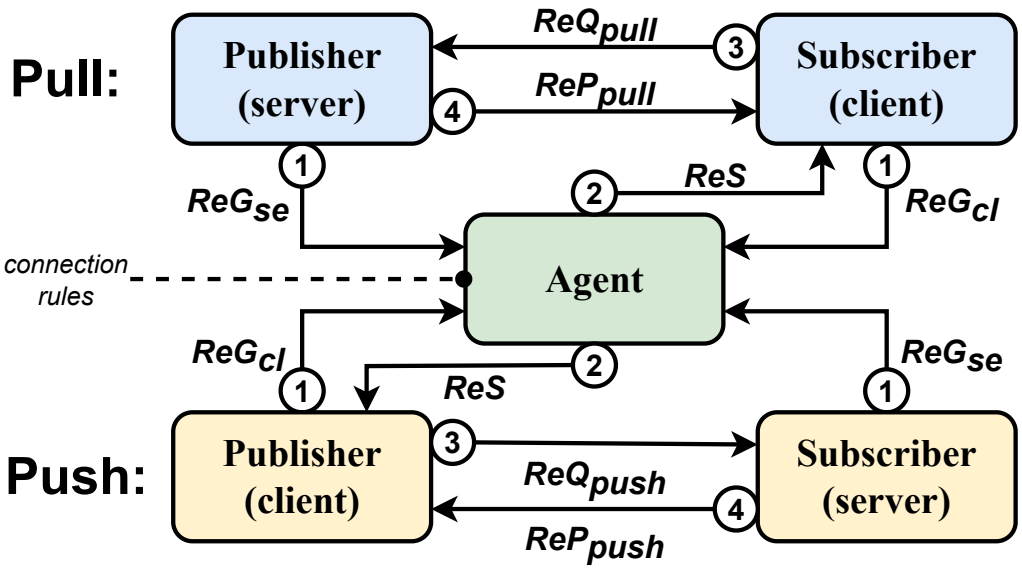


Figure 3.2.1: The gRPC Pub-Sub framework.

subscriber responds with the requested data. We detail them as follows.

Step 1 (ReG): Every node should send a register message to the Agent defined as follows:

$$\begin{aligned} ReG_{mode} =& (Node, Pub, Sub, IP), \text{ where} \\ mode \in& \{se = 0, cl = 1\} \end{aligned} \tag{3.2.1}$$

Here, *mode* is a flag representing the node's operational mode, where *cl* and *se* respectively denote client and server modes. *Node* contains a unique identification. *Pub* and *Sub* represent the sets of publishment and subscription topics required by the node, respectively. *IP* is the address of the server and it will be null if the node is a client.

Step 2 (ReS): When the Agent receives a *ReG* packet, information will be registered to a database. The Agent will try to link the subscribers to the publishers according to the user-defined connection rules, which will be described in Section 3.3.1. The Agent will periodically update the gRPC client with a resolving packet with a set of topic connection pairs denoted as follows:

$$ReS = \{(node_{pub}, node_{sub}, topic_{pub}, topic_{sub}, IP)_c\} \tag{3.2.2}$$

where $node_{pub} \in Node$ and $node_{sub} \in Node$ represent a publishment node and a subscription node, respectively, while $topic_{pub} \in Pub$ and $topic_{sub} \in Sub$ represent a publishment topic and a subscription topic, respectively. IP is the address connected to the gRPC server. $c \in [0, |ReS|)$ represents the index of each connection pair.

Step 3 (ReQ): When a gRPC client receives a ReS defined above, it will establish a bi-directional stream with the corresponding gRPC server using the request/response pattern, employing either the push or pull method based on its characteristics. By specifying $method \in \{push, pull\}$, the request message is defined as follows:

$$ReQ_{method} = (node, topic, payload) \tag{3.2.3}$$

If the gRPC client is a subscriber, it will employ the pull method by fetching data from the gRPC server (publisher). The client will initiate a ReQ_{pull} to the server using the received ReS, where $ReQ_{pull}[node]$ and $ReQ_{pull}[topic]$ will respectively contain the client node information $node_{sub}$ and the server publishment $topic_{pub}$. $ReQ_{pull}[payload]$ will be reserved. If the gRPC client serves as a publisher, a push method will be employed and the gRPC server will serve as a subscriber, where data is sent directly by the client in a request ReQ_{push} according to the ReS. $ReQ_{push}[node]$ represents the client node information $node_{pub}$, while $ReQ_{push}[topic]$ denotes the subscription topic $topic_{sub}$ provided by the server. Data will be attached in $ReQ_{push}[payload]$.

Step 4 (ReP): The server will reply to a response message according the *method* defined as follows:

$$ReP_{method} = (node, topic, payload, code). \tag{3.2.4}$$

When $method = pull$, the server will verify the $ReQ_{pull}[topic]$ in its Pub and check if there is unsent data in the corresponding buffer. It will then create a ReP_{pull} packet with a success status code in $ReP_{pull}[code]$. $ReP_{pull}[node]$ will be

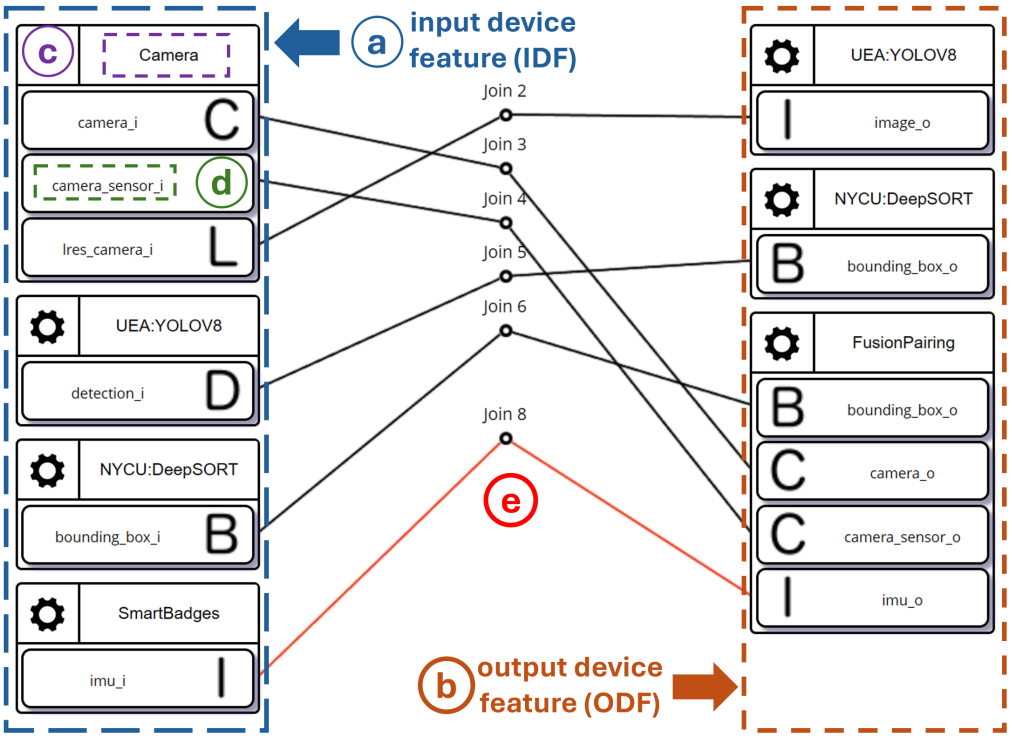


Figure 3.3.1: A DAIoTtalk device-object pairing project reference from [61].

the server node information and $ReP_{pull}[topic] = ReQ_{pull}[topic]$. The requested data will be attached to $ReP_{pull}[payload]$. Since the connection is established in a bidirectional stream, the server will persistently send the ReP_{pull} messages until there is no new data, but the client will periodically send the ReQ_{pull} message to maintain the connection.

On the other hand, if $method = push$ when receiving the ReQ_{push}, the server will respond with a ReP_{push} message. It will verify if the $ReQ_{push}[topic]$ belongs to it and indicate the success code in $ReQ_{push}[code]$. Additionally, it will include its node information in $ReP_{push}[node]$ and set $ReP_{push}[topic] = ReQ_{push}[topic]$. The $ReP_{push}[payload]$ will be reserved at this time. In the push method, the client will keep pushing ReQ_{push} and the server will keep providing feedback ReP_{push}, forming a bidirectional stream.

3.3 The Proposed DAIoTtalk

DAIoTtalk is evolved from IoTtalk [17]. IoTtalk is a platform that facilitates low-code deployment through a user-friendly graphical user interface (GUI). It provides efficient project management capabilities accessible to users of varying technical backgrounds. We integrate the gRPC Pub-Sub framework into IoTtalk, resulting in the creation of DAIoTtalk. Fig. 3.3.1 shows a DAIoTtalk device-object pairing project derived based on the IoTtalk framework. The application is detailed on [61], chaining multiple AI models with video streaming. Details will be described in Section 3.3.5.

In IoTtalk, data is classified into two categories: input device feature (IDF) as illustrated in Fig. 3.3.1a and output device feature (ODF) as in Fig. 3.3.1b. Each IoT node is constructed as a device model (DM) like Fig. 3.3.1c, comprising various device features (DF), where $DF \in \{IDF, ODF\}$ as depicted in Fig. 3.3.1d. A DF serving as a data source is categorized as an IDF. Conversely, a DF that receives data is classified as an ODF. A user can easily build a connection rule for Eq. (3.2.2) between an IDF and an ODF by drawing a 'Join' line as shown in Fig. 3.3.1e in the GUI.

3.3.1 Integration of DAIoTtalk

To integrate the gRPC Pub-Sub frameworks into the IoTtalk system, we allocate the IDF as the publisher and the ODF as the subscriber. Fig. 3.3.2 depicts how we extend IDF and ODF by incorporating gRPC using the example in Fig. 3.3.1. This architecture consists of two domains: node domain and network domain.

- The node domain contains all device and service nodes under the DAIoTtalk. For each node, we implement an application programming interface (API) called 'AGAPI' as shown in Fig. 3.3.2a. The API integrates the IoTtalk DA, which is responsible for registration with the IoTtalk server, and the

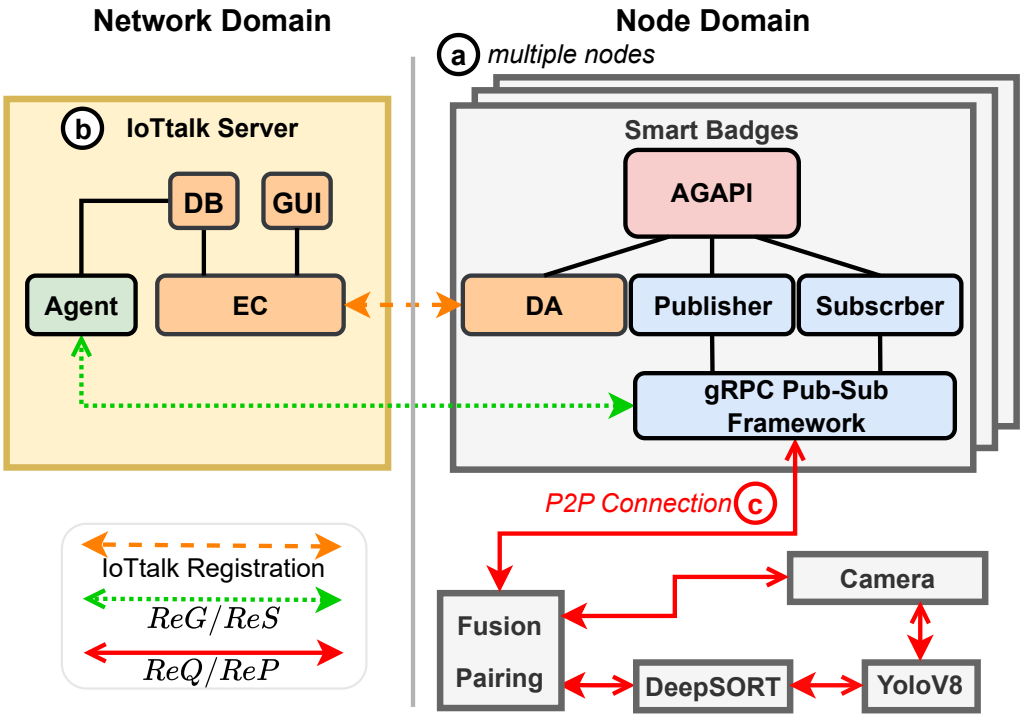


Figure 3.3.2: Extension of IDF and ODF in DAIoTtalk.

Pub-Sub framework outlined in Section 3.2 for communication among other nodes.

- The network domain addresses the remote service for managing the AIoT solution. Fig. 3.3.2b is the IoTtalk server with the Agent component of the pub-sub framework. It is deployed in the network domain. The execution and communication (EC) system receives messages from the DA and collaborates with the GUI, as well as a database (DB) that stores the configuration data. The Agent components receive registrations from gRPC clients and gRPC server nodes using the *ReG*. It then scrapes the relevant configuration from DB to construct the *ReS* and sends it to the corresponding nodes.

Upon receiving the connection rules from the Agent, the nodes in the node domain establish P2P communication, as illustrated in Fig. 3.3.2c, following the steps *ReQ* and *ReP*.

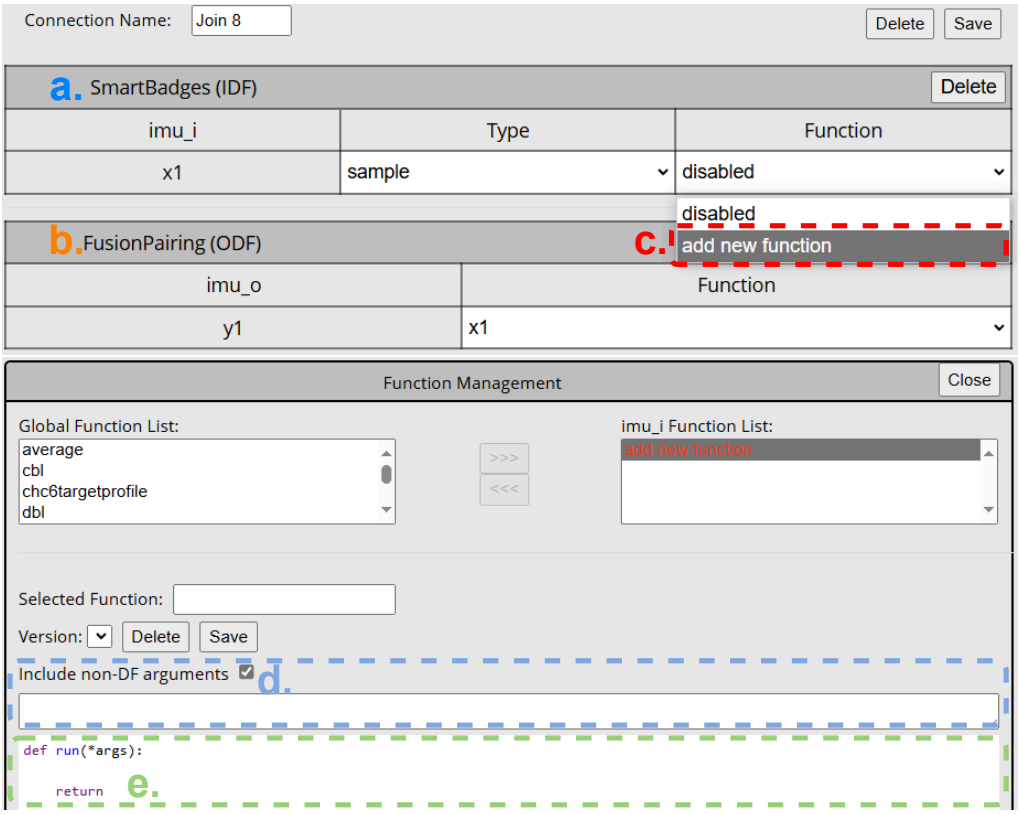


Figure 3.3.3: The GUI for configuring JFs in IoTtalk.

3.3.2 Join Function

When a join connection is established in IoTtalk, as shown in Fig. 3.3.1e, the user can define a *Join Function (JF)*, which is a script that processes data before transmission or reception. Fig. 3.3.3 illustrates the GUI for configuring this function. The function can be set up for both the IDF and ODF, as shown in Fig. 3.3.3a and Fig. 3.3.3b, respectively. The user can access the *function management* interface by selecting the option shown in Fig. 3.3.3c. In this interface, the user can define function arguments and write the Python script[62], as depicted in Fig. 3.3.3d and Fig. 3.3.3e, respectively.

Since Python is a cross-platform interpreted language [63] that does not require compilation, this design allows for flexible configuration and supports high-level deployment.

The Join Function (JF) is executed on the centralized server in IoTtalk, whereas

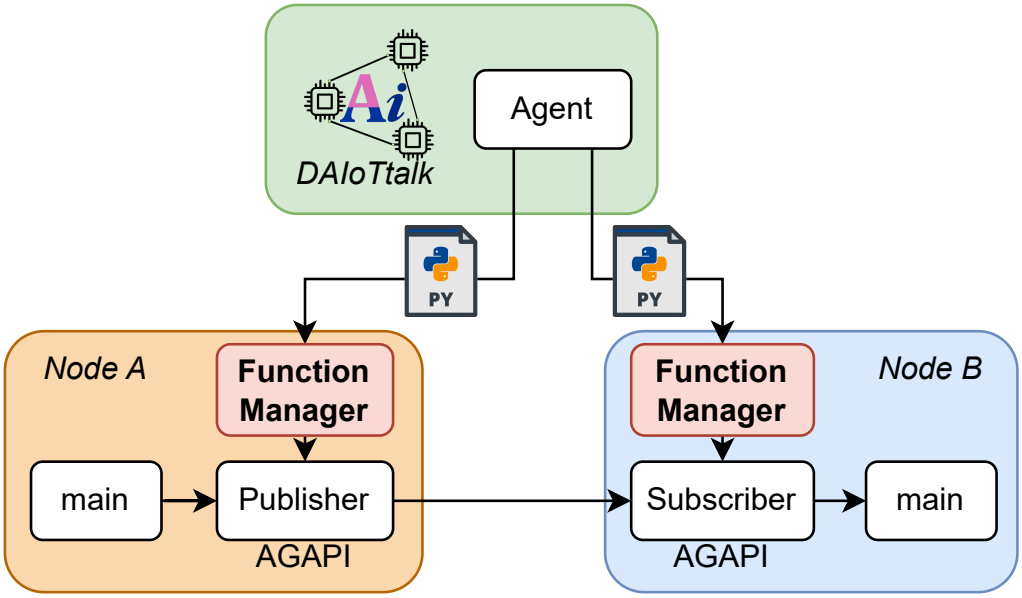


Figure 3.3.4: The data flow for delivering JFs to nodes in DAIoTtalk.

in DAIoTtalk, the scripts must be delivered to decentralized nodes. Fig. 3.3.4 illustrates the delivery process. The Agent distributes the scripts to the corresponding nodes based on the specified configurations. In each node, the *function manager* in AGAPI receives and manages the scripts. To verify whether the script version matches the one already received, an *MD5* hash is generated and compared using the message-digest algorithm [64]. If the MD5 hashes are matched, the script is considered identical, and the latecomer is rejected. Otherwise, the script is imported into the Publisher for data processing before transmission or into the Subscriber before the main thread receives it. With the data-decentralized design, a more complex JF can be executed to enhance flexibility.

3.3.3 Connectivity Configuration

Fig. 3.3.5 illustrates the detailed procedure of the configuration cycle in DAIoTtalk, which consists of three phases: setup, connection configuration, and function configuration. This cycle is repeated periodically. Each phase is described in detail below.

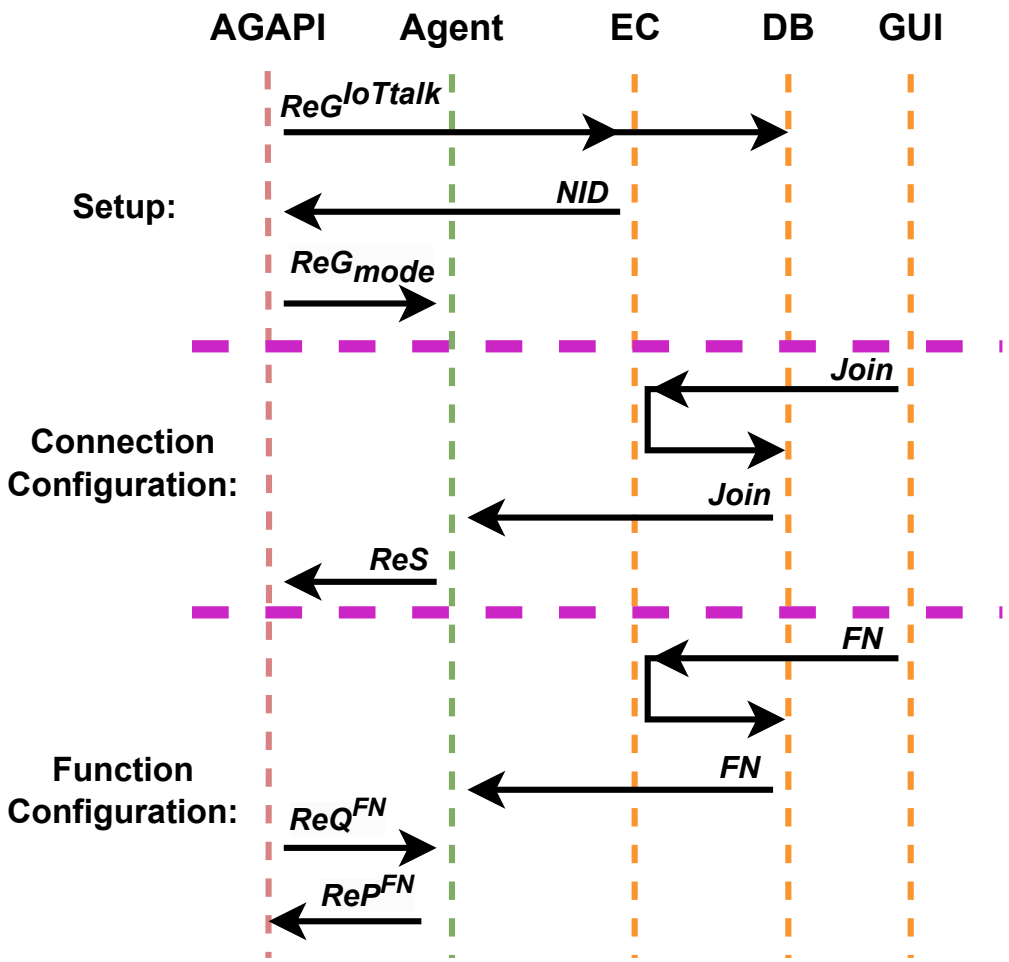


Figure 3.3.5: The procedure for configuring connectivity in DAIoTtalk.

Setup Phase

A node needs to register with the EC through the AGAPI. The API will transmit a message containing the DM along with lists of IDF and ODF defined as follows:

$$\begin{aligned} ReG^{IoTtalk} &= (DDM, \{IDF_{i\geq 0}\}, \{ODF_{o\geq 0}\}), \text{where} \\ DDM &= Domain + DM \end{aligned} \tag{3.3.1}$$

In IoTtalk, a single DM created in the GUI can only be assigned to one device. To address this limitation, we introduce the DDM, which combines the DM with an optional field $Domain$, allowing a group of nodes to share the same connection rules within DAIoTtalk. The EC will store the $ReG^{IoTtalk}$ into DB and return a

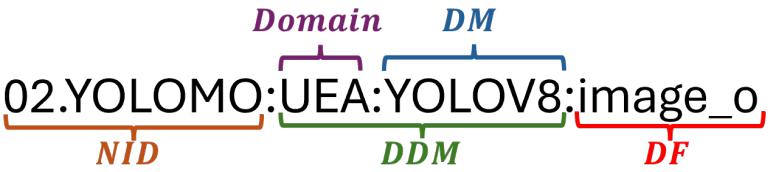


Figure 3.3.6: An example of *Pub* or *Sub* topic name.

Table 3.1: The four types of DFO are used to control the broadcasting of topics.

Priority	DFO Type	DFO Format
0	Native DFO	$\{Node\}:\{Domain\}:\{DM\}:\{DF\}$
1	DDM DFO	$\%:\{Domain\}:\{DM\}:\{DF\}$
2	Domain DFO	$\%:\{DM\}:\{DF\}$
3	DM DFO	$\{DM\}:\{DF\}$

unique identification denoted as NID. Next, the AGAPI will initialize a ReG_{mode} message described in Eq. (3.2.1) for the Pub-Sub Agent. We express the $Node$, Pub, and Sub field in ReG_{mode}as follow:

$$Node = (NID, DDM) \tag{3.3.2}$$

$$Pub = \{Pub_i\} = \{(NID + DDM + IDF_i)_i\} \tag{3.3.3}$$

$$Sub = \{Sub_o\} = \{(NID + DDM + ODF_o)_o\} \tag{3.3.4}$$

Fig. 3.3.6 shows the topic content for the the example in Fig. 3.3.1. Through this expression, the Agent can flexibly manage nodes and their associated topics.

