LEGO+: Redefining the Redundancy Removal for IoT Sensing Edge-End Systems

Chong Zhang[1,2,3], Han Wang[1], Qianhe Meng[1], Yizhe Zhao[1], Yihang Song[1], Kanglin Xu[1], Jinzhe Li[1], Li Lu[1] ✉

[1]University of Electronic Science and Technology of China (UESTC), Chengdu, China

[2]Southwest Petroleum University, Chengdu, China

[3]Sichuan Artifcial Intelligence Research Institute, Yibin, China

{zhangchong,wang_han,qianhe,zhaoyize,kanglin,uestc_lijinzhe}@std.uestc.edu.cn,songyihang@uestc.edu.cn

✉Corresponding Author, luli2009@uestc.edu.cn

ABSTRACT

The Internet of Things (IoT) can only thrive if IoT sensor nodes can be effortlessly deployed and maintained without compromising their general-purpose nature. However, existing low-power sensor systems fail to strike a balance between these two issues, leaving the widespread of IoT sensor nodes as an open problem. In this paper, we propose LEGO+ as a minimalist yet general-purpose sensing edge-end architecture. Instead of running embedded software on a redundant general-purpose microprocessor, LEGO+ can directly construct the desired control functionality for various IoT sensing applications through hardware-level logic orchestration. To achieve this, we first conduct an in-depth analysis of the underlying unit behaviors within IoT sensor systems and, based on this, abstract a uniform logic orchestration model. Next, to enable sensor nodes to comprehend and execute the generated logic, we devise a hierarchical atomic control circuit with negligible overheads. Finally, we develop a task state prediction scheme to further improve the overall operation efficiency among multiple nodes. We prototype LEGO+ for proof-of-concept and conduct comprehensive experiments, and the results demonstrate that LEGO+ can reduce the overall power consumption of sensor nodes by 86% and enhance task efficiency by 49%, thereby facilitating a wider array of IoT sensing applications.

CCS CONCEPTS

• **Computer systems organization** → **Sensor networks**; *High-level language architectures*; • **Hardware** → *Programmable logic elements.*

KEYWORDS

Internet of things, Edge-End System, Sensor Nodes

ACM Reference Format:

Chong Zhang[1,2,3], Han Wang[1], Qianhe Meng[1], Yizhe Zhao[1], Yihang Song[1], Kanglin Xu[1], Jinzhe Li[1], Li Lu[1] ✉. 2025. LEGO+: Redefining the Redundancy Removal for IoT Sensing Edge-End Systems. In *The 23rd Annual International Conference on Mobile Systems, Applications and Services (MobiSys '25), June 23–27, 2025, Anaheim, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3711875.3729126

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MobiSys '25, June 23–27, 2025, Anaheim, CA, USA

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-1453-5/2025/06

https://doi.org/10.1145/3711875.3729126

1 INTRODUCTION

As of 2023, the global population of Internet of Things (IoT) connections reached approximately 17 billion, and this figure is projected to grow to 29.7 billion by 2027, representing an annual growth rate of 11% [2]. The ever-increasing IoT applications enable people to access a plethora of information from the vast physical world through ubiquitous sensing technologies, which facilitates interactions that transcend spatial boundaries, including precision agriculture [11, 40], smart city [32, 39] and environmental monitoring [1, 6]. The demand for massive sensing applications calls for large-scale, long-term, and low-cost data acquisition [19, 28, 42, 43], which requires IoT sensor nodes to be designed to satisfy the following three requirements.

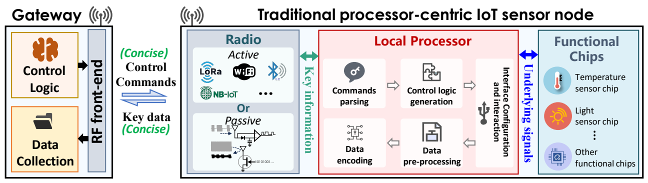
❶ **Minimalist architecture.** Large-scale applications require sensor nodes to be designed as simply as possible with low-power consumption to minimize deployment and maintenance costs.

❷ **Ubiquitous adoption.** Sensor nodes ought to be adapted to a wide range of application scenarios to support diverse sensing tasks corresponding to different sensor chips and communication performance requirements.

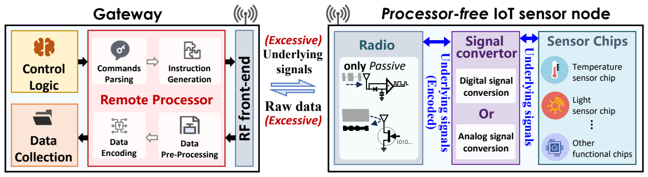
❸ **High task efficiency.** The node should be designed simultaneously to be both low-cost and high-efficiency without impairing the nodes' task efficiency brought by its simplified architecture. Otherwise, it may not pay for itself.

If achieved, IoT sensor nodes can be widely spread and set up as easily as dandelion seeds in the wind, thereby enabling seamless ambient sensing and data collection at any given time and location. However, to the best of our knowledge, there is no existing architecture that satisfies the target.

Specifically, as illustrated in Figure 1(a), to meet the requirements of versatile IoT applications, traditional sensor nodes have to rely on general-purpose computing platforms centered on embedded processors, so that they can control various sensor chips (e.g., sensor chips) by running their application-specific embedded software [12]. However, in such an approach, the required control logic for the target application is essentially "simulated" by running embedded



(a) Traditional IoT sensing edge-end system with processor-centric sensor node.



(b) IoT sensing edge-end system with existing processor-free node.

Figure 1: Architecture of two existing IoT sensor systems. The brute-force approach of completely shifting processor functionality, adopted by the processor-free IoT sensor node, cannot touch the root cause of the redundancy issue suffered by traditional processor-centric IoT sensor nodes.

programs on a general-purpose computer platform[1]. Hence, the microprocessor-based architecture exhibits good versatility, but it is practically inefficient for lightweight sensing tasks due to its complex architecture. That is, when coping with low task loads in commonly simple yet ubiquitous applications (e.g., temperature sensing), its energy consumption does not reduce proportionately as the workload decreases. Besides, for its applications, developers also need to customize the program whenever sensor chips are to be integrated into different types of microprocessors, which increases the development difficulty and thus limits the reduction of application costs.***(contrary to ❶)***

For this issue, numerous efforts have been made, referred to as processor-free architecture [26, 27, 38, 44], aiming to simplify the architecture by replacing the local microprocessor with a specially designed signal conversion mechanism, so that users can directly read digital [21, 22] or analog sensor chips [5, 20] on the node through a nearby gateway using wireless signals, as shown in Figure 1(b). However, such a solution essentially still relies on a remote microprocessor on the gateway side to read on-node sensors. Consequently, a large amount of raw information that was originally exchanged between the processor and sensor chips is transferred to the air interface, resulting in a surge in communication overhead. Therefore, for low-power considerations, those efforts are primarily confined to specific scenarios employing low-power, yet vulnerable and short-range backscatter communications.***(contrary to ❷)***

More seriously, as all control and data processing have been shifted to the gateway, such a processor-free approach cannot be applied to scenarios with stringent real-time requirements. For instance, even a task as simple as a button-pressing operation on the node necessitates a must-round trip to the gateway, not to mention that fewer tasks can be executed for a given bandwidth that is already scarce. ***(contrary to ❸)***

In this paper, we rethink the underlying logic of IoT sensor node control and propose LEGO+, a novel cost-efficient edge-end architecture for next-generation IoT sensing systems. Specifically, we redesign the sensing architecture and simplify the sensor node from a highly complex, independent embedded computer system into a simple smart *wireless peripheral* of the gateway. Instead of relying on general-purpose computing platforms to run embedded programs that "indirectly" simulate the control logic required for target applications, LEGO+ directly outputs the target control logic on sensor nodes by orchestrating unit control logic at the minimal hardware level, akin to constructing with LEGO bricks.

By this, the design of LEGO+ not only simplifies the architecture but also streamlines the development process. To scale the system, developers only need to plug in new sensor chips and focus on the implementation of the sensing task itself, without having to develop the specific microprocessor program. Besides, as LEGO+ directly generates control logic in hardware, which is more efficient than indirectly "simulating" it through the execution of embedded programs, it enables more efficient operation with a simplified architecture and is thus helpful for reducing costs and power consumption in large-scale applications.

However, putting LEGO+ into practice requires to address the following technical challenges:

Challenge 1. First, it is hard for a LEGO+ node to provide the required control logic for diverse applications without relying on a general-purpose computer platform. Specifically, the control functional requirements for massive sensor node applications vary significantly, including the scheduling of their onboard sensor chips and data processing operations. To accommodate these heterogeneous control requirements, as discussed before, existing sensor nodes have to execute the corresponding specified embedded control programs running on either a local general-purpose microprocessor or its gateway-side counterpart.

In LEGO+, we conduct an in-depth analysis of the task logic of IoT sensor nodes and devise a novel Hierarchical Sensing Model. Specifically, the sensing tasks executed on the nodes can be broken down into orchestrations of unit operations (we call them atomic operations, *i.e.*, AtOps) across two distinct levels: the App. task layer and the chipset layer. Building upon this, on the gateway side, we develop a handy Hierarchical Task Description Language (HTDL), accompanied by a lightweight compiler. This combination allows for the uniform description and construction of heterogeneous control logic for a variety of sensing applications by orchestrating these AtOps. On the node side, we build a Hierarchical Atomic Control Circuit (HACC) that parses instructions from the gateway and provides direct atomic logic combinations tailored to diverse sensing tasks at the minimal hardware level while ensuring minimal communication overhead with the gateway.

