network

MDPI

Review

State of the Art in Internet of Things Standards and Protocols for Precision Agriculture with an Approach to Semantic Interoperability

Eduard Roccatello [1], Antonino Pagano [2,3,*], Nicolò Levorato [4] and Massimo Rumor [4,*]

[1] 3DGIS, 35138 Padua, Italy; eduard.roccatello@3dgis.it
[2] Department of Engineering, University of Palermo, 90128 Palermo, Italy
[3] CNIT (National Inter-University Consortium for Telecommunications), 90128 Palermo, Italy
[4] Department of Information Engineering, University of Padua, 35138 Padua, Italy
* Correspondence: antonino.pagano@unipa.it (A.P.); massimo.rumor@unipd.it (M.R.)

Abstract: The integration of Internet of Things (IoT) technology into the agricultural sector enables the collection and analysis of large amounts of data, facilitating greater control over internal processes, resulting in cost reduction and improved quality of the final product. One of the main challenges in designing an IoT system is the need for interoperability among devices: different sensors collect information in non-homogeneous formats, which are often incompatible with each other. Therefore, the user of the system is forced to use different platforms and software to consult the data, making the analysis complex and cumbersome. The solution to this problem lies in the adoption of an IoT standard that standardizes the output of the data. This paper first provides an overview of the standards and protocols used in precision farming and then presents a system architecture designed to collect measurements from sensors and translate them into a standard. The standard is selected based on an analysis of the state of the art and tailored to meet the specific needs of precision agriculture. With the introduction of a connector device, the system can accommodate any number of different sensors while maintaining the output data in a uniform format. Each type of sensor is associated with a specific connector that intercepts the data intended for the database and translates it into the standard format before forwarding it to the central server. Finally, examples with real sensors are presented to illustrate the operation of the connectors and their role in an interoperable architecture, aiming to combine flexibility and ease of use with low implementation costs.

Keywords: Internet of Things (IoT); wireless sensor networks; precision agriculture; IoT standards; IoT protocols; interoperability; sensor integration; connector devices; interoperable IoT architecture; OGC SensorThings

Academic Editor: Patrick Seeling

Received: 10 January 2025
Revised: 28 February 2025
Accepted: 13 March 2025
Published: 21 April 2025

Citation: Roccatello, E.; Pagano, A.; Levorato, N.; Rumor, M. State of the Art in Internet of Things Standards and Protocols for Precision Agriculture with an Approach to Semantic Interoperability. *Network* **2025**, *5*, 14. https://doi.org/10.3390/network5020014

1. Introduction

The growing scarcity of arable land due to various human and climatic factors, coupled with the growing global demand for food, underscores the need for sustainable and productive agricultural management [1,2]. To address these challenges, the Internet of Things (IoT) is recognized as a powerful tool, owing to its potential to make agriculture more data-driven, leading to timely, economical, and efficient farming systems [3].

Precision agriculture solutions, which are becoming increasingly prevalent, involve management practices based on spatial measurements that utilize IoT technologies [4,5]. For example, fertilizers can be applied only where necessary, and water management can

be automated to minimize waste while adapting to the specific needs of crops [6,7]. This is in addition to monitoring the parameters of the crops themselves [8,9].

Smart agriculture is evolving beyond these applications, enhancing the use of spatial data in real-time events. Farmers can respond quickly to significant changes in weather, humidity, and air quality, as well as the health of crops and soil [10]. This is facilitated by systems that integrate sensors, intelligent agricultural vehicles, drones, and autonomous robots [11]. In these applications, the IoT supports the documentation and supervision of various activities, along with the traceability of products through systems for data analysis, visualization, and management. However, one of the main challenges in the design of IoT systems is addressing the need for interoperability between devices. It is estimated that interoperability could increase the potential market value of IoT by 40% [12]. Many complex IoT systems, such as smart agriculture, require the coordination of different IoT devices that often use different data formats [13]. The goal is to achieve *semantic interoperability*, defined as the ability to represent data in a form whose meaning is independent of the application that generates or uses it [14,15].

In digital agriculture systems, most of the data exchanged between machines and heterogeneous services consist of readings from sensors and commands for actuators [16]. These data are generally numerical and specific to the type of device and the measurement domain, for example, a temperature reading, sometimes accompanied by additional information such as the nature of the measurements and the units of measure [17]. Alternatively, they may be control strings used to modify parameters or activate actuators [18]. Receiving-side applications must be able to understand and interpret the received data in order to process them and act accordingly [15]. In other words, the two IoT endpoints involved in communication must ensure *semantic interoperability* in order to interact correctly. This implies that the data must be formatted in such a way that they can be interpreted identically by both ends involved in the message exchange. Indeed, in the field of precision agriculture, there is currently no dedicated and universally shared standard. There are various IoT standards that could meet the needs of this field by modelling data exchange between different sensors [19]. However, at present, the market offers a series of heterogeneous proprietary devices, each with its own specific mode of communication.

The contribution of this research is, in the first phase, to analyse the available standards and evaluate the pros and cons of each, in order to choose the one that best meets the requirements in the field of precision agriculture. In the second phase, a new interoperable architecture is proposed to overcome the challenges associated with the standards and protocols present in the smart agriculture sector.

The remaining sections of this article are organized as follows: Section 2 provides the background and highlights the contributions of this paper. Section 3 reviews the IoT standards and protocols used in precision agriculture. Section 4 introduces an interoperable IoT architecture designed for precision agriculture applications. Section 5 describes a use case implemented using the Open Geospatial Consortium (OGC) SensorThings standard. Section 6 details the implementation of the SensorThings connectors, while Section 7 discusses the proposed architecture and its implications for agriculture. Finally, Section 8 concludes the article with a summary of the findings.

2. Background and Paper Contribution

2.1. IoT for Precision Agriculture

An IoT platform is an abstract and formal system to manage certain types of IoT object, often including their properties, relationships, and the operations that can be performed on them [20]. IoT platforms typically include (i) the designation of the type of physical object, that is, what it is and what it does, for example, a humidity sensor; (ii) the data

formats and how they are represented and interpreted, for example, values in the form of a string, number, integer, or float; (iii) interactions and access methods, i.e., how to access the object to obtain its status and data or to activate intrinsic functions and outputs; (iv) metadata describing the attributes of objects and the context in which the data are acquired; and finally, (v) links to other objects, used to represent structural connections and the composition of objects.

In precision agriculture, critical data are collected in real time during the agricultural production process and transmitted through wireless networks through the M2M (Machine-to-Machine) support platform [21]. This allows real-time environmental data to be obtained using a network-based protocol, both IP or non-IP based, with an cellular-based SMS fallback, helping farmers make timely decisions, improve production, and respond more quickly to changes and unpredictable events [22,23]. In essence, the adoption of IoT technologies has the potential to improve agriculture in many ways. Some of the possibilities that these technologies offer are as follows:

- *Data collected from sensors*, such as weather conditions, soil quality, crop growth trends, and livestock health, can be used to monitor the general state of operations, as well as staff performance and equipment efficiency.
- *Improved control over internal processes*, and consequently reduced production risks. The ability to predict production output allows for better product distribution planning [24].
- *Cost Optimization and waste reduction* due to increased control over production [25]. By detecting anomalies in crop growth or livestock health, it is possible to mitigate the risks associated with crop loss.
- *Greater business efficiency* through process automation. By using smart sensors, multiple processes during the production cycle can be automated, e.g., irrigation, fertilization, or pest control.
- *Improvement in product quality and volume.* Thanks to the use of new technologies, it is possible to achieve better control over the process and maintain a higher quality of the crops and growth capacity through automation [26].

2.2. Heterogeneous Sensors in Precision Agriculture

To achieve the objectives and benefits outlined in the previous subsection, it is essential that different working environments are monitored using IoT devices. Often, the use of IoT devices involves placing heterogeneous sensors at strategic points dedicated to monitoring specific parameters, such as soil moisture levels, temperature, humidity, and crop health.

Temperature sensor: Temperature sensors are fundamental in two key categories of smart agriculture: environmental condition monitoring. Ice wine harvesting, for instance, is conducted within a specific temperature range, typically when ambient temperatures fall between −10 °C and −12 °C at the beginning of the harvest season [27]. For the ice wine industry, highly accurate sensors for temperature and humidity, coupled with reliable predictive temperature forecasting, are indispensable [28]. In addition to monitoring the environmental conditions of the physical space, temperature sensors are vital for numerous applications in the monitoring and management of smart agricultural resources.

Soil moisture sensor: Soil moisture plays a crucial role in defining the water availability for crops. Abnormal soil moisture levels, whether too high or too low, can disrupt the healthy growth of crops above ground. Maintaining an appropriate level of soil moisture is essential to balance root water uptake and leaf transpiration, which together support the optimal development of crop roots [29,30].

Soil Composition Sensor: Nutrient availability is just as vital for plant growth. Indeed, to maximize a plant's growth potential and ensure high-yield production, it is essential

to develop a thorough and quantitative understanding of the soil conditions that support agricultural output [30]. The soil morphology and the presence of fertilizers can be analysed by measuring electrical conductivity, volumetric water content, soil water potential, and oxygen levels [31]. IoT-enabled sensors, such as Nitrogen, Phosphorus, and Potassium (NPK) sensors, are critical for soil analysis in precision agriculture. These sensors provide critical feedback on nutrient deficiencies in the soil or the presence of undesirable chemicals. Additionally, they enable smart agriculture to track soil pH and nutrient level fluctuations on daily, weekly, monthly, and annual scales, offering valuable insights to further educate the agricultural sector [28]. Discussing the integration of data from these sensors into the proposed architecture highlights its potential to enhance comprehensiveness and applicability in supporting precision agriculture practices.

Light Sensor: The application of light sensors in greenhouse plantations can help growers accurately understand the laws of the duration of sunlight, the point of saturation of light, and the point of compensation of light for plant growth [32]. This enables them to adjust their light preferences through manual control technology to manage and enhance the scientific growth of crops for higher yields. Crops constantly absorb CO_2 from the atmosphere for photosynthesis, a process that produces nutrients essential for their growth and development. Research indicates that higher atmospheric carbon dioxide levels substantially boost the efficiency of photosynthesis in plants [33,34].

Atmospheric Parameters: Agricultural weather stations are autonomous units that are placed in various locations within growing fields. These stations are equipped with a combination of sensors suitable for the local crops and climate [35]. Data such as air temperature, soil temperature at different depths, precipitation, leaf moisture, chlorophyll content, wind speed, dew point, wind direction, relative humidity, solar radiation, and atmospheric pressure are collected and logged at set intervals. This information is then transmitted wirelessly to a central data logger at regular intervals. Due to their portability and falling costs, weather stations have become an appealing option for farms of various sizes [36].

2.3. Research Challenge and Problem Formulation

However, in the collection of data from these sensors, a problem arises: when the system is composed of heterogeneous devices, there is currently no shared communication standard among all devices for data collection and communication. The current market offers multiple IoT devices for precision agriculture, some of which can be managed using proprietary software provided by the manufacturer, while others can be programmed by the user. Each device has its own method for collecting and transmitting data, resulting in a set of heterogeneous data sent with different specifications, making it difficult for the end user to consult. When a sensor sends data, especially in the IoT domain, the transmitted information is compressed to minimize resource consumption. Part of the information necessary for understanding the data (such as units of measurement, accuracy, type of measurement, etc.) is left implicit because, using the standard system provided by the sensor manufacturer, it is processed during the reception and data processing stages. If sensors from different manufacturers were to be used, the result would be a set of incomplete or undecipherable data that would be of little use for analysis. To better understand the problem, we analyse an example payload from a sensor, in this case, the SensorData sensor produced by Digital Matter [37], to which a sensor for measuring atmospheric pressure is connected. The payload sent by the sensor is as follows:

"*AF*4*D*3276*CE*5*F*3334801260618231006*A*18"

Without the use of the software provided by the manufacturer, it would be impossible to decipher the information contained in the string, which represents the geographical position of the sensor and the value of two measurements taken.

In this research, the problem is addressed by proposing a system architecture based on a standard that can be shared among the various connected IoT devices, in order to create a network that can accommodate different types of device without compromising the understanding and analysis of the final data. Furthermore, in order to propose an interoperable architecture, an intensive analysis was carried out of existing standards and protocols to be adopted in agricultural contexts. This analysis is defined by the following Research Questions (RQs):

- **RQ1**: *What are the requirements to be met by IoT standards for precision agriculture?*
- **RQ2**: *Which IoT protocols are most suitable for precision agriculture?*
- **RQ3**: *What is the most suitable IoT standard to use in agriculture?*
- **RQ4**: *What are the requirements and challenges for building an interoperable architecture for precision agriculture?*
- **RQ5**: *Which open source software is best suited to implement an interoperable architecture in agriculture?*

These Research Questions (RQs) aim to explore and define technological solutions to enhance the adoption of the Internet of Things (IoT) in precision agriculture. Specifically, RQ1 examines the key requirements that IoT standards must fulfil in this context, while RQ2 identifies the most suitable IoT protocols to optimize the efficiency of agricultural applications. Subsequently, RQ3 focuses on determining the most appropriate IoT standard to address the specific needs of agriculture. RQ4 addresses the challenges and requirements for designing an interoperable architecture, facilitating integration among diverse technologies and systems. Finally, RQ5 investigates the most suitable open-source software to implement such an architecture, emphasizing the role of open software in reducing implementation costs. This is a crucial factor in overcoming one of the primary barriers to large-scale IoT adoption in agriculture: high development costs. By lowering these costs, advanced technologies become more accessible, even for farmers with limited resources.

3. IoT Standards and Protocols for Precision Agriculture

3.1. IoT Standards for Precision Agriculture (RQ1)

An IoT standard defines protocols and guidelines for middleware layers, which act as intermediaries between IoT applications and networks. These middleware implementations provide application interfaces linked to the network, enabling connected nodes to interact seamlessly [38]. IoT standards have four main design objectives: (i) reduce development time and bring IoT solutions to market faster; (ii) reduce the apparent complexity of implementing and operating an IoT network; (iii) enhance the portability and interoperability of applications; and (iv) improve functionality, reliability, and maintainability. Given the wide range of communication methods used by IoT devices, it is unsustainable for applications to manage every existing method. Standards aim to reduce this complexity by proposing a common model for transferring data to higher communication layers. Standards organizations, which are not formal regulatory bodies but are more generally referred to as Standards Definition Organizations (SDOs), help achieve the aforementioned goals through the standardization of interconnection frameworks, message-passing interfaces, and the data definitions used by applications [39]. Frameworks simplify IoT networks by creating an abstraction layer over IoT device networks, hiding much of the underlying complexity while exposing standard interfaces and functions, thus facilitating *interoperability* between devices [40]. The framework provides a semantic description of nodes, objects,

and interactions between IoT devices, enabling IoT network designers to focus solely on the semantics of node interaction rather than connectivity details. All the illustrated standards model physical objects, such as sensors and actuators, into software representations that servers hosting instances of various resources can use to visualize their states. An IoT standard acts as a layer between the services offered by the application and the network on which it operates [41]. In Table 1, we can see how the IoT standard is positioned as the top layer within the TCP/IP communication model [42]. Thanks to the services provided by the server, IoT clients use the standard to create their own representation instances and interpret the data responses from these instances. The use of a common specification results in both syntactic and *semantic interoperability* in interpreting the meaning of data.

Table 1. IoT standards within the TCP/IP stack.