Connection Configuration Phase

A Join is created to connect an IDF and an ODF in the IoTtalk GUI. A set of Join messages for the connection is sent to EC and then stored into the DB. These messages carry information as follows:

$$Join = \{(naID_{n\geq 0}, DFO_{d\geq 0})\}, \tag{3.3.5}$$

where $naID_n$ is an identification for each Join.

DFO_d is the Device Feature Object (DFO), representing a format of topics used to control broadcasting. Table 3.1 lists the four types of DFO: Native DFO, DDM DFO, Domain DFO, and DM DFO:

- **Native DFO** unicasts only to the exactly matched topic.
- **DDM DFO** broadcasts to topics matching the DDM.
- **Domain DFO** broadcasts to topics matching the Domain.
- **DM DFO** broadcasts to topics matching the DM.

Additionally, a lower-priority DFO cannot overwrite the settings of a higher-priority DFO.

When a set of DFO_d shares the same $naID_n$, it means that they are connected in the same Join. The Agent will periodically retrieve the Join information from the DB and classify whether a DFO_d is an IDF or an ODF by matching them with the registered Pub and Sub. If a pair of $topic_{pub}$ and $topic_{sub}$ is matched, the Agent will update a row of ReS described in Eq. (3.2.2) for the corresponding gRPC client-side node.

Function Configuration Phase

When a JF has been configured in the IoTtalk GUI, the configuration will be stored in the DB by the EC. We denote the configuration set as FN

$$FN = \{\, FN_f \,\} = \{(DFO_d, fnID_f, Py_f, Args_f)_f | f \geq 0\}, \tag{3.3.6}$$

Here, each configuration FN_f contains the script ID $fnID_f$, the Python script Py_f, and the arguments $Args_f$, which are associated with the DFO_d. The Agent will immediately extract the FN from the DB.

For each node, AGAPI will periodically send a set of requests $ReQ^{FN} = \{\, ReQ^{FN}_{topic} \,\}$ to Agent to request updated JFs for every

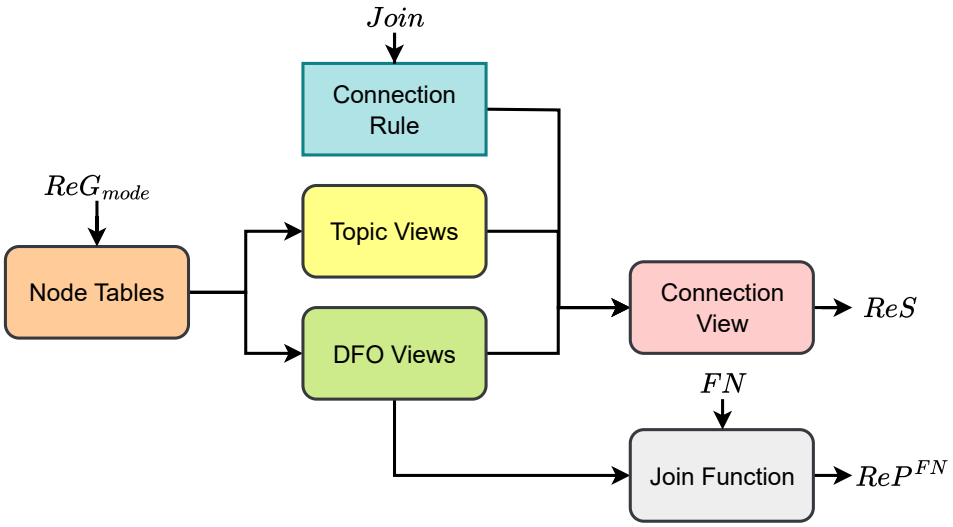


Figure 3.3.7: The architecture of the Agent database.

$topic \in (Pub \cup Sub)$ that it publishes or subscribes to:

$$ReQ^{FN}_{topic} = (Node, topic) \tag{3.3.7}$$

The Agent then sends the responses $Rep^{FN} = \{ Rep^{FN}_{topic} \}$ to AGAPI for each requested $topic$ along with the corresponding Py_f and $Args_f$:

$$Rep^{FN}_{topic} = (topic, Py_f, Args_f) \tag{3.3.8}$$

3.3.4 Agent Database

The connectivity of DAIoTtalk is primarily managed by the Agent database, as illustrated in Fig. 3.3.7.

When a ReG_{mode} package in Eq. (3.2.1) is received, the Agent stores its information in the node tables, which then generate the corresponding topic and DFO views.

Upon receiving a $Join$ package in Eq. (3.3.5), the Agent updates the connection rule table, linking topic and DFO views to generate the connection view. This enables the Agent to extract the necessary information to construct the ReS

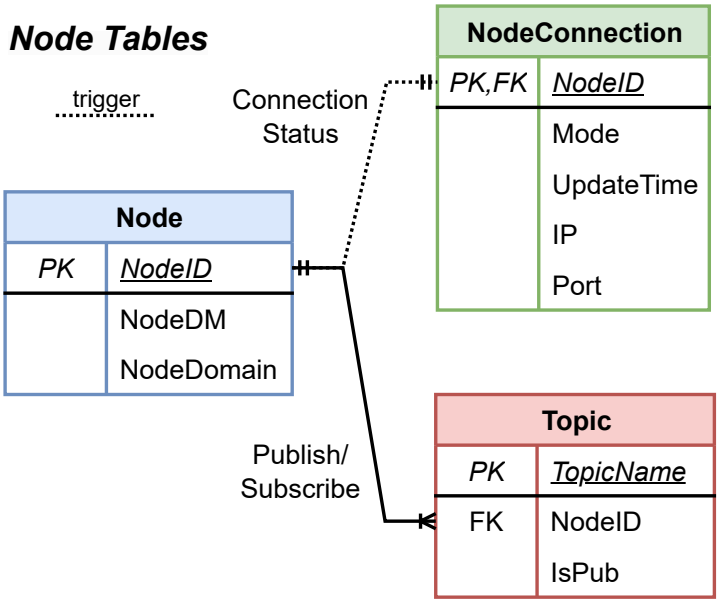


Figure 3.3.8: The architecture of the node tables in the Agent database.

package in Eq. (3.2.2).

Conversely, when an FN package in Eq. (3.3.5) is received, the Agent processes it through the JF module and constructs the ReP^{FN} package in Eq. (3.3.8).

In the following, we detail each component of the Agent database in Fig. 3.3.7.

Node Tables

Fig. 3.3.8 illustrates the tables that store the registration ReG_{mode} in Eq. (3.2.1), including the Node table, Topic table, and NodeConnection table.

The Node table records the node information from ReG_{mode}, as specified in Eq. (3.3.2). It includes the NodeID (NID), NodeDM (DM), and NodeDomain ($Domain$). The NodeID is the primary key and prevents inserting a duplicate NodeID.

The NodeConnection table stores connection information from ReG_{mode} specified in Eq. (3.2.1). The columns include the connection mode ($mode$) and the address (IP), which consists of an IP and port. The UpdateTime field

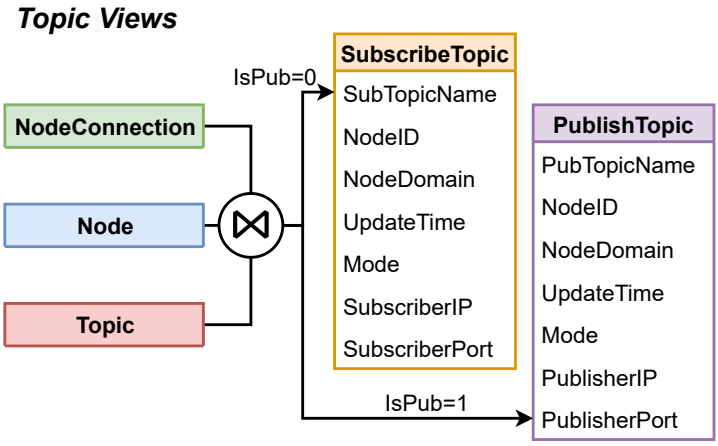


Figure 3.3.9: The architecture of the topic views in the Agent database.

records the timestamp when a ReG_{mode} is received. The NodeID serves as both the primary key and a foreign key referencing the NodeID in the Node table. A trigger is created to initialize the NodeConnection table when a new node is inserted into the Node table. Since the Node table and NodeConnection table have a one-to-one relationship, the trigger executes only once per row in the Node table.

The Topic table stores topic information from ReG_{mode}. TopicName is the primary key and corresponds to either a publishment (Pub_i) from Eq. (3.3.3) or a subscription (Sub_o) from Eq. (3.3.4). The IsPub flag indicates whether the TopicName represents a Pub_i ($IsPub = 1$) or a Sub_o ($IsPub = 0$). NodeID is the foreign key that links each record to the corresponding NodeID in the Node table. The relationship between the Node table and the Topic table is one-to-many, meaning a node can publish or subscribe to multiple topics.

In addition, the foreign key deletion constraints for the NodeConnection and Topic tables are set to cascade. When a row in the Node table is removed, the corresponding records in the NodeConnection and Topic tables are automatically deleted.

Topic Views

Fig. 3.3.9 illustrates two views that summarize the topic status recorded in the Node, NodeConnection, and Topic tables. These views, SubscribeTopic and PublishTopic, represent subscription and publishment information, respectively. The SubscribeTopic view, denoted as V^{ST}, joins information from the Node, NodeConnection, and Topic tables by their NodeID fields, represented as TB^N,TB^{NC} and TB^T, respectively:

$$\begin{aligned} V^{ST} &= TB^N \bowtie TB^{NC} \bowtie TB^T, \text{ ON} \\ TB^N.NodeID &= TB^{NC}.NodeID, \\ TB^N.NodeID &= TB^T.NodeID, \\ TB^T.IsPub &= 0 \end{aligned} \tag{3.3.9}$$

Similarly, the PublishTopic view, denoted as V^{PT}, joins the three tables when $IsPub = 1$ in Topic table:

$$\begin{aligned} V^{PT} &= TB^N \bowtie TB^{NC} \bowtie TB^T, \text{ ON} \\ TB^N.NodeID &= TB^{NC}.NodeID, \\ TB^N.NodeID &= TB^T.NodeID, \\ TB^T.IsPub &= 1 \end{aligned} \tag{3.3.10}$$

DFO Views

Fig. 3.3.10 illustrates the process of translating topics into DFOs in the database, as described in Table 3.1. Each type of DFO is recorded in its corresponding view along with the original topic name and the IsPub flag. The views include NativeDFO (denoted as V^{NaDFO}), DDMDFO (denoted as V^{DDMDFO}), DomainDFO (denoted as V^{DomDFO}), and DMDFO (denoted as V^{DMDFO}). These views are grouped by the DFO and TopicName fields.

Additionally, the DFOTranslation view, denoted as V^{DFO}, provides a summarized

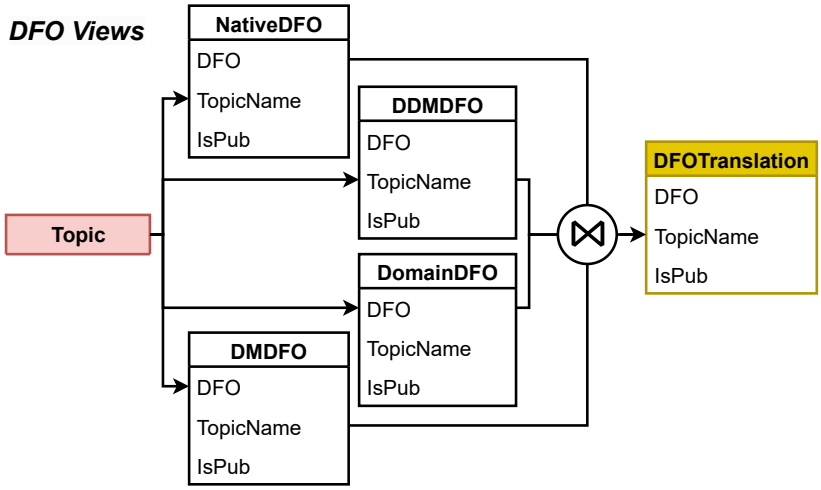


Figure 3.3.10: The architecture of the DFO views in the Agent database.

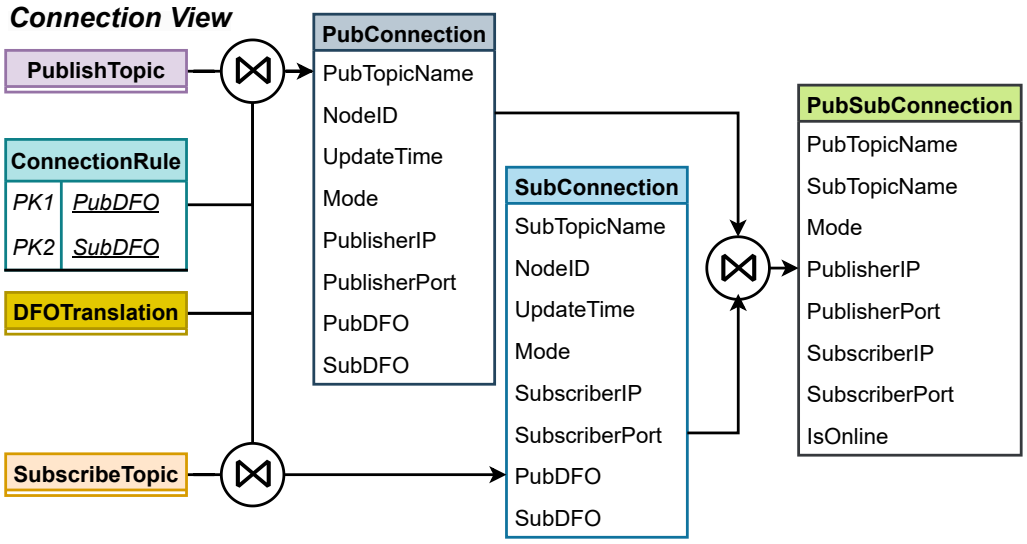


Figure 3.3.11: The architecture of the connection views in the Agent database.

representation by aggregating data from the four individual DFO views.

$$V^{DFO} = V^{NaDFO} \bowtie V^{DDMDFO} \bowtie V^{DomDFO} \bowtie V^{DMDFO} \tag{3.3.11}$$

Connection Views

Fig. 3.3.11 illustrates the process of establishing a connection between publishment and subscription in the database. When a *Join* request defined in Eq. (3.3.5) is received, the Agent matches topics that share the same $naID_n$ and

the corresponding DFO_d in the DFOTranslation table to determine whether they are registered as publishments or subscriptions. The matched pairs are then used to construct the ConnectionRule table, denoted as TB^J, with both the publishment (PubDFO) and subscription (SubDFO) serving as composite keys. This ensures that each pair is inserted only once.

Next, the PubConnection and SubConnection views are generated to summarize the topic information and the paired DFO. The PubConnection view, denoted as V^{PC}, is constructed for publishment by joining the PublishTopic view V^{PT} from Eq. (3.3.10) and the DFOTranslation view V^{DFO} from Eq. (3.3.11), along with TB^J:

$$\begin{aligned} V^{PC} &= V^{PT} \bowtie V^{DFO} \bowtie TB^J, \text{ ON} \\ TB^J.PubDMF &= V^{DFO}.DMF, \\ V^{DFO}.TopicName &= V^{PT}.PubTopicName, \\ V^{DFO}.isPub &= 1 \end{aligned} \tag{3.3.12}$$

Similarly, the SubConnection view, denoted as V^{SC}, is constructed for subscription by joining the SubscribeTopic view V^{ST} from Eq. (3.3.9) and the V^{DFO}, along with TB^J:

$$\begin{aligned} V^{SC} &= V^{ST} \bowtie V^{DFO} \bowtie TB^J, \text{ ON} \\ TB^J.SubDMF &= V^{DFO}.DMF, \\ V^{DFO}.TopicName &= V^{ST}.SubTopicName, \\ V^{DFO}.isPub &= 0 \end{aligned} \tag{3.3.13}$$

Finally, a PubSubConnection view, denoted as V^{Conn}, is constructed to combine

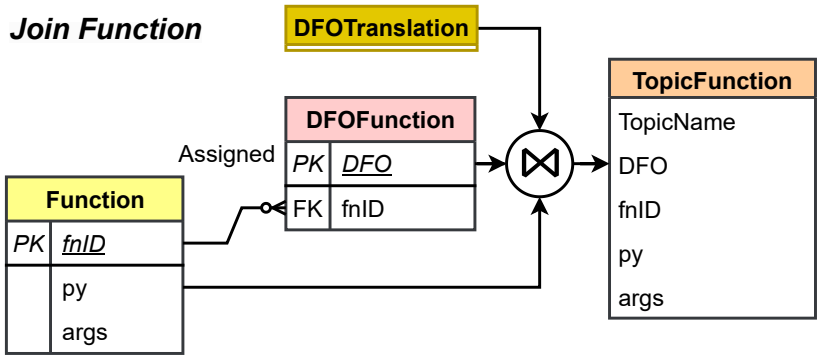


Figure 3.3.12: The architecture of the join function in the Agent database.

V^{PC}, and V^{SC}:

$$\begin{aligned} V^{Conn} &= V^{PC} \bowtie V^{SC}, \text{ ON} \\ V^{PC}.PubDMF &= V^{SC}.PubDMF, \\ V^{PC}.SubDMF &= V^{SC}.SubDMF, \text{ where} \\ V^{Conn}.Mode &= (V^{PC}.Mode - V^{SC}.Mode), \\ V^{Conn}.IsOnline &= min(V^{PC}.UpdateTime, V^{SC}.UpdateTime) \\ &> (now - \delta_{timeout}) \end{aligned} \tag{3.3.14}$$

Here, if $V^{Conn}.Mode \leq 0$, the Pull method in Fig. 3.2 will be applied; otherwise, the Push method will be used. The IsOnline field is a flag to filter out nodes that have been disconnected for longer than the threshold $\delta_{timeout}$. The Agent constructs ReS in Eq. (3.2.2) based on the rows in the PubSubConnection view. Notably, if both the publishment address (PublisherIP and PublisherPort) and the subscription address (SubscriberIP and SubscriberPort) are empty, the Agent will discard the row, as this indicates that both the publisher and subscriber are operating in client mode (i.e., $mode = cl$ in Eq. (3.2.1))

Join Function

Fig. 3.3.12 illustrates the pairing between topics and JFs in the database. When an FN from Eq. (3.3.6) is received by the Agent, the JF information is stored in two tables: the Function table and the DFOFunction table.

The Function table manages script information, including the script ID 'fnID' ($fnID_f$), the Python script 'py' (Py_f), and the arguments 'args' ($Args_f$). The $fnID_f$ is the primary key for queries.

The DFOFunction table stores the relationship between a DFO (DFO_d) and the script's fnID. The DFO is the primary key, while the fnID is a foreign key referencing the Function table. The relationship between the Function table and the DFOFunction table is one-to-many, meaning multiple DFOs can share a single script.

To summarize the relationship between a topic and a script, a TopicFunction view, denoted as V^{TF}, is generated by joining the Function table (TB^{FN}), the DFOFunction table (TB^{DFOFN}), and the DFOTranslation view (V^{DFO}) from Eq. (3.3.11):

$$\begin{aligned} V^{TF} &= TB^{FN} \bowtie TB^{DFOFN} \bowtie V^{DFO}, \text{ ON} \\ TB^{DFOFN}.DFO &= V^{DFO}.DFO \\ TB^{FN}.fnID &= TB^{DFOFN}.fnID \end{aligned} \tag{3.3.15}$$

When a $ReQ^{FN}topic$ from Eq. (3.3.7) is received, the Agent uses the *topic* to query V^{TF} and constructs $Rep^{FN}topic$ from Eq. (3.3.8) to distribute the JF script to the corresponding nodes.

3.3.5 Case Study: Deployment of AI Device-Object Pairing

We developed an AI device-object pairing project as a demonstration for our DAIoTtalk, as illustrated in Fig. 3.3.1. The project and data are modified from the author's previous work [61], which pairs several smart badges with the people detected by a camera. The project involves 5 DMs which are $Camera$, $SmartBadges$, $UEA : YOLOV8$, $NYCU : DeepSORT$, and $FusionPairing$, with corresponding DFs.

- The *Camera* publishes original (1280x720 pixel) and resized (640x360 pixel) camera frames, encoded in JPG format, on $camera\_i$ and $lres\_camera\_i$, respectively. It also publishes the $camera\_sensor\_i$ with the orientation of the camera.

- The *SmartBadges* represents the devices placed on the body. The built-in inertial measurement unit (IMU) data will be published by imu_i.

- The $UEA : YOLOV8$ is a service utilizing the YoloV8 [65] object detection model. The model subscribes to the input image from $image_o$ to detect objects within it. The detection results are then published through the $detection\_i$ interface.

- The $NYCU : DeepSORT$ is a service utilizing the DeepSort [66] tracking model. The model subscribes detection results from $bounding\_box\_o$ and assigns tracking ID to each object within it. The tracking results are then published through the $bounding\_box\_i$ interface.

- The *FusionParing* involve the device-object pairing model from [61]. The node takes camera frames and the orientation from $canmera\_o$ and $camera\_sensor\_o$ respectively. The model receives the IMU data from imu_o. The tracking result will be received by $bounding\_box\_o$. The node lastly outputs the labeled frame.

In short, The *Camera* is responsible for sending low-resolution camera frames to $UEA : YOLOV8$ and transmitting the original frames and camera orientation to *FusionPairing*. The *SmartBadges* send IMU data related to a person to *FusionPairing*. The $UEA : YOLOV8$ processes the camera frames using the YOLO [65] object detection model, then sends the detection results to $NYCU : DeepSORT$. The $NYCU : DeepSORT$ assigns a tracking ID to each detected object using the DeepSORT model [66] and forwards this information to *FusionPairing*. Finally, *FusionPairing* fuses all the collected data and generates a device-object pairing result.

4

SewingTalk - A Product Completion Estimation System with Unsupervised Learning for Smart Sewing Machines

---

[This section is redacted due to Intellectual Property access restrictions.]

5

GNSS-EStalk - A Novel AI Temporal-Spatial Analysis Approach for GNSS Error Source Recognition

---

5.1 Chapter Introduction

Global navigation satellite systems (GNSS), including the global positioning system (GPS), have been developed to offer comprehensive positioning, navigation, and timing (PNT) services with worldwide coverage. In these systems, L-band radio-frequency signals are transmitted from satellites and received by ground-based GNSS receivers. By processing these signals, the receivers calculate their distances from the observed satellites, enabling them to determine an accurate PNT solution. Nevertheless, the accuracy of PNT solutions heavily depends on the quality of the GNSS observable, which is affected by various GNSS error sources [67]. These include satellite clock and ephemeris errors, atmospheric delays, cycle slips, interference and jamming, etc. All of these errors can be expressed in units of distance, and must be detected and corrected to improve accuracy [68].

Conventional error detection methods encounter several limitations. Firstly, methods such as those described by [69][70] typically focus on analyzing common receiver output parameters like elevation, observation data,

signal-to-noise ratio (S/N), and PNT results. However, these methods lack the depth compared to a more comprehensive analysis of the parameters involved in calculating PNT [71]. Besides, the PNT system generates an enormous volume of data daily, making it highly challenging to extract consistent error segments. Detecting these segments becomes a fundamental step in diagnosing the root cause of errors and improving the accuracy of the overall PNT system [72][73].

In this chapter, we propose an artificial intelligence (AI)-based temporal-spatial approach for the automatic recognition of noise types in segments using classification models. However, training these models typically requires manual labeling of noise data, which is both costly and demands significant expertise. To address this, we perform clustering to group highly similar noise segments, then apply a z-score normalization filtering (ZFilter) strategy to select the tightest cluster. This approach not only extracts segments with high consistency but also assists in building a pseudo-labeled dataset for model training.

We make the following contributions. First, rather than analyzing surface-level receiver data, we focus on the ionosphere misclosure [74][75], a deeper-level PNT parameter, to develop new GNSS error detection methods. We categorize six types of noise from our dataset, each representing different potential errors. Next, we employ a temporal-spatial analysis approach that considers both time sequences and value distributions to analyze the noise segments. Then, to identify consistent error segments from large volumes of daily data, we apply clustering, using the ZFilter strategy to pinpoint segments that closely resemble our reference records. Meanwhile, we use deep learning models to extract and categorize features to automatically classify the noise types from this data. Lastly, we experiment with pseudo-labeled dataset generation from the clustering model to minimize the need for manually labeled data and improve the classification model with semi-supervised learning.

