rosaOS: Agentic Operating System for Embodied LLMs

Yijun Ge* **Kushal Mujral*** **Karthik Nambiar** **Jimmy Lin**
David R. Cheriton School of Computer Science, University of Waterloo
{l2ge, kmujral, k3nambia, jimmylin}@uwaterloo.ca

Abstract

We present rosaOS, an open-source agentic operating system for embodied LLMs: interactive, LLM-driven agents coordinate various software tools and physical devices through a desktop companion, the Reachy Mini robot. Existing LLM–robotic systems are generally built as a tight, intertwined stack, making it difficult to switch hardware, add extra capabilities, or expand to multiple devices without bespoke integration. Our system aims to provide a classic OS-inspired architecture where an agentic kernel manages all task execution and mediates device access, while process agents invoke tools to perform actions. We adopt industry-standard interfaces with MCP for agentic tool-calling and ROS for robot interactions, and demonstrate rosaOS on a multi-device setup including a quadruped robot, a wheeled mobile robot, and a smart lamp, all controlled through interactions with the Reachy Mini. By incorporating MCP extensibility with ROS hardware interoperability, rosaOS enables a plug-and-play ecosystem for building embodied agentic systems. Our OS is available at rosaos.ai.

1 Introduction

With the rise and rapid advancements of large language models (LLMs) in recent years, the focus of further development has gradually shifted to agentic capabilities, where language models make use of their reasoning capabilities and available tools to complete complicated tasks. The Model Context Protocol (MCP)[1] was introduced by Anthropic in 2024 as a way to bridge the gap between LLMs and tools, providing an open standard for exposing tools to LLM agents.

A natural way forward for this trend is the embodiment of LLMs, giving them the ability to interact in physical space through robots and other hardware. The standard in the robotics development community for robot manipulation and interaction lies in the Robot Operating System (ROS) (Quigley et al., 2009), which provides users a communication middleware and tooling used across the majority of research and many commercial robotic systems today (Macenski et al., 2022). Connecting LLM agents to physical hardware in a principled, extensible way remains an open challenge. We have been working on this problem since December 2025, concurrent with the surge in popularity of agentic frameworks such as OpenClaw,[2] driven by increasingly capable frontier models.

We introduce rosaOS, an open-source agentic operating system for embodied LLMs, to connect language models with coordinated hardware control by applying agents to traditional operating system design principles. Our system integrates industry standards across software and hardware with MCP and ROS, creating an open, extensible platform to help researchers and enterprise practitioners further the work of bringing LLMs into the tangible world.

2 Related Work

Agentic Harnesses. Agentic frameworks provide LLMs with tool-use, memory, and sub-agent orchestration capabilities through a unified interface. The current landscape of autonomous task completion mixes general-purpose agentic harnesses, such as OpenClaw, with domain-specific platforms, such as the coding-oriented Claude Code and Codex. The successful coexistence of these systems highlights a clear opportunity for the development of novel, specialized frameworks tailored to other domain-specific tasks such as embodiment.

While OpenClaw serves as a robust foundation for general software tasks, its architecture was not originally designed with physical, multi-device coordination as a primary concern. Recent efforts have explored extending agentic frameworks to

* Equal Contribution
[1] modelcontextprotocol.io
[2] openclaw.ai

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), pages 473–480

robotics; in particular, two groups each released a system named ROSClaw concurrently with our work on rosaOS: Cardenas et al. (2026) introduced ROSClaw as an OpenClaw ROS 2 framework for agentic robot control, while Zhao et al. (2026) developed a system of the same name that uses a vision-language model to coordinate multiple heterogeneous robots across simulated and real-world environments. Our system differs from both in scope and interface: rather than extending an existing framework, it is built around an OS-inspired kernel-process architecture from the outset, adopts ROS and MCP as complementary industry standards for hardware and tool integration, and uses a physical desktop robot as its primary user interface rather than a software terminal.

Operating Systems and Agents. The modern operating system provides a unified interface that allows many programs to run concurrently while coordinating shared resources; layers of abstraction built on this foundation keep complex workloads manageable and multi-program interactions dependable (Silberschatz et al., 2018). In parallel, LLMs have transitioned from simple turn-based text interfaces to multimodal systems capable of actions beyond simple conversation (OpenAI, 2023). Now with standardized interfaces like MCP, agents can easily call upon external services.

