Enabling Modular and Intelligent IoRT Systems: Integrating the Model Context Protocol for Semantic Decoupling at the Edge

Qize Guo, Yan Chen, Tarik Taleb, and Chao Yang

***Abstract*—The Internet of Robotic Things (IoRT) is shifting the paradigmatic of device-environment engagement through the seamless converge of sensing, actuation, and intelligent reasoning. However, contemporary IoRT systems exhibit strong inter-dependencies between devices and application services, which hinders extensibility and prolongs technological innovation cycles, especially in the context of the rapidly evolving Edge AI landscape. In this paper, we propose a novel integration of the recently-introduced Model Context Protocol (MCP) into edge-level IoRT systems to enable modular service composition and semantic decoupling. By treating devices as callable resources and tools, MCP allows large language models (LLMs) to orchestrate complex workflows through natural language prompts. We design a unified framework where heterogeneous IoRT nodes expose their capabilities through MCP-compliant interfaces, enabling scalable interaction and task automation. A proof-of-concept implementation based on digital twin edge nodes demonstrates the feasibility of the approach, including modular access and decoupled service. We also discuss key challenges such as semantic standardization, security, and edge-level optimization. This work opens new avenues for building generalizable, intelligent, and adaptable IoRT platforms.**

***Index Terms*—Internet of Robotic Things, Model Context Protocol, Modular Service Composition, Semantic decoupling, Large Language Models, Robotic, and Internet of Things.**

I. Introduction

THE Internet of Robotic Things (IoRT) is a specialized subdomain within the Internet of Things (IoT) framework that integrates robotic systems with sensing, communication, computation, and control capabilities. Recently, IoRT has substantially enhanced the autonomy of machines by leveraging advanced sensing and computing technologies within the framework of edge intelligence, which has driven significant innovations in various verticals, including industrial automation [1], agriculture [2], healthcare [3], and logistics. IoRT is expected to play a pivotal role in enabling a wide range of intelligent use cases in future 6G-enabled systems.

As IoRT rapidly evolves, technologies like digital twins, extended reality, and Artificial Intelligence (AI) are becoming integral to its architecture. This convergence enhances coordination, deepens human–machine collaboration, and fosters distributed intelligence. To manage the growing complexity and diversity of devices and services, robust interoperability frameworks are essential for seamless system integration. The middleware layer handles core functions such as device registration, data normalization, and system-wide coordination.For example, in smart farming, IoRT systems integrate sensor data, construct digital twin models, and enable applications such as crop planning and autonomous machinery control. Interactive visual interfaces further provide real-time insights to support human oversight and data-driven decision-making [4].

With the proliferation of IoRT in diverse industries, the inherent heterogeneity of sensors, actuators, and service input requirements poses significant challenges in establishing unified interaction strategies and well-orchestrated workflows. Middleware frameworks (e.g., Robot Operating System (ROS) [5] and FIWARE [6]) have been developed to tackle these issues by providing standardized infrastructure models, unified data service protocols, and orchestration capabilities. Besides, several other frameworks address modularity and orchestration in IoT and edge computing. EdgeX Foundry [7] adopts a microservice-based architecture to enable device interoperability across heterogeneous environments, while Eclipse ioFog [8] focuses on containerized orchestration for distributed edge deployments. KubeEdge [9] extends Kubernetes to the edge, enabling containerized application orchestration and device management across cloud–edge environments. However, while these frameworks facilitate interoperability at the device and service interface levels, they often fall short in supporting dynamic integration and flexible orchestration. Incorporating new hardware or services still requires manual development of adapters, mapping of communication topics or data schemas, and reconfiguration of orchestration workflows. For example, consider two distinct livestock-focused IoRT solutions aimed at improving poultry production: RFID-assisted monitoring [10] and energy-adaptive environmental control [11]. For an operated IoRT system, the extension of new capabilities necessitates extensive evaluation to assess their suitability, requiring costly parallel trials. Subsequently, the selected solution typically demands a comprehensive reconfiguration of the existing middleware. Although middleware frameworks

This work was supported in part by the European Union's under the 6G-Path project (Grant No. 101139172) and the 6GSandbox project (Grant No. 101096328);This research was also partially conducted at ICTFICIAL Oy. The paper reflects only the authors' views, and the European Commission bears no responsibility for any utilization of the information contained herein.

Qize Guo, Yan Chen, and Tarik Taleb are with the Faculty of Electrical and Information, Ruhr University of Bochum, Bochum, 44801, Germany. They are also with ICTFICIAL Oy, Espoo, 02130, Finland. e-mail: Qize.Guo@rub.de, yanchen@ieee.org, tarik.taleb@rub.de.Yan Chen is the corresponding author.

Chao Yang is with the Energy and Mechanical Engineering Department, Aalto University, Espoo, 20150, Finland. e-mail:chao.1.yang@aalto.fi

mitigate certain scalability issues, they tend to confine applications within rigid, domain-specific silos. Given the rapid pace of innovation in electronic information technologies and the continual emergence of novel IoRT devices and services, the demand for modular, semantically interoperable, dynamically orchestrable and truly plug-and-play IoRT architectures has become increasingly urgent.