Application	**IoT Standard**
	HTTP MQTT
Transport	TCP UDP
Internet	IPV4/V6 6LoWPAN
Network access	SigFox LoRaWAN
	Device

3.1.1. IoT Standard Requirements for Precision Agriculture

The IoT devices used in precision agriculture monitoring systems often have different technical specifications, set by manufacturers, which allow interaction through sending and receiving messages regardless of the physical communication protocol used [10]. However, these specifications introduce challenges within the system. In addition to having to understand the communication methods and syntax of each type of device, it is also necessary to account for any variations in the technical specifications introduced by the manufacturer to resolve specific issues or support new functionalities [3]. To achieve seamless communication, independent of the types of devices used, the proposed system introduces a logical layer of **interoperability** at the data storage level, allowing data originating from sensors to be effectively utilized. The fundamental requirements for an IoT sensor standard, that must be met to implement the logical layer introduced in the project, are litest below:

- Lightweight data flow
- Simplicity and ease of use
- Easily adaptable based on use cases
- Open platform
- Support for geolocation
- Support for various types of data
- Availability of open-source implementations
- Suitable for the needs of the precision agriculture sector

3.1.2. Analysed IoT Standards

The state of the art in standards offers numerous solutions that provide a comprehensive framework for communication between IoT devices while meeting the identified requirements [39]. In this research, an analysis of these solutions was performed, evaluating them based on market adoption, origin, and requirements for agriculture. The key features of the analysed IoT standards are summarised in Table 2. In relation to the requirements for precision agriculture, the analysed IoT standards offer several complementary features. IPSO stands out for its light data flow and simplicity, making it ideal for low-power devices

in remote environments. Haystack, due to the flexibility of user-defined tags, is highly adaptable but can be complex to configure. OCF combines advanced functionality with good simplicity, offering a balanced solution for more interactive applications. SensorThings, with its native support for geolocation and various data types, is particularly well suited to complex scenarios with advanced analysis needs, although it may be overkill for simpler applications. The choice of standard will therefore depend on the specific context and implementation requirements.

Table 2. Comparison of IoT standards for precision agriculture

	IPSO	Haystack	OCF	SensorThings
Base Model	LwM2M	-	-	OData OASIS
Object Modeling	Through predefined models	Through user-defined tags	Through predefined resource types and interfaces	Through standard-defined entities
Object Properties	Defined by the model and identified by an ID	Defined by the tags used	Defined by resource types with customizable additions	Defined by the entity type
Geolocation	Expressed as an object resource	Expressed as a property through tags	Defined by resource type	Native through entities

IP for Smart Objects (IPSO)

The goal of IPSO is to establish a common design model and data model capable of providing high-level interoperability between sensors and software applications for the IoT [43]. In the IPSO standard, the structure representing the object includes a definition of the object type and a collection of resources formally named by the standard. Each resource represents a simple value that describes a specific function of the object. Objects and resources are mapped through URI paths, with each URI uniquely identifying an object [44]. The structure is composed of three 16-bit unsigned integers separated by the character "/" in the following form:

$$ObjectID/InstanceID/ResourceID$$

In IPSO, objects are typed containers defining the semantic type of their instances, which hold resources representing observable properties. For example, a temperature sensor URI like "3303/0/5700" uses "3303" as the object ID, "0" as the instance ID, and "700" as the resource ID for the sensor value. Objects represent single points of measurement or control, while resources define specific properties like current, minimum, or maximum values. Semantic interoperability is achieved through standardized object and resource identifiers, enabling consistent representation of data, such as temperature readings or limits. IPSO defines over 50 smart object types, including sensors (e.g., temperature, humidity, barometers), input/output devices (digital, analog, and generic I/O), presence detectors, accelerometers, power monitors, load and lighting controls, actuation devices, and set points.

Haystack

The Haystack Project focuses on developing solutions for the semantic modelling of data related to smart devices, including building equipment systems, automation and control devices, sensors and detection systems, and building automation systems [45]. Haystack provides a common methodology for defining metadata through a series of tags, expressed as name–value pairs, which are used to describe attributes and properties associated with system entities. It defines a common vocabulary in the form of tag libraries created by the community. These tags have fixed names, and the libraries allow for the definition of the attributes or properties they describe. Although Haystack does not specify

which or how many tags should be used to describe a specific piece of data, it dictates how they should be named when employed [46]. In Haystack, an endpoint called a "point" can be a sensor, a command to an actuator (cmd), or a set point (sp) that can be written. For instance, a point representing a temperature sensor may have some or all of the following tags associated with it, among others:

$$temp, sensor, unit, air, discharge, equipRef, siteRef$$

All these are Haystack tag names that must be named as specified in the appropriate tag library if used. The philosophy behind Haystack is that system designers/installers have the freedom to annotate whatever they consider useful, while applications use and interpret the tags they find relevant for their purpose in order to understand the context. Haystack payload data serialization is supported in formats like JSON, CSV (Comma-Separated Values), Zinc (similar to CSV but with more data types), and grids (two-dimensional tabular data structures).

Open Connectivity Foundation (OCF)

The Open Connectivity Foundation (OCF) establishes IoT standards, ensures device interoperability, and provides certification. Its framework covers information modelling, discovery, messaging, device management, security, and client–server interactions for data and actions [47]. In OCF, physical entities are represented as resources, following a resource and interaction model. Interactions use the CRUDN model (Create, Read, Update, Delete, Notify) aligned with the CoAP protocol. Each resource type has a specification detailing its behaviour for all possible interactions [48]. In OCF, every resource includes two required attributes: "if" (interface types) and "rt" (resource types), with variants detailed in the specification. Additional properties, expressed as name–value pairs, represent data and metadata. For instance, a sensor resource may include attributes like "value", "range", "unit", and "link", the latter enabling connections to other resources. The "rt" attribute specifies the type of physical device. OCF uses API specifications to define methods and interfaces for resources and JSON schema to describe payloads, properties, and data types [49].

OGC SensorThings

The Open Geospatial Consortium (OGC), consisting of over 500 organizations, develops open standards for geospatial data and IoT services, including the "SensorThings" standard for IoT sensors.

The OGC SensorThings API offers a unified, open, and geospatial method to connect IoT devices, data, and applications on the Web, designed for resource-limited devices [50]. It follows REST principles, uses JSON encoding, and supports the OASIS OData protocol, MQTT extension, and CoAP/HTTP protocols [51]. The SensorThings API provides an interoperable model for managing observations, especially for reconciling differences between diverse sensing systems, such as in situ and remote sensors. IoT devices are modelled as objects with multiple locations and DATASTREAMS, where each DATASTREAM monitors a property through a sensor and groups related observations. Observations focus on a FEATURE OF INTEREST, which represents a real-world property. The API offers two main functionalities: the Sensing section, which manages and retrieves observations and metadata from various IoT sensor systems, and the Tasking section, which standardizes the way IoT devices (e.g., sensors or actuators) are parameterized or tasked, similar to the OGC Sensor Planning Service (SPS) [52].

3.1.3. IoT Standard Selected for Precision Agriculture

A comparison of the advantages and disadvantages that each of the described standards would bring to the system if adopted has been carried out. The analysis considered both the system requirements identified earlier and the unique characteristics of each standard. In Table 3, the strengths and weaknesses for each IoT standard have been identified. By comparing the analysis of the standards with the requirements of the system for precision agriculture, the only one that fully meets every point is the OGC SensorThings standard, as shown in Table 4. As previously mentioned, the presented standards largely meet the requirements for precision agriculture, and each of them could be used within the proposed architecture without much difficulty. However, the decision was made to proceed with the OGC SensorThings standard because, in addition to meeting all the requirements, it has some intrinsic features that make it functional for creating an interoperable architecture.

Table 3. Advantages and disadvantages of IoT standards for precision agriculture

Ipso		**Haystack**	
Pros	**Cons**	**Pros**	**Cons**
Simple to use for simple objects [53]	Necessary to use the available IDs [43]	Customizable thanks to the free choice of tags [45]	Numerous tags make data filtering complicated [54]
Wide list of pre-modeled objects to choose from [43]	If none of the available IDs is sufficient to model the object, a request must be made to OMA to add a new ID [55]	Minimizes exchanged information, keeping only what is significant for the application [45]	Each device needs an arbitrary number of tags to describe it, chosen by the user
Each object belonging to the same category has the same properties, ensuring consistent data	For modelling more complex objects, composite objects must be used, complicating the understanding of the standard for less experienced users	Tags are named in a way that makes their purpose easy to understand [45] It is possible to choose tags from a predefined list or create new ones [45]	Risk of similar objects being modelled with different tags, making data inconsistent [54]
OCF		**SensorThing**	
Pros	**Cons**	**Pros**	**Cons**
Database modeling a wide range of devices [49]	Very complex and not intuitive, understanding a payload requires in-depth knowledge of the standard	Each entity has fixed properties, making installation easy and ensuring consistent data [51]	Limited customization
Possibility to add custom properties to objects [56]	The object being modeled must be associated with a predefined one from the database	Simple and intuitive [51]	Some properties may be redundant for certain devices [57]
Already supported and used by several companies in the market	Each object type has its own interface type Some object properties may require manual input, complicating the installation phase	Native support for geolocation [51] Native support for time and date of measurements [52] Based on ISO 19156 standard [51]	

Table 4. Standards comparison according to requirements for precision agriculture.

Features	**Ipso**	**Haystack**	**OCF**	**SensorThings**
Open source implementation	✓	✓	✓	✓
Lightweight data flow	✓	✓	✓	✓
Agile modeling		✓		✓
Ease of use		✓		✓
Support for diverse data	✓		✓	✓
Native geolocation support				✓

3.2. IoT Protocols in Precision Agriculture (RQ2)

In order to select the appropriate standard in precision agriculture, it is first necessary to understand the most widely used communication protocols for transmitting information in the IoT ecosystem [58]. Communication protocols formally define the formats and rules that must be followed for the transmission of digital messages. Various protocols exist, some of which have been specifically designed for environments with limited available resources, such as the sensors we have analysed. These protocols enable the generation of a consistent and understandable data flow from most network-connected devices. The most commonly used protocols in precision agriculture are described below.

3.2.1. IP Protocols Comparison

Table 5 summarizes and compares the communication IP protocols available for precision agriculture. The comparison highlights the strengths of each protocol for IoT applications in precision agriculture. HTTP [59], with its RESTful architecture and reliable TCP-based transport, is well-suited for web-based systems but has high overhead and lacks publish/subscribe communication support [60,61]. MQTT [15], with minimal overhead and a publish/subscribe model, is ideal for low-bandwidth and scalable IoT deployments but lacks REST compliance [62]. CoAP combines RESTful features with lightweight UDP transport, making it efficient for constrained environments while maintaining flexibility through reliable and unreliable communication options [15,63]. Based on the agricultural requirements outlined in Section 3.1.1, both MQTT and CoAP align well with the needs of precision agriculture, offering lightweight data flow, open platforms, and robust open-source implementations, whereas HTTP's traditional point-to-point architecture and inefficiency make it less suitable. Moreover, an extension of MQTT, namely MQTT-SN, proves particularly interesting for agricultural applications, as it has been specifically designed to handle lightweight messaging in resource-constrained environments, common in precision agriculture. Features such as gateway support, efficient message encoding, and reduced overhead make MQTT-SN ideal for sensor networks where energy and bandwidth are valuable resources [64,65].

Table 5. Comparison of IP communication protocols [66].

	HTTP	**MQTT**	**CoAP**
General	HTTP relies on TCP as its transport protocol. It uses GET, POST, PUT, and DELETE to specify the intended action on the targeted resource.	MQTT enables two-way communication between clients via a central server. It employs a publish/subscribe messaging model that supports one-to-many message distribution and separates the applications.	CoAP offers two levels of communication: a reliability layer and a REST layer, which imitates HTTP with methods like POST, PUT, GET, and DELETE. The PATCH and FETCH methods were introduced to CoAP in RFC 8132.
Transport	TCP	TCP	UDP
REST Architecture Support	Yes	No	Yes
Pub/Sub Support	No	Yes	No
Payload Structure	The header overhead is quite high due to the use of human-readable text instead of binary encoding. Also, URI parameters need to be encoded in Base64. With pipelining and keep-alive, the TCP socket stays open, minimizing the number of TCP messages for connection setup and teardown.	MQTT has minimal protocol overhead (the fixed-length header is just 2 bytes).	CoAP has a flexible header length, with a minimum of 4 bytes. It uses binary encoding for both the header and the options within it.

3.2.2. Non-IP Protocols

An Internet connection that requires minimal resource expenditure is a critical issue for low-power embedded devices used within sensors. Conventional wireless communication technologies are inadequate when considering coverage, energy consumption, and cost. Low Power Wide Area Networks (LPWANs) aim to address these issues, making them scalable and adaptable for large-scale deployments of low-power devices [67]. LPWANs are expected to operate at low data transmission rates to cover distances of several kilometres in both densely populated urban areas and suburban regions. Technologies such as LoRaWAN and SigFox, as examples of sub-GHz communication technologies, successfully provide these functionalities. LPWANs have several common characteristics that distinguish them from traditional communication networks: (i) low power consumption, as the network of end nodes should consume minimal energy; (ii) reduced communication costs; (iii) integration of geolocation capabilities; and (iv) robust modulation to prevent radio networks from becoming congested in high-density urban areas [68].

SigFox

SigFox is a proprietary communication protocol designed for IoT wireless networks, requiring a paid license managed nationally. It belongs to the LPWAN family of technologies [69] and is used for low-volume data transmission (from a few bytes to hundreds of kilobytes), with excellent range (tens of kilometres) and low current consumption (in the mA range per transmission). SigFox uses D-BPSK (Differential Binary Phase-Shift Keying) modulation, with a fixed bandwidth of 100 Hz and transmission rates of 100 bps in Europe or 600 bps in the USA, within an unlicensed spectrum below 1 GHz (868 MHz for Europe, 915 MHz for the USA) [70]. The benefits of D-BPSK include efficient spectrum usage and simple implementation, along with the ability to use low-cost components due to the low bit rate. The gateway receiver is highly sensitive, able to demodulate signals close to the noise threshold without any coding layer.

LoRaWAN

LoRa (long range) is a low-power Wide Area Network (WAN) technology owned by the "LoRa Alliance". It operates in the unlicensed spectrum across multiple bands ranging from 169 to 430 MHz, with 868 MHz (Europe) and 915 MHz (North America) being the most common. Its range extends up to 10 km in open areas and closer to 1 km in densely populated spaces like cities [71]. The bandwidth can reach up to 50 Kbps, depending on the distance and transmission power [15,72]. The network is effectively half-duplex, meaning it supports bidirectional communication, but only in one direction at a time. Consequently, senders and receivers must coordinate to switch directions. The LoRa specification consists of PHY and MAC layers that can support applications with raw data. It defines three types of nodes: class A (Bidirectional End Devices), class B (Bidirectional End Devices with Scheduled Reception Slots), and class C (Bidirectional End Devices with Maximum Reception Slots). Finally, LoRa employs spread spectrum modulation and an Adaptive Data Rate (ADR) algorithm, enabling the data rates on each link to adjust according to signal strength and radio power [10]. The data rate can vary between 0.3 and 27 Kbps.

Table 6 compares WiFi and Bluetooth with the most widely used LPWAN technologies in the modern market. In the field of precision agriculture, the most prevalent technologies are LoRaWAN and SigFox, two proprietary protocols designed for IoT devices that manage to reconcile coverage over distances on the order of kilometers with low energy consumption.

Table 6. Comparison of IoT protocols [67,73].

