

# Model Context Protocol for Internet of Things Sensors: Control IoT Systems via Large Language Model

Tingjun Chen

Nortel Networks Assistant Professor

ECE Department, Duke University

Yiran Chen

John Cocke Distinguished Professor

Fellow of AAAS/ACM/IEEE/NAI, Member of EASA

ECE Department, Duke University

August 15, 2025

In this course, students will be introduced to two foundational concepts shaping the future of intelligent systems: the Model Context Protocol (MCP), proposed by Anthropic in November 2024, and the Internet of Things (IoT)—the vast ecosystem of interconnected devices that collect, share, and process data in real time, driving innovation across industries from smart homes to industrial automation.

Although there is strong interest in enabling large language models (LLMs) to control IoT systems, this goal has proven difficult to achieve in practice. One major barrier is the diversity and fragmentation of IoT device interfaces—different manufacturers use different communication protocols, data formats, and security mechanisms, making direct integration with LLMs complex and unreliable. Additionally, security and privacy concerns arise when granting LLMs access to real-world devices, as improper controls could lead to unauthorized actions or data leaks.

In order to enable LLMs to control IoT systems, the core problem is that there is no unified and secure protocol that allows LLMs to reliably access and control a wide range of heterogeneous IoT devices. Without standardized interfaces and robust security frameworks, seamless and safe integration between LLMs and IoT systems is difficult to achieve. To address these challenges, the Model Context Protocol (MCP) provides a standardized and secure framework for connecting large language models (LLMs) with external tools and hardware. By defining clear interfaces and robust security mechanisms, MCP makes it possible for LLMs to reliably access and control a wide range of heterogeneous devices. This protocol overcomes the fragmentation of device interfaces and helps ensure safe, flexible, and scalable integration between AI models and the physical world.

By exploring both MCP and IoT, students will gain a holistic understanding of how modern AI systems can be integrated with physical devices, unlocking powerful new applications in automation, data processing, and intelligent control.

This course will provide a high-level overview of both concepts, discuss their key principles, and feature a small demo of controlling a sensor reading system via Claude (an advanced conversational AI model by Anthropic) using an existing MCP Server. Students will experience the excitement of using large language models to interact with real-world IoT devices, even without building the full system themselves. Through this demonstration, students will gain a hands-on understanding of how AI can interact with IoT systems and explore the new possibilities enabled by advanced AI protocols.

# Contents

<b>Lecture 1: Internet of Things Sensors</b>	<b>1</b>
1.1 What is the Internet of Things ? . . . . .	1
1.2 What is an IoT Sensor? . . . . .	1
1.3 How to build a LLM based IoT sensor reading system? . . . . .	2
<b>Lecture 2: Model Context Protocol</b>	<b>3</b>
2.1 What is Model Context Protocol? . . . . .	3
2.2 How MCP offers a proper protocol for controlling IoT systems? . . . . .	4
2.2.1 MCP offers a Unified Approach . . . . .	4
2.2.2 MCP ensures Security & Extensibility . . . . .	4
2.2.3 LLM-Based IoT Sensor Reading System via MCP . . . . .	4
2.3 Core Feature: MCP Server . . . . .	5
2.3.1 What are MCP tools and how do they work? . . . . .	5
2.3.2 Screenshot Demo: A Sample MCP Server . . . . .	5
<b>Lecture 3: MCP Servers for IoT Sensors</b>	<b>7</b>
3.1 How to design an MCP Server for a specific IoT Sensor? . . . . .	7
3.2 DHT11 MCP Server: A Sample Implementation . . . . .	7



# Lecture 1: Internet of Things Sensors

## 1.1 What is the Internet of Things ?

### Definition 1.1

**The Internet of Things (IoT)** refers to a network of physical devices—such as sensors, appliances, vehicles, and other objects—that are connected to the internet. These devices can collect, share, and exchange data, and they can be monitored or controlled remotely.

