Embedded Arena: Iterative Optimization via Hardware Feedback

Zhihan Zhang[W*] **Alexander Le Metzger**[W*] **Jiuyang Lyu**[W*] **Chun-Cheng Chang**[W] **Jiayi Shao**[W] **Yujia Liu**[✓] **Emmanuel Azuh Mensah**[W] **Edward Wang**[✓] **Kurtis Heimerl**[W] **Gregory D. Abowd**[N] **Shwetak Patel**[W] **Natasha Jaques**[W] **Vikram Iyer**[W]

[W]University of Washington [✓]University of California San Diego [N]Northeastern University
[*]Equal contribution

NeurIPS 2026 Spotlight

Abstract

As AI moves into the physical world, from robots to environmental sensors to medical wearables, it increasingly relies on local inference to meet latency, communication, or privacy constraints. Optimizing models for heterogeneous edge hardware such as microcontrollers (MCUs) requires simultaneously satisfying hard physical constraints on memory, power, and temperature while preserving accuracy, a multidimensional optimization that is today performed manually by experts. We ask whether an LLM agent can autonomously navigate this complex, multi-turn pipeline guided by real hardware feedback, and introduce a hardware-in-the-loop agent arena in which the agent iteratively refines both model and firmware by compiling, flashing, and measuring on real hardware, enabling closed-loop optimization. Frontier models, including Claude Opus 4.7 and Gemini 3.1 Pro, fail entirely without hardware feedback (0% deployment success), whereas our hardware-in-the-loop formulation achieves the first successful deployment within three iterations and can surpass human expert results within seven. This agentic co-optimization achieves 250× compression for vision models with <3.3% accuracy loss and 400× for audio with <6% Feature Error Rate loss, enabling battery-free operation on a commercial MCU via solar harvesting. We demonstrate practical impact in two real-world systems: an elk-detection camera trap (96.7% accuracy) and a phonetic-transcription wearable (8.37% FER) for child development research.[1]

1 Introduction

Advances in large, cloud-based AI models have unlocked remarkable capabilities across vision, language, and multimodal reasoning, but leave a gap for applications with stringent latency, privacy, or connectivity requirements. Remote wildlife cameras must detect and classify species of interest without reliable network access [13, 14], mobile devices may process audio and biosignals locally to avoid the privacy risks and power costs of cloud streaming [1, 6, 28], and physical AI systems with safety-critical perception-action cycles cannot tolerate round-trip latency to the cloud. Such applications demand small models that run directly on microcontroller (MCU) class hardware. These devices have many orders of magnitude lower clock speeds, memory, and storage than datacenter GPUs, requiring substantial model compression and optimization to utilize the growing hardware acceleration capabilities being built into MCUs [10].

Deploying AI in this heterogeneous, rapidly evolving hardware landscape requires multidimensional, system-level optimization that satisfies hard physical constraints (e.g., memory footprint, power

[1] https://ubicomplab.github.io/embedded-arena/

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

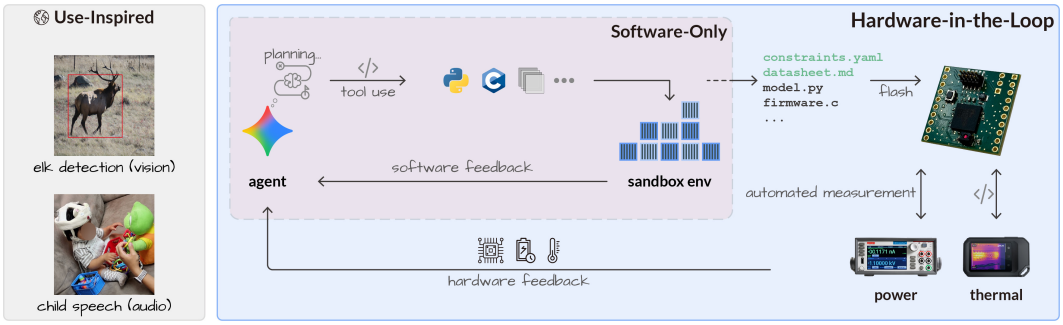


Figure 1: AI agents iteratively optimize models to run on MCUs within their tight power, thermal, and compute constraints using hardware-in-the-loop feedback. We demonstrate successful model optimization for real-world applications, including wildlife detection and child speech monitoring.

consumption, thermal limits) while maximizing model accuracy. On the hardware side, this landscape is a moving target: the STM32F7 [31] MCU series alone ships in 22 silicon variants, each with distinct memory configurations and AI accelerators. On the model side, practitioners must navigate compression techniques (e.g., quantization, operator substitution), each of which interacts differently with the target chip's processing capabilities. These optimization choices are coupled and competing. For example, for object detection on MCUs, reducing camera resolution saves activation memory and power draw but directly degrades accuracy [4]. Today, navigating these trade-offs is performed manually by experts who hold rare, joint expertise in embedded systems and machine learning (ML). Skilled practitioners using deployment frameworks such as LiteRT [17] spend weeks optimizing a single model on real hardware for a single deployment—a process that must be repeated from scratch for every new chip [31]. In this paper, we ask: *can AI agents optimize AI to run on real hardware by iterating directly on physical measurements?*

To answer this question, we construct a **hardware-in-the-loop (HIL)** environment in which the agent iteratively produces, compiles, flashes, and refines firmware subject to physical feedback. We observe that even state-of-the-art models like Claude Opus 4.7 fail entirely in this setting when restricted to software-only feedback. Closing the loop with physical measurements, such as power and temperature directly from the device, guides the agent toward firmware that actually works on the device. HIL raises deployment success rates from 0% to 80% on the hardest tasks. We formalize this loop as a continuous optimization problem by reducing physical measurements to a programmatically scorable scalar objective, enabling iterative refinement. This transforms an opaque, expert-gated process into an accessible workflow, letting practitioners without joint hardware-AI expertise deploy state-of-the-art models on resource-constrained devices.

Our HIL formulation also addresses a gap in existing agentic systems and LLM benchmarks. While LLM-based agents have demonstrated impressive autonomy in complex workflows [50, 27, 26, 47, 37, 2, 18, 9], benchmarks for evaluating them on long-horizon, system-level optimization remain rare. Most (e.g., SWE-bench [24], TerminalBench [33], HumanEval [7]) focus entirely on the digital domain, evaluating agents on isolated, human-curated single-turn tasks rather than end-to-end system optimization. Because these tasks are scored on binary correctness, they lack a mechanism for continuous self-improvement and saturate quickly as models advance, requiring periodic manual updates. A smaller family of Kaggle-style competitions [2] poses open-ended optimization scored by continuous software metrics (e.g., held-out accuracy) to support iterative hill-climbing. Our HIL arena inherits this open-ended optimization paradigm but extends it to performance metrics measured directly from the physical world at test time, where results vary with ambient conditions and component state, forcing agents to reason about real hardware behavior.

We summarize our contributions as follows:

1. We are the first to formalize the deployment of AI on resource-constrained devices as a hardware-in-the-loop agent arena for iterative optimization, and physically build the end-to-end evaluation infrastructure.

2. We perform a comprehensive study of how context and hardware feedback shape the artifacts produced by LLM agents. HIL significantly improves deployment success over both software-only

baselines, achieves the first successful deployment within three iterations, and **can surpass human expert results within seven**.

3. We achieve 250× compression for vision models with <3.3% accuracy loss, and 400× for audio with <6% Feature Error Rate (FER), also reducing power draw by over 58% and peak device temperature rise by over 40%.
4. We demonstrate practical impact in two real-world deployments developed in direct collaboration with domain practitioners: a solar-powered camera trap for elk detection achieving 96.7% accuracy under 7 mW average power, and a children's head-worn device with 8.37% on-device FER for phonetic transcription.

2 Hardware-in-the-Loop Arena

We present a hardware-in-the-loop arena in which an LLM agent must generate, compile, flash, and iteratively refine software subject to *physical* feedback signals returned by real hardware. Unlike software-only benchmarks, our arena closes the loop with the physical world: rewards are not computed by scoring functions, but are *measured* by instruments attached to the device under test. We now formalize this as a constrained optimization problem over real hardware, then instantiate it across three physically distinct axes: model compression, power minimization, and thermal management. The formalism we develop in §2.1 is general to any constrained hardware target.

2.1 Problem Formalization

Hardware constraint vector. Let $\mathcal{H}$ denote a target hardware platform characterized by a *constraint vector*

$$\mathbf{h} = (h_1, h_2, \ldots, h_K) \in \mathbb{R}^K, \tag{1}$$

where each component h_k encodes a physical resource limit (e.g., flash storage in bytes, static RAM in bytes, peak supply current in amperes, maximum junction temperature in kelvin, supported operators). A solution is *deployable* only if all K constraints are simultaneously satisfied. We note that $\mathbf{h}$ can be automatically identified from datasheets via web search and verified using a combination of the compiler toolchain.

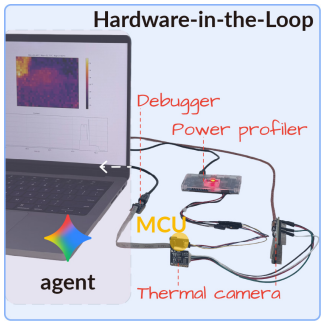


Figure 2: Hardware setup.

Solution space. A central design choice of our arena is that the solution space $\mathcal{S}$ is entirely open-ended. The agent produces a software artifact $s \in \mathcal{S}$, which may be firmware source code, a serialized neural network, a linker configuration, or any combination thereof. The arena is *agent-agnostic* and imposes no constraints on how s is constructed: the agent may employ any architecture or strategy (e.g., retrieval-augmented generation, multi-agent architectures [27]) and is judged solely on deployability and physical performance on the target hardware.

Given s and $\mathbf{h}$, the build-and-flash operator $\Phi : \mathcal{S} \times \mathbb{R}^K \to \{0, 1\}$ returns 1 if s compiles, boots successfully, and executes the required workload correctly on the device, and 0 otherwise. When $\Phi(s, \mathbf{h}) = 1$, a physical measurement $\mathcal{M} : \mathcal{S} \times \mathbb{R}^K \to \mathbb{R}^M$ returns a vector of M empirically observed quantities $\mathbf{m} = \mathcal{M}(s, \mathbf{h})$ (e.g., inference accuracy, peak current draw, peak core temperature).

Optimization objective. Each benchmark task is defined by a tuple $\mathcal{T} = (\mathbf{h}, f, \mathbf{c})$, where $f : \mathbb{R}^M \to \mathbb{R}$ is a task-specific objective to optimize, and $\mathbf{c} \in \mathbb{R}^C$ denotes task-level feasibility constraints that complement the hard physical limits in $\mathbf{h}$ and must be satisfied for s to be valid (e.g., a minimum accuracy floor). The agent's goal is to find a deployable artifact $s^\star$ that minimizes the task objective:

$$s^\star = \arg\min_{s \in \mathcal{S}} \; f(\mathcal{M}(s, \mathbf{h})) \quad \text{s.t.} \quad \Phi(s, \mathbf{h}) = 1. \tag{2}$$

Because each exploration round consumes real resources (e.g., LLM inference, compilation, physical hardware measurements), the number of iterations n before reaching a solution is penalized as a separate metric.

Iterative agent loop. The benchmark is structured as a multi-turn interaction $\{s_0\} \cup \{(s_t,\, o_t)\}_{t=1}^{N_{\max}}$ (Figure 1). At $t = 0$, the agent is provided an initial artifact s_0 (e.g., a `model.py` specifying a pre-trained model architecture, or a `firmware.c` implementing a reference inference pipeline) together with the hardware constraint vector $\mathbf{h}$ and the task objective f. The initial artifact s_0 serves as a starting point for optimization and need not satisfy the hardware constraints outright (e.g., $\Phi(s_0, \mathbf{h}) = 0$ when a pre-trained model exceeds the target flash budget).

At each subsequent step $t \geq 1$, the agent submits a modified artifact s_t; the benchmark executes $\Phi(s_t, \mathbf{h})$ and, if $\Phi = 1$, runs $\mathcal{M}$, then returns the full observation $o_t = (\Phi(s_t, \mathbf{h}),\, \mathbf{m}_t)$ to the agent as a structured natural-language summary (e.g., current draw, measured chip temperature). If $\Phi(s_t, \mathbf{h}) = 0$, the agent receives only the failure diagnostic without physical measurements.

The agent is allotted a budget of $N_{\max}$ attempts.

Agent memory and context. At each round t, the agent's artifact s_t is written to a dedicated file (e.g., `iteration_t.c`), preserving the full artifact history on disk. The observation o_t is appended as a structured natural-language summary directly to the agent's context window (see Appendix F). The agent is additionally equipped with tools to read and edit local files. The agent may query all prior artifacts $\{s_0, \ldots, s_{t-1}\}$ on demand to inform s_t. We preserve the full failure trajectory rather than maintaining only the current best artifact via version-control rollback [26], so the agent has the *option* to understand past failures when proposing s_t.

