IoT-MCP: Bridging LLMs and IoT Systems Through Model Context Protocol

Ningyuan Yang, Guanliang Lyu, Mingchen Ma, Yiyi Lu, Yiming Li, Zhihui Gao, Hancheng Ye, Jianyi Zhang, Tingjun Chen, Yiran Chen
Department of Electrical and Computer Engineering, Duke University

Abstract

The integration of Large Language Models (LLMs) with Internet-of-Things (IoT) systems faces significant challenges in hardware heterogeneity and control complexity. The Model Context Protocol (MCP) emerges as a critical enabler, providing standardized communication between LLMs and physical devices. We propose IoT-MCP, a novel framework that implements MCP through edge-deployed servers to bridge LLMs and IoT ecosystems. To support rigorous evaluation, we introduce *IoT-MCP Bench*, the first benchmark containing 114 Basic Tasks (e.g., "What is the current temperature?") and 1,140 Complex Tasks (e.g., "I feel so hot, do you have any ideas?") for IoT-enabled LLMs. Experimental validation across 22 sensor types and 6 microcontroller units demonstrates IoT-MCP's 100% task success rate to generate tool calls that fully meet expectations and obtain completely accurate results, 205ms average response time, and 74KB peak memory footprint. This work delivers both an open-source integration framework (https://github.com/Duke-CEI-Center/IoT-MCP-Servers) and a standardized evaluation methodology for LLM-IoT systems.

CCS Concepts

• **Hardware** → **Sensors and actuators**.

Keywords

Internet-of-Things (IoT), Model Context Protocol (MCP), Generative AI

ACM Reference Format:
Ningyuan Yang, Guanliang Lyu, Mingchen Ma, Yiyi Lu, Yiming Li, Zhihui Gao, Hancheng Ye, Jianyi Zhang, Tingjun Chen, Yiran Chen. 2025. **IoT-MCP**: Bridging LLMs and IoT Systems Through Model Context Protocol. In *ACM Workshop on Wireless Network Testbeds, Experimental evaluation & Characterization (WiNTECH '25), November 4–8, 2025, Hong Kong, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3737895.3768303

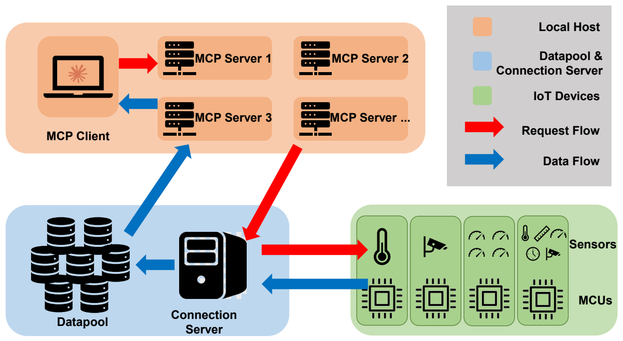


Figure 1: Workflow of the developed IoT-MCP framework, illustrating the request and data flow between the MCP servers and clients on the Local Host (Orange), Datapool and Connection Server (Blue), and IoT Devices (Green).

1 Introduction

The proliferation of Internet-of-Things (IoT) systems has created unprecedented opportunities for physical world digitization, yet managing heterogeneous devices remains challenging [5, 6]. Large Language Models (LLMs) [16, 17] offer transformative potential for IoT systems by enabling natural language interaction with IoT devices, but also require intelligent and robust interfaces to bridge the digital and physical worlds. The recently emerged Model Context Protocol (MCP) [10] addresses this need through its protocol-agnostic design and extensible tool framework, empowering LLMs to perceive, influence, and interact with IoT systems. With industry adoption accelerating [1, 4], rigorous MCP evaluation in the context of different IoT systems becomes critical. However, real-world IoT deployments face significant challenges: extreme hardware heterogeneity complicates scalability; low-latency requirements demand careful data/connection balancing; and performance assessment lacks standardized metrics, unlike structured domains.

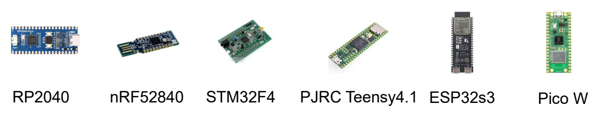


Figure 2: The 6 MCUs passed test in IoT-MCP, each from different MCU family.

To address these challenges, in this paper, we introduce **IoT-MCP** (see Figure 1), a comprehensive framework that enables LLM to interact with IoT systems through standardized MCP interfaces. IoT-MCP adopts a decoupled architecture design and can be divided into three modules. *Local Host* implements a lightweight MCP architecture to ensure stable and efficient interaction between users and LLM. *Datapool and Connection Server* centrally manages data request interactions between all MCP servers and MCUs, establishing efficient and long connections. *IoT Devices* have deployed highly scalable implementations that support processing requests and reading data based on different connected sensors.