	SigFox	NB-IoT	LoRaWAN	WiFi	Bluetooth
Standard	SigFox	3GPP	LoRa Alliance	IEEE 802.11	Bluetooth SIG
Modulation	BPSK	QSPK	CSS	DSSS, OFDM	GFSK
Frequencies	ISM: 433 MHz, 868 MHz, 915 MHz	Licensed under LTE	ISM: 433 MHz, 868 MHz, 915 MHz	ISM: 2.4 GHz, 5 GHz	2.4 GHz
Bandwidth	100 Hz	200 kHz	125 kHz, 250 kHz	20 MHz, 40 MHz, 80 MHz, 160 MHz	1 MHz
Coverage	10–40 km	2–20 km	1–10 km	10–100 m	10–100 m
Duty Cycle	144 packets/day	Unlimited	1%	Unlimited	Unlimited
Max Data Rate	100 bps	200 kbps	50 kbps	Gbps	2 Mbps
Power Consumption	Low	Low	Low	High	Low
Security	Low	High	High	Low-High	Low-High

3.2.3. Communication Between Non-IP and IP Protocols

As of today, end-to-end LPWAN solutions are not widely available or always feasible, and they often requires working with multiple vendors to acquire nodes, gateways, backend servers, and other parts of the ecosystem separately. While this provides significant flexibility in applications, it places the burden on developers to implement the network by managing packetization, downlink control, multicast, and more. Low-Power Wide-Area Networks (LPWANs), such as LoRaWAN and SigFox, are a set of long-range communication technologies designed for IoT implementation. A low-power WAN creates a point-to-point connection between a device and a server. Many LPWAN applications and implementations are closely related to the underlying LPWAN technology [74]. As a result, entering new markets and integrating additional technologies often requires the time-consuming re-engineering of both the cloud and device applications.

These emerging network technologies do not follow the IP-based Internet model [75]. They are not interoperable with each other and are difficult to integrate with existing architectures and information systems. Each IoT project typically involves first selecting a technology, then adjusting the choice of devices and platforms, and managing integration with the existing architecture. This has led the market to remain somewhat siloed by technology. In the Internet world, *interoperability* and multiconnectivity are native properties. In fact, the Internet Protocol (IP) is open, allowing every user to operate around a universal reference standard. The Internet uses a layered model, where each layer implements a protocol and abstracts the complexity of the others. This hourglass model isolates the application from the network, saving significant design effort for every player in the value chain. At its core, IP provides optimized services such as addressing and routing, services that any application can inherit without needing to reimplement everything. For practical purposes, in this section, we will use LoRaWAN technology as an example to demonstrate how a non-IP protocol communicates with an IP-based protocol.

Due to strict bandwidth limitations, LoRaWAN is unable to carry the heavy overhead of the IP protocol, so it implements the third layer of the OSI network model (Table 7), which is normally occupied by IP, in its own way [76]. Specifically, LoRaWAN handles the implementation of the second and third layers of the OSI network model (the data link and network layers). LoRaWAN's implementation of the third layer means that a LoRaWAN network cannot communicate with an IP network without the help of a gateway acting as a "bridge" between the two networks. LoRaWAN gateways are radio modules with a LoRa concentrator, allowing them to receive data packets. A LoRaWAN gateway is mainly used to transmit IoT device data to the cloud. The LoRaWAN technology itself is

only used for communication between end devices and their connected gateways. These gateways include an operating system that manages the forwarding of LoRa packets to the target network. While the data rate between the end device and the LoRaWAN gateway is relatively low, this is a necessary trade-off to ensure long battery life and extended radio range.

Table 7. Open Systems Interconnection (OSI) model

Application	From network process to application
Presentation	Data representation and encryption
Session	Inter-host communication
Transport	End-to-end connection and reliability
Network	Path determination and logical addressing
Data link	Physical addressing (MAC and LLC)
Physical	Media, signal, binary transmission

The gateway is then linked to a high-bandwidth IP network, such as WiFi, Ethernet, or cellular, enabling end-to-end connectivity between LoRaWAN end devices and the application server [77]. Figure 1 shows how communication occurs between a LoRaWAN network and the Internet, from the IoT nodes to the application server in precision agriculture. It shows how end devices, such as low-power IoT sensors, collect data from the environment and communicate with gateways. These gateways, equipped with LoRa concentrators, receive transmissions from the end devices and forward the data to the cloud [78,79]. The network server acts as the central hub, routing messages between the end devices and the application layer. It ensures that data packets reach the correct destination, facilitating communication in both directions: from devices to applications, and vice versa.

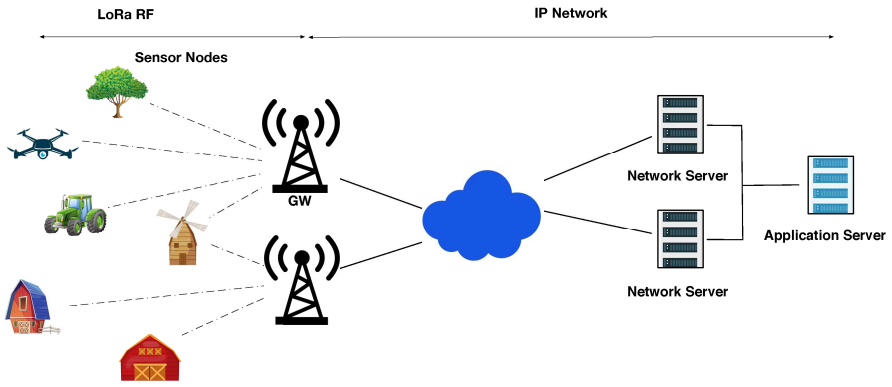

Figure 1. LoRaWAN architecture in precision agriculture.

Finally, the application, running on a server, processes the data, providing insights or automated actions, such as controlling irrigation systems or generating alerts. This architecture allows efficient long-range communication with low power consumption, making it ideal for precision agriculture. In summary, all LoRaWAN gateways within the range of a specific end device receive data from the device. These gateways subsequently forward the data to the network server, which handles tasks such as removing duplicate packets, verifying data integrity, and conducting security checks before passing the message to the application server.

3.2.4. IoT Protocols for Precision Agriculture Requirements

Table 8 presents a comparative analysis of IoT protocols for precision agriculture requirements. Each protocol listed in the table is evaluated against the fundamental re-

quirements for IoT sensor standards in the precision agriculture sector. SigFox, with its lightweight data flow and long-range coverage (10–40 km), supports simplicity and geolocation but may be limited by its low data rate (100 bps) and constrained duty cycle, making it less suitable for high-frequency data transmission. NB-IoT provides high security, adaptability, and support for diverse data types due to its integration with LTE networks, while maintaining low power consumption, making it highly versatile for precision agriculture. LoRaWAN, with its open platform and availability of open-source implementations, excels in supporting geolocation and lightweight data flow, but its lower data rate (50 kbps) might restrict use in high-bandwidth scenarios. WiFi, while offering high data rates and bandwidth, has higher power consumption and limited range (10–100 m), which may hinder its practicality in expansive agricultural areas. Finally, Bluetooth, with its simplicity and support for multiple use cases, is suitable for short-range applications but is constrained by its limited range and lower bandwidth compared to WiFi. By analysing these protocols, LoRaWAN and NB-IoT emerge as strong candidates for addressing the specific needs of the precision agriculture sector, such as geolocation, open platforms, and energy-efficient long-range connectivity.

Table 8. Comparison of IoT protocols based on precision agriculture requirements

Requirement	SigFox	NB-IoT	LoRaWAN	WiFi	Bluetooth
Lightweight Data Flow	✓	✓	✓		
Simplicity	✓		✓		✓
Easily adaptable		✓	✓	✓	✓
Open Platform			✓		
Geolocation	✓	✓	✓		
Support various data types		✓		✓	
Open Source			✓		
Precision Agri. Needs		✓	✓		

3.3. OGC SensorThings for Precision Agriculture (RQ3)

To implement an interoperable architecture for precision agriculture, the OGC SensorThings standard was selected. In addition to meeting all the requirements (listed in Table 4) for developing an IoT platform for precision agriculture, it also possesses the four following features that make it ideal for creating an interoperable architecture.

Object Modelling: The OGC SensorThings standard ensures that all objects are modelled in the same way: every object described with the OGC SensorThings standard possesses the same types of entities and properties, facilitating the standardization of data received from sensors. Specifically, the device responsible for "translating" the data does not need to assign a specific code or tag to each object or property but is free to model any object using the structure provided by the OGC SensorThings standard without the risk of having outdated or incomplete information.

Geolocation: Another advantage of using the SensorThings standard is its handling of geolocation; unlike the other standards analysed, where the position of the device is expressed through a property of the object, OGC SensorThings manages the geolocation of objects natively. Specifically, while with other standards the location history of objects must be managed externally by extracting positions from various properties, SensorThings automatically collects all positions occupied by devices and stores them in an "entity" called HISTORICAL LOCATION whenever the geographic location of a device changes.

Date and Time: Another key feature for selecting the OGC SensorThings standard is its native support for managing date and time. Every observation made by the device and every reported position includes a property that contains the timestamp of the measurement. If the sensor does not send the date and time of the measurement, it is the responsibility of the receiving server to provide them, as specified by the standard.

In the field of precision agriculture, it is crucial to know the exact time of day when a measurement is taken for accurate analysis, especially when it is necessary to cross-reference data to determine the position of the sensor at the time of the observation.

Ease of Use: The final factor that influenced the choice of the OGC SensorThings standard is its ease of use. The standard's specifications are clear and intuitive, with various entities and properties named in a way that makes their purpose easily understandable. Additionally, there is no need to memorize large amounts of data or codes for its implementation. This feature is especially important during the installation phase, when part of the data must be initialized in the database and entered manually. This task is usually performed by users who may not have in-depth knowledge of IoT standards.

3.4. The Entities in OGC SensorThings Database

The rest of this section provides a more detailed analysis of the APIs of the OGC SensorThings standard. Figure 2 represents the relational model with which the standard is structured. The image summarizes the entities required for the creation of an OGC SensorThings database. For clarity, only the properties that must be explicitly defined for the creation of each entity are highlighted. The following specifications provide a brief description of the entities that make up the interoperable schema.

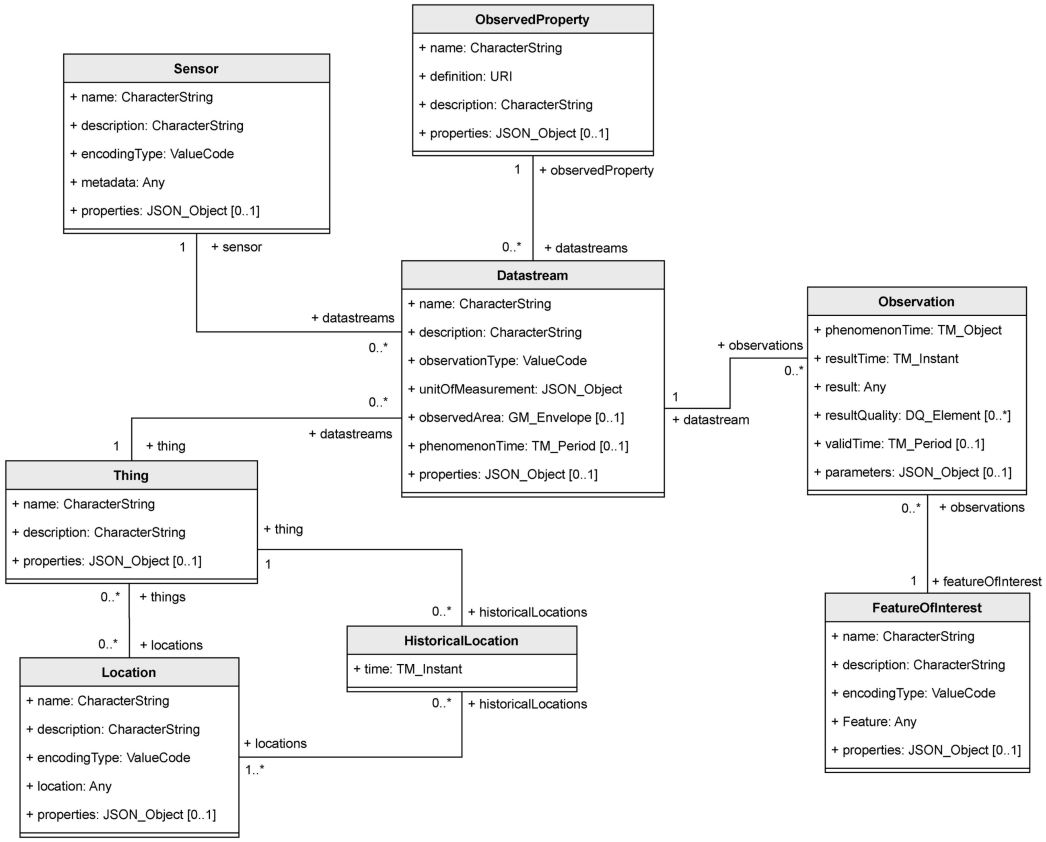

Figure 2. Diagram of the relational model structure of the OGC SensorThings standard [52]. '0..*' indicates a cardinality of "zero to many", meaning that an element may be associated with none, one, or multiple elements of the other entity. Similarly, '1..*' indicates a cardinality of "one to many", meaning that an element must be associated with at least one or more elements of the other entity.

Thing: The OGC SensorThings API follows the ITU-T definition, which states that, in the context of the Internet of Things, a Thing is an object from the physical world (physical things) or the information world (virtual things) that can be identified and integrated into communication networks [51].

Location: This entity identifies the last known position of a Thing. In the context of IoT, the primary position of interest is typically associated with the location of the Thing, especially for in situ detection applications. For example, the location of interest for a WiFi connected sensor should be the barn or greenhouse where the smart sensor is located. However, the ultimate location of a Thing is not always the location of the Thing itself [51]. In these use cases, the content of a Thing's location differs from the content of the characteristic of interest of the Thing's observations.

Historical Location: The Historical Location entity provides the timestamp of the current position (i.e., the last known) and previous positions of the associated Thing entity.

Sensor: A device that observes a property or phenomenon with the goal of producing an estimate of the property's value.

Observed Property: An Observed Property specifies the phenomenon of an observation.

Observation: The act of measuring or otherwise determining the value of a property. Mandatory properties are as follows: (i) Timestamp of the observed phenomenon: TM_Object (ISO 8601 or Time Interval String) (corresponds to the time period during which the observed phenomenon evolves); (ii) Timestamp of the measurement: TM_Instant (exact moment when the measurement is made); (iii) Results: Any [52].

Feature of Interest: An observation translates into a value assigned to a phenomenon. The phenomenon is a property of an element, and this element is the Feature of Interest of the observation. In the context of IoT, the Feature of Interest of many observations can be the location of the Thing. For example, the Feature of Interest of a WiFi connected may be the location of the sensor (i.e., the living room where the sensor is located). In the case of remote sensing, the Feature of Interest is the geographical area detected [80].

Using the HTTP GET method, data can be requested from the server, and responses are returned in JSON format. For instance, the URI http://example.org/v1.0/ addresses entities in the database (accessed on 1 January 2025). The response includes a JSON object with a `value` property containing a JSON array, where each element specifies the name of an entity set (e.g., `THING`, `DATASTREAM`, `OBSERVATION`) and its corresponding URI. CRUD operations (Create, Update, Delete) are carried out using HTTP methods such as POST, PATCH, PUT, and DELETE. These methods often include additional information in the body of the request to complete the operation [48].

4. An Interoperable IoT Architecture for Precision Agriculture (RQ4)

This section proposes a possible interoperable architecture that involves using the OGC SensorThings standard to collect non-standardized data from sensors. To convert between different formats, the project includes additional tools that act as intermediaries between the sensors and the data collection database. The focal point of the project is the deployment of a device, which we will henceforth refer to as a "connector", that gathers data sent by various gateways, converts them according to the standard's specifications, and sends them to the central database [81]. Finally, a client of choice can be connected to the database to visualize and process the data.