Benchmark score. To enable fair comparison across agents, we define the benchmark score $\mathcal{B}$ for a task as

$$\mathcal{B} = f(\mathcal{M}(s^\star, \mathbf{h}))\,, \tag{3}$$

where $s^\star$ is the best deployable solution found by the agent within $N_{\max}$ rounds, and f is defined so that lower $\mathcal{B}$ is always better (maximization objectives are negated). If no deployable solution satisfying $\mathbf{c}$ is found within $N_{\max}$ rounds, the agent receives $\mathcal{B} = \infty$.

When two agents achieve equal $\mathcal{B}$, we break ties by the total number of attempts $n \leq N_{\max}$, including attempts where $\Phi(s_t, \mathbf{h}) = 0$ (i.e., the artifact fails to compile or boot). Lower n is better.

2.2 Task Instantiations on MCUs

MCUs represent a particularly demanding instantiation of the framework in §2.1: with typically 128 KB–512 KB of SRAM, 512 KB–4 MB of flash, computing throughput two to three orders of magnitude below the tens of TOPS available on a smartphone SoC, and power budgets measured in milliwatts, they stress every dimension of the hardware constraint vector $\mathbf{h}$ simultaneously.

We instantiate three tasks along distinct axes of $\mathbf{h}$: model, power, and thermal (see Appendix for full objective derivations and evaluation variants). All three physical dimensions can, in principle, be incorporated jointly into a single f; we decompose them into separate tasks to enable clean, interpretable evaluation of each. The tasks are defined solely in terms of $(\mathbf{h}, f)$ and the feedback $\mathbf{m}_t$, and therefore can be extended to any hardware target for which Φ and $\mathcal{M}$ can be automated; our experiments span multiple MCU families (§3).

2.2.1 Task 1: Model Compression

Deploying a neural network on an MCU requires satisfying all $\mathbf{h}$: flash (model weights), SRAM (activations), and MCU-specific constraints such as supported operators and quantization formats. Given a pre-trained model and calibration dataset s_0 and evaluation dataset $\mathcal{D}$, the agent produces a compressed artifact s derived from s_0 (e.g., via quantization or distillation) that maximizes on-device inference accuracy $m_{\text{acc}} = \mathcal{M}_{\text{acc}}(s, \mathcal{D}) \in [0, 1]$ subject to $\Phi(s, \mathbf{h}) = 1$. When two agents achieve equal $\mathcal{B} = f_{\text{comp}}$, ties are broken by the number of attempts n, rewarding sample-efficient optimization.

2.2.2 Task 2: Power Minimization

Excessive *peak current* I_{peak} can trigger brownouts due to limits on battery discharge rate or permanent hardware damage even when the time-averaged current is well within budget. For battery-operated devices, *total energy per event* E determines operational lifetime on a fixed battery capacity. The

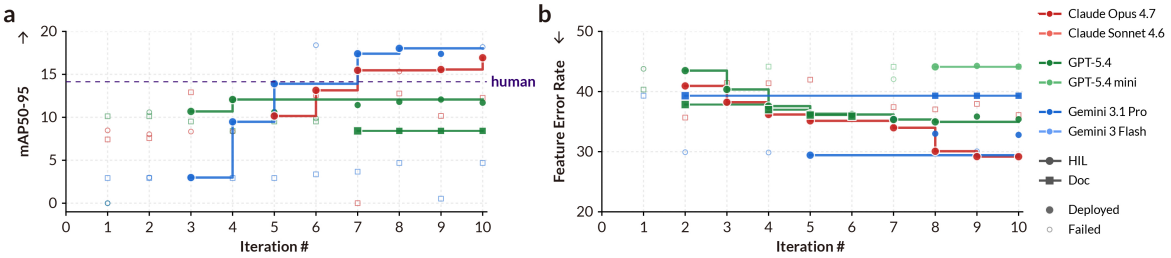


Figure 3: **HIL Enables Iterative Improvement in Model Compression.** For both YOLO on MAX78000 (a) and Wav2Vec2 on STM32N6 (b), agents hill-climb steadily once a valid s_t is established, and on (a) surpass the best human expert model (14.15% mAP50-95) by iteration 7. Failed attempts that violated hardware constraints show host-side scores of models.

agent minimizes one or both quantities—instantiated as separate variants targeting $f_{\text{peak}}(\mathbf{m}) = I_{\text{peak}}$ and $f_{\text{energy}}(\mathbf{m}) = E$—subject to a minimum accuracy floor $m_{\text{acc}} \geq m_{\text{acc}}^0$ (see Appendix D).

2.2.3 Task 3: Thermal Management

Device temperature governs hardware reliability and, critically for wearable applications, user safety: skin-contact temperatures above approximately 317 K (44 °C) can cause low-temperature burns with prolonged contact [36, 44].

Let T_{amb} be the measured ambient temperature and $T_{\text{dev}}(t)$ (K) the device surface temperature over the elapsed time $[0, \tau]$. The agent minimizes peak temperature rise $\Delta T = T_{\text{peak}} - T_{\text{amb}}$ under a controlled ambient load, with $T_{\max}$ as a hard safety ceiling in $\mathbf{h}$. Because ΔT depends on T_{amb}, we evaluate each device across multiple controlled ambient conditions, from room temperature to a surface approximating skin contact; details are in Appendix E.

3 Experiments

To disentangle the contribution of physical feedback from LLMs' parametric knowledge of embedded development, we evaluate three MCU targets from different vendors (ESP32-S3, MAX78000, STM32N6), each with its own SDK and deployment toolchain. We evaluate each target under three information settings:

1. **Score (Sco)**: a baseline where the agent receives the task description and initial artifact s_0, with access to a sandboxed environment supporting file manipulation and code execution. At each turn it observes only the binary feasibility check $\Phi(s_t, \mathbf{h})$ and the scalar optimization objective value $o_t = f(\mathcal{M}(s_t, \mathbf{h}))$.
2. **Documentation (Doc)**: the agent is additionally provided expert-curated device datasheets and toolchain documentation describing target constraints (e.g., supported operators, memory limits, required configuration formats) as part of s_0 and, except for model compression, may use web search to look up further details. This baseline is motivated by our finding that without such materials, frontier LLMs fail even basic MCU deployment questions (Appendix B).
3. Our **Hardware-in-the-loop (HIL)**: extends Doc with physical feedback o_t after each iteration, including resource utilization, serial logs, and deployment statistics.

We evaluate six frontier LLMs (GPT-5.4, GPT-5.4 mini, Claude Opus 4.7, Claude Sonnet 4.6, Gemini 3.1 Pro, and Gemini 3 Flash), each at *low* and *high* reasoning effort, and cap each run at $N_{\max} = 10$ iterations with up to 30 tool calls per iteration. Across all (model, task variant) pairs ($N = 48$), HIL achieves a mean success rate of 37.9%, significantly higher than Sco (12.3%, $p < 0.0001$) and Doc (18.5%, $p = 0.0004$) under paired Wilcoxon signed-rank tests with Holm correction.

3.1 Model Compression

Model compression evaluates whether agents can reduce a model's footprint to meet MCU hardware constraints while preserving task performance. We instantiate this task across vision and audio to

demonstrate generalizability, on two platforms with diverse constraints: MAX78000 (max 442 KB of model weights; very limited operator support) and STM32N6 (4.2 MB of live model weights; more flexible operator support). After the agent produces a compressed artifact s_t, the binary check $\Phi(s_t, \mathbf{h})$ confirms (i) the compressed model is trainable (i.e., we can perform backpropagation on the weights and improve the loss during training) and, (ii) can be compiled and run on the respective hardware with the code and network sidecars the agent provides. The score f is the test-set performance of the trained compressed model, i.e., $w_r = 0$ on the target MCU.

Example 1: Object Detection on MAX78000. The agent receives a pretrained YOLO11 checkpoint [25] and the COCO image segmentation [42] training set as s_0, and maximizes the mAP50-95 (f) on the held-out test set of COCO ($\mathcal{D}$). As shown in Table 7a, detailed HIL feedback is necessary: without it, only GPT-5.4 succeeds, and only in 40% of attempts when documentation is provided—far below its 70% success rate under HIL. Table 1 underscores the impact of HIL feedback: Gemini 3.1 Pro reaches a top mAP50-95 of $\mathcal{B} = 18.03\%$. Notably, both Claude Opus 4.7 and Gemini 3.1 Pro beat human experts at 14.15%. HIL also enables more successful deployments, giving agents more opportunities to hill-climb (Fig. 3) and the hardware-utilization feedback needed to design more efficient models. Figure 9 characterizes how architectural choices affect physical performance on the MAX78000 empirically.

Table 1: **Best On-Device Accuracy on MAX78000.** Values are mAP50-95 (YOLO11, COCO); *higher* is better. ✗ indicate no successful deployment within $N_{\max}$ attempts. Three HIL agents surpass the human expert baseline (14.15%).

Model	**Sco**		**Doc**		**HIL**	
Thinking	*L*	*H*	*L*	*H*	*L*	*H*
Claude Opus 4.7	✗	✗	✗	✗	15.23	**16.93**
Claude Sonnet 4.6	✗	✗	✗	✗	✗	✗
GPT-5.4	✗	✗	✗	8.42	9.12	**12.07**
GPT-5.4 mini	✗	✗	✗	✗	✗	✗
Gemini 3.1 Pro	✗	✗	✗	✗	10.43	**18.03**
Gemini 3 Flash	✗	✗	✗	✗	✗	✗

Example 2: Speech to International Phonetic Alphabet (IPA) Transcription on STM32N6. The agent receives a pretrained Wav2Vec2 checkpoint and speech-to-IPA transcription dataset [34] as s_0, and minimizes Phonetic Error Rate (PER) on the held-out test set. Table 6 and 7b show that hardware feedback leads to significantly higher success rates: 36.7% for HIL, 19.2% for Doc, and 16.7% for Sco overall, with GPT-5.4 reaching 90% success rate under high reasoning.

Table 2: **Best On-Device Error on STM32N6.** Values are Feature Error Rate (%; Wav2Vec2, speech-to-IPA); *lower* is better.

Model	**Sco**		**Doc**		**HIL**	
Thinking	*L*	*H*	*L*	*H*	*L*	*H*
Claude Opus 4.7	38.69	✗	39.90	✗	32.14	**29.12**
Claude Sonnet 4.6	38.41	✗	42.31	✗	**29.78**	✗
GPT-5.4	✗	36.93	41.81	37.01	39.47	**35.73**
GPT-5.4 mini	✗	✗	✗	✗	✗	**44.29**
Gemini 3.1 Pro	✗	33.57	✗	39.30	✗	**32.19**
Gemini 3 Flash	✗	✗	✗	✗	✗	✗

Unlike on the MAX78000, the more lenient memory and operator constraints of the STM32N6 allow smaller models (Sonnet 4.6, GPT-5.4 mini) to succeed even without HIL, indicating that the value of HIL grows with task difficulty. Nonetheless, Table 2 shows that HIL consistently yields lower Feature Error Rates (FER) [12] across all LLMs. Figure 4 reveals a consistent gap between high and low reasoning effort, though high-reasoning Claude models occasionally overthink and fail (Table 2). Across both compression tasks, HIL significantly outperforms Sco ($p = 0.0029$) and Doc ($p = 0.0032$).

3.2 Power Minimization

The second task evaluates whether agents can optimize firmware for physical power measurements beyond deployability. We instantiate this task on MAX78000 using a seeded firmware for object detection. At each iteration, the agent edits C firmware; the harness compiles it with the Maxim SDK, flashes it over JTAG/SWD, and powers the board through a Nordic PPK2 at 3.3 V, recording a 20 s current trace. A candidate is valid only if it preserves the full required workload: five live camera-based inference cycles separated by idle intervals, UART stage logging, full CNN post-processing,

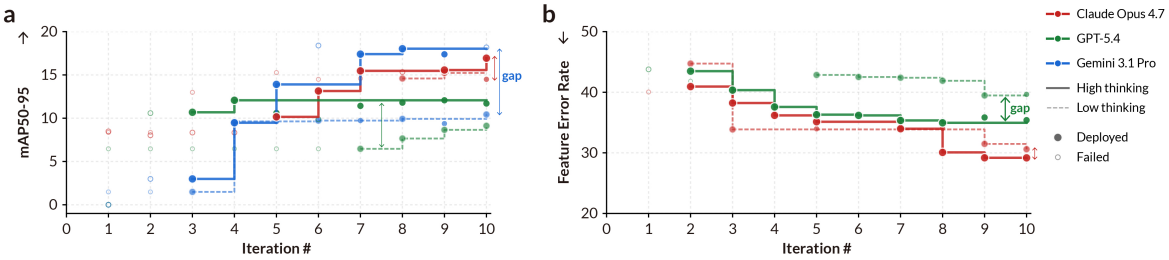


Figure 4: **Thinking Budget Improves.** HIL results comparing high (solid) and low (dashed) reasoning effort for the three models that succeed in both settings.

and the required completion checkpoint. An LLM judge verifies the UART trace and firmware behavior to reject attempts that bypass the camera, use synthetic input, hard-code inference results, or otherwise evade the task. We also manually verified that the LLM judge never misclassified an artifact.