Beyond the demands for rapid deployment and application, human–machine interaction constitutes a foundational pillar of the IoRT ecosystem, facilitating more effective collaboration through visualized digital twin interfaces and advanced decision support tools. Recent advances in generative AI have fostered the development of powerful Large Language Models (LLMs), such as GPT-4, which exhibit remarkable capabilities in semantic understanding, task decomposition, and logical reasoning [12]. To overcome the limitations of static pretrained knowledge and rigid dialogue structures, OpenAI introduced the "function calling" mechanism, which enables LLM outputs to be programmatically linked to predefined APIs, thereby enhancing system integration and extending functional capabilities. This paradigm has catalyzed a new class of IoRT applications that exploit LLM-driven intent recognition for control, manipulation, data filtering, and augmentation, including the authors' own work on semantic-based remote robotic operation [13]. However, despite these advancements, each LLM platform maintains its own API schema and integration logic. Achieving model-agnostic interoperability thus necessitates the development of custom adapter layers, along with substantial effort in function lifecycle management, version control, testing, and orchestration.

The Model Context Protocol (MCP) has emerged as a promising standardized solution to address integration challenges in systems driven by LLMs [14]. MCP defines a unified interface that enables LLMs to interact with external tools, databases, and services in a structured, secure, and semantically meaningful manner. As a model-agnostic and extensible protocol, MCP allows external functions to be registered as callable resources, each accompanied by formal descriptions that guide LLMs in their invocation. This architecture supports a wide range of operations, such as querying, calculating, and manipulating data, utilizing the semantic reasoning capabilities of LLMs. With growing adoption, MCP servers now provide standardized access to components like code editors, computational engines, and knowledge bases, enabling uniform, model-agnostic interaction across diverse systems. As a result, MCP functions as a semantic integration layer that connects LLM-based reasoning with executable system functions, helping to overcome the persistent fragmentation between data, services, and applications within IoRT environments, complementing existing device-centric and service-centric paradigms.

Despite the increasing adoption of MCP and its rapid advancement in computer software applications, its application within IoRT systems remains largely underexplored. In this work, we introduce a novel framework that integrates MCP into the IoRT architecture, enabling modular and semantically-consistent access to both data sources and control interfaces. By decoupling perception components from high-level task logic, the framework facilitates greater flexibility, reusability, and scalability in the design of IoRT systems. We demonstrate the effectiveness of this approach through a prototype implementation centered on environmental digital twins, and further discuss how the proposed principles can be generalized to support broader classes of IoRT applications. Our major contributions can be summarized as follows:

- We present the first integration of MCP and IoRT systems, enabling modular access to device functionalities through structured and semantically-describable interfaces.
- We design a framework that decouples perception capabilities from application logic, enhancing the flexibility, composability, and maintainability of robotic and sensing systems.
- We implement and evaluate a prototype system that demonstrates how environmental digital twin nodes can expose their sensing and control interfaces to LLMs and other applications through MCP.
- We analyze the extensibility of the MCP-enabled framework across heterogeneous IoRT scenarios, suggesting a generalizable model for future intelligent deployments.

The rest of this paper is organized as follows. Section II introduces the proposed framework that integrates MCP into IoRT systems. A use case implementation based on environmental digital twins is presented in Section III. Section IV discusses the technical challenges and limitations of the proposed framework. Finally, Section V concludes the paper and outlines future research directions.

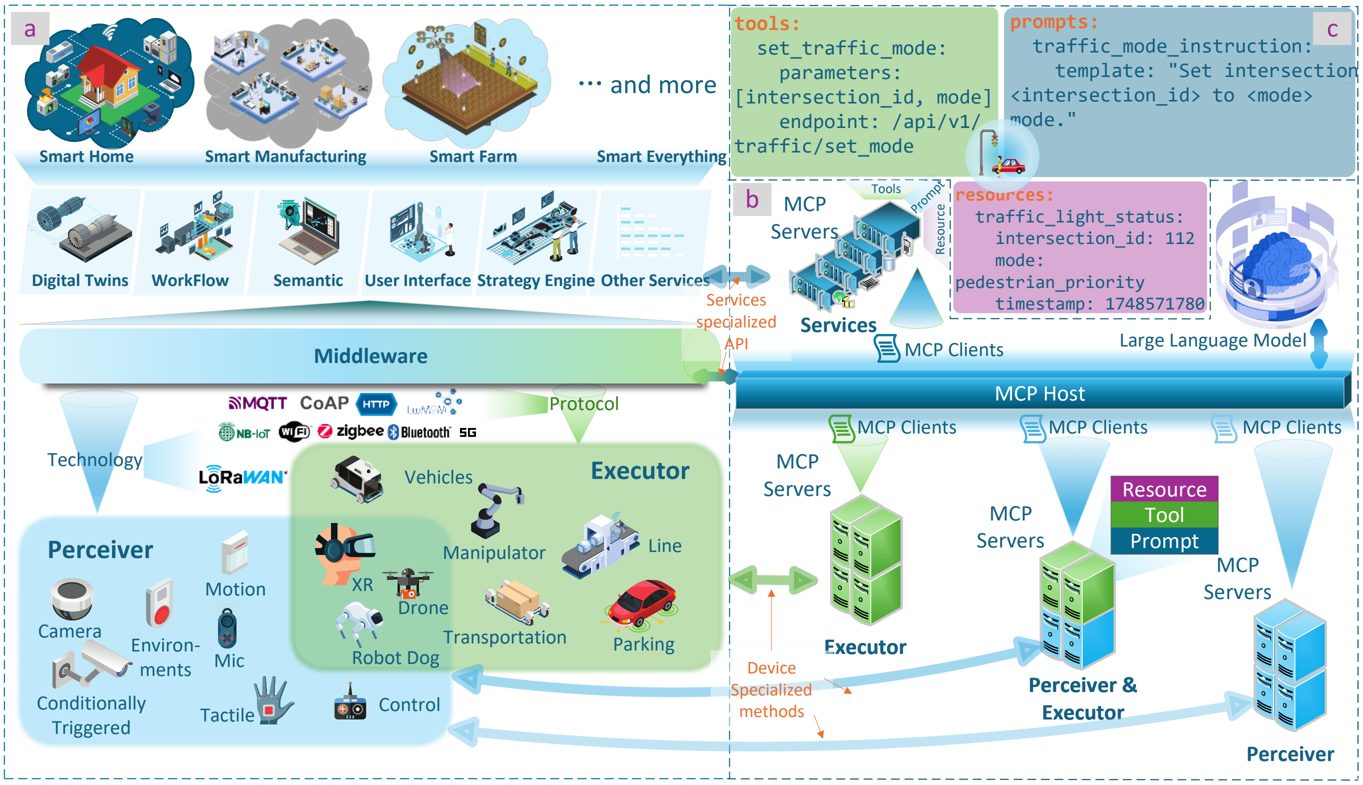
II. The Framework for MCP-Enabled IoRT Systems