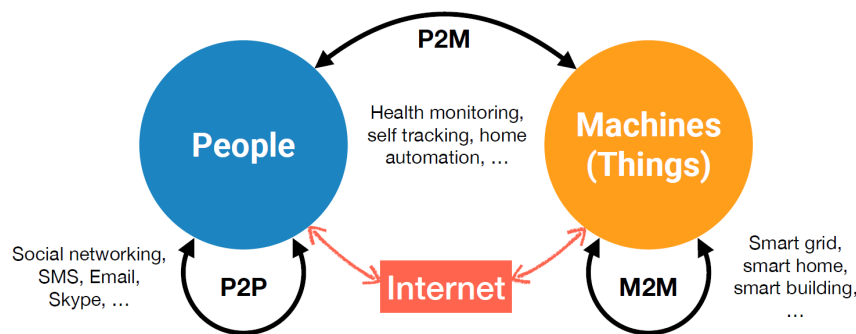


Figure 1: IoT Systems

Figure 1 illustrates the core communication paradigms in the Internet of Things (IoT) ecosystem. These paradigms encompass the interactions between people and machines, all mediated by the Internet. The figure distinguishes three principal modes: (1) Person-to-Person (P2P) communication, as seen in social networking and messaging platforms; (2) Person-to-Machine (P2M) interaction, which appears in applications such as health monitoring and home automation; and (3) Machine-to-Machine (M2M) communication, which supports smart infrastructures like power grids and intelligent buildings. In all cases, the Internet serves as the essential medium, enabling reliable and efficient connectivity across distributed users and devices.

## 1.2 What is an IoT Sensor?

The rapid growth of the Internet of Things (IoT) has made sensors an integral part of our daily lives and industrial environments. Consider the following scenarios:

- In a smart home, temperature sensors adjust the heating system automatically for optimal comfort.
- In precision agriculture, soil moisture sensors provide real-time feedback to optimize irrigation and conserve water.
- In healthcare, wearable devices use heart rate sensors to continuously monitor patients and alert medical staff in case of anomalies.
- In industrial automation, vibration and pressure sensors help monitor machinery health and predict maintenance needs.

These examples highlight how IoT sensors collect critical data from the physical world, making intelligent, data-driven decisions possible across various domains.

### Definition 1.2

An **IoT sensor** is a device that detects physical or environmental changes and transmits the data to a connected system via the internet.

**To transmit data**, a sensor requires an edge device, such as a microcontroller unit (MCU), to read its data and a communication protocol to transmit data from the MCU to the internet or another system.

In simpler terms: the sensor collects raw data, the edge device processes and formats the data, and the communication protocol ensures the data is transmitted reliably to its destination (e.g., the cloud or a centralized system).

## 1.3 How to build a LLM based IoT sensor reading system?

In this course, students will explore how users can communicate with IoT sensors indirectly, through issuing natural language instructions to a large language model (LLM). Recall from 1.1, such a structured framework is considered as Person-to-Machine (P2M) interaction. The LLM acts as an intelligent intermediary: it interprets the user's requests, translates them into actionable commands, and interacts with sensor systems to perform data collection or environmental sensing. Through hands-on examples and practical demonstrations, students will develop an intuitive understanding of how advanced AI can simplify and enhance human-IoT interaction.

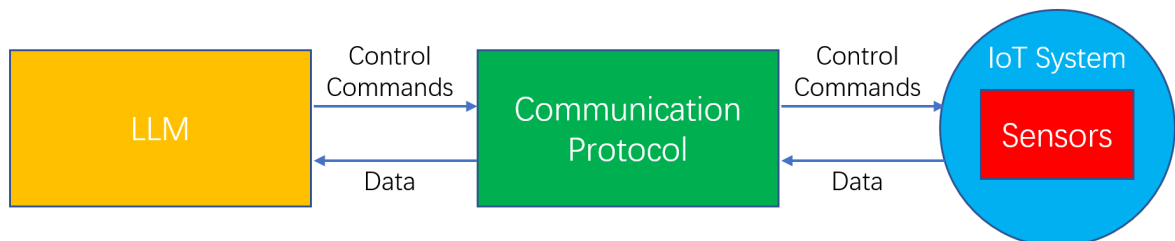


Figure 2: IoT Sensor Reading Agent

This layered architecture depicted above forms the foundation for integrating AI into IoT systems. To enable large language models (LLMs) to control or interact with IoT sensors, it is essential to establish an effective interface between LLMs and the communication protocol that bridges to IoT systems.

In the next lecture, we will introduce the Model Context Protocol (MCP), which provides an effective way to bridge large language models (LLMs) with IoT communication protocols.

