SensorMCP: A Model Context Protocol Server for Custom Sensor Tool Creation

Yunqi Guo, Guanyu Zhu, Kaiwei Liu, Guoliang Xing
The Chinese University of Hong Kong
yunqiguo@cuhk.edu.hk, 1155226376@link.cuhk.edu.hk, 1155189693@link.cuhk.edu.hk, glxing@ie.cuhk.hk

Abstract

The rising demand for customized sensor systems, such as wildlife and urban monitoring, underscores the need for scalable, AI-driven solutions. The Model Context Protocol (MCP) enables large language models (LLMs) to interface with external tools, yet lacks automated sensor tool generation. We propose SensorMCP, a novel MCP server framework that enables LLMs to dynamically generate and operate sensor tools through a tool-language co-development pipeline. Our contributions include: (1) a SensorMCP architecture for automated tool and language co-evolution, (2) an automated sensor toolbox generating tailored tools, and (3) language assets producing tool descriptions and linguistic modules. A preliminary evaluation using real-world zoo datasets demonstrates the practicality and efficiency of SensorMCP, achieving up to 95% tool success rate in scenarios like animal monitoring. This work advances sensor systems by pioneering the co-evolution of LLMs and sensor tools, offering a scalable framework for customized sensing in mobile systems. The source code and dataset are publicly available at https://sensormcp.github.io/sensor-mcp/.

CCS Concepts

• **Computer systems organization → Sensor networks**; • **Computing methodologies → Artificial intelligence**.

Keywords

SensorMCP, Model Context Protocol (MCP), Large Language Models (LLMs), Sensor Systems

ACM Reference Format:
Yunqi Guo, Guanyu Zhu, Kaiwei Liu, Guoliang Xing. 2025. SensorMCP: A Model Context Protocol Server for Custom Sensor Tool Creation. In *3rd International Workshop on Networked AI Systems (NetAISys '25), June 23–27, 2025, Anaheim, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3711875.3736687

1 Introduction

The rapid expansion of sensor-driven applications, from wildlife conservation to smart infrastructure, has heightened the demand for customized sensor systems tailored to specific use cases. For example, monitoring endangered wildlife, such as tigers and lions, in dense jungle environments requires smart cameras with precise motion detection and low-light capabilities, while IoT deployments

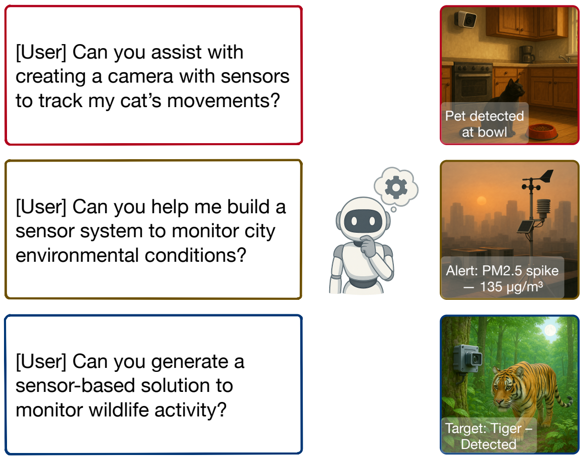
[User] Can you assist with creating a camera with sensors to track my cat's movements?
[User] Can you help me build a sensor system to monitor city environmental conditions?
[User] Can you generate a sensor-based solution to monitor wildlife activity?
Pet detected at bowl
Alert: PM2.5 spike — 135 μg/m³
Target: Tiger – Detected

Figure 1: SensorMCP enables LLM agents to dynamically generate and operate context-aware sensor tools for diverse real-world applications.

for environmental sensing demand adaptable temperature or humidity monitors suited to urban or rural contexts. These scenarios underscore a critical need for sensor systems that can be rapidly configured to meet diverse domain-specific requirements and hardware constraints. However, developing such systems remains labor-intensive, often requiring manual design of software and tools for each sensor type and application use case, limiting scalability and flexibility. This challenge is further complicated by diverse sensor hardware (cameras, thermometers, motion detectors), the demands of real-time data processing, and the need for specialized configurations. Traditional approaches, which rely on manual tuning or fixed software stacks, lack the flexibility needed for new applications.