Challenge 2. It is non-trivial for LEGO+ to handle multi-node task hardware-level coordination without relying on the general-purpose computer platform. Specifically, for large-scale deployments, severe signal collisions may occur when multiple sensor nodes upload their data concurrently, which would decrease the

[1]The microprocessor is essentially a single-chip microcomputer that incorporates the Harvard (or von Neumann) architecture, featuring a complete structure comprising an Arithmetic Logic Unit (ALU), memory (RAM, ROM), Input/Output (I/O) interfaces, and other necessary components, albeit with simplifications specifically in hardware specifications [13, 18, 34]

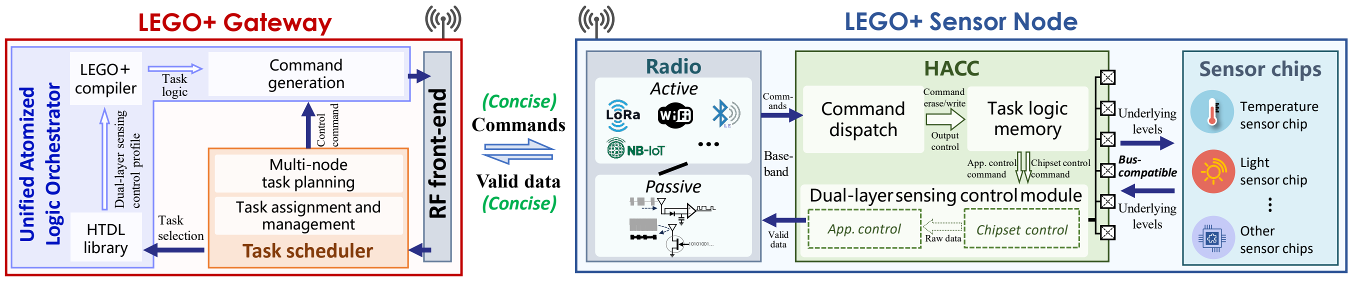


Figure 2: LEGO+ architecture, with system-level low redundancy and high general-purpose advantage.

network throughput and also increase energy consumption and network delay. For this issue, in LEGO+, we design a novel task prediction framework for coordinating multiple sensor nodes, which helps the gateway to accurately forecast data upload time slots for individual sensor nodes by leveraging their unique hardware profiles and task characteristics. Thus, the gateway optimizes the task scheduling for surrounding sensor nodes and effectively avoids data-uploading collisions.

We implemented a comprehensive sensing edge-end system prototype with the edge gateway implemented on Raspberry Pi and the on-node HACC implemented on ultra-low-power FPGA. HACC's control logic part (excluding registers) only takes 2.6 k logic gates. With its ultra-simple architecture, the LEGO+ nodes reduce 86% power consumption compared to traditional processor-centric architectures and exhibit a 4.5× task efficiency compared to previous processor-free architectures. Our contributions are as follows:

❶ We reveal the underlying cause of redundancy and argue that the brute-force approach employed by the processor-free architecture, which shifts control to a remote processor on the gateway with huge communication overhead, is not practical.
❷ We summarize all the task logic of versatile IoT sensing applications and induce it into a few AtOps, and directly assemble the required logic and execute it by orchestrating these AtOps on our newly designed Hierarchical Sensing Model and the HACC, respectively.
❸ We design a lightweight task prediction framework to efficiently coordinate the control of multiple sensor nodes without the need for a general-purpose microprocessor on each node.
❹ We undertake extensive system implementation and comprehensive real-world scenario testing to substantiate the viability and efficacy of the novel LEGO+ architecture, which involves the direct atomization and orchestration of task logic at the hardware level, rather than relying on embedded software running on complex general-purpose computer platforms.

2 LEGO+ IN A NUTSHELL

As shown in Figure 2, a LEGO+ system architecture includes a gateway and multiple sensor nodes, simply put as follows.

2.1 LEGO+ Gateway

The gateway entity in this system can be any sensor node with a communication interface that can be configured with the appropriate operating environment. As presented in the left part of Figure 2, the gateway consists of three components: Hierarchical Sensing Model, Task Scheduler, and radio frequency (RF) front-end.

1) Unified atomized logic orchestrator. It entails the atomized orchestration of atomic control logic on the sensor node, which contains a handy hierarchical task description language (HTDL), along with its lightweight compiler. The syntax of HTDL encompasses a suite of atomic operation (AtOp) keywords, facilitating a unified description of the heterogeneous data processing and sensor control logic inherent in the node. The language structure of HTDL, in this paper, is divided into an App. task layer and a chipset layer. The App. task layer serves to articulate the logic for node-scale data acquisition, data processing and transmission, while the chipset layer is to describe the underlying signal interaction logic for on-board sensor chips. The HTDL compiler is designed to transform the task logic in the HTDL sensing control profile into binary bits, which can be recognized by the HACC. We also elaborated the encoding format for every command to minimize the downlink communication overhead. The details are presented in § 3.

2) Task scheduler. To coordinate tasks on multiple sensor nodes, we design the task scheduler with a task prediction framework. Based on the hardware profiles and task characteristics of the target sensor nodes, the scheduler can predict the time slot for each sensor node to upload sensory data. Then, it will make a timeline plan for the uplink-downlink communication between the gateway and sensor nodes to avoid data collisions and improve efficiency in handling tasks on more sensor nodes without requiring a local processor on each node.

3) RF front-end. The communication technology is not confined due to LEGO+'s optimized communication overhead, which can be active radios, including but not limited to LoRa, WiFi, and BLE, and passive radios (*e.g.*, backscatter).

2.2 LEGO+ Sensor Node

LEGO+ node is accountable for sensing environmental data, performing pre-processing on the data, and then uploading processed valid data to the gateway. As presented in the right part of Figure 2, It has three parts, including HACC, sensor chips, and radio.

1) HACC. With the microprocessor removed from the node, we design the HACC to assemble the required logic by orchestrating atomized operations (AtOps) directly at the minimal hardware level. The circuit contains three modules: command dispatch, task logic memory, and dual-layer sensing control. When it works, the circuit hierarchically stores gateway commands in the memory through the command dispatch module and drives the dual-layer sensing control module to output corresponding control logic (i.e., sensor chip scheduling and data pre-processing) according to an orchestrated sequence generated by the Hierarchical Sensing Model. In contrast to microprocessors, HACC can execute the general-purpose task logic generated by the gateway, which is the crux of the sensor node's simplification, without sacrificing the general purpose. It's a low-redundancy, low-power computing circuit specialized for versatile IoT sensing tasks.

2) Sensor chips. With the universal description for the IoT tasks and implementation of HACC, LEGO+ node can support almost all the commercial off-the-shelf (COTS) sensor chips for diversities of sensing application.

3) Radio. LEGO+ nodes connect to the same network as the gateway. Users can deploy the radio using a UART, SPI, or GPIO interface to connect to the HACC.

3 HIERARCHICAL SENSING MODEL

One major barrier to simplifying IoT sensor nodes lies in the diverse sensing and control requirements of various applications, as it requires the current sensor node control to rely on a general-purpose computer platform with either a local microprocessor or a remote one (located on the gateway). However, this approach also poses difficulties in simplifying the node design (processor-centric architecture), or is accompanied by huge communication overhead (brute-force processor-free architecture). To settle this issue, we analysis in-depth the task logic of IoT sensor nodes and design a novel Hierarchical Sensing Model. It contains HTDL, along with a lightweight compiler, to uniformly describe and construct the heterogeneous control logic for diverse sensing applications. As shown in Figure 3, we design the model by decomposing the gateway-to-sensor node control logic into four layers, which are, from top to bottom, as follows:

Task control layer. This layer of control logic is located at the gateway side. Upon the initial operational startup of the system, the gateway will issue tasks to the nodes in accordance with the preset assignment strategy. Subsequently, the gateway will analyze the incoming sensor node data and orchestrating the tasks of each sensor node to guarantee the efficient system functioning.

App. task layer. In this layer, the node invokes the sensor control logic within the chipset layer to capture raw sensor data. It then proceeds to pre-process the raw data according to commands issued by the gateway. The outcome of this data processing determines the necessity and specificity of data uploading, determining whether and which data should be uploaded to the gateway.

Chipset layer. The layer for sensing control, where nodes handle diverse underlying signaling with sensor chips and provides the sensor outputs to the App. task layer.

Sensor Chip layer. The bottom layer of the closed-loop control. In fact, the sensor chips on the nodes function as the receptors that receive control, where the interaction logic of the upper layer (chipset layer) should be aligned with the internal state machine logic and interface signals of the controlled sensor chip to achieve control and data retrieval of the target sensor chip.

Guided by the theory of the hierarchical sensing model, we design the hierarchical task description language (as detailed in § 4),

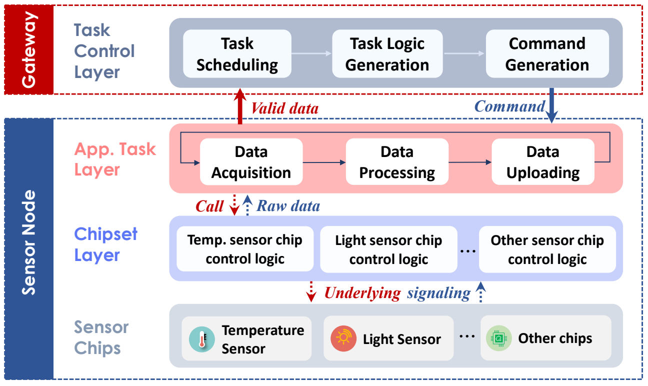


Figure 3: Execution process of Hierarchical Sensing Model.

task scheduler (§ 5) for generating sensing control logic on the gateway, and build a novel Hierarchical Atomic Control Circuit (HACC, detailed in § 6) on the sensor node side to handle sensing task control by orchestrating atomized operations (AtOps) directly at the minimal hardware level and with low communication overheads.