This section provides a structured overview of prevailing architectural paradigms in the IoRT, introduces the MCP framework, and presents our proposed integration approach that embeds MCP within the IoRT ecosystem to enhance modularity, scalability, and semantic-level control.

A. Conventional IoRT Architecture

As illustrated in Fig. 1a, the typical conventional architecture of IoRT consists of a layered system comprising perception and actuation components at the edge, interconnected via middleware platforms that provide abstraction and service coordination. Perception components include a broad range of sensors, such as environmental sensors (e.g., temperature, humidity, and pressure), spatial sensors (e.g., GPS), motion detectors, and audio-visual devices. Actuation components range from simple mechanical devices such as motors and switches to complex entities such as robotic arms, autonomous vehicles, and drones, many of which may simultaneously serve as sensors and actuators.

These heterogeneous components are integrated through middleware solutions, which often handle protocol translation and provide standardized interfaces for data acquisition and control commands. Communication is facilitated through various networks (e.g., Zigbee, Wi-Fi, LoRa, NB-IoT, Bluetooth, 5G) and protocols (e.g., MQTT, CoAP, HTTP, LwM2M), selected based on the deployment context.



```
tools:
  set_traffic_mode:
    parameters:
[intersection_id, mode]
    endpoint: /api/v1/
traffic/set_mode
```

```
prompts:
  traffic_mode_instruction:
    template: "Set intersection
<intersection_id> to <mode>
mode."
```

```
resources:
  traffic_light_status:
    intersection_id: 112
    mode:
pedestrian_priority
    timestamp: 1748571780
```

Fig. 1: The proposed MCP-enabled IoRT framework: (a) Conventional IoRT architecture; (b) The MCP framework; (c) Examples of tools, resources, and prompts within the MCP server.

The middleware acts as a bridge between the physical layer and the application layer, enabling upper-layer services such as authentication, logging, monitoring, persistent storage, rule engines, orchestration, reasoning, multi-agent coordination, digital twins, visualization, and Human-Machine Interaction (HMI). Despite the availability of middleware solutions (e.g., ROS, FIWARE, etc.), their coordination typically requires substantial customization to accommodate different message formats, data flows, and system functionalities, leading to significant development overhead and lengthy verification and deployment cycles.

B. MCP Framework

The MCP architecture introduces a novel mechanism for semantically integrating external services, tools, and data into LLM-based systems through structured task interpretation. The standalone MCP architecture is illustrated in Fig. 1b and comprises three primary components: the MCP Host, MCP Client, and MCP Server, each playing a distinct role in facilitating modular, dynamic interaction between LLMs and external functionalities.

1) MCP Host: The MCP Host provides the runtime environment in which the LLM operates and executes commands. It embeds the MCP Client and enables seamless interaction with registered external integrations.

2) MCP Client: The MCP Client serves as an intermediary between the LLM and one or more MCP Servers. It translates high-level, LLM-generated task representations into structured protocol requests (e.g., JSON-RPC 2.0), dispatches them to the appropriate endpoints, and manages runtime behaviors such as progress tracking and interruption handling. The Client also transmits execution results back to the LLM for further reasoning or action.

3) MCP Server: The MCP Server registers external integrations and exposes their capabilities through three core abstractions: Resources (readable data elements such as sensor values or logs), Tools (executable services or functions), and Prompts (predefined templates guiding LLM interactions). Prompts facilitate task orchestration by supporting parameter injection, dynamic configuration, and workflow composition. The MCP Server maintains a queryable and hot-pluggable registry of these elements using standardized formats such as YAML and JSON (as shown in Fig. 1c), thereby supporting scalable and flexible integration.

When used in conjunction with an LLM, MCP enables dynamic tool invocation based on natural language input. For example, in the smart traffic system (Fig. 1c), an instruction like ”*Set intersection 112 to pedestrian priority mode*” is parsed by the LLM, the appropriate MCP tool (e.g., *set_traffic_mode(intersection_id, mode)*) will be identified, and then it will be executed with inferred parameters.

The MCP framework follows a request–response pattern where tools are registered with semantic descriptors. Each workflow step is invoked through intent-based calls, and responses are serialized in JSON for cross-platform compatibility.