Recent advances in large language models' (LLMs) ability to interact with basic operational tools present an opportunity to simplify sensor system customization. LLMs are particularly good at understanding natural language commands, making complex systems easier to control. The Model Context Protocol (MCP) [3] enhances this capability through an open, standardized framework that enables LLMs to dynamically invoke external tools. Using interfaces like JSON-RPC via stdio or HTTP, MCP allows LLMs to call functions, such as querying APIs or controlling devices, with structured inputs and outputs. This protocol powers applications like code interpreters [32] and application operation [1], demonstrating its potential to bridge AI and external systems. Some MCP tools even enable the operation of sensor systems like Home Assistant [24]. However, these implementations remain general-purpose and lack optimization for custom sensor systems. They neither support automated generation of sensor-specific tools nor incorporate semantic understanding of sensor data, such as distinguishing wildlife movements, like those of tigers or lions, from background noise or interpreting time-series environmental signals. Additionally, they cannot build processors for specific user requirements, like tracking pet feeding frequency. As a result, integrating LLM-generated tools into physical systems still demands substantial manual effort.

This gap motivates our central research question: *Can we enable the LLM agents to dynamically generate and operate sensor tools tailored to specific needs?* Addressing this question entails several challenges. First, automating tool creation for diverse sensor hardware, ranging from cameras to environmental sensors, requires a flexible, scalable pipeline. Second, LLMs must understand sensor-specific contexts, such as interpreting image data for wildlife detection or time-series signals for environmental trends, to generate relevant tools. Finally, integrating these tools seamlessly into the current MCP client-server model demands a robust processing framework and flexible adaptability.

To address these challenges, we propose **SensorMCP**, an innovative MCP server framework that empowers LLMs to create and operate sensor tools dynamically through a language-tool co-development pipeline. SensorMCP automates tool generation, ensuring the tool set evolves alongside the LLM agent to meet user requirements. For instance, a prompt like "monitor wildlife" triggers the creation of a tailored object-detection tool, with feedback refining both the tool and the LLM's understanding. By integrating sensor-specific language assets, SensorMCP enables LLMs to interpret and act on complex sensor systems, enhancing precision and usability.

In summary, this paper makes the following contributions:

(1) We propose a novel SensorMCP framework that enables automated co-development of sensor tools and language models, streamlining sensor system customization.
(2) We develop an automated sensor toolbox that generates customized tools on demand based on specific user requirements and contexts (e.g., for wildlife tracking, such as tigers and lions).
(3) We design an automated language assets that produce tool descriptions and linguistic modules, enabling LLMs to interact seamlessly with sensors.
(4) We implement and evaluate a prototype using real-world zoo datasets, demonstrating the system's feasibility in realistic scenarios.

These contributions advance the coevolution of sensor tools and language in mobile sensing systems, aligning with Varela's enactivist pragmatism philosophy [29], which posits that cognition arises from the bidirectional coupling of agents and their environments. To promote community collaboration and real-world deployment, we release SensorMCP's code and real-world zoo datasets through https://sensormcp.github.io/sensor-mcp/. This will facilitate sensor system applications such as wildlife monitoring, smart home systems, and smart cities.

2 Related Work

Sensor Systems Customization. IoT frameworks like Home-Assistant support customized automation for cameras and environmental sensors [21]. However, their manual configuration limits scalability for dynamic tasks like wildlife monitoring. Recent work on edge IoT frameworks optimizes sensor deployment for tasks such as activity tracking [15]. These systems enhance hardware efficiency but lack LLM-driven tool generation, leaving a gap in intelligent, automated sensor customization critical for applications like tiger tracking for the end users.

LLM-Tool Interaction Frameworks. Frameworks like Tool-LLM [17], HuggingGPT [22], TaskMatrix.AI [9], LLMind [4], enable LLM agents to operate tools by chaining prompts and APIs [25]. Several existing toolchains, such as LangChain [25], provide convenient implementations of such functionality. Building upon this paradigm, [12, 17] further explore better API selection strategies. However, their ad-hoc integrations lack standardization for sensor applications. TaskSense [10] proposes a sensor language designed for operating the sensor tools through LLM interactions. However, it is limited to the predefined tools. Open-source LLMs, such as LLaMA [26], DeepSeek [5, 6], face similar integration challenges [26]. Other works, including ToolFormer [20], achieve this by treating the function calls as a special kind of tokens to train the model. However, these methods have limited scalability due to their training overhead. The Model Context Protocol, introduced by Anthropic in 2024 [3], standardizes LLM-tool communication via JSON-RPC. Similarly, OpenAI's Function Calling API [16] and Google's Agent2Agent (A2A) protocol [14] enable agentic tool invocation. However, these frameworks are generic, lacking sensor-specific automation or data semantics for IoT and mobile sensing.