4.1. Functional Requirements of IoT Architecture

The requirements of the system architecture have been identified. Some of these are met by the chosen standard, while others are inherent in the technology being used.

The proposed architecture aims to be as "adaptable" as possible to the types of needs that may arise in the context of precision agriculture. The main requirements are listed below.

- Absence of model constraints in sensor usage
- Support for various communication protocols
- Ensured data homogeneity
- Easily scalable
- Data storage capability
- Data visualization capability based on user-defined filters
- Sensor status monitoring
- Ease of use
- Use of open-source software
- Accessibility from mobile devices
- Autonomous with minimal routine maintenance

The proposed architecture is shown in Figure 3 The diagram illustrates the communication flow between devices and highlights the protocols used in the various sections that make up it. The system consists of three sections: the first section represents the various sensors located in agricultural fields that periodically send data to the gateways. The second section includes the gateways that communicate with *connectors* , which act as "translators" for the data being sent to the third section, which comprises the server and the central database [82]. Regarding the first section, we deploy heterogeneous sensors produced by different companies and equipped with various communication methods. The various devices communicate with the associated gateway using different technologies, which may fall under the LPWAN (Low-Power Wide-Area Network) category, such as LoRaWAN or SigFox, or utilize a wireless connection to the Internet [73].

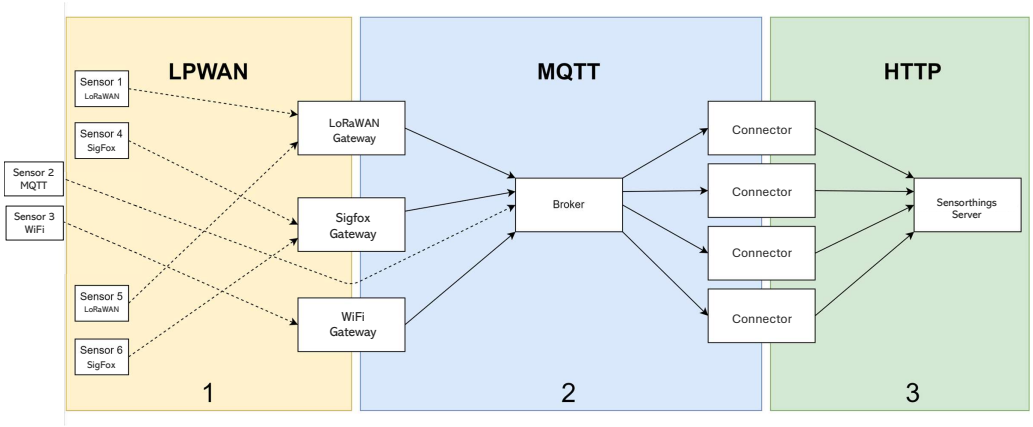

Figure 3. Block diagram of the interoperable IoT architecture for precision agriculture.

In the second part, the gateways act as bridges between non-IP protocols, used to communicate with the sensors, and MQTT, the protocol used to communicate with the connectors. Unlike the sensors, the gateways do not face significant issues regarding autonomy and are connected to the Internet. Their task is to forward the data received from the sensors to the broker, which in turn will publish the messages to their respective topics. The *connectors* are used to collect heterogeneous data from various gateways, convert it into the OGC SensorThings format, and redirect it to the server via the RESTful protocol [83]. In the network to be created, a dedicated connector is set up for each sensor model connected to the network. In the MQTT network, the gateways are essentially the "publishers", while the connectors play the role of "subscribers". A gateway publishes to the "topic" corresponding to the sensor model from which it received the payload. A connector "listens" only to the topic on which the payloads it can convert are published. The use of MQTT

helps keep the payload size small and allows some sensors to communicate directly with the connector, bypassing the gateway if they have their own Internet connection [62].

Another advantage of this configuration is the ease of system scalability. In fact, if additional sensor models need to be added to the network later, it will only be necessary to add a new "topic" to the MQTT network and associate a connector with it, without modifying the pre-existing configurations. In the third section, the server implements one of the available open-source software solutions for managing an OGC SensorThings database. In the following subsection, the available alternatives will be analysed, and the one most suitable for the proposed system will be selected. These implementations use an SQL database for storing information, with support for managing spatial geographic features [84]. Finally, the central server is connected to a client that can request, process, and visualize the data contained in the database.

4.2. Sensors and Gateways

The lowest level of the architecture coincides with the sensors positioned in agricultural fields at strategic points, to obtain a data sample that is as significant as possible for the uses and analyses that will be carried out subsequently. These sensors, as described earlier, can have different methods for collecting data and use different protocols for communication. The user is free to choose and add to the system the sensors they consider most suitable for the intended purpose. During installation, sensors are connected to the associated gateway based on their communication protocol. In the event that a sensor has an autonomous Internet connection, it is directly connected to the broker. Since limiting the types of protocols used would otherwise undermine the ultimate purpose of the system, a gateway is provided within the system for each type of communication protocol used by the sensors. There are numerous types of gateway available on the market that can be configured as desired to forward the received packets along a predefined channel, as we analysed in previous chapters. The gateway required for our system must be able to communicate with an MQTT broker [85]. The purpose of the gateway is to receive data from the connected sensors, identify which sensor the data came from, and publish it to the associated "topic". As mentioned earlier, there is a "topic" for each type of sensor present in the system. A gateway can publish to multiple "topics", as it can be connected to sensors from different manufacturers. It is not the gateway's responsibility to manage duplicate packets or the correct reception of information. These tasks are delegated to the network server, based on the level of QoS that is to be maintained within the system.

4.3. Connectors

The main focus of this research is the introduction and development of connectors. A connector is software that enables communication between various sensors deployed in the real world and the end user (such as a farmer) [81]. The system is designed with as many connectors as there are different types of data to convert. For example, in our system, there are five types of sensor produced by different manufacturers, each with different payload formats. As a result, five connectors will be set up to handle the data conversion for each sensor type. The deployment of multiple connectors brings significant advantages to the system compared to a single-connector configuration. Some potential advantages are listed below.

Workload reduction on the connector: The connector listens on a channel where it is certain that the received data is convertible, so there is no need to verify the origin of the data.

System scalability: If there is a need to add or replace sensors in the system, it is sufficient to work only with the involved connectors, adding new ones if necessary, without affecting the other connectors.

Software maintenance: In case of bugs or sensor firmware updates, it is enough to update the software of the associated connector.

As mentioned earlier, the purpose of a connector is to receive the data, convert them, and then send them to the central database. Its operation consists of four phases, which are listed below:

1. **Waiting for Data**: The connector remains in a waiting state to receive data on the "topic" to which it has subscribed. It is the broker's role to act as an intermediary between the gateway and the connectors. Each connector is subscribed to one and only one "topic" for the benefits mentioned earlier.
2. **Data Verification**: Upon receiving the data, the connector checks that the data are intact and that there are no errors in the payload. If any issues are found, the received data are discarded.
3. **Data Conversion**: Following the integrity check, if successful, the conversion occurs. In practice, the connector reads the data and, using a predefined schema associated with the data type, converts it into alphanumeric values suitable for the OGC SensorThings standard and collects them into a JSON object to be sent to the central database.
4. **Data Sending**: The JSON object is sent to the OGC SensorThings server to the corresponding entity using the methods described in the previous sections.

4.4. Server and Client

The final part of the system is structured around the central server, which manages the exchange of information between the database and the requesting clients.

The central server primarily responds to two types of external event:

Data Request: Regardless of the origin of the request, the server checks that the request is correctly formulated according to the specifications dictated by the standard. If it is valid, the server converts the HTTP request into an SQL query to submit to the database, extracting the requested data and forwarding them in the format specified by the standard to the origin of the request. If the request is incorrectly formulated or the requested data are not present in the database, the server terminates the communication, notifying of the error.

Data Publication: The server checks that the request and data formatting are correct. If valid, the server consults the database to verify that the submitted data are consistent with the data already present. The server communicates a successful outcome if all operations are completed successfully. Otherwise, as before, the server terminates the communication, notifying the error.

The operation of the connector is shown in the activity flow chart in Figure 4. In particular, the diagram illustrates all the steps described so far and the points where the connector must make decisions regarding the incoming data. The connector also validates the data by checking its integrity upon receipt, ensuring that the payload is free of errors. If any issues or discrepancies are identified, the data are rejected and discarded. Furthermore, if added robustness in transmission is desired, redundancy in the payload can be implemented, as proposed in [86].

A major advantage of the SensorThings API (STA) for clients is its powerful query and search functionality [51]. Discovering what type of data is served by the STA instance is straightforward. In most cases, a simple web browser is sufficient, as all entities can be accessed simply by following the navigation links provided by the API. Through the client, data requests can be made following the specifications of the APIs provided by the SensorThings standard. The purpose of the client is to provide a tool for data extraction and

analysis. For the visualization of geospatial data on a map, it is the client's responsibility to extract the coordinates from the received data and request the relevant map from the central server (if it provides this service by implementing geospatial data management software, such as GeoServer) [87].

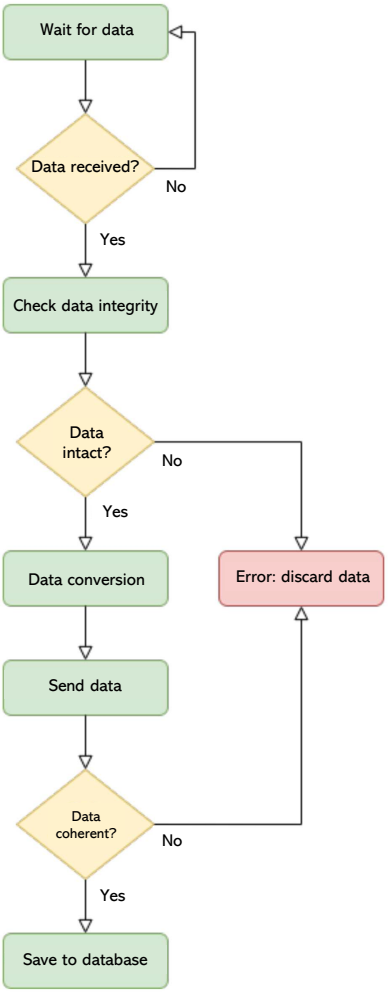

Figure 4. Flowchart of activities related to the operation of the connector.

4.5. Comparison of Open-Source Software (RQ5)

One of the functional requirements of the system is the use of open-source software in order to reduce development costs for precision agriculture. In fact, implementation costs often represent a major hurdle to overcome in the digitization of farms [10,88]. Currently, there are few open-source alternatives for the implementation of the OGC SensorThings server. However, some projects are available to the public without licensing. For this project, the most comprehensive implementations currently available were selected and analysed.

Whiskers is a framework built on the OGC SensorThings API, featuring a JavaScript client and a lightweight server designed for IoT gateway devices like Raspberry Pi. Its goal is to foster an open and healthy IoT ecosystem, in contrast to one that relies on proprietary information protocols [89,90].

FROST-Server is an open-source server for the OGC SensorThings API [91]. The software is composed of two parts: The first, FROST-Server, implements the entire specification described by the standard, including all the extensions added later [92]. The second part, called FROST-Client, is a Java client library for communication with a server compatible with the SensorThings API.

Mozilla STA is an implementation of the OGC SensorThings API node developed by Mozilla [93].

52N SensorThingsAPI is an open-source implementation of the OGC SensorThings API. Key features include compatibility with 52N SOS (which implements the OGC Sensor Observation Service), customizable database mappings, and various extensions. It can be deployed as a Docker container, on Apache or Tomcat, or as a standalone application [94].

Table 9 shows the comparison of the open-source software for the OGC SensorThings server, while Table 10 shows an analysis of open-source software in relation to interoperable requirements (RQ4) discussed in Section 4. For the implementation of the interoperable platform proposed in this work, we chose to use the FROST-Server software. Compared to its competitors, the software offered by the Fraunhofer Institute is more mature and complete, implementing all the specifications described in the OGC SensorThings standard. Additionally, it provides libraries for interfacing with it, which simplify the development of applications that utilize the standard. The libraries help to focus on application development by masking all the CRUD operations with simplified calls.

Table 9. Advantages and disadvantages of open-source software OGC SensorThings server.

Whiskers		**Mozilla STA**	
Pros	**Cons**	**Pros**	**Cons**
It provides an open-source client for communication with the server side	Only partial support for the standard's entities.	Implements all the basic functionalities required by the standard	No client
Support for resource-constrained devices (Raspberry Pi, BeagleBone)			Project not updated for several years
52N SensorThingsAPI		**FROST-Server**	
Pros	**Cons**	**Pros**	**Cons**
Docker support	No client	Support for MQTT and MQTTp	The client does not support the MQTT protocol.
MQTT support	Does not support the MultiDatastream extension	Easy and fast installation through Docker	The client does not support "batch requests" from the standard
Good documentation		Well documented	
Interoperability with the 52N SOS service		Built-in authentication support	

Table 10. Open-source software in relation to the interoperable requirements discussed in Section 4.

Requirement	**Whiskers**	**Mozilla STA**	**52N SensorThingsAPI**	**FROST-Server**
No model constraints	✓	✓	✓	✓
Support for various protocols	✓	✓	✓	✓
Data homogeneity		✓	✓	✓
Scalability	✓	✓	✓	✓
Data storage				✓
Data visualization with filters				
Sensor status monitoring	✓	✓	✓	✓
Ease of use	✓	✓	✓	✓
Open-source	✓	✓	✓	✓
Mobile access	✓	✓	✓	✓
Minimal maintenance			✓	✓

5. Use Case Description with the OGC SensorThings Standard

In this section, the typical behaviour of a user/farmer is analysed in the proposed system. Figure 5 shows a diagram that represents the various users, how they use the system, and their permissions. The two most common ways for users/farmers to interface with the system are as follows: requesting and viewing data, and sending data to the database. In the first case, the farmer logs in, formulates a query that is translated by the client into the OGC standard, and waits for the response from the server. In the case of a positive outcome, the data will be displayed in the client. In the second case, the farmer logs in with privileges for data submission and sends the data to the server, which will check that the data are in the correct format and consistent with the database before saving them to the latter.

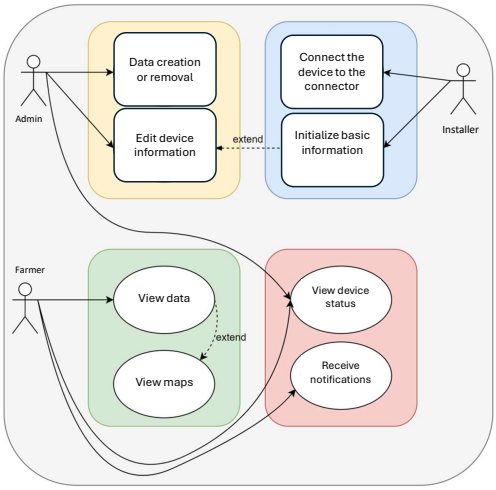

Figure 5. Diagram illustrating the different users, their system interactions, and corresponding permissions.

Farmer: The farmer receives and views data through the client. The client communicates with the server via the HTTP protocol. Requests to the server are made by the client through the service interface of the SensorThings standard, in a way that is transparent to the farmer.

For example, when the farmer wants to view the first 10 results of the "Observations" entity sorted by ID, the client will send a GET request to the server at the following address: http://example.org/v1.0/Observations?\protect\T1\textdollarorderby=ID&\protect\T1\textdollartop=10 (accessed on 1 January 2025).

