Skilled AI Agents for Embedded and IoT Systems Development

Yiming Li, Yuhan Cheng, Mingchen Ma, Yihang Zou, Ningyuan Yang, Wei Cheng,
Hai "Helen" Li, Yiran Chen, and Tingjun Chen
Department of Electrical and Computer Engineering, Duke University

Abstract

Large language models (LLMs) and agentic systems have shown promise for automated software development, but applying them to hardware-in-the-loop (HIL) embedded and Internet-of-Things (IoT) systems remains challenging due to the tight coupling between software logic and physical hardware behavior. Code that compiles successfully may still fail on real devices due to timing constraints, peripheral initialization requirements, or hardware-specific behaviors. We introduce a skills-based agentic framework for HIL embedded development together with IoT-SkillsBench, a benchmark that systematically evaluates AI agents in real embedded programming environments across three platforms, 23 peripherals, and 42 tasks at three difficulty levels. Across 378 hardware-validated experiments under three agent configurations (no-skills, LLM-generated skills, and human-expert skills), we show that concise human-expert skills enable near-perfect success rates across platforms.

1 Introduction

Recent advances in LLMs and agentic systems have shown promise in assisting software development through automated code generation [5, 13, 19], debugging [6], and reasoning over complex documentation [17]. This has motivated interest in applying such agents to hardware-in-the-loop (HIL) embedded and Internet-of-Things (IoT) development [4, 9, 11, 12, 15, 18], which introduces fundamentally different challenges than purely software-based settings.

First, *the embedded ecosystem is highly fragmented.* MCUs rely on diverse vendor-specific toolchains and RTOS environments (e.g., ESP-IDF, Zephyr, AVR-Libc), and peripheral interactions span many protocols (I2C, SPI, UART, wireless stacks). Interfacing with hardware requires strict adherence to low-level constraints such as timing, initialization ordering, interrupt handling, and pin multiplexing, where minor configuration mismatches can cause silent failures. Existing approaches mitigate this by augmenting agent memory [14] with library documentation [16] or resolving header/API conflicts via state flow analysis [4], but these substantially expand context windows and incur high token cost [2].

Second, *successful compilation and flashing do not guarantee correct runtime behavior.* Code that builds without errors may still misbehave due to timing violations, peripheral misconfigurations, or hardware edge cases that cannot be deterministically validated through software-only verification. Hardware emulators such as QEMU [10] and Wokwi [3] suffer from limited peripheral fidelity and incomplete timing models, and existing compiler-in-the-loop [15] or flashing-in-the-loop [16] approaches fail to capture behavioral errors during physical execution.

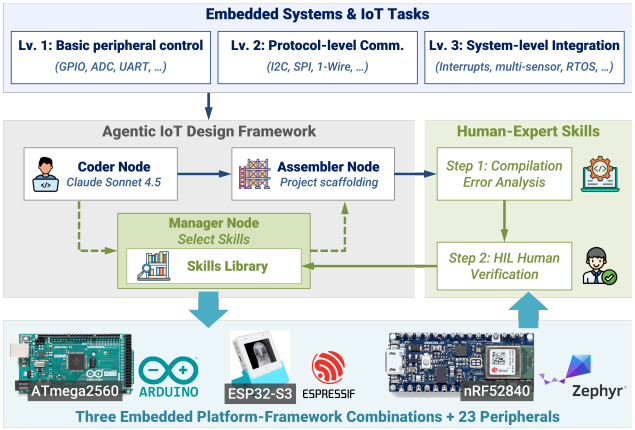

Figure 1. Overview of the skills-based agentic framework for embedded and IoT systems development and the IoT-SkillsBench benchmark.

To address these challenges, we propose a *skills-based agentic framework for embedded and IoT development* (Fig. 1). Rather than injecting large volumes of SDK documentation or datasheets into the prompt, each skill distills the essential programming patterns, initialization constraints, and known failure modes for a specific peripheral, MCU, or framework combination into a compact, human-readable document. This allows the agent to reason over a constrained, validated knowledge set, reducing token overhead while improving reliability. The skills-based design [8] also scales naturally: adding a new platform or peripheral requires only authoring the corresponding skills.

To evaluate the role of structured hardware knowledge in agentic embedded development, we introduce *IoT-SkillsBench*, a HIL benchmark spanning three platform-framework combinations (ATmega2560 + Arduino, ESP32-S3 + ESP-IDF, nRF52840 + Zephyr) and 42 tasks across three difficulty levels: basic peripheral control, protocol-level communication, and system-level integration. Each task is evaluated under three agent configurations (no-skills, LLM-generated skills,

human-expert skills), yielding 378 hardware-validated evaluation instances.