This separation ensures both system efficiency and operational robustness, allowing for scalable deployment across a variety of IoT devices with different microcontroller units (MCUs) and sensor peripherals, and application scenarios. Overall, MCP-IoT supports six MCU families and 22 sensors, which are listed in Figure 2 and Table 1 alongside their functionalities.

To provide an objective and comprehensive evaluation of the performance of IoT-MCP, we also develop **IoT-MCP Bench**, a benchmark suite that comprises 1,254 tasks and 3 system performance metrics, such as Tool Execution, Response Time, Memory Usage, and Prompt Robustness. This benchmark systematically assesses various aspects of IoT task execution, including sensor data interpretation, device control accuracy, and system response reliability. Through extensive experiments and using the developed IoT-MCP Bench, IoT-MCP achieves 100% accuracy on Tool Execution Performance, 99% accuracy on Prompt Robustness Assessment, with 205 ms average response time and 74 KB average memory usage, proving strong concurrency and scalability.

Both the design of IoT-MCP design and the IoT-MCP Bench are open-sourced at https://github.com/Duke-CEI-Center/IoT-MCP-Servers.

2 Related Work

Tool calling and MCP. The Chain-of-Thought (CoT) [18] approach demonstrated that structured reasoning enhances the performance of LLMs, laying the foundation for subsequent LLM-based tool invocation. Follow-up works, such as Chameleon [13] and Gorilla [15], further improved model capabilities by integrating external tools via API calls. Authors of [11] subsequently proved that zero-shot prompting with tool documentation is sufficient to elicit accurate tool usage. Against this backdrop, MCP was proposed as a unified protocol for contextual tool documentation transmission.

Table 1: List of the 22 sensors used in IoT-MCP with their functionalities. For sensors with more than one function, one special function, "read-all", is provided.

Sensor	Functions	Sensor	Functions
LTR390	UV & light	HW080	detection
HC-SR501	motion	TTP223	touch
WaterSensor	water	GY302	light
SW420	vibration	HC-SR04	distance
LM35	temp. & light	KY018	light
KY004	button	KY020	tilt
KY037	touch & light	KY023	joystick
KY021	reed	KY038	sound
KY010	light	KY026	flame
KY036	touch	KY035	field & switch
DHT11	temperature & humidity		
MPU6050	temperature & angle & acceleration & gyro.		

MCP-based IoT integration. Recent work has explored the integration of MCP with IoT systems, particularly in the sensor domain. SensorMCP [9] presents a framework for automated sensor tool generation and language-driven sensor operation through MCP interfaces. Their approach enables large language models to automatically generate and operate sensor tools via a tool-language co-development pipeline, demonstrating effectiveness in scenarios such as wildlife monitoring systems. However, SensorMCP exhibits several limitations that constrain its practical applicability. First, the framework shows strong dependency on specific sensor types, limiting its extensibility to diverse IoT environments. Second, the integration of model training procedures within the framework results in prolonged response times, requiring approximately 30 minutes to accommodate new requirements. These limitations highlight the need for more scalable and responsive MCP-based IoT solutions.

MCP performance evaluation. Traditional LLM evaluation frameworks mainly focus on language comprehension abilities, but with the development of tool-enhanced AI, benchmarks specifically designed to evaluate tool usage abilities have become particularly important. Existing evaluation methods attempt to establish multi-dimensional evaluation standards [12, 19], but there is no control over a unified context transfer protocol.

The evaluation of MCP-based systems has been addressed by MCP-RADAR [8], which introduces the first comprehensive benchmark for assessing LLM performance within the MCP framework. The benchmark employs a five-dimensional evaluation approach, providing valuable insights into LLM capabilities with standardized tool integration frameworks. While MCP-RADAR offers important evaluation metrics, it primarily focuses on assessing model capabilities rather than MCP system performance itself. This focus stems from the inherent challenge that different MCP servers execute distinct functions, making direct comparisons difficult. Moreover, system-level metrics such as operational efficiency and accuracy are challenging to evaluate directly within the existing framework.

Limitations of existing approaches. Current MCP-based IoT solutions face several critical limitations. First, existing frameworks lack comprehensive hardware compatibility, often targeting specific sensor types or device categories. Second, the integration of training procedures within operational frameworks introduces significant latency, hindering real-time IoT applications. Third, evaluation methodologies primarily assess model performance rather than system-level effectiveness, leaving gaps in understanding the MCP framework's performance in IoT contexts.