C. MCP-enabled IoRT framework

To address limitations in traditional middleware-based IoRT systems with tight coupling, static configurations, and a lack of semantic understanding, we propose a framework that embeds MCP components across the IoRT system. Fig. 1 provides a comprehensive view of the complete system architecture, demonstrating the coordination and interaction among all framework components. At the infrastructure layer, each sensor or actuator can optionally be paired with an MCP server. Perception elements are registered as resources, while actuation functions are exposed as tools. Middleware functions can be also encapsulated via MCP, enabling semantic access to otherwise specific protocol operations. Upper-layer services can utilize their own MCP servers to define reusable prompts for common workflows. The MCP Host provides a centralized, LLM-based orchestration interface, allowing for system-wide semantic interaction. Although MCP integration introduces flexibility and modularity, middleware remains essential for real-time data routing and low-latency control. Therefore, the proposed architecture supports coexistence between MCP and conventional middleware, with deployment strategies determined by specific application requirements. The advantages brought by MCP in IoRT are as follows:

1) Modular Interoperability: Devices, services, or components can be encapsulated into MCP modules by implementing the MCP client/server interface. This promotes uniformity in packaging and invocation.

- Encapsulation abstraction: Devices register capabilities independent of hardware types or network protocols. For example, a traffic node may expose *get_vehicle_count()* as a resource and *set_signal_timing()* as a tool.
- Invocation uniformity: Any MCP-compatible LLM or service can invoke registered tools semantically, enabling domain agnostic integration and inter-system coordination.

2) Decoupling Data from Services: MCP enables logical separation between data sources and service logic. This allows independent evolution of system components through its model-agnostic design and standardized LLM interaction protocols. For example, disparate traffic nodes with varied capabilities can be uniformly managed by a single inference service without requiring interface implementations.

3) Intent-Driven Interaction: By embedding MCP into IoRT, systems gain a full-stack semantic interface. Users can issue natural language commands (e.g., "Schedule a comfortable meeting environment at 3 PM") that are interpreted into actions such as adjusting Heating, Ventilation, and Air Conditioning (HVAC) systems, without manual configuration.

The framework supports real-time hot-plugging of devices and workflows through the native properties of MCP. New devices register as MCP servers without requiring system restart, and their capabilities are automatically exposed as tools in the registry. Workflow updates are achieved by modifying semantic descriptions (typically a few lines of YAML), after which the orchestrator immediately discovers and integrates the new functionalities. This seamless process minimizes developer effort and enables dynamic adaptation in heterogeneous IoRT environments.

D. MCP vs Middleware in IoRT

While traditional middleware has been instrumental in enabling scalable IoRT systems, it falls short in dynamic environments that demand semantic interpretation, service discovery, and workflow agility. Table I contrasts middleware and MCP approaches across several dimensions. MCP excels in flexibility, semantic integration, and modularity, whereas middleware retains its advantage in latency-critical scenarios and legacy system support. Therefore, MCP is not a replacement but rather an augmentation of middleware, providing the semantic layer necessary for future-proof, intelligent IoRT systems.

TABLE I: Comparison of middleware and MCP features

Features	**Middleware**	**MCP**
Access Mechanism	API with custom message formats	Semantic declaration via tools/resources
Workflow Design	Predefined pipelines	Prompt-based, intent-driven workflows
Component Onboarding	Manual configuration	Auto-discovery via MCP Client
Scalability	Tight-coupled, brittle	High, modular and extensible
HMI Integration	Custom UI or interfaces	Native semantic HMI support
Real-Time Action	Strong	Weaker (inference overhead)
Legacy Support	Plug-and-play	Requires redesign

III. Use case in environment digital twins

To validate the feasibility and effectiveness of the proposed integration framework, we implemented a demonstration system that incorporates the MCP protocol into a simplified Internet of Robotic Things (IoRT) scenario. We demonstrate the framework with a self-designed edge device, named Digital Twin Box (DTB), which serves as a representative perception-actuation unit with built-in environmental sensing and multimedia control capabilities. DTB can be flexibly deployed on board static or mobile robots across various intelligent application domains, including smart factories, smart homes, and precision agriculture.

A. DTB Device Overview

DTB is built on RK3588-based computing platform, offering high computational performance, extensive I/O interfaces, and support for on-device AI inference. On the sensing side, it supports configurable environmental sensors (e.g., temperature, humidity, barometric pressure, and CO_2 concentration), enabling adaptation to specific application requirements. DTB provides basic visual sensing capabilities and enables control of multiple environmental switches (e.g., fans, heaters, actuators), with optional modules for networking and display output.

The proof-of-concept was implemented on the DTB with 4 GB RAM, running Ubuntu 20.04. The MCP runtime and orchestration services were implemented in Python 3.8 with MCP-over-MQTT for messaging.

B. MCP Integration Strategy

To evaluate the semantic orchestration capabilities of MCP in an IoRT context, we deployed MCP servers both in the DTB device and the user side for workflow, as shown in Figs. 2b and 2d. Notably, we did not deploy a MCP server in the cloud backend, as the focus was to validate end-to-end task composition between user operations and edge devices via MCP.

On DTB, the MCP server exposes the following resources and tools:

- *resource://iot/devices* provides the online devices for reference for LLM parsing.
- *get_device_state(device_id)* provides real-time environmental data, and returns it in JSON structure.
- *control_device(device_id, operation)* and enable control of the switches for the environmental regulator, the operation can be chosen as one of the following: $temperature\_up$, $temperature\_down$, fan_open, fan_close.

We conducted two sets of experiments in different scenarios to further validate the advantages of MCP integration, specifically its support for modular device/service access and the decoupling of data sources from service logic.