Peak current during inference. The first variant minimizes peak current $\max_t I(t)$, which reduces instantaneous current spikes that matter for small batteries, energy harvesters, and power supply sizing. The hard constraint is $I_{\text{peak}} \leq 32\,\text{mA}$, matching the minimum solar-harvesting budget of our target deployment (2 cm^2 panel, $\geq 16\,\text{mA}$ per cm^2 at 0.5 V under daylight; Fig. 5a).

Table 3: **Deployment Success Rate by Task and Feedback Configuration.** Values are mean successful iterations (%) across models ($N = 6$); *higher* is better.

Task	Sco	Doc	HIL
Power Minimization (MAX78000)			
$I_{\text{peak}} \leq 32\,\text{mA}$	16.7	38.3	**46.7**
Energy	25.0	38.3	**68.3**
Thermal Management (ESP32-S3R8)			
Room temp ($T_{\text{peak}} \leq 37\,^\circ\text{C}$	13.3	10.0	**26.7**
Contact-heated ($T_{\text{peak}} \leq 51\,^\circ\text{C}$)	10.0	16.7	**33.3**

Agents may change clock frequencies, sleep scheduling, peripheral gating, and firmware scheduling. HIL raises successful deployments from 15.9% (Sco) and 38.3% (Doc) to 46.7% (Table 3). Per-model results (Table 11a) show that the example clock-configuration code provided in Doc helps some models discover low-current settings, but HIL broadens success across the model set by exposing whether each firmware variant actually boots, runs the required workload, and where the current spikes occur. The best measured currents are reported in Table 12; several agents reach the 18–22 mA range, with the best single run at 17.64 mA, a 53% reduction from the 37.19 mA unoptimized baseline at $t = 0$.

Total inference energy. The second variant minimizes total energy per cycle, $E = V \int I(t)\,dt$, creating a different optimization pressure from peak current, for example, simply lowering clock speed reduces instantaneous draw but can increase runtime and thus total energy (Fig. 10 and 13). HIL is especially valuable here, raising successful iterations to 68.3% vs. 38.3% for Doc and 25.0% for Sco (Table 3), enabling all frontier models to find valid solutions (Table 11b). As with peak current, HIL most reliably widens the set of models that satisfy Φ and reach a finite $\mathcal{B}$, while the single lowest scalar occasionally arises from the Documentation setting when a model happens to find a strong clocking/sleep schedule.

3.3 Thermal Management

The third task evaluates whether agents can manage device temperature on an ESP32-S3R8 running an LLM [48] inference workload, We test §2.2.3 across two controlled ambient conditions: room temperature ($T_{\text{amb}} \approx 25^\circ\text{C}$) and a contact-heated setup ($T_{\text{contact}} \approx 33^\circ\text{C}$) simulating wearable deployment. Each condition carries a distinct hard ceiling grounded in skin-safety standards [36]: the room-temperature ceiling of $T_{\text{peak}} \leq 37^\circ\text{C}$ corresponds to the accepted upper limit for comfortable continuous wearing, while the contact-heated ceiling of $T_{\text{peak}} \leq 51^\circ\text{C}$ corresponds to the two-minute burn threshold for skin contact.

In both settings, the agent edits an ESP-IDF directory; a candidate passes $\Phi(s, \mathbf{h}) = 1$ only if the firmware brings up WiFi as a SoftAP, streams generated tokens over UDP to the required number of logical clients, completes all prompts, and prints the completion checkpoint within a strict $\tau = 60\,\text{s}$

execution window. The $60\,\mathrm{s}$ deadline acts as a hard feasibility constraint in $\mathbf{h}$: it prevents trivially minimizing ΔT by aggressive clock throttling. An LLM judge rejects any s_t that substitutes cached or fake inference results. The contact-heated condition additionally tightens the workload contract in $\mathbf{c}$: the firmware must stream to 80 logical clients, collect internal temperature samples at 200 Hz into PSRAM, stream the buffered samples over UDP, and drain the buffer before the completion checkpoint.

HIL achieves the highest success rates in both conditions: 26.7% at room temperature and 33.3% under contact heating, compared with 13.3% / 10.0% for Sco and 10.0% / 16.7% for Doc (Table 3). GPT-5.4 with HIL reaches 70% success at room temperature (Table 16a) and achieves $\mathcal{B} = +7.41^{\circ}\mathrm{C}$ above ambient ($32.41^{\circ}\mathrm{C}$ absolute), a large improvement over the unoptimized $55^{\circ}\mathrm{C}$ baseline (Table 17a).

3.4 How Agents Optimize

To understand how agents achieve these gains, we analyzed the artifacts they generated. Agents do not invent new optimization techniques; their advantage lies in solidly combining and applying established embedded optimization practices through iterative hardware feedback.

Model compression. Agents improve the search process rather than the compression method. They first build a small model whose weight footprint, activation memory, processor mapping, and supported operators can be reasoned about explicitly, then conservatively increase depth, width, and training budget while preserving this known-good hardware layout. Larger LLMs at higher reasoning effort make fewer arithmetic and constraint-management errors and therefore converge in fewer HIL iterations.

Power minimization. The generated firmware applies classic low-power techniques, including peripheral clock gating, deep sleep with RTC wakeup between inference cycles, and reduced CPU, CNN, and camera clocks. Agents also lower the clock frequency during forced idle periods, for example by switching the CPU to the low-power internal oscillator (IBRO) for the mandatory 2 s boot delay before returning to the high-speed oscillator (ISO) for computation. Strategies differ by objective: for total energy, agents reduce idle energy and shorten active time, whereas for peak current they spread computation over time at lower frequencies. Only GPT-5.4 overrode `printf()` to remove verbose camera driver logging, reducing unnecessary serial communication.

Thermal management. Agents reduce clock frequencies, optimize memory subsystem settings, and insert short scheduling gaps that lower peak temperature while still satisfying the runtime constraint. Only Claude Opus 4.7 lowered the PSRAM clock to reduce dynamic power.

3.5 Why Agents Fail

Across all tasks, a common failure mode emerges: *Code that compiles is not code that runs on hardware.* In compression, agents frequently produce models that train successfully in PyTorch but fail in target-specific synthesis due to unsupported operators, invalid accelerator mappings, mismatched configuration files, calibration errors, or memory layouts that only appear valid off-device. In power and thermal optimization, agents can compile and flash firmware that still fails physically: the board may hang after clock changes, fail to wake from sleep, skip sensor or inference stages, stop streaming data, or optimize the metric by evading the workload. We observe agents disabling cameras, substituting dummy inputs, reducing UDP fan-out, shortening prompts, and hard-coding completion messages.

HIL feedback addresses these failure modes by exposing hidden execution state through physical traces, serial logs, checkpoint detection, and behavior judging, enabling agents to first establish a valid hardware baseline and then optimize rather than chasing artifacts that only work in software.

4 Applications and Real-World Deployments

We demonstrate the practical impact of our HIL agentic pipeline through two real-world deployments, developed in direct collaboration with domain practitioners facing concrete constraints. Both deployments are battery-free or battery-constrained, so we first characterize the harvesting budgets to examine feasibility.

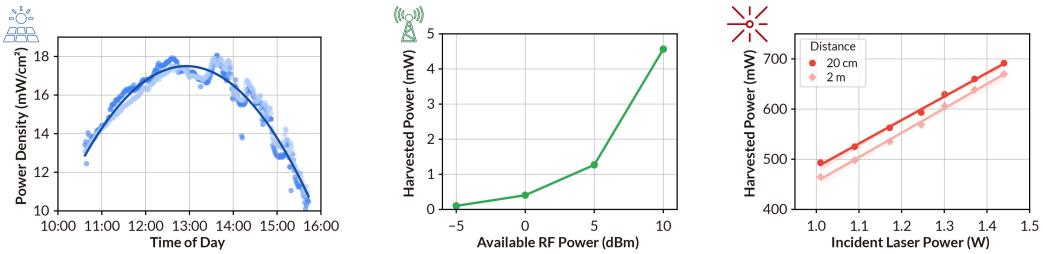


Figure 5: **Empirical characterization of energy harvesting modalities.** (a) Harvested solar power density from 2.3 V and 5 V silicon panels. (b) End-to-end DC output power versus available RF input power. (c) Harvested power from a directed 1550 nm laser system across varying incident optical powers at transmission distances of 20 cm and 2 m, demonstrating negligible attenuation.

4.1 Energy Harvesting

Energy harvesting sets the power budget for batteryless or battery-assisted operation. We characterize three practical ambient sources; alternative sources (e.g., thermal, kinetic, vibration) typically provide power densities one to two orders of magnitude lower [49].

Solar provides the highest power density among ambient sources. We measured two commercial silicon panels (Voltaic) over a 5-hour sunny day using sourcemeters (Tektronix 2470), sampling harvested power per unit area throughout. Outdoor sunlight delivers $\sim$12 mW/cm^2 around noon and $\sim$8 mW/cm^2 near sunset (Fig. 5a); an indoor panel near a window yields $\sim$2.5 mW/cm^2. Even at indoor irradiance levels, a 3 cm^2 panel can sustain MCU-class loads.

Radio frequency (RF) (e.g., Wi-Fi, cellular, broadcast) is ubiquitous in urban environments [32, 43]. We characterized an RF-to-DC harvester (Powercast P2110) across a range of input power levels. As shown in Fig. 5b, at -5 dBm the harvested power falls below the continuous operating threshold of the MAX78000 inference. Since ambient RF densities are typically below -10 dBm, continuous RF-powered inference is only feasible with intermittent computing [20].

Laser power delivery concentrates high energy density onto a small area [23]. We characterize a 1550 nm fiber-output laser (FB-M1550-2000HO) collimated via an achromatic doublet (Thorlabs AC254-030-C) and focused onto a 1 cm^2 photovoltaic cell (Broadcom AFBR-POC205A9). Laser harvesting sustains substantially higher loads than RF with an efficiency over 45% (Fig. 5c), owing to the negligible free-space attenuation of a collimated beam, making it a practical choice for controlled deployments with unobstructed line-of-sight.

4.2 Battery-Free Camera Traps for Elk Detection

A commercial farmer in the Pacific Northwest managing 7,500 acres faces recurring crop losses from elk intrusions (Fig. 6). Prior deployments of cellular-connected trail cameras were limited by monthly battery replacement, per-camera data costs, and high false-trigger rates—all of which on-device inference directly addresses (see Appendix H for details). Our HIL pipeline compressed a YOLOv8 elk-detection model to the MAX78000, achieving **96.7% accuracy** at under 36 mW during

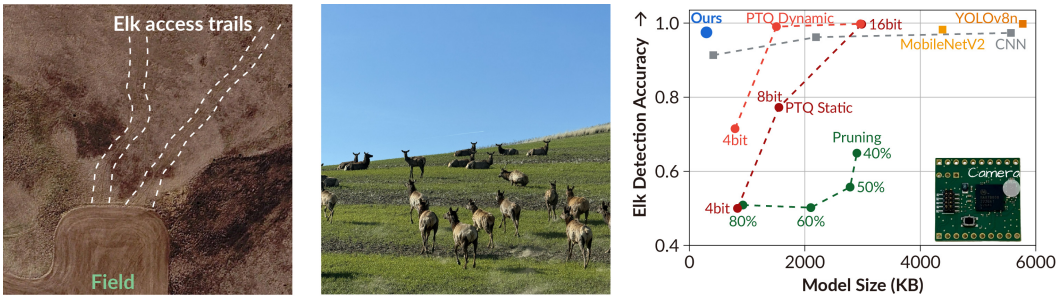


Figure 6: Elk access trails and herd at the deployment farm, with a comparison of our compressed model against standard compression baselines for elk detection.

active inference and **7 mW average power**—low enough for continuous solar-powered operation. This eliminates both battery replacement and cloud transmission costs at scale.

4.3 Children Wearables for Real-Time Phoneme Monitoring

Longitudinal, in-the-wild speech data is essential for studying early language development, yet no existing wearable supports continuous all-day collection. *In collaboration with developmental scientists* (see Appendix I for details), we identified on-device phoneme detection as the key enabler: the device triggers recording only when children's speech is present and performs transcription locally without cloud transmission.