To demonstrate the deployment versatility of the DAIoTtalk platform, we simulate the GNSS error source deployment by creating a testbed within

DAIoTtalk. This showcases the testing and adjustment of the functions and models. The resulting system is named GNSS-EStalk.

The rest of this chapter is organized as follows. Related works are reviewed in Section 5.2. Section 5.3 describes the detection of noise in our dataset and introduces the AI-driven temporal-spatial analysis approach. The deployment of GNSS-EStalk is detailed in Section 5.4. Section 5.5 presents our evaluation results. The chapter concludes with discussions on future work in Section 5.6.

5.2 Related Works

Previous research (e.g.,[69], [70], [76], [77]) commonly considers surface-level data output by the receiver. The authors in [69] utilized elevation, S/N, and user speed as features in their machine-learning models to characterize multipath error distributions, [70] employed S/N for jamming detection, [76] analyzed signal strength and pseudorange residue for multipath detection, and [77] perform signal analysis for radio frequency interference (RFI). However, the surface-level data often provide an incomplete view compared to deeper-level data like [78], which employs the cross ambiguity function to detect GNSS spoofing. Here, we consider the ionosphere misclosure, which is a deeper-level parameter used to calculate the PNT result.

AI-based models (e.g., [69], [76], [77], [78]) are widely used in GNSS error analysis. [69] employed a neural network to classify multipath noise, while [76] utilized a support vector machine. [77] classified RFI using a convolutional neural network (CNN). Similarly, [78] used a CNN to identify spoofing signals and a Gaussian mixture model to cluster and summarize the number of signals. In this research, we experiment with clustering models using a ZFilter strategy to identify highly consistent noise segments and generate a pseudo-labeled dataset, and classification models to categorize the noise segments by different potential causes.

Temporal and spatial features are typically extracted in AI-based analysis. [79]

proposed a spatio-temporal attention network for video action recognition, incorporating pixel-related spatial attention and frame-related temporal attention. [80] introduced a revisiting spatio-temporal similarity model for traffic prediction, capturing traffic flow and volume as spatial dependencies and daily and weekly patterns as temporal dependencies. In the context of GNSS, [81] presented a spatial-temporal technique using an antenna array for GNSS anti-spoofing, featuring spatial channel separation and temporal cross-correlation peak monitoring. In our AI models, we consider both temporal sequences and spatial value distribution information.

5.3 Methodology

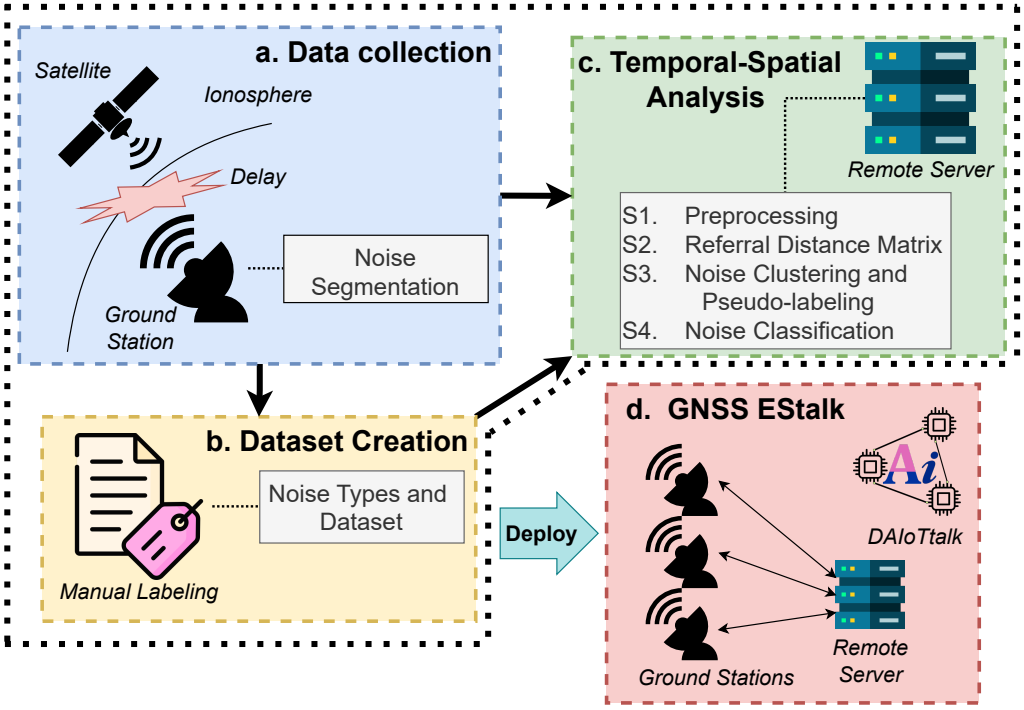


Figure 5.3.1: Overview of GNSS error source analyzing.

In this research, we extract noise segments from the ionosphere misclosure, which refers to the discrepancies between the estimations and observations of the regional ionosphere. Fig. 5.3.1 provides an overview of the process, encompassing data collection, dataset creation, and temporal-spatial analysis, and the GNSS-EStalk system.

In Fig. 5.3.1a, we collected data over four days from 53 ground stations and approximately 90 satellite units across the GPS, Galileo, and Beidou systems. As a satellite transmits GNSS signals to a ground station, the signals pass through the atmosphere, including the ionosphere, introducing various delays. The ground station predicts ionospheric conditions by considering factors such as weather and the signal path to mitigate these delays. Multiple factors, though, can affect these estimations and diverge from observed values that take longer to generate. Most discrepancies are due to white noise, while some are dismissed as errors in the modeling process. This work also includes the application of noise segmentation to identify and extract these errors.

In Fig. 5.3.1b, a dataset is created for analysis and validation. After noise segmentation, manual labeling is performed based on the error model. This research focuses on six types of errors potentially caused by factors such as multipath effects, jamming, and ionospheric irregularities. However, labeling these data is costly and requires domain expertise in GNSS error sources. Additionally, datasets may need to be tailored to specific regions due to variations in local ionospheric conditions, geographic features, and signal environments. Therefore, we expect the reduction of labeling costs through automation techniques by grouping highly consistent noise segments.

In Fig. 5.3.1c, a temporal-spatial analysis is performed on the remote server after noise segmentation and dataset creation. Fig. 5.3.2 illustrates the approach, which comprises three key components: a temporal pipeline for processing 1D sequential data, a spatial pipeline for handling 2D binary images, and a main pipeline that integrates both temporal and spatial information. Each pipeline follows four processing stages: Stage 1 (S1): preprocessing noise segments of varying sizes to standardize them into uniform input dimensions for the models; Stage 2 (S2): constructing a Referral Distance Matrix (RDM) to extract similarity features between segments; Stage 3 (S3): clustering segments to identify consistent noise patterns and generate a pseudo-dataset with minimal manual labeling; Stage 4 (S4): training a classifier to identify different noise types.

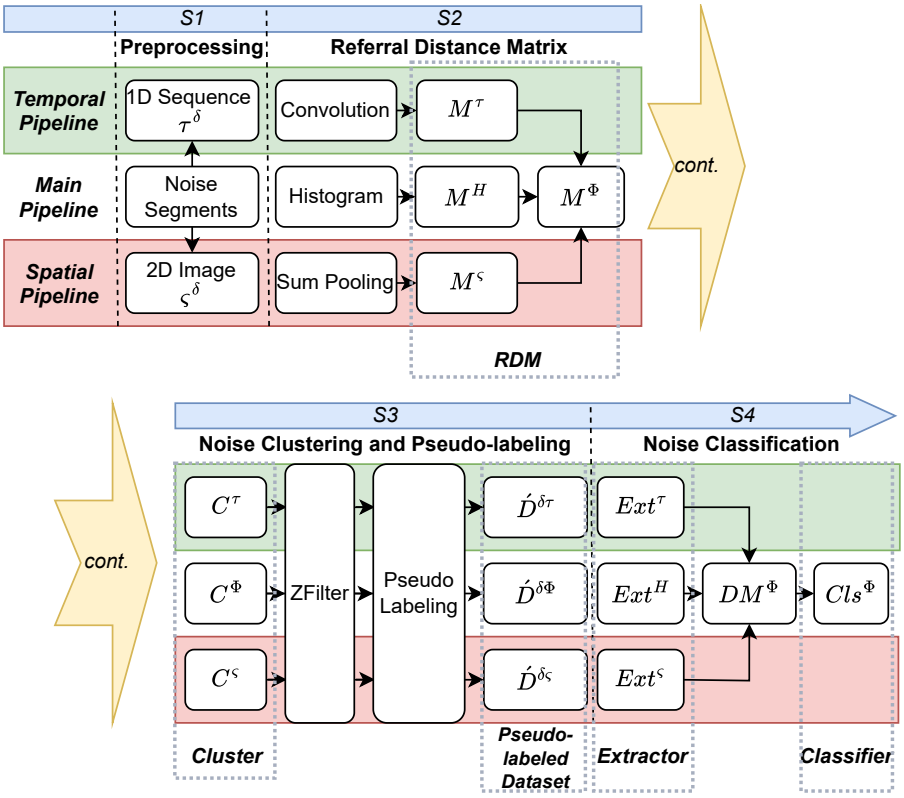


Figure 5.3.2: Process pipeline of the temporal-spatial approach.

In Fig. 5.3.1d, we deploy the analysis model on the DAIoTtalk platform. The platform establishes P2P connections between multiple ground stations and the remote analysis server. Each ground station transmits ionospheric misclosure data to the remote server for noise recognition. We name this system 'GNSS-EStalk' and will be described in Section 5.4.

In the following sections, we will detail each of these processes.

5.3.1 Noise Segmentation

Fig. 5.3.3 illustrates the noise segmentation process applied to the data. This process includes normalization, threshold selection, noise value masking, masking of noise windows, and merging noise windows and filtering. Let $\Lambda = \{\Lambda_{li}\} = \{(\delta_{li}, \theta_{li})\}$ represent an ionosphere misclosure with a sequence of values calculated from one GPS unit by a ground station for a single day, as shown in Fig. 5.3.3a, where $\delta = (\delta_{li}|\delta_{li} \in (-\infty, +\infty))$ denotes the misclosure values and $\theta = (\theta_{li}|\theta_{li} \in$

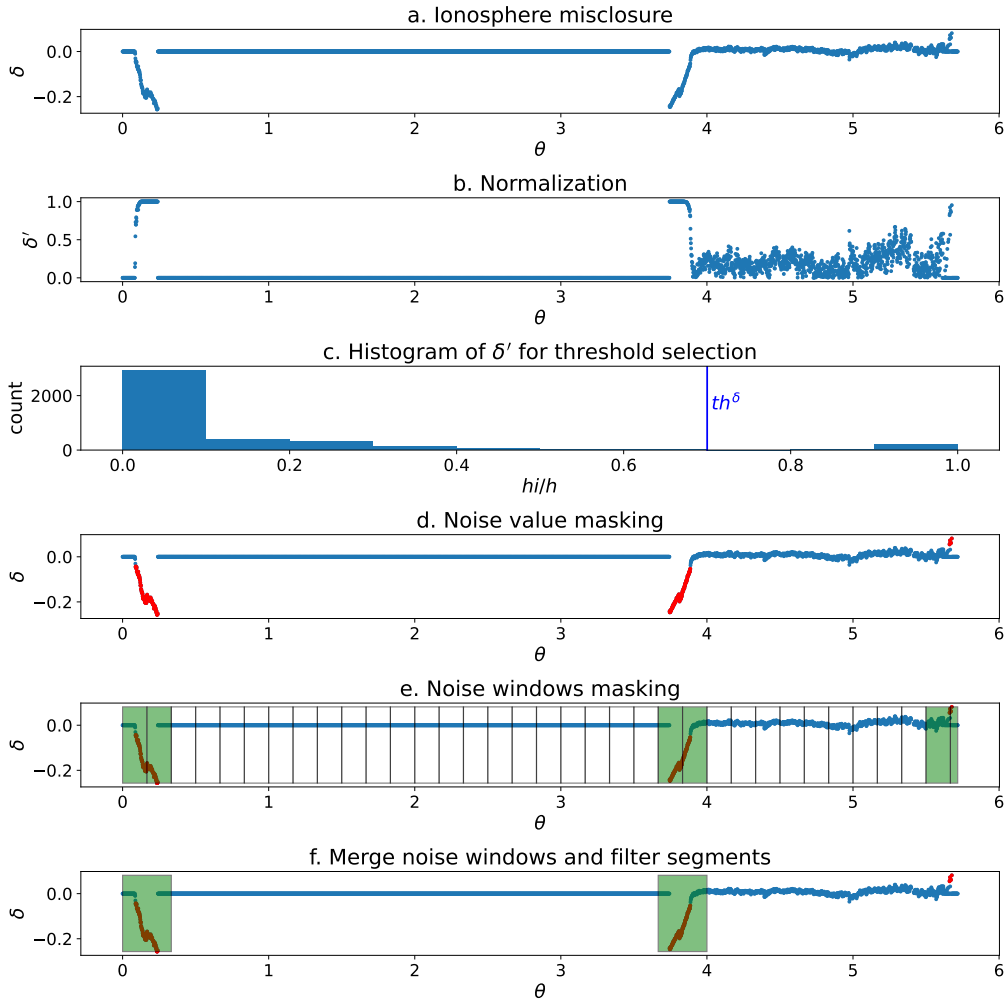


Figure 5.3.3: Noise Segmentation on ionosphere misclosure.

$[0, 24))$ represents the corresponding relative hourly timestamps. The set $I^{\Lambda} = \{li|li \in [0, |\Lambda|)\}$ is the index set associated with Λ. Our first task is to identify the noise values within δ and to segment these values accordingly into distinct segments.

Normalization is necessary due to the varying range of δ. 2 types of normalization we have applied, which are z-normalization and min-max normalization. Given a list of data $s = (s_i|i \in [0, |s|))$ with varying range, z-normalization with the cumulative distribution function (CDF,Φ) used to standardize s into normal distribution space:

$$Z(s) = \Phi\left(\frac{s_i - \mu_s}{\sigma_s}\right), \tag{5.3.1}$$

where μ_s and σ_s is the mean and standard deviation of s. On the other hand, the min-max normalization can transform s to a rescaled sequence $\widetilde{sq}$ with a range of $[0, 1]$:

$$\widetilde{s} = \frac{s - min(s)}{max(s) - min(s)} \tag{5.3.2}$$

Assuming the estimation and observation are ideally close, we expect the misclosure to approach zero as the one in Fig. 5.3.3a. Therefore, we set $\mu_s = 0$ and calculate σ_s in Eq. (5.3.1) for noise detection. By combining Eq. (5.3.1) and Eq. (5.3.2), we normalize the absolute value of δ as δ' in Fig. 5.3.3b:

$$\delta' = \widetilde{Z(abs(\delta))} \tag{5.3.3}$$

Next, we construct an h-bin histogram $H^{|\delta|} = \{H_{hi}^{|\delta|} | hi \in [0, h)\}$ to observe the distribution of δ', where each $H_{hi}^{|\delta|} \in [0, 1]$. Here, we choose $h = 10$:

$$H^{|\delta|} = Histogram(\delta', h) \tag{5.3.4}$$

In the ideal case, the highest peak $hi^{max} = argmax(H^{|\delta|})$ will be near zero as in Fig. 5.3.3c. We apply valley detection on the histogram to determine the local noise filtering threshold $th^{\delta} \in [0, 1]$:

$$\begin{aligned} th^{\delta} &= \frac{hi^{min}}{h}, where \\ hi^{min} &= argmin(\{H_{hi}^{|\delta|} | hi \in [hi^{max}, h)\}) \end{aligned} \tag{5.3.5}$$

As the example in Fig. 5.3.3c, the blue line represents th^{δ}, with values on the right side indicating noise. Using th^{δ}, we initialize an mask array $\psi^{\delta} = (\psi_{li}^{\delta} | \psi_{li}^{\delta} \in \{0, 1\})$ to indicate whether each Λ_{li} is classified as noise.

$$\psi_{li}^{\delta} = \begin{cases} 1, & \text{if } abs(\delta_{li}) > TH^{\delta}, \delta'_{li} > th^{\delta}. \\ 0, & \text{otherwise.} \end{cases} \tag{5.3.6}$$

The result, shown in Fig. 5.3.3d, is obtained after applying ψ^{δ}, with red dots representing the noise values.

To segment noise from Λ, we first preform chronological splits to divide Λ into multiple subsets $\lambda_{mi} \subset \Lambda$, each with a corresponding index set $I^{\lambda}_{mi} \subset I^{\Lambda}$ and covering a fixed m-hour window, where $mi \in [0, |\Lambda|)$. Here, we set $m = 0.1$ hours (10 minutes). A λ_{mi} is classified as a noise subset if any $\Lambda_{li} \in \lambda_{mi}$ is determined to be noise by Eq. (5.3.6). We then create a window mask $\psi^{\lambda} = (\psi^{\lambda}_{mi} | \psi^{\lambda}_{mi} \in \{0, 1\})$:

$$\psi^{\lambda}_{mi} = \begin{cases} 1, & \text{if } \vee \psi^{\delta}_{li \in I^{\lambda}_{mi}} = 1 \\ 0, & \text{otherwise.} \end{cases} \tag{5.3.7}$$

Fig. 5.3.3e shows the split windows with the applied masking in green.

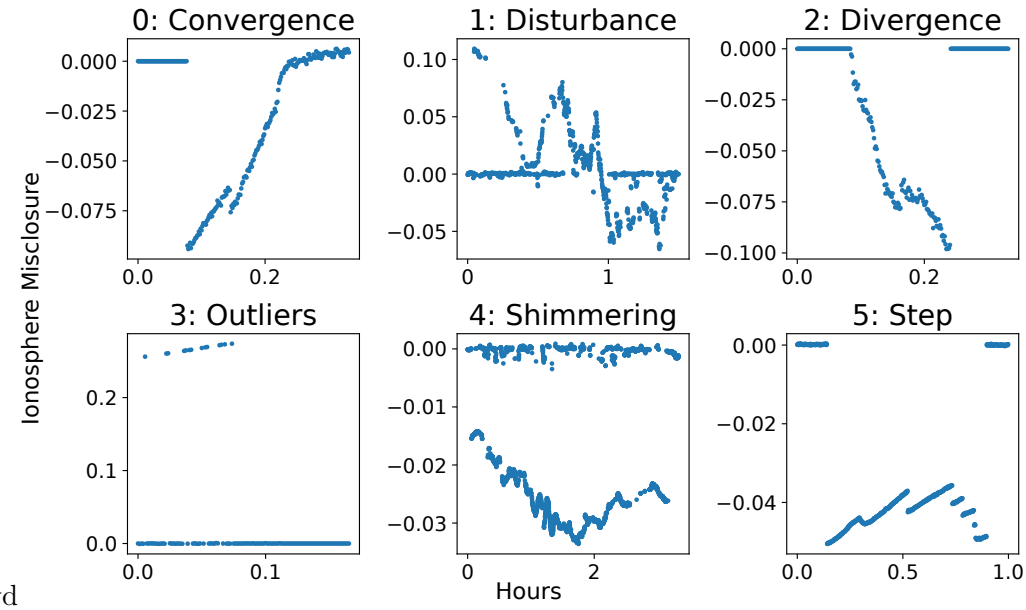
Algorithm 1 Noise segmentation using ψ^{λ}

procedure WINDOWSMERGING(ψ^{λ}, X^{δ})
  $st \leftarrow$ None ▷ Start index of the segment
  for $i \leftarrow 0$ **to** $|\psi^{\lambda}|$ **do** ▷ Iterate through ψ^{λ}
    if $\psi^{\lambda}_{i} = 1$ **and** (i $= 0$ **or** $\psi^{\lambda}_{i-1} = 0$) **then**
      $st \leftarrow i$ ▷ Start index of the segment
    else if $\psi^{\lambda}_{i} = 0$ **and** $\psi^{\lambda}_{i-1} = 1$ **then**
      $x \leftarrow \cup_{mi \in [st,i)} \lambda_{mi}[\delta]$ ▷ Extract the segment
      if $max(abs(x)) > TH^{\delta}$ **then** ▷ Filter by global threshold
        $X^{\delta} \leftarrow X^{\delta} \cup x$ ▷ Add the segment to dataset
      end if
    end if
  end for
  if $\psi^{\lambda}_{|\psi^{\lambda}|-1} = 1$ **then** ▷ Handle the last segment if it ends with 1
    $i \leftarrow |\psi^{\lambda}| - 1$
    $x \leftarrow \cup_{mi \in [st,i)} \lambda_{mi}[\delta]$
    if $abs(x) > TH^{\delta}$ **then**
      $X^{\delta} \leftarrow X^{\delta} \cup x$
    end if
  end if
  return X^{δ} ▷ Return the set of segments
end procedure

Finally, we concatenate the nearby noisy λ_{mi} to create noise segments with their disclosure δ, denoted as $\lambda_{mi}[\delta] \subset \delta$, using Algorithm 1. In Algorithm 1, we take ψ^{λ} as input and detect consecutive instances where $\psi^{\lambda}_{i} = 1$, indicating noisy

values, to merge them into cohesive noise segments. The segments from each satellite and station are then accumulated to form the noise segment set $X^{\delta} = \{X^{\delta}_{di} | di \in [0, |X^{\delta}|)\}$, where each segment is a sequence defined as $X^{\delta}_{di} = (X^{\delta}_{di}[dj] | dj \in [0, |X^{\delta}_{di}|))$, with each element $X^{\delta}_{di}[dj] \in (-\infty, +\infty)$. $X^{\delta} = \{\}$ during initialization. Notice that a preset global threshold TH^{δ} is set to filter the segment if the noise value does not exceed. In this research, we set $TH^{\delta} = 0.2$ based on recommendations from the data provider. The final segmentation result is shown in Fig. 5.3.3f, with some of the noise segments filtered by TH^{δ}.

5.3.2 Noise Types and Dataset



wd

Figure 5.3.4: The noise types in the GNSS error source dataset.

We manually identify six types of noise, as illustrated in Fig. 5.3.4, after the extraction of X^{δ}. The scale and magnitude of the noise segments vary, with each type caused by different physical factors. Below, we present some potential examples:

- Convergence occurs as a result of model recalculation when tracking is lost or when clock or ephemeris errors are detected.

- Disturbance caused by interference from multiple systems or jamming by other equipment.
- Divergence arises from a mismatch between the error model and the observation.
- Outliers may result from incorrect carrier-phase ambiguity fixes.
- Shimmering can occur as a consequence of repeated carrier-phase ambiguity fixes, especially in the presence of atmospheric delays.
- Steps caused by cycle slips due to signal delays and distortions from ionospheric irregularities.

Table 5.1: Class Sizes in the GNSS Error Source Dataset

	Labeled L^δ						**Unlabeled U^δ**	**Total**
Type (Y^δ)	**0**	**1**	**2**	**3**	**4**	**5**		
Size	140	540	504	526	569	515	2114	**4906**

Denote our dataset $D^\delta = \{D^\delta_{di}\} = \{(X^\delta_{di}, Y^\delta_{di})\}$ and the index set $I^\delta = \{di | di \in [0, |D|\}$. The dataset D^δ contains the noise segments $X^\delta = \{X^\delta_{di}\}$ and their corresponding labels $Y^\delta = \{Y^\delta_{di}\}$. Each segment X^δ_{di} consists of ordered real-valued observations that may have different lengths. We manually label a subset of the segments by their indices $I^{L\delta} \subset I$ to create a labeled dataset $L^\delta = \{D^\delta_{di} | di \in I^{L\delta}\}$, where the labels $Y^\delta_{di} \in [0, p)$ if $di \in I^{L\delta}$, here $p = 6$ as shown in Fig. 5.3.4. The remaining indices $I^{U\delta} = I \backslash I^{L\delta}$ form an unlabeled dataset $U^\delta = \{D^\delta_{di} | di \in I^{U\delta}\}$, with labels $Y^\delta_{di} = -1$ if $di \in I^{U\delta}$. Table 5.1 lists the sizes of each class.

5.3.3 Preprocessing (S1)

Since the range and length of each noise segment X^δ_{di} can vary, it is necessary to normalize the range and standardize the length of the segments to ensure uniform contribution from each segment and maintain consistent characteristics across them. Fig. 5.3.5 is the example to resize a sequence sq into a uniform length sl.