Our work addresses these limitations by proposing IoT-MCP, a decoupled architecture that separates connection management from MCP communication, and IoT Bench, a specialized benchmark that evaluates both model capabilities and system performance in the context of IoT systems.

3 Architecture

3.1 Design Challenges

The proposed IoT-MCP framework aims to facilitate seamless integration between MCP and IoT systems, enabling LLMs to effectively interact with a wide range of IoT devices. However, the design of IoT-MCP presents three fundamental challenges that must be addressed as illustrated below.

Response time on various MCU devices. IoT-MCP acting as protocol adapters between LLMs and physical devices, interface directly with language models, necessitating rapid response times to maintain conversational flow. However, MCUs and sensors may experience temporary disconnections or hardware failures, potentially causing processing pipelines to stall. This mismatch between the high-availability expectations of LLM interactions and the inherent unreliability of distributed IoT devices creates a significant architectural challenge.

Heterogeneous sensory data management. The diversity of IoT sensors generates sensor-collected data in varied formats with distinct storage and processing requirements. These operations may be deployed locally on MCUs or in the cloud, making it difficult to establish unified data handling protocols. This heterogeneity complicates the development of standardized interfaces and consistent data processing workflows.

Computational resource constraints on IoT and edge devices. MCUs typically operate with limited computational resources, including restricted memory, processing power, and energy budgets. These constraints make it challenging to handle concurrent requests or perform complex data processing tasks. The need to support multiple sensor types and communication protocols further strains these limited resources, requiring careful optimization of both hardware utilization and software efficiency.

3.2 IoT-MCP

To address these challenges, we decouple the system into three distinct domains: (*i*) Local Host, (*ii*) Datapool and Connection Server, and (*iii*) IoT Devices, as illustrated in Figure 1.

Local Host. The Local Host hosts the LLMs and multiple specialized MCP servers within a controlled computing environment. It maintains direct communication with the MCP clients while remaining isolated from potential disruptions in the IoT Devices. Each MCP server is designed to control a specific sensor function, allowing for more precise tool selection and execution with improved overall system reliability. Specifically, when the user enters a natural language request and calls the MCP Server, the following JSON command will be generated on the Local Host.

```
{
    "command": [READ_XXX],
    "duration": [DURATION],
    "interval": [INTERVAL]
}
```

where "duration" refers to the time duration of one reading, and "interval" refers to the time interval between two readings. This instruction will be passed to the IoT devices.

Datapool and Connection Server. This domain serves as an intermediary layer, which can be deployed either on the local host or in the cloud, depending on application-specific requirements such as:

- *Deployment scale*: Local deployment suits small-scale networks (e.g., 10 MCUs)
- *Access demand*: Cloud deployment supports massive concurrent data requests
- *Resource constraints*: Offloads processing from resource-limited MCUs

This flexibility addresses the heterogeneous sensory data management challenge by providing a standardized interface for data collection, storage, and analysis. The Datapool and Connection Server includes a *connection server* that manages communication with MCU devices, and a *datapool* for persistent data storage. In Connection Server, each instruction will be assigned a unique ID for unified management. By buffering the requests on the Connection Server, this architecture avoids the temporary MCU disconnections from impacting the responsiveness of the Local Host.

IoT Devices. Each IoT device features a lightweight, extensible microservice architecture specifically designed for computational resource-constrained environments. Each MCU runs a minimal service framework that supports multiple communication protocols that can either be wireless (e.g., Wi-Fi, Bluetooth) or wired (e.g., I2C [14]). After receiving the instruction, the MCU will generate the following JSON information and send it back to the Local Host.

```
{
    "write_time": [REAL_WORLD_TIME],
    "timestamp": [TIMESTAMP],
    "id": [UUID],
    "sensor": [SENSOR_NAME],
    [DATA_TYPE]: [DATA]
}
```

Here, the field [DATA_TYPE] is the returned sensory data, depending on the sensors (e.g., temperature). The microservice design allows for dynamic extension based on connected sensors, enabling efficient resource utilization while maintaining broad hardware compatibility. This approach addresses computational constraints by distributing processing loads and allowing for modular functionality expansion.

3.3 IoT-MCP Bench

3.3.1 Evaluation metrics. The IoT-MCP Bench comprises two core components to validate the IoT-MCP architecture. First, a specialized dataset containing 114 *Basic Tasks* representing fundamental sensor operations, and 1,140 procedurally-generated *Complex Tasks* with linguistic variations. Second, IoT-MCP Bench, a dedicated testing toolkit. It is designed to systematically assess the framework's performance across three key metrics: (*i*) Success Rate, (*ii*) Average Response Time, and (*iii*) Peak Memory Footprint Based on these metrics, the evaluation framework focuses on seven dimensions directly aligned with experimental validation goals. For example, tool execution reliability measures the task success rates and data accuracy across all supported sensors. Response time analysis measures end-to-end latency while also isolating the time distribution across system components, including dedicated measurement of network connection overhead through idle response tests. Memory utilization assessment tracks peak consumption during operations and idle states to characterize resource allocation patterns.