## Lecture 2: Model Context Protocol

### 2.1 What is Model Context Protocol?

Although large language models (LLMs), such as GPT and Claude, have demonstrated outstanding abilities across many domains, natural language alone is still insufficient for managing complex, multi-step workflows—especially when integrating external tools, data, or devices. Existing APIs and plugins often face integration challenges and lack a unified context. These issues limit the practical application of AI in real-world scenarios.

To address these limitations, Anthropic introduced the Model Context Protocol (MCP) in November 2024. MCP offers a standardized and structured protocol for LLMs to interact with tools, resources, devices, and humans. Through declarative registration and coordinated workflows, MCP enables robust task automation and seamless AI integration into business and industrial processes.

#### Definition 2.3

**Model Context Protocol (MCP)** is a standard that enables large language models to interact with external tools, resources, and devices in a structured and coordinated manner .

The key participants in the MCP architecture are: <sup>1</sup>:

- **MCP Host**: The AI application that coordinates and manages one or multiple MCP clients
- **MCP Client**: A component that maintains a connection to an MCP server and obtains context from an MCP server for the MCP host to use
- **MCP Server**: A program that provides context to MCP clients

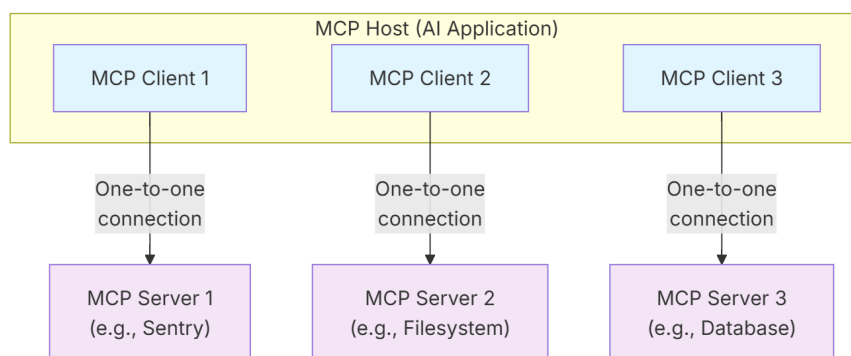


Figure 3: Key Participants in MCP Architecture

Specifically for this course:

- The **MCP Host** is the Claude App (the AI application itself).
- The **MCP Client** corresponds to each individual session within the Claude App (each new chat or task creates a client context).

<sup>1</sup>Source: <https://modelcontextprotocol.io/docs/learn/architecture>

- The **MCP Server** is the component you need to build for IoT sensors. This server is a standalone program or service that registers the relevant tools, resources, and prompts for the AI to access through the MCP protocol.

## 2.2 How MCP offers a proper protocol for controlling IoT systems?

Controlling IoT systems is challenging due to the diversity of devices, protocols, and data formats. Traditional solutions often require custom integrations for each device type, making automation complex and fragile.

### 2.2.1 MCP offers a Unified Approach

MCP addresses these challenges by providing a standardized protocol that allows large language models to interact with IoT devices through registered tools. Each device or sensor can be exposed as an MCP tool, with well-defined interfaces and metadata.

### 2.2.2 MCP ensures Security & Extensibility

MCP also handles permission management and access control, ensuring that only authorized models or users can control sensitive devices. Its extensible design means new devices can be added to the ecosystem without changing the overall integration logic.