Installer: The installer creates the new entities required within the database so that the sensor can be connected to the system. To do this, they use the CRUD operations provided by the SensorThings service interface. Using the POST method of the HTTP protocol, they can send the various instances of the entities necessary to initialize the sensor within the system. For example, if the installer needs to install a sensor with the following specifications—*name*: Sens-IA12, *location*: [−114.06, 51.05], *sensor type*: atmospheric pressure gauge—they will send the following JSON object through the POST method to the address: http://example.org/v1.0/Things (accessed on 1 January 2025). Additionally, they will update its location by sending a new LOCATION entity to the address: http://example.org/v1.0/Locations (accessed on 1 January 2025).

Admin: the system administrator intervenes when it is necessary to modify and/or delete data within the database. For these operations, they use the PATCH/PUT and DELETE methods. For example, if the administrator wants to modify an instance of the entity THING, specifically to change the "description" property, they will send the following JSON object through the PATCH/PUT method to the address: http://example.

org/v1.0/Things(1) (accessed on 1 January 2025). The "description" property of the instance will be overwritten and replaced with the one sent by the administrator.

5.1. General Model of the Prototype Based on SensorThings

The proposed architecture consists of sensors from different manufacturers communicating with their respective connectors and a server for data collection. In particular, the architecture is composed of:

- An instance of **FROST-Server** for data collection;
- An instance of **Mosquitto** [95] as the MQTT broker;
- Sensors with different payloads;
- A Java application to simulate the behaviour of the connectors.

In particular, for the connectors, the FROST client libraries were used for the creation of entities and communication with the server. The advantage of using these libraries is the abstraction of the HTTP POST operations, which is entirely handled by the library, greatly simplifying both the creation of new entities and their retrieval from the server. Finally, regarding the server, an instance of FROST-Server was chosen using the Docker distribution provided directly by the authors [96].

For correct operation, each DATASTREAM entity associated with a device must be identified by a unique name. Furthermore, within the OGC SensorThings database, the entities THING associated with the devices, OBSERVED PROPERTY, SENSOR, and FEATURE OF INTEREST (for example, LOCATION) associated with the DATASTREAM must be preloaded.

The use of internal databases within the device has the advantage of being able to add and update information related to it without having to modify the connector's code. In the prototype, the internal database was implemented using SQLite [97], which allows for the creation of small-sized databases suitable for our purpose and also provides libraries for interfacing. However, the proposed architecture can easily integrate other production-grade databases, such as PostgreSQL, MySQL, MongoDB (NoSQL), and others.

The first operation the client performs once it receives the message is to identify the device from which it was sent. To do this, the client must know the unique ID associated with the device. This ID can either be sent by the sensor itself or added by the gateway in cases where the sensor model does not include the information by default. Subsequently, the connector queries its internal database to find out which entity in the database is associated with the device. The entity associated with the device is an instance of the THING entity in the OGC SensorThings model. It is necessary to know the THING entity to which the device is associated because the related DATASTREAMS can be derived from it, enabling the publication of the related sensor observations.

Next, we move on to the phase of "translating" the information: this operation is specific to each sensor model and characterizes each connector. Later, we will present examples that will address this operation more specifically; for now, we will limit ourselves to describing, in general terms, the operations performed by each connector:

1. The connector extracts the measurement data from the received payload.
2. It interprets the data according to the guidelines provided by the manufacturer.
3. It determines the type of measurement and its values.
4. It searches among the DATASTREAMS associated with the THING entity for the one related to the measurement.
5. If not present, it creates a new DATASTREAM entity with the information available in its database: first, it extracts from the internal database the SENSOR and OBSERVED PROPERTY entities related to the DATASTREAM to be created; subsequently, it

extracts the data related to their properties and completes the creation. In the event that one or more SENSOR or OBSERVED PROPERTY entities are unavailable, the connector will create new ones using the data available in its database.

6. It publishes a new OBSERVATION linked to the related DATASTREAM, containing the received values, and associates the relevant FEATURE OF INTEREST.

The steps related to the creation and linking of the SENSOR and OBSERVED PROPERTY entities are illustrated only in the first example and are subsequently omitted for clarity. In the case that the sent measurement is the geographical location of the sensor, instead of publishing an OBSERVATION, a new instance of the LOCATION entity will be created to associate with the THING instance of the sensor.

5.2. Tested Sensors

The proposed architecture focuses on translating the payloads sent by sensors from different manufacturers. In fact, each measurement sent by the sensors is formatted according to the technical specifications provided by the respective manufacturers, and specifically, two sensors from different companies, such as SensorData [98] and Libellium [99], have been tested.

5.2.1. SensorData Digital Matter

The payload for SensorData sensors produced by Digital Matter is composed of a binary hexadecimal string where data are passed through a key-value model: a byte is assigned a key, which is used to identify the type of measurement it represents, while the subsequent n bytes (n varies depending on the type of data) represent the value of the measurement [98]. Data are always transmitted in a single message to avoid complexity and limitations. Related values are grouped together, while unrelated ones can appear in the same message. The payload starts at byte 0 with no header, and the first measurement's key is determined by the port number. SigFox has a fixed payload size of 12 bytes, while LoRaWAN's payload size varies by region and spreading factor, with a minimum of 11 bytes in the EU868 region at SF 12 [100].

Payload Syntax: each data measured or transmitted by the sensor is called a "data field", identified by an ID (or "key") and containing a value. The software decoding the data needs to know the IDs and their lengths. The manufacturer provides a table with the IDs, their corresponding data fields, value sizes, and data types (e.g., UINT8, BYTE). Key data fields for the system include those related to the device's satellite position and the measurements it performs.

- **10 GPS Position**: The position is associated with the identifying ID value 10, and the associated data is 6 bytes long. The values associated with the measurement are expressed as two numbers in INT24 format, so the first three bytes of the value field represent the sensor's latitude, while the last three bytes represent the longitude.
- **SDI-12 Measurement n**: Sensor measurements are labelled as "SDI-12 measurement" followed by a number (1 to 5). Each sensor can send up to five types of readings, and if the data exceeds the byte limit of the communication frame (e.g., 11 bytes for LoRaWAN), the reading may be split into two parts. The first 4 bits of the first byte communicate the number of values for that measurement (maximum of 10 values for each measurement), while the last 4 bits indicate the data type of the values according to the encoding in the table. The values contained from the second byte onward correspond to the actual measurement values. The five types of data for the values are (i) Soil moisture UINT8, (ii) SDI-12 Temperature UINT8, (iii) SDI-12 Generic INT16 $\times$ 100, (iv) SDI-12 Generic INT32 $\times$ 1000, and (v) SDI-12 Generic INT12.

It is important to note that every other measurement produced by the sensor (internal temperature, battery charge level, etc.) behaves similarly to the external measurements of the sensor and is treated by the system in the same way, that is, with observations that are linked to a DATASTREAM.

Additional Requirements: In addition to those listed in the general model, the SensorData connector requires additional requirements:

- The payload received by the connector must contain the identification ID of the device from which the message was sent.
- The payload sent from a LoRaWAN gateway must incorporate the port number through which the gateway received the message, as it identifies the ID of the first measurement.
- The DATASTREAM entities related to external measurements must be preloaded into the OGC SensorThings system, since the connector does not have enough data to create them.

5.2.2. Libellium Waspmote

The second type of payload analysed is that produced by Waspmote devices from the company Libellium [99]. These devices are user-programmable and can produce payloads in two formats: the first format presents the payload as a string made up of ASCII characters, while the second format presents the string as a series of bytes [101].

- **ASCII String**: This format prioritizes ease of understanding the data over packet size. ASCII characters allow the sensor to send data transparently without converting them to specific measurement types. The connector can directly extract values by reading the payload, without needing to know the data types. However, each character takes up one byte, making compression difficult and resulting in larger frame sizes.
- **Binary String**: This format is designed for more compressed frames, sacrificing readability. Each byte represents a specific field's value, aiming to minimize payload size and maximize information transmission. The sensor converts each measurement to its corresponding data type based on the manufacturer's specifications. The connector must know these specifications to extract and "translate" the values before sending them to the central server.

Payload Syntax: Both ASCII and binary frames share a similar structure, consisting of two parts: a fixed header with device and communication information, and variable fields with sensor measurements in a key-value format. The key difference is that, in ASCII frames, the character "#" delimits each field, while in binary frames, "#" marks the end of a variable-length field, and other fields have fixed lengths. Measurements are expressed as key-value pairs, and the manufacturer provides a table detailing the sensors' possible measurements, including properties like ID, size, unit, and precision. This information is crucial for the connector to extract and convert data for the server.

Additional Requirements: In addition to those listed in the general model, the Libellium connector requires further specifications:

- The THING entity in the OGC SensorThings system must be associated with the unique Waspmote serial ID.
- The device transmits only simple frames, meaning that all information is contained in a single frame and is not divided into multiple frames.

6. Analysis and Results

In this section, we provide details on the implementation of the data model and SensorThings STA for the development of an interoperable architecture for precision agriculture.

For demonstration purposes, we have implemented instances of the data model and the API as part of the deployment of Libellium Waspmote and Digital Matter SensorData.

6.1. SensorData Connector: Payload Analysis and Results

In this section, we will test the behaviour of a SensorData connector when receiving a payload. After taking some measurements, the sensor transmits the obtained values through a payload containing the hexadecimal string that follows to the gateway:

$$\text{“}1B089C7157A33348012606182310 06A18\text{”}$$

As shown in Figure 6, the LoRaWAN gateway receives the payload through port 10 and in turn forwards it to the system broker in a message that includes the unique ID identifying the sensor, in this case set to “001”. The broker, upon receiving the packet, publishes it to the respective SensorData topic, where the connector associated with the SensorData devices is listening.

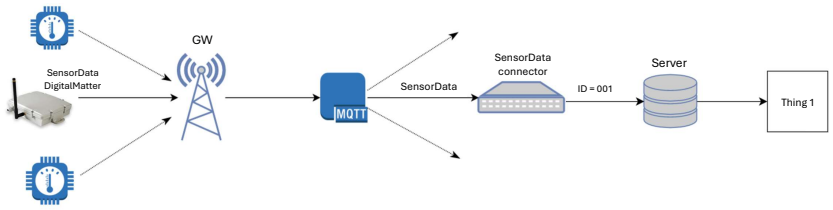

Figure 6. Architecture for testing the Digital Matter SensorData device and the corresponding connector.

The connector receives the message as shown in Listing 1, and the first operation performed by the connector is to extract the THING entity from its internal database, which is present in the SensorThing system of the central server, correlated with the received ID according to the following Listing 2.

Listing 1. SensorData payload example and THING association.

```
{
  ID: 001,
  PORT: 10,
  PAYLOAD: AF4D3276CE5F3334801260618231006A18
}
```

Listing 2. SensorData THING extraction.

```
{
    Name: "Sens-IA12",
    Description: "Device SensorData 1"
}
```

In this way, the connector recognizes which device sent the message and can proceed to update the data present on the server based on the information received. Furthermore, as specified, the LoRaWAN port number corresponds to the ID of the first “data field” sent; in this case, the number 10 corresponds to the “data field” GPS, while the payload starts from byte 0 with the values related to the “data field” corresponding to the port. The connector proceeds to analyse the received payload according to the following specifications:

■ **1B089C7157A**: From the port number, it can be inferred that the first 6 bytes of the payload contain the position value of the device. Specifically, the first 3 bytes (1B089C) correspond to the device’s latitude, while the last 3 bytes (7157A) correspond to the longitude. Therefore, in this case, after the appropriate conversions, the latitude and longitude coordinates obtained are *Latitude* = 45.3549056 and *Longitude* = 11.884812, as shown in Listing 3. With these values, the connector has created a new LOCATION entity to associate with the

device in the system. In particular, the sensor is located at the coordinates that correspond to a rural geographic area near Padova.

Listing 3. Association SensorData LOCATION of the device in the system.

```
{
  encodingType:application/vnd.geo+json,
  location: {
    type:Feature,
    geometry:{
      type: Point,
      coordinates: [45.3549056,11.884812]
        }
    }
}
```

■ **3334**: From byte '33', we derive ID 51, which corresponds to the battery level indicator, while from byte '34', we obtain the measured value, which is 52 (Listing 4). The connector searches among the DATASTREAMs linked to the THING entity of the device for the one related to the remaining battery measurement (Listing 5).

Listing 4. Battery level observation.

```
{
    result: 52
}
```

Listing 5. DATASTREAM associated with the battery Digital Matter sensor.

```
{
  name: "Battery remain",
  description: "Remaining battery of the device",
  unitOfMeasurement:
  {
    name:"Percentage",
    symbol:"%",
    definition: ""
  },
  observationType:"OM_Observation",
}
```

If the DATASTREAM is not present in the OGC SensorThings system, the connector extracts its properties and the related entities from its internal database and creates a new one. The same operation is performed for the SENSOR and OBSERVED PROPERTY entities if they are missing in the central system, as shown in Listings 6 and 7.

Listing 6. SensorData OBSERVED PROPERTY entity.

```
{
    name:"Battery remain",
    definition:"HTTPs://en.wikipedia.org/wiki
    /Electric_battery"
    description: "The remaining energy level of an
    electric battery"
}
```

Listing 7. SENSOR entity.

```
{
    name:"SensorData",
    description: "SensorData LoRaWAN is a battery
    or line powered data communicator that interfaces
    to a range of sensors, GPS, inputs and outputs,
    and uploads data via LoRaWAN networks"
    encoding type: "application/pdf",
    metadata:"HTTPs://digmat.freshdesk.com/helpdesk
    /attachments/16073699325"
}
```

Finally, the measurement is published, linking it to the corresponding FEATURE OF INTEREST entity. In fact, similar to the previous entities, there is a table within the

connector containing information related to the FEATURE OF INTEREST entities, as shown in Listing 8.

Listing 8. FEATURE OF INTEREST entity.

```
{
    name: "Battery",
    description: "Device battery",
    encodingType:"application/vnd.geo+json",
    feature:""
}
```

Note how the internal database links the FEATURE OF INTEREST with the DATASTREAM and the unique ID that identifies the device. While the previous entities (OBSERVED PROPERTY and SENSOR) are common to all devices of the same model, the FEATURE OF INTEREST is specific to each individual device, as measurements can be taken in spaces with different dimensions and/or locations. If there is no FEATURE OF INTEREST to link to the observation, the central server will derive one from the LOCATION entity with which it is connected. The steps related to the creation of the OBSERVED PROPERTY, SENSOR, and FEATURE OF INTEREST entities discussed so far will be omitted from the upcoming payload analyses to make the explanation clearer.

■ **80126061**: The first byte has a decimal value of 128, which corresponds to the SDI-12 Measurement 1 "data field". The first value of the second byte (1) indicates the data type used to express the values, in this case, "Temperature UINT8", while the second value (2) represents the number of observations, in this case, 2. The last 2 bytes correspond to the measurement values *measurement*1 = 8 and *measurement*2 = 8.5, which represent two measurements of soil temperature. The connector does not have enough information about the nature of "SDI-12 Measurement" type measurements to create a new DATASTREAM for them (Listing 9). Therefore, it can only search among the associated DATASTREAMS to check if one has been previously loaded. Once the DATASTREAM is obtained, as shown in Listing 10.