Experiments on IoT-SkillsBench reveal three key insights. First, modern LLMs perform well on simple tasks for well-documented platforms like Arduino, but performance degrades rapidly with protocol-level reasoning or system integration. Second, LLM-generated skills provide inconsistent benefits and can degrade performance while substantially increasing token usage. Third, concise human-expert skills dramatically improve reliability, achieving near-perfect success rates with moderate token overhead. IoT-SkillsBench is open-source.

2 IoT-SkillsBench: Platforms, Tasks, and Skills

Platforms and peripherals. We consider three platform-framework combinations covering diverse MCU architectures (AVR, Xtensa, ARM), HAL paradigms, and ecosystems: **ATmega2560** (Arduino Mega 2560 Rev3) with the Arduino framework (CLI v1.4.1, arduino:avr v1.8.7); **ESP32-S3** (ESP32-S3-BOX-3) with ESP-IDF (v5.1.2); and **nRF52840** (Arduino Nano 33 BLE Rev2) with Zephyr via nRF Connect SDK (v2.7.0). Beyond Arduino [16], our framework also supports ESP-IDF and Zephyr, two industrial frameworks powering many commercial IoT devices. We further consider 23 peripherals spanning simple actuators and indicators (LEDs, buzzers, relays, buttons) to sensors using analog output and digital communication buses.

Tasks. The benchmark contains 42 tasks across three difficulty levels. *Level 1* covers basic peripheral control (GPIO toggling, ADC sampling, UART communication). *Level 2* requires protocol-level reasoning over I2C, SPI, and 1-Wire, including correct initialization sequences and timing. *Level 3* demands system-level integration such as interrupt-driven workflows, multi-sensor/actuator integration, and task scheduling. Each task is defined by (*i*) a natural language description optionally specifying peripherals and required logic (e.g., debouncing), and (*ii*) a target platform with required pin assignments. Pin referencing conventions differ across platforms (e.g., ESP-IDF uses numeric GPIO numbers; Zephyr uses device tree aliases), so platform-specific pin descriptors are provided. Each task was manually validated on hardware, and reference implementations were verified to produce correct HIL test outcomes.

Skills specifications. We construct two skills sets. The first is generated by prompting Claude Sonnet 4.6 with each task and platform-framework combination, asking it to summarize the relevant embedded knowledge from parametric memory only, without datasheets or external documentation. The second is authored by embedded systems experts with hands-on experience on the target platforms, with the goal of producing skills that are *compact, targeted, and directly actionable for the agent.* During authoring, experts are given the LLM-generated code (without skills), corresponding compiler error logs, and observed runtime behaviors, providing a grounded view of typical failure modes. Experts are instructed to write concisely, favoring descriptive prose over code snippets, and to keep each skill focused on a single peripheral or framework-specific concern. Both skills sets follow a unified format inspired by the Agent Skills specification [1]: a YAML header with metadata (name, target platform, peripherals) followed by a Markdown body, allowing the agent to scan headers during retrieval without loading full bodies.

3 Agent Setup and Evaluation Protocol

We implement the agent using LangGraph [7] with a three-node architecture (*manager*, *coder*, *assembler*). We intentionally adopt this minimal structure to isolate the effect of skills from other agentic complexities such as multi-step planning, tool use, or reflective loops, which could otherwise confound the evaluation.

When no skills are provided, the manager node is skipped and the prompt goes directly to the coder. With skills enabled, the manager receives the list of available skill headers and task requirements, selects relevant skills, and forwards them to the coder along with the task prompt. The coder, prompted as an embedded engineer, generates the firmware file (`*.c` for ESP-IDF/Zephyr; `*.ino` for Arduino). The assembler formats the raw output into a ready-to-compile project, generating platform-specific scaffolding (e.g., `CMakeLists.txt`, Zephyr device tree overlays).

Evaluation protocol. We use Claude Sonnet 4.5 as the backbone model, executing each task-skills-platform combination five times. All projects are compiled with identical toolchain commands and configurations (§2), then flashed and tested on physical hardware, with results validated by a human evaluator. *Due to the HIL nature of the benchmark, the full evaluation required approximately 100 hours of human-in-the-loop validation.*

Performance metrics. Each task attempt yields one of three outcomes: **Compile Failure (CF)**, the firmware fails to compile; **Behavior Failure (BF)**, it compiles and flashes but fails the hardware test (including hangs and watchdog resets); or **Behavior Correct (BC)**, it produces the correct observable behavior. We additionally report **pass@1** (fraction of tasks reaching BC on the first attempt) and **pass@5** (at least one of five attempts is BC), capturing both efficiency and ceiling performance. **Token usage** per LLM call is recorded from response metadata and reported per node.