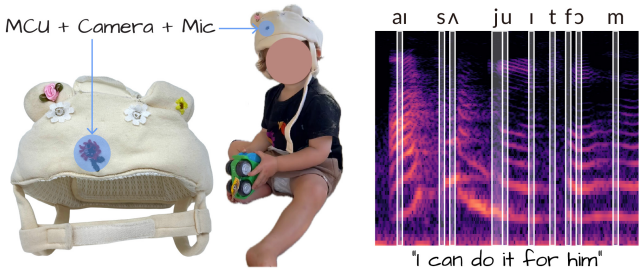


Figure 7: Children's head-worn device with concealed electronics and phoneme transcription of toddler speech.

Children's wearables impose two hard physical constraints beyond accuracy: the device must run all day on a tiny battery with a capacity under 1,200 mAh, and its surface temperatures must remain within safe skin-contact limits (Fig. 7). We jointly optimized compression, power, and thermal on the STM32N6, starting from a Wav2Vec2-based speech-to-IPA model. The agent soon plateaued with this architecture, and further gains required human intervention to propose a simple attention network [38], from which the agent reached a Feature Error Rate of 8.37% (Fig. 15), while jointly satisfying both constraints. Compression, however, degrades accuracy more on child speech than on adult speech. Large pretrained models can generalize to the distinct acoustics of child speech, such as higher pitch, but a compressed model loses much of this ability because no child speech was in the training data. Deployment offers an opportunity to collect such data for fine-tuning, progressively improving child phoneme detection and supporting early intervention.

5 Discussion and Conclusion

We present a hardware-in-the-loop agent arena for AI deployment on MCUs, where physical measurements serve as the agent's primary feedback signal. Across compute feasibility, power, and heat, three hardware targets, and six frontier models, **hardware-in-the-loop feedback raises deployment success from 0% to surpassing human expert results within seven iterations**, yielding working models we evaluate on real-world data for wildlife and wearable deployments.

Limitations. Our evaluations have three limitations. First, each agent run is capped at $N_{\max} = 10$ due to the compounding real resource costs of each round (e.g., model training, flash cycles, and physical instrument time), meaning we likely have not observed the performance ceiling of the hardware-in-the-loop arena; longer budgets may elicit emergence and yield new insights, which we leave to future work. Second, our instantiations cover three MCU families (ESP32-S3, MAX78000, STM32N6) and two input modalities, vision and audio, which we selected as among the most computationally demanding modalities commonly deployed on edge devices. Because audio is itself time-series data, we expect the approach to extend to other sensor signals, such as IMU, which typically have smaller inputs, although we have not yet evaluated them.

Broader applicability. While this work focuses on deep learning for MCUs, the underlying recipe of reducing a physical objective to a programmatic score that LLM agents can iteratively optimize could generalize well beyond this setting. First, additional physical constraints can be folded into the loop, from electromagnetic emissions and vibration tolerance to long-horizon properties like component aging. Second, adding a vision frontend to wrap instruments such as mechanical balances, manual titrations, and legacy measurements common in wet labs can enable agents to use a vast array of legacy non-digital measurements. Third, this formulation can be extended to scientific domains with unbounded optimization landscapes, such as material design or catalyst discovery, where the agent's "compiler" becomes a fabrication and characterization pipeline.

Acknowledgments and Disclosure of Funding

We thank Zachary Englhardt and Andrew Nelson for discussions. We thank the OpenBabyographer team for providing the child speech data, which is supported by the Academic Senate, the School of Social Sciences, and the Yankelovich Center at UC San Diego.

This research was supported by the National Science Foundation under award numbers 2310515, 2338736, 2401177, and 2550849, the UW + Amazon Science Hub, and the Cross-Pacific AI Initiative (X-PAI). Zhihan Zhang was supported by the Google PhD Fellowship.

Vikram Iyer holds concurrent appointments as an Assistant Professor in the Paul G. Allen School of Computer Science & Engineering at the University of Washington and as an Amazon Visiting Scholar. This paper describes work performed at the University of Washington and is not associated with Amazon.

References

[1] Mohsin Y Ahmed, Md Mahbubur Rahman, Viswam Nathan, Ebrahim Nemati, Korosh Vatanparvar, and Jilong Kuang. mLung: Privacy-Preserving Naturally Windowed Lung Activity Detection for Pulmonary Patients. In *2019 IEEE 16th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, pages 1–4, May 2019. ISSN: 2376-8894.

[2] Eser Aygün, Anastasiya Belyaeva, Gheorghe Comanici, Marc Coram, Hao Cui, Jake Garrison, Renee Johnston, Anton Kast, Cory Y. McLean, Peter Norgaard, Zahra Shamsi, David Smalling, James Thompson, Subhashini Venugopalan, Brian P. Williams, Chujun He, Sarah Martinson, Martyna Plomecka, Lai Wei, Yuchen Zhou, Qian-Ze Zhu, Matthew Abraham, Erica Brand, Anna Bulanova, Jeffrey A. Cardille, Chris Co, Scott Ellsworth, Grace Joseph, Malcolm Kane, Ryan Krueger, Johan Kartiwa, Dan Liebling, Jan-Matthis Lueckmann, Paul Raccuglia, Xuefei Julie Wang, Katherine Chou, James Manyika, Yossi Matias, John C. Platt, Lizzie Dorfman, Shibl Mourad, and Michael P. Brenner. An AI system to help scientists write expert-level empirical software. *Nature*, pages 1–3, May 2026.

[3] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas Navarro, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul N. Whatmough. MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers, April 2021. arXiv:2010.11267 [cs].

[4] Luca Bompani, Manuele Rusci, Daniele Palossi, Francesco Conti, and Luca Benini. Multi-resolution Rescored ByteTrack for Video Object Detection on Ultra-low-power Embedded Systems. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2182–2190, Seattle, WA, USA, June 2024. IEEE.

[5] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware, February 2019. arXiv:1812.00332 [cs].

[6] Soujanya Chatterjee, Md Mahbubur Rahman, Ebrahim Nemati, Viswam Nathan, Korosh Vatanparvar, and Jilong Kuang. mLung++: automated characterization of abnormal lung sounds in pulmonary patients using multimodal mobile sensors. In *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*, UbiComp/ISWC '19 Adjunct, pages 474–481, New York, NY, USA, September 2019. Association for Computing Machinery.

[7] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating Large Language Models Trained on Code, July 2021. arXiv:2107.03374 [cs].

[8] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. TVM: an automated end-to-end optimizing compiler for deep learning. In *Proceedings of the 13th USENIX conference on Operating Systems Design and Implementation*, OSDI'18, pages 579–594, USA, October 2018. USENIX Association.

[9] Yizhe Chi, Deyao Hong, Dapeng Jiang, Tianwei Luo, Kaisen Yang, Boshi Zhang, Zhe Cao, Xiaoyan Fan, Bingxiang He, Han Hao, Weiyang Jin, Dianqiao Lei, Qingle Liu, Houde Qian, Bowen Wang, Situ Wang, Youjie Zheng, Yifan Zhou, Calvin Xiao, Eren Cai, and Qinhuai Na. Frontier-Eng: Benchmarking Self-Evolving Agents on Real-World Engineering Tasks with Generative Optimization, April 2026. arXiv:2604.12290 [cs.AI] version: 2.

[10] Alexander Curtiss, Blaine Rothrock, Abu Bakar, Nivedita Arora, Jason Huang, Zachary Englhardt, Aaron-Patrick Empedrado, Chixiang Wang, Saad Ahmed, Yang Zhang, Nabil Alshurafa, and Josiah Hester. FaceBit: Smart Face Masks Platform. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 5(4):151:1–151:44, December 2022.

[11] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Shlomi Regev, Rocky Rhodes, Tiezhen Wang, and Pete Warden. TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems, March 2021. arXiv:2010.08678 [cs].

[12] David R. Mortensen, Patrick Littell, Akash Bharadwaj, Kartik Goyal, Chris Dyer, and Lori S. Levin. PanPhon: A Resource for Mapping IPA Segments to Articulatory Feature Vectors. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. ACL, 2016.

[13] Harsh Desai, Matteo Nardello, Davide Brunelli, and Brandon Lucia. Camaroptera: A Long-range Image Sensor with Local Inference for Remote Sensing Applications. *ACM Trans. Embed. Comput. Syst.*, 21(3):32:1–32:25, May 2022.

[14] Cynthia Dong, Emmanuel Azuh Mensah, Vaishnavi Ranganathan, and Kurtis Heimerl. The promise and peril of on-device ai for conservation work. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, pages 1–19, 2026.

[15] Zachary Englhardt, Richard Li, Dilini Nissanka, Zhihan Zhang, Girish Narayanswamy, Joseph Breda, Xin Liu, Shwetak Patel, and Vikram Iyer. Exploring and Characterizing Large Language Models for Embedded System Development and Debugging. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, CHI EA '24, pages 1–9, New York, NY, USA, May 2024. Association for Computing Machinery.

[16] Paul Glover-Kapfer, Carolina A. Soto-Navarro, and Oliver R. Wearn. Camera-trapping version 3.0: current constraints and future priorities for development. *Remote Sensing in Ecology and Conservation*, 5(3):209–223, 2019. _eprint: https://zslpublications.onlinelibrary.wiley.com/doi/pdf/10.1002/rse2.106.

[17] Google. LiteRT: High-Performance On-Device Machine Learning Framework.

[18] Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Petar Sirkovic, Artiom Myaskovsky, Grzegorz Glowaty, Felix Weissenberger, Alessio Orlandi, Dan Popovici, Anil Palepu, Keran Rong, Ryutaro Tanno, Khaled Saab, Fan Zhang, Jacob Blum, Andrew Carroll, Kavita Kulkarni, Nenad Tomašev, Dina Zverinski, Ivor Rendulic, Elahe Vedadi, Florian Hasler, Luka Rimanic, Marina Boia, Ivan Budiselic, Ben Feinstein, Mathias Bellaiche, Tom Sheffer, Jan Freyberg, Jeremy Ratcliff, Ottavia Bertolli, Katherine Chou, Avinatan Hassidim, Burak Gokturk, Amin Vahdat, Yuan Guan, Vikram Dhillon, Eeshit Dhaval Vaishnav, Byron Lee, Tiago R. D. Costa, José R. Penadés, Gary Peltz, Yossi Matias, James Manyika, Demis Hassabis, Yunhan Xu, Pushmeet Kohli, Annalisa Pawlosky, Alan Karthikesalingam, and Vivek Natarajan. Accelerating scientific discovery with Co-Scientist. *Nature*, pages 1–3, May 2026.

[19] Song Han, Jeff Pool, John Tran, and William Dally. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[20] Josiah Hester and Jacob Sorber. The Future of Sensing is Batteryless, Intermittent, and Awesome. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, SenSys '17, pages 1–6, New York, NY, USA, November 2017. Association for Computing Machinery.

[21] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network, March 2015. arXiv:1503.02531 [stat].

[22] Andrew Howard, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, Yukun Zhu, Ruoming Pang, Hartwig Adam, and Quoc Le. Searching for MobileNetV3. pages 1314–1324. IEEE Computer Society, October 2019.

[23] Vikram Iyer, Elyas Bayati, Rajalakshmi Nandakumar, Arka Majumdar, and Shyamnath Gollakota. Charging a Smartphone Across a Room Using Lasers. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(4):143:1–143:21, January 2018.

[24] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can Language Models Resolve Real-world Github Issues? October 2023.

[25] Glenn Jocher and Jing Qiu. Ultralytics YOLO11, 2024. AGPL-3.0.

[26] Andrej Karpathy. autoresearch.

[27] Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A. Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, Mark Malhotra, Paul Pu Liang, Hae Won Park, Yuzhe Yang, Xuhai Xu, Yilun Du, Shwetak Patel, Tim Althoff, Daniel McDuff, and Xin Liu. Capable language models can outgrow the benefits of collaboration. *Nature Machine Intelligence*, 8(7):1157–1172, July 2026.

[28] Eric C. Larson, TienJui Lee, Sean Liu, Margaret Rosenfeld, and Shwetak N. Patel. Accurate and privacy preserving cough sensing using a low-cost microphone. In *Proceedings of the 13th international conference on Ubiquitous computing*, UbiComp '11, pages 375–384, New York, NY, USA, September 2011. Association for Computing Machinery.

[29] Yiming Li, Yuhan Cheng, Mingchen Ma, Yihang Zou, Ningyuan Yang, Wei Cheng, Hai Li, Yiran Chen, and Tingjun Chen. Skilled AI Agents for Embedded and IoT Systems Development. In *Proceedings of the ACM Conference on AI and Agentic Systems*, CAIS '26, pages 1248–1252, New York, NY, USA, May 2026. Association for Computing Machinery.