Listing 9. Soil temperature DATASTREAM.

```
{
    name: "Soil temperature",
    description: "Temperature 20cm under the soil",
    unitOfMeasurement:
    {
        name:"Degrees Celsius",
        symbol:"°C",
        definition: ""
    },
    observationType:"OM_Measurement",
}
```

Listing 10. Soil temperature OBSERVATION.

```
{
    result: 8
}
{
    result: 8.5
}
```

■ **8231006A18**: Similarly to the previous "data field", the ID is derived from the first byte (82), which corresponds to 130, SDI-12 Measurement 2. From the second byte (31), an INT32 type measurement is obtained, consisting of a single value. After conversion, the resulting value is 1600, which corresponds to the measurement of the nitrogen level in the soil. In this case, the connector searches for the corresponding DATASTREAM as per Listing 11 and publishes a new OBSERVATION, as shown in Listing 12.

Listing 11. Nitrogen concentration DATASTREAM.

```
{
    name: "Nitrogen concentration",
    description: "Nitrogen concentration in the
    soil, mg on Kg",
    unitOfMeasuremen":
    {
        name:"milligrams to kilogram",
        symbol:"mg/Kg",
        definition: ""
    },
    observationType:"OM_Measurement",
    resultTime:"2024-11-03T13:00:00Z/2024-11-
                03T13:00:00Z"
}
```

Listing 12. Nitrogen concentration OBSERVATION.

```
{
  resultTime:"2024-11-03T13:00:00Z/2024-11-
              03T13:00:00Z",
  result": 1600
}
```

6.2. Libellium Connector: Implementation and Observations

In this subsection, we will test the Waspmote sensors by Libellium. Generally, this type of sensor sends all the necessary information to the connector within its own frames, and in this specific case there is no need for additional information encapsulation by the gateway. Two examples of translation performed by the Libellium connector are presented: one for ASCII encoding and one for binary encoding.

6.2.1. ASCII Encoding

The Waspmote sensor transmits its value with a payload containing the hexadecimal string shown in Listing 13, forwarding it to the connector following the path shown in Figure 7.

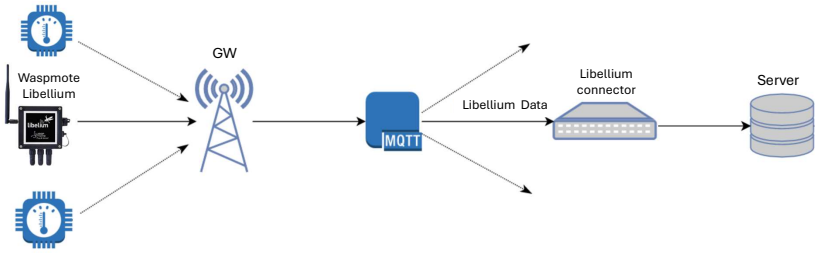

Figure 7. Architecture for testing the Waspmote Libellium device and the corresponding connector.

Listing 13. ASCII encoding Waspmote payload.

```
3C3D338003202333353639303238342023204E4F44455C5F30303
1233231342354434 13A3335234750533A33312E3230303B34322E
3130234241543A3837
```

Upon receiving the payload, the first operation the connector performs is checking the frame encoding. The byte carrying information about the frame (0x80) has the most significant bit set to 1, so the connector will interpret the frame as the ASCII string shown in Listing 14.

Listing 14. Waspmote payload interpreted by the connector.

```
<=> 0x80 0x03 # 35690284 # NODE 001 214 # TCA:35 #
GPS:31.200;42.100 # BAT:87
```

The second operation performed by the connector is to identify the sensor within the system. To do this, it reads the frame field containing the unique device ID, in this case, 35690284, consults its internal database, and retrieves the ID of the associated THING entity present on the server, as shown in Listing 15. Finally, proceed to the analysis of the information contained in the payload.

Listing 15. Waspmote Libellium THING extraction.

```
{
    Name: "Waspnote001",
    Description: "Sensor Waspmote 1",
}
```

■ **TCA:35**: This field corresponds to a temperature measurement. The Libellium connector searches among the DATASTREAMs associated with the Waspmote sensor for the one corresponding to temperature; if not found, it creates a new one (Listing 16). Once the DATASTREAM is identified, it creates an OBSERVATION entity to link to it, containing the measurement, as shown in Listing 17.

Listing 16. Libellium temperature DATASTREAM.

```
{
    name: "Temperature Celsius",
    description: "Temperatura",
    unitOfMeasurement:
    {
        name:"Celsius",
        symbol:"°C",
    },
    observationType:"OM_Measurement",
    resultTime :"2024-11-03T13:00:00Z/2024-11-
                03T13:00:00Z"
}
```

Listing 17. Libellium temperature OBSERVATION.

```
{
 resultTime:"2024-11-03T13:00:00Z/2024-11-
            03T13:00:00Z",
 result: 35
}
```

■ **GPS:45.354;11.884**: In this case, the measurement corresponds to the GPS position of the device. The first value corresponds to latitude, while the second value corresponds to longitude. The connector creates a new LOCATION entity to be associated with the THING entity of the sensor, as shown in Listing 18.

Listing 18. Association of Libellium LOCATION.

```
{
    encodingType:"application/vnd.geo+json",
    location: {
        type:Feature,
        geometry:{
        type: "Point",
        coordinates: [45.354,11.884]
    }
 }
}
```

■ **BAT:87**: The device communicates the remaining battery percentage, and the connector handles this measurement like any other produced by the sensor (Listing 19). Consequently, it searches for the associated DATASTREAM and, if it does not find one, creates a new one. Subsequently, it creates the corresponding OBSERVATION entity.

Listing 19. DATASTREAM and OBSERVATION of battery in Libellium device.

```
{
    name: "Battery",
    description: "Battery in Libellium device",
    unitOfMeasurement:
        {
        name:"Percentuale",
        symbol:"%",
        definition: ""
    },
    observationType:"OM_Observation",
    resultTime:"2024-11-03T13:00:00Z/2020-11-03T13:
                00:00Z"
    result: 87
}
}
```

6.2.2. Binary Encoding

Similarly to previous cases, the sensor transmits the following payload to the connector: "3C3D3E 00 03 74F94515 53 2624 180000DA41 0FD7A3CA42". The second field of the payload (0x00) contains only zeros, so the connector will interpret the frame as binary, while the fourth field contains the unique device ID, which is used to retrieve the associated THING entity from the database with an ID equal to 002, as shown in Listing 20.

Listing 20. Waspmote Libellium THING extraction.

```
{
    Name: "Waspnote002",
    Description: "Sensor Waspmote 2",
}
```

■ **2624**: The first byte corresponds to the "Leaf Wetness" measurement, which has a value of an unsigned integer equal to 36 (Listing 21).

Listing 21. Datastream and observation of Leaf Wetness sensor.

```
{
    name : "Leaf Wetness",
    description : "Leaf Wetness value",
    unitOfMeasurement :
    {
        name :"Percentuale",
        symbol :"%",
    },
observationType :OM_CountObservationn,
resultTime :"2024-11-03T13:00:00Z/2020-11-03T13:
            00:00Z"
result : 36
}
```

■ **180000DA41**: The first byte corresponds to the "Temperature Celsius" measurement, which has a value in float format. Therefore, the result is contained in the following 4 bytes and should be read in "Little-Endian" order, as shown in Listing 22.

Listing 22. DATASTREAM and OBSERVATION of temperature sensor.

```
{
    name : "Temperature Celsius",
    description : "Temperature in degrees Celsius",
    unitOfMeasurement :
    {
        name :"Degrees Celsius",
        symbol :"°C",
    },
observationType :OM_Measurement,
result : 27.25
}
```

■ **0FD7A3CA42**: The last measurement corresponds to "Atmospheric Pressure", and it also has a value in float format (Listing 23).

Listing 23. DATASTREAM and OBSERVATION of pressure atmospherics sensor.

```
{
    name : "Pressure atmospherics",
    description : "Pression atmosfericas",
    unitOfMeasurement :
    {
        name :"Kilo Pascal",
        symbol :"kPA",
     },
observationType :OM_Measurement,
result: 101.32
}
```

7. Discussion: Implications for Precision Agriculture

The Research Questions (RQs) addressed in this work have enabled the identification of suitable IoT standards and protocols to meet the design requirements of an interoperable architecture in precision agriculture.

In accordance with RQ1, this work discusses the importance of IoT standards as a middleware layer that facilitates interaction between IoT applications and networks. The main requirements of IoT standards in agriculture include simplifying implementation, reducing both costs and the complexity of managing an IoT network [3]. Additionally, IoT standards aim to improve the portability and interoperability of applications, allowing them to operate across different platforms and devices [40]. Moreover, RQ1 proposes a common model for data transfer, thereby facilitating integration and scalability among various devices and systems in the context of precision agriculture.

Building on the requirements identified in RQ1, RQ2 examines the IoT protocols that are most suitable for implementing these standards in precision agriculture. In particular, LPWAN (Low Power Wide Area Network) protocols are well-suited for collecting data from sensors and sending them to a Gateway (GW); indeed, they enable long-range communication with minimal energy consumption, making them ideal for large-scale agricultural deployments [68]. MQTT (Message Queuing Telemetry Transport) is generally used to connect the GWs to the Internet, but on certain occasions it can also be used to send data directly from the sensors to applications, as it allows for efficient transmission and is suitable for agricultural environments where connectivity may be unstable [102]. Finally, in order to create an interoperable architecture, the HTTP (Hypertext Transfer Protocol) is employed to connect the various connectors to web applications [103].

RQ3 focuses on determining the IoT standard best suited to implement an interoperable and scalable architecture for precision agriculture. In our solution, the OGC (Open Geospatial Consortium) SensorThings [104] standard was chosen because it fulfills all the necessary requirements for developing an IoT platform in this context. Its main features, such as the ability to interface with *connectors* that act as intermediaries between various sensors, enhance communication between heterogeneous devices, making it an ideal solution. Regarding data security, the SensorThings API (Application Programming Interface) does not define security capabilities of its own; however, its design is compatible with IoT security best practices and integrates with general security capabilities, supporting cross-cutting security [105]. In fact, since the SensorThings API falls under the Service Support and Application Support Layer of the IoT reference model, security is treated as a cross-cutting component acting on all layers of the proposed architecture.

RQ4 explores the requirements and challenges for creating an architecture capable of achieving interoperability across diverse IoT devices in precision agriculture. Thanks to the introduction of the connector concept [82], the proposed architecture presents itself as a solution to the problem of *interoperability* among IoT devices. This solution offers several advantages, making it easy to implement and well-suited for the precision agriculture sector. First of all, the resources required by a connector are minimal: the operations performed

do not require substantial computational power, and the memory usage is very limited. A connector can therefore be easily implemented on an embedded device with an Internet connection, such as a Raspberry Pi, making it economically efficient [106]. Each connector connected to the network can be considered a module of the system, making it adaptable to various needs. The network will only host the necessary modules, reducing non-essential functionalities that could add complexity and incompatibility to the system. Additionally, if a connector fails or its software needs updating to comply with new guidelines from manufacturers, it is sufficient to address the single module by replacing or modifying it without the need to reconfigure the entire system.

RQ5 aims to identify the most suitable open-source software to implement the proposed architecture, focusing on solutions that align with the identified requirements and enhance system scalability and ease of deployment. Based on the analysis presented in Section 4.5, the FROST-Server (FRaunhofer Opensource SensorThings-Server) [96] emerges as the most suitable open-source solution for implementing an interoperable architecture in precision agriculture. This is primarily due to its comprehensive implementation of the OGC SensorThings API, which includes all necessary specifications and extensions for effective data management and interoperability. FROST-Server's ability to function as a Docker container enhances its deployment flexibility, making it easier for users to configure and manage the server in various environments. Additionally, the use of open-source software reduces architecture implementation costs, which are often a barrier to the widespread adoption of IoT in precision agriculture [10].

Moreover, the interoperability of the proposed system makes the architecture suitable for both local and large-scale contexts, facilitating the scalability of the solution. Each connector is an independent unit and can be added or removed from the network without compromising its integrity. This way, it is possible to adapt the system's size to the needs of the context without interruptions in the data flow. The combination of the connector device and the chosen standard makes the approach to the system simple and intuitive. The end user can focus on analysing the collected data without needing a deep understanding of the architecture. The development of third-party applications to interface with the data collection server requires only knowledge of the standard's specifications, which have been specifically chosen for their intuitiveness to make the experience for end users as seamless as possible.

7.1. Future Work

In this work, the development and analysis of the software for connector devices has focused on specific types of sensors produced by Libellium and Digital Matter. However, the architecture is designed to be flexible, allowing for the addition of other sensors and devices from different manufacturers. By incorporating their respective technical specifications, we can easily integrate additional functionalities into the system. Each implementation of a new type of connector can be developed as needed and added to the system once completed, thanks to the modular nature of the architecture. Furthermore, as demonstrated in the literature, the integration of machine-learning algorithms for pattern recognition in incoming data represents an opportunity to improve the scalability and interoperability of the system [107,108]. These algorithms could be employed to automatically classify data from various sensors and map it in real time to the required standard, thereby reducing the need to manually design specific connectors for each sensor family [109]. This approach could not only enhance operational efficiency but also lower maintenance costs and facilitate the expansion of the system to accommodate new devices.

The future goal is to develop a wider variety of connectors to support the majority of commonly used sensors in the agriculture market and to integrate Large Language Model

(LLM) algorithms to enhance the scalability of the proposed architecture [110,111]. Furthermore, to enhance the system's completeness, future implementations may include support for communication with actuator devices. The second part of the OGC SensorThings standard provides a communication method for interfacing with actuators. Therefore, it will be the responsibility of the connector to receive information sent by users and "translate" it into the device's language, effectively performing the reverse operation of what has been outlined in this work. The integration of actuators allows the system to go beyond simple monitoring and reporting, providing automation that increases resource efficiency, reduces human labour, and optimizes conditions for healthy crops, for example, automating fertilization systems [112] and implementing smart deficit irrigation systems, where the aim is to reduce water wastage without compromising fruit production [113]. Finally, while the proposed architecture is scalable and cost-effective, potential challenges such as integrating legacy systems and ensuring data security must be addressed to promote adoption across diverse agricultural contexts.

7.2. Impact on Precision Agriculture

The proposed architecture is a fresh approach to agricultural resource management that fosters productivity, sustainability, and resource efficiency. By the integration of various IoT devices in a modular and holistic system, the architecture enables precise, real-time monitoring of the most critical variables such as temperature of the soil, nitrogen concentration, leaf wetness, and climatic conditions. In this manner, natural resources such as water are utilized at optimal levels, while waste and inefficiency in irrigation are reduced. Moreover, simple modification and customisation of system parts owing to the use of open-source platforms such as FROST-Server and modular connectors facilitate the usage of the technology in small-sized and large-sized agricultural settings. The use of data-driven approaches ensures predictive crop care, increased output, and smaller environmental impact [114]. Therefore, the architecture will be able to support a more sustainable agriculture that is able to address climate change issues and limited resources and provide more productive but less wasteful farm production.

7.3. Comparison with Other Solutions

Interoperability is probably the biggest challenge in precision agriculture, as farms will use a wide variety of sensors and devices from different manufacturers [115]. This creates environments where systems cannot communicate well with each other and integration and data management become impossible. The literature shows some IoT platforms, such as FIWARE [116], mySense [117] and Cayenne [118], as potential solutions to these problems. FIWARE is an open-source initiative to provide standardised interfaces for integrating heterogeneous devices. In contrast, mySense is an architecture aimed at integrating sensor data in agriculture, using LoRa technology. Meanwhile, mySense presents a 4-layers technology: sensor nodes, networks, cloud services, and front-end applications. Cayenne offers a simple solution for small- and medium-sized farms gradually moving towards the adoption of smart technologies. However, to the best of our knowledge, the solutions proposed in the literature did not delve into the integration of third-party commercial devices such as Libellium and Digital Matter, as is addressed in this work. Furthermore, the integration of specific and heterogeneous agricultural sensors, such as soil temperature, nitrogen concentration, leaf wetness, and atmospheric pressure, into a single architecture represents the novelty of this work.