4 Experimental Results

Table 1 and Figs. 2–3 report performance and token usage. The no-skills baseline resolves simple Level 1 tasks but struggles as difficulty grows, especially on Zephyr, likely due to scarcer Zephyr-specific material in pretraining data;

Table 1. Detailed CF/BF/BC@5 outcomes across platforms, skills configurations, and task difficulty levels.

Skills	Platform	L1 (12)	L2 (16)	L3 (14)	Total (42)
No-Skills	Arduino	0/0/12/12	0/0/16/16	0/0/14/14	0/0/42/42
	ESP-IDF	0/0/12/12	2/2/12/16	1/6/7/14	3/8/31/42
	Zephyr	0/1/11/12	0/5/11/16	0/8/6/14	0/14/28/42
LLM	Arduino	0/0/12/12	0/0/16/16	1/0/13/14	1/0/41/42
	ESP-IDF	0/0/12/12	1/4/11/16	0/10/4/14	1/14/27/42
	Zephyr	0/2/10/12	1/6/9/16	0/6/8/14	1/14/27/42
Human-Expert	Arduino	0/0/12/12	0/0/16/16	0/0/14/14	0/0/42/42
	ESP-IDF	0/0/12/12	0/0/16/16	0/1/13/14	0/1/41/42
	Zephyr	0/0/12/12	0/1/15/16	0/0/14/14	0/1/41/42

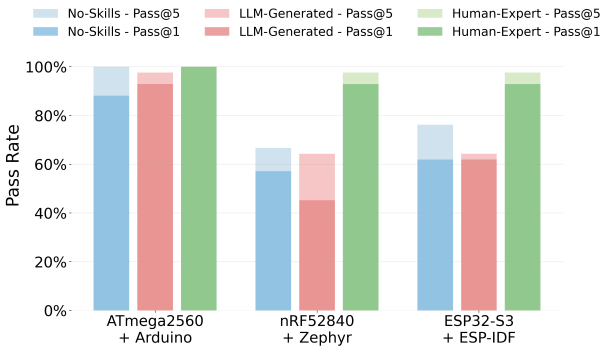

Figure 2. Pass rates (pass@1 and pass@5) across platforms, skills configurations, and task difficulty levels.

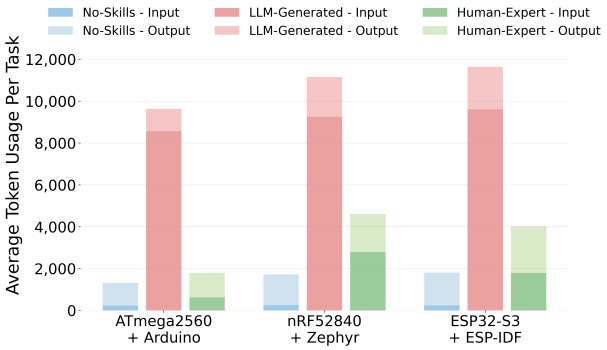

Figure 3. Average per-task token usage across platforms, skills configurations, and task difficulty levels.

for Level 3 the agent solves only 7/14 (ESP-IDF) and 6/14 (Zephyr). Token cost is minimal (~300 input, ~1,200 output per task).

LLM-generated skills produce mixed results. Arduino shows minor degradation (41/42 vs. 42/42), suggesting little additional value over the model's existing knowledge for well-documented platforms. ESP-IDF drops more noticeably (27/42 vs. 31/42), with Level 3 falling from 7/14 to 4/14, indicating that LLM-synthesized skills can reinforce incorrect platform-specific assumptions. Zephyr shows a slight overall drop (27/42 vs. 28/42) but a modest Level 3 gain (8/14 vs. 6/14). LLM-generated skills also incur the highest token cost (8,500–9,500 input, 1,500–2,000 output per task); output tokens stay elevated despite code-only instructions because the model often reasons aloud about the skill before producing code.