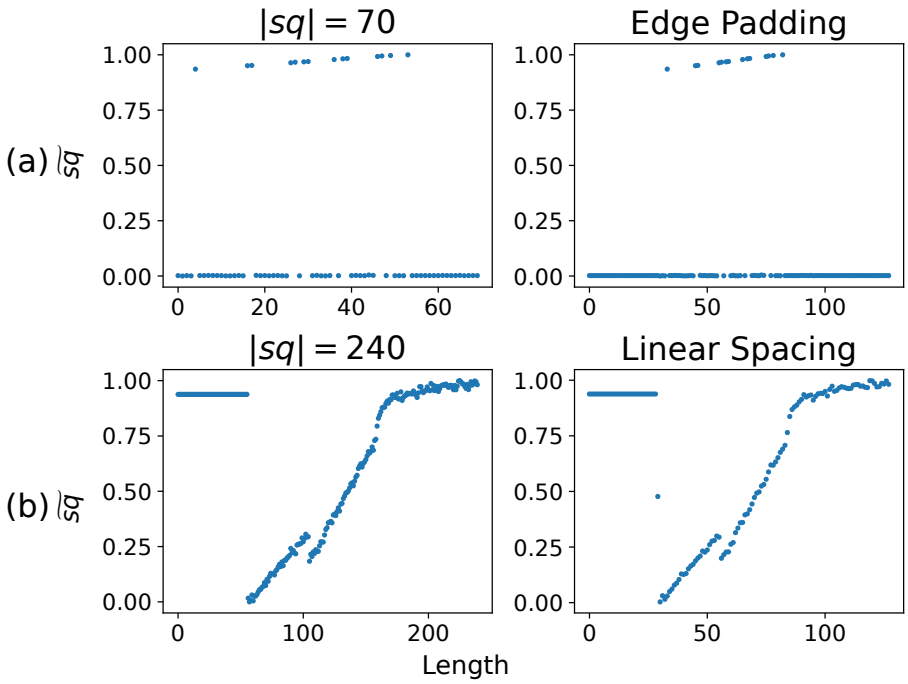


Figure 5.3.5: Example of a unified function to standardize a normalized sequence to a length of 128.

In Fig. 5.3.5a, if its size $|sq|$ is shorter then sl, we apply the edge-padding process:

$$\begin{aligned} EP(sq, sl) &= (sq[0])|_{1\times lp} \cup sq \cup (sq[|sq|-1])|_{1\times rp} \\ \text{where } lp &= \lfloor \frac{sl-|sq|}{2} \rfloor \text{ and } rp = \lceil \frac{sl-|sq|}{2} \rceil \end{aligned} \tag{5.3.8}$$

Otherwise, as shown in Fig. 5.3.5b, we use the linear spacing process to down-sample sq by taking the averages of the corresponding subset:

$$\begin{aligned} LS(sq, sl)[LSa] &= \overline{sq[mask]}, \text{where } 0 \leq LSa < sl, \\ \text{and } mask &= (\lfloor \frac{\{[0, |sq|)\}}{|sq|} * sl \rfloor == LSa) \end{aligned} \tag{5.3.9}$$

By combining Eq. (5.3.2),Eq. (5.3.8), and Eq. (5.3.9), we define the following unified function UF for a sq:

$$UF(sq, sl) = \begin{cases} EP(\widetilde{sq}, sl), & \text{if } |sq| < sl. \\ LS(\widetilde{sq}, sl), & \text{otherwise.} \end{cases} \tag{5.3.10}$$

Next, we divide the process into temporal, spatial pipelines, and main pipelines:

Temporal Pipeline

To standardize the noise segments of varying sizes and values, we apply Eq. (5.3.10) to uniform X_{di}^{δ} into sequential data $\tau_{di}^{\delta} = \{\tau_{di}^{\delta}[dj]\}^{1\times 128}$ where $\tau_{di}^{\delta}[dj]|_{dj\in[0,128)} \in [0,1]$:

$$\tau_{di}^{\delta} = UF(X_{di}^{\delta}, 128) \tag{5.3.11}$$

Spatial Pipeline

We further transform τ_{di}^{δ} into a 2D space to generate a binary image $\varsigma_{di}^{\delta} = \{\varsigma_{di}^{\delta}[h,w]\}^{128\times 128}$ where $\varsigma_{di}^{\delta}[h,w]|_{h\in[0,128),w\in[0,128)} \in \{0,1\}$, enabling the extraction of distributional information:

$$\varsigma_{di}^{\delta}[h,w] = \begin{cases} 1, & \text{if } h = \lfloor \tau_{di}^{\delta}[w] \times 127 \rfloor. \\ 0, & \text{otherwise.} \end{cases} \tag{5.3.12}$$

Main Pipeline

By applying Eq. (5.3.2) to normalize each X_i^{δ} into $\widetilde{X^{\delta}}_i$, a normalized segment set $\widetilde{X^{\delta}} = \{\widetilde{X^{\delta}}_i\}$ is obtained.

After preprocessing is completed, the min-max normalized segments $\widetilde{X^{\delta}}$, the temporal segments τ^{δ}, and the spatial images ς^{δ} are ready for the next stage.

5.3.4 Referral Distance Matrix (S2)

A global distance matrix is generated to compare all the distances of the segment pairs as a similarity feature during feature extraction. However, calculating all distances becomes inefficient when the dataset is large. We randomly select a

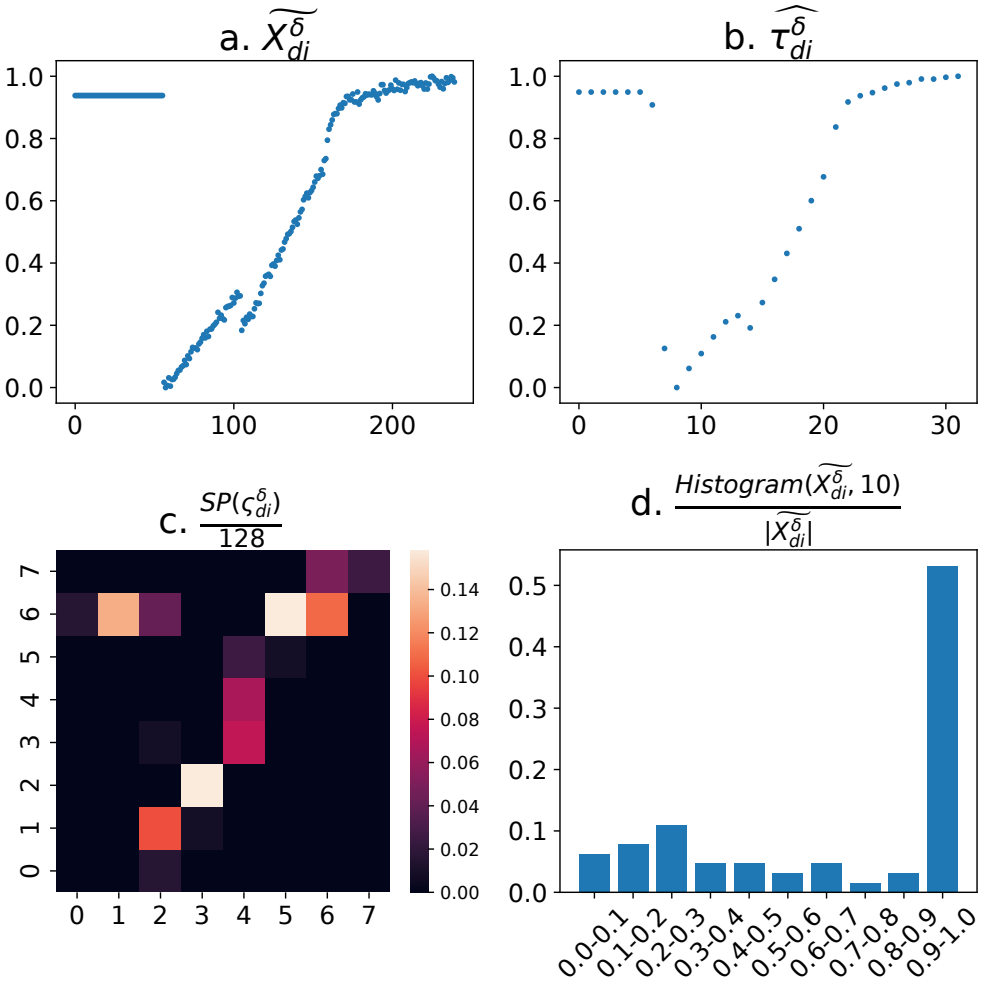


Figure 5.3.6: Example of transformation of a noise segment.

subset of segments from the labeled dataset as references $R^{\delta} \subseteq L^{\delta}$. The RDM $M|_{|D|\times|R^{\delta}|}$ can then be computed:

$$M(X^{\delta}, Dist)[di, r] = Dist(X^{\delta}_{di}, R^{\delta}_{r}) \tag{5.3.13}$$

where R^{δ}_{r} is a reference segment, $r \in [0, |R^{\delta}|)$, and $Dist \in \{Eucl, DDTW\}$ is the metric used to compare distances. Depending on the pipelines, we apply either Euclidean distance ($Eucl$) or derivative dynamic time warping ($DDTW$)[82].

When computing the distance matrix, each pipeline transforms the segments to reduce the vector dimensions. Fig. 5.3.6 illustrates an example of this transformation applied to the sequence shown in Fig. 5.3.6a, with the detailed

process outlined below.

Temporal Pipeline

We use τ^{δ} as the input. We apply convolution with a kernel $\nu = \{1\}|_{1\times 5}$ to smooth the sequence τ^{δ}_{di} and further uniform it to $\widehat{\tau^{\delta}_{di}}|_{1\times 32}$ by Eq. (5.3.10) as shown in Fig. 5.3.6b:

$$\widehat{\tau^{\delta}_{di}} = UF((\tau^{\delta}_{di} * \nu)_{|\tau^{\delta}_{di}|}, 32). \tag{5.3.14}$$

After that, we compute the temporal RDM $M^{\tau} = M(\widehat{\tau^{\delta}}, DDTW)$ via Eq. (5.3.13), using DDTW distance.

Spatial Pipeline

We use ς^{δ} as the input. We apply sum-pooling (SP) with a kernel of 16×16 to generate a heat map of the binary image ς^{δ}_{di} as shown in Fig. 5.3.6c, then flatten into $\widehat{\varsigma^{\delta}_{di}}|_{1\times 64}$:

$$\widehat{\varsigma^{\delta}_{di}} = Flatten\left(\frac{SP(\varsigma^{\delta}_{di})}{128}\right). \tag{5.3.15}$$

After that, we compute the spatial RDM $M^{\varsigma} = M(\widehat{\varsigma^{\delta}}, Eucl)$ via Eq. (5.3.13), using Euclidean distance.

Main Pipeline

We directly extract the h-bin histogram feature from $\widetilde{X^{\delta}}$ as shown in Fig. 5.3.6d and compute the difference in value distribution using $Eucl$. Again, we set $h = 10$.

$$M^{H} = M\left(\left\{\frac{Histogram(\widetilde{X^{\delta}_{di}}, h)}{|\widetilde{X^{\delta}_{di}}|}\right\}, Eucl\right). \tag{5.3.16}$$

This is then concatenated with M^{τ} and M^{ς} to obtain the hybrid RDM:

$$M^{\Phi} = [M^{H}, M^{\varsigma}, M^{\tau}] \tag{5.3.17}$$

The temporal RDM M^{τ}, the spatial M^{ς}, the histogram RDM M^{H}, and the hybrid RDM M^{Φ} are used as inputs for the remaining stage.

5.3.5 Noise Clustering and Pseudo-labeling (S3)

It is necessary to identify consistent noise to ensure accuracy and precision in PNT. However, the daily generation of large amounts of unlabeled data complicates the analysis process. By giving a small set of manually labeled examples, this data can be compared using the clustering approach to identify similar noise patterns. Additionally, pseudo-labels could be assigned to the unlabeled data, which would facilitate further training through semi-supervised learning.

Clustering is performed to group similar segments into clusters $C^{\delta} = \{C^{\delta}_{dk} | dk \in [0, |C^{\delta}|)\}$ along with the corresponding index set $I^{\delta}_{dk} \subset I^{\delta}$. Here, $D^{\delta}_{di} \in C^{\delta}_{dk}$ if a segment X^{δ}_{di} belongs to the dk-th cluster, which implies that $di \in I^{\delta}_{dk}$ as well. Note that each segment belongs to only one cluster but some segments may not fit into any cluster (i.e. $dk < 0$) and are excluded from consideration. Moreover, the number of clusters $|C^{\delta}|$ should be sufficiently large to ensure that the segments within each cluster are as similar as possible. M^{τ}, M^{ς}, and M^{Φ} are the inputs to the clustering algorithm, generating the temporal cluster C^{τ}, spatial cluster C^{ς}, and hybrid cluster C^{Φ}, respectively.

To select the clusters with higher consistency segments, we apply ZFilter to identify more confident clusters. Firstly, the average intra-cluster distance $AICD_{dk}$ is calculated to assess the tightness within C^{δ}_{dk} using the corresponding

RDM features $M[di]$:

$$AICD_{dk} = \frac{\sum_{di \in I^{\delta}_{dk}} Eucl(M[di], \overline{M[I^{\delta}_{dk}]})}{|I^{\delta}_{dk}|} \tag{5.3.18}$$

where $|I^{\delta}_{dk}|$ and $\overline{M[I^{\delta}_{dk}]}$ denote the size and centroid of C^{δ}_{dk}, respectively. Following this, a z-normalized confidence score $Z^{\delta C}_{dk} \in [0, 1]$ is computed based on $AICD_{dk}$ for each C_k, along with the overall mean μ and standard deviation σ. The score is normalized using the cumulative distribution function:

$$Z^{\delta C}_{dk} = \Phi\left(\frac{AICD_{dk} - \mu}{\sigma}\right) \tag{5.3.19}$$

A smaller $Z^{\delta C}_{dk}$ indicates that the segments within the cluster C_k are more similar. Therefore, we can define a threshold $th^{\delta C} \in [0, 1]$ to filter the clusters.

A pseudo-labeled dataset $\acute{D}^{\delta} \subset U^{\delta}$, can also be generated from unlabeled dataset U^{δ} using the ZFilter strategy applied to the labeled dataset L^{δ}, thereby increasing the sample size during classification model training. For each cluster C^{δ}_{dk}, we compute the label score $LS_{dk}|_{1\times 6}$, which represents the weightings for each noise type:

$$LS_{dk} = \begin{cases} \sum_{di \in (I^{\delta}_{dk} \cap I^{L\delta})} \ddot{Y}^{\delta}_{di}, & \text{if } Z^{\delta C}_{dk} \leq th^{\delta C} \\ \{0\}|_{1\times 6}, & \text{otherwise.} \end{cases} \tag{5.3.20}$$

where $\ddot{Y}^{\delta}_{di}$ is the one-hot encoded Y^{δ}_{di}. Next, we can create the $\acute{D}^{\delta}$:

$$\begin{aligned} \acute{D}^{\delta} = \bigcup_{dk \in [0, |C^{\delta}|)} \{D^{\delta}_{di} | di \in (I^{\delta}_{dk} \cap I^{U\delta})\}, \\ \text{if } Z^{\delta C}_{dk} \leq th^{\delta C} \text{ and } \sum LS_{dk} > 0 \end{aligned} \tag{5.3.21}$$

Label smoothing will also be applied to adjust the weighting of the pseudo-labels,

helping to prevent the model from becoming overly confident in the predictions:

$$\ddot{Y}_{di}^{\delta} = \begin{cases} \ddot{Y}_{di}^{\delta}, & \text{if } di \in I^{L\delta} \\ (1-\alpha) * \frac{LS_{dk}}{\sum LS_{dk}} + \frac{\alpha}{6} & \text{if } D_{di}^{\delta} \in \acute{D}^{\delta}, di \in I_{dk}^{\delta} \\ \{0\}|_{1\times 6}, & \text{otherwise.} \end{cases} \tag{5.3.22}$$

where α is a hyperparameter that determines the amount of smoothing.

Afterward, we can generate the temporal pseudo-labeled dataset $\acute{D}^{\delta\tau}$, the spatial pseudo-labeled dataset $\acute{D}^{\delta\varsigma}$, and the hybrid pseudo-labeled dataset $\acute{D}^{\delta\Phi}$, using C^{τ}, C^{ς}, and C^{Φ}, respectively.

5.3.6 Noise Classification (S4)

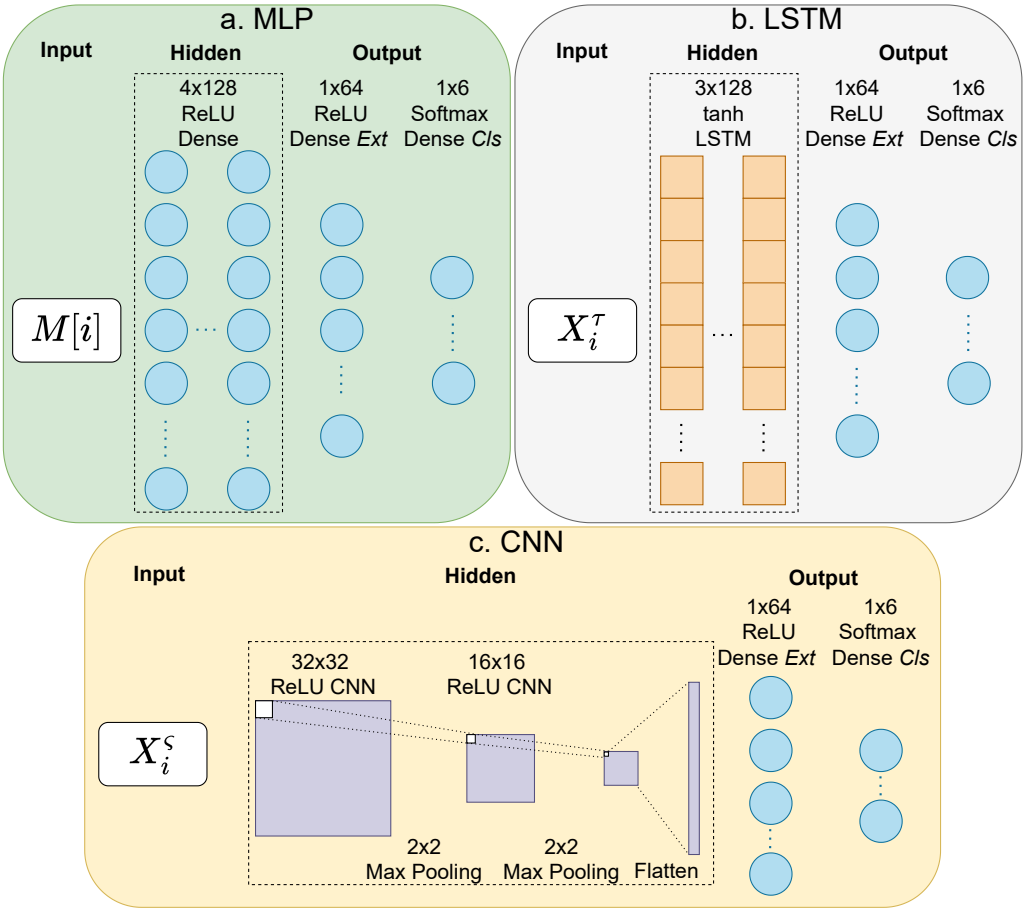


Figure 5.3.7: The baseline classification models.

To classify noise segments, we experimented with various deep-learning models. Fig. 5.3.7 illustrates the architectures of our baseline models, including the multilayer perceptron (MLP) in Fig. 5.3.7a, long short-term memory (LSTM) in Fig. 5.3.7b, and CNN in Fig. 5.3.7c. They are designed to process RDM, sequences, and binary images, respectively. While each model received different input types and had distinct hidden layer configurations, they all shared a common output structure. The detailed design of each model is outlined below:

MLP: The MLP is a basic type of neural network typically designed to take tabular data as input. In Fig. 5.3.7a, a $M^{\tau}[di]$ is taken as input and processed through four dense hidden layers, each containing 128 neurons. We use the rectified linear unit (ReLU) activation function for all hidden layers. The output consists of a shared structure represented as a list of deep features $Ext|_{1\times 64}$ with ReLU activation, followed by a final dense layer with softmax activation to generate classification probabilities, denoted as $Cls|_{1\times 6}$.

LSTM: The LSTM is a type of recurrent neural network (RNN) designed to handle sequential data and time-dependent patterns. In Fig. 5.3.7b, a τ^{δ}_{di} is taken as input and processed through 3 LSTM hidden layers, each containing 128 neurons. We use the tanh activation function for all hidden layers. Again, The output consists of Ext and the Cls.

CNN: The CNN is primarily used for data that has a grid-like structure. In Fig. 5.3.7c, a ς^{δ}_{di} is taken as input. The hidden layers first process it by a 32×32 ReLU CNN layer. Next, the data is resized using a $2x2$ max pooling layer and then processed by a 16×16 ReLU CNN layer. Finally, it is resized again by a $2x2$ max pooling layer and flattened. Similarly, the output consists of the Ext and the Cls.

Each pipeline utilizes different baseline models. In the temporal pipeline, we input M^{τ} into the MLP or τ^{δ} into the LSTM, the procedures are referred to as 'TMLP' or 'TLSTM', respectively. In the spatial pipeline, we input M^{ς} into the MLP or ς^{δ} into the CNN, referred to as 'SMLP' or 'SCNN', respectively. In the main

pipeline, we input M^H into the MLP, named 'HMLP'.

To perform temporal-spatial classification, we use the deep features output from the baseline models: Ext^H from HMLP, Ext^τ from TMLP or TLSTM, and Ext^ς from SMLP or SCNN. These features are combined to generate a hybrid deep RDM, denoted as $DM^\Phi|_{|D|\times|R^\delta|}$:

$$\begin{aligned} DM^\Phi &= M(X^\Phi, Eucl), \\ \text{where } X^\Phi &= [Ext^H, Ext^\tau, Ext^\varsigma] \end{aligned} \tag{5.3.23}$$

Finally, we input DM^Φ into the MLP model to train a hybrid classifier, denoted as Cls^Φ as shown in Fig. 5.3.2. The hybrid classifier leverages latent representations from baseline models across different domains to enhance prediction performance. Combining the models in Fig. 5.3.7, four Cls^Φ are trained: 'TMLP_SMLP', 'TLSTM_SMLP', 'TLSTM_SCNN', and 'TMLP_SCNN'.

5.4 Deployment of GNSS-EStalk

After developing the temporal-spatial analysis approach, we deploy the models onto the DAIoTtalk platform, as illustrated in Fig. 5.4.1. Fig. 5.4.1a depicts the data flow between each component. When multiple ground stations generate regional ionospheric misclosure data, noise segments are extracted and then collected by the remote server node, named 'CHC Error Model'. The remote server converts the segments into features and then passes the data to the classification model node, referred to as 'GNSS Error Classification,' for error recognition. Once the process is complete, the error results are returned to the remote server node and subsequently sent back to the corresponding ground station.

The communication is divided into two projects: Project A, named 'GNSS Error Source', as shown in Fig. 5.4.1b, and Project B, named 'GNSS Error Recognition', as shown Fig. 5.4.1c. Project A manages communication between

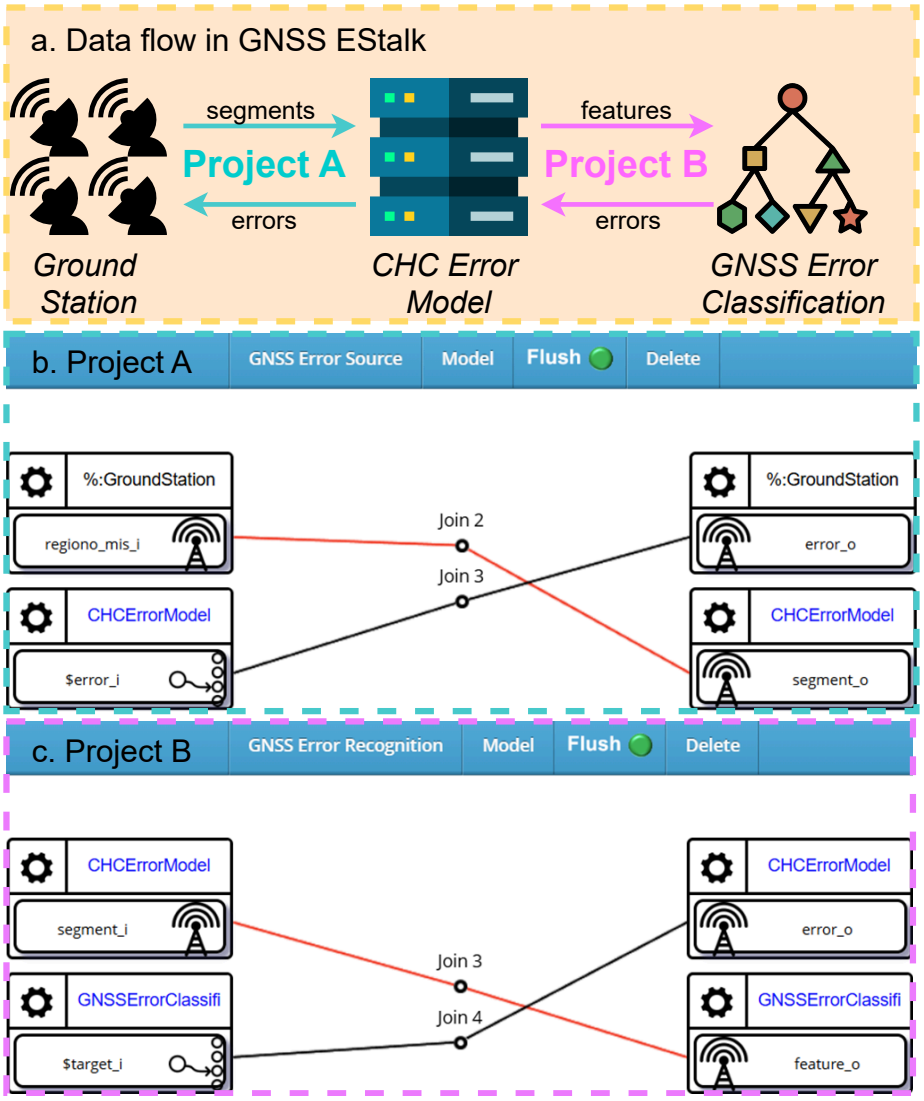


Figure 5.4.1: Deployment of GNSSEStalk on 2 DAIoTtalk project.

multiple ground station nodes and the remote server node, embedding the noise segmentation algorithm. Meanwhile, Project B facilitates communication between the remote server node and the classification node, utilizing the noise feature extraction algorithm and selection of models. Detailed descriptions of the deployments and configurations for the two projects are provided in the following sections.