C. Basic Scenario

We begin by a simple scenario involving two devices, each connected to environmental sensors that provide temperature and humidity data, as well as actuators for environmental regulation (such as air conditioners and ventilation fans illustrated in Fig. 2a). Within a smart home context, the system is capable of autonomously regulating indoor comfort. To increase the logical complexity, we assume that decisions regarding the environmental adjustment actions of device *B* are made based on environmental data collected from device *A*. To clearly demonstrate the workflow enabled by MCP integration, we use *Claude Sonnet 4* as the interface for conversational demonstration, though other MCP-compatible LLMs would achieve similar results.

Within our use case, the MCP server defines a tool, *get_policy*, with an *env_adjustment* parameter. The policy returned, as shown in Fig. 2e, contains a comprehensive description field that maps the device IDs to their functions within specific scenarios. Once the functions of each MCP server are properly configured, workflow execution can proceed accordingly.

We utilize natural language commands via the conversational interface to instruct *Claude Sonnet 4* to commence an environmental adjustment workflow. Empowered by the MCP protocol, *Claude Sonnet 4* interprets the semantic meaning of the user input and launches a workflow (as shown in Fig. 2f). It first queries the relevant workflow from the MCP server by invoking *get_policy("env_adjustment")*, and then parses the semantic content of the workflow. After understanding the prescribed sequence, it begins execution. For example, *Claude Desktop* may invoke the *get_device_state("5a5a5b0db496c231")* method on the DTB MCP server to retrieve real-time sensor data, which it then semantically analyzes. If the temperature is found to be above the human comfort range, it proceeds to call *control_device("20a52f2f4e792abb", "temperature_down")* to adjust the environmental controls on device *B* (e.g., activate cooling functions). The entire process, including the reasoning and tool invocation chain, is transparently presented to the user via the interactive interface (see Fig. 2c), enabling a clear understanding of the workflow and its decision logic.

Fig. 2g illustrates the time consumption for the entire workflow in our demonstration. In particular, item *T0* represents the total execution time for the basic scenario. As shown, the vast majority of the time is consumed by the MCP Host, which accounts for 96.84% of the total workflow duration (with the overall process taking 17.20 seconds, of which the MCP Host is responsible for 16.65 seconds).

To further illustrate the modularity and ease of integration enabled by MCP, we extend this demonstration to additional use cases. Given the similarity of the principles and methods, we omit further interface screenshots for brevity and instead summarize key advantages as follows.

D. Scenario for Modular Device/Service Integration

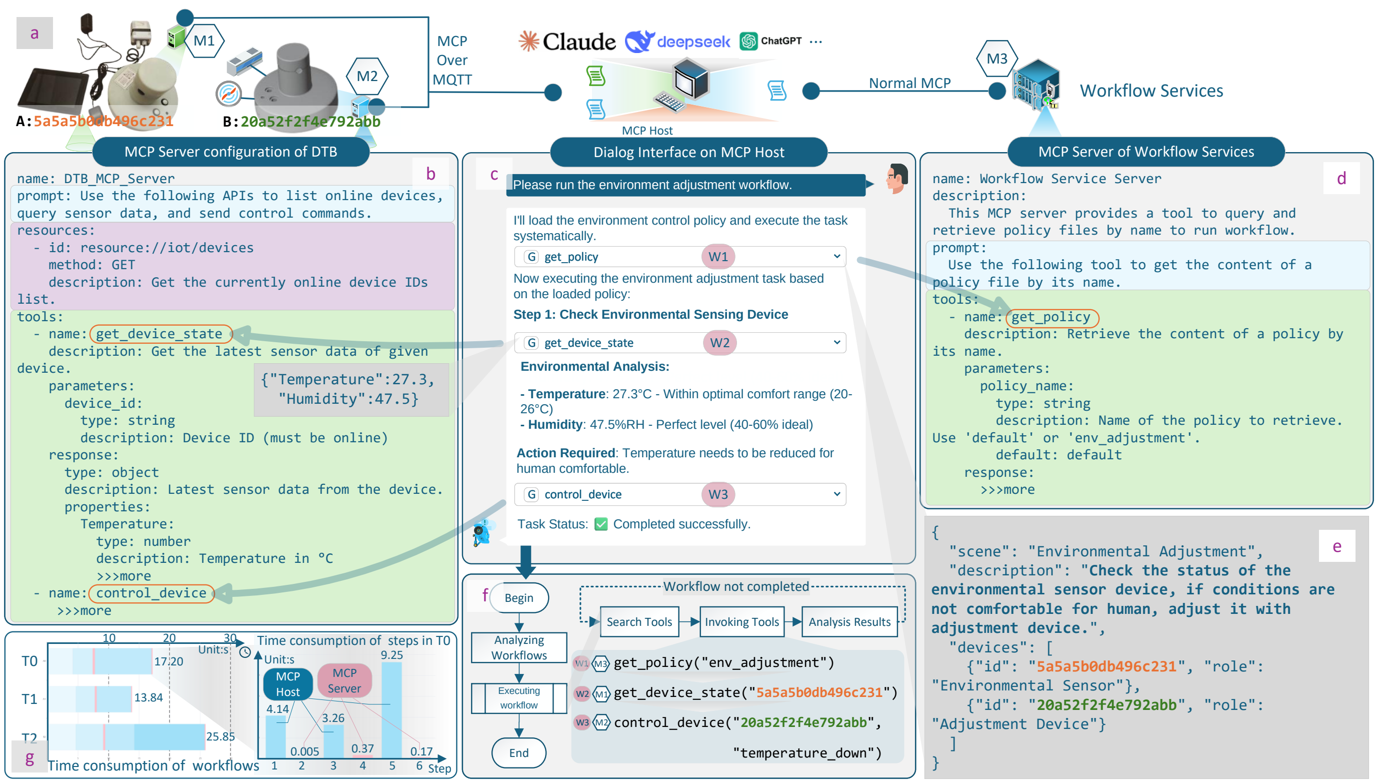
Building on the baseline scenario, the integration of new devices or features becomes straightforward by connecting the relevant device to the framework via an MCP server to enable management and control immediately.