### 2.2.3 LLM-Based IoT Sensor Reading System via MCP

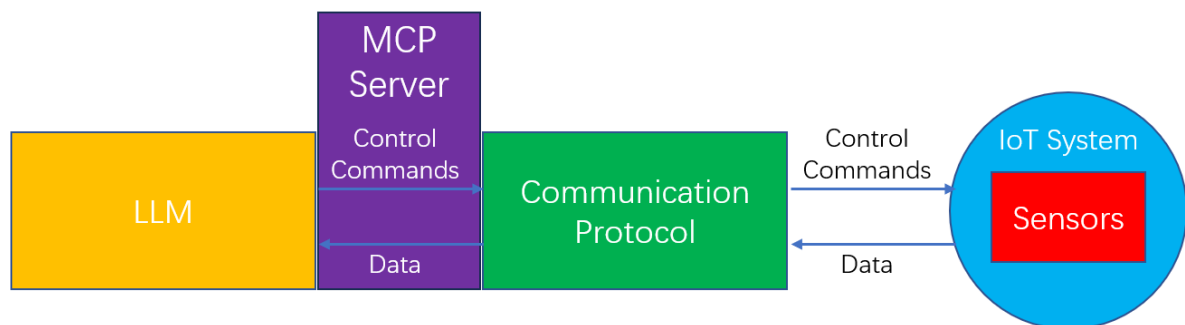


Figure 4: LLM-Based IoT Sensor Reading System via MCP

As discussed in Lecture 1, the goal is to bridge large language models (LLMs) with IoT communication protocols. The MCP Server provides the solution by exposing IoT sensor capabilities as callable tools for the LLMs, bridging LLMs to IoT Communication Protocol.

For this course, we will focus on how to implement MCP server which can bridge LLMs and IoT Communication Protocol. To achieve this goal, it is sufficient to implement an MCP Server for IoT sensors, which only needs to include the necessary MCP Tools.

In summary, MCP addresses the key limitations of natural language-based interaction by providing a unified, structured protocol for large language models to coordinate with



external systems. In the context of this course, MCP serves as the essential bridge between LLMs and IoT sensor systems, making it possible for LLMs to control IoT systems.

Next, we will introduce MCP Server.

## 2.3 Core Feature: MCP Server

As discussed in the previous section, what is needed to be built is a MCP server for IoT sensors. Such MCP Server is only expected to include MCP tools. First, let's first understand what MCP tools are.

### 2.3.1 What are MCP tools and how do they work?

#### Definition 2.4

**MCP tools** (Model Context Protocol tools) are modular, callable services registered on an MCP server. They can be invoked by large language models to perform specific tasks.

MCP tools allow LLMs to systematically interact with external systems, data sources, devices, and APIs. Each tool provides a specific function, such as querying a database, controlling IoT devices, or processing files, and can be combined into complex workflows.

### 2.3.2 Screenshot\_Demo: A Sample MCP Server

**Example.** Let's define a simple MCP Server that could take a screenshot. <sup>a</sup>

Listing 1: FastMCP Screenshot Example

```

1  """
2  FastMCP Screenshot Example
3
4  Give Claude a tool to capture and view screenshots.
5  """
6
7  import io
8
9  from mcp.server.fastmcp import FastMCP
10 from mcp.server.fastmcp.utilities.types import Image
11
12 # Create server
13 mcp = FastMCP(
14     "Screenshot_Demo",
15     dependencies=["pyautogui", "Pillow"]
16 )
17
18 @mcp.tool()
19 def take_screenshot() -> Image:
20     """
21 
```

```

22  Take a screenshot of the user's screen and return it as an image. Use
23  this tool anytime the user wants you to look at something they're doing
24  """
25  import pyautogui
26
27  buffer = io.BytesIO()
28
29  # if the file exceeds ~1MB, it will be rejected by Claude
30  screenshot = pyautogui.screenshot()
31  screenshot.convert("RGB").save(buffer, format="JPEG", quality=60,
32                                optimize=True)
33  return Image(data=buffer.getvalue(), format="jpeg")

```

This MCP Server is named `Screenshot_Demo` and it has a single tool named `take_screenshot`. Each time it is called, it captures a screenshot automatically. Here is a comparison of taking screenshot *with* this MCP Server and *without* this MCP Server.

- With MCP Server:
  - Connect to MCP Server
  - Call `take_screenshot`
- Without MCP Server:
  - write the entire python code in the function `take_screenshot`
  - execute the code

<sup>a</sup>Source: <https://github.com/modelcontextprotocol/python-sdk/blob/main/examples/fastmcp/screenshot.py>

According to the example above, creating a MCP Server which only composed MCP tools is simple:

- Create a MCP Server Instance
- Register desired MCP tools.

In the next lecture, we will learn how to build MCP Server for IoT Sensors.

## Lecture 3: MCP Servers for IoT Sensors

### 3.1 How to design an MCP Server for a specific IoT Sensor?

As discussed in Lecture 2, MCP tools are modular, callable services registered on an MCP Server. For a specific IoT sensor, an MCP Server should support all functions provided by the sensor. Therefore, each MCP tool should correspond to a specific function of the sensor. To design an MCP Server for a specific IoT sensor, you need to:

- Identify all the functions of the specific IoT sensor.
- Encapsulate each distinct function as a separate MCP tool.