AI-Driven Sensing. Foundation models, such as vision-language models (VLMs) like CLIP, excel at general tasks like object recognition from sensor data [18]. Multimodal LLMs for IoT integrate data streams, such as video and audio [7], but their high computational demands make them infeasible for edge devices or long-term sensing tasks [2]. Agentic systems like AutoGen orchestrate tasks such as code generation [31], but they lack pipelines for generating executable sensor tools, limiting their applicability to sensing domains.

Unlike prior work, SensorMCP integrates the standardized Model Context Protocol with a co-development pipeline to automate sensor-specific tool generation and language asset development. This approach bridges large language models and sensor systems, enabling scalable customization for applications such as wildlife monitoring and smart cities. By addressing limitations in existing frameworks, Name advances AI-driven mobile sensing.

3 SensorMCP Design

The SensorMCP framework leverages the Model Context Protocol (MCP) to enable LLM agents to dynamically generate and operate sensor tools tailored to specific sensor applications, such as wildlife monitoring, smart city, and home care systems. The framework's design comprises four key components: 1) a system architecture with a co-development pipeline, 2) an automated sensor toolbox, 3) an automated language asset system, and 4) an MCP server-client model. Working in concert, these components automate tool creation and operation, enable LLM agents to interpret user requirements and generate appropriate sensor tool sets, and integrate tools seamlessly with sensor hardware. The design prioritizes scalability and adaptability, making SensorMCP a novel contribution to AI-driven mobile sensing systems.

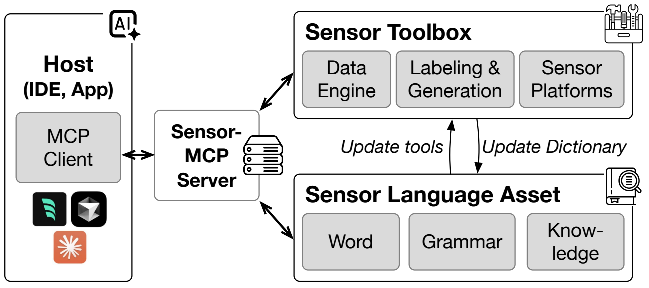

Figure 2: SensorMCP architecture.

3.1 System Overview

SensorMCP is an MCP server framework that enables LLMs to generate and operate sensor tools on demand while improving their understanding of sensor contexts and operations. Unlike traditional MCP implementations, which rely on predefined tools, SensorMCP introduces a *co-development pipeline* that evolves tools and LLMs in tandem. In this pipeline, an LLM issues a prompt (e.g., "monitor tigers"), triggering SensorMCP to generate a tailored tool (e.g., tiger_tracker). The tool then provides feedback, such as performance metrics, to enhance the LLM agent's comprehension and improve future tool designs. This iterative process ensures that tools align with user requirements and that LLMs can operate them effectively.

The SensorMCP architecture is a four-tier system, as depicted in Figure 2:

- **Host with MCP Client**: An LLM application (e.g., Claude Desktop or Cursor) that issues natural language prompts, such as "monitor tigers." The integrated MCP client translates these prompts into structured JSON-RPC requests and manages tool discovery by querying the server's tool registry.
- **SensorMCP Server**: Orchestrates tool generation and language asset creation by controlling the Sensor Toolbox and Language Asset system. It handles tool invocation, processes feedback, and ensures seamless integration with sensor hardware via APIs like `create_tool` and `invoke_tool`.
- **Sensor Toolbox**: Automates the generation of scenario-specific tools, such as tiger trackers, from MCP requests, producing trained models and function descriptions. It also supports extensibility through compatibility with existing sensor platforms like Home Assistant [21] and Mi-Home, enabling integration with diverse IoT ecosystems.
- **Sensor Language Asset**: Maintains a dynamic repository of tool descriptions and schemas, enhancing LLM understanding of tool capabilities. It updateIns this "menu" of sensor tools based on performance feedback, ensuring agents can accurately invoke and interpret tool functions.

This architecture automates the entire tool creation process, eliminating manual efforts and enabling rapid deployment for diverse sensing scenarios. The co-development pipeline is central to SensorMCP's innovation, ensuring tools are both functional and interpretable by LLMs, a critical requirement for real-world mobile sensing applications.