This progress pushes LLM systems into a new integration problem: enabling coordination of heterogeneous tools safely through a single agentic interface, especially when multiple tasks could be contending for the same resources. This problem motivates an OS-like structure since resources are no longer just compute and context, but also real-world actuators and components, which cannot be accessed concurrently without explicit arbitration. Mei et al. (2025) demonstrate this with AIOS, an LLM agent operating system whose kernel provides scheduling, context management, memory management, and access control for concurrent software agents; rosaOS extends this paradigm to encompass physical hardware through standardized device interfaces.

LLMs in Robotics. Recently, LLMs have been used as controllers for robotics, specifically as high-level planners or as policy components mapping out perception into control outputs. This shows that natural language can provide an interface for manipulation and navigation, but these examples often are demonstrated as tightly knit components

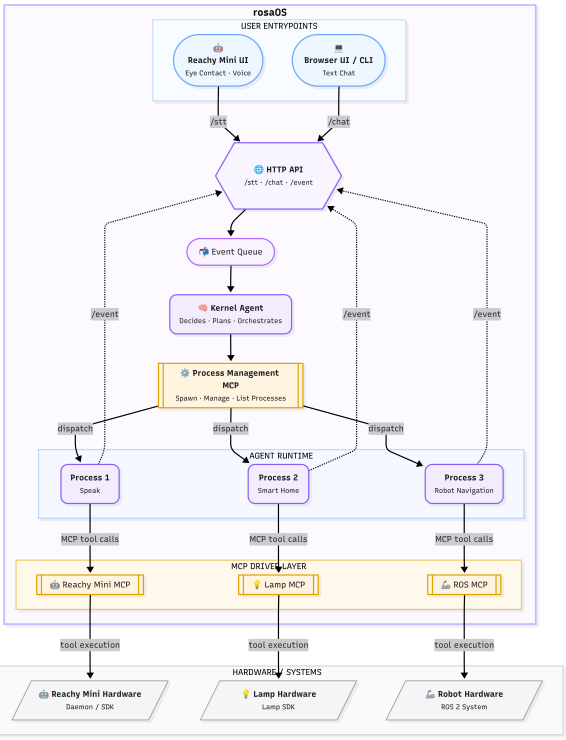


Figure 1: The rosaOS architecture. User input enters a central event queue consumed by the kernel agent, which dispatches process agents to complete tasks via MCP tool calls to hardware devices. Process agents report back to the kernel upon completion.

where the robot, controller, and model interface are designed together (Driess et al., 2023; Ichter et al., 2023; Zitkovich et al., 2023; Jiang et al., 2023). ROS, on the other hand, has become a popular middle layer used to connect multiple devices to enable modularity and standardize message passing for robots (Quigley et al., 2009). Bridging LLM tool-use to ROS in a repeatable manner remains an open engineering gap, especially in situations where multiple robots and devices must be coordinated in real time (Vemprala et al., 2023; Mower et al., 2024).

3 Architecture Overview

At its core, rosaOS is an agentic loop wrapped in operating-system abstractions, where a kernel agent reads from an event queue, decides what to do, and either responds to the user or delegates work to a process agent; process agents execute their tasks using MCP tools and finally post their results back onto the same queue. Everything the system does flows through this queue and is handled by the kernel one at a time, including user speech, chat input, worker agent callbacks, and er-

rors. The queue serializes a chaotic, asynchronous world of multiple devices, in-flight tasks, and input modalities into a coherent stream the kernel agent can reason about turn by turn, as shown in Figure 1. Users interact with this underlying loop through the Reachy Mini desktop robot as the primary user interface and any other robots or hardware devices they configure.

Our LLM agents are built on the Pydantic AI[3] framework and support proprietary models from OpenAI, Anthropic, or Gemini, open-source models from the inference provider Groq, or any model with an OpenAI-compatible endpoint. All external API dependencies have self-hosted alternatives, for example, a locally hosted OpenAI-compatible endpoint for LLM inference, local Whisper (Radford et al., 2023) for speech-to-text (STT), and pyttsx3 for speech synthesis,[4] enabling fully offline deployments for those who prefer to manage their own infrastructure.