8. Conclusions

This work highlights the crucial importance of IoT in precision agriculture, emphasizing how technology can address the challenges related to the scarcity of arable land and the increasing demand for food. However, the lack of shared communication standards among various IoT devices poses a significant barrier to the integration and efficiency of these systems. The proposal for an interoperable architecture is essential to overcome these challenges, allowing for smooth and comprehensible communication between heterogeneous devices. In this context, the OGC SensorThings standard plays a pivotal role. By providing a unified framework for interconnecting IoT devices, data, and applications, SensorThings facilitates *semantic interoperability*, enabling different devices to communicate effectively regardless of their underlying technologies. This standard supports resource-constrained devices and utilizes REST principles, JSON encoding, and various communication protocols, making it particularly suitable for the diverse needs of precision agriculture. To further enhance *interoperability*, we have developed specific connectors that serve as software interfaces between various sensors and the OGC SensorThings framework. These connectors are designed to translate and standardize the data from different sensor types, ensuring that information can be seamlessly integrated into the system. By enabling communication between devices from different manufacturers and accommodating various data formats, the connectors play a crucial role in achieving a cohesive and interoperable architecture. The adoption of the OGC SensorThings standard will not only enhance data collection and analysis but also contribute to more sustainable and productive agricultural management by ensuring that data from various sources can be integrated and understood seamlessly. This approach will ensure a more sustainable and productive future for global agriculture.

Author Contributions: Conceptualization, E.R. and M.R.; methodology, N.L. and A.P.; software, E.R. and N.L.; validation, A.P., E.R. and M.R.; formal analysis, A.P., E.R. and M.R.; investigation, N.L. and A.P.; writing—original draft preparation, N.L.; writing—review and editing, A.P. and M.R.; visualization, E.R. and M.R.; supervision; E.R. and M.R.; project administration, M.R. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by The European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001—program "RESTART").

Data Availability Statement: Data available on request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Barbier, E.B. *Economics, Natural-Resource Scarcity and Development (Routledge Revivals): Conventional and Alternative Views*; Routledge: Abingdon, UK , 2013.
2. Pretty, J. Agricultural sustainability: Concepts, principles and evidence. *Philos. Trans. R. Soc. B Biol. Sci.* **2008**, *363*, 447–465.
3. Tzounis, A.; Katsoulas, N.; Bartzanas, T.; Kittas, C. Internet of Things in agriculture, recent advances and future challenges. *Biosyst. Eng.* **2017**, *164*, 31–48.
4. Bhat, S.A.; Huang, N.F. Big data and ai revolution in precision agriculture: Survey and challenges. *IEEE Access* **2021**, *9*, 110209–110222.
5. Smara, M.; Pathan, A.S.K. An Enhanced Mechanism for Fault Tolerance in Agricultural Wireless Sensor Networks. *Network* **2024**, *4*, 150–174. [CrossRef]
6. Lavanya, G.; Rani, C.; GaneshKumar, P. An automated low cost IoT based Fertilizer Intimation System for smart agriculture. *Sustain. Comput. Inform. Syst.* **2020**, *28*, 100300.
7. García, L.; Parra, L.; Jimenez, J.M.; Lloret, J.; Lorenz, P. IoT-based smart irrigation systems: An overview on the recent trends on sensors and IoT systems for irrigation in precision agriculture. *Sensors* **2020**, *20*, 1042. [CrossRef]

8. Caicedo-Ortiz, J.G.; De-la Hoz-Franco, E.; Ortega, R.M.; Piñeres-Espitia, G.; Combita-Niño, H.; Estévez, F.; Cama-Pinto, A. Monitoring system for agronomic variables based in WSN technology on cassava crops. *Comput. Electron. Agric.* **2018**, *145*, 275–281.
9. Pagano, A.; Amato, F.; Ippolito, M.; De Caro, D.; Croce, D.; Motisi, A.; Provenzano, G.; Tinnirello, I. Internet of Things and Artificial Intelligence for Sustainable Agriculture: A Use Case in Citrus Orchards. In Proceedings of the 2023 IEEE 9th World Forum on Internet of Things (WF-IoT), Aveiro, Portugal, 12–27 October 2023; IEEE: New York, NY, USA, 2023; pp. 1–6.
10. Pagano, A.; Croce, D.; Tinnirello, I.; Vitale, G. A survey on LoRa for smart agriculture: Current trends and future perspectives. *IEEE Internet Things J.* **2022**, *10*, 3664–3679.
11. Bai, Y.; Zhang, B.; Xu, N.; Zhou, J.; Shi, J.; Diao, Z. Vision-based navigation and guidance for agricultural autonomous vehicles and robots: A review. *Comput. Electron. Agric.* **2023**, *205*, 107584.
12. Manyika, J.; Chui, M.; Bisson, P.; Woetzel, J.; Dobbs, R.; Bughin, J.; Aharon, D. *The Internet of Things: Mapping the Value Beyond the Hype*; McKinsey Global Institute: New York, NY, USA, 2015.
13. Heikkilä, M.; Suomalainen, J.; Saukko, O.; Kippola, T.; Lähetkangas, K.; Koskela, P.; Kalliovaara, J.; Haapala, H.; Pirttiniemi, J.; Yastrebova, A.; et al. Unmanned Agricultural Tractors in Private Mobile Networks. *Network* **2021**, *2*, 1–20. [CrossRef]
14. Heflin, J.; Hendler, J. Semantic interoperability on the web. In Proceedings of the Extreme Markup Languages, Montréal, QC, Canada, 15–18 August 2000; Volume 2000, pp. 111–120.
15. Milenkovic, M. *Internet of Things: Concepts and System Design*; Springer: Berlin/Heidelberg, Germany, 2020.
16. Jiang, S.; Angarita, R.; Chiky, R.; Cormier, S.; Rousseaux, F. Towards the integration of agricultural data from heterogeneous sources: Perspectives for the French agricultural context using semantic technologies. In Proceedings of the Advanced Information Systems Engineering Workshops: CAiSE 2020 International Workshops, Grenoble, France, 8–12 June 2020; Proceedings 32; Springer: Berlin/Heidelberg, Germany, 2020; pp. 89–94.
17. Nayyar, A.; Puri, V. Smart farming: IoT based smart sensors agriculture stick for live temperature and moisture monitoring using Arduino, cloud computing & solar technology. In Proceedings of the International Conference on Communication and Computing Systems (ICCCS-2016), Gurgaon, India, 9–11 September 2016; pp. 9781315364094–121 .
18. Xie, D.; Chen, L.; Liu, L.; Chen, L.; Wang, H. Actuators and sensors for application in agricultural robots: A review. *Machines* **2022**, *10*, 913. [CrossRef]
19. Ojha, T.; Misra, S.; Raghuwanshi, N.S. Wireless sensor networks for agriculture: The state-of-the-art in practice and future challenges. *Comput. Electron. Agric.* **2015**, *118*, 66–84. [CrossRef]
20. Fahmideh, M.; Zowghi, D. An exploration of IoT platform development. *Inf. Syst.* **2020**, *87*, 101409. [CrossRef]
21. Bayano-Tejero, S.; Sola-Guirado, R.R.; Gil-Ribes, J.A.; Blanco-Roldán, G.L. Machine to machine connections for integral management of the olive production. *Comput. Electron. Agric.* **2019**, *166*, 104980. [CrossRef]
22. Beza, E.; Reidsma, P.; Poortvliet, P.M.; Belay, M.M.; Bijen, B.S.; Kooistra, L. Exploring farmers' intentions to adopt mobile Short Message Service (SMS) for citizen science in agriculture. *Comput. Electron. Agric.* **2018**, *151*, 295–310. [CrossRef]
23. Radočaj, D.; Šiljeg, A.; Marinović, R.; Jurišić, M. State of major vegetation indices in precision agriculture studies indexed in web of science: A review. *Agriculture* **2023**, *13*, 707. [CrossRef]
24. Sudhamathi, T.; Perumal, K. Ensemble regression based Extra Tree Regressor for hybrid crop yield prediction system. *Meas. Sensors* **2024**, *35*, 101277. [CrossRef]
25. Nosirov, B.; Fakhriddinova, D. Reducing the cost of products in agroclusters in the digital economy. *J. New Century Innov.* **2023**, *23*, 19–24.
26. Singh, M.K.; Kumar, H.; Gupta, M.; Madaan, J. Analyzing the determinants affecting the industrial competitiveness of electronics manufacturing in India by using TISM and AHP. *Glob. J. Flex. Syst. Manag.* **2018**, *19*, 191–207. [CrossRef]
27. Srivastava, A.K.; Saxena, S.; Yadav, S.K.; Ashok, P. Smart sensing solutions for the growth of agriculture. In *Next-Generation Smart Biosensing*; Elsevier: Amsterdam, The Netherlands, 2024; pp. 43–66.
28. Arrow Electronics. *Agriculture Sensors: Top 5 Sensors Used in Agriculture*; Arrow Electronics, Inc.: Mississauga, ON, Canada, 2020.
29. Pramanik, M.; Khanna, M.; Singh, M.; Singh, D.; Sudhishri, S.; Bhatia, A.; Ranjan, R. Automation of soil moisture sensor-based basin irrigation system. *Smart Agric. Technol.* **2022**, *2*, 100032. [CrossRef]
30. Wang, J.; Zhang, X.; Xiao, L.; Li, T. Survey for Soil Sensing with IOT and Traditional Systems. *Network* **2023**, *3*, 482–501. [CrossRef]
31. Vyavahare, G.D.; Lee, Y.; Seok, Y.J.; Kim, H.N.; Sung, J.; Park, J.H. Monitoring of Soil Nutrient Levels by an EC Sensor during Spring Onion (*Allium fistulosum*) Cultivation under Different Fertilizer Treatments. *Agronomy* **2023**, *13*, 2156. [CrossRef]
32. Li, H.; Guo, Y.; Zhao, H.; Wang, Y.; Chow, D. Towards automated greenhouse: A state of the art review on greenhouse monitoring methods and technologies based on internet of things. *Comput. Electron. Agric.* **2021**, *191*, 106558. [CrossRef]
33. Taub, D. Effects of rising atmospheric concentrations of carbon dioxide on plants. *Nat. Educ. Knowl.* **2010**, *3*, 21 .
34. Thompson, M.; Gamage, D.; Hirotsu, N.; Martin, A.; Seneweera, S. Effects of elevated carbon dioxide on photosynthesis and carbon partitioning: A perspective on root sugar sensing and hormonal crosstalk. *Front. Physiol.* **2017**, *8*, 578.

35. Estévez, J.; Gavilán, P.; Giráldez, J.V. Guidelines on validation procedures for meteorological data from automatic weather stations. *J. Hydrol.* **2011**, *402*, 144–154.
36. Singh, D.K.; Sobti, R.; Jain, A.; Malik, P.K.; Le, D.N. LoRa based intelligent soil and weather condition monitoring with internet of things for precision agriculture in smart cities. *IET Commun.* **2022**, *16*, 604–618.
37. Mydevices. Digital Matter SensorData LoRaWAN. Available online: https://mydevices.com/product/digital-matter-sensordata-lorawan/?srsltid=AfmBOopDsLPIJ5Nj8vm_Df9MVBDKJLsMJiynZL4O9fVnoqcCNWrTJz2C (accessed on 10 October 2024).
38. Ponnusamy, K.; Rajagopalan, N. Internet of things: A survey on IoT protocol standards. In *Progress in Advanced Computing and Intelligent Engineering: Proceedings of ICACIE 2016, Volume 2*; Springer: Singapore, 2018; pp. 651–663.
39. Lee, E.; Seo, Y.D.; Oh, S.R.; Kim, Y.G. A Survey on Standards for Interoperability and Security in the Internet of Things. *IEEE Commun. Surv. Tutorials* **2021**, *23*, 1020–1047.
40. Khatoon, P.S.; Ahmed, M. Importance of semantic interoperability in smart agriculture systems. *Trans. Emerg. Telecommun. Technol.* **2022**, *33*, e4448.
41. Gurunath, R.; Agarwal, M.; Nandi, A.; Samanta, D. An overview: Security issue in IoT network. In Proceedings of the 2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC) I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2018 2nd International Conference on, Palladam, India, 30–31 August 2018; IEEE: New York, NY, USA, 2018; pp. 104–107.
42. Castillo, A.; Juiz, C.; Bermejo, B. Delay and Disruption Tolerant Networking for Terrestrial and TCP/IP Applications: A Systematic Literature Review. *Network* **2024**, *4*, 237–259. [CrossRef]
43. Jimenez, J.; Koster, M.; Tschofenig, H. IPSO smart objects. In Proceedings of the Position paper for the IOT Semantic Interoperability Workshop, San Jose, CA, USA, 17–18 March 2016.
44. Sahni, N.; Bose, J.; Das, K. Web apis for internet of things. In Proceedings of the 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Bangalore, India, 19–22 September 2018; IEEE: New York, NY, USA, 2018; pp. 2175–2181.
45. Quinn, C.; McArthur, J. Comparison of Brick and Project Haystack to Support Smart Building Applications. *arXiv* **2022**, arXiv:2205.05521.
46. Quan, D.; Huynh, D.; Karger, D.R. Haystack: A platform for authoring end user semantic web applications. In Proceedings of the International Semantic Web Conference, Sanibel Island, FL, USA, 20–23 October 2003; Springer: Berlin/Heidelberg, Germany, 2003; pp. 738–753.
47. Li, Y.; Huang, X.; Wang, S. Multiple protocols interworking with open connectivity foundation in fog networks. *IEEE Access* **2019**, *7*, 60764–60773.
48. Lee, J.C.; Kim, H.J.; Kim, S.H. Bridging OCF devices to legacy IoT devices. In Proceedings of the 2017 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Republic of Korea, 18–20 October 2017; IEEE: New York, NY, USA, 2017; pp. 616–621.
49. Cavalieri, S.; Salafia, M.G.; Scroppo, M.S. Towards interoperability between OPC UA and OCF. *J. Ind. Inf. Integr.* **2019**, *15*, 122–137.
50. Paulau, P.; Hurka, J.; Middelberg, J.; Koch, S. Centralised monitoring and control of buildings using open standards. *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci.* **2024**, *10*, 169–176.
51. Liang, S.; Khalafbeigi, T.; van Der Schaaf, H.; Miles, B.; Schleidt, K.; Grellet, S.; Beaufils, M.; Alzona, M. *OGC SensorThings API Part 1: Sensing Version 1.1*; Open Geospatial Consortium: Arlington, VA, USA, 2021.
52. Horsburgh, J.S.; Lippold, K.; Slaugh, D.L. Adapting OGC's SensorThings API and data model to support data management and sharing for environmental sensors. *Environ. Model. Softw.* **2025**, *183*, 106241.
53. Sánchez López, T.; Ranasinghe, D.C.; Harrison, M.; McFarlane, D. Adding sense to the Internet of Things: An architecture framework for Smart Object systems. *Pers. Ubiquitous Comput.* **2012**, *16*, 291–308.
54. Rahman, F.; Ahamed, S.I. Looking for needles in a haystack: Detecting counterfeits in large scale RFID systems using batch authentication protocol. In Proceedings of the 2012 IEEE International Conference on Pervasive Computing and Communications Workshops, Lugano, Switzerland, 19–23 March 2012; IEEE: New York, NY, USA, 2012; pp. 811–816.
55. IPSO Alliance. *IPSO Smart Object Guideline*; IPSO Alliance: San Diego, CA, USA, 2014.
56. Rizwan, A.; Khan, A.N.; Ahmad, R.; Kim, D.H. Optimal environment control mechanism based on OCF connectivity for efficient energy consumption in greenhouse. *IEEE Internet Things J.* **2022**, *10*, 5035–5049.
57. von Straussenburg, A.F.A.; Aldenhoff, T.T.; Riehle, D.M. Extending the SensorThings API Data Model–Improving Interoperability and Use Case Flexibility in IoT. In Proceedings of the 43rd International Conference on Conceptual Modeling (ER 2024) , Pittsburgh, PA, USA, 28–31 October 2024.
58. Al-Sarawi, S.; Anbar, M.; Alieyan, K.; Alzubaidi, M. Internet of Things (IoT) communication protocols. In Proceedings of the 2017 8th International Conference on Information Technology (ICIT), Amman, Jordan, 17–18 May 2017; IEEE: New York, NY, USA, 2017; pp. 685–690.