For devices not previously supported or functionally equivalent devices from different manufacturers, the deployment of a compatible MCP server will be standard practice in the future; device vendors may offer ready-to-use MCP server modules, which can be downloaded, configured, and run to enable seamless device onboarding. This approach also facilitates effortless device substitution, as similar devices from different vendors can be seamlessly replaced through their respective MCP server adaptations. This eliminates the need for developing APIs or adapters for every new device, greatly streamlining integration.

Alternatively, for scenarios requiring more deterministic control, suppose we do not wish to rely on LLM-generated policies but instead introduce a dedicated MCP server for handling complex strategies. We can design a tool such as *env_adjustment(temperature, humidity)* within the strategy server, and modify the workflow description to reference this tool for environmental adjustment decisions. During workflow execution, the LLM calls this tool and bases its actions on the returned results. We have implemented and tested a rule-based policy engine using this approach. The time consumption of this case is shown in the item *T2* of Fig. 2g.

E. Scenario for Decoupling of Devices and Services

In our framework, workflows and strategies are treated as modular services, separate from the underlying hardware. To demonstrate workflow substitution, consider a scenario where devices A and B are located in an office and a meeting room, respectively, each with independent environmental regulators. To adjust the office environment, we simply modify the semantic policy in the strategy server. For example, updating



```
name: DTB_MCP_Server
prompt: Use the following APIs to list online devices,
query sensor data, and send control commands.
resources:
  - id: resource://iot/devices
    method: GET
    description: Get the currently online device IDs
list.
tools:
  - name: get_device_state
    description: Get the latest sensor data of given
device.
    parameters:
      device_id:
        type: string
        description: Device ID (must be online)
    response:
      type: object
      description: Latest sensor data from the device.
      properties:
        Temperature:
          type: number
          description: Temperature in °C
          >>>more
  - name: control_device
   >>>more
```

```
{"Temperature":27.3,
 "Humidity":47.5}
```

Please run the environment adjustment workflow.

I'll load the environment control policy and execute the task systematically.

G get_policy W1

Now executing the environment adjustment task based on the loaded policy:

Step 1: Check Environmental Sensing Device

G get_device_state W2

Environmental Analysis:

- **Temperature**: 27.3°C - Within optimal comfort range (20-26°C)
- **Humidity**: 47.5%RH - Perfect level (40-60% ideal)

Action Required: Temperature needs to be reduced for human comfortable.

G control_device W3

Task Status: ✓ Completed successfully.

```
name: Workflow Service Server
description:
  This MCP server provides a tool to query and
retrieve policy files by name to run workflow.
prompt:
  Use the following tool to get the content of a
policy file by its name.
tools:
  - name: get_policy
    description: Retrieve the content of a policy by
its name.
    parameters:
      policy_name:
        type: string
        description: Name of the policy to retrieve.
Use 'default' or 'env_adjustment'.
        default: default
    response:
      >>>more
```

```
{
  "scene": "Environmental Adjustment",
  "description": "Check the status of the
environmental sensor device, if conditions are
not comfortable for human, adjust it with
adjustment device.",
  "devices": [
    {"id": "5a5a5b0db496c231", "role":
"Environmental Sensor"},
    {"id": "20a52f2f4e792abb", "role":
"Adjustment Device"}
  ]
}
```

```
W1 M3 get_policy("env_adjustment")
W2 M1 get_device_state("5a5a5b0db496c231")
W3 M2 control_device("20a52f2f4e792abb",
                     "temperature_down")
```

Fig. 2: The context and key components of the use case demonstration: (a) Device/service configuration and MCP integration within the scenario; (b) MCP server configuration for DTB devices; (c) Dialog interface of the environment adjustment workflow, with detailed context shown in other subfigures; (d) Example MCP server configuration for services; (e) Natural language workflow returned to the MCP host; (f) End-to-end workflow execution process; (g) Time consumption statistics for different workflows and steps.

the description to "Check the environment in the office, and operate the device to make it comfortable." Device identifiers (e.g., *5a5a5b0db496c231* for the office, *20a52f2f4e792abb* for the meeting room) are mapped accordingly. This enables workflow invocation through natural language and semantic parsing, allowing the LLM to automatically retrieve and interpret the environment data, and execute corresponding actions based on the result.

Policy updates are equally efficient. For example, the system may shift from targeting *human comfort* (20–26°C) to *food preservation* (0–4°C) simply by modifying the semantic description in the policy. The LLM will interpret the new requirement and adjust its reasoning accordingly, making the transition seamless. The time consumption of this case is illustrated in the item *T1* of Fig. 2g.