Human-expert skills achieve the best overall performance: 42/42 on Arduino and 41/42 on both ESP-IDF and Zephyr. The two unresolved tasks reflect fundamental hardware ambiguities rather than agent limitations: a 5V RTC module that destabilizes the 3.3V ESP32-S3 (a hardware-level voltage incompatibility unsolvable in firmware), and a rotary encoder whose CW/CCW direction is not standardized across models, making a universally correct prompt-only solution impossible. Token consumption remains moderate (650–2,900 input, 1,700–4,600 output per task), striking a better performance/-efficiency balance than LLM-generated skills.

5 Conclusions

We presented IoT-SkillsBench, a hardware-validated benchmark for evaluating agentic systems in embedded and IoT development, together with a skills-based agent framework integrating structured procedural knowledge into the code generation pipeline. HIL evaluations across multiple MCU platforms and difficulty levels show that raw LLM capabilities alone are insufficient for reliable embedded development, while compact human-expert skills significantly improve reliability with moderate token cost. Grounded domain knowledge—not merely larger models—is essential for agentic programming in hardware-constrained environments. We hope IoT-SkillsBench will serve as a foundation for future research on AI-assisted embedded development, HIL agent evaluation, and scalable knowledge representations for physical-system programming.

Acknowledgments

This work was supported in part by NSF grant CNS-2112562.

A Example Tasks Across Difficulty Levels

Below, we provide example tasks in IoT-SkillsBench across three task difficulty levels.

A.1 Level 1: Basic Peripheral Control

Task: Button Status with Debouncing

```
Read the state of a pull-down button and implement
software debouncing to avoid multiple triggers.
Print "Button Pressed!" to the serial console when
the button is pressed.
```

Task: TMP36

```
Read the analog output of the TMP36 temperature
sensor, where the output voltage is linearly
proportional to the temperature in degrees Celsius,
and print the temperature value to the serial
console.
```

A.2 Level 2: Protocol-Level Communication

Task: MPU6050 Data Reading (I2C)

Read raw accelerometer and gyroscope data from the MPU6050 via I2C and print to the serial console.

Task: BME280 Data Reading (SPI)

Read the humidity and temperature from a BME280 sensor using the SPI bus, and print the humidity and temperature values to the serial console.

A.3 Level 3: System-Level Integration

Task: Safe Box with Display

Write the program that will read the password input from the 16 key keypad, there are in total 8 lines connected to each 4 rows and 4 cols. In particular, the password will be set to "1234". The program will read the key input from the 16 key keypad, and if the input matches the password, the program will connect the relay to unlock the safebox.

Task: Variable Frequency LED with Buzzer

Build the program that integrates the button press with both the buzzer and the timer-based LED control. After the system reset, when you press the button:
- For the 1st time, a timer is triggered that toggles the external LED at 1 Hz;
- For the 2nd time, a timer is triggered that toggles the external LED at 2 Hz;
- For the 3rd time, a timer is triggered that toggles the external LED at 4 Hz;
- For the 4th time, the timer is stopped and the external LED will not blink;

The process repeats and the toggling frequency of the LED will undergo the sequence of 1 Hz, 2 Hz, 4 Hz, N/A, 1 Hz, 2 Hz, 4 Hz, N/A, . . . as you press the button. In addition, every time the button is pressed, the buzzer will go off, indicating that the button has been pressed. The button and buzzer must be connected to separate GPIO pins, and the buzzer operation will be triggered by its own connected GPIO pin.

B Prompt for LLM-based Skills Generation

The following prompt is used to generate the LLM-generated skills:

Prompt: LLM-based Skills Generation

Read tasks in tasks/level<X>.txt, please follow these steps:

1. Analyze the task requirements and identify what domain knowledge, hardware platforms, or connectivity protocols, etc, are needed.

2. Consider implementation based on three MCU + development framework combinations: 1) ESP32 + ESP-IDF, 2) ATMega2560 + Arduino, 3) nRF52840 + Zephyr.

3. Write modular skill documents that would help solve these tasks. Each skill should: focus on a specific MCU, board, IoT tool, protocol, framework, or embedded technique; provide code examples and usage patterns; be reusable for similar tasks.

4. Save each skill as a markdown file in the skills-llm-generated/ directory with a descriptive name.

C Human-Expert Skills Examples

Below, we provide example human-expert skills in IoT-SkillsBench for GPIO best practices.

Zephyr Skills: GPIO Best Practices

- Define pin polarity (GPIO_ACTIVE_HIGH/GPIO_ACTIVE_LOW) in the devicetree, not in code.
- Use gpio_pin_set(dev, pin, 1), logical ON to turn ON; gpio_pin_set(dev, pin, 0), logical OFF to turn OFF. Let Zephyr's GPIO driver handle the physical HIGH/LOW translation based on DT flags. Never assume physical voltage level.
- Use GPIO_INT_EDGE_TO_ACTIVE / GPIO_INT_EDGE_TO_INACTIVE instead of RISING/FALLING — stays polarity-agnostic.