Table 1: Sensor Toolbox Examples

Input (MCP Request)	Output (Tool)	Function Description
`{goal: "monitor", subject: "tigers"}`	Tiger Tracker	`track_video(): detect tigers in real-time`
`{goal: "measure", subject: "temp"}`	Temp Logger	`log_temp(): record temperature at intervals`

3.2 Sensor Toolbox

In SensorMCP server, the Sensor Toolbox is an automated pipeline that produces sensor tools on demand from the user and MCP host requests, enabling scenario-specific tool creation without requiring users to provide data or labels. It processes a structured MCP request, such as a JSON object *{goal: "object monitor", subject: "tigers"}*, to produce a fully functional tool, like a tiger tracker with a trained model, deployment libraries, and function descriptions in the returned output. Operating without manual intervention, the pipeline ensures scalability across diverse sensor types and applications. Additionally, it maintains a repository of predefined and generated tools and machine learning modules, allowing users to access and reuse them efficiently.

The pipeline consists of four sequential steps, as illustrated in Figure 3:

(1) **Data Engine**: The data engine retrieves relevant data for creating the tool based on the request. For example, a tiger monitoring tool fetches tiger images from public datasets like Roboflow [19] and Unsplash [28], ensuring domain-specific training material.
(2) **Foundation Model-Based Labeling**: This module uses the large machine learning models, such as YOLO-World [30], Grounding DINO [11], annotates the collected data to identify relevant features (e.g., tigers in images), producing labeled datasets for training.
(3) **Tool Generation**: The pipeline trains a compact, efficient model, such as YOLOv8, YOLOv10, and YOLOv11 [27, 30], optimized for real-time performance on resource-constrained sensor hardware (e.g., Raspberry Pi).
(4) **Tool Packaging**: The trained model is bundled with metadata, including function descriptions (e.g., "tiger_ tracker: detect tigers, invoke via track_video"), enabling MCP-compliant invocation.

The pipeline's dynamic generation capability ensures that tools are tailored to specific scenarios, such as zoo-based wildlife monitoring, without requiring predefined templates. Table 1 provides examples of pipeline inputs and outputs.

Through automated tool creation, the Sensor Toolbox enables SensorMCP to support diverse mobile sensing applications, from environmental monitoring to object tracking, while maintaining minimal manual operation and high adaptability.

3.3 Sensor Language Asset

Beyond tool generation, SensorMCP maintains a Sensor Language Asset, a specialized dictionary that helps LLM agents understand the tools and their functions. This asset automatically generates and refines tool descriptions and schema, allowing LLM agents to interact seamlessly with the generated tools. It solves a crucial challenge: ensuring LLMs comprehend sensor-specific contexts,

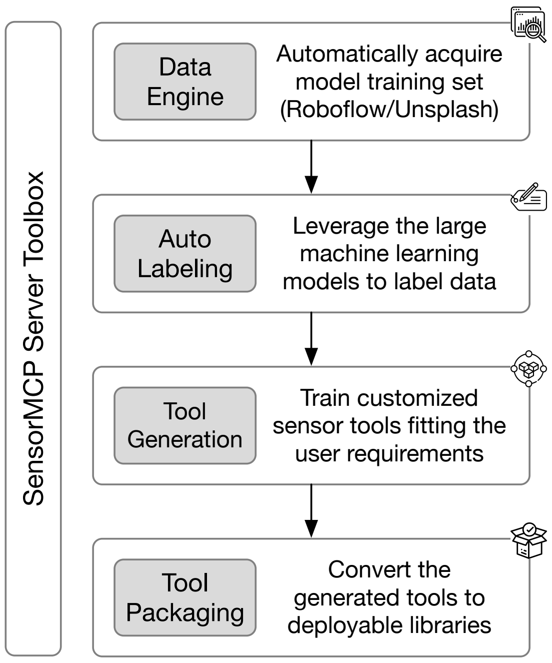
SensorMCP Server Toolbox
Data Engine
Automatically acquire model training set (Roboflow/Unsplash)
Auto Labeling
Leverage the large machine learning models to label data
Tool Generation
Train customized sensor tools fitting the user requirements
Tool Packaging
Convert the generated tools to deployable libraries

Figure 3: Automated tool generation pipeline.

such as the operational limitations of a tiger tracker. Specifically, the Sensor Language Asset consists of three core components.