While such simple strategies can be achieved by both traditional and MCP-based frameworks, real-world IoRT applications often involve more complex scenarios with domain-specific logic. For these, it suffices to implement new strategy tools in the MCP server and reference them in workflow descriptions, which eliminates the need to adapt existing devices or middleware for new policies. The LLM will automatically convert input data to the required format for the new strategies during invocation. Through these demonstrations, we have illustrated the modular device/service onboarding and the functional decoupling between device capabilities and service logic achieved by our MCP-integrated IoRT framework.

F. Analysis on Time Consumption

In our described use case, we evaluated both the basic scenario and its evolved variants, and recorded the corresponding time consumption statistics, as shown in Fig. 2g (with some details previously discussed). It is important to note that integrating MCP into IoRT introduces a significant trade-off: the reliance on LLM-based semantic analysis can result in considerable processing delays, as the LLM must parse data and invoke subsequent tools. In our use case, given the relative simplicity of both the MCP server tools and workflow, the majority of the total execution time was spent on semantic parsing and tool invocation by the MCP host, accounting for over 95% of the total workflow duration.

Beyond latency measurements, we note that once an MCP server is registered, both new device functions and workflow updates become immediately available. The LLM orchestrator discovers and incorporates these changes in real time, ensuring that system capabilities are dynamically updated without additional reconfiguration overhead.

This observation highlights that when handling IoRT tasks via the MCP protocol, the characteristics of the task itself must be carefully considered. For tasks that are triggered frequently or are time-sensitive, the response latency introduced by LLM-driven workflows may not be acceptable, and MCP alone may not fully meet workflow requirements. Therefore, for such tasks, it remains necessary to utilize middleware or other traditional solutions within the proposed framework to ensure timely execution. In such scenarios, leveraging MCP to generate corresponding contextual interaction standards and enable automatic deployment could also be a viable approach, though this falls beyond the scope of this paper. Nevertheless, for general-purpose or latency-tolerant applications, integrating MCP into existing IoRT frameworks offers significant advantages, and its benefits are expected to play an important role in accelerating the evolution of IoRT systems.

IV. Challenges

The MCP-integrated IoRT framework provides flexibility and modularity, but its integration also introduces critical challenges. As MCP inherently relies on LLM capabilities for orchestration and decision-making, its deployment inherits many of the fundamental challenges associated with LLM integration in IoRT systems. This section discusses the main issues encountered in typical IoRT deployments and highlights obstacles that must be addressed for practical adoption.

A. Real-Time and Safety of IoRT scenarios

In safety-critical IoRT applications such as industrial automation and medical monitoring, systems require deterministic responses within strict time bounds [15]. Control loops may require actions of sub-10 ms, while medical alerts often require reactions within 100 ms. However, LLM inference introduces variable latency and probabilistic behavior, which undermine predictability and reliability in such environments. The unpredictable nature of language model processing creates fundamental challenges for deployment in time-sensitive environment.

The potential mitigation strategies include hybrid architectures that combine deterministic real-time controllers with MCP-enabled high-level coordination, lightweight MCP runtimes optimized for time-critical paths, edge deployment of small language models to reduce latency, and fallback mechanisms to conventional control.

B. Security and Access Control for intent injection

The semantic nature of MCP registration and LLM-driven orchestration introduces attack surfaces beyond traditional IoRT security. Malicious services may inject harmful tools or misleading prompts, while semantic ambiguity can be exploited for intent spoofing, leading to unsafe or unintended actions. LLM-based orchestration further adds risks, including hallucinated tool invocations, inconsistent behavior in edge cases, excessive computational overhead, and dependencies on external services that may create single points of failure.

To mitigate these threats, access control must go beyond traditional role-based models. An intent-level authorization framework can provide finer-grained control by assigning permissions to semantic intents rather than devices alone. Complementary measures include certification and sandboxing of MCP tools, semantic intent validation to reduce spoofing, distributed consensus for safety-critical actions, and runtime monitoring to detect anomalous LLM behavior.

C. Semantic Standardization on MCP servers

As IoRT deployments scale, heterogeneous MCP servers and devices introduce semantic ambiguity that complicates orchestration. Similar functions may carry inconsistent meanings among vendors or scenarios; for example, a *get_temperature* service could refer to greenhouse climate control, server room monitoring, or medical equipment. In contrast, large-scale deployments often involve numerous identical sensors and actuators, creating challenges in precise semantic identification and disambiguation among functionally equivalent devices. This heterogeneity in parameter structures, measurement units, and operational semantics undermines interoperability, complicates workflow automation, and hinders cross-vendor substitution.

To address these challenges, standardized semantic descriptors are essential. A schema-based approach can unify tool descriptions by specifying functionality, parameters, context, and interoperability metadata. Industry-wide efforts should prioritize hierarchical taxonomies of IoRT capabilities, contextual metadata standards that capture spatial and temporal dimensions, cross-vendor compatibility frameworks, and automated schema validation to ensure compliance. These measures will reduce ambiguity, enhance interoperability, and support scalable MCP integration in diverse IoRT environments.

D. Transparency and auditability in MCP host

IoRT applications require workflow transparency and auditability to support system maintenance, debugging, and operational oversight. However, LLM-based orchestration introduces significant opacity challenges. The probabilistic nature of LLM responses creates non-deterministic behavior, where identical inputs may produce different outputs, undermining system predictability and troubleshooting efforts. Current MCP implementations lack explainability for decision-making processes and standardized traceability mechanisms, creating difficulties when investigating system behaviors or performance issues.