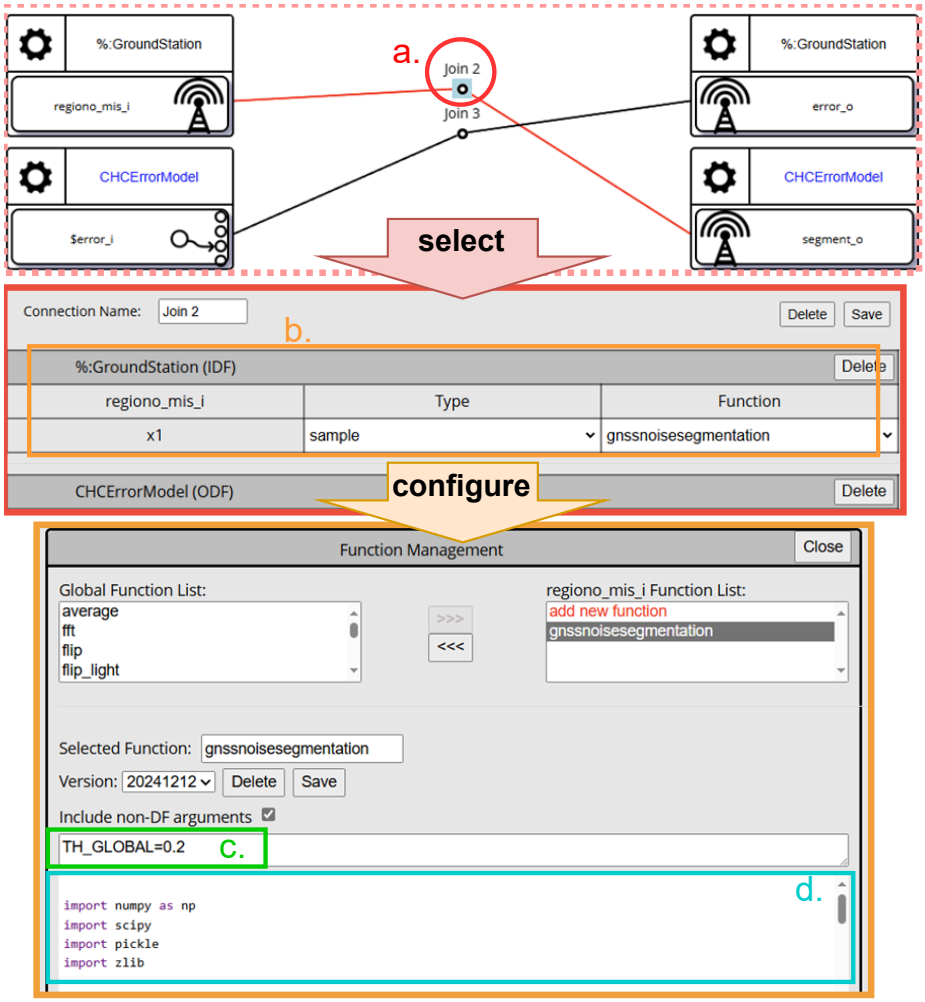


Figure 5.4.2: Deployment of noise segmentation algorithm on GNSS-EStalk.

5.4.1 Project A: Deployment of noise segmentation algorithm

Fig. 5.4.2 illustrates the deployment of the noise segmentation algorithm. This project involves two device models: *GroundStation* for multi-domain ground station nodes and *CHCErrorModel* for one remote server node. The *GroundStation* publishes a broadcast IDF, *regiono_mis_i*, which outputs regional ionospheric misclosure data, and subscribes to an ODF, *error_o*, which receives recognized errors in the data. The *CHCErrorModel* publishes a unicast IDF, *error_i*, representing the recognized errors, and subscribes to an ODF, *segment_o*, which receives the noise segments. There are two join connections: one links *regiono_mis_i* of *GroundStation* with *segment_o* of *CHCErrorModel* to transfer the data to be processed, and the other links

$error_i$ of $CHCErrorModel$ with $error\_o$ of $GroundStation$ to transfer the processed results.

In Fig. 5.4.2a, $regiono\_mis\_i$ publishes the misclosure data, while $segment_o$ subscribes to the noise segments. The segmentation function is deployed to extract noise segments from the misclosure data. By selecting this join connection, a configuration UI, as shown in Fig. 5.4.2b, is opened. Here, the function *gnssnoisesegmentation* is implemented, allowing the adjustment of the argument TH_GLOBAL, as shown in Fig. 5.4.2c. This argument represents the global threshold TH^{δ} in Algorithm 1. Finally, the Python implementation of the noise segmentation algorithm is placed in Fig. 5.4.2d. The algorithm is subsequently saved into the DAIoTtalk Agent's database.

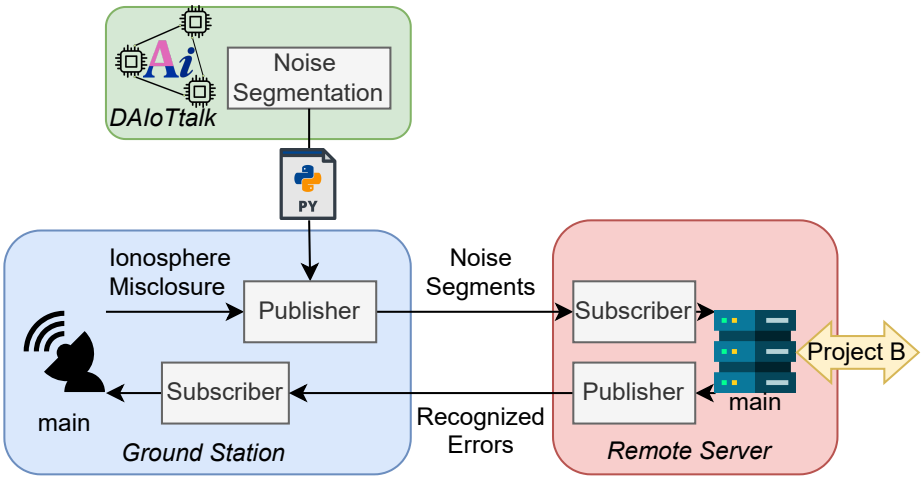


Figure 5.4.3: Data flow from a ground station to the remote analysis server in GNSS-EStalk.

Fig. 5.4.3 details the data flow of the deployment. The noise segmentation function stored in the DAIoTtalk database is sent to the ground station node, where it is saved as a Python module and imported by the Publisher of AGAPI. When the main thread publishes the ionospheric misclosure data, the Publisher processes the data using the module to extract the noise segments. These segments are then received by the remote server node Subscriber handler of AGAPI and routed back to the main thread for further processing in Project B. Once the remote server receives the recognized errors, it returns them to the corresponding ground station. Note that if no noise is detected during publication, the message will be dropped

by the Publisher at the ground station.

5.4.2 Project B: Deployment of noise classification models

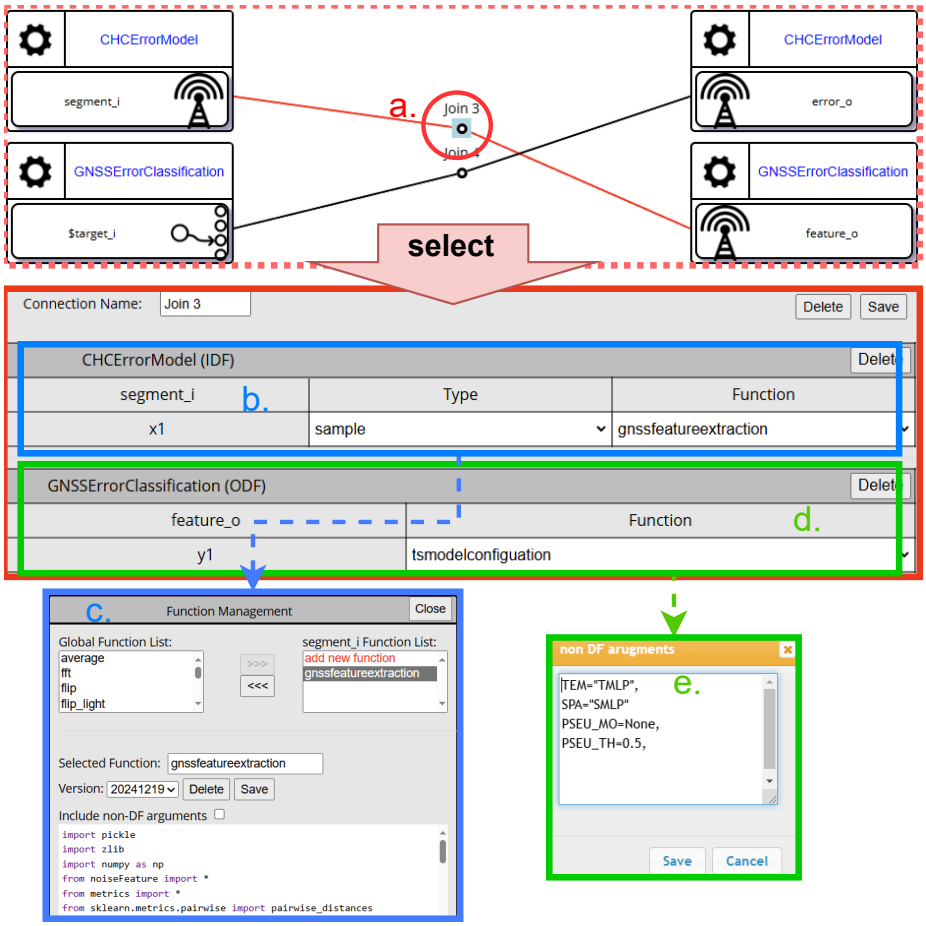


Figure 5.4.4: Deployment of noise feature extraction and model configuration on GNSS-EStalk.

Fig. 5.4.4 and Fig. 5.4.5 illustrate the deployment setup in Project B. Two device models are involved: $CHCErrorModel$ and $GNSSErrorClassification$. The $CHCErrorModel$ utilizes the same remote server nodes as in Project A, but with different IDF and ODF topics. The IDF, $segment\_i$, bridges the error segments received from Project A, while the ODF, $error\_o$, forwards the recognized errors back to Project A. The $GNSSErrorClassification$ is a classification node with the IDF $feature\_o$ and a unicast ODF $target\_i$. There are two join connections: one links $segment\_i$ of $CHCErrorModel$ with $feature\_o$ of

$GNSSErrorClassification$ to transfer the segment to be processed, and the other links $target\_i$ of $GNSSErrorClassification$ with $error\_o$ of $CHCErrorModel$ to transfer the processed results.

In Fig. 5.4.4a, the $segment\_i$ sends segments, while the $feature_o$ receives feature instances. To extract features from segments, we deploy the function $gnssfeatureextraction$ on the IDF side, as shown in Fig. 5.4.4b. Given that $\check{di} \in [0, \infty]$ represents the unique segment ID, the Python module $gnssfeatureextraction$ in Fig. 5.4.4c generates feature instances along with the RDM M^{Φ} derived from the segments, as introduced in Section 5.3.3 and Eq. (5.3.17), respectively. These are denoted as $\check{X}^{\delta}_{\check{di}}$:

$$\check{X}^{\delta}_{\check{di}} = (\widetilde{X^{\delta}_{\check{di}}}, \tau^{\delta}_{\check{di}}, \varsigma^{\delta}_{\check{di}}, M^{\Phi}_{\check{di}}) \tag{5.4.1}$$

Meanwhile, we deploy the $tsmodelconfiguration$ on the ODF side, as shown in Fig. 5.4.4d, enabling the selection of a pre-trained model. There are four adjustable arguments, as shown in Fig. 5.4.4e: TEM, SPA, $PSEU\_MO$, and $PSEU\_TH$. The

$$TEM \in \{\, TMLP, TLSTM \,\}, \text{and} \tag{5.4.2}$$

$$SPA \in \{\, SMLP, SCNN \,\} \tag{5.4.3}$$

parameters allow the temporal and spatial model selection, respectively, as described in Section 5.3.6. The

$$PSEU\_MO \in \{\, None, Temporal, Spatial, Hybrid \,\}, \text{and} \tag{5.4.4}$$

$$PSEU\_TH \in \{\, 0.1, 0.3, 0.5, 0.7, 0.9 \,\} \tag{5.4.5}$$

parameters determine the pseudo-labelling method and the threshold $th^{\delta C}$, as described in Section 5.3.5. These arguments can be easily modified through the UI shown in Fig. 5.4.4e.

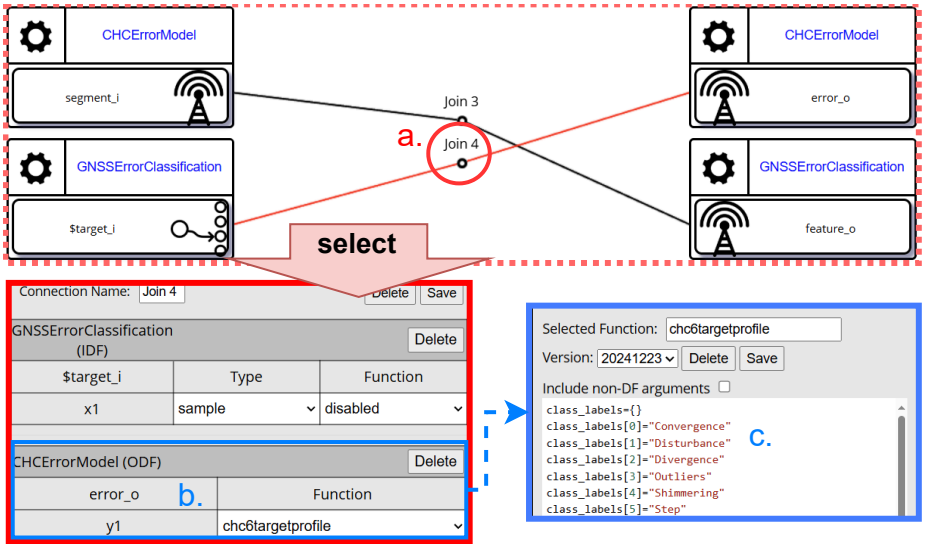


Figure 5.4.5: Deployment of error target profile on GNSS-EStalk.

In Fig. 5.4.5a, the $target\_i$ outputs the predicted target probabilities $Cls_{\check{d}i}$ for segments $\check{X}^{\delta}_{\check{d}i}$, as introduced in Fig. 5.3.7, while $error\_o$ receives the recognized errors. We deploy the $chc6targetprofile$ function, as shown in Fig. 5.4.5b. This function outputs the target class $\check{Y}^{\delta}_{\check{d}i}$ for each $\check{X}^{\delta}_{\check{d}i}$:

$$\check{Y}^{\delta}_{\check{d}i} = argmax(Cls_{\check{d}i}) \tag{5.4.6}$$

Additionally, the function can also return error information, as shown in Fig. 5.4.5c, including the details described in Section 5.3.2. The design allows for the modification of the profile depending on the application.

Fig. 5.4.6 depicts the data flow for the classification process under GNSS-EStalk. The feature extraction function, which includes preprocessing and RDM generation, is described in Section 5.3.3 and Section 5.3.4, respectively. The DAIoTtalk delivers the function to the remote server as a Python module, enabling its Publisher to import the module and convert each noise segment received from Project A into features $\check{X}^{\delta}_{\check{d}i}$ before publishing them to the classification node.

On the other hand, The model configuration for the classifier, stored in the DAIoTtalk, is received by the Subscriber handler of the classification node. It is

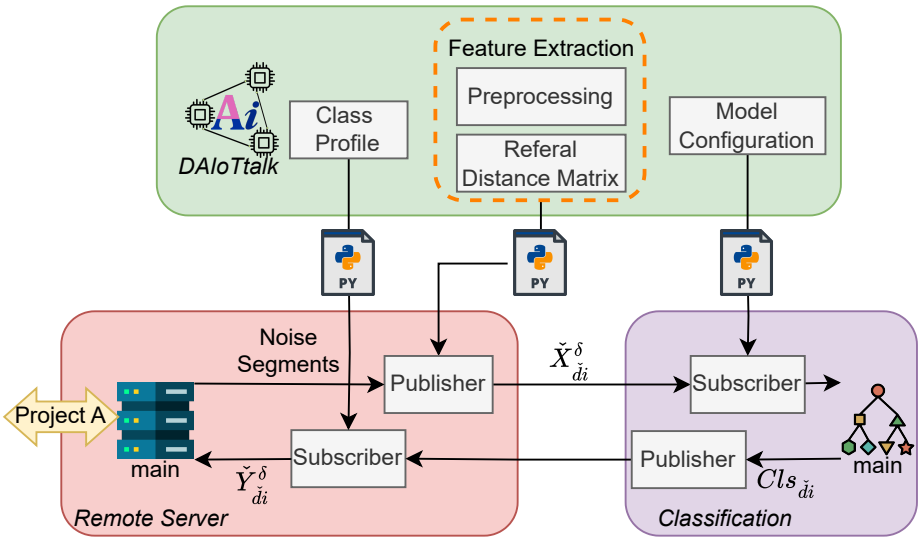


Figure 5.4.6: Data flow from the remote analysis server to the classification node in GNSS-EStalk.

queued alongside the $\check{X}^{\delta}_{\check{d}i}$ inputs before being accessed by the main thread, allowing the main thread to reload the model configuration before processing the next instance.

Once the main thread of the classification node generates the target results $Cls_{\check{d}i}$, the AGAPI publishes them, and the Subscriber of the remote server receives them. The handler imports the class profile provided by the DAIoTtalk to convert the target results into class labels $\check{Y}^{\delta}_{\check{d}i}$ using Eq. (5.4.6), along with the corresponding error profile. The error results are then relayed to Project A for transmission back to the corresponding ground station.

5.5 Evaluation

The experiment runs on an Ubuntu 18.04 server with an R9-5950x CPU, 32GB RAM, and RTX3090 GPU. It uses Python 3.10 and Tensorflow 2.17. The RDM reference sizes are set to 60. All models in Fig. 5.3.7 employ categorical focal cross-entropy loss. The models are trained for 100 epochs with restored best weights based on loss. Each model is trained with 10 trials, and the average result is presented.

In the following section, we first evaluate the noise classifiers through an epoch-loss comparison. Next, we assess the classifiers using test data. Then, we experiment with noise clustering models. Finally, we test the classifiers on a pseudo-labeled dataset.

5.5.1 Evaluation of Model Performance by Epoch

We first perform an epoch-loss comparison of the models shown in Fig. 5.5.1 and Fig. 5.5.2. The comparison involves four types of loss functions for references: categorical focal cross-entropy (CFEC), Kullback–Leibler divergence (KLD), mean absolute error (MAE), and mean squared error (MSE). The experiments are conducted using four different training sizes: 10%, 30%, 50%, and 70%.

Fig. 5.5.1 presents the epoch-loss comparison across the five baseline models. In general, a 10% training size exhibits the slowest convergence, with all the losses decreasing gradually and models often failing to reach optimal values. As the training size increases to 30% or more, a notable improvement in convergence speed is observed. Furthermore, Larger training sets (above 70%) also show faster convergence and improved loss stability, indicating better model generalization.

When compared with the other models, SCNN demonstrates the fastest convergence, achieving optimal performance across all loss functions within 20 epochs for all training scales. Furthermore, it has the lowest final loss value, indicating that SCNN is more efficient at extracting information from the sequence-generated heat map. On the other hand, TLSTM exhibits the most unstable convergence speed and loss landscape. With a 10% training size, TLSTM shows the highest final loss value, but its performance improves significantly as the training size increases. This suggests that LSTM models require more data to capture sequence patterns effectively.

Fig. 5.5.2 presents the epoch-loss comparison of the four hybrid models. In general, convergence speed improves significantly when the training size

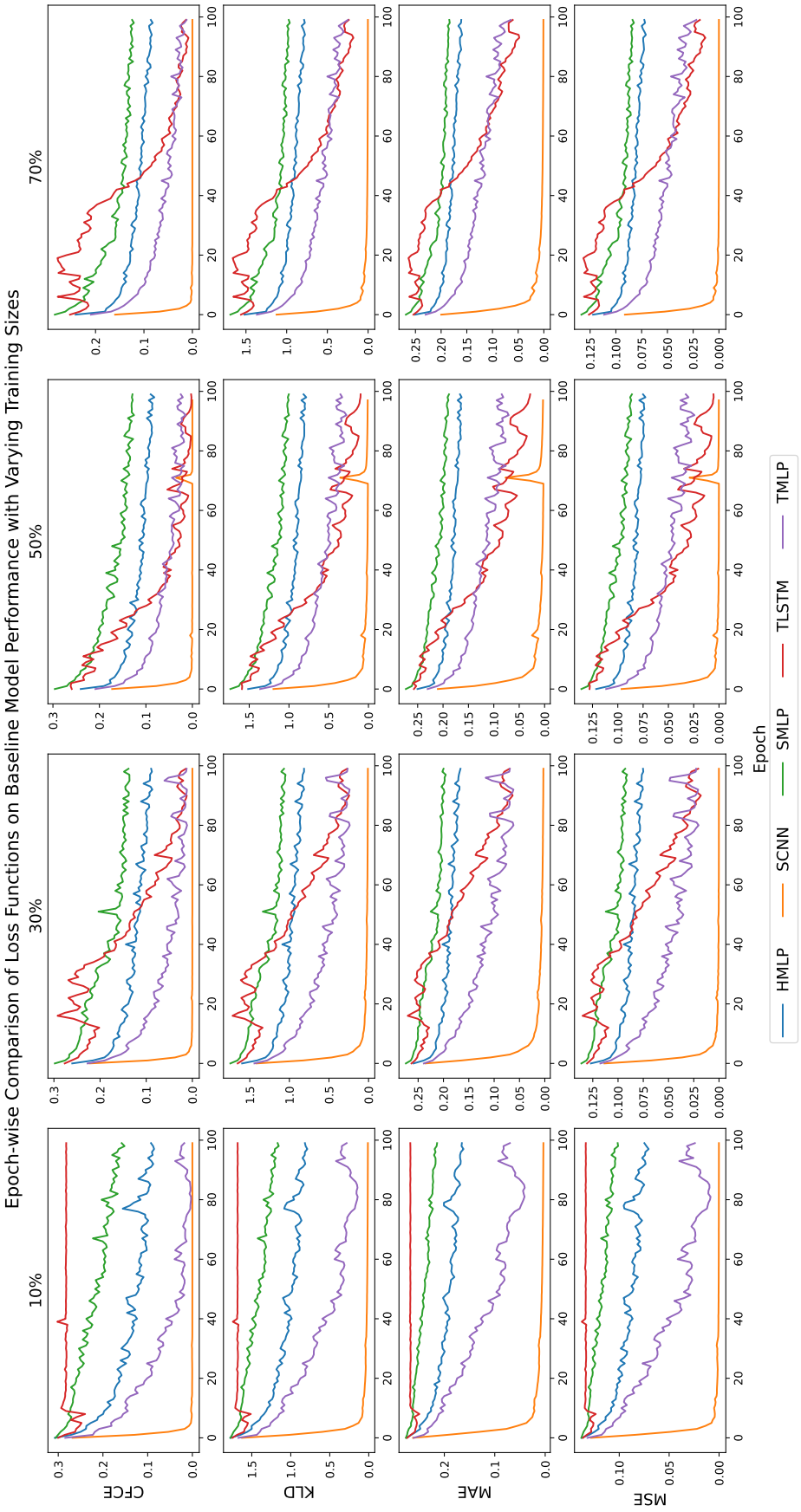


Figure 5.5.1: Baseline model performance in 100-Epoch.