- **Word Formation**: Defines tool affordances and features in natural language (e.g., "tiger_tracker: detects tigers in real-time using camera input").
- **Grammar Formation**: Generates operational schemas that specify tool behavior (e.g., "[tiger_tracker] activates if [motion > threshold during daytime]").
- **Embedded Knowledge**: Incorporates sensor data samples, such as image metadata, to enhance LLM context understanding (e.g., associating "tiger" with specific visual patterns).

The generation and refinement process begins when the Sensor Toolbox generates a tool. The system automatically creates descriptions and schemas based on the tool's model, training data, and metadata information. For instance, a tiger tracker's model output (e.g., bounding boxes) informs a schema such as "[tiger_tracker] reports [detection] at [timestamp]." LLMs use these assets to invoke tools and interpret their outputs, such as "tiger detected at 14:00." A feedback loop continuously refines these assets—tool performance metrics (e.g., false positives) trigger updates to descriptions and schemas, enhancing accuracy over time. These linguistic elements correlate directly with sensor development modules, as shown in Table 2: tool functions influence word formation, schemas align with syntax, and performance metrics shape contextual narratives.

This automated approach ensures LLMs "speak" the language of sensors, enabling precise tool operation and adaptation. By embedding sensor context and refining assets dynamically, the system enhances the co-development pipeline, making SensorMCP a robust solution for AI-driven sensing.

Table 2: Correlation between sensor development modules and linguistic elements in SensorMCP.

Sensor Module	Linguistic Element	Enactivist Relation
Tool Functions	Word Formation	Sensor tools shape metaphors: "tracking" tigers mirrors visual detection tasks.
Operational Schemas	Grammar Formation	Tool sequences inform syntax: tool-action-object mirrors prompt-tool-output.
Tool Performance	Contextual Narratives	Tool successes/failures drive descriptions: e.g., "tiger detected" logs refine usage.
Embedded Knowledge	Pragmatic Context	Tool metadata creates jargon; jargon guides tool invocation (e.g., "track_video").

Table 3: SensorMCP Server APIs

API Call	Description
`create_tool(goal, subject)`	Generates a new tool based on request parameters
`list_tools()`	Returns available tools in the registry
`invoke_tool(tool_name, params)`	Executes a tool with given parameters

3.4 MCP Server and Client

The SensorMCP server and client form the backbone of tool generation and invocation, integrating the toolbox and language assets into a cohesive system. The server exposes these components via MCP's JSON-RPC interface, supporting two primary operations:

- **Tool Generation**: Processes requests like `create_tool( goal="monitor",subject="tigers")` to trigger the Sensor Toolbox pipeline.
- **Tool Invocation**: Executes commands like `invoke_tool( "tiger_tracker")` on sensor hardware.

The **server** supports both local sensors (via stdio) and remote sensors (via HTTP with Server-Sent Events[SSE]), ensuring flexibility across deployment scenarios. It maintains a dynamic tool registry that updates in real-time when new tools are generated, and it enforces security scopes to restrict LLM access, for example, making sensor data read-only.

The **client** converts LLM prompts into server requests by translating natural language commands (e.g., "monitor tigers") into structured APIs (e.g., `create_tool`). It dynamically discovers available functions by querying the tool registry. Table 3 presents these two categories of server APIs.

This design ensures seamless integration of generated tools and language assets, enabling LLMs to operate sensors with minimal overhead. The server-client model, combined with the automated pipeline and language system, positions SensorMCP as a scalable framework for mobile sensing applications.