Establishing transparency requires comprehensive logging and audit capabilities. Essential components include execution provenance tracking, explainable AI integration for decision rationale, and workflow replay mechanisms for debugging purposes. Implementation should encompass structured logging protocols, audit trail preservation, real-time visualization dashboards, and automated anomaly detection. These mechanisms must maintain minimal performance overhead to avoid impacting overall system efficiency in IoRT deployments.

V. Conclusion

This paper presents a novel architectural approach for the Internet of Robotic Things (IoRT) by introducing the Model Context Protocol (MCP) as a semantic orchestration layer bridging edge devices and intelligent agents. The proposed framework demonstrates how MCP facilitates modular registration of sensing and actuation functionalities, effectively decoupling device and service integration from the constraints of traditional middleware pipelines. By leveraging prompt-driven execution, LLM-based controllers are empowered to dynamically orchestrate device behaviors, thereby reducing development complexity and enhancing adaptability in heterogeneous environments.

Our digital twin node implementation validates the practical advantages of MCP integration, illustrating how IoRT nodes can expose both real-time and historical data as structured resources and provide actuator control through standardized tools. This paradigm paves the way for the next generation of IoRT systems, which are characterized by openness, reconfigurability, and enhanced alignment with human-centric semantic reasoning.

Looking forward, further research and development in lightweight MCP runtimes, real-time protocol extensions, and interoperability standards will be essential to drive the adoption of this architecture in large-scale, mission-critical IoRT deployments.

REFERENCES

[1] O. Aouedi and K. Piamrat, "Toward a scalable and energy-efficient framework for industrial cloud-edge-IoT continuum," *IEEE Internet of Things Magazine*, vol. 7, no. 5, pp. 14–20, 2024-09. [Online]. Available: https://ieeexplore.ieee.org/document/10643987/

[2] F. K. Shaikh, S. Karim, S. Zeadally, and J. Nebhen, "Recent trends in internet-of-things-enabled sensor technologies for smart agriculture," *IEEE Internet of Things Journal*, vol. 9, no. 23, pp. 23 583–23 598, 2022-12. [Online]. Available: https://ieeexplore.ieee.org/document/9903855/

[3] C. Das, A. A. Mumu, M. F. Ali, S. K. Sarker, S. M. Muyeen, S. K. Das, P. Das, M. M. Hasan, Z. Tasneem, M. M. Islam, M. R. Islam, F. R. Badal, M. H. Ahamed, and S. H. Abhi, "Toward IoRT collaborative digital twin technology enabled future surgical sector: Technical innovations, opportunities and challenges," *IEEE Access*, vol. 10, pp. 129 079–129 104, 2022.

[4] A. Milella, S. Rilling, A. Rana, R. Galati, A. Petitti, M. Hoffmann, J. L. Stanly, and G. Reina, "Robot-as-a-service as a new paradigm in precision farming," *IEEE Access*, vol. 12, pp. 47 942–47 949, 2024.

[5] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS: a practical introduction to the Robot Operating System.* " O'Reilly Media, Inc.", 2015.

[6] F. Cirillo, G. Solmaz, E. L. Berz, M. Bauer, B. Cheng, and E. Kovacs, "A standard-based open source iot platform: Fiware," *IEEE Internet of Things Magazine*, vol. 2, no. 3, pp. 12–18, 2020.

[7] J. John, A. Ghosal, T. Margaria, and D. Pesch, "Dsls for model driven development of secure interoperable automation systems with edgex foundry," in *2021 Forum on specification & Design Languages (FDL)*. IEEE, 2021, pp. 1–8.

[8] M. Alam, N. Ahmed, R. Matam, and F. A. Barbhuiya, "iofog: Prediction-based fog computing architecture for offline iot," in *2021 International Wireless Communications and Mobile Computing (IWCMC)*. IEEE, 2021, pp. 1387–1392.

[9] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, "Extend cloud to edge with kubeedge," in *2018 IEEE/ACM Symposium On Edge Computing (SEC)*. IEEE, 2018, pp. 373–377.

[10] B. Wu, J. Li, Q. Liu, and K.-L. Du, "An RFID-assisted smart livestock and poultry farming system on the cloud," in *2023 15th International Conference on Computer and Automation Engineering (ICCAE)*, 2023-03, pp. 120–124. [Online]. Available: https://ieeexplore.ieee.org/document/10111202/

[11] M. Rahman, M. S. R. Kohinoor, and A. A. Sami, "Enhancing poultry farm productivity using IoT-based smart farming automation system," in *2023 26th International Conference on Computer and Information Technology (ICCIT)*, 2023-12, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/10441525/

[12] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, W. Ye, Y. Zhang, Y. Chang, P. S. Yu, Q. Yang, and X. Xie, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 3, pp. 1–45, Jun. 2024.

[13] C. Yang, Q. Guo, H. Yu, Y. Chen, T. Taleb, and K. Tammi, "Semantic-enhanced digital twin for industrial working environments," in *Global Internet of Things and Edge Computing Summit*, M. Presser, A. Skarmeta, S. Krco, and A. González Vidal, Eds. Springer Nature Switzerland, 2025, pp. 3–20.

[14] A. Singh, A. Ehtesham, S. Kumar, and T. T. Khoei, "A survey of the model context protocol (MCP): Standardizing context to enhance large language models (LLMs)," Apr. 2025.

[15] C. Yang, H. Yu, Q. Guo, T. Taleb, J. C. Requena, and K. Tammi, "Deterministic networking empowered robotic teleoperation," *IEEE Network*, vol. 39, no. 4, pp. 280–289, 2025.