4 HIERARCHICAL TASK DESCRIPTION LANGUAGE

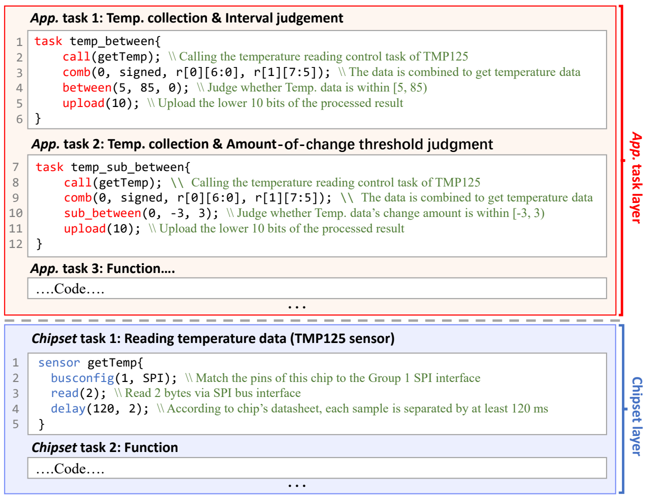
To realize the offloading of task logic generation from the sensor node to the gateway, it's imperative to design a universal description of heterogeneous sensor node task logic.

Basic syntax demonstration. Leveraging the reduced sensor node task model and hierarchical sensor node control framework, we devise HTDL, which serves to articulate both the App. task logic and the chipset logic of the node. The App. task layer syntax is used to describe the data acquisition, data processing, and data upload logic of the node. And the chipset layer syntax is responsible for describing the sensor control logic such as bus read/write, pin level read/write, and event detection. The keywords of the two layers can be found in the anonymous link[2]. We show here an example of how HTDL accomplishes the description of a specific task, as shown in Figure 4. A dual-layer control profile for TMP125 temperature sensor demonstrates that: with only 17 lines of code, it defines two App. tasks and one sensor control task. The App. layer calls the data obtaining function in lower chipset layer and performs accordingly operation on the returned raw sensor data, and then upload the valid reasonable data back to gateway.

These optimized profiles are stored in the HTDL repository, where the gateway's task scheduler dynamically retrieves required configurations. The LEGO+ compiler then processes these profiles to generate binary task command streams.

LEGO+ compiler design. In general, the LEGO+ compiler interprets the dual-layer sensing control profile to formulate the local control logic for LEGO+ nodes, as shown in Figure 5. Specifically, for a given application, users will craft a profile that encapsulates the control drivers for the chipset installed on the LEGO+ node alongside the App. control logic. During runtime, the gateway sequentially parses the HTDL statements within this profile, wherein

[2]The detailed syntax and coding map can be found from this **anonymous link**.


App. task layer
Chipset layer

***App.* task 1: Temp. collection & Interval judgement**

```
task temp_between{
    call(getTemp); \\ Calling the temperature reading control task of TMP125
    comb(0, signed, r[0][6:0], r[1][7:5]); \\ The data is combined to get temperature data
    between(5, 85, 0); \\ Judge whether Temp. data is within [5, 85)
    upload(10); \\ Upload the lower 10 bits of the processed result
}
```

***App.* task 2: Temp. collection & Amount-of-change threshold judgment**

```
task temp_sub_between{
    call(getTemp); \\ Calling the temperature reading control task of TMP125
    comb(0, signed, r[0][6:0], r[1][7:5]); \\ The data is combined to get temperature data
    sub_between(0, -3, 3); \\ Judge whether Temp. data's change amount is within [-3, 3)
    upload(10); \\ Upload the lower 10 bits of the processed result
}
```

***App.* task 3: Function....**

```
....Code....
```

...

***Chipset* task 1: Reading temperature data (TMP125 sensor)**

```
sensor getTemp{
  busconfig(1, SPI); \\ Match the pins of this chip to the Group 1 SPI interface
  read(2); \\ Read 2 bytes via SPI bus interface
  delay(120, 2); \\ According to chip's datasheet, each sample is separated by at least 120 ms
}
```

***Chipset* task 2: Function**

```
....Code....
```

...

Figure 4: Dual-layer sensing control profile of a TMP125 temperature sensor in an application.

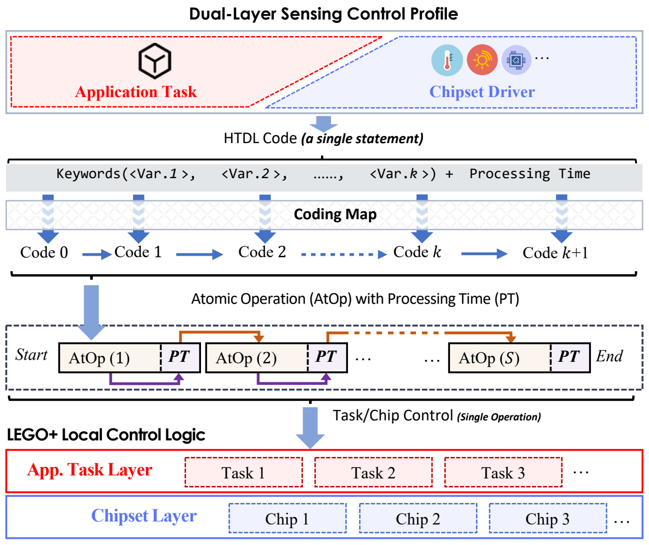

Dual-Layer Sensing Control Profile
Application Task
Chipset Driver
HTDL Code (a single statement)
Keywords(<Var.1>, <Var.2>, ..., <Var.k>) + Processing Time
Coding Map
Code 0
Code 1
Code 2
Code k
Code k+1
Atomic Operation (AtOp) with Processing Time (PT)
Start
AtOp (1)
PT
AtOp (2)
PT
AtOp (S)
PT
End
Task/Chip Control (Single Operation)
LEGO+ Local Control Logic
App. Task Layer
Task 1
Task 2
Task 3
Chipset Layer
Chip 1
Chip 2
Chip 3

Figure 5: LEGO+ Compiler.

each statement's keywords, variables, and the total processing time will be mapped to corresponding encoding. This results in the formation of an indivisible AtOp. (*i.e.*, an AtOp as summarized by HTDL, constituting the smallest on-board operation, which controls a single transition of the sensor's internal state machine, thus indivisible.)

Subsequently, multiple AtOps are concatenated to form the control logic for a specific task or chipset. In practical applications, a certain correlation exists between App. and chipset. For example, in a temperature threshold monitoring application, the App. layer includes invoking sensor read operations (e.g., call TMP125 read), temperature evaluation, and feedback, while the chipset layer executes data acquisition via AtOp sequences managing internal state machine transitions. Hardware constraints (e.g., mandatory sensor state transition delays) enforce sequential execution, where subsequent operations proceed only after prior transitions complete, necessitating an optional 12-bit timing parameter per AtOp. Compiling these operations generates LEGO+'s application-specific local control logic, with chipset drivers fixed per hardware selection and App. logic adaptable to application needs.

5 TASK SCHEDULER

5.1 Control Command Design

The gateway primarily issues two types of control commands to sensor nodes: task assignment commands and task management commands. They have two common field of sensor node ID (identify the node) and command type. The subsequent fields vary according to the command type, with each command having a distinct purpose and encoding format, as outlined below.

5.1.1 Task assignment command. Once a user defines the sensing control profile for a sensor node, the HTDL compiler compiles it into a binary task command stream. The sensor node cannot recognizes and stores these commands until receiving the gateway's task assignment command. The task assignment command enables the node's task command buffer to discern task IDs and allocate memory accordingly. For App. tasks, the command also encapsulates execution periods and start-up delays, allowing for customizable task scheduling. This flexibility in scheduling, especially start-up delays for App. tasks can help mitigate uplink data collisions.

5.1.2 Task management command set. The task management command set includes the task adjustment command, task start-stop command, data retransmission command, and status query command, which are sketched below. ① **Task adjustment command.** This command allows the user to set the execution frequency and startup delay of the App. tasks in real-time during regular system operation; ② **Task start / stop command.** With this command, users can start and stop sensor node tasks at any time through the gateway, which is set to broadcast mode. If the node ID specified in the command matches the broadcast ID (1111), the command becomes applicable to all sensor nodes. Similarly, when the App. task ID within the command coincides with the broadcast ID (111), the command assumes control over the initiation and cessation of all App. tasks associated with the target sensor node; ③ **Data retransmission command and status query command.** When the gateway detects an uplink data collision, it will send a data retransmission command to failed sensor nodes, and the retransmission data type (event / periodic) needs to be designated. The status query command can enable the gateway to know the result of a task assignment on a specific sensor node.