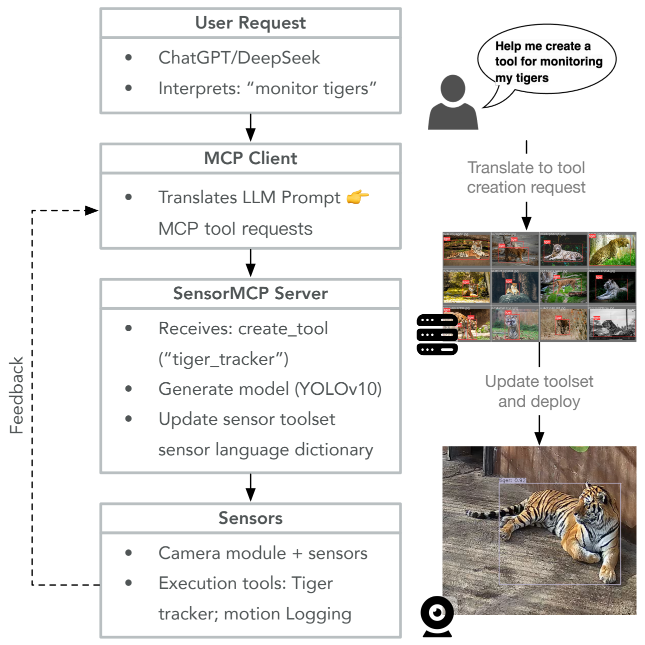
User Request
ChatGPT/DeepSeek
Interprets: "monitor tigers"
MCP Client
Translates LLM Prompt 👉 MCP tool requests
SensorMCP Server
Receives: create_tool ("tiger_tracker")
Generate model (YOLOv10)
Update sensor toolset sensor language dictionary
Sensors
Camera module + sensors
Execution tools: Tiger tracker; motion Logging
Feedback
Help me create a tool for monitoring my tigers
Translate to tool creation request
Update toolset and deploy

Figure 4: Prototype workflow of SensorMCP: a "monitor tigers" or "monitor lions" prompt triggers tool generation, delivering an object detection model to a programmable smart camera. The camera processes live feeds and visualizes detected tiger or lion events.

4 Implementation and Evaluation

To demonstrate the feasibility of SensorMCP, we developed a prototype implementation and conducted a preliminary evaluation using real-world data collected from a zoo. This section describes the prototype setup, including the tools generated and the workflow, followed by an evaluation of tool generation accuracy and sensor effectiveness. Our results highlight SensorMCP's potential to enable dynamic, AI-driven sensing.

4.1 Implementation

We implemented the SensorMCP server in Python, leveraging an open-source MCP SDK to handle client-server communication via JSON-RPC over HTTP. The prototype comprises four components: the host LLM, MCP client, SensorMCP server, and sensor hardware. For the host LLM, we used an open-source tool FastAgent [23], interfaced through an MCP client that translates natural language prompts into server requests. The SensorMCP server automates tool generation, exposing interfaces including `create_tool` and `invoke_tool` built based on FastMCP [13]. To collect real-world data, we deployed three smart cameras in a zoo over one month.

Two example tools were generated to validate the system: a tiger tracker and a lion tracker, both based on object detection. These trackers process camera feeds using a distilled YOLOv10 model trained on real-world zoo images. Each tool continuously analyzes incoming frames to detect and track tigers or lions in real time. The workflow follows a prompt-driven pipeline: a user inputs a request (e.g., "monitor tigers" or "monitor lions"), the LLM generates a tool specification, the SensorMCP server produces the tool – including the detection model, metadata, and deployment setup in Python environment – and the tool is then deployed on the hardware for execution.

Figure 4 illustrates the prototype workflow, showing a tiger or lion tracker in action: a prompt triggers tool generation, the server delivers a motion-detection model to process the video feed, displaying detected events. This setup demonstrates SensorMCP's ability to automate tool creation and integrate LLM agents with sensor hardware seamlessly.

Table 4: SensorMCP Evaluation: Wildlife Tracking

Test Case	Success (%)	Precision/Recall (%)	Time
Tiger Tracking	95	96.9 / 85.8	27m 13s
Lion Tracking	90	93.9 / 86.7	27m 5s

Table 5: Impact of Tool Customization

Test Case	Method	Precision (%)	Recall (%)
Tiger Tracking	Pre-trained only	88.7	68.9
	SensorMCP	**96.9**	**85.8**
Lion Tracking	Pre-trained only	82.1	47.9
	SensorMCP	**93.9**	**86.7**

4.2 Evaluation

We evaluated SensorMCP across two dimensions: tool generation success rate and sensor effectiveness, using real-world data collected from a zoo to ensure realism. Our methodology focused on two test cases for tiger and lion monitoring. The datasets comprised video footage captured by three smart cameras over one month in a zoo under varying conditions (e.g., daytime and nighttime).

Tool Generation Success Rate. We measured the success rate of generating tools that match user prompts. Across 40 test prompts (e.g., "monitor tigers"), we assessed whether the generated tool correctly implemented the requested functionality. For instance, a "monitor tigers" or "monitor lions" prompt should yield a motion-detection tool with a YOLOv10 model configured for tiger or lion recognition, respectively.