LLMs are connected to tools through MCP servers over HTTP via the FastMCP[5] framework, and to hardware through device-specific libraries and the ROS framework. To show the extensibility of our platform, we integrate a quadruped robot, a wheeled mobile robot, and a smart lamp, all controlled through the Reachy Mini desktop robot. The system is highly configurable, for example, the set of active devices, agent system prompts, and choice of LLM can all be adjusted without modifying the core codebase. The complete codebase can be found at `rosaos.ai`.

We trace a concrete example—the user asking the Reachy Mini to "turn on the light"—through each layer of the system, and include references to relevant sections for more details.

1. **Wake and listen.** The Reachy Mini continuously monitors for a wake phrase or eye contact; once triggered, its antennas wave and it begins capturing audio (Section 4.3).

2. **Speech-to-text.** Captured audio is transcribed and the transcript is POSTed to the kernel's `/stt` endpoint, where it enters the event queue (Section 4.3).

3. **Kernel reasoning.** The kernel agent dequeues the event and decides the appropriate action—here, turning on the light requires the smart light MCP server (Section 4.1).

4. **Process dispatch.** The kernel calls the launch process tool, supplying a task-specific system prompt and the list of permitted MCP servers for the new process agent (Sections 4.1 and 4.2).

5. **Tool execution.** The process agent runs and invokes the relevant MCP tool—e.g., the smart light server's on/off tool—to complete the task (Sections 4.2, 4.4, 5, and 5.2).

6. **Callback.** Upon completion, the process agent POSTs its result to the kernel's `/event` endpoint; this callback re-enters the event queue (Section 4.2).

7. **Resolution.** The kernel agent processes the callback: on success it acknowledges the result; on failure it may re-issue the task, escalate to the user via the Reachy Mini's speak tool, or abandon gracefully. This retry logic is LLM-driven rather than rule-based, allowing flexible responses to novel failure modes (Section 4.1).

4 Implementation Details

The components of rosaOS map directly onto operating system abstractions: a kernel agent manages processes and coordinates resources, process agents carry out tasks on behalf of the kernel, the Reachy Mini desktop robot serves as the primary user interface through which the user issues requests and receives responses, and MCP servers act as device drivers exposing hardware as callable tools. We now describe each of these components in turn.

4.1 Agentic Kernel with Process Management

The kernel of the operating system is what ties everything together. Our kernel is responsible for managing processes, deciding which hardware devices a process agent can access, and responding to user requests. It is implemented as an agent with a specialized system prompt.

The agent is connected to an internal MCP server started automatically with the kernel. This MCP server has tools for managing processes: it can launch a new process, terminate a process, and get the list of currently running processes. When launching a process, the kernel interprets the

[3] `ai.pydantic.dev`
[4] `pyttsx3.com`
[5] `gofastmcp.com`

present conversational context to provide a system prompt for the new process agent, which includes instructions regarding the task to complete and any relevant context from the conversation needed to perform the task.

The kernel also decides which hardware MCP servers are visible to the process agent, assigning control to only the devices necessary for that task. This per-process scoping acts as a lightweight access control mechanism: the kernel should not assign the same hardware MCP server to two concurrent processes unless the device supports concurrent access (e.g., read-only camera tools), relying on LLM reasoning to avoid conflicting actuation commands to the same actuator. Our kernel does not have access to the hardware MCP servers itself, instead delegating hardware control to processes based on the list of available devices from its system prompt to keep it free for live management of the system and responses to the user.

To interact with the rest of the system, we wrap a FastAPI[6] server around the kernel to expose endpoints that other components can communicate with. Three POST endpoints accept payloads which are added to an event queue that is constantly read from by the kernel agent as user input. These endpoints correspond to different sources: `/event` for callback from process agents, `/stt` for the Reachy Mini's STT, and `/chat` for the browser UI or CLI.

When processing an event, we inject the appropriate context of the event to formulate the message to the kernel agent; for example, we prepend the tag "[Worker callback]" for messages received from process agents and include the worker ID and task status from the payload. The kernel agent runs with this message, calling MCP tools for process management as it deems appropriate; for example, launching a new process agent with access to the Reachy Mini to alert the user of a task's completion by speaking through its speaker.