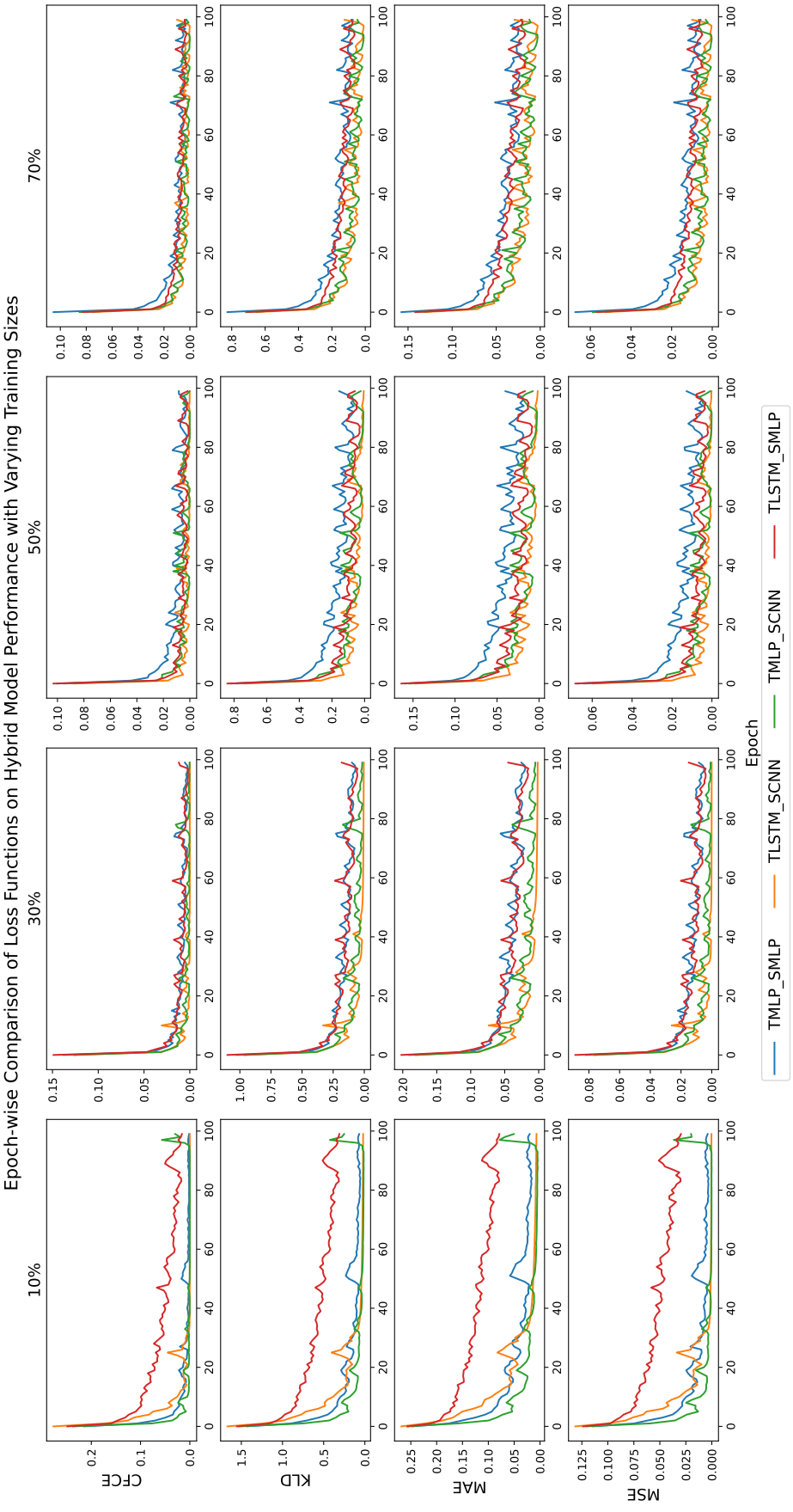


Figure 5.5.2: Hybrid model performance in 100-Epoch.

increases from 10% to 30% or higher. All models reach optimal values within 20 epochs for all loss functions. Stability also shows slight improvement when the training size reaches 70%.

When comparing the models, SCNN-based models (TLSTM_SCNN, TMLP_SCNN) generally demonstrate faster convergence, reaching near-optimal loss values within 20 epochs for most of the training sizes and loss functions. TLSTM_SCNN requires more epochs to stabilize when the training size is less than 30%, but there is a significant improvement when the training size increases to 50%. On the other hand, TMLP_SMLP has slower convergence than the other models when the training size exceeds 50%, with the difference being huge for loss functions such as KLD, MAE, and MSE.

Considering Fig. 5.5.1 and Fig. 5.5.2, the loss values of the baseline models decrease more consistently over epochs for smaller datasets (10%, 30%) than for larger datasets (50%, 70%). Across all configurations, Baseline models are more turbulent across configurations in the initial epochs. In contrast, hybrid models, which combine sequence and spatial components, achieve a more rapid loss reduction in the initial epochs. Furthermore, the variance in loss across epochs is less pronounced in hybrid models, indicating smoother convergence. As a result, hybrid models generally outperform baseline models by demonstrating faster convergence and achieving lower final loss values.

5.5.2 Evaluation of Baseline and Hybrid Noise Classification

Fig. 5.5.3 presents the average results from 10 trials based on classification accuracy and F1-macro score using only the labeled dataset. The training size varies from 0.05 to 0.7. In the baseline models, TMLP outperforms the others when the training size is below 0.3 for both metrics. The end-to-end methods, including TLSTM and SCNN, demonstrate better performance when the training size exceeds 0.5, with over 70% accuracy. Table 5.2 presents the summary of 10 trials using a 70% training size. As shown in the table, the

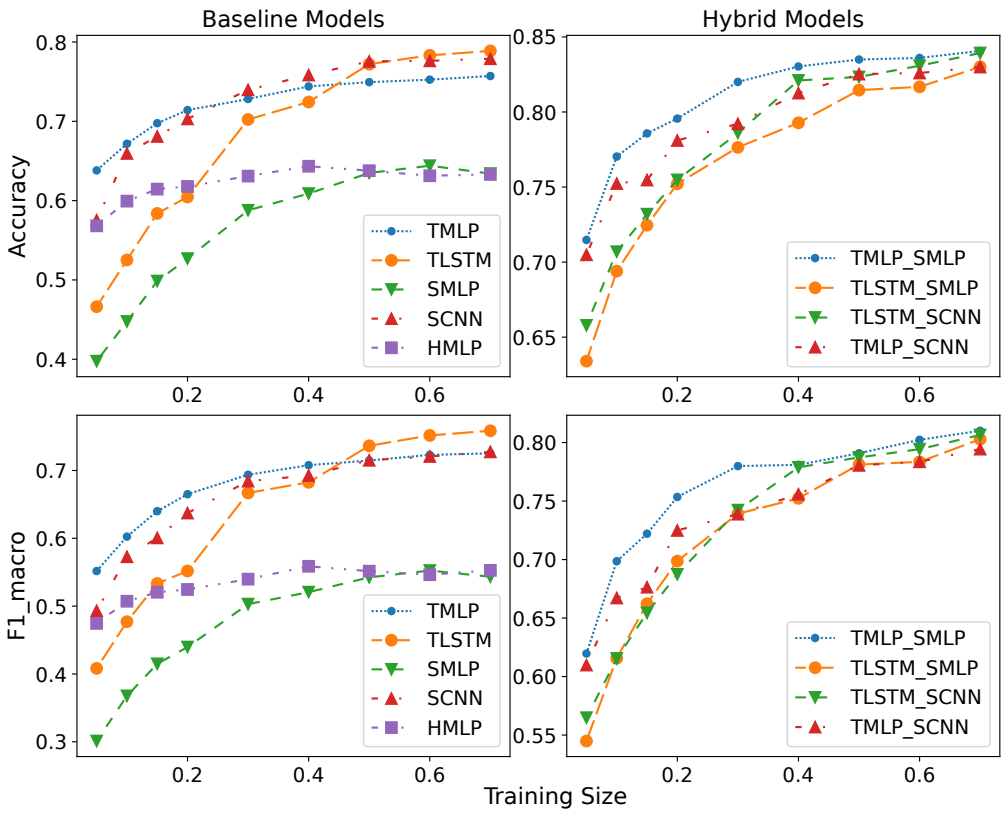


Figure 5.5.3: The comparison of accuracy and F1-macro score among baseline and hybrid models using different training sizes.

Table 5.2: Summary of accuracy and F1-macro scores across 10 trials for baseline and hybrid models using a 70% training size.

Model	Accuracy					F1-Marco				
	Mean	SD	Q1	Q2	Q3	Mean	SD	Q1	Q2	Q3
HMLP	0.633	0.012	0.626	0.634	0.641	0.553	0.016	0.543	0.549	0.562
TMLP	0.757	0.022	0.752	0.764	0.769	0.725	0.024	0.721	0.727	0.741
SMLP	0.634	0.022	0.626	0.632	0.652	0.543	0.022	0.532	0.545	0.559
TLSTM	0.789	0.077	0.797	0.808	0.814	0.759	0.077	0.764	0.779	0.788
SCNN	0.779	0.009	0.772	0.778	0.784	0.727	0.011	0.719	0.727	0.732
TMLP_SMLP	**0.841**	0.010	0.836	0.837	0.845	**0.810**	0.013	0.804	0.809	0.817
TLSTM_SMLP	0.830	0.008	0.826	0.828	0.831	0.803	0.010	0.798	0.801	0.804
TLSTM_SCNN	0.839	0.010	0.835	0.841	0.844	0.806	0.015	0.804	0.807	0.815
TMLP_SCNN	0.830	0.009	0.824	0.832	0.836	0.794	0.014	0.786	0.793	0.803

RDM-based model 'TMLP_SMLP' outperforms the others in both metrics, achieving approximately 84% accuracy and an 80% F1-macro score. Additionally, the differences of the hybrid models diminish as the training size increases. Compared to the baseline models, all hybrid models generally perform

better.

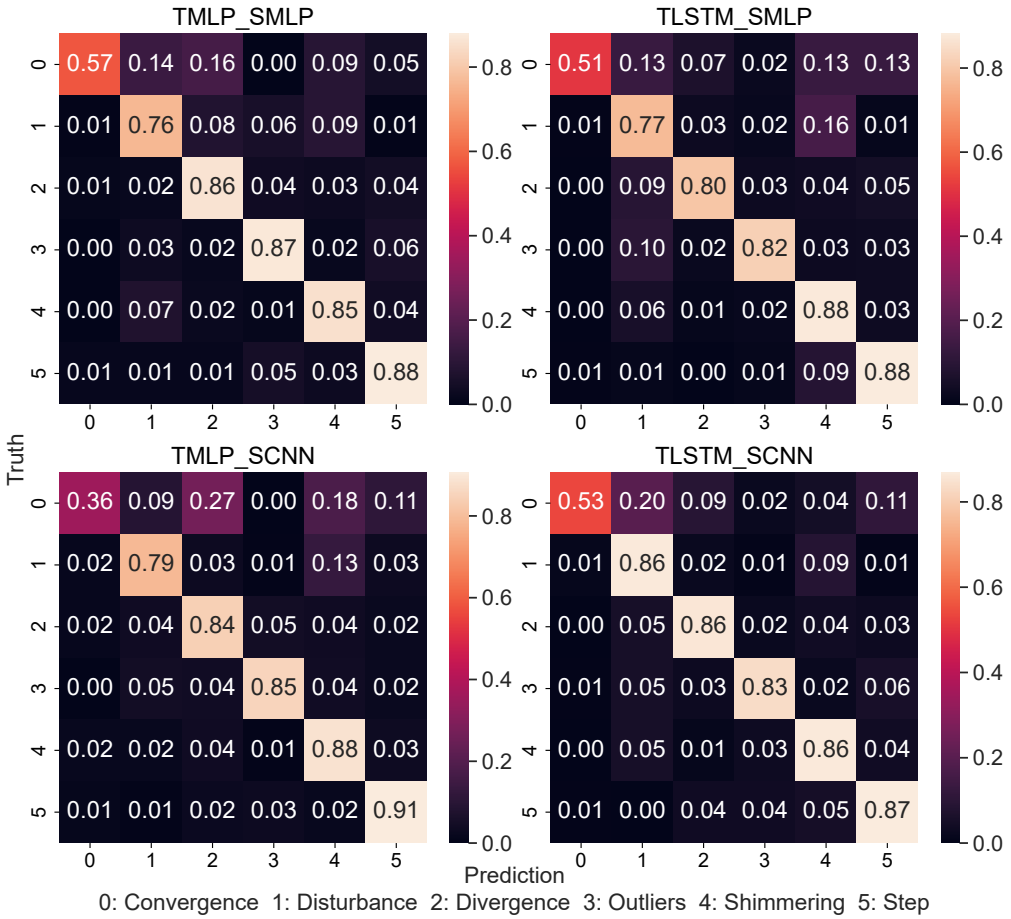


Figure 5.5.4: Normalized confusion matrices of the hybrid models.

Fig. 5.5.4 shows the normalized confusion matrices comparing the true labels with the predicted targets of the hybrid models when the training size is 0.7. According to the results, shimmering and Step are generally easy for the models to detect. On the other hand, the predictions for convergence segments are less accurate due to dataset imbalance. Introducing data augmentation or Balanced Batch Sampling may help address this issue. Overall, 'TLSTM_SCNN' demonstrates more balanced predictions across the classes than the other models.

5.5.3 Evaluation of Noise Clustering

Table 5.3 presents the results of the clustering experiments by adjusting $th^{\delta C}$. The training size is set to 0.2, and the testing data is mixed with the unlabeled

Table 5.3: The comparison of accuracy and data increment of clustering models.

		Accuracy			**Increment**		
$th^{\delta C}$	*Model*	*HDBSCAN*	*Hierarchical*	*KMeans*	*HDBSCAN*	*Hierarchical*	*KMeans*
	C^{τ}	0.958	0.974	**1**	0.378	**0.526**	0.172
0.1	C^{ς}	**1**	0.992	**1**	0.498	**0.535**	0.401
	C^{Φ}	**1**	0.979	**1**	0.397	**0.543**	0.259
	C^{τ}	**0.967**	0.95	0.916	0.523	**0.926**	0.535
0.3	C^{ς}	0.973	0.95	**0.984**	0.689	**1.055**	0.695
	C^{Φ}	0.971	0.935	**1**	0.512	**0.965**	0.584
	C^{τ}	**0.943**	0.88	0.897	0.664	**1.474**	1.229
0.5	C^{ς}	**0.944**	0.867	0.894	0.87	**1.659**	1.263
	C^{Φ}	0.973	0.915	**0.986**	0.645	**1.523**	0.896
	C^{τ}	**0.917**	0.86	0.856	0.833	2.025	**2.105**
0.7	C^{ς}	**0.892**	0.822	0.851	1.132	**2.279**	2.145
	C^{Φ}	**0.942**	0.858	0.929	0.815	**2.462**	1.792
	C^{τ}	**0.889**	0.799	0.795	1.084	2.707	**3.087**
0.9	C^{ς}	**0.84**	0.78	0.765	1.611	3.32	**3.354**
	C^{Φ}	**0.903**	0.828	0.855	1.184	**3.279**	3.145

data to generate the pseudo-labeling dataset. The evaluation focuses on two metrics: the accuracy of the testing data and the overall increase in the number of generated pseudo-labels. We test three clustering models: HDBSCAN, hierarchical clustering, and KMeans. Both hierarchical clustering and KMeans are configured to cluster the segments into 1000 classes, which is close to the number of clusters generated by HDBSCAN.

According to Table 5.3, as $th^{\delta C}$ increases, the overall number of pseudo-labels increases, but accuracy decreases. Considering the clustering models, HDBSCAN is more accurate, while hierarchical clustering generates more pseudo-labels. From the perspective of the pipelines, C^{Φ} is relatively more accurate, whereas C^{ς} generates more pseudo-labels. Overall, the noise segments extracted using the clustering method with ZFilter exhibit greater consistency than those obtained through classification.

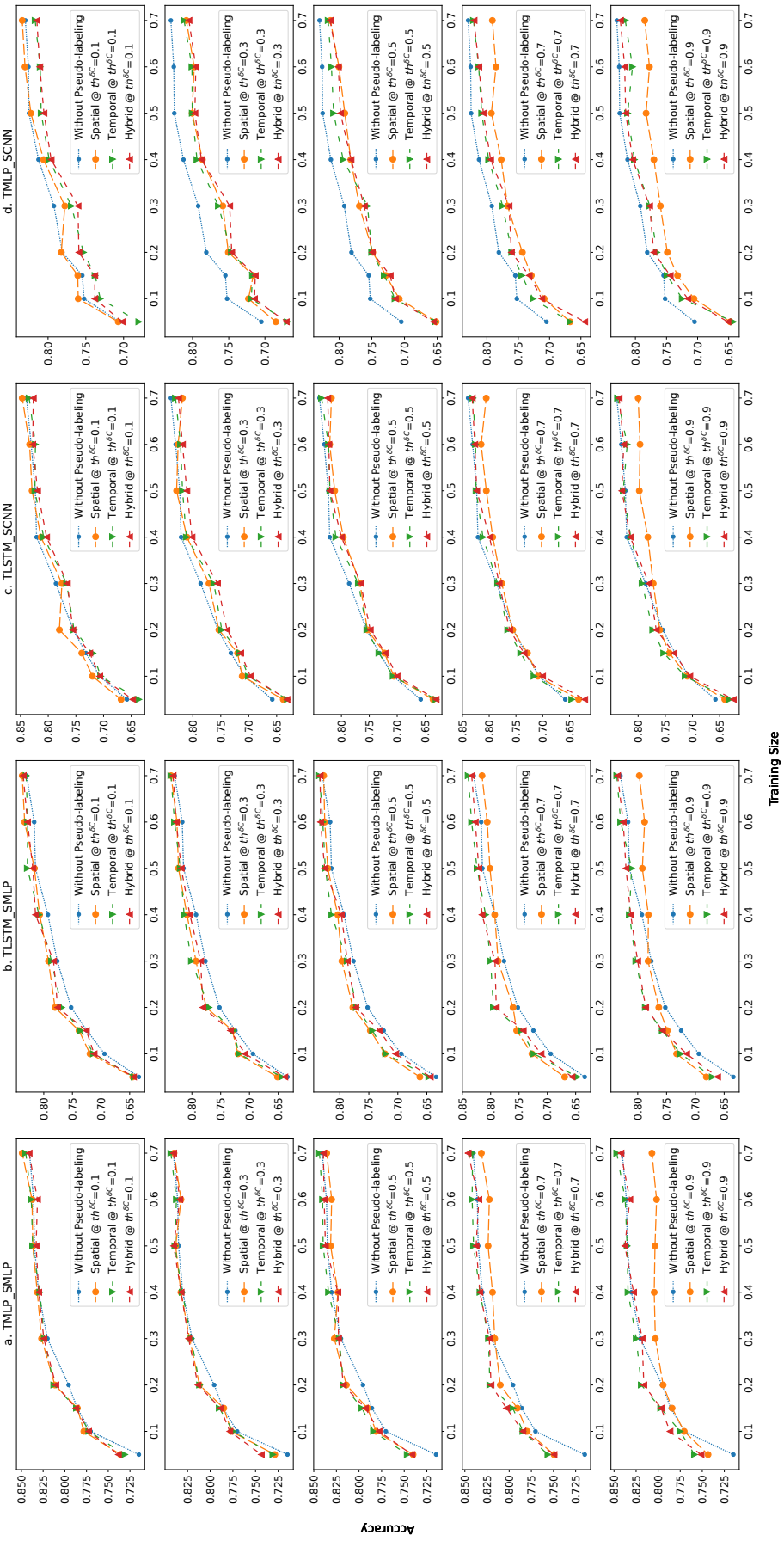


Figure 5.5.5: Experimental results on noise classification with pseudo-labeling.

5.5.4 Noise Classification Experiment Using Pseudo-Labeling

Fig. 5.5.5 displays the experimental results on noise classification using hybrid models with pseudo-labeling datasets based on HDBSCAN. The α in Eq. (5.3.22) is set to 0.2. We compare $th^{\delta C}$ values of 0.1, 0.3, 0.5, 0.7, and 0.9. As shown in the results, pseudo-labeling enhances the performance of TMLP_SMLP and TLSTM_SMLP by approximately 3% when the training size is low. Notably, when the training size is less than 0.2, the accuracy of TMLP_SMLP increases from around 78% to over 82% when $th^{\delta C} = 0.5$. However, TMLP_SCNN and TLSTM_SCNN perform worse with pseudo-labeled datasets, possibly due to the CNN model's increased sensitivity to false-positive data. Generally, the performance of $\acute{D}^{\delta\tau}$ and $\acute{D}^{\delta\Phi}$ outperforms $\acute{D}^{\delta\varsigma}$. When comparing the case where $th^{\delta C} = 0.9$ with the others, it generates more pseudo-labels, but this results in reduced performance due to the inclusion of more inaccurate data. The results indicate that ZFilter effectively selects the most similar noise segments, enhancing the model's performance when the labeled dataset is small. To further improve performance, incorporating additional unlabeled data can help generate more pseudo-labels.

5.6 Chapter Conclusions

In conclusion, we constructed a framework for GNSS error source analysis named GNSS-EStalk. Based on the regional ionospheric misclosure from the deeper-level receiver data, we derived a dataset of GNSS noise segments. An innovative AI temporal-spatial analysis approach was applied to handle the large volumes of daily data. We apply clustering along with the ZFilter strategy to extract highly consistent noise segments and generate a pseudo-labeled dataset. We achieved an accuracy of 84% recognizing the noise types in the segments with the hybrid classification model.

There are several potential future research directions. More deeper-level

parameters, such as orbit clock update residuals and tropospheric misclosure, can be considered to further characterize error sources like multipath interference, tropospheric delays, and receiver clock errors. Error forecasting can be performed by considering additional factors like ionospheric activity and tropospheric conditions. Consistent noise data can also be used to validate and enhance existing GNSS error models.

6

Evaluation and Discussion

6.1 Experiment Setup

We have implemented DAIoTtalk on a cluster of servers. Table 6.1 lists the machines used in our experiments. We deploy DAIoTtalk on VM1, publisher on PC1, and subscriber on PC2. To simulate actual deployment conditions, each node, including the DAIoTtalk in the virtual machine, is connected to a remote WireGuard [83] virtual private network (VPN) with a bandwidth of 67 Mbps. The timestamp is offset using a local NTP server. In the following experiments, we evaluate our platform against other baseline methods. These include asynchronous HTTP/1 with binary payload (AIOHTTP-BIN) and REST payload (AIOHTTP-REST), as well as MQTT with binary payload. Both the HTTP server and MQTT broker are deployed on the publisher machine. They operate within the domain of P2P communication, where data is sourced from the server machine. MQTT quality of service (QoS) is set to 0.