5.2 Multi-node Task Planning

For the extensive deployment of IoT sensor nodes, the sheer number of these nodes is expected to cause significant data collisions, ultimately leading to a decline in network performance. Traditional IoT sensing edge-end systems utilize an asynchronous message-passing protocol for coordinating among sensor nodes. The timing of uplink and downlink communications is often random and unpredictable. However, in LEGO+, the gateway is responsible for

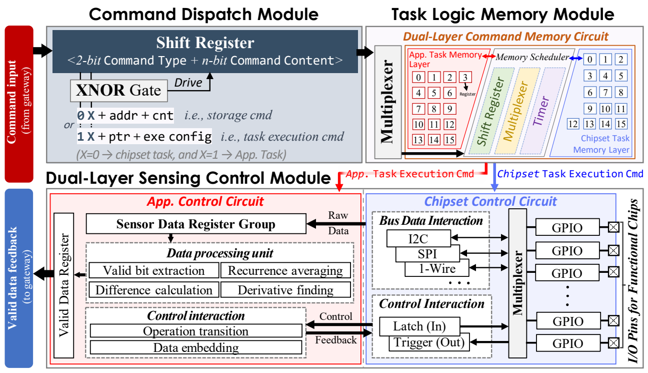


Figure 6: The architecture of HACC.

all task logic, task assignment, and management for the nodes. As a result, the gateway has a precise understanding of each sensor node's task information, execution timing, the number of clock cycles consumed by commands within each task, and the clock frequency.

Specifically, when an unpredictable data collision is encountered (e.g., event-based task conflicts with the data upload of another task), the task scheduler broadcasts a `task stop` command to all sensor nodes to prevent further data collisions. Subsequently, the task scheduler broadcasts an event-based `data retransmission` command and sends periodic task `data retransmission` commands to sensor nodes that were uploading data during the collision time period. The sensor node that receives the retransmission command will re-upload the data after a random delay period. When the collision data has been processed, the gateway will broadcast a task start command, thus resuming the sensor nodes' normal operation.

5.3 Command Generator

The role of the command generator is to create the atomized task command and control command, and finally output the binary data stream, which can be recognized by the node, via the communication interface. The command generation format is described as follows.

Every data transmission dispatched from the gateway to the node incorporates a preamble. The sensor node will start parsing the following control commands only after it recognizes the preamble, while synchronizing all sensor nodes to minimize the risk of packet mis-transmission. Followed by the preamble is the control command content, which can enable the node to identify the node ID and control code, and to configure its other parameters. When the control command is a task assignment command, two field, contains the task command length and content respectively, will be attached behind the control command.

6 HIERARCHICAL ATOMIC CONTROL CIRCUIT

The HACC empowers the node with low redundancy and high general-purpose advantages. This circuit is capable of assembling target control logic driven by concise gateway commands without relying on traditional general-purpose computation systems (*e.g.*, processors). Circuit overview is illustrated in Figure 6, which contains three main components, including a Command Dispatch Module, Task Logic Memory Module, and Dual-layer Sensing Control Module.

1) Command Dispatch Module. This module consists of a *(2+n)*-bit shift register and a *2*-bit XNOR gate. Before commands are issued by the gateway, a 2-bit command type will be added in front of the command content. According to the type field of the incoming command, the Command Dispatch Module will feed subsequent commands into the corresponding modules. Specifically, when the type is `0X` (*i.e.*, this incoming command is identified as *memory command*), Command Dispatch Module will instruct the multiplexer in Task Logic Memory Module to activate the corresponding memory area, and subsequent commands will be stored (*see point 2 for details*). On the other hand, when the type is `1X` (*i.e.*, this incoming command is *task execution command*), Command Dispatch Module will instruct the Dual-layer Sensing Control Module to read the stored commands from Task Logic Memory Module and perform the target sensing control task in the specified configuration (*see point 3 for details*).

2) Task Logic Memory Module. This module consists of a multiplexer and a dual-layer command memory circuit. Each layer of the memory circuit encompasses a register group, with each individual register dedicated to storing the entirety of a command, which facilitates the execution of an App. function or a chipset control function. Specifically, after the Command Dispatch Module identifies the incoming command as a *memory command* and parses its command content, which comprises the `Addr` and the command count (`cnt`), the Task Logic Memory Module will direct a specified count of subsequent commands (*i.e.*, the output of the shift register in Command Dispatch Module) towards the target register. The count is indicated by the `cnt` field within *memory command*. For example, `01+0011_001100` represents the *memory command* (`01`, for chipset task), pointing to the 3_{rd} register (`0011`) in the chipset task command memory layer, and 12 (`001100`) commands received subsequently by the node will be stored from this register, constituting a complete chipset sensing control logic (*e.g.*, data reading and operating mode setting for temperature sensor chip).

3) Dual-layer Sensing Control Module. With the incorporation of the above two modules, the gateway is capable of storing all chipset and application data. Control logic is necessary for managing node control within the DCSC system. Subsequently, scheduling will enable the establishment of the desired sensor node control logic as intended. This module consists of two circuits: one for chipset control and another for application control. It can read commands from Task Logic Memory Module and execute corresponding sensing control operations under the drive of the 32-bit *task execution command*, with the format being: 1X + 4-bit *task pointer* (for function selection) + 26-bit *operational configuration* (including 12-bit *execution frequency* + 10-bit *duration* + 4-bit *priority*). For example, the codes `10 + 0100 + 000111110100_0100101100_0011` indicates scheduling the 4_{th} chipset function, running at a frequency of 500 Hz for 300 seconds (5 minutes), and with a priority level of 3. Upon reception of this chipset *task execution command*, the Command Dispatch Module will drive the Dual-layer Sensing

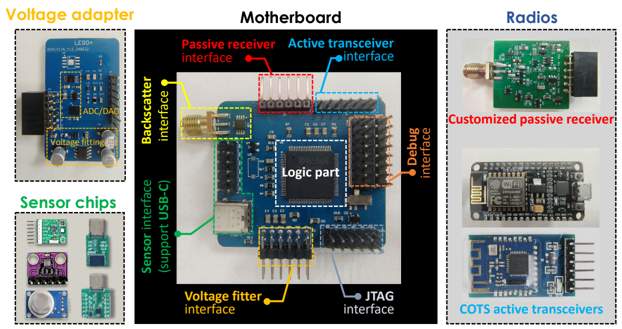


Figure 7: Prototype system of LEGO+ node.

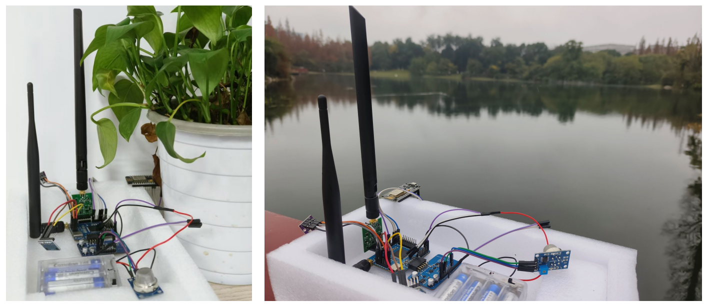


Figure 8: Experiment field. (left: indoor; right: outdoor.)

Control Module to read pre-stored task execution command from the 4_{th} register at the chipset task command memory layer and drive the chipset control circuit to run.

Based on the above design, LEGO+ reduces both communication and computational overhead through a lightweight circuit design. Despite the fact that controlling its local logic generation requires the assistance of an additional layer on the gateway, this process is only necessary when the LEGO+ nodes are first deployed or when their functions need updating. It does not impose any additional communication overhead on the daily operational tasks of the nodes, and it merely affects the normal operation of the gateway.

7 IMPLEMENTATION

We implement LEGO+ gateway using a Raspberry Pi 4B with a 1.5GHz quad-core 64-bit ARM Cortex-A72 CPU, 4GB memory and 16GB TF card space, which is integrated with the same radio module as the node. As for the LEGO+ node, as shown in Figure 7, we design a 4-layer FR4 printed circuit board (PCB) with a set of sensor chips for proof-of-concept.

Specifically, the motherboard of the LEGO+ node contains the HACC (logic part, executed on a low-power AGLN125V2 FPGA [25]) and several interfaces. The HACC is designed to be ultra-lightweight, with its logic part (excluding registers) only taking 2.6 k logic gates. Users can use customized passive radio or COTS active radio (*e.g.*, WiFi, BLE, LoRa, etc.) with a UART, SPI, or GPIO interface to connect to the HACC. Besides, we also provide a voltage adapter to fit the output voltage of some analog sensors, as sometimes their output voltage standards do not match our motherboards. Additionally, LEGO+ can support hundreds of COTS sensor chips and customized sensors with Standard Pins (*e.g.*, 2.54 mm) or USB-C interface for convenient plug-and-play.

In the initial prototype, for the purpose of being convenient to implement, we utilize a low-power FPGA to verify the feasibility and functionality of the proposed digital circuitry in LEGO+. This is a common development process in the field of computer architecture. In the future, if LEGO+ can be successfully promoted and widely adopted, we will IC-enable LEGO+ for much lower cost and power consumption.

8 EVALUATION

In this section, we conduct extensive experiments to evaluate our system and State-of-the-Art (SOTA) solutions.

LEGO+ enables the direct hardware-level atomic logic orchestration of sensor nodes' control logic through the creative HACC design, thus eliminating the need to rely on the local/remote processor. First of all, we evaluate the runtime power consumption and task throughput. We select two representative benchmarks from the processor-centric and -free architecture, respectively, for comparison.

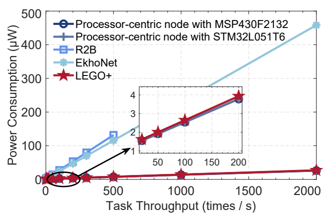
Benchmark of traditional ultra-low-power processor-centric nodes: We let two common low-power processors, STM32L051T6 [36] and MSP430F2132 [14], serve as the processor of two processor-centric sensor nodes. To be fair, we construct minimal system boards for the evaluation, as development boards typically incorporate additional peripherals that increase power consumption, which could lead to unfair experimental conditions.