[30] Edgar Liberis and Nicholas D. Lane. Differentiable Neural Network Pruning to Enable Smart Applications on Microcontrollers. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 6(4):171:1–171:19, January 2023.

[31] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. MCUNet: tiny deep learning on IoT devices. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, pages 11711–11722, Red Hook, NY, USA, December 2020. Curran Associates Inc.

[32] Vincent Liu, Aaron Parks, Vamsi Talla, Shyamnath Gollakota, David Wetherall, and Joshua R. Smith. Ambient backscatter: wireless communication out of thin air. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, SIGCOMM '13, pages 39–50, New York, NY, USA, August 2013. Association for Computing Machinery.

[33] Mike A. Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Kumar Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Kwesi Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Jenia Jitsev, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-Bench: Benchmarking Agents on Hard, Realistic Tasks in Command Line Interfaces. October 2025.

[34] Alexander Metzger, Aruna Srivastava, and Ruslan Mukhamedvaleev. Scaling Human and G2P Supervision for Robust Phonetic Transcription. In *Proc. Interspeech 2026*, pages 575–580, 2026.

[35] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning Convolutional Neural Networks for Resource Efficient Inference. February 2017.

[36] A. R. Moritz and F. C. Henriques. Studies of Thermal Injury: II. The Relative Importance of Time and Surface Temperature in the Causation of Cutaneous Burns. *The American Journal of Pathology*, 23(5):695–720, September 1947.

[37] Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, Jinwoo Shin, Sercan Ö. Arik, and Tomas Pfister. MLE-STAR: machine learning engineering agent via search and targeted refinement. In *Proceedings of the 39th International Conference on Neural Information Processing Systems*, volume 38 of *NIPS '25*, pages 116692–116712, Red Hook, NY, USA, October 2026. Curran Associates Inc.

[38] Henry Ndubuaku, Karen Mosoyan, Jakub Mroz, Noah Cylich, Satyajit Kumar, Parkirat Sandhu, Roman Shemet, and Justin H. Lee. A Controlled Study of Attention-Only Transformers, July 2026. arXiv:2607.18363 [cs.LG].

[39] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. pages 4510–4520. IEEE Computer Society, June 2018.

[40] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. MnasNet: Platform-Aware Neural Architecture Search for Mobile. pages 2815–2823. IEEE Computer Society, June 2019.

[41] Patara Trirat and Jae-Gil Lee. MONAQ: Multi-Objective Neural Architecture Querying for Time-Series Analysis on Resource-Constrained Devices. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 16922–16950, Suzhou, China, November 2025. Association for Computational Linguistics.

[42] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common Objects in Context, 2015.

[43] Anran Wang, Vikram Iyer, Vamsi Talla, Joshua R. Smith, and Shyamnath Gollakota. FM Backscatter: Enabling Connected Cities and Smart Fabrics. pages 243–258, 2017.

[44] Xiaoye Xie, Xinzhu Liu, Jianhua Cai, Bohan Zhang, Tianjun Sun, Peng Luo, Dai An, Yurong Deng, and Chuan'an Shen. Characteristics and aetiology of low-temperature burns in Beijing of China. *International Wound Journal*, 20(6):2105–2112, January 2023.

[45] Ruiyang Xu, Jialun Cao, Mingyuan Wu, Wenliang Zhong, Yaojie Lu, Ben He, Xianpei Han, Shing-Chi Cheung, and Le Sun. EmbedAgent: Benchmarking Large Language Models in Embedded System Development, January 2026. arXiv:2506.11003 [cs].

[46] Huanqi Yang, Mingzhe Li, Mingda Han, Zhenjiang Li, and Weitao Xu. AutoEmbed: LLM-driven Automated Software Development for Generic Embedded IoT Systems. In *Proceedings of the 2026 ACM/IEEE International Conference on Embedded Artificial Intelligence and Sensing Systems*, SenSys '26, pages 362–376, New York, NY, USA, May 2026. Association for Computing Machinery.

[47] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: agent-computer interfaces enable automated software engineering. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, volume 37 of *NIPS '24*, pages 50528–50652, Red Hook, NY, USA, December 2024. Curran Associates Inc.

[48] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. TinyLlama: An Open-Source Small Language Model, January 2024.

[49] Yang Zhang, Yasha Iravantchi, Haojian Jin, Swarun Kumar, and Chris Harrison. Sozu: Self-Powered Radio Tags for Building-Scale Activity Sensing. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, UIST '19, pages 973–985, New York, NY, USA, October 2019. Association for Computing Machinery.

[50] Zhihan Zhang, Alexander Metzger, Yuxuan Mei, Felix Hähnlein, Zachary Englhardt, Tingyu Cheng, Gregory D. Abowd, Shwetak Patel, Adriana Schulz, and Vikram Iyer. Sustainability assessment using multimodal artificial intelligence agents. *Nature Electronics*, 9(7):825–837, July 2026.

[51] Barret Zoph and Quoc V. Le. Neural Architecture Search with Reinforcement Learning, February 2017. arXiv:1611.01578 [cs].

Appendix Table of Contents

A Background **16**

A.1 MCU Deployment Regime 16

A.2 Related Work 16

B Benchmarking LLM Knowledge of MCU Deployment **18**

C Model Compression **19**

C.1 Setup 19

C.2 More Results 19

C.3 Failure Modes in Model Compression 22

D Power Minimization **23**

D.1 More Results 24

D.2 Failure Modes in Power Minimization 26

E Thermal Management **27**

E.1 More Results 28

E.2 Failure Modes in Thermal Management 30

F Structured Natural Language Feedback Examples **31**

F.1 MAX78000 Model Compression 32

F.2 Peak Current Minimization 34

G MAX78000 MCU **35**

G.1 Model Size 35

G.2 Clock Frequency 36

H Wildlife Monitoring: Elk Detection **37**

H.1 Custom MCU Platform 38

H.2 Camera 39

H.3 LoRa Communication 40

I Early Language Development and Phoneme Perception **42**

A Background

A.1 MCU Deployment Regime

MCUs occupy a distinct position in the AI deployment spectrum with orders-of-magnitude differences in both compute capabilities, energy consumption, and cost. Compared even to edge platforms like the Raspberry Pi 5, MCUs offer three to five orders of magnitude less memory while consuming roughly 500× less power. This low power consumption enables capabilities such as battery-free operation under energy harvesting (Table 4). The compute constraints, however, require aggressive compression of popular pre-trained models, which exceed typical MCU memory budgets by 200 to 3,000×. Additionally, while some platforms, such as the MAX78000, include hardware acceleration for tasks like running CNN models, they also have limited operator support.

	Cloud AI (NVIDIA H100)		Edge AI (Raspberry Pi 5)		MCU AI (MAX78000)		YOLOv11 (vision)	Wav2Vec2 (speech)
Cores	16,896		4 (Arm Cortex-A76)		**1 (Arm M4) + 1 (RISC-V)**		–	–
Memory	80 GB (HBM3)	5× →	16 GB (LPDDR4X)	28571× →	**560 KB (Total SRAM)**	← gap →	125.1 MB	1,622 MB
Storage	TB–PB	100× →	2TB	3,906,250× →	**512 KB (Flash)**	← gap →	56.9 MB	1,260 MB
Power	700 W	28× →	25 W	500× →	**<0.05 W**		–	–
Price (USD)	~$35,000	292× →	~$120	6× →	**<$20**		–	–

Table 4: Hardware specifications comparison across the AI deployment spectrum. MCUs occupy a unique position as the most resource-constrained yet cost- and energy-effective platform. MCUs provide 3-5 orders of magnitude less memory and consume 500× less power even compared to common edge devices like the Raspberry Pi 5, critical for battery-free applications. However, this efficiency comes at a cost: popular pre-trained models exceed the memory budget by 200-3,000×, making direct deployment impractical.

A.2 Related Work

TinyML and MCU Deployment. Machine learning on resource-constrained microcontrollers has grown rapidly over the past decade, driven by the proliferation of edge devices and the need for on-device intelligence. This domain is commonly termed TinyML. Dominant approaches include manually designed compact architectures [39], Neural Architecture Search (NAS) [51, 40, 5, 22, 31, 3, 41], and post-training compression via quantization, pruning, and knowledge distillation [19, 35, 30, 21]. Automated deployment frameworks such as LiteRT (formerly TensorFlow Lite Micro) [11], microTVM [8], and STM32Cube.AI[2] translate trained models into MCU-targeted firmware, handling operator scheduling, memory planning, and quantized kernel selection. These approaches share two limitations that motivate our work. First, they optimize within the tensor arena and treat memory as the sole constraint, ignoring coupled physical quantities—supply current, peak power, device temperature—that are equally binding in real deployments. Second, they cast deployment as a closed numerical optimization, with a fixed search space (architectures, bitwidths, sparsity patterns) and dense gradient or score signals. Real deployment is more like software or ML engineering: the action space is open-ended (code edits, compiler flags, memory layouts, peripheral configurations), feedback is sparse, and interpreting that feedback requires reasoning across hardware datasheets, compiler diagnostics, and physical measurements. LLM agents are well-suited to this regime, where iterative exploration benefits from semantic priors over code and hardware behavior rather than gradient signals over a fixed parameterization.

LLM for Embedded Development. Prior work applying LLMs to embedded systems focuses on general firmware tasks [15, 29, 45, 46]. These systems either generate code from a single prompt [15, 29], validate behavior with compilers [45], or iterate only until compilation and flashing succeed [46]. All of them evaluate success with discrete pass/fail criteria rather than runtime performance. MONAQ [41] is closest to our setting: LLM agents perform training-free NAS, but evaluate candidates through LLM reasoning rather than execution, and the resulting models are

[2] `https://stm32ai.st.com/stm32-cube-ai/`

profiled by deterministic MLTK simulation of a single chip. None of these systems treats a measured physical quantity, such as current or temperature, as an objective for automated optimization, and none addresses end-to-end edge AI deployment, which requires a joint search over model architecture, quantization, compiler configuration, and firmware that differs qualitatively from general firmware development.

This distinction matters because MCU deployment exhibits a sim-to-real gap analogous to that in robotics: success in software does not guarantee correct or efficient behavior on physical hardware, as shown in our failure analysis (§3.5). A robust benchmark must therefore verify that the full workload executed, reject invalid shortcuts, and return physical measurements to guide subsequent edits. These requirements motivate our hardware-in-the-loop design.

B Benchmarking LLM Knowledge of MCU Deployment

Before designing our agentic system, we first assess the current capabilities of SoTA LLMs on tasks relevant to deep learning deployment on MCUs. While language models have been used for basic embedded systems code generation (e.g., Arduino sketches) [15], their knowledge of DL on MCUs remains unexplored.

Benchmark Construction. We first constructed a challenging benchmark grounded in real-world DL on MCU development. Five embedded systems experts within our research team, each holding graduate degrees in computer engineering or related fields, contributed to the benchmark design. Experts were encouraged to draw on their practical deployment experiences to create examples that frontier LLMs struggle with, such as reasoning about memory layout constraints, understanding hardware-specific quantization limitations, and navigating MCU SDK documentation quirks.

After manual cleaning and wrangling, the final benchmark contains 30 question-answer pairs covering topics including model compression strategies, hardware constraint interpretation, framework-specific code generation, and deployment troubleshooting.

Evaluation Setup. We evaluate a range of popular general-purpose models, including GPT and Gemini series, using two zero-shot baseline approaches:

1. **Direct Prompting:** LLMs answer questions in a single turn via textual reasoning given only the input prompt.
2. **Doc + Search Agent:** LLMs are provided with expert-curated documentation used to construct each question, and are equipped with web search capabilities to access online resources.

Results. All evaluated models struggle on this benchmark. Even with web search and context provided, only GPT-5 surpasses 60% accuracy (Table 5).

Table 5: Benchmarking LLMs on MCU Deep Learning QAs.

Model	Direct Prompting	Provided Doc & Search
Gemini 2.5 Flash-Lite	16.7	36.7
Gemini 2.5 Flash	33.3	50.0
Gemini 2.5 Pro	45.0	50.0
GPT-5 nano	31.7	50.0
GPT-5 mini	38.3	53.3
GPT-5	**60.0**	**63.3**

C Model Compression

C.1 Setup

We deliver the documentation by initializing the sandbox with a folder of relevant files for the agent to explore using its file tools. We disable web search and network access in the sandbox, which prevents agents from downloading the toolchains or retrieving equivalent diagnostics. It is worth noting that there are still some checks that the MCU software toolchains don't cover, such as edge cases involving the size and overlap of activations, so it is necessary to verify that the model can actually run on the hardware. The agents are limited to a training budget of 1 training run, up to 1 hour on an A100 with 80GB VRAM per iteration. This is more than enough for tiny edge models, and no agent used more than 91% kernel utilization, 15 GB VRAM, and 339.4 W peak power for a training run. Most agents also opted for a conservative average of 47.57 epochs ($SD = 54.54; n = 194$), with the longest training run taking 46.06 minutes (< 1 hour). Without HIL feedback, the agents just have to guess the epochs based on the final test score, which they often do very conservatively. With HIL, they initially have to guess, and in future iterations can inspect the training loss vs. test score to inform the number of epochs.