Cross-model compatibility testing evaluates performance variance across different language model backends using the tool execution success rate as the primary metric. Concurrency performance analysis measures response times and memory usage under increasing simultaneous request loads. Prompt robustness assessment employs the Complex Tasks dataset to quantify success rates with linguistically challenging inputs. Finally, deployment stability validation conducts extended real-world operation monitoring with automatic reconnection testing during simulated network interruptions.

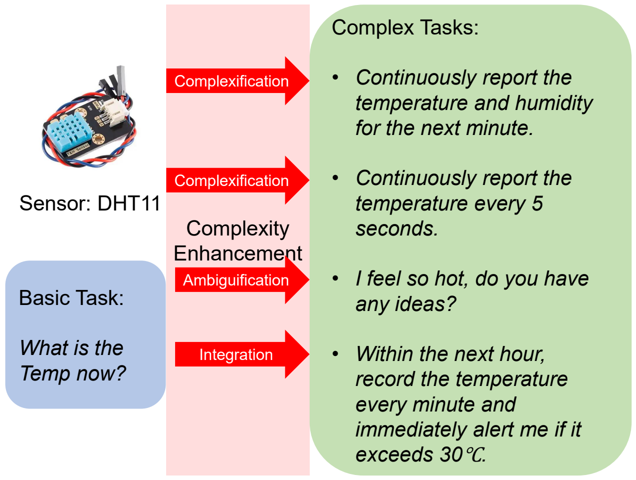

Sensor: DHT11
Basic Task:
What is the Temp now?
Complexification
Complexification
Complexity Enhancement
Ambiguification
Integration
Complex Tasks:
Continuously report the temperature and humidity for the next minute.
Continuously report the temperature every 5 seconds.
I feel so hot, do you have any ideas?
Within the next hour, record the temperature every minute and immediately alert me if it exceeds 30℃.

Figure 3: Example of complexity enhancement: Using the DHT11 sensor as an example, a simple temperature reading task is progressively transformed into a sequence of increasingly complex tasks.

3.3.2 Dataset Generation. The IoT Bench dataset is constructed through a two-stage hybrid generation approach that combines human expertise with automated complexity enhancement to create a comprehensive and challenging evaluation dataset.

Stage 1: Basic Task Generation. We begin by manually crafting 114 basic tasks based on the capabilities of 22 commonly used sensors supported by the IoT-MCP framework.

These sensors support a variety of functionalities, including environmental monitoring (e.g., the DHT11 digital temperature and humidity sensor), motion detection (e.g., the MPU6050 6-axis accelerometer gyroscope sensor), imaging (e.g., camera), and specialized sensors (e.g., pH meter and gas monitor). Each fundamental task is designed to test specific sensor functionalities that will be invoked in different real-world IoT application scenarios. The basic tasks are carefully designed to cover the full spectrum of sensor operations, including sensor initialization, configuration, and data reading.

Stage 2: Complexity Enhancement. To create a more challenging and realistic evaluation environment, we employ LLMs to systematically enhance and expand the 114 basic tasks through three transformation strategies. This

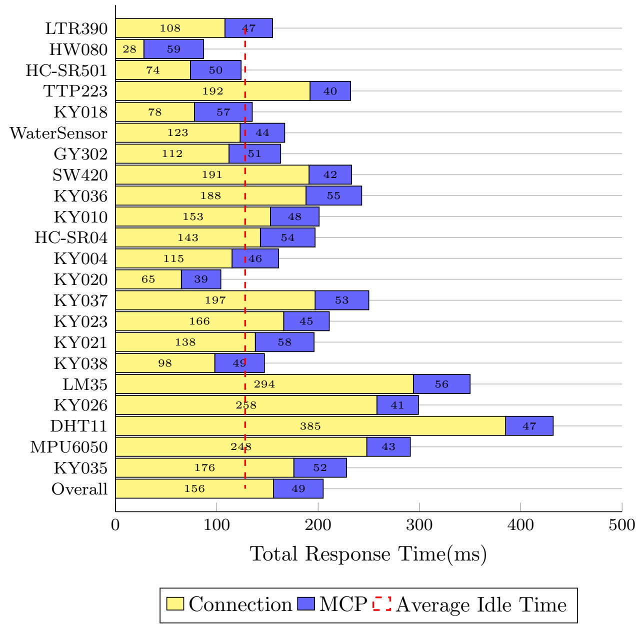


Figure 4: Measured average response time for the MCP servers. The yellow part represents the parsing and transmission time in the Connection Server and MCU, and the blue part represents the time consumption of the MCP Server. The red dashed line represents the average idle time.