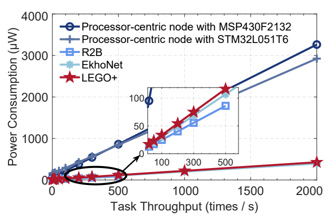
Benchmark of processor-free nodes: Next, for comparing processor-free sensor nodes, we first select the design of R2B [22], a novel Radio-to-SPI-Bus communication scheme to read sensor chips with on-node SPI buses under gateway signals, as the representative benchmark. Next, another computation-free scheme, Ekhonet [46], is also selected to be one benchmark of processor-free nodes.

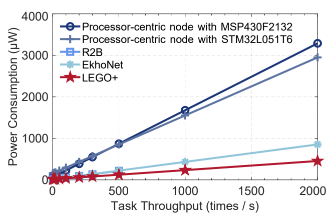
We evaluate all the benchmarks and LEGO+ using the same task, as shown in Figure 8 including: 1) reading the 3-axis acceleration data from the ADXL1004 sensor 6 times and computing the average value; 2) reading temperature data from the TMP125 sensor 2 times and computing the difference value (*i.e.*, mirrors the temperature change); 3) backhaul the result to the gateway. We employ the M8831 micro-power meter [10] to measure the power consumption of every component on the LEGO+ sensor node. We calibrate the experimental results according to the datasheet of used sensors, and the results of processor-free nodes are obtained by the careful analysis of provided experimental results from their papers.

8.1 Power Consumption Comparison

8.1.1 Power consumption with passive communication. In this testing, we employ the passive communication paradigm (*i.e.*, envelope detector for downlink reception and on-off backscatter for uplink backhaul), and evaluate nodes' power of sensor access control (*i.e.*, computation) and communication, as depicted in Figure 9(a), 9(b) and 9(c). We can see that LEGO+ can achieve 77.5 - 90.6% significant reduction, despite that higher communication power, compared to those processor-centric nodes, for a more dramatic cut-down of 78.3 - 91.5% from sensor access control. Compared to the minimal


(a) Communication power.


(b) Computation power.


(c) Total power.

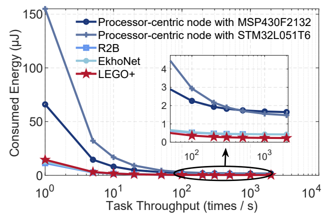
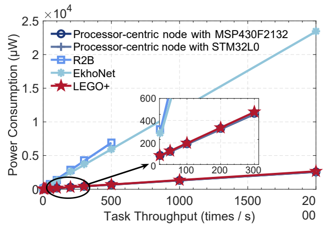
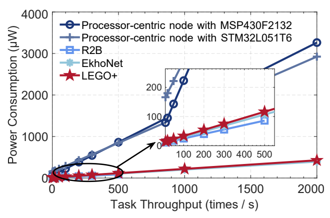
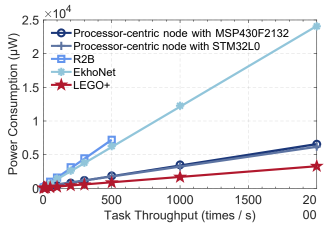

(d) Power per task.

Figure 9: Power with passive communication.


(a) Communication power.


(b) Computation power.


(c) Total power.

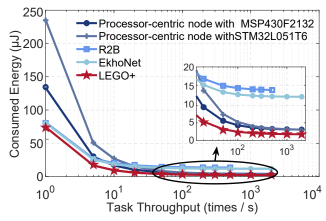

(d) Power per task.

Figure 10: Power with active communication.

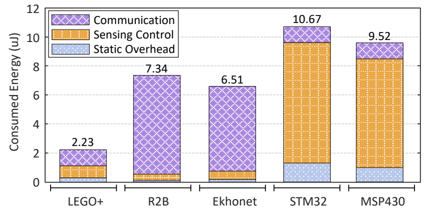
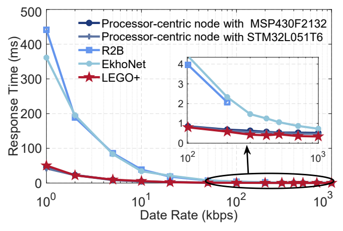


Figure 11: Power breakdown for a single task (average value under active BLE and passive backscatter communication).


(a) Clock freq. = 1MHz.

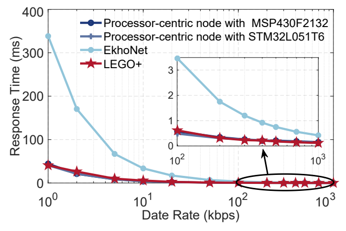

(b) Clock freq. = 4MHz.

Figure 12: The response time per task under different clock frequencies.

processor-free nodes, LEGO+ draws more current when the task throughput is very low. However, when the task throughput becomes higher, LEGO+ outperforms them a lot due to the surge of data over the air of processor-free benchmarks. Specifically, when a task needs to be executed, LEGO+ and processor-centric architecture need to transmit/receive 40-bit data, while R2B and Ekhonet need 384 bits and 336 bits, respectively. This is because the solutions of R2B and Ekhonet essentially rely on a remote microprocessor on the gateway to control the on-node sensor chip, thus a large amount of raw information that was originally exchanged between the processor and sensor chips is transferred to the air interface, leading to a surge in communication overhead. In addition, as an R2B node can only achieve a maximum data rate of 200kbps, it thus has a task throughput of only 500Hz.

8.1.2 Power consumption with active communication. We change the radio to the BLE module, E105-BS21X, which has a maximum data rate of 4 Mbps with 5.3 mA low current consumption. We record node power consumption across different task throughput levels, as shown in Figures 10(a), 10(b), and 10(c). From the results, we can observe that LEGO+ nodes reduce power consumption by 77.3% to 81.9% and 82.7% to 85.2% at task throughput rates of 100Hz and 500Hz, respectively, compared to processor-free nodes. This is because LEGO+ is more efficient than other processor-free solutions that rely on direct control from a remote microprocessor on the gateway side with extensive communication interaction. Besides, for processor-based nodes that control sensing tasks locally, LEGO+ also reduces more than 70% power consumption by its hardware-level atomic control logic orchestration, which systematically minimizes both computational and communication overheads.

8.1.3 Power breakdown per task. Next, we evaluate the power consumption per task executed on nodes. From Figure 9(d) and 10(d), we observe that the power per task becomes lower with improved task throughput. LEGO+ outperforms all benchmarks in this evaluation, achieving 55.2% to 94.7% lower power consumption.

Next, to better clarify where LEGO+ specifically saves energy compared to other benchmarks, we breakdown the average energy consumption in each part of a single task in the previous evaluation, and summarize the details in Figure 11.

It can be observed that, benefiting from the lightweight architecture of LEGO+, the energy consumed by its static overhead and

Table 1: Selected sensors in the evaluation

	Sensor Type	Functions
1	ADXL1004	3-axis Acceleration Sensor
2	ADPD188GG	Digital Light Intensity Sensor
3	LIS2MDL	Magnetic field sensor
4	MA782	Rotational Angle Sensor
5	SMT172	Temperature Sensor
6	VM1010	Low-power Microphone
7	LTC2361	12-bit Analog to Digital Converter
8	MAX30102	Heart Rate and Blood Oxygen Saturation

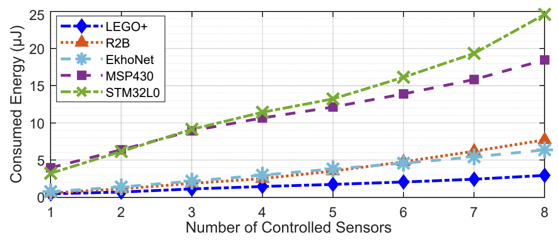
sensing task control in a single task is significantly lower than that of traditional low-power architectures (based on STM32 and MSP430). Furthermore, although the processor-less architectures of R2B and EknoNet achieve lower energy consumption in static overhead and sensing task control, their communication overhead is much higher due to their direct reliance on remote microprocessors at the gateway. Therefore, they are confined to backscatter communication for low-power considerations. But when considering the average power consumption of both backscatter and BLE communications, their overhead is significantly higher than that of other solutions.

In contrast, the architectural design of LEGO completely eliminates direct dependence on microprocessors (both local solutions at the node side and remote ones at the gateway side); its lightweight architecture significantly reduces local task power consumption while avoiding an increase in communication overhead.

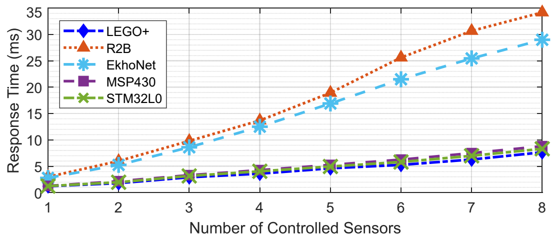
8.2 Task Response Time

Next, we evaluate the task response promptness of nodes. Specifically, we execute the same task as §8.1 on four benchmarks and LEGO+ node, and record the elapsed time required per task execution. To precisely evaluate the performance of each node, the results eliminate the measurement delay inherent to the sensor itself, thereby mitigating the confounding effects introduced by the deployed sensors. We set the nodes to run at two fixed clock frequencies of 1MHz and 4MHz (notably, R2B's clock is provided by the gateway, so its clock frequency is actually equal to its data rate. However, since the maximum data rate of R2B is 200 kbps, it is not tested under a 4 Mbps clock), vary the communication data rate between the gateway and the nodes, and record the elapsed time required for the tasks. The above procedure is looped 1000 times, and we take the average value as the experimental result, as shown in Figure 12.