6.2 Impact of Packet Size

To evaluate communication efficiency, we first compared the latency when transmitting packets of different sizes in Fig. 6.2.1. Table 6.2 lists the improvement with the following metric:

Table 6.1: Experiment Platform

Machine	CPU	GPU	OS	RAM
PC1	AMD R9-5950x	RTX4090	Windows 10	128 GB
VM1			WSL Ubuntu 18.04	(VM 32 GB)
PC2	Intel I9-13900h	RTX4060m	Windows 11	16 GB

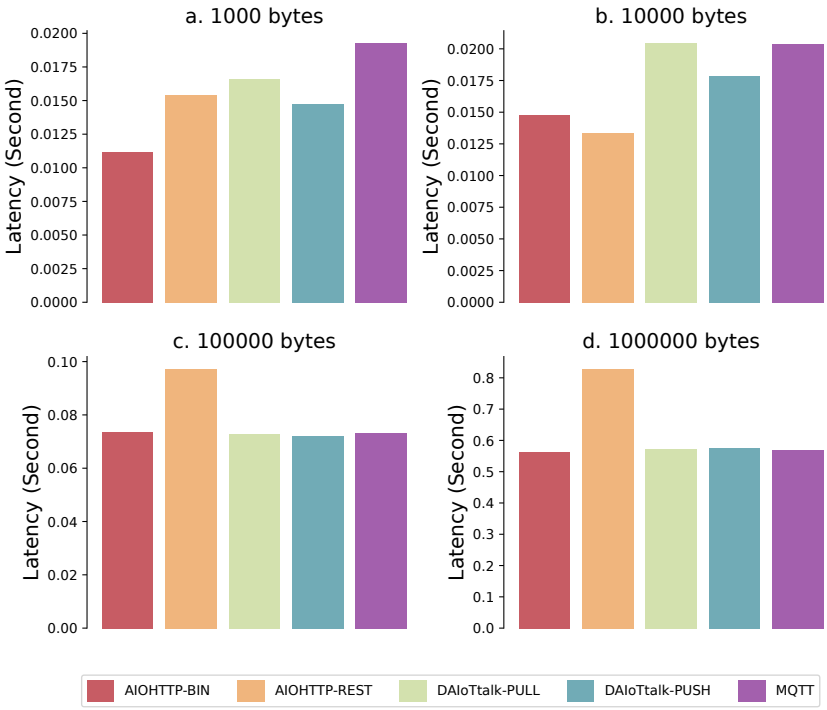


Figure 6.2.1: Comparison of latency when transmitting packets of different sizes at 1 Hz.

$$Improvement_{latency} = \frac{latency - latency_{max}}{latency_{max}} * 100\% \tag{6.2.1}$$

Lower latency indicates higher communication efficiency. We tested four packet sizes: 1,000 bytes, 10,000 bytes, 100,000 bytes, and 1,000,000 bytes. The first

Table 6.2: Improvement of latency when transmitting packets of different sizes at 1 Hz

Packet Size	AIOHTTP-BIN	AIOHTTP-REST	DAIoTtalk-PULL	DAIoTtalk-PUSH	MQTT
1000	-42%	-20%	-14%	-24%	0%
10000	-28%	-35%	0%	-13%	0%
100000	-24%	0%	-25%	-26%	-25%
1000000	-32%	0%	-31%	-31%	-31%

two were to simulate common IoT data such as instructions and sensor readings. The other two were to simulate common AI-related applications with regular images and high-resolution images. Packets were sent with a one-second interval (i.e., 1 Hz). The results were based on the average of 10 packets. In the cases of Fig. 6.2.1a and Fig. 6.2.1b, AIOHTTP showed a slight advantage over both DAIoTtalk-PULL and DAIoTtalk when dealing with small packets. This could be attributed to the overhead incurred when the publisher sends a packet with buffering, whereas the HTTP server simply generates a packet and responds immediately. As the packet size increased, as in the cases of Fig. 6.2.1c and Fig. 6.2.1d, the advantage of binary encoding-based schemes became obvious compared to the string-encoded REST method; the improvement was around 25% to 35% according to Table 6.2 when packet size larger than 100000 bytes. In this case, AIOHTTP and our DAIoTtalk performed similarly.

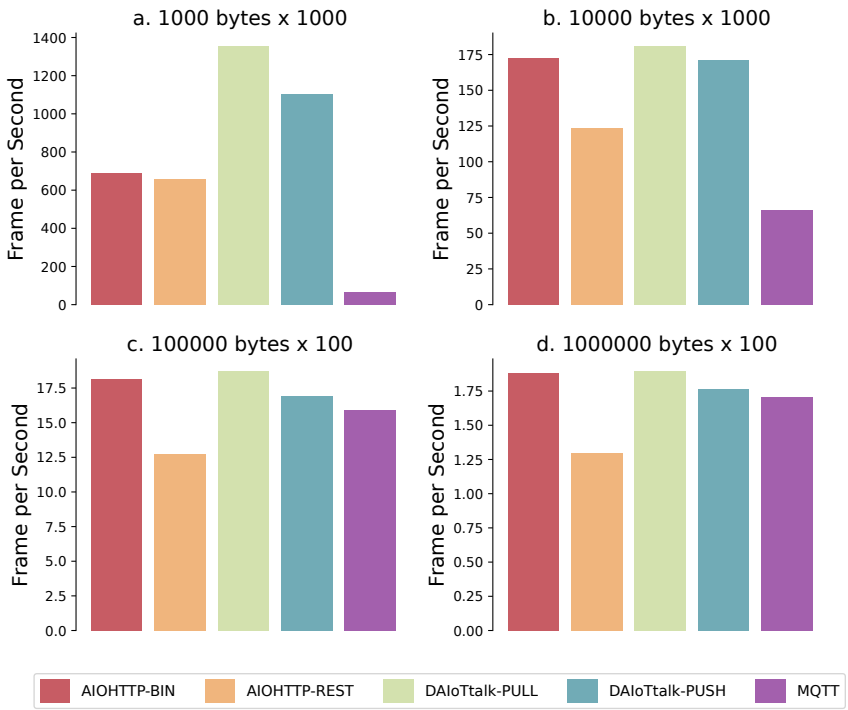


Figure 6.2.2: Comparison of FPS when flushing a buffer of different numbers of packets of various packet sizes.

In the next experiment, we simulated a more stressful scenario. We inserted a large amount of data in a buffer and tried to flush the data in the buffer. We

Table 6.3: Improvement of FPS when flushing a buffer of different numbers of packets of various packet sizes.

Bytes ×#	AIOHTTP-BIN	AIOHTTP-REST	DAIoTtalk-PULL	DAIoTtalk-PUSH	MQTT
1000×1000	957%	910%	1989%	1596%	0%
10000×1000	160%	86%	173%	158%	0%
100000×100	42%	0%	47%	33%	25%
1000000×100	45%	0%	46%	36%	32%

then observed the frames per second (FPS) of different protocols. A higher FPS indicates higher communication efficiency. The results are shown in Fig. 6.2.2. Table 6.3 lists the improvement with the following metric:

$$Improvement_{FPS} = \frac{FPS - FPS_{min}}{FPS_{min}} * 100\% \tag{6.2.2}$$

We tested with 1,000 packets of sizes 1,000 and 10,000 bytes each, plus 100 packets of sizes 100,000 and 1,000,000 bytes apiece. In Fig. 6.2.2a, both DAIoTtalk-PULL and DAIoTtalk-PUSH outperformed the singleplexing MQTT and the multiplexing AIOHTTP by at least 1596% and 65% respectively, according to Table 6.3. They took advantage of HTTP/2 with its native multiplexing support, allowing multiple requests and responses over a single TCP connection. While AIOHTTP-BIN also utilized multiplexing via asynchronous optimization, it slightly lagged DAIoTtalk-PULL in Fig. 6.2.2b and Fig. 6.2.2c. As the frame size increased to 1,000,000 bytes, the gap narrowed in Fig. 6.2.2d since the TCP connection was quite full and multiplexing had no advantage. However, the binary encoded approaches still lead the string encoded AIOHTTP-REST by at least 32% according to Table 6.3. In conclusion, DAIoTtalk achieves at least a 33% improvement over the traditional protocol in this experiment.

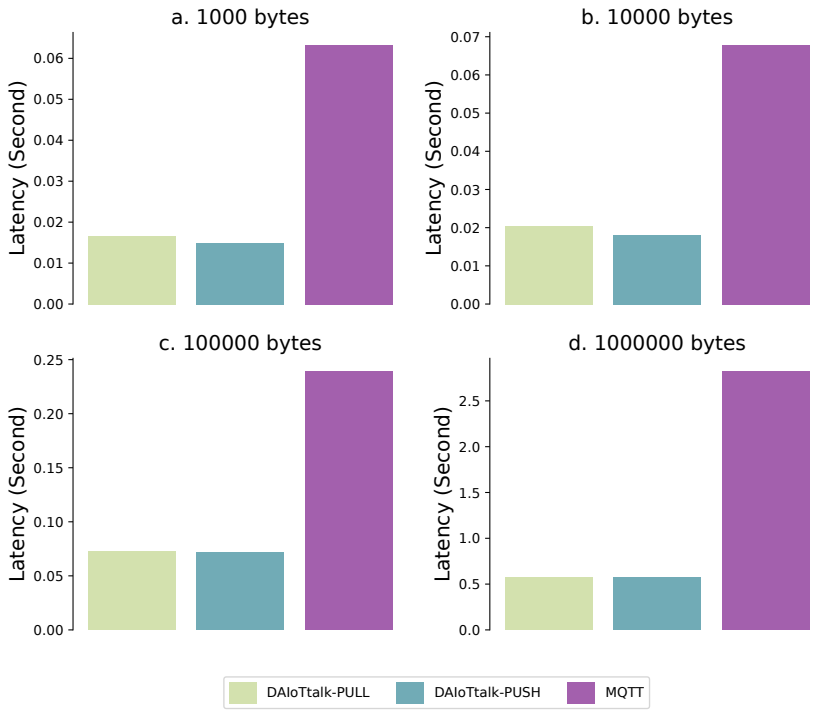


Figure 6.3.1: Comparison of latency between data-centralized and data-decentralized design.

Table 6.4: Improvement of latency between data-centralized and data-decentralized design.

	DAIoTtalk-PULL	DAIoTtalk-PUSH	MQTT
1000	-74%	-77%	0%
10000	-70%	-74%	0%
100000	-70%	-70%	0%
1000000	-80%	-80%	0%

6.3 Data-Centralized vs. Data-Decentralized Approaches

To compare the gap in communication efficiency between a data-centralized and a data-decentralized design, we relocated the MQTT broker along with the DAIoTtalk server to simulate a data-centralized approach. Here, we again compare the latency for different packet sizes. Fig. 6.3.1 shows the results after

the adjustment and Table 6.4 lists the improvement using Eq. (6.2.1). According to Fig. 6.3.1, DAIoTtalk-PULL and DAIoTtalk-PUSH outperformed the MQTT method with a remote broker by a significant margin, achieving at least 3 times lower latency in transmitting small packets (Fig. 6.3.1a and Fig. 6.3.1b) and nearly 5 times lower latency in transmitting large packets (Fig. 6.3.1c and Fig. 6.3.1d) under common bandwidth conditions. According to Table 6.4, P2P communication offers at least a 70% reduction in latency compared to the server/broker approach. Comparing the differences between Fig. 6.2.1 and Fig. 6.3.1, it is evident that P2P communication significantly reduces traffic overhead on the broker with MQTT, from over 2.5s reduced to under 0.6s.

6.4 Simulation of Offloading with the Join Function

To evaluate the resource scalability with the JF, we follow [84] to implement a script for clustering the Iris dataset using K-Means. In the simulation, multiple publisher nodes on PC1 publish the Iris dataset, which is received by a single subscriber on PC2. The script is deployed on the publisher side, where the dataset is clustered before being sent. Given N publishers, each sending a packet 10 times at 1 FPS, a total of $N \times 10$ JF execution loops will be performed.

Since the transmission is asynchronous, we measure only the ideal case using processing time to calculate the speed-up rate $R^{offload}$:

$$R^{offload} = \frac{t^{Seq}}{max(t^{JF})} \tag{6.4.1}$$

where t^{Seq} represents the processing time for the sequential execution of $N \times 10$ loops on PC1, and $t^{JF} = \{ t^{JF}_{n \in [0,N)} \}$ is the set of parallel processing times for each node, with each node executing 10 loops. A higher $R^{offload}$ indicates better resource scalability.

Fig. 6.4.1 presents the simulation results. In Fig. 6.4.1a, we compare the linear sequential processing time with the maximum, median, and minimum parallel

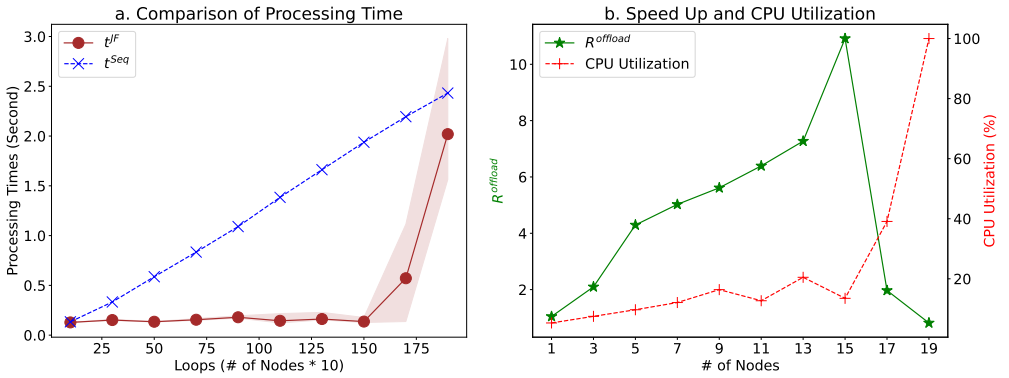


Figure 6.4.1: Simulation results on parallel processing with JF.

processing times. When the loop size is below 150 (with 15 nodes), t^{JF} remains around 0.2 seconds. However, as the loop size exceeds 170, t^{JF} increases significantly. In the worst case, with 19 nodes running, it takes nearly 3 seconds to complete 10 loops—longer than t^{Seq} for 190 loops.

Fig. 6.4.1b illustrates the speed-up rate alongside CPU utilization on PC1. When nodes are fewer than 15, CPU usage remains under 20%, and parallel processing with JF operates efficiently, ideally achieving a speed-up of over 10 times compared to the sequential process. However, once hardware limits are reached, performance drops significantly and may even be inefficient. In real-world deployment, an optimal configuration needs to consider factors such as node distribution across devices and the complexity of each JF.

6.5 Case Study Experiment

Fig. 6.5.1 illustrates the simulation deployment setup for the case study with multiple communication types to showcase the deployment versatility of DAIoTtalk. The deployment considers conditions of LAN communication within the local site, WAN communication between the local site and a remote service server via a VPN tunnel, and inter-process communication (IPC) within the remote server. The test data includes approximately 1 minute of sensor and camera data, published through 3 *SmartBadges* nodes and 1 *Camera* node.

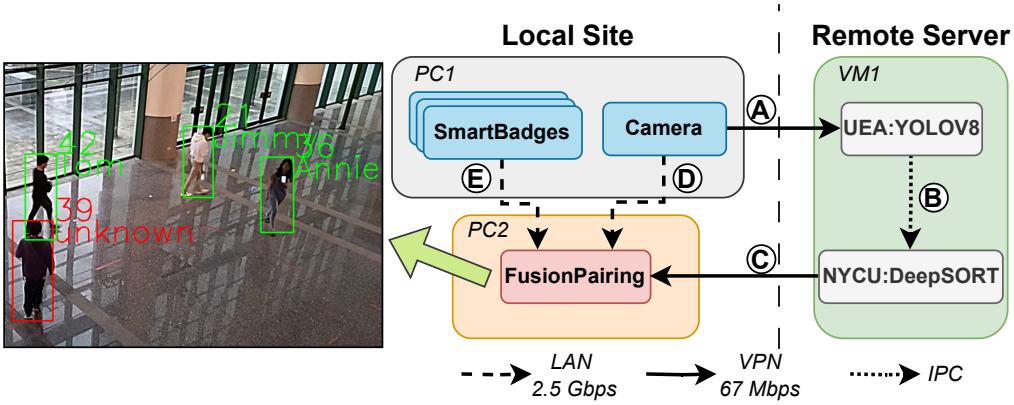


Figure 6.5.1: Network Deployment for Case Study.

Table 6.5: Data Flow Measurement in the Case Study Deployment

Data Flow	# of Frames	Average Size (bytes)	Transmission Latency (s)			Process Latency (s)	Accumulated Latency (s)
			Mean	Q1	Q3		
A	516	161005	0.097	0.0602	0.132	0.061	**0.158**
B	407	160892	0.049	0.0227	0.0684	0.192	**0.353**
C	349	455	0.088	0.0626	0.136		
D	495	466875	0.1	0.0626	0.136	0.45	**0.543**
E	1483	1555	0.15	0.0814	0.215		

Only the updated data will be transmitted upon receiving a request. The *SmartBadges* nodes operate using the DAIoTtalk-PUSH mode, while the others will use the DAIoTtalk-PULL mode. We monitor each communication's data flow (A-E), and the results are presented in Table 6.5.

As shown in Table 6.5, data flows A and D are respectively sending low-resolution pictures via VPN and high-resolution pictures via LAN with similar latency. This shows that our design balances network traffic by adjusting multimedia data according to application requirements; Data flow B demonstrates the potential use of IPC. The accumulated latency can be maintained at around 0.5 ms, enabling real-time AI surveillance applications. This also shows the deployment versatility of the platform.

On the other hand, DAIoTtalk-PUSH is less stable than DAIoTtalk-PULL, as indicated by data flow E. Further optimization in AGAPI could enhance its performance. Besides, the data flow chaining from A to C shows that some

Table 6.6: Comparison of latency over 100 packets sent from the sewing machine in SewingTalk.

Platform	Average Size (Bytes)	Latency (s)					Average Improvement
		Mean	SD	Q1	Q2	Q3	
DAIoTtalk	803261.4158	0.635916	0.904305	0.182388	0.34377	0.544737	-43.03%
MQTT	858882.3861	1.116269	1.058549	0.344866	0.590394	1.738467	0

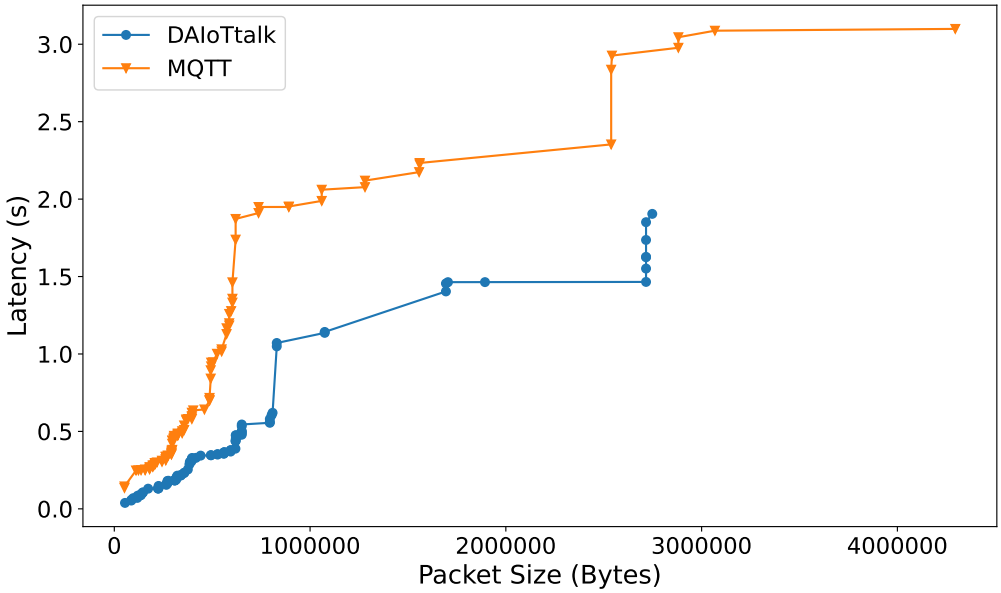


Figure 6.6.1: Comparison of latency within two standard deviations across different packet sizes in SewingTalk.

frames may be dropped due to connection issues or process latency. This issue could be mitigated by introducing QoS control.

6.6 Evaluation on SewingTalk

To demonstrate the connection efficiency of DAIoTtalk in the smart sewing industry, we reproduce the experiments in Section 6.3 and compare the transmission latency of machine logs sent from a sewing machine using both DAIoTtalk and MQTT. A total of 100 machine logs are randomly selected from the dataset and transmitted. Due to the large data size, the transmission rate is reduced to 0.2Hz (one packet every 5 seconds) to ensure sufficient session time for completing the transmission. Table 6.6 summarizes the improvement,

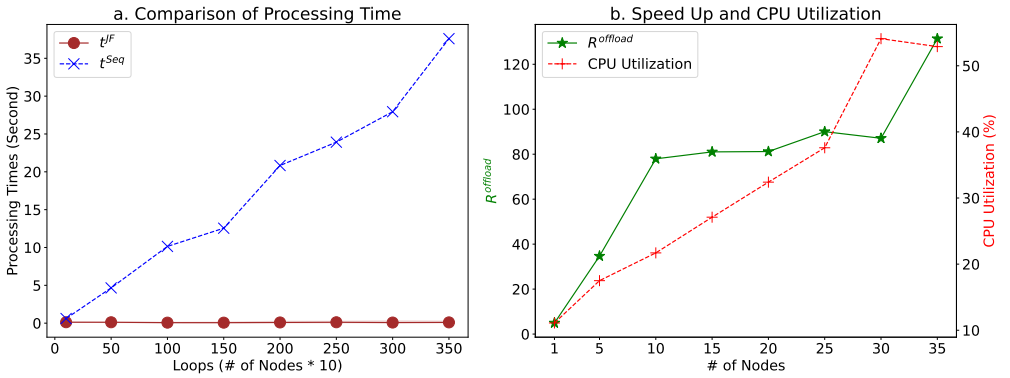


Figure 6.6.2: Evaluation of offloading of tokenization on sewing machine logs using JF.

calculated using Eq. (6.2.1). On average, latency is improved by 43% when using DAIoTtalk instead of MQTT. Fig. 6.6.1 illustrates how latency varies with different packet sizes. The two-standard-deviation empirical rule is applied to filter out sudden network delays. As shown, the latency gap between DAIoTtalk and MQTT increases with packet size, indicating that the data-decentralized design of DAIoTtalk effectively reduces network traffic turbulence by avoiding triangle routing.

In the deployment of SewingTalk, the tokenization function is implemented using JF to offload processing to the sewing machine. To assess the resource scalability provided by DAIoTtalk's JF in the smart sewing industry, we reproduce the simulation described in Section 6.4 using real machine log data. The results are presented in Fig. 6.6.2. In Fig. 6.6.2a, the offloaded processing time (t^{JF}) remains under one second as the task is parallelized across up to 35 nodes, whereas the sequential processing time (t^{Seq}) increases linearly. Fig. 6.6.2b shows the speed-up rate, calculated using Eq. (6.4.1), indicating that the offloading approach achieves up to a 120× speed-up compared to sequential processing when utilizing 35 nodes. These results showcase that JF enables effective horizontal scalability with sewing machines.

Table 6.7: Comparison of latency over 100 packets sent from the ground station in GNSS-EStalk.

Platform	Average Size (Bytes)		Latency (s)					Average Improvement
	Data	Packet	Mean	SD	Q1	Q2	Q3	
DAIoTtalk	1907525.99	1954703	1.045778	0.145055	0.889009	1.096319	1.163973	51.026%
DAIoTtalk_JF	1926303	48013.26	0.014572	0.011273	0.005598	0.012621	0.020497	99.138%
MQTT	1968470.98	2015895	2.18699	0.256347	2.097883	2.246862	2.376696	0

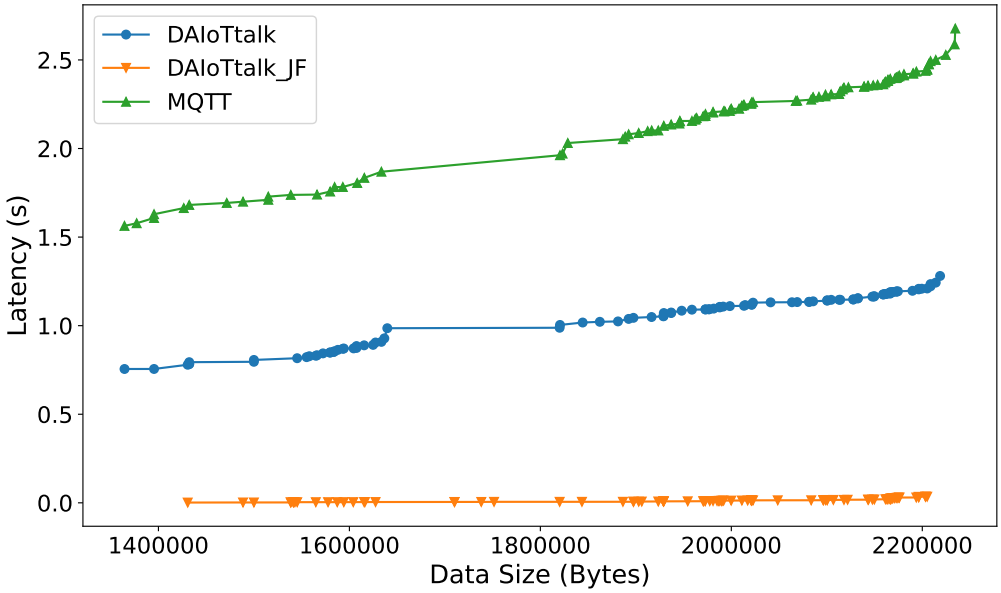


Figure 6.7.1: Comparison of latency within two standard deviations across different packet sizes in GNSS-EStalk.