When the kernel agent finishes generating, its output is printed to the console and also put into a list accessible to our browser UI and CLI. This output is not necessary to use the operating system, as the intended primary user interface is through the Reachy Mini, though it allows the user to chat directly with the kernel through text messages.

[6] `fastapi.tiangolo.com`

4.2 Agentic Processes

Processes are agents instructed to perform a single task. The process manager MCP server has a tool used by the kernel agent to launch a process, which is run in a Python `subprocess.Popen`. The arguments used to construct a process agent consist of the worker's ID, the task-specific system prompt generated by the kernel, and the list of MCP servers that the kernel is allowing the process to access. The instructions generated by the kernel and device-specific prompts for every accessible MCP server are appended to the base system prompt for process agents to create the final system prompt.

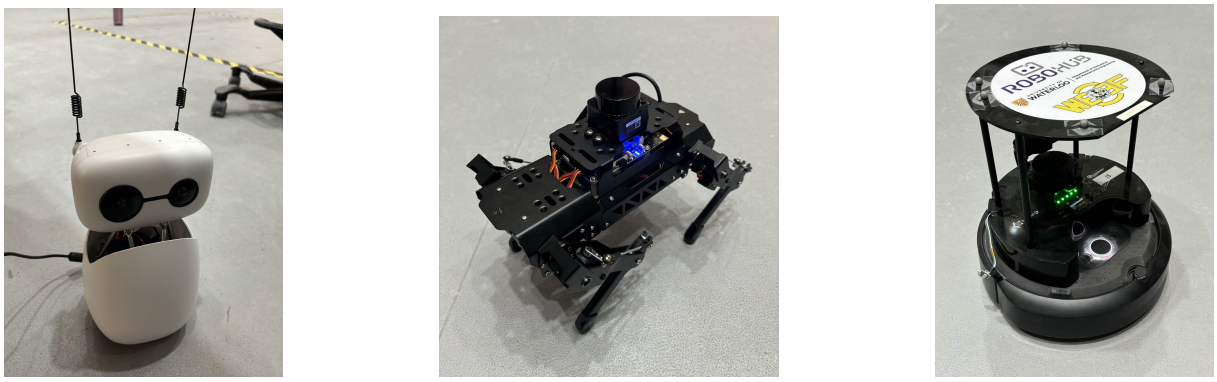
All process agents run with the same first user message, which is a simple instruction to complete its task. Running the agent returns the agent's output or any errors encountered and a flag indicating whether there was an error in the execution. This information, along with the process agent's worker ID, its system prompt, and a message indicating success based on the error flag, are sent to the kernel by posting to its `/event` endpoint, where the kernel agent will eventually run with this payload as input and respond as needed.

Starting a fresh child process for every agent run adds latency that dominates very short jobs. To address this, we have a pool of two long-lived process agents, allowing us to reuse the same Python subprocess and simply pass in different arguments for new agents. These are launched with the kernel, and in the launch process tool, we check if any are available and only launch a new subprocess if all are occupied.

4.3 User Interface

We choose the Reachy Mini, shown in Figure 2, as the primary user interface because the desktop robot is personable and makes interactions feel natural. It is the default MCP server exposed to agents through our setup, giving them access to its speak tool that they are instructed to use to report to the user. We run a set of dedicated system processes around this UI to ensure a smooth and consistent experience.

A background thread for STT records audio in chunks, starting when voice activity is detected and ending after a few moments of silence. These chunks are transcribed with the whisper-large-v3-turbo model (Radford et al., 2023); by default we call Groq's API (Groq, Inc., 2024), but any OpenAI-compatible endpoint can be used.



Reachy Mini | Hiwonder PuppyPi | TurtleBot4

Figure 2: Integrated robots used with rosaOS (left to right): the Reachy Mini desktop robot (primary user interface), the Hiwonder PuppyPi quadruped, and the TurtleBot4 mobile robot.