The experimental results show that the LEGO+ node can complete tasks faster than both the traditional processor-centric node and the processor-free node at the same communication data rate and local clock frequency. The reason for this is twofold. First, compared to traditional processor-centric nodes, the LEGO+ node employs a direct hardware circuit programming method for control logic, which is more efficient than the indirect method used by traditional nodes. The latter relies on software programs running on the processor to simulate the desired control logic. Specifically, the LEGO+ node completes a single task in just 280 clock cycles,



(a) Energy Consumption

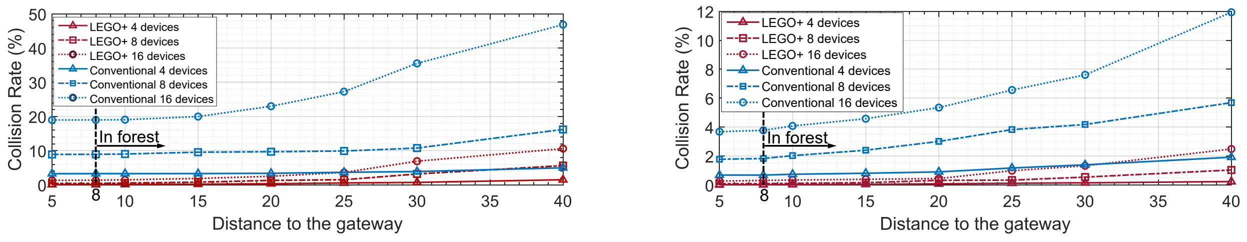


(b) Time Cost

Figure 13: Energy consumption and time cost for a single task under different complexities (with different numbers of sensors needing to be controlled).

whereas the STM32L051T6 processor requires 384 instruction cycles. Given that the STM32L051T6's command execution performance is 0.95 DMIPS (meaning it can execute only 0.95 million commands per million clock cycles), it effectively takes over 400 clock cycles to accomplish the same task. Second, the total amount of data to be transferred by LEGO+ is only slightly higher than that of the traditional processor-centric node. Specifically, only a small amount of control commands need to be conveyed during configuration, while the data transmission load during actual task execution is almost the same as that of the traditional node. As a result, under the same communication data rate and local clock, LEGO+ has achieved a 20% to 48% improvement in task throughput compared to the traditional node.

Meanwhile, for processor-free nodes, the LEGO+ node operates independently of the gateway microprocessor, whereas the processor-free node must interact extensively with the gateway to exchange substantial amounts of raw data. LEGO+ node transmits only 80 bits of data per task, in contrast to R2B and Ekhonet, which require the transmission of hundreds of bits of raw data. For R2B, all task data is managed by the gateway communication, making its task duration nearly equivalent to its communication time. However, due to the significant volume of data, it must transfer — encompassing not only the raw data from sensors but also various signals (such as chip select/interrupt) necessary for sensor I/O control, which are implemented through coding — the overall process is considerably prolonged. While Ekhonet transmits slightly less data than R2B, it still far exceeds the data volume of LEGO+ nodes. Ekhonet must cache raw data locally, and the local clock cycles consumed are proportional to the amount of communication data. As a result, in some scenarios, its total time consumption can be marginally higher than that of the R2B node. Therefore, for processor-free nodes, the LEGO+ node can reduce task time by



(a) 20 Kbps.

(b) 100 Kbps.

Figure 14: Collision rate under different data rates and deployment conditions.

48.1% to 79.5% under the same communication data rate and local clock frequency. These improvements of LEGO+ underscore the realization of the high task efficiency envisioned in §1.

8.3 Application performance under different task complexity

8.3.1 Energy consumption. Next, we evaluate the application performance of our LEGO+ node and the benchmarks across different task complexities, including power consumption and response efficiency.

Specifically, in practical applications, the complexity of tasks is positively correlated with the number of sensor chips that need to be controlled. For this reason, we select a total of 8 different sensor chips for testing (as summarized in Table 1). We first deploy the first sensor on each node and gradually add subsequent sensors that need to be controlled on the node to change the task complexity. In the evaluation, each task involves reading all the onboard sensors and computing the average of 4 consecutive readings to reduce sensing errors and improve accuracy, and then making data aggregation from the controlled sensors and uploading it to the gateway. Considering that the benchmarks of R2B and Ekhonet are tailored for low-power passive communication, hence we use the same passive communication configurations in Section 8.1.1. The evaluation results are recorded in Figure 13(a).

We find that LEGO+ exhibits the lowest power consumption, as well as the slowest increase in power consumption as task complexity (i.e., the number of sensors requiring control) increases. Although the processor-free architectures of R2B and EkhoNet also exhibit lower power consumption compared to traditional architectures (based on msp430 and stm32), their overall power consumption is still significantly higher than that of LEGO+, as they essentially rely on a remote microprocessor at the gateway to control local sensors, resulting in substantial communication overhead (despite using ultra-low-power backscatter communication). Furthermore, as task complexity increases, their overall power consumption rises more rapidly due to the need for increased information exchange with the gateway to control the additional deployed sensors. In contrast, LEGO+ handles tasks locally with its minimalist circuit design, achieving a simplified architecture without significantly increasing communication overhead, thus attaining the lowest overall power consumption.

8.3.2 Response performance. Next, we evaluate the response performance of each node under different task complexities (with a different number of sensors required to be controlled). We record the time elapsed from when a command is issued by the gateway until the data is received back. Meanwhile, to avoid interference introduced by the sensors themselves and to obtain more precise results regarding the response performance of each node, we exclude the time required for the sensors' own measurements. The results are shown in Figure 13(b).

We can see that LEGO+ achieves the lowest response time under different task complexities, which is also even slightly lower than that of conventional processor-based architectures. This is benefited by the fact that LEGO+ directly generates control logic at the hardware level, which is more efficient than running embedded programs on a general-purpose computing platform (microprocessor) to "indirectly" generate the target control logic. By contrast, R2B and EkhoNet have the longest response times because they essentially rely on remote microprocessors at the gateway side for control; the transmission of a large amount of raw information and data significantly increases the time cost. Finally, despite LEGO+ also introducing an extra layer on the gateway to manage its local logic generation, it is only executed once when the LEGO+ node is first deployed and when subsequent functional updates are required, which would not increases communication overhead and response time for daily tasks.

8.4 Multi-node Networking Performance

Next, we evaluate the networking performance of the system. We conduct the testing under both unobstructed conditions and in complex obstruction scenarios. We configure each node to read data from the accelerometer ADXL1004, the microphone VM1010, and the 12-bit ADC LTC2361 for a random number of times between 1 and 16, at a sampling rate of 200 Hz. We then calculate the average value and upload it back to the gateway. Besides, to evaluate the performance under harsh application conditions, we employed low-power but vulnerable backscatter communication.

For the scenario, we deploy the gateway by the lake as shown in Figure 8, and the nodes are deployed within the forest on the same side as the gateway. There is a 6-meter-wide grassy area by the lake, so when the nodes are deployed within 6 meters of the gateway, the signal transmission faces no obstruction. However, when the distance exceeds that, the nodes are deployed within the forest, resulting in significant obstruction of the signal during transmission. We deploy varying numbers of our LEGO+ nodes and conventional nodes (based on MSP430 or STM32) at different distances from the

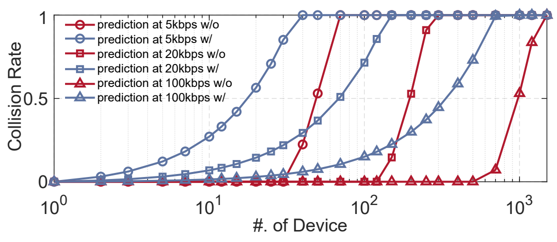


Figure 15: Collision rate at different data rate.

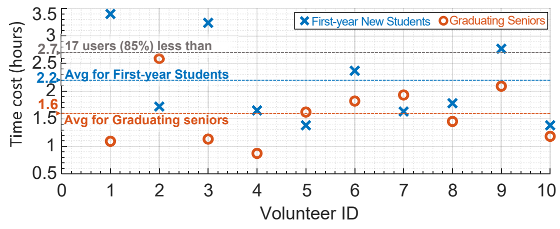


Figure 16: Time cost for learning HTDL.

gateway, and conduct evaluations at communication data rates of 20 kbps and 100 kbps, respectively. We record the collision rate during task execution, as shown in Figure 14.

It can be observed that, under the same experimental conditions, the collision rate of LEGO+ is consistently lower than that of traditional nodes. This is attributed to the anti-collision mechanism integrated into the multi-node task scheduler (see §5) in our LEGO+ design, which effectively reduces the collision rate through coordinated task management. However, when the nodes are deployed beyond eight meters (entering the forest that presents complex obstructions in data transmission), the collision rates for all nodes increase. This is due to the unstable signal transmission leading to packet loss, causing the device to need to retransmit, which in turn elevates the collision rate.

In contrust, benefiting from the anti-collision coordination mechanism in LEGO+, its collision rate still increases at the slowest pace. Specifically, at a communication data rate of 20 kbps and a distance of 30 meters from the gateway, the packet collision rate for 16 LEGO+ nodes is below 10%, whereas that of traditional nodes exceeds 30%. At 100 kbps, as the transmission of data packets reduces, the collision rate decreases, but LEGO+ also achieves the lowest value compared to that of traditional nodes. Moreover, benefiting from the low-power architecture, when data collisions occur, the energy consumption for retransmission in LEGO+ is also lower than that of traditional architectures. Therefore, LEGO+ is conducive to large-scale applications.