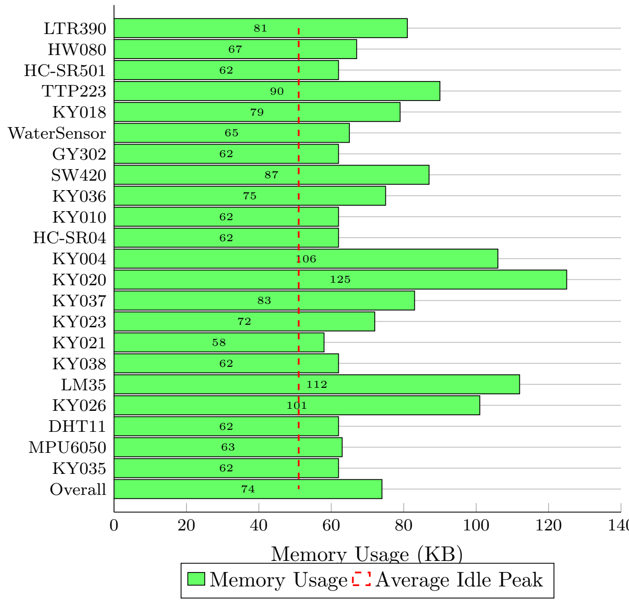


Figure 5: Peak memory footprint on MCU side. The red dashed line represents the average idle memory usage.

process generates 1,140 complex tasks (10 variants per basic task), resulting in a comprehensive dataset that spans simple single-sensor operations to complex multi-device orchestration scenarios. The dataset includes tasks with varying levels of ambiguity, different terminology choices, and diverse complexity levels, ensuring robust evaluation across different usage patterns and user expertise levels.

Specifically, *Complexification* involves combining multiple sensor operations into composite tasks that require sequential or parallel execution across different devices. *Ambiguification* introduces natural language variations and implicit requirements that test the system's ability to interpret user intent from less structured inputs. *Integration* creates scenarios that require coordination between multiple sensors and data fusion operations, reflecting real-world IoT application complexity. An example task is provided in Figure 3.

4 Experiments and Results

4.1 Experimental Setup

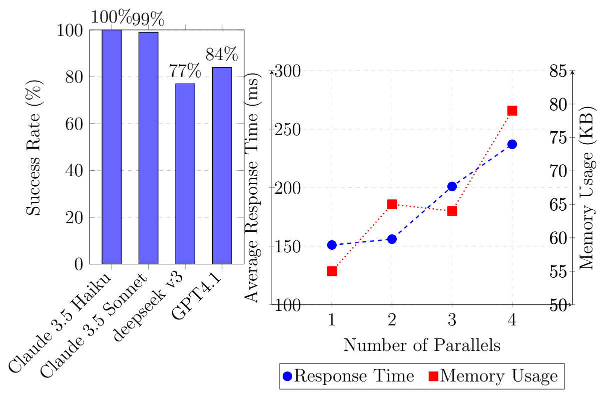
Unless otherwise specified, the following experiments are conducted using the Espressif ESP32-S3 microcontrollers [7] as the primary MCU platform, Claude 3.5 Haiku [2] as the base LLM, WiFi as the basic connection channel, and the Basic Task dataset described in Section 3.3.2 All experiments are performed under controlled conditions to ensure reproducibility and statistical significance. All MCP Servers' designs we have released and their respective supported features are shown in Table 1.

4.2 Metrics Evaluation

Tool execution performance. To evaluate the reliability of the developed IoT-MCP framework, we conduct comprehensive testing of the tool execution success rates and data accuracy across all supported sensors. Specifically, each Basic Task is independently executed 10 times to establish statistical confidence, with results aggregated across different sensor categories and operation types. The results show that all tasks can be successfully executed.

Response time analysis. System responsiveness is critical for maintaining interactive user experiences in IoT applications. We measure end-to-end response times and analyze time distribution across system components through repeated 10 times execution of the Basic Task dataset.

As illustrated in Figure 4, the system achieves an average response time of 205 milliseconds. The analysis reveals that sensors utilizing I2C bus communication, particularly the MPU6050 accelerometer/gyroscope module, exhibit notably longer response times. Furthermore, our component-level timing analysis identifies the Connection Server as the primary performance bottleneck, accounting for approximately 30–75% of the total response time. This finding suggests that



(a) Model Comparison (b) Response Time and Memory Usage

Figure 6: (*Left*) Success rate of calling the MCP Server for different models; (*Right*) Response time under multiple concurrent tasks.