Tool Effectiveness. We assessed the quality of the tool output, focusing on the precision and recall of the tiger and lion trackers. Tests were conducted using real-world zoo scenarios, such as varying lighting conditions and angles for tiger and lion tracking.

Metrics. Evaluation metrics included: 1) Tool success rate: Percentage of prompts yielding functional tools. 2) Precision and Recall: Performance of tools generated by the MCP server. 3) Latency: Time for tool generation and invocation via the MCP server.

Results are summarized in Table 4. Across 40 prompts, SensorMCP achieved a 95% tool success rate for tiger tracking, with 38 tiger tracking tools correctly matching user intent (e.g., tiger trackers detecting motion as specified). The tiger tracker exhibited a 96.89% precision rate and 85.82% recall rate. Latency averaged 26 minutes and 52 seconds for tool generation and 21 seconds for invocation, suitable for real-time applications. Results for tiger tracking, based on real-world data, suggest SensorMCP's viability, though performance depends on prompt clarity and data quality.

To assess the impact of sensor tool customization enabled by SensorMCP, we conducted a comparative evaluation between pre-trained models and our system's auto-generated tools. The pre-trained models consisted of a YOLO model trained on the general-purpose Open Image [8] dataset, while our approach leveraged task-specific data during tool generation. As shown in Table 5, our tools significantly outperformed the generic ones, achieving over 8.2% higher precision and 16.9% higher recall in the monitoring tasks. These results validate the effectiveness of custom models with context-specific sensor observations.

5 Conclusion and Future Work

This paper introduces SensorMCP, a novel framework that leverages the Model Context Protocol to enable LLM agents to dynamically generate and operate sensor tools. By integrating an automated sensor toolbox and sensor language assets, SensorMCP streamlines the creation of custom tools for various applications. Our prototype demonstrates the feasibility of this approach, achieving promising results in scenarios such as tiger tracking with a 95% tool success rate. SensorMCP simplifies the development of tailored sensor systems. Its co-development pipeline ensures that tools evolve alongside LLMs, enhancing adaptability and precision in real-time sensing tasks.

Future work will focus on integrating with existing sensor operation platforms like Home Assistant [21] to support commands such as "help me monitor my dog's diet" and facilitate hardware customization requests like "help me build a pet companion that looks like a toy duck."

Acknowledgment

This work was partly supported by the Hong Kong Research Grants Council (RGC) under the Theme-based Research Scheme (Grant Number: T43-513/23-N).

References

[1] Siddharth Ahuja. 2025. BlenderMCP: Blender Integration through the Model Context Protocol. https://github.com/ahujasid/blender-mcp Accessed: 2025-04-15.

[2] Tianyi Bai, Hao Liang, Binwang Wan, Yanran Xu, Xi Li, Shiyu Li, Ling Yang, Bozhou Li, Yifan Wang, Bin Cui, Ping Huang, Jiulong Shan, Conghui He, Binhang Yuan, and Wentao Zhang. 2024. A Survey of Multimodal Large Language Models from a Data-centric Perspective. *arXiv preprint arXiv:2405.16640* (2024). https://doi.org/10.48550/arXiv.2405.16640

[3] Model Context Protocol Contributors. 2024. *Model Context Protocol (MCP): Specification and SDKs*. Model Context Protocol. https://github.com/modelcontextprotocol

[4] Hongwei Cui, Yuyang Du, Qun Yang, Yulin Shao, and Soung Chang Liew. 2024. Llmind: Orchestrating ai and iot with llm for complex task execution. *IEEE Communications Magazine* (2024).

[5] DeepSeek-AI. 2024. DeepSeek-V3 Technical Report. https://arxiv.org/abs/2412.19437.

[6] DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. https://arxiv.org/abs/2501.12948.

[7] Difei Gao, Yiwen Liu, Zixuan Zhang, Binfeng Jin, Zhendong Chen, Wenhao Dai, Yu Yang, Yi Wang, Hang Zhang, and Zhao Yang. 2024. Mini-InternVL: A Flexible-Transfer Vision-Language Model for IoT Sensing. *arXiv preprint arXiv:2408.15254* (2024). https://doi.org/10.48550/arXiv.2408.15254

[8] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, Tom Duerig, and Vittorio Ferrari. 2020. The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale. *International Journal of Computer Vision (IJCV)* (2020).

[9] Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. 2024. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *Intelligent Computing* 3 (2024), 0063.