Next, to verify the theoretical performance of networking in large-scale node deployment, we conduct a simulation experiment. Specifically, based on the operational mechanism of LEGO+ nodes, we implement a simulation model on a computer. The LEGO+ node clock frequency is set to 200 kHz, and the experiment is carried out at communication data rates of 5 kbps, 20 kbps, and 100 kbps. The task collision rate, which varies with the number of nodes at different data rates, is recorded and is shown in Figure 15.

The results show that the multi-node control performance is significantly improved with our task prediction scheme. At 5kbps, without the task prediction scheme, due to the random upload of node data, even if only 10 nodes are running simultaneously, the task packet collision rate is as high as 27.07%. However, after adopting it, the gateway can predict the upload time of the task data packet according to nodes' hardware parameters and task plan made by itself. At 20kbps and 100kbps, it shows a significant advantage, with data collisions occurring after 120 and 500 nodes, respectively. In contrast, without the task prediction, even at 100 kbps, 13% of the data from 10 nodes will collide.

8.5 Easy-to-learn of HTDL

We design LEGO+ to simplify not only the architecture but also to streamline the development process. Specifically, the HTDL in LEGO+ is directly designed for controlling sensor chips and performing sensing tasks, which contains only a few keywords and has a simple syntax, making it convenient for use in practical sensing applications.

To evaluate the accessibility of learning HTDL, we recruited 20 computer science students (comprising 10 first-year students and 10 graduating seniors) as volunteers. We systematically document the time required from their initial exposure to HTDL until they are able to develop control programs for the sensing tasks on the sensor chip in Section 8.1 using the framework. The results are illustrated in Figure 16.

We find that HTDL is easy to learn, as the learning time for 17 volunteers (85%) is less than 2.7 hours. For first-year students, the average learning time is 2.2 hours, while for graduates, the time cost is only 1.6 hours. Besides, as HTDL is directly oriented towards sensing tasks. Hence, to scale the system with new functions in practical applications, developers only need to plug in new sensor chips and focus on implementing the sensing task itself. It is user-friendly and conducive to large-scale applications.

9 RELATED WORK

Today, low-end IoT sensor nodes hold the potential for realizing the real ubiquitous sensing expectations benefiting from their low-cost advantage. To reduce application cost and power consumption, a rich set of efforts have been made to simplify IoT sensor nodes, mainly in three ways:

Design of hardware abstracted systems: To reduce power consumption, many solutions have focused on hardware-abstracted processing systems, such as deploying tiny operating systems (Tiny OS) [4, 7, 29, 45] on the sensor nodes. However, while TinyOS offers great versatility and flexibility for various applications, its inherent complexity is higher than that required for lightweight sensing tasks such as temperature, humidity, and acceleration sensing. Therefore, it is not efficient for the lightweight sensing scenarios targeted by LEGO+.

Removing the computation: To reduce power consumption, many attempts [26, 27, 33, 38] have been made to directly control RF switches with the output voltage of on-board analog sensors for processor-free backscatter communication, so as to upload the target sensory data without requiring a local microprocessor on the nodes. However, such an approach necessitates the customization of an RF front-end solely for a specific sensor chip and application, inherently limiting its flexibility to adapt to diverse contexts.

Shifting the computation to the gateway: To possess a certain degree of universality while simplifying the architecture, some efforts try to simplify the on-board computation by shifting it to the gateway, including radio chain components [23, 35, 41], data streaming [46] and sensor control [21, 22, 24, 44]. However, in such an approach, the universal compatibility of sensor nodes is facilitated by a remote microprocessor situated on the gateway side, albeit accompanied by significant communication overheads. When it works, the gateway executes the intended embedded program via its microprocessor and converts the internal signals into wireless signals for interaction with the sensor nodes, thereby facilitating control over the sensor chips located on the node side. This process is accompanied by significant communication overheads, including raw data exchange and control interaction overhead.

While there are also ultra-low power analog-only approaches [5, 20], they essentially still rely on remote microprocessors reading local analog sensors directly at nodes via gateways. On the one hand, their RF front-end must be customized or finely tuned to accommodate specific analog sensors, hence lacking universality; on the other hand, they necessitate the transmission of a substantial amount of raw data and control information, thereby also incurring significant communication overhead in practical applications.

Hence, the aforementioned solutions mainly focus on ultra-low-power backscatter communications [3, 8, 9, 15–17, 30, 31, 37] for low-power considerations. However, owing to the low amplitude of the backscattered signal, these efforts not only demand the use of a costly high-sensitivity receiver but also result in short-range and unreliable wireless communication, thereby limiting their large-scale applicability.

In contrast, with the novel architectural design, LEGO+ removes on-board computation redundancy while maintaining low communication overhead for diverse IoT applications through direct hardware-level atomic logic orchestration.

10 DISCUSSION

We discuss four key concerns about the current system prototype.

Sensing function and application scenarios. In the initial stage, we designed LEGO+ as a cost-effective solution for lightweight sensing applications. Its control circuit (HACC) incorporates the necessary atomic operation units to orchestrate the required control logic for digital sensors, and it can also control analog sensors by deploying an analog-to-digital converter (ADC) chip. Therefore, theoretically, arbitrary sensor logic can be implemented using HACC without incurring excessive hardware costs for lightweight sensing tasks. In the future, we may also design pluggable hardware units for complex tasks (*e.g.*, image/video capture and processing) for users in need.

System scalability. The lightweight architectural design of LEGO+ not only simplifies node design but also eliminates the cumbersome development process found in traditional processor-based systems, where on-board sensor chips can be directly controlled by a simple circuit, without the need to program a microprocessor. Therefore, to scale the system with new functions, developers only need to plug in new sensor chips and focus on the implementation of the sensing task itself. It is easy to use and conducive to large-scale applications. Besides, since each LEGO+ node operates independently, the deployment scale of the nodes does not impact the complexity of their local tasks.

11 CONCLUSION

In this paper, we present LEGO+, an ultra-lightweight and universally applicable control architecture tailored for a broad spectrum of IoT sensor-based applications characterized by multi-layered functionality and swift re-configurability. This architecture is distinctive in its ability to directly synthesize the necessary control logic within a minimalist hardware framework, thereby circumventing the need for executing software on intricate, general-purpose computing platforms. To achieve this, we undertake several key initiatives: Firstly, we devise a novel Hierarchical Task Description Language and harness edge gateways to facilitate the seamless orchestration of end-side task logic. Subsequently, we innovate a hierarchical control circuit, which, under the direction of gateway instructions, outputs the requisite control logic for a diverse array of sensing tasks through direct, hardware-level atomic logic orchestration. Furthermore, we develop a suite of task scheduling mechanisms aimed at augmenting the overall efficiency of controlling multiple sensor nodes. Our experimental results show that the LEGO+ architecture achieves a remarkable 86% reduction in the overall power consumption of the node, while enhancing task efficiency by 49%, which sheds light to effortless IoT sensing system deployment.

ACKNOWLEDGMENTS

We sincerely thank all reviewers for their time in reviewing our paper and providing valuable feedback. This paper is partially supported by the National Natural Science Foundation of China under Grant U21A20462 and by the Youth Science Foundation Project of Sichuan Province under Grant 2024NSFSC1454.

REFERENCES

[1] Md Abdulla Al Mamun and Mehmet Rasit Yuce. 2019. Sensors and systems for wearable environmental monitoring toward IoT-enabled applications: A review. *IEEE Sensors Journal* 19, 18 (2019), 7771–7788.

[2] IoT Analytics. 2023. State of IoT 2023. https://iot-analytics.com/number-connected-iot-devices.

[3] Dinesh Bharadia, Kiran Raj Joshi, Manikanta Kotaru, and Sachin Katti. 2015. Backfi: High throughput wifi backscatter. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 283–296.

[4] Ab Wahid Bhat and Abhiruchi Passi. 2022. Wireless sensor network motes: A comparative study. In *2022 9th International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE, 141–144.

[5] Nagarjun Bhat, Agrim Gupta, Ishan Bansal, Harine Govindarajan, and Dinesh Bharadia. 2024. ZenseTag: An RFID assisted Twin-Tag Single Antenna COTS Sensor Interface. In *Proceedings of the 22nd ACM Conference on Embedded Networked Sensor Systems*. 336–350.

[6] Sourav Kumar Bhoi, Sanjaya Kumar Panda, Kalyan Kumar Jena, Kshira Sagar Sahoo, NZ Jhanjhi, Mehedi Masud, and Sultan Aljahdali. 2022. IoT-EMS: An internet of things based environment monitoring system in volunteer computing environment. *Intell. Autom. Soft Comput* 32, 3 (2022), 1493–1507.

[7] Rajashree V Biradar and Anita Patil. 2020. Priority Arbiter for TinyOS: The need of renown OS for WSN and IoT. *International Journal of Engineering and Advanced Technology* 10, 1 (2020), 281–286.

[8] Yoon Chae, Zhenzhe Lin, Kang Min Bae, Song Min Kim, and Parth Pathak. 2024. {mmComb}: High-speed {mmWave} Commodity {WiFi} Backscatter. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 1713–1729.

[9] Zicheng Chi, Xin Liu, Wei Wang, Yao Yao, and Ting Zhu. 2020. Leveraging ambient lte traffic for ubiquitous passive communication. In *Proceedings of the*

Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication. 172–185.