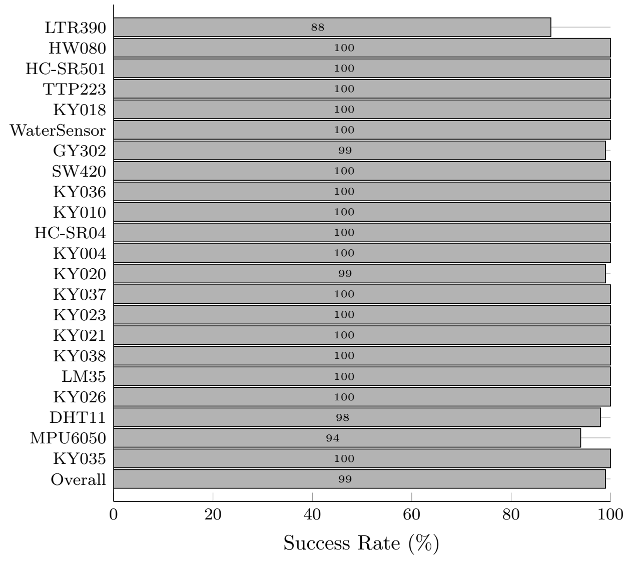


Figure 7: Success rate of the developed MCP Servers with complex tasks.

network communication and protocol translation represent the most significant latency sources in the current implementation. Concurrently, in order to differentiate between time overhead attributed to network connection and core time overhead associated with sensor data reading, the idle response time test was repeated ten times, yielding an average value of 128 ms.

Memory utilization characteristics. We monitor peak memory consumption across all system components during 10 times dataset execution, capturing both individual sensor requirements and system-wide memory usage patterns. Figure 5 demonstrates that the system maintains an average peak memory consumption of 74 KB across all tested scenarios. The results indicate relatively balanced memory allocation across different sensor implementations, with no single sensor type dominating system memory requirements. In a similar manner to the methodology employed for the idle time recording test, the idle memory usage test was repeated ten times, resulting in the attainment of an average value of 51 KB. This accounts for approximately 40–80% of the average peak usage. This finding indicates that the primary factor contributing to memory consumption in the MCU is the establishment of stable TCP connections. This observation suggests that the probability of our implementation encountering memory bottlenecks under high concurrency is low.

4.3 System Robustness Evaluation

Model robustness. To assess the generalizability of our framework across different language models, we evaluate system performance using alternative base models, including Claude 3.5 Sonnet [2], DeepSeek V3 [20], and GPT-4.1 [3]. This evaluation focuses on tool execution success rates as the primary metric for cross-model compatibility.

Results presented in Figure 6(*Left*) demonstrate that our IoT-MCP Server implementations exhibit optimal compatibility with Claude models, achieving the highest success rates (99–100%) in this configuration. Performance with alternative models shows modest degradation, with success rates decreasing by approximately 77% for DeepSeek V3 and 84% for GPT-4.1. This variation primarily stems from differences in tool calling conventions and parameter interpretation strategies across different model architectures.

Concurrency performance analysis. Real-world IoT deployments often require handling multiple simultaneous requests. To evaluate system behavior under concurrent load, we select the KY010 and KY036 sensors based on the experimental results presented earlier in this section as representative test cases and measure average response times and memory usage under varying concurrency levels.

Figure 6(*Right*) illustrates the system's response to increasing concurrent request loads. The results demonstrate graceful performance degradation with increasing concurrency, maintaining acceptable response times (150–250 ms) even under high-load conditions. It was also observed that the growth brought about by high concurrency resulted in smoother memory usage (55–79 KB). The system exhibits linear scaling characteristics up to four concurrent requests.

Prompt robustness assessment. In order to evaluate system performance under actual usage conditions and respond to diverse and potentially ambiguous user inputs, a complex task dataset is generated through the fuzzification and