...

ESP-IDF Skills: GPIO Best Practices

- **Prevent Floating States:** Explicitly configure internal resistors (.pull_up_en, .pull_down_en). If a pin must default to LOW/HIGH, use gpio_set_level() immediately after gpio_config() to guarantee a deterministic state before peripheral power-on delays.
- **Interrupts for Transients:** For catching fast, transient state changes (e.g., <5ms pulses), avoid while(1) polling. Prefer Interrupt Service Routines via gpio_install_isr_service() and configure edge-triggered interrupts (e.g., GPIO_INTR_NEGEDGE).

...

References

[1] 2025. Agent Skills are a simple, open format for giving agents new capabilities and expertise. https://github.com/agentskills/agentskills.

[2] Lingjiao Chen, Matei Zaharia, and James Zou. 2023. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176* (2023).

[3] CodeMagic LTD. 2026. Wokwi Simulator. https://wokwi.com/.

[4] Kaijie Gong, Wei Dong, Hao Wang, Yingqi Peng, and Yi Gao. 2025. Programming Embedded IoT Applications in Natural Language with IoTPilot. In *Proc. ACM MobiSys'25*.

[5] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974* (2024).

[6] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. SWE-Bench: Can Language Models Resolve Real-World GitHub Issues? *arXiv preprint arXiv:2310.06770* (2023).

[7] LangChain Inc. 2026. LangGraph: Low-Level Orchestration Framework for Building, Managing, and Deploying Long-Running, Stateful Agents. https://github.com/langchain-ai/langgraph.

[8] Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, et al. 2026. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. *arXiv preprint arXiv:2602.12670* (2026).

[9] 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.

[10] QEMU Project. 2026. QEMU: A Generic and Open Source Machine Emulator and Virtualizer. https://www.qemu.org/.

[11] Leming Shen, Qiang Yang, Xinyu Huang, Zijing Ma, and Yuanqing Zheng. 2025. GPIoT: Tailoring Small Language Models for IoT Program Synthesis and Development. In *Proc. ACM SenSys'25*.

[12] Leming Shen, Qiang Yang, Yuanqing Zheng, and Mo Li. 2025. AutoIoT: LLM-Driven Automated Natural Language Programming for AIoT Applications. In *Proc. ACM MobiCom'25*.

[13] Yang Song, Anoushka Vyas, Zirui Wei, Sina Khoshfetrat Pakazad, Henrik Ohlsson, and Graham Neubig. 2026. NEMO: Execution-Aware Optimization Modeling via Autonomous Coding Agents. *arXiv preprint arXiv:2601.21372* (2026).

[14] Nandan Thakur, Jimmy Lin, Sam Havens, Michael Carbin, Omar Khattab, and Andrew Drozdov. 2025. Freshstack: Building realistic benchmarks for evaluating retrieval on technical documents. *arXiv preprint arXiv:2504.13128* (2025).

[15] Ruiyang Xu, Jialun Cao, Mingyuan Wu, Wenliang Zhong, Yaojie Lu, Ben He, Xianpei Han, Shing-Chi Cheung, and Le Sun. 2025. EmbedAgent: Benchmarking Large Language Models in Embedded System Development. *arXiv preprint arXiv:2506.11003* (2025).

[16] Huanqi Yang, Mingzhe Li, Mingda Han, Zhenjiang Li, and Weitao Xu. 2024. EmbedGenius: Towards Automated Software Development for Generic Embedded IoT Systems. *arXiv preprint arXiv:2412.09058* (2024).

[17] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-Agent: Agent-Computer Interfaces Enable Automated Software Engineering. *Adv. Neural Inf. Process. Syst.* 37 (2024), 50528–50652.

[18] Ningyuan Yang, Guanliang Lyu, Mingchen Ma, Yiyi Lu, Yiming Li, Zhihui Gao, Hancheng Ye, Jianyi Zhang, Tingjun Chen, and Yiran Chen. 2025. IoT-MCP: Bridging LLMs and IoT Systems Through Model Context Protocol. In *Proceedings of the ACM Workshop on Wireless Network Testbeds, Experimental evaluation & Characterization*. 73–80.

[19] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. 2024. BigCodeBench: Benchmarking Code Generation with Diverse Function Calls and Complex Instructions. *arXiv preprint arXiv:2406.15877* (2024).