[10] Kaiwei Liu, Bufang Yang, Lilin Xu, Yunqi Guo, Guoliang Xing, Xian Shuai, Xiaozhe Ren, Xin Jiang, and Zhenyu Yan. 2025. TaskSense: A Translation-like Approach for Tasking Heterogeneous Sensor Systems with LLMs. In *Proceedings of the 23rd ACM Conference on Embedded Networked Sensor Systems*. 213–225.

[11] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, et al. 2024. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. In *European Conference on Computer Vision*. Springer, 38–55.

[12] Zhaoyang Liu, Zeqiang Lai, Zhangwei Gao, Erfei Cui, Ziheng Li, Xizhou Zhu, Lewei Lu, Qifeng Chen, Yu Qiao, Jifeng Dai, et al. 2024. Controlllm: Augment language models with tools by searching on graphs. In *European Conference on Computer Vision*. Springer, 89–105.

[13] Jonathan Lowin. 2025. FastMCP: A Pythonic Framework for Building MCP Servers and Clients. https://github.com/jlowin/fastmcp. Accessed: April 18, 2025.

[14] MarkTechPost Team. 2025. *Google Introduces Agent2Agent (A2A): A New Open Protocol that Allows AI Agents Securely Collaborate Across Ecosystems*. https://www.marktechpost.com/2025/04/04/google-introduces-agent2agent-a2a-a-new-open-protocol-that-allows-ai-agents-securely-collaborate-across-ecosystems-regardless-of-framework-or-vendor/

[15] Shentong Mo, Russ Salakhutdinov, Louis-Philippe Morency, and Paul Pu Liang. 2024. IoT-LM: Large Multisensory Language Models for the Internet of Things. *arXiv preprint arXiv:2407.09801* (2024). https://doi.org/10.48550/arXiv.2407.09801

[16] OpenAI Team. 2024. *Function Calling with OpenAI API*. https://platform.openai.com/docs/guides/function-calling

[17] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789* (2023).

[18] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*. 8748–8763. https://doi.org/10.48550/arXiv.2103.00020

[19] Roboflow. 2025. Roboflow Universe: Open Source Computer Vision Datasets and Models. https://universe.roboflow.com. Accessed: 2025-04-15.

[20] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems* 36 (2023), 68539–68551.

[21] Paulus Schoutsen and the Home Assistant Community. 2024. *Home Assistant: Open Source Home Automation*. Open Home Foundation. https://www.home-assistant.io/

[22] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems* 36 (2023), 38154–38180.

[23] Shaun Smith. 2025. FastAgent: Define, Prompt, and Test MCP-Enabled Agents and Workflows. https://github.com/evalstate/fast-agent. Accessed: April 18, 2025.

[24] Tevon Smith. 2025. HomeAssistant-MCP: Model Context Protocol Server for Home Assistant. https://github.com/tevonsb/homeassistant-mcp Accessed: 2025-04-15.

[25] LangChain Team. 2024. *LangChain: Framework for LLM-Powered Applications*. LangChain Inc. https://github.com/langchain-ai/langchain

[26] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971* (2023). https://doi.org/10.48550/arXiv.2302.13971

[27] Ultralytics. 2025. Ultralytics YOLO11: A State-of-the-Art Object Detection Framework. https://github.com/ultralytics/ultralytics. Accessed: 2025-05-11.

[28] Unsplash. 2025. Hermosas imágenes y fotos gratuitas. https://unsplash.com/es Retrieved May 10, 2025.

[29] Francisco J. Varela, Evan Thompson, and Eleanor Rosch. 2017. *The Embodied Mind: Cognitive Science and Human Experience* (revised ed.). MIT Press, Cambridge, MA. https://doi.org/10.7551/mitpress/9780262529365.001.0001

[30] Ao Wang, Hui Chen, Lihao Liu, Lin Song, Jungong Han, and Guiguang Ding. 2024. YOLOv10: Real-Time End-to-End Object Detection. *arXiv preprint arXiv:2405.14458* (2024). https://arxiv.org/abs/2405.14458

[31] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework. *arXiv preprint arXiv:2308.08155* (2023). https://doi.org/10.48550/arXiv.2308.08155

[32] Edward Z. Yang. 2025. codemcp: Coding Assistant MCP for Claude Desktop. https://github.com/ezyang/codemcp Accessed: 2025-04-15.