Prompting the interface begins by either sustaining eye contact with the Reachy Mini's camera or saying "hello". If it hears the wake phrase, the robot will spin around to face the direction of audio, based on its directional microphones, and adjust its head pose to look at the user if it can detect a face. While actively listening, the Reachy Mini's antennas run a waving animation to indicate so. The motion continues until audio is fully captured and then stops while the Reachy Mini responds and returns to the inactive state. The transcript of the recorded audio is POSTed to the kernel at the `/stt` endpoint, where it is added to the event queue to be processed.

A secondary way the user can interact with the operating system is by chatting with the kernel agent directly by text through our browser UI or CLI. Messages sent by the user through these channels are POSTed to the kernel at the `/chat` endpoint and the output is returned and displayed. To obtain other kernel outputs from the event queue, such as worker callback or STT events, the underlying scripts poll the kernel at regular intervals, where other outputs are stored in a list for this purpose.

4.4 Device Drivers

Hardware is exposed to the system through MCP servers that act as device drivers, presenting physical actuators and sensors as callable tools to the kernel and process agents. Any hardware with an accessible API or SDK can be wrapped in an MCP server and registered with rosaOS without modifying the core system. The devices currently integrated with rosaOS are described in Section 5.

The Reachy Mini occupies a special role: it is the default MCP server present in every process agent's context, since agents always need it to speak to the user—making it simultaneously the primary user interface and a first-class device driver. For robot platforms, we use ROS (Quigley et al., 2009) as the standard communication middleware, bridging robot hardware to MCP via the ROS MCP Server and the Rosbridge protocol (Mace et al., 2013); this means any ROS-compatible robot can be easily integrated into rosaOS.

5 Device Integrations

We currently integrate four devices in rosaOS: the Reachy Mini desktop robot (which also serves as the primary user interface), two ROS-enabled robots, and a smart lamp. Each is connected via an MCP server exposing a set of tools, demonstrating that the same driver pattern applies uniformly across robotics platforms and consumer smart home hardware.

5.1 Reachy Mini

The Reachy Mini,[7] created by Hugging Face's Pollen Robotics, is an open-source and fully programmable desktop robot shown in Figure 2. It has two hardware versions: Wireless, with a built-in Raspberry Pi, and Lite, which relies on a tethered host for compute. We use the Lite version for simplicity and run its daemon server on the same computer as the primary compute for the system.

The Reachy Mini's daemon is started before its MCP server to establish a connection to the robot. The MCP server provides tools for the agent to move the Reachy Mini in place, play specific dances corresponding to emotions, capture pictures

[7] huggingface.co/reachy-mini

and analyze them, and speak. We take advantage of the Reachy Mini's daemon API, which has built-in concurrency, to schedule and interrupt the robot's movement.

Image analysis capabilities across our system are consolidated within the Reachy Mini's MCP server, as it is the default MCP server that agents will almost always have access to, since they need it to speak to the user to report on their actions. We decouple vision from the core language model, to retain flexibility in choosing models that excel at reasoning and tool usage for our agents without requiring built-in multimodality. To facilitate this, we provide a dedicated image description MCP tool that calls Groq's API (Groq, Inc., 2024) for Llama 4 Scout (Meta AI, 2025), a strong multimodal model. This tool sends the base64-encoded image with a question passed in as a tool parameter as the prompt to the Llama model. By default, this prompt is a simple request to describe the image, but exposing it as a tool parameter allows the agent to customize it to their needs, for example, to inquire after a specific detail. Additionally, we offer tools for detecting faces in an image, saving detected face images to a dedicated directory with a corresponding name, and analyzing a face for attributes such as emotion. To make the images accessible across our system, we use file paths within the LLM context with regard to a dedicated directory.

For speech synthesis, the speak tool includes support for ElevenLabs[8] as the highest-quality TTS option, Groq's TTS API (Groq, Inc., 2024), and a local fallback to the pyttsx3 package if APIs are unavailable.

5.2 ROS Integration

At present, rosaOS integrates two ROS2-enabled robots: the Hiwonder PuppyPi[9], a small quadruped robot, and the TurtleBot4[10], a wheeled mobile robot, both shown in Figure 2. We run one instance of the ROS MCP Server[11] per robot, bridging ROS to the LLM kernel via the Rosbridge protocol (Mace et al., 2013), which enables JSON-based messaging over WebSockets without requiring a native ROS installation on the host machine. This architecture is consistent with efforts made for robot control via natural language (Mower et al., 2024).