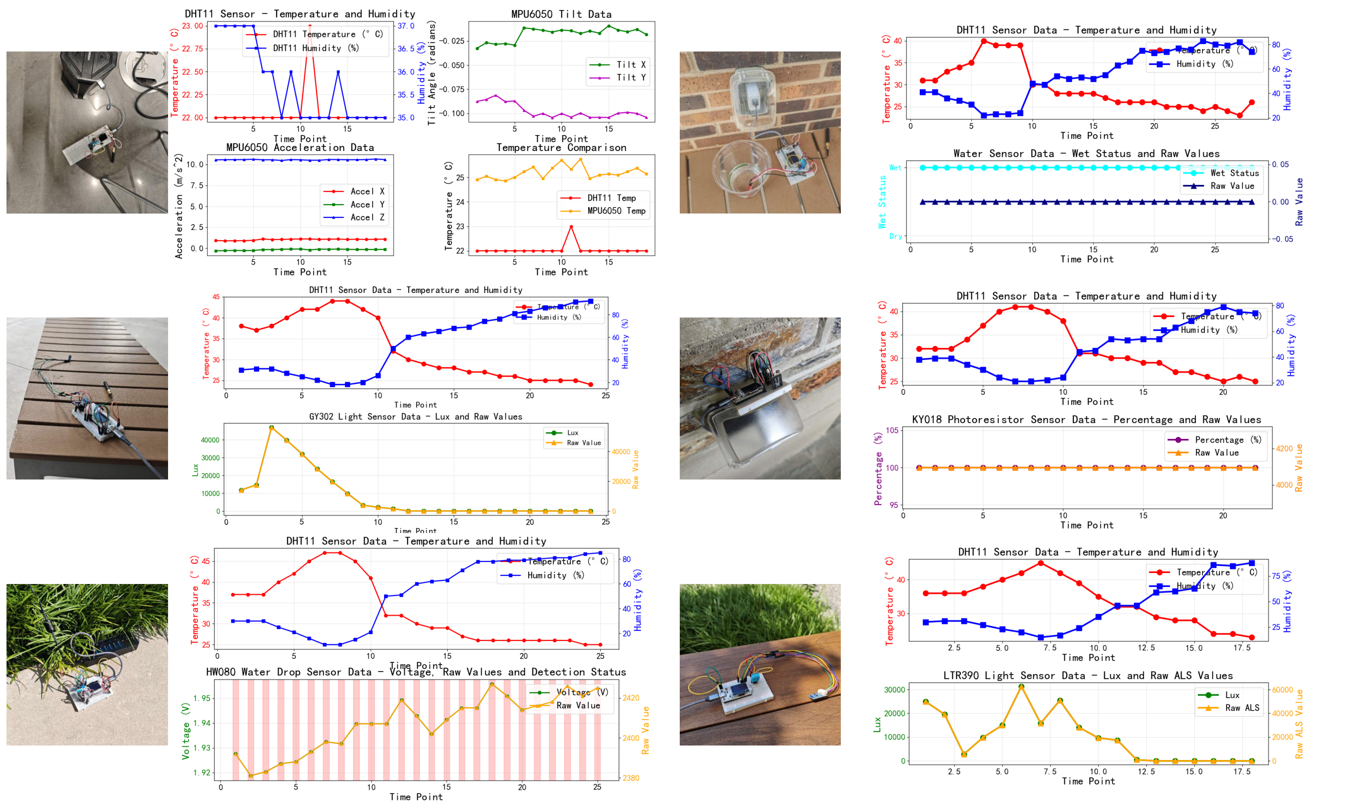


Figure 8: Data recorded in read-world deployment tests. 13 sensors (7 types) are connected to 6 MCUs and uniformly connected to IoT-MCP for a 12-hour test.

complexification process. This complex task dataset is then subjected to repeated testing on each MCP Server until 100 tasks are completed. This assessment tests the framework's ability to interpret and execute commands despite linguistic variations.

As shown in Figure 7, the system maintains high performance even with challenging, ambiguous inputs, achieving an overall success rate of 99%. It was observed that the servers that exhibited substandard performance (e.g., LTR390 and MPU6050) were precisely those that supported multiple data reads, while the tool instructions that generated errors were frequently designated as 'read-all', a practice that should have been avoided. This robust performance demonstrates the effectiveness of our natural language processing approach and validates the framework's readiness for deployment in diverse real-world scenarios where user inputs may not follow strict formatting conventions.

4.4 Real-World Deployment Validation

To validate the practical applicability and reliability of our framework, we conduct a comprehensive 12-hour deployment test within a multi-story building environment. The deployment consists of 6 ESP32-S3 microcontrollers equipped with 7 different types, 12 sensors, all connected to a WiFi network to simulate realistic IoT infrastructure conditions.

Figure 8 presents continuous monitoring results over the 12-hour period. As expected, all sensors returned continuous results. Based on the Connection Server's design, the IoT-MCP can maintain stable connections and automatically restore them after unexpected disconnection, such as during a power outage or network abnormality. This real-world validation confirms the framework's readiness for production deployment and validates the architectural decisions underlying the IoT-MCP design.