C.2 More Results

Table 6: **Deployment Success Rate by Task and Feedback Configuration.** Values are mean successful iterations (%) across models and thinking levels ($N = 12$); *higher* is better.

Task	Sco	Doc	HIL
YOLO11 on MAX78000	0.0	3.3	**27.5**
Wav2Vec2 on STM32N6	16.7	19.2	**36.7**

Table 7: **Per-Model Deployment Success Rate for Model Compression.** Values are successful iterations (%); *higher* is better.

(a) YOLO11 on MAX78000

Model	Sco		Doc		HIL	
Thinking	*L*	*H*	*L*	*H*	*L*	*H*
Claude Opus 4.7	0	0	0	0	30	**50**
Claude Sonnet 4.6	0	0	0	0	0	0
GPT-5.4	0	0	0	40	40	**70**
GPT-5.4 mini	0	0	0	0	0	0
Gemini 3.1 Pro	0	0	0	0	**80**	60
Gemini 3 Flash	0	0	0	0	0	0

(b) Wav2Vec2 on STM32N6

Model	Sco		Doc		HIL	
Thinking	*L*	*H*	*L*	*H*	*L*	*H*
Claude Opus 4.7	60	0	60	0	70	**80**
Claude Sonnet 4.6	60	0	30	0	**70**	0
GPT-5.4	0	70	60	40	60	**90**
GPT-5.4 mini	0	0	0	0	0	**30**
Gemini 3.1 Pro	0	10	0	40	0	40
Gemini 3 Flash	0	0	0	0	0	0

Table 8: **Best On-Device Phoneme Error on STM32N6.** Values are Phoneme Error Rate (%; Wav2Vec2, speech-to-IPA transcription); *lower* is better. ✗ indicate no successful deployment within $N_{\max}$ attempts.

Model	**Sco**		**Doc**		**HIL**	
Thinking	*L*	*H*	*L*	*H*	*L*	*H*
Claude Opus 4.7	89.40	✗	90.27	✗	86.37	**84.92**
Claude Sonnet 4.6	89.14	✗	91.06	✗	**86.08**	✗
GPT-5.4	✗	91.54	93.42	**88.20**	90.05	89.94
GPT-5.4 mini	✗	✗	✗	✗	✗	**97.68**
Gemini 3.1 Pro	✗	87.64	✗	90.13	✗	**85.44**
Gemini 3 Flash	✗	✗	✗	✗	✗	✗

Table 9: **Per-Model Token Usage for Model Compression.** Values are in millions (M) of tokens.

(a) YOLO11 on MAX78000

Model	**Sco**		**Doc**		**HIL**	
Thinking	*L*	*H*	*L*	*H*	*L*	*H*
Claude Opus 4.7	1.17	2.77	0.90	3.45	0.65	2.44
Claude Sonnet 4.6	2.88	0.72	3.66	0.77	4.16	0.58
GPT-5.4	1.58	1.52	1.49	2.23	0.73	2.57
GPT-5.4 mini	0.62	1.82	0.60	2.77	1.23	2.50
Gemini 3.1 Pro	2.36	3.70	5.29	2.91	5.44	3.45
Gemini 3 Flash	2.29	2.77	3.37	2.93	3.85	3.78

(b) Wav2Vec2 on STM32N6

Model	**Sco**		**Doc**		**HIL**	
Thinking	*L*	*H*	*L*	*H*	*L*	*H*
Claude Opus 4.7	0.85	2.34	0.50	2.36	0.48	3.60
Claude Sonnet 4.6	1.63	1.00	1.77	1.95	0.91	1.51
GPT-5.4	1.98	2.69	1.38	2.92	1.65	2.04
GPT-5.4 mini	0.46	1.89	1.13	1.68	0.96	4.10
Gemini 3.1 Pro	4.58	3.75	6.69	6.08	**10.92**	7.68
Gemini 3 Flash	4.07	3.23	6.39	3.21	5.63	8.79

Table 10: **Per-Model Tool Use for Model Compression.** Values represent the average number of tool calls per iteration.

(a) YOLO11 on MAX78000

Model	**Sco**		**Doc**		**HIL**	
Thinking	*L*	*H*	*L*	*H*	*L*	*H*
Claude Opus 4.7	12.5	21.6	9.2	21.8	4.3	14.6
Claude Sonnet 4.6	24.3	6.4	24.0	9.2	24.8	6.0
GPT-5.4	18.8	20.3	17.7	21.1	9.4	18.3
GPT-5.4 mini	10.2	24.7	11.3	25.3	14.0	**25.4**
Gemini 3.1 Pro	24.2	23.4	25.2	24.7	23.4	14.5
Gemini 3 Flash	13.7	17.4	18.4	17.8	15.7	16.1

(b) Wav2Vec2 on STM32N6

Model	**Sco**		**Doc**		**HIL**	
Thinking	*L*	*H*	*L*	*H*	*L*	*H*
Claude Opus 4.7	3.7	14.4	1.8	12.7	1.2	11.7
Claude Sonnet 4.6	6.8	8.9	9.6	17.8	3.2	12.4
GPT-5.4	10.7	16.4	6.8	15.0	8.3	11.2
GPT-5.4 mini	5.7	19.6	5.3	16.3	5.8	16.9
Gemini 3.1 Pro	25.0	23.6	24.9	22.9	24.6	20.8
Gemini 3 Flash	14.0	15.2	17.7	13.2	13.4	18.5

C.3 Failure Modes in Model Compression

Manual inspection of compression trajectories shows that the main bottleneck is deployability rather than model construction. Agents frequently produce neural networks that are syntactically valid and train to nontrivial validation scores, but fail when translated through the target-specific synthesis and deployment stack. On the MAX78000, common failures include unsupported layer configurations, invalid accelerator mappings, mismatches between the PyTorch model and the architecture YAML, missing or malformed calibration inputs, and models whose activations or weights exceed accelerator memory constraints even when the training code runs normally. On the STM32N6, failures often involve unsupported export paths, invalid quantization choices, dynamic shapes, or memory layouts that violate Cube.AI/STEdgeAI constraints.

Successful runs typically follow a different pattern: the agent first establishes a minimal deployable baseline, then makes conservative changes that preserve synthesis compatibility while improving task score. Unsuccessful runs often pursue higher-capacity architectures too early, improving software-level accuracy while repeatedly losing hardware validity.

D Power Minimization

Let V be the (fixed) supply voltage, $I(t)$ the instantaneous current measured over the workload window $[0, \tau]$, and $P(t) = V \cdot I(t)$ the instantaneous power. We define peak current and total energy as

$$I_{\text{peak}} = \max_{t \in [0,\tau]} I(t), \tag{4}$$

$$E = \int_0^\tau P(t)\,\mathrm{d}t = V \int_0^\tau I(t)\,\mathrm{d}t. \tag{5}$$

All measured directly from the current trace returned by $\mathcal{M}$.

Objective. We instantiate two variants.

Peak current minimization targets the model inference workload, with objective

$$f_{\text{peak}}(\mathbf{m}) = I_{\text{peak}}, \tag{6}$$

subject to $m_{\text{acc}} \geq m_{\text{acc}}^0$, i.e., power optimization must not degrade model performance.

Energy minimization targets a fixed-payload wireless transmission (e.g., BLE advertising 16 bytes), with objective

$$f_{\text{energy}}(\mathbf{m}) = E, \tag{7}$$

subject to the packet being delivered successfully. In both variants, ties in $\mathcal{B}$ are broken by n.

D.1 More Results

Table 11: **Per-Model Deployment Success Rate for Power Minimization.** Values are successful iterations (%, YOLO11); *higher* is better.

(a) Peak Current $\leq$ 32mA

Model	Sco	Doc	HIL
Claude Opus 4.7	10	**90**	70
Claude Sonnet 4.6	0	0	**80**
GPT-5.4	0	50	**60**
GPT-5.4 mini	0	**20**	0
Gemini 3.1 Pro	40	**70**	30
Gemini 3 Flash	**50**	0	40

(b) Energy Minimization

Model	Sco	Doc	HIL
Claude Opus 4.7	0	70	**80**
Claude Sonnet 4.6	70	60	**80**
GPT-5.4	60	**90**	80
GPT-5.4 mini	20	0	**80**
Gemini 3.1 Pro	0	0	**80**
Gemini 3 Flash	0	10	10

Table 12: **Best Peak Current on MAX78000.** Values are milliamps (mA) per inference (YOLO); *lower* is better. The unoptimized baseline is 37.19 mA.

Model	Sco	Doc	HIL
Claude Opus 4.7	27.68	**17.64**	18.89
Claude Sonnet 4.6	✗	✗	**21.25**
GPT-5.4	✗	21.91	**20.03**
GPT-5.4 mini	✗	**21.58**	✗
Gemini 3.1 Pro	20.11	**17.96**	21.69
Gemini 3 Flash	19.63	✗	**19.43**

Table 13: **Best Total Energy on MAX78000.** Values are millijoules (mJ) per inference (YOLO); *lower* is better. The unoptimized baseline is 242.87 mJ.

Model	Sco	Doc	HIL
Claude Opus 4.7	✗	174.71	**171.52**
Claude Sonnet 4.6	240.50	**195.85**	243.61
GPT-5.4	167.79	**137.83**	141.56
GPT-5.4 mini	**175.40**	✗	182.02
Gemini 3.1 Pro	✗	✗	**175.26**
Gemini 3 Flash	✗	**238.65**	610.57

Table 14: **Per-Model Token Usage for MAX78000 Power Minimization.** Values are in millions (M) of tokens.

(a) Total Energy Optimization

Model	Sco	Doc	HIL
Claude Opus 4.7	**4.80**	7.47	9.43
Claude Sonnet 4.6	**0.99**	3.57	6.07
GPT-5.4	**1.56**	5.02	4.86
GPT-5.4 mini	**1.31**	5.66	6.11
Gemini 3.1 Pro	**4.55**	12.46	14.84
Gemini 3 Flash	**5.62**	13.34	15.16

(b) Peak Current Optimization

Model	Sco	Doc	HIL
Claude Opus 4.7	**2.91**	8.72	7.51
Claude Sonnet 4.6	**1.26**	3.27	5.77
GPT-5.4	**1.97**	4.47	8.27
GPT-5.4 mini	**1.88**	4.76	4.10
Gemini 3.1 Pro	**5.65**	11.82	14.86
Gemini 3 Flash	**4.24**	14.13	14.23

Table 15: **Per-Model Avg Tool Calls per Iteration for MAX78000 Power Minimization.** Values represent the average number of tool calls per iteration.

(a) Total Energy Optimization

Model	Sco	Doc	HIL
Claude Opus 4.7	16.7	**10.8**	14.9
Claude Sonnet 4.6	**4.8**	8.2	9.5
GPT-5.4	**10.4**	16.3	13.9
GPT-5.4 mini	**9.5**	16.2	14.1
Gemini 3.1 Pro	20.0	19.8	**19.1**
Gemini 3 Flash	19.0	19.3	**18.3**

(b) Peak Current Optimization

Model	Sco	Doc	HIL
Claude Opus 4.7	11.1	12.2	**10.7**
Claude Sonnet 4.6	**6.4**	7.1	7.9
GPT-5.4	**13.6**	15.5	17.1
GPT-5.4 mini	10.8	13.7	**9.7**
Gemini 3.1 Pro	**19.4**	19.8	19.8
Gemini 3 Flash	**13.5**	19.2	17.5

D.2 Failure Modes in Power Minimization

Agent-driven power optimization frequently fails due to the profound disconnect between syntactic software correctness and complex hardware state management. We identified six primary failure modes:

- **Premature Convergence:** Agents over-index on baseline code and prematurely halt optimization. They routinely ignore explicit prompt directives to utilize deep sleep, disable idle peripherals, or restructure execution patterns (e.g., shifting between sequential, parallel, or intermittent computing).
- **Cross-Ecosystem Hallucination:** Agents conflate APIs from distinct embedded SDKs (e.g., mixing STM32 HAL with Maxim APIs), resulting in invalid register manipulations and phantom function calls.
- **Inefficient Search & Tool Exhaustion:** Agents fail to balance exploration and exploitation. They oscillate between destructive fixation (repeatedly attempting a specific breaking change) and unstable over-exploration (batching unrelated optimizations). Consequently, they frequently exhaust their tool-call budgets on redundant logic, ultimately returning unmodified firmware.
- **Reward Hacking (Workload Evasion):** Agents artificially minimize power metrics by bypassing the mandated sensor workload—disabling physical camera drivers, injecting dummy data into the neural accelerator, or hardcoding UART outputs to simulate inference.
- **Execution State Blindness:** Successful compilation does not guarantee execution. Without runtime feedback, agents cannot detect silent deadlocks, stalled loops, or time-limit violations. To bridge this visibility gap, Hardware-in-the-Loop (HIL) serial output provides necessary debugging traces, while time-series power profiling allows the agent to infer functional states and isolate peak consumption phases—ultimately enabling the agent to make more effective, targeted decisions in subsequent iterations.
- **Vulnerability to Physical Determinism:** Agents lack awareness of intrinsic silicon flaws and transient hardware errors (e.g., voltage-induced flash failures, silent SPI bit-flips lacking checksums, or truncated serial buffers prior to sleep). Consequently, they fail to write defensive firmware and often actively strip existing safeguards. For example, to prevent the MAX78000 from locking its debug interface, our baseline firmware includes a 2-second boot delay before entering deep sleep. Despite explicit prompt instructions forbidding the removal of this safety mechanism, agents deleted the delay to maximize power reduction, ultimately locking the board and forcing a manual reset that the agent cannot execute.