ROS organizes robot software around five communication primitives: nodes (independent computational units), topics (named data streams facilitating asynchronous communication via a publisher-subscriber pattern), services (synchronous request-response mechanisms), actions (asynchronous goal-based tasks), and parameters (persistent configuration data). Each ROS MCP Server instance exposes the full interface of these primitives as MCP tools. Tools for topics allow agents to list all running topics, review their types and publisher/subscriber counts, subscribe to receive messages, and publish messages. Service tools expose service discovery, type introspection, and service calls. Action tools allow listing available action servers, sending a goal, and cancelling an in-progress goal. Parameter tools cover listing, getting, setting, and deleting node parameters.

Camera integration avoids passing binary data directly to a potentially text-only LLM context, as described previously. A tool for capturing images subscribes to a configurable ROS camera topic, decodes the incoming image, saves it to a shared images directory accessible across all MCP servers, and returns the relative file path. The agent then passes this path to the Reachy Mini's image analysis tool for LLM description.

5.3 Smart Light

To demonstrate the extensibility of rosaOS beyond specific robotic hardware, we integrate a TP-Link Tapo L530E smart light bulb[12] into the ecosystem. This device is managed by an MCP server utilizing the open-source python-kasa[13] library. This server exposes eight primary tools, allowing agents to manage device state (on/off/toggle), brightness, color temperature, and color values.

The inclusion of the lamp illustrates that any hardware with an API or SDK can be converted into an MCP device driver with minimal code, allowing the kernel to orchestrate complex smart home and robotic tasks simultaneously.

6 Demonstration

Demo videos are available on our website at rosaos.ai. To illustrate the capabilities of rosaOS, we walk through an example scenario that exercises multi-device coordination and parallel execution.

[8] elevenlabs.io
[9] hiwonder.com/products/puppypi
[10] turtlebot.com
[11] github.com/robotmcp/ros-mcp-server
[12] tp-link.com/ca/home-networking/smart-bulb/tapo-l530e/
[13] github.com/python-kasa/python-kasa

The example uses five MCP servers: Reachy Mini, TurtleBot4, Hiwonder PuppyPi, TP-Link Tapo Light, and Spotify. The Reachy Mini serves as the user interface; the user engages it using a wake phrase. We use Claude Opus 4.6 (Anthropic, 2026), one of the best agentic LLMs as of February 2026, and the ElevenLabs API for TTS.

The user wakes the Reachy Mini with "hello" and issues a multi-step request: undock the TurtleBot, search the environment for a person, and when someone is found, turn the TP-Link Tapo lamp green. Once instructed, the TurtleBot undocks and begins navigating and taking pictures until it detects a person, at which point the kernel processes the callback and dispatches a process agent to set the lamp color. This highlights that a single natural language request can expand into a multi-step plan spanning multiple devices.

While the TurtleBot is still searching, the user reengages the Reachy Mini and asks the PuppyPi to walk forward and play a song on Spotify. The new prompt results in a separate process agent scoped to the PuppyPi and Spotify MCP servers, executing concurrently with the first. This validates that rosaOS can accept and dispatch new user requests without blocking ongoing device operations.

All tasks complete successfully: the TurtleBot finds a person and the lamp turns green, while the PuppyPi walks forward and music begins playing, each managed by a process agent under the kernel.

7 Conclusion

We present rosaOS, an agentic operating system for embodied LLMs that applies OS design principles to multi-device coordination: a kernel agent manages task execution and hardware arbitration, process agents carry out device-specific work via MCP, and ROS provides a standard interface to heterogeneous robots. Our demonstration shows that this architecture supports natural language control of multiple physical devices concurrently, with new requests accepted and dispatched without blocking ongoing tasks. We release rosaOS as an open platform and hope it serves as a foundation for future research in embodied agentic systems.

8 Limitations

Our implementation is a research prototype and has several limitations. First, the system depends on external LLM and speech APIs, which introduce latency, cost, and potential reliability issues. As noted, self-hosted alternatives exist for every external dependency, but performance can vary across models and deployment environments. This tradeoff is an inherent property of LLM-based systems rather than a limitation specific to rosaOS.