As discussed in Lecture 1, what needs to be established is an effective interface between LLMs and the communication protocol that connects to MCUs. Thus, the implementation of an MCP Server should consist of 4 main steps:

- Preliminaries
  - Import all required packages
  - Specify the Communication Protocol
- Create an MCP Server instance.
- Register all tools corresponding to the sensor's functions. Note that each tool should define a structured command that is sent to the communication protocol.
- Design the main entry

### 3.2 DHT11 MCP Server: A Sample Implementation

**Example.** Let's illustrate a sensor-reading MCP Server designed for DHT11, an IoT sensor that can monitor temperature and humidity.

Listing 2: DHT11 MCP Server

```
"""
DHT11 MCP Server Script

This script implements a simple MCP (Model Context Protocol) server for a
DHT11 temperature and humidity sensor,
using the FastMCP framework. It defines three MCP tools for reading
temperature, humidity, or both from the DHT11 sensor,
and communicates with an underlying hardware communication protocol server
via HTTP requests.
The server exposes these sensor functions to be called by external AI
agents through MCP.
"""
```

#### Step 0: Preliminaries

Listing 3: Import all required packages

```

1 from typing import Dict, Any, Optional
2 import requests
3 from mcp.server.fastmcp import FastMCP

```

Listing 4: Specify the Communication Protocol

```

1 # The hostname or IP address of the communication protocol server (usually
   # running locally)
2 COMMUNICATION_PROTOCOL_HOST = 'localhost'
3 # The port number where the protocol server is listening
4 COMMUNICATION_PROTOCOL_PORT = 8000
5 # The full URL endpoint for sending commands to the communication protocol
   # server
6 URL = f"http://{COMMUNICATION_PROTOCOL_HOST}:{COMMUNICATION_PROTOCOL_PORT}/
   send_cmd"

```

## Step 1: create an MCP server instance

Listing 5: MCP Server Initialization

```

1 server = FastMCP(
2     "DHT11",
3     description="""DHT11 PC data monitoring via MCP. Supports
4         general_accuracy temperature and humidity reading."""
5 )

```

## Step 2: register all desired MCP tools

The DHT11 sensor supports two main functions: reading temperature and reading humidity. To provide complete access, we will register three MCP tools:

- `read_temperature`: Retrieves the current temperature from the DHT11 sensor.
- `read_humidity`: Retrieves the current humidity value from the DHT11 sensor.
- `read_all`: Reads both temperature and humidity values together.

### Note:

Always provide a clear and concise description for each MCP tool. This helps the LLM (large language model) understand the purpose and usage of each tool.

Registering these tools ensures that all available sensor functionalities are accessible through the MCP server.

Listing 6: MCP Tools Registration

```

1 @server.tool()
2 async def read_temperature() -> Optional[Dict[Any, Any]]:
3     """
4     Read temperature values from the DHT11 sensor.
5
6     Returns:

```

```

7  A json-format data of temperature with time stamp.
8  """
9
10 cmd = {"sensor": "DHT11", "command": "read_temp"}
11 temperature = requests.post(URL, json=cmd)
12 return temperature
13
14 @server.tool()
15 async def read_humidity() -> Optional[Dict[Any, Any]]:
16     """
17     Read humidity values from the DHT11 sensor.
18
19     Returns:
20     A json-format data of humidity with time stamp.
21     """
22     cmd = {"sensor": "DHT11", "command": "read_hum"}
23     humidity = requests.post(URL, json=cmd)
24     return humidity
25
26 @server.tool()
27 async def read_all() -> Optional[Dict[Any, Any]]:
28     """
29     Read temperature & humidity values from the DHT11 sensor.
30
31     Returns:
32     A json-format data of temperature & humidity with time stamp.
33     """
34     cmd = {"sensor": "DHT11", "command": "read_all"}
35     all_data = requests.post(URL, json=cmd)
36     return all_data

```

### Step 3: design main entry point

This section defines the main entry point for the script. It is common to all MCP servers and ensures that the server runs when the script is executed directly.

Listing 7: Main Entry Point

```

1  if __name__ == '__main__':
2      server.run()

```

## Notes