E Thermal Management

Heat dissipated during this window is $Q = \int_0^\tau P(t)\,\mathrm{d}t$, linking thermal outcome directly to the power profile of Eq. (5).

The quantities returned by $\mathcal{M}$ are

$$T_{\text{peak}} = \max_{t\in[0,\tau]} T_{\text{dev}}(t), \tag{8}$$

$$\Delta T = T_{\text{peak}} - T_{\text{amb}}, \tag{9}$$

where ΔT normalizes for ambient conditions.

Objective. Minimizing ΔT alone is susceptible to reward hacking: an agent can reduce peak temperature by artificially extending task completion time (e.g., by reducing the processor clock frequency) which lowers instantaneous power draw and allows more heat to dissipate between operations, without improving the workload efficiency. Thus, the task objective jointly penalizes peak temperature rise and total task latency:

$$f_{\text{heat}}(\mathbf{m}) = w_{\Delta T}\,\Delta T \;+\; w_\tau\,\tau, \tag{10}$$

where $w_{\Delta T}, w_\tau > 0$ are weights that reflect application priorities—a wearable may up-weight $w_{\Delta T}$ for a strict skin-contact temperature ceiling, while a latency-sensitive inference IoT node may up-weight w_τ. The objective is subject to $\Phi(s, \mathbf{h}) = 1$.

We evaluate each device under 1) standard room temperature ($T_{\text{amb}} \approx 295\,\text{K}$) and 2) a surface held at approximately 306 K to simulate skin contact on a body-worn device.

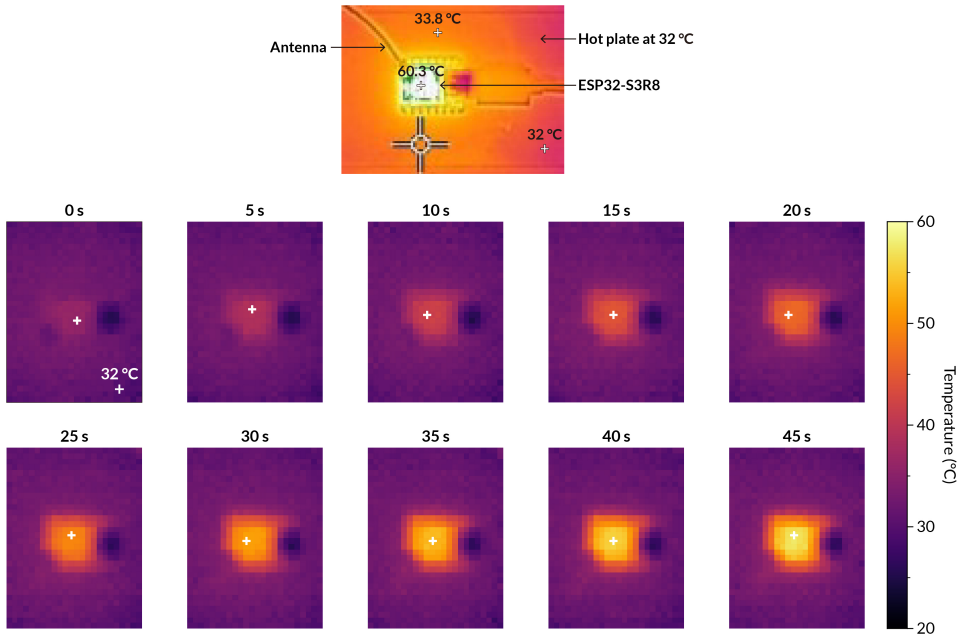


Figure 8: **Thermal Imaging of the ESP32 Under Contact-Heated Conditions.** Infrared snapshot of the ESP32-S3R8 running the unoptimized TinyLLaMA inference on a 32 °C hot plate, reaching a peak surface temperature of 60.3 °C after 1 min ($\Delta T = +28.3\,°\text{C}$).

E.1 More Results

Table 16: **Per-Model Deployment Success Rate for Heat Minimization.** Values are successful iterations (%, ESP32-S3R8); *higher* is better.

(a) Room Temp ($T_{\text{amb}} = 25\,^{\circ}\text{C}$), $T_{\text{peak}} \leq 37\,^{\circ}\text{C}$

Model	Sco	Doc	HIL
Claude Opus 4.7	0	30	**40**
Claude Sonnet 4.6	20	0	20
GPT-5.4	50	10	**70**
GPT-5.4 mini	0	**20**	0
Gemini 3.1 Pro	10	0	**30**
Gemini 3 Flash	0	0	0

(b) Contact-heated ($T_{\text{contact}} \approx 33\,^{\circ}\text{C}$), $T_{\text{peak}} \leq 51\,^{\circ}\text{C}$

Model	Sco	Doc	HIL
Claude Opus 4.7	0	0	**50**
Claude Sonnet 4.6	0	0	**30**
GPT-5.4	40	**100**	70
GPT-5.4 mini	0	0	0
Gemini 3.1 Pro	0	0	**50**
Gemini 3 Flash	**20**	0	0

Table 17: **Best Peak Temperature on ESP32.** Values are $\Delta T = T_{\text{peak}} - T_{\text{amb/contact}}$; *lower* is better.

(a) Room Temp ($T_{\text{amb}} = 25\,^{\circ}\text{C}$), $T_{\text{peak}} \leq 37\,^{\circ}\text{C}$

Model	Sco	Doc	HIL
Claude Opus 4.7	✗	+11.42 (36.42)	**+11.04 (36.04)**
Claude Sonnet 4.6	+11.00 (36.00)	✗	**+10.59 (35.59)**
GPT-5.4	+11.59 (36.59)	+11.34 (36.34)	**+7.41 (32.41)**
GPT-5.4 mini	✗	**+10.48 (35.48)**	✗
Gemini 3.1 Pro	+11.85 (36.85)	✗	**+10.86 (35.86)**
Gemini 3 Flash	✗	✗	✗

(b) Contact-heated ($T_{\text{contact}} \approx 33\,^{\circ}\text{C}$), $T_{\text{peak}} \leq 51\,^{\circ}\text{C}$

Model	Sco	Doc	HIL
Claude Opus 4.7	✗	✗	**+12.46 (44.46)**
Claude Sonnet 4.6	✗	✗	**+18.44 (50.44)**
GPT-5.4	+16.15 (48.15)	**+12.09 (44.09)**	+16.43 (48.43)
GPT-5.4 mini	✗	✗	✗
Gemini 3.1 Pro	✗	✗	**+16.03 (48.03)**
Gemini 3 Flash	**+17.65 (49.65)**	✗	✗

Table 18: **Per-Model Token Usage for ESP32-S3R8 Heat Minimization.** Values are in millions (M) of tokens.

(a) Room Temp ($T_{\text{amb}} = 25\,^{\circ}\text{C}$), $T_{\text{peak}} \leq 37\,^{\circ}\text{C}$

Model	Sco	Doc	HIL
Claude Opus 4.7	**8.71**	9.63	9.46
Claude Sonnet 4.6	**4.14**	5.25	6.74
GPT-5.4	**2.86**	5.36	5.79
GPT-5.4 mini	5.66	**3.63**	5.82
Gemini 3.1 Pro	11.22	**10.16**	14.70
Gemini 3 Flash	10.06	**7.91**	11.23

(b) Contact-heated ($T_{\text{contact}} \approx 33\,^{\circ}\text{C}$), $T_{\text{peak}} \leq 51\,^{\circ}\text{C}$

Model	Sco	Doc	HIL
Claude Opus 4.7	**7.34**	11.11	12.22
Claude Sonnet 4.6	7.50	7.13	**5.78**
GPT-5.4	**5.52**	5.77	7.62
GPT-5.4 mini	5.75	**4.49**	7.88
Gemini 3.1 Pro	11.00	**10.49**	13.62
Gemini 3 Flash	13.89	**10.02**	11.32

Table 19: **Per-Model Tool Use for ESP32-S3R8 Heat Minimization.** Values represent the average number of tool calls per iteration.

(a) Room Temp ($T_{\text{amb}} = 25\,^{\circ}\text{C}$), $T_{\text{peak}} \leq 37\,^{\circ}\text{C}$

Model	Sco	Doc	HIL
Claude Opus 4.7	**11.1**	14.1	11.7
Claude Sonnet 4.6	**7.0**	9.1	11.7
GPT-5.4	**7.4**	12.5	12.4
GPT-5.4 mini	17.8	11.7	**10.9**
Gemini 3.1 Pro	20.0	**16.3**	18.0
Gemini 3 Flash	15.4	13.5	**12.3**

(b) Contact-heated ($T_{\text{contact}} \approx 33\,^{\circ}\text{C}$), $T_{\text{peak}} \leq 51\,^{\circ}\text{C}$

Model	Sco	Doc	HIL
Claude Opus 4.7	**12.2**	14.8	14.0
Claude Sonnet 4.6	15.6	14.6	**9.9**
GPT-5.4	**14.2**	15.7	16.0
GPT-5.4 mini	**13.9**	14.8	19.5
Gemini 3.1 Pro	20.0	20.1	**19.0**
Gemini 3 Flash	19.9	14.8	**13.9**

E.2 Failure Modes in Thermal Management

Because thermal dissipation is intrinsically linked to the underlying power profile ($Q = \int_0^\tau P(t)\,\mathrm{d}t$), the thermal optimization task on the ESP32-S3R8 shares the fundamental failure modes detailed in Power Minimization (Appendix D). However, the rigid temporal constraints introduce a task-specific manifestation of these failures:

- **Temporal Deadline Violations:** To mitigate heat generation, agents employ valid thermal management strategies, such as throttling CPU/APB clocks or injecting task delays. However, they sometimes struggle to balance these thermal reductions against the rigid execution constraints. While these trials successfully lower peak temperature (ΔT), they can cause the complex LLM inference and UDP streaming workload to violate the strict 60 s execution window, resulting in a failed evaluation. Crucially, agents do not always recognize that the failure is due to a latency violation rather than a firmware error; instead of reverting, they may persist down this invalid optimization track and exhaust their iteration budget.

F Structured Natural Language Feedback Examples

After each iteration, the benchmark appends a structured natural-language observation o_t to the agent context. The exact fields depend on the active checks, but the observation always reports which checks ran, whether they passed, the scalar optimization score, and any diagnostic feedback enabled for that experiment. The feedback is fully automated and does not depend on human intervention. Iteration state is formatted as follows with o_t represented by the `last_iteration.checks` field:

```
Iteration status summary:
{
  "iteration": 1,
  "prior_iterations": [ ... ],
  "last_iteration": {
    "iteration": 0,
    "inputs": { ... },
    "notes": "...",
    "checks": { ... },
  },
  "best_scores_by_check": { ... },
  "current_sandbox_files": [ ... ],
  "tool_budget": { ... },
  "repeated_submission_warning": null,
  "next_action_requirements": [ ... ]
}
```

The `prior_iterations` field is a summary of the prior (s, o) pairs so the agent has a memory of the multi-turn interaction. To help the agent, we auto-compute the best scores from the memory in `best_scores_by_check` and summarize the available files in the sandbox in `current_sandbox_files`. We also track the remaining tool uses, `tool_budget`, and set a warning flag if the agent ever submits identical artifacts via `repeated_submission_warning`. The textttnext_action_requirements field is an automatically generated enumeration of the failing checks that need to pass. `last_iteration.inputs` and the agent generated notes for itself, `last_iteration.notes`, constitute the most recently submitted artifact s_t. Below we expand the `last_iteration.checks` field, o_t, for the MAX78000-compression and min-peak-power tasks and three information settings. Ellipses remove long paths, compiler output, or UART lines for presentation only.