59. Atalla, S.; Tarapiah, S.; Gawanmeh, A.; Daradkeh, M.; Mukhtar, H.; Himeur, Y.; Mansoor, W.; Hashim, K.F.B.; Daadoo, M. Iot-enabled precision agriculture: Developing an ecosystem for optimized crop management. *Information* **2023**, *14*, 205.
60. Alshagri, A.; Mutairi, A. Evaluation of Modern Internet Transport Protocols over GEO Satellite Links. *Network* **2023**, *3*, 451–468. [CrossRef]
61. Matsuzawa, T.; Ichikawa, K. Implementation and Evaluation of HTTP/3 Connectivity Check Using Happy Eyeballs Algorithm. *Network* **2022**, *2*, 389–397. [CrossRef]
62. Simla. A, J.; Chakravarthy, R.; Leo. L, M. An experimental study of iot-based topologies on MQTT protocol for agriculture intrusion detection. *Meas. Sensors* **2022**, *24*, 100470.
63. Davoli, L.; Ramzan, H.H.M.; Belli, L.; Ferrari, G. CoAP-based Digital Twin Modelling of Heterogeneous IoT Scenarios. In Proceedings of the 10th International Food Operations and Processing Simulation Workshop, Tenerife, Spain, 18–20 September 2024; FoodOPS: Athens, Greece, 2024; pp. 1–5.
64. Dos Santos, R.P.; Leithardt, V.R.Q.; Beko, M. Analysis of MQTT-SN and LWM2M communication protocols for precision agriculture IoT devices. In Proceedings of the 2022 17th Iberian Conference on Information Systems and Technologies (CISTI), Madrid, Spain, 22–25 June 2022; pp. 1–6. [CrossRef]
65. Fathy, C.; Ali, H.M. A secure IoT-based irrigation system for precision agriculture using the expeditious cipher. *Sensors* **2023**, *23*, 2091. [CrossRef]
66. Open Mobile Alliance. *Internet of Things Protocol Comparison*; Open Mobile Alliance: San Diego, CA, USA, 2018.
67. Ertürk, M.A.; Aydın, M.A.; Büyükakkaşlar, M.T.; Evirgen, H. A survey on LoRaWAN architecture, protocol and technologies. *Future Internet* **2019**, *11*, 216. [CrossRef]
68. Sahu, R.; Tripathi, P. A Brief Review on LPWAN Technologies for Large Scale Smart Agriculture. In Proceedings of the International Conference on Advanced Network Technologies and Intelligent Computing, Varanasi, India, 20–22 December 2023; Springer: Cham, Switzerlan, 2023; pp. 96–113.
69. Routis, G.; Roussaki, I. Low Power IoT Electronics in Precision Irrigation. *Smart Agric. Technol.* **2023**, *5*, 100310.
70. Lavric, A.; Petrariu, A.I.; Popa, V. Long range sigfox communication protocol scalability analysis under large-scale, high-density conditions. *IEEE Access* **2019**, *7*, 35816–35825.
71. Piroddi, A.; Torregiani, M. Machine Learning Applied to LoRaWAN Network for Improving Fingerprint Localization Accuracy in Dense Urban Areas. *Network* **2023**, *3*, 199–217. [CrossRef]
72. Rosa, R.L.; Boulebnane, L.; Pagano, A.; Giuliano, F.; Croce, D. Towards mass-scale IoT with energy-autonomous LoRaWAN sensor nodes. *Sensors* **2024**, *24*, 4279. [CrossRef]
73. Mekki, K.; Bajic, E.; Chaxel, F.; Meyer, F. A comparative study of LPWAN technologies for large-scale IoT deployment. *ICT Express* **2019**, *5*, 1–7.
74. Rosendo, M.; Granjal, J. Energy-aware security adaptation for low-power iot applications. *Network* **2022**, *2*, 36–52. [CrossRef]
75. Tiwari, M.K.; Pal, R.; Singh, R.; Singh, A.K.; Kumar, V.; Sharma, S.; Zaib, N. The Comprehensive Review: Internet Protocol (IP) Address a Primer for Digital Connectivity. *Asian J. Res. Comput. Sci.* **2024**, *17*, 178–189.
76. Fraihat, A. Computer networking layers based on the OSI model. *Test Eng. Manag* **2021**, *83*, 6485–6495.
77. Matni, N.; Moraes, J.; Oliveira, H.; Rosário, D.; Cerqueira, E. LoRaWAN gateway placement model for dynamic Internet of Things scenarios. *Sensors* **2020**, *20*, 4336. [CrossRef]
78. Correia, F.P.; Silva, S.R.d.; Carvalho, F.B.S.d.; Alencar, M.S.d.; Assis, K.D.R.; Bacurau, R.M. Lorawan gateway placement in smart agriculture: An analysis of clustering algorithms and performance metrics. *Energies* **2023**, *16*, 2356. [CrossRef]
79. Das, V.V.; Sathyan, A.; Divya, D. Establishing lora based local agri-sensor network through sensor plugin modules and lorawan data concentrator for extensive agriculture automation. In Proceedings of the 2022 IEEE 19th India Council International Conference (INDICON), Kochi, India, 24–26 November 2022; IEEE: New York, NY, USA, 2022; pp. 1–6.
80. ITU-T Technical Specification. TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU , 19 July 2019, International Telecommunication Union. Available online: https://www.itu.int/dms_pub/itu-t/opb/fg/T-FG-DPM-2019-5-PDF-E.pdf (accessed on 18 March 2025).
81. Nostro, N.; Spalazzese, R.; Di Giandomenico, F.; Inverardi, P. Achieving functional and non functional interoperability through synthesized connectors. *J. Syst. Softw.* **2016**, *111*, 185–199. [CrossRef]
82. Nast, M.; Rother, B.; Golatowski, F.; Timmermann, D.; Leveling, J.; Olms, C.; Nissen, C. Work-in-progress: Towards an international data spaces connector for the internet of things. In Proceedings of the 2020 16th IEEE International Conference on Factory Communication Systems (WFCS), Porto, Portugal, 27–29 April 2020; IEEE: New York, NY, USA, 2020; pp. 1–4.
83. Nguyen, H.V.; Iacono, L.L. RESTful IoT authentication protocols. In *Mobile Security and Privacy*; Elsevier: Amsterdam, The Netherlands, 2017; pp. 217–234.
84. Khan, W.; Kumar, T.; Zhang, C.; Raj, K.; Roy, A.M.; Luo, B. SQL and NoSQL database software architecture performance analysis and assessments—A systematic literature review. *Big Data Cogn. Comput.* **2023**, *7*, 97. [CrossRef]

85. Longo, E.; Redondi, A.E.; Cesana, M.; Arcia-Moret, A.; Manzoni, P. Mqtt-st: A spanning tree protocol for distributed mqtt brokers. In Proceedings of the ICC 2020—2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020; IEEE: New York, NY, USA, 2020; pp. 1–6.
86. Giuliano, F.; Pagano, A.; Croce, D.; Vitale, G.; Tinnirello, I. Adaptive algorithms for batteryless lora-based sensors. *Sensors* **2023**, *23*, 6568. [CrossRef] [PubMed]
87. Dai, Y.; Duan, Z.; Ai, D. Construction and application of field investigation support platform for land spatial planning based on GeoServer. *Proc. J. Physics Conf. Ser.* **2020**, *1621*, 012059. [CrossRef]
88. Sinha, B.B.; Dhanalakshmi, R. Recent advancements and challenges of Internet of Things in smart agriculture: A survey. *Future Gener. Comput. Syst.* **2022**, *126*, 169–184. [CrossRef]
89. Eclipse Whiskers.js. Available online: https://github.com/eclipse-archived/whiskers.js (accessed on 7 December 2024).
90. Eclipse Whiskers. Available online: https://projects.eclipse.org/projects/iot.whiskers (accessed on 7 December 2024).
91. FROST®-Server—An Open Source Implementation of OGC SensorThings API. Available online: https://www.iosb.fraunhofer.de/en/projects-and-products/frost-server.html (accessed on 7 December 2024).
92. FROST-Server Documentation. Available online: https://fraunhoferiosb.github.io/FROST-Server/ (accessed on 7 December 2024).
93. Mozilla SensorWeb SensorThings. Available online: https://github.com/mozilla-sensorweb/sensorthings (accessed on 7 December 2024).
94. 52 North SensorWeb Server STA. Available online: https://github.com/52North/sensorweb-server-sta (accessed on 7 December 2024).
95. Hwang, K.; Lee, J.M.; Jung, I.H.; Lee, D.H. Modification of mosquitto broker for delivery of urgent MQTT message. In Proceedings of the 2019 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE), Yunlin, Taiwan, 3–6 October 2019; IEEE: New York, NY, USA, 2019; pp. 166–167.
96. Paniagua, R.; Sultan, R.; Refaey, A. Unified Pandemic Tracking System Based on Open Geospatial Consortium SensorThings API. *arXiv* **2023**, arXiv:2401.10898.
97. Xing, J.; Zhang, Y. Design of Embedded Data Acquisition Integrated System Based on SQLite Database. In Proceedings of the Cyber Security Intelligence and Analytics, Haikou, China, 28–29 February 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 420–428.
98. SensorData LoRaWAN Datasheet. Available online: https://www.digitalmatter.com/wp-content/uploads/2020/11/SensorData-LoRaWAN-Datasheet.pdf (accessed on 7 December 2024).
99. Waspmote Plug & Sense Encapsulated Line. Available online: https://cloud.libelium.com/app/docs/waspmote/waspmote-plug-amp-sense-encapsulated-line (accessed on 7 December 2024).
100. Pires, L.; Martins, J. *Experimental Investigation of Spreading Factor, Payload Length and Collision Effects in LoRaWAN Radio Interface*; Eliva Press: Chișinău, Republic of Moldova, 2024.
101. Fachrizal, F.; Zarlis, M.; Efendi, S. Waspmote-based landslide early disaster detection system with GSM communication. *Proc. J. Physics Conf. Ser.* **2020**, *1566*, 012036. [CrossRef]
102. Mukherji, S.V.; Sinha, R.; Basak, S.; Kar, S.P. Smart agriculture using internet of things and mqtt protocol. In Proceedings of the 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), Faridabad, India, 14–16 February 2019; IEEE: New York, NY, USA, 2019; pp. 14–16.
103. Ferrández-Pastor, F.J.; García-Chamizo, J.M.; Nieto-Hidalgo, M.; Mora-Pascual, J.; Mora-Martínez, J. Developing ubiquitous sensor network platform using internet of things: Application in precision agriculture. *Sensors* **2016**, *16*, 1141. [CrossRef] [PubMed]
104. Huang, C.; Wu, C. Design and implement an interoperable Internet of Things application based on an extended OGC sensorthings API Standard. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* **2016**, *41*, 263–266. [CrossRef]
105. OGC. *OGC Best Practice for using SensorThings API with Citizen Science*; OGC: Arlington, VA, USA, 2021.
106. James, A.; Seth, A.; Mukhopadhyay, S.C.; James, A.; Seth, A.; Mukhopadhyay, S.C. Programming Raspberry Pi for IoT System. In *IoT System Design: Project Based Approach*; Springer: Cham, Switzerland, 2022; pp. 51–79.
107. Nemer, M.A.; Azar, J.; Makhoul, A.; Bourgeois, J. Leveraging AI for Enhanced Semantic Interoperability in IoT: Insights from NER Models. In Proceedings of the 2024 International Wireless Communications and Mobile Computing (IWCMC), Ayia Napa, Cyprus, 27–31 May 2024; IEEE: New York, NY, USA, 2024; pp. 1351–1357.
108. Nawaratne, R.; Alahakoon, D.; De Silva, D.; Chhetri, P.; Chilamkurti, N. Self-evolving intelligent algorithms for facilitating data interoperability in IoT environments. *Future Gener. Comput. Syst.* **2018**, *86*, 421–432.
109. Karabulut, E.; Sofia, R.C. An Analysis of Machine Learning-Based Semantic Matchmaking. *IEEE Access* **2023**, *11*, 27829–27842.
110. Shaikh, T.A.; Rasool, T.; Veningston, K.; Yaseen, S.M. The role of large language models in agriculture: Harvesting the future with LLM intelligence. *Prog. Artif. Intell.* **2024**. [CrossRef]

111. Fountas, S.; Espejo-García, B.; Kasimati, A.; Gemtou, M.; Panoutsopoulos, H.; Anastasiou, E. Agriculture 5.0: Cutting-edge technologies, trends, and challenges. *IT Prof.* **2024**, *26*, 40–47.
112. Li, S.; Liu, J.; Yang, Y.; Shen, R.; Jiang, J. Thinking as Human: Self-Reflective Reinforcement Learning Framework for Fertilization Decision-Making. *Smart Agric. Technol.* **2025**, *10* , 100841. [CrossRef]
113. Pagano, A.; Amato, F.; Ippolito, M.; De Caro, D.; Croce, D.; Motisi, A.; Provenzano, G.; Tinnirello, I. Machine learning models to predict daily actual evapotranspiration of citrus orchards under regulated deficit irrigation. *Ecol. Inform.* **2023**, *76*, 102133.
114. Li, X.; Gong, Z.; Zheng, J.; Liu, Y.; Cao, H. A survey of data collaborative sensing methods for smart agriculture. *Internet Things* **2024**, *28*, 101354.
115. Roussaki, I.; Doolin, K.; Skarmeta, A.; Routis, G.; Lopez-Morales, J.A.; Claffey, E.; Mora, M.; Martinez, J.A. Building an interoperable space for smart agriculture. *Digit. Commun. Netw.* **2023**, *9*, 183–193.
116. López-Riquelme, J.; Pavón-Pulido, N.; Navarro-Hellín, H.; Soto-Valles, F.; Torres-Sánchez, R. A software architecture based on FIWARE cloud for Precision Agriculture. *Agric. Water Manag.* **2017**, *183*, 123–135. [CrossRef]
117. Silva, N.; Mendes, J.; Silva, R.; dos Santos, F.N.; Mestre, P.; Serôdio, C.; Morais, R. Low-cost IoT LoRa® solutions for precision agriculture monitoring practices. In Proceedings of the Progress in Artificial Intelligence: 19th EPIA Conference on Artificial Intelligence, EPIA 2019, Vila Real, Portugal, 3–6 September 2019; Proceedings, Part I 19; Springer: Cham, Switzerland, 2019; pp. 224–235.
118. Taji, K.; Ghanimi, F. Proposed Architecture for Smart Irrigation System: Leveraging IoT and LoRaWAN. In Proceedings of the The International Workshop on Big Data and Business Intelligence, Bangkok, Thailand, 16–19 November 2024; Springer: Cham, Switzerland, 2024; pp. 11–22.