Second, the kernel and process abstraction relies on LLM reasoning for tool use and task planning. As with all LLM-based agents, behavior is sensitive to the choice of model and prompting, and may exhibit hallucinations or incorrect tool invocation. While hardware access is scoped through MCP servers, we do not provide formal safety guarantees for physical actuation. Hardware-level safety is instead delegated to device-native mechanisms—for example, the Reachy Mini enforces built-in movement constraints through its SDK, and mobile platforms such as the TurtleBot4 carry no greater risk than a human-operated equivalent. Our platform's role is analogous to an operating system exposing a system call interface: it enforces *who* may invoke a command, but does not second-guess the physical outcome.

Third, the current work lacks systematic empirical evaluation. The demonstration establishes qualitative viability of multi-device coordination, but does not provide quantitative analysis of latency, task success rate, or robustness across different models or command types. We consider a structured evaluation benchmark for embodied agentic systems an important direction for future work.

Finally, subprocess-based process management introduces overhead and is not optimized for large-scale multi-robot deployments. The current implementation is therefore best suited for high-level orchestration rather than real-time control.

Acknowledgements

This work was supported in part by the Tang Family Foundation and the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

Anthropic. 2026. Claude Opus 4.6. anthropic.com/news/claude-opus-4-6.

Irvin Steve Cardenas, Marcus Anthony Arnett, Natalie Catherine Yeo, Lucky Shah, and Jong-Hoon Kim. 2026. ROSClaw: An OpenClaw ROS 2 Framework for Agentic Robot Control and Interaction. *arXiv preprint arXiv:2603.26997*.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan

Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, and 3 others. 2023. PaLM-E: An Embodied Multimodal Language Model. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 8469–8488. PMLR.

Groq, Inc. 2024. Groq API Documentation. `console.groq.com/docs`.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, and 26 others. 2023. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 287–318. PMLR.

Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. 2023. VIMA: General Robot Manipulation with Multimodal Prompts. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 14975–14995. PMLR.

Jonathan Mace, Russell Toris, and Jihoon Lee. 2013. rosbridge Suite. `wiki.ros.org/rosbridge_suite`.

Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. 2022. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66).

Kai Mei, Xi Zhu, Wujiang Xu, Wenyue Hua, Mingyu Jin, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. 2025. AIOS: LLM Agent Operating System. *In Proceedings of the 2nd Conference on Language Modeling (COLM 2025)*.

Meta AI. 2025. The Llama 4 Herd: The Beginning of a New Era of Natively Multimodal AI Innovation. `ai.meta.com/blog/llama-4-multimodal-intelligence/`.

Christopher E. Mower, Yuhui Wan, Hongzhan Yu, Antoine Grosnit, Jonas Gonzalez-Billandon, Matthieu Zimmer, Jinlong Wang, Xinyu Zhang, Yao Zhao, Anbang Zhai, Puze Liu, Daniel Palenicek, Davide Tateo, Cesar Cadena, Marco Hutter, Jan Peters, Guangjian Tian, Yuzheng Zhuang, Kun Shao, and 4 others. 2024. ROS-LLM: A ROS Framework for Embodied AI with Task Feedback and Structured Reasoning. *arXiv preprint arXiv:2406.19741*.

OpenAI. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*.

Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. 2009. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*. Workshop paper.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2023. Robust Speech Recognition via Large-Scale Weak Supervision. *arXiv preprint arXiv:2212.04356*.

Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. 2018. *Operating System Concepts*, 10 edition. John Wiley & Sons.

Sai Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. 2023. ChatGPT for Robotics: Design Principles and Model Abilities. *arXiv preprint arXiv:2306.17582*.

Rongfeng Zhao, Xuanhao Zhang, Zhaochen Guo, Xiang Shao, Zhongpan Zhu, Bin He, and Jie Chen. 2026. ROSClaw: A Hierarchical Semantic-Physical Framework for Heterogeneous Multi-Agent Collaboration. *arXiv preprint arXiv:2604.04664*.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, and 35 others. 2023. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 2165–2183. PMLR.