6.7 Evaluation on GNSS-EStalk

To demonstrate the connection efficiency and deployment versatility of GNSS-EStalk in GNSS error source service, we reproduce the experiments in Section 6.3 at 0.2Hz and compare the transmission latency of ground station correction data sent from a ground station using both DAIoTtalk and MQTT. A total of 100 correction data points are randomly selected from the dataset and transmitted. Table 6.7 summarizes the improvement, calculated using Eq. (6.2.1). The DAIoTtalk_JF configuration transmits only the noise segments preprocessed by JF, reducing the packet size from nearly 2MB to under 0.5MB. On average, latency is improved by 51% using DAIoTtalk and by up to 99% when combined with JF preprocessing, compared to MQTT. Fig. 6.7.1 illustrates how latency varies with different data sizes. The

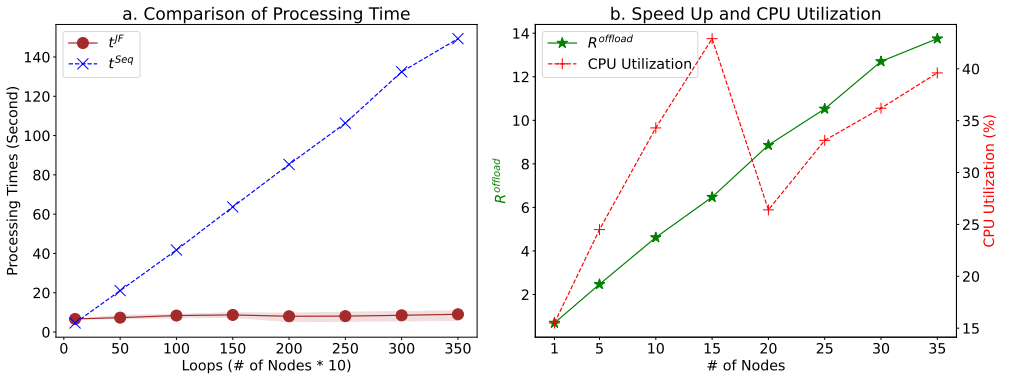


Figure 6.7.2: Evaluation of offloading of segmentation on GNSS error data using JF.

two-standard-deviation empirical rule is applied to filter out sudden network delays. As shown, DAIoTtalk already outperforms MQTT, while DAIoTtalk_JF further enhances efficiency by transmitting only the necessary segmented data. These results demonstrate that communication efficiency can be improved not only through P2P communication, which avoids triangular routing, but also by selectively transmitting essential data segments, highlighting the deployment versatility enabled by JF preprocessing.

To evaluate the resource scalability of GNSS-EStalk for noise segmentation tasks, we reproduce the simulation described in Section 6.4 using real ground station data. The results are presented in Fig. 6.7.2. In Fig. 6.7.2a, the offloaded processing time (t^{JF}) remains below 10 seconds as the task is parallelized across up to 35 nodes, while the sequential processing time (t^{Seq}) increases linearly. Fig. 6.7.2b illustrates the speed-up rate computed using Eq. (6.4.1), showing that the offloading approach achieves up to a 14× speed-up over sequential processing when leveraging 35 nodes. These results demonstrate that JF enables efficient horizontal scalability by remotely deploying JF processing to ground station nodes.

6.8 Chapter Conclusion

This chapter evaluated a novel data-decentralized pub-sub communication framework based on the gRPC protocol. The framework is integrated into IoTtalk, forming AIoTtalk, to support P2P communication and enhance communication efficiency compared with traditional methods. Experimental results demonstrate that the platform satisfies the resource scalability and flexibility requirements of AI applications. Furthermore, real-world case studies validate the deployment versatility of the platform, which also further improves application efficiency. Overall, the results confirm that the proposed platform effectively overcomes the limitations of existing centralized IoT architectures in AIoT applications.

7

Conclusions

7.1 Conclusions

The growing demand for Internet of Things (IoT) applications, especially within Artificial Intelligence of Things (AIoT), requires suitable platforms that can enable efficient data exchange as well as allow scalable deployment of applications. Traditional IoT platforms, whether data-cloud-based or data-centralized, commonly rely on server-mediated architectures. The architecture introduces challenges such as triangle routing, network bottlenecks, and limited scalability. The challenges are especially pronounced in AIoT scenarios requiring high-volume, low-latency data streams. To address these limitations, this work introduced DAIoTtalk, a data-decentralized AIoT platform built as an extension of IoTtalk. Leveraging peer-to-peer (P2P) communications powered by customized gRPC within a publish-subscribe (Pub-Sub) framework, DAIoTtalk facilitates direct sender-to-receiver data exchanges via a remote "Agent" dedicated to connectivity establishment. Two case studies, SewingTalk and GNSS-EStalk, are implemented to showcase showcase its potential to transform industries and services.

Through experiments and case studies, we have validated that our platform enhances communication efficiency, resource scalability, and deployment versatility. For communication efficiency, the gRPC pub-sub framework benefits from binary encoding and HTTP/2 multiplexing, which demonstrates at least

25% improvement in latency and 33% improvement in FPS compared to the traditional method, respectively. The data-decentralized approach also improves 70% latency over the data-centralized approach. For resource scalability, the JF can dynamically and remotely offload processing tasks to edge nodes, enabling parallel handling of larger data volumes and heavier computational loads as more edge nodes are added. For deployment versatility, case studies such as SewingTalk and GNSS-EStalk demonstrate the flexible modularization of AIoT components and dynamic network allocation in different environments. These results indicate that DAIoTtalk is well-suited for modern AIoT applications.

7.2 Furture Works

Several research directions emerge for future exploration. For the DAIoTtalk platform, current implementations focus on node deployment using WireGuard, with potential extensions into dynamic DNS (DDNS) and port forwarding for more flexible connectivity. Support for communication protocols other than gRPC can be enabled through additional APIs. To enhance security, a new form of topic-level encryption using ciphers or tokens can be introduced. Additionally, a novel IP resolving mechanism can be implemented on the Agent to support dynamic service scaling more effectively.

Further research is also planned for SewingTalk and GNSS-EStalk. For SewingTalk, we are looking for self-supervised learning to fusion product designs and procedure diagrams into the model for transfer learning. The system can then be extended for worker performance assessment and production line optimization. For GNSS-EStalk, deeper-level parameters, such as orbit clock residuals and tropospheric disclosure, could refine error characterization, while forecasting models incorporating ionospheric and tropospheric conditions may enhance PNT performance. Across the DAIoTtalk platform, modularization of AI models and algorithms remains a priority to improve adaptability and scalability.

Bibliography

[1] S. J. Palmisano. "A smarter planet: The next leadership agenda," Accessed: Jul. 21, 2025. [Online]. Available: https://www.youtube.com/watch?v=i_j4-Fm_Svs.

[2] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business and Inf. Systems Eng.*, vol. 6, no. 4, pp. 239–242, Aug. 2014, ISSN: 1867-0202. DOI: 10.1007/s12599-014-0334-4.

[3] C. J. B. Yann LeCun Corinna Cortes. "The MNIST database of handwritten digits," Accessed: Jul. 21, 2025. [Online]. Available: https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html.

[4] K. Kowsari, M. Heidarysafa, D. E. Brown, K. J. Meimandi, and L. E. Barnes, "RMDl: Random multimodel deep learning for classification," in *Proc. of the 2nd Int. Conf. on Inf. System and Data Mining*, ser. ICISDM '18, Lakeland, FL, USA: Association for Comput. Machinery, 2018, pp. 19–28. DOI: 10.1145/3206098.3206111.

[5] J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proc. of the 2012 IEEE Conf. on Comput. Vis. and Pattern Recognit. (CVPR)*, ser. CVPR '12, USA: IEEE Comput. Society, 2012, pp. 3642–3649.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[7] Papers With Code. "Image classification on ImageNet," Accessed: Jul. 21, 2025. [Online]. Available: https://paperswithcode.com/sota/image-classification-on-imagenet.

[8] C. Gordon. "How general AI will eventually reshape everything," Accessed: Jul. 21, 2025. [Online]. Available: https://www.forbes.com/sites/cindygordon/2023/09/30/how-general-ai-will-eventually-reshape-everything/.

[9] K. L. Lueth. "State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time," Accessed: Jul. 21, 2025. [Online]. Available: https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/.

[10] IDC. "IoT growth demands rethink of long-term storage strategies, says IDC," Accessed: Jul. 21, 2025. [Online]. Available: https://iotbusinessnews.com/2020/07/29/20898-iot-growth-demands-rethink-of-long-term-storage-strategies-says-idc/.

[11] Epic Games. "Unreal engine | the most powerful real-time 3d creation tool," Accessed: Jul. 21, 2025. [Online]. Available: https://www.unrealengine.com/en-US.

[12] Unity Technologies. "Unity," Accessed: Jul. 21, 2025. [Online]. Available: https://unity.com/.

[13] IFTTT. "IFTTT," Accessed: Jul. 21, 2025. [Online]. Available: https://ifttt.com/.

[14] D. NamIoT and M. sneps-sneppe, "On micro-services architecture," *Int. J. Open Inf. Technol.*, vol. 2, pp. 24–27, Sep. 2014.

[15] J. Wytrębowicz, K. Cabaj, and J. Krawiec, "Messaging protocols for IoT systems — a pragmatic comparison," *Sensors*, vol. 21, no. 20, 2021. DOI: 10.3390/s21206904.

[16] C. Gündoğan, P. Kietzmann, M. Lenders, H. Petersen, T. C. Schmidt, and M. Wählisch, "NDN, CoAP, and MQTT: a comparative measurement study in the IoT," in *Proc. 5th ACM ICN '18*, Boston, Massachusetts, pp. 159–171, ISBN: 9781450359597. DOI: 10.1145/3267955.3267967.

[17] Y.-B. Lin, Y.-W. Lin, C.-M. Huang, C.-Y. Chih, and P. Lin, "IoTtalk: A management platform for reconfigurable sensor devices," *IEEE Internet of Things J.*, vol. 4, no. 5, pp. 1552–1562, 2017. DOI: 10.1109/JIOT.2017.2682100.

[18] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, "A survey of communication protocols for Internet of Things and related challenges of fog and cloud computing integration," *ACM Comput. Surv.*, vol. 51, no. 6, Jan. 2019, ISSN: 0360-0300. DOI: 10.1145/3292674.

[19] C. Severance, "Roy T. fielding: Understanding the REST style," *Comput.*, vol. 48, no. 6, pp. 7–9, 2015. DOI: 10.1109/MC.2015.170.

[20] E. R. Fielding and E. J. Reschke. "Hypertext transfer protocol (HTTP/1.1): Message syntax and routing," Accessed: Jul. 21, 2025. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7230.

[21] D. Happ, N. Karowski, T. Menzel, V. Handziski, and A. Wolisz, "Meeting IoT platform requirements with open pub/sub solutions," *Ann. of Telecommunications*, vol. 72, no. 1-2, pp. 41–52, 2016. DOI: 10.1007/s12243-016-0537-4.

[22] OASIS Message Queuing Telemetry Transport (MQTT) TC. "MQTT version 5.0," Accessed: Jul. 21, 2025. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[23] Z. Shelby, K. Hartke, and C. Bormann. "The constrained application protocol (CoAP)," Accessed: Jul. 21, 2025. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7252.

[24] OASIS Advanced Message Queuing Protocol (AMQP) TC. "OASIS Advanced Message Queuing Protocol (AMQP) version 1.0," Accessed:

Jul. 21, 2025. [Online]. Available: http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html.

[25] P. Hintjens. "ZeroMQ - the guide," Accessed: Jul. 21, 2025. [Online]. Available: https://zguide.zeromq.org/.

[26] gRPC Authors. "Introduction to gRPC," Accessed: Jul. 21, 2025. [Online]. Available: https://grpc.io/docs/what-is-grpc/introduction/.

[27] TC39. "ECMA-404 the JSON data interchange syntax," Accessed: Jul. 21, 2025. [Online]. Available: https://www.ecma-Int..org/publications-and-standards/standards/ecma-404/.

[28] G. Developers. "Protocol buffers," Accessed: Jul. 21, 2025. [Online]. Available: https://developers.google.com/protocol-buffers.

[29] B. Krebs. "Beating JSON performance with Protobuf," Accessed: Jul. 21, 2025. [Online]. Available: https://auth0.com/blog/beating-json-performance-with-protobuf/.

[30] Thingsboard. "Thingsboard cloud documentation," Accessed: Jul. 21, 2025. [Online]. Available: https://thingsboard.io/docs/paas/.

[31] Temboo. "Temboo docs and guides," Accessed: Jul. 21, 2025. [Online]. Available: https://temboo.com/docs.

[32] SensorCloud. "Sensorcloud getting started," Accessed: Jul. 21, 2025. [Online]. Available: https://sensorcloud.com/welcome.

[33] Fiware. "Ngsi-v2 step-by-step," Accessed: Jul. 21, 2025. [Online]. Available: https://fiware-tutorials.readthedocs.io/en/latest/.

[34] OpenRemote. "Openremote: Get started with the free IoT platform," Accessed: Jul. 21, 2025. [Online]. Available: https://openremote.io/get-started-iot-platform/.

[35] Y.-W. Lin, Y.-B. Lin, and C.-Y. Liu, "AItalk: A tutorial to implement ai as IoT devices," *IET Networks*, vol. 8, no. 3, pp. 195–202, 2019. DOI: 10.1049/iet-net.2018.5182. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-net.2018.5182.

[36] P. P. Ray, "A survey of IoT cloud platforms," *Future Comput. and Informatics J.*, vol. 1, no. 1, pp. 35–46, 2016, ISSN: 2314-7288. DOI: 10.1016/j.fcij.2017.02.001.

[37] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma, "A gap analysis of Internet-of-Things platforms," *Comput. Communications*, vol. 89-90, pp. 5–16, 2016, Internet of Things Research challenges and Solutions, ISSN: 0140-3664. DOI: 10.1016/j.comcom.2016.03.015.

[38] H. Hejazi, H. Rajab, T. Cinkler, and L. Lengyel, "Survey of platforms for massive IoT," in *2018 IEEE Int. Conf. on Future IoT Technologies (Future IoT)*, 2018, pp. 1–8. DOI: 10.1109/FIOT.2018.8325598.

[39] Y.-B. Lin, H.-C. Tseng, Y.-W. Lin, and L.-J. Chen, "NB-IoTtalk: A service platform for fast development of NB-IoT applications," *IEEE Internet of Things J.*, vol. 6, no. 1, pp. 928–939, 2019. DOI: 10.1109/JIOT.2018.2865583.

[40] F. Pérez and B. E. Granger, "IPython: A system for interactive scientific computing," *Computing in Science Engineering*, vol. 9, no. 3, pp. 21–29, May 2007, ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.53.

[41] E. Rescorla. "Rfc8446: The transport layer security (TLS) protocol version 1.3," Accessed: Jul. 21, 2025. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8446.

[42] H. Ed. "Rfc6749: The OAuth 2.0 authorization framework," Accessed: Jul. 21, 2025. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6749.

[43] Y.-W. Lin, Y.-B. Lin, M.-T. Yang, and J.-H. Lin, "ArduTalk: An arduino network application development platform based on IoTtalk," *IEEE Systems J.*, vol. 13, no. 1, pp. 468–476, 2019. DOI: 10.1109/JSYST.2017.2773077.

[44] Y.-W. Lin, Y.-B. Lin, and T.-H. Yen, "SimTalk: Simulation of IoT applications," *Sensors*, vol. 20, no. 9, 2020, ISSN: 1424-8220. DOI:

10 . 3390 / s20092563. [Online]. Available: https://www.mdpi.com/1424-8220/20/9/2563.

[45] Y.-B. Lin, L.-K. Chen, M.-Z. Shieh, Y.-W. Lin, and T.-H. Yen, "CampusTalk: IoT devices and their interesting features on campus applications," *IEEE Access*, vol. 6, pp. 26 036–26 046, 2018. DOI: 10.1109/ACCESS.2018.2832222.

[46] Y.-B. Lin, S.-K. Tseng, T.-H. Hsu, and C. D. Tseng, "HouseTalk: A house that comforts you," *IEEE Access*, vol. 9, pp. 27 790–27 801, 2021. DOI: 10.1109/ACCESS.2021.3058364.

[47] L.-Y. Zhang, H.-C. Lin, K.-R. Wu, Y.-B. Lin, and Y.-C. Tseng, "FusionTalk: An IoT-based reconfigurable object identification system," *IEEE Internet Things J.*, vol. 8, no. 9, pp. 7333–7345, 2021. DOI: 10.1109/JIOT.2020.3039518.

[48] W.-L. Chen et al., "AgriTalk: IoT for precision soil farming of turmeric cultivation," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5209–5223, 2019. DOI: 10.1109/JIOT.2019.2899128.

[49] Y.-B. Lin and H.-C. Tseng, "FishTalk: An IoT-based mini aquarium system," *IEEE Access*, vol. 7, pp. 35 457–35 469, 2019. DOI: 10.1109/ACCESS.2019.2905017.

[50] W.-E. Chen, Y.-B. Lin, and L.-X. Chen, "PigTalk: An AI-based IoT platform for piglet crushing mitigation," *IEEE Trans. on Industrial Informatics*, vol. 17, no. 6, pp. 4345–4355, 2021. DOI: 10.1109/TII.2020.3012496.

[51] Y.-W. Lin, Y.-B. Lin, C.-Y. Liu, J.-Y. Lin, and Y.-L. Shih, "Implementing AI as cyber IoT devices: The house valuation example," *IEEE Trans. Ind. Inform.*, vol. 16, no. 4, pp. 2612–2620, 2020. DOI: 10.1109/TII.2019.2951847.

[52] H.-H. Chen, Y.-B. Lin, I.-H. Yeh, H.-J. Cho, and Y.-J. Wu, "Prediction of queue dissipation time for mixed traffic flows with deep learning," *IEEE*

Open J. of Intelligent Transportation Systems, vol. 3, pp. 267–277, 2022. DOI: 10.1109/OJITS.2022.3162526.

[53] Y.-B. Lin, C.-C. Cheng, and S.-C. Chiu, "Musictalk: A microservice approach for musical instrument recognition," *IEEE Open J. of the Comput. Society*, vol. 5, pp. 612–623, 2024. DOI: 10.1109/OJCS.2024.3476416.

[54] T. Alam, "Cloud-based IoT applications and their roles in smart cities," *Smart Cities*, vol. 4, no. 3, pp. 1196–1219, 2021, ISSN: 2624-6511. DOI: 10.3390/smartcities4030064.

[55] A. W. Services. "AWS IoT," Accessed: Jul. 21, 2025. [Online]. Available: https://aws.amazon.com/IoT/.

[56] Microsoft. "Microsoft Azure IoT," Accessed: Jul. 21, 2025. [Online]. Available: https://azure.microsoft.com/en-us/solutions/IoT.

[57] S. Li, L. Xu, and S. Zhao, "The Internet of Things: A survey," *Inf. Syst. Front.*, vol. 17, Apr. 2014. DOI: 10.1007/s10796-014-9492-7.

[58] S.-R. Yang, Y.-C. Lin, P. Lin, and Y. Fang, "AIoTtalk: A SIP-based service platform for heterogeneous artificial intelligence of things applications," *IEEE Internet Things J.*, vol. 10, no. 16, pp. 14 167–14 181, 2023. DOI: 10.1109/JIOT.2023.3265674.

[59] Y.-C. Liang, K.-R. Wu, K.-L. Tong, Y. Ren, and Y.-C. Tseng, "An exchange-based AIoT platform for fast AI application development," in *Proc. 19th ACM Q2SWinet '23*, , Montreal, Quebec, Canada, pp. 105–114, ISBN: 9798400703683. DOI: 10.1145/3616391.3622770.

[60] Nabto. "Nabto Edge Documentation," Accessed: Jul. 21, 2025. [Online]. Available: https://docs.nabto.com/developer/guides.html.

[61] K. L. Tong, K.-R. Wu, and Y.-C. Tseng, "The Device–Object pairing problem: Matching IoT devices with video objects in a multi-camera environment," *Sensors*, vol. 21, p. 5518, Aug. 2021. DOI: 10.3390/s21165518.

[62] PYTHON. "Python," Accessed: Jul. 21, 2025. [Online]. Available: https://www.python.org/.

[63] Free Code Camp. "Interpreted vs compiled programming languages: What's the difference?" Accessed: Jul. 21, 2025. [Online]. Available: https://www.freecodecamp.org/news/compiled-versus-interpreted-languages/.

[64] R. Rivest. "The MD5 message-digest algorithm," Accessed: Jul. 21, 2025. [Online]. Available: https://www.ietf.org/rfc/rfc1321.txt.

[65] G. Jocher, A. Chaurasia, and J. Qiu. "Ultralytics YOLO." version 8.0.0, Accessed: Jul. 21, 2025. [Online]. Available: https://github.com/ultralytics/ultralytics.

[66] N. Wojke and A. Bewley, "Deep cosine metric learning for person re-identification," in *2018 IEEE Winter Conf. on Applications of Comput. Vis. (WACV)*, IEEE, 2018, pp. 748–756. DOI: 10.1109/WACV.2018.00087.

[67] A. Leick, L. Rapoport, and D. Tatarnikov, *GPS satellite surveying*, 4th ed. Wiley, 2015, p. 257.

[68] P. Teunissen and O. Montenbruck, *Springer Handbook of Global Navigation Satellite Systems*, en. Springer, May 23, 2017, ISBN: 9783319429267.

[69] H. No and C. Milner, "Machine learning based overbound modeling of multipath error for safety critical urban environment," in *Proc. 34th. ION GNSS+ 2021*, Oct. 13, 2021.

[70] S. Jada, M. Psiaki, S. Landerkin, S. Langel, A. Scholz, and M. Joerger, "Evaluation of PNT situational awareness algorithms and methods," in *Proc. 34th. ION GNSS+*, Oct. 13, 2021, pp. 816–833. DOI: 10.33012/2021.17935.

[71] W. Stock, R. T. Schwarz, C. A. Hofmann, and A. Knopp, "Survey on opportunistic PNT with signals from LEO communication satellites," *IEEE Commun. Surv. & Tutor.*, pp. 1–1, 2024. DOI: 10.1109/COMST.2024.3406990.

[72] J. Zidan, O. Alluhaibi, E. I. Adegoke, E. Kampert, M. D. Higgins, and C. R. Ford, "3D mapping methods and consistency checks to exclude GNSS multipath/NLOS effects," in *Proc. UCET*, 2020, pp. 1–4. DOI: 10.1109/UCET51115.2020.9205423.

[73] R. Sun, L. Fu, Q. Cheng, K.-W. Chiang, and W. Chen, "Resilient pseudorange error prediction and correction for GNSS positioning in urban areas," *IEEE Internet Things J.*, vol. 10, pp. 9979–9988, 2023.

[74] S. Schaer, G. Beutler, L. Mervart, M. Rothacher, and U. Wild, "Global and regional ionosphere models using the GPS double difference phase observable," in *Proc. IGS Workshop*, 1995, pp. 77–92.

[75] Z. Nie, P. Zhou, F. Liu, Z. Wang, and Y. Gao, "Evaluation of orbit, clock and ionospheric corrections from five currently available SBAS L1 services: Methodology and analysis," *Remote Sens.*, vol. 11, no. 4, 2019, ISSN: 2072-4292. DOI: 10.3390/rs11040411.

[76] L.-T. Hsu, "GNSS multipath detection using a machine learning approach," in *Proc. 20th ITSC*, 2017, pp. 1–6. DOI: 10.1109/ITSC.2017.8317700.

[77] A. Elango, S. Ujan, and L. Ruotsalainen, "Disruptive GNSS signal detection and classification at different power levels using advanced deep-learning approach," *Proc. ICL-GNSS*, pp. 1–7, 2022.

[78] P. Borhani-Darian, H. Li, P. Wu, and P. Closas, "Detecting GNSS spoofing using deep learning," en, *EURASIP J. Adv. in Sig. Pr.*, vol. 2024, no. 1, Jan. 18, 2024, ISSN: 1687-6180. DOI: 10.1186/s13634-023-01103-1.

[79] J. Li, X. Liu, W. Zhang, M. Zhang, J. Song, and N. Sebe, "Spatio-temporal attention networks for action recognition and detection," *IEEE Trans. on Multimedia*, vol. 22, no. 11, pp. 2990–3001, 2020. DOI: 10.1109/TMM.2020.2965434.

[80] H. Yao, X. Tang, H. Wei, G. Zheng, and Z. Li, "Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction," *Proc. of the*

AAAI Conf. on Artificial Intelligence, vol. 33, no. 01, pp. 5668–5675, Jul. 2019. DOI: 10.1609/aaai.v33i01.33015668.

[81] Y. Zhao, F. Shen, G. Xu, and G. Wang, "A spatial-temporal approach based on antenna array for GNSS anti-spoofing," *Sensors*, vol. 21, no. 3, 2021, ISSN: 1424-8220. DOI: 10.3390/s21030929.

[82] E. J. Keogh and M. J. Pazzani, "Derivative dynamic time warping," in *Proc. SDM*, 2001.

[83] J. A. Donenfeld. "Wireguard: Fast, modern, secure VPN tunnel," www.wireguard.com, Accessed: Mar. 30, 2024. [Online]. Available: https://www.wireguard.com/#about-the-project.

[84] S. Khotijah. "K-means clustering of iris dataset," Accessed: Jul. 21, 2025. [Online]. Available: https://www.kaggle.com/code/khotijahs1/k-means-clustering-of-iris-dataset.