[10] Maynou Electronic. [n. d.]. M8831, Micro-amp Programmable DC Power Supply. https://www.dc-electronic-load.com/micro-amp-programmable-dc-power-supply/.

[11] Robin Gebbers and Viacheslav I Adamchuk. 2010. Precision agriculture and food security. *Science* 327, 5967 (2010), 828–831.

[12] Gaoyang Guan, Wei Dong, Yi Gao, Kaibo Fu, and Zhihao Cheng. 2017. TinyLink: A holistic system for rapid development of IoT applications. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking.* 383–395.

[13] Bill Huston. 1982. Single-chip microcomputers can be easy to program. In *Proceedings of the June 7-10, 1982, National Computer Conference* (Houston, Texas) *(AFIPS '82)*. Association for Computing Machinery, New York, NY, USA, 85–93. https://doi.org/10.1145/1500774.1500786

[14] Texas Instruments. [n. d.]. 16 MHz MCU with 8KB Flash, 512B SRAM, 10-bit ADC, comparator, I2C/SPI/UART. https://www.ti.com/product/MSP430F2132.

[15] Jinyan Jiang, Zhenqiang Xu, Fan Dang, and Jiliang Wang. 2021. Long-range ambient LoRa backscatter with parallel decoding. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking.* 684–696.

[16] Bryce Kellogg, Aaron Parks, Shyamnath Gollakota, Joshua R Smith, and David Wetherall. 2014. Wi-Fi backscatter: Internet connectivity for RF-powered devices. In *Proceedings of the 2014 ACM Conference on SIGCOMM.* 607–618.

[17] Bryce Kellogg, Vamsi Talla, Shyamnath Gollakota, and Joshua R Smith. 2016. Passive {Wi-Fi}: Bringing Low Power to {Wi-Fi} Transmissions. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16).* 151–164.

[18] Kynix. [n. d.]. What is A MCU's internal Structure: Single Chip Micro-Computer. https://www.kynix.com/Blog/What-is-A-MCU-s-internal-Structure-Single-Chip-Micro-Computer.html.

[19] Asif Ali Laghari, Kaishan Wu, Rashid Ali Laghari, Mureed Ali, and Abdullah Ayub Khan. 2021. A review and state of art of Internet of Things (IoT). *Archives of Computational Methods in Engineering* (2021), 1–19.

[20] Liyao Li, Bozhao Shang, Yun Wu, Jie Xiong, Xiaojiang Chen, and Yaxiong Xie. 2024. Cyclops: A Nanomaterial-based,Battery-Free Intraocular Pressure (IOP) Monitoring System inside Contact Lens. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24).* 1659–1675.

[21] Songfan Li, Qianhe Meng, Yanxu Bai, Chong Zhang, Yihang Song, Shengyu Li, and Li Lu. 2023. Go Beyond RFID: Rethinking the Design of RFID Sensor Tags for Versatile Applications. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking.* 1–16.

[22] Songfan Li, Chong Zhang, Yihang Song, Hui Zheng, Lu Liu, Li Lu, and Mo Li. 2020. Internet-of-microchips: Direct radio-to-bus communication with SPI backscatter. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking.* 1–14.

[23] Songfan Li, Hui Zheng, Chong Zhang, Yihang Song, Shen Yang, Minghua Chen, Li Lu, and Mo Li. 2022. Passive {DSSS}: Empowering the downlink communication for backscatter systems. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22).* 913–928.

[24] Qianhe Meng, Han Wang, Chong Zhang, Yihang Song, Songfan Li, Li Lu, and Hongzi Zhu. 2024. Processor-Sharing Internet of Things Architecture for Large-scale Deployment. In *Proceedings of the 22nd ACM Conference on Embedded Networked Sensor Systems.* 211–224.

[25] Microchip. 2024. IGLOO nano low power FPGA. https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ProductDocuments/DataSheets/microsemi_ds0110_igloo_nano_low_power_flash_fpgas_ds.pdf.

[26] Xin Na, Xiuzhen Guo, Zihao Yu, Jia Zhang, Yuan He, and Yunhao Liu. 2023. Leggiero: Analog WiFi backscatter with payload transparency. In *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services.* 436–449.

[27] Saman Naderiparizi, Mehrdad Hessar, Vamsi Talla, Shyamnath Gollakota, and Joshua R Smith. 2018. Towards {Battery-Free} {HD} video streaming. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18).* 233–247.

[28] Sandro Nižetić, Petar Šolić, Diego Lopez-de-Ipiña Gonzalez-De, Luigi Patrono, et al. 2020. Internet of Things (IoT): Opportunities, issues and challenges towards a smart and sustainable future. *Journal of cleaner production* 274 (2020), 122877.

[29] Anita Patil, RM Jagadish, and Sheetal Janthakal. 2024. The TinyOS operating system for wireless sensor networks with a hybrid scheduler. In *2024 Second International Conference on Networks, Multimedia and Information Technology (NMITCON).* IEEE, 1–6.

[30] Yuxiang Peng, Shiyue He, Yu Zhang, Zhiang Niu, Lixia Xiao, and Tao Jiang. 2022. Ambient LoRa Backscatter System With Chirp Interval Modulation. *IEEE Transactions on Wireless Communications* 22, 2 (2022), 1328–1342.

[31] Yao Peng, Longfei Shangguan, Yue Hu, Yujie Qian, Xianshang Lin, Xiaojiang Chen, Dingyi Fang, and Kyle Jamieson. 2018. PLoRa: A passive long-range data network from ambient LoRa transmissions. In *Proceedings of the 2018 conference of the ACM special interest group on data communication.* 147–160.

[32] Aneta Poniszewska-Marańda and Mateusz Kubiak. 2023. Be-in/Be-out System for a Smart City using iBeacon Devices. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking.* 1–3.

[33] Vaishnavi Ranganathan, Sidhant Gupta, Jonathan Lester, Joshua R Smith, and Desney Tan. 2018. Rf bandaid: A fully-analog and passive wireless interface for wearable sensors. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 2 (2018), 1–21.

[34] CADENCE PCB SOLUTIONS. 2022. What is an MCU and How do Microcontroller Units Work. https://resources.pcb.cadence.com/blog/2020-what-is-an-mcu-and-how-do-microcontroller-units-work.

[35] Yihang Song, Li Lu, Jiliang Wang, Chong Zhang, Hui Zheng, Shen Yang, Jinsong Han, and Jian Li. 2023. {μMote}: Enabling Passive Chirp De-spreading and {μW-level} {Long-Range} Downlink for Backscatter Devices. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23).* 1751–1766.

[36] STMicroelectronics. [n. d.]. Ultra-low-power Arm Cortex-M0+ MCU with 32 Kbytes of Flash memory, 32 MHz CPU. https://www.st.com/en/microcontrollers-microprocessors/stm32l051t6.html.

[37] Vamsi Talla, Mehrdad Hessar, Bryce Kellogg, Ali Najafi, Joshua R Smith, and Shyamnath Gollakota. 2017. Lora backscatter: Enabling the vision of ubiquitous connectivity. *Proceedings of the ACM on interactive, mobile, wearable and ubiquitous technologies* 1, 3 (2017), 1–24.

[38] Vamsi Talla, Bryce Kellogg, Shyamnath Gollakota, and Joshua R Smith. 2017. Battery-free cellphone. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 2 (2017), 1–20.

[39] Shuai Tong and Jiliang Wang. 2023. Designing, Building, and Characterizing Large-Scale LoRa Networks for Smart City Applications. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking.* 1–3.

[40] Deepak Vasisht, Zerina Kapetanovic, Jongho Won, Xinxin Jin, Ranveer Chandra, Sudipta Sinha, Ashish Kapoor, Madhusudhan Sudarshan, and Sean Stratman. 2017. {FarmBeats}: an {IoT} platform for {Data-Driven} agriculture. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17).* 515–529.

[41] Han Wang, Yihang Song, Qianhe Meng, Zetao Gao, Chong Zhang, and Li Lu. 2024. Sisyphus: Redefining Low Power for LoRa Receiver. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '24).* 1177–1191.

[42] Jianxin Wang, Ming K Lim, Chao Wang, and Ming-Lang Tseng. 2021. The evolution of the Internet of Things (IoT) over the past 20 years. *Computers & Industrial Engineering* 155 (2021), 107174.

[43] Chong Zhang, Ke Lei, Xin Shi, Yang Wang, Xin Wang, Chuanhui Zhang, Lihu Zhou, Yan Chen, and Hongjun Zhu. 2024. FAST: A Ubiquitous Inference Computation Model for Temperature and Humidity Sensing. *IEEE Sensors Journal* (2024).

[44] Chong Zhang, Songfan Li, Yihang Song, Qianhe Meng, Minghua Chen, YanXu Bai, Li Lu, and Hongzi Zhu. 2023. LEGO: Empowering Chip-Level Functionality Plug-and-Play for Next-Generation IoT Devices. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3.* 404–418.

[45] Ke Zhang, Shunren Hu, and Wei Liu. 2023. Power Consumption Benchmarking of TinyOS Applications with A Dedicated Measurement Adapter. In *2023 5th International Conference on Circuits and Systems (ICCS).* IEEE, 283–287.

[46] Pengyu Zhang, Pan Hu, Vijay Pasikanti, and Deepak Ganesan. 2014. Ekhonet: High speed ultra low-power backscatter for next generation sensors. In *Proceedings of the 20th annual international conference on Mobile computing and networking.* 557–568.

Received 09 December 2025; revised 04 March 2025; accepted 08 April 2025