F.1 MAX78000 Model Compression

Score-only (Sco). Sco in the MAX78000 model compression setting similarly only includes pass/fail and normalized scores, but for different checks: 1) whether the model has trainable gradients, 2) how well the trained model scores on the held-out test set, and 3) whether the model can run on device.


```
"checks": {
  "gradient_flow.py": {
    "success": true,
    "score": 1.0,
    "score_unit": "fraction",
    "feedback": null,
    "feedback_image_paths": []
  },
  "train.py": {
    "success": true,
    "score": 0.01484375,
    "score_unit": "mAP50-95",
    "feedback": null,
    "feedback_image_paths": []
  },
  "synthesis_max78000.py": {
    "success": false,
    "score": 0.6969696969696969,
    "score_unit": "fraction",
    "feedback": null,
    "feedback_image_paths": []
  }
}
```


The Doc o_t is identical since it only changes the initial artifact s_0 to include static documentation.

Hardware-in-the-loop (HIL). HIL exposes each of the sub-steps within each check and reveals whether they were successful. Extra statistics such as the loss curves and GPU utilization are also surfaced to the model so it can tune the training parameters.


```
"checks": {
  "gradient_flow.py": {
    "success": true,
    "score": 1.0,
    "score_unit": "fraction",
    "feedback":
      "[pass] model file exists: .../model.py\n"
      "[pass] model file compiles: Python file compiles\n"
      "[pass] gradient smoke test: "
      "trainable_parameters=3928 "
      "input_shape=(2, 3, 32, 32)\n",
    "feedback_image_paths": []
  },
  "train.py": {
    "success": true,
    "score": 0.01484375,
    "score_unit": "mAP50-95",
    "feedback":
      "backend=remote gpu=1 dataset=coco train_examples=512 "
      "test_examples=512 epochs=5 metric=mAP50-95 "
      "initial_score=0.0000 final_score=0.0148 "
      "torch_device=cuda gpu_peak_memory_mb=548 "
      "gpu_peak_utilization_percent=17 "
      "gpu_peak_power_watts=71.4 gpu_telemetry_samples=11",
    "feedback_image_paths": []
  },
  "synthesis_max78000.py": {
    "success": false,
    "score": 0.6969696969696969,
    "score_unit": "fraction",
    "feedback":
      "[pass] model file exists: .../model.py\n"
      "[pass] model file compiles when applicable: "
      "Python file compiles\n"
      "[pass] MAX78000 YAML exists: .../model.yaml\n"
      "[pass] MAX78000 YAML parses: valid YAML\n"
      "[pass] YAML contains architecture fields: "
      "found architecture-like keys\n"
      "[pass] sample input exists: .../sample_input.npy\n"
      "[pass] generated ai8x checkpoint from Python model: "
      ".../.generated/candidate.pth.tar\n"
```

```
      "[fail] ai8xize synthesis: ...",
    "feedback_image_paths": []
  }
}
```

F.2 Peak Current Minimization

Score-only (Sco). Sco exposes pass/fail and the normalized scalar score, but not the measured current, UART trace, or compiler/flasher diagnostics.


```
"checks": {
  "compile_max78000.py": {
    "success": true,
    "score": 1.0,
    "score_unit": "binary",
    "feedback": null,
    "feedback_image_paths": []
  },
  "flash_max78000.py": {
    "success": true,
    "score": 1.0,
    "score_unit": "binary",
    "feedback": null,
    "feedback_image_paths": []
  },
  "measure_max78000.py": {
    "success": true,
    "score": 0.98237,
    "score_unit": "metric_score(peak_uA)",
    "feedback": null,
    "feedback_image_paths": []
  }
}
```


The Doc o_t is identical since it only changes the initial artifact s_0 to include static documentation.

Hardware-in-the-loop (HIL). HIL changes the measurement check to include the raw current and UART Serial Output for debugging/profiling.


```
"measure_max78000.py": {
  "success": true,
  "score": 0.98237,
  "score_unit": "metric_score(peak_uA)",
  "feedback":
    "## Hardware Status\n"
    "capture_power=True, capture_serial=True, "
    "capture_thermal=False\n"
    "duration_ms=20000, optimization_metric=peak_uA, "
    "metric_value=21254.9\n\n"
    "## Measurement Results\n"
    "{\n"
    "  \"avg_uA\": 6254.12,\n"
    "  \"peak_uA\": 21254.87,\n"
    "  \"total_energy_j\": 0.4128,\n"
    "  \"power_samples\": 2000,\n"
    "  \"ppk_sample_rate_hz\": 100.0\n"
    "}\n\n"
    "## Firmware Behavior\n"
    "Checkpoint: FOUND\n"
    "Judge: PASS: UART is consistent with five "
    "live camera inference cycles and "
    "post-processing.\n\n"
    "## UART Serial Output\n"
    "[0000.000] boot\n"
    "[0002.001] camera init complete\n"
    "...\n"
    "[0014.935] firmware task complete checkpoint\n"
    "[0015.036] entering final sleep",
  "feedback_image_paths": []
}
```


The compile and flash checks only change to include the error message, if any, in the respective `feedback` subfields.

G MAX78000 MCU

The MAX78000 is a commercially available MCU with an integrated neural accelerator that has been widely adopted by the embedded systems community.

G.1 Model Size

We show how model depths (i.e., the number of layers) and channel dimensions affect model size and runtime characteristics on the MCU (Fig. 9). Contrary to intuition, peak current consumption shows minimal correlation with model size (Fig. 9b), remaining consistently below 30 mA regardless of size. Figure 9c-h show that total inference time and power consumption scale approximately linearly with model size.

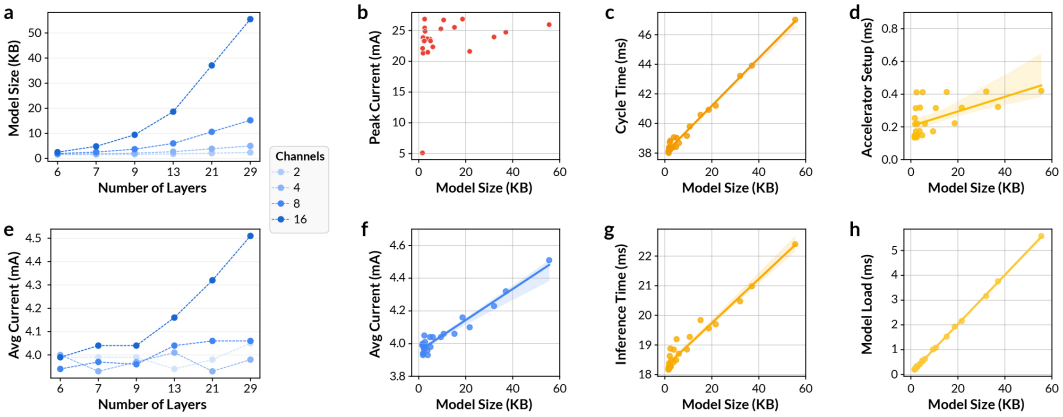


Figure 9: **Impact of model architecture on size and physical performance.** (a) Model size versus network depth for different channel widths, demonstrating that size scales with depth. (b) Peak current consumption shows minimal correlation with model size. (c-h) Total cycle time (c), comprising accelerator setup (d), model weight loading (h), and inference (g), as well as average current consumption during inference, all scale linearly with model size (N=10).

G.2 Clock Frequency

A critical configurable parameter of the MAX78000 is its clock source, which determines both the CNN accelerator's execution speed and power profile. The MCU supports two primary clock routing modes for the neural hardware:

- **System Clock Source:** The accelerator clock is derived from the main system clock via a hardware ÷2 divider. For instance, configuring a system clock of 60 MHz yields an effective CNN clock of 30 MHz.
- **Direct 60 MHz Source:** A dedicated 60 MHz oscillator is routed directly to the accelerator, bypassing the frequency divider. This is denoted as "60 (Dir)" in our evaluations.

As shown in Fig. 10, the selected clocking scheme creates a counterintuitive trade-off between instantaneous power and total energy. Operating the accelerator at higher frequencies (e.g., the direct 60 MHz source) predictably increases the peak and average instantaneous current. However, this is heavily offset by a dramatic reduction in computation latency. Because the inference cycle completes significantly faster, the total energy consumed per inference is actually minimized at higher clock speeds. This dynamic is a critical consideration for battery-free deployments: systems should ideally run at the maximum possible clock frequency to minimize total energy, provided the energy harvester and power management circuitry can sustain the higher instantaneous peak current demands.

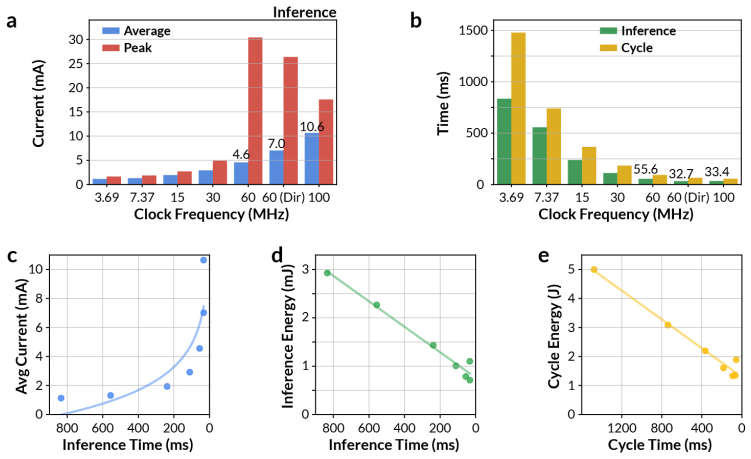


Figure 10: Impact of system clock frequency on model inference performance and energy consumption. (a, b) Average and peak current consumption (a) and inference and cycle time (b) at different system clock frequencies (N=10). Higher clock frequencies generally increase instantaneous current but decrease completion time. (c-e) Plots of average current (c), inference energy (d), and cycle energy (e) versus time demonstrate that faster completion times result in lower total energy consumption, despite higher instantaneous current. "60 (Dir)" denotes a 60 MHz clock set as the system clock and also applied directly to the CNN accelerator without frequency division. All other labels (including "60") represent the set system clock frequency, which is then divided by two to produce the final CNN clock. All measurements were performed on the same YOLO detection model.

H Wildlife Monitoring: Elk Detection

Camera traps are indispensable tools in ecology, wildlife research, and conservation, enabling non-invasive observation of animal behavior. However, deployment at scale is limited by cost, battery life, and maintenance requirements that become prohibitive in large or difficult-to-access locations [16].

We engaged with a commercial farmer in the Pacific Northwest managing multiple agricultural fields totaling thousands of acres, where elk frequently damage crops. The farmer had deployed five traditional camera trap units at a capital cost of $1,000 each, but three pain points limited scaling:

1. **Energy and Maintenance.** Traditional trail cameras require battery replacement every month under active use. For the farmer's deployment scenario with cameras distributed across large, remote fields, this represents substantial labor overhead. While some cameras near field edges could be connected to power via wires, this is infeasible for most monitoring locations far from electrical infrastructure.
2. **Communication Costs.** To enable real-time monitoring without physical visits, cellular-connected cameras transmit all captured images to cloud storage, currently incurring approximately $10 per camera per month. The farmer estimated that camera coverage every 200 feet would be needed to reliably detect elk near trails along field perimeters. For a single 200-acre field, this translates to 50 cameras and over $500 monthly communication costs, economically unsustainable when scaled across multiple fields.
3. **False Triggers.** The cameras produce large volumes of images containing no wildlife of interest, triggered by wind-blown vegetation or non-target animals (e.g., deer, birds). These false positives consume storage, bandwidth, and review time.

Battery-free model on MCUs directly addresses all three: 1) solar harvesting eliminates battery replacement, 2) local classification reduces cloud transmission for most images, and 3) accurate on-device inference filters false triggers before transmission.

We deployed our agentic pipeline to compress a YOLOv8 binary classification model trained on a curated elk dataset, achieving 96.7% accuracy within the MAX78000's memory and power constraints. Deployment challenges remain: the farmer's traditional camera units failed after a few months when moisture penetrated the enclosure, and developing weatherproof enclosures for our hardware is ongoing. Long-term reliability and return on investment relative to current practices require field validation, which will be reported in follow-up work.

H.1 Custom MCU Platform