5 Discussions

5.1 Hardware Scope Limitations

The current framework's exclusive focus on sensor-based devices constrains applications to monitoring scenarios, omitting actuator integration. However, the decoupled architecture inherently supports extending to control mechanisms. Such expansion would enable closed-loop environmental control systems, transforming passive monitoring platforms into active intervention systems capable of executing complex sensor-actuator sequences for user feedback.

5.2 Functional Complexity Limitations

While supporting basic device invocation, the existing architecture lacks capabilities for dynamic workflow composition, a limitation rooted in positioning LLM+MCP clients as mere *callers* rather than *designers*. Future work will reposition clients as system designers through four interconnected capabilities: composition engines for generating execution plans; workflow management systems with robust failure handling; performance-based optimization mechanisms; safety protocols ensuring graceful degradation during operation.

6 Conclusions

We presented IoT-MCP, a novel framework bridging MCP and IoT systems to address scalable control challenges. Specifically, the design of IoT-MCP is decoupled into three domains: (*i*) Local Host, (*ii*) Datapool and Connection Server, and (*iii*) IoT Devices. We also introduced a comprehensive benchmark dataset comprising 1,254 tasks that span 22 sensors and 6 MCUs from different families, systematically evaluating IoT-MCP across (*i*) Success Rate, (*ii*) Average Response Time, and (*iii*) Peak Memory Footprint. In summary, IoT-MCP represents a significant advancement toward real-world LLM-IoT integration in the era of LLMs.

Acknowledgments

The work was supported in part by NSF grants CNS-2112562 and CNS-2330333.

References

[1] [n. d.]. *Model Context Protocol (MCP) - OpenAI Agents SDK.* https://openai.github.io/openai-agents-python/mcp/

[2] 2024. *Claude 3.5 Sonnet.* https://www.anthropic.com/news/claude-3-5-sonnet Accessed: 2025-08-15.

[3] 2025. *Introducing GPT-4.1 in the API.* https://openai.com/index/gpt-4-1/ Accessed: 2025-08-15.

[4] 2025. *Introducing the Model Context Protocol.* https://www.anthropic.com/news/model-context-protocol

[5] Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The Internet of Things: A survey. *Computer networks* 54, 15 (2010), 2787–2805.

[6] Tingjun Chen, Javad Ghaderi, Dan Rubenstein, and Gil Zussman. 2018. Maximizing Broadcast Throughput Under Ultra-Low-Power Constraints. *IEEE/ACM Transactions on Networking* 26, 2 (2018).

[7] Espressif Systems. 2025. *ESP32 Series.* https://www.espressif.com/en/products/socs/esp32 Accessed: 2025-08-15.

[8] Xuanqi Gao, Siyi Xie, Juan Zhai, Shqing Ma, and Chao Shen. 2025. MCP-RADAR: A Multi-Dimensional Benchmark for Evaluating Tool Use Capabilities in Large Language Models. arXiv:2505.16700 [cs.AI] https://arxiv.org/abs/2505.16700

[9] Yunqi Guo, Guanyu Zhu, Kaiwei Liu, and Guoliang Xing. 2025. A Model Context Protocol Server for Custom Sensor Tool Creation. In *3rd International Workshop on Networked AI Systems (NetAISys '25)*. ACM, Anaheim, CA, USA. doi:10.1145/3711875.3736687

[10] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2025. Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions. arXiv:2503.23278 [cs.CR] https://arxiv.org/abs/2503.23278

[11] Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. 2023. Tool Documentation Enables Zero-Shot Tool-Usage with Large Language Models. arXiv:2308.00675 [cs.CL] https://arxiv.org/abs/2308.00675

[12] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs. arXiv:2304.08244 [cs.CL] https://arxiv.org/abs/2304.08244

[13] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-Play Compositional Reasoning with Large Language Models. arXiv:2304.09842 [cs.CL] https://arxiv.org/abs/2304.09842

[14] NXP Semiconductors. 2014. *I2C-bus specification and user manual* (rev. 6.0 ed.). NXP Semiconductors. https://www.nxp.com/docs/en/user-guide/UM10204.pdf Accessed: 2025-08-16.

[15] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large Language Model Connected with Massive APIs. arXiv:2305.15334 [cs.CL] https://arxiv.org/abs/2305.15334

[16] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[18] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903 [cs.CL] https://arxiv.org/abs/2201.11903

[19] Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. 2023. On the Tool Manipulation Capability of Open-source Large Language Models. arXiv:2305.16504 [cs.CL] https://arxiv.org/abs/2305.16504

[20] Chars Yang. 2025. *DeepSeek v3 - Advanced AI & LLM Model Online.* DeepSeek v3. https://deepseekv3.org/ Accessed: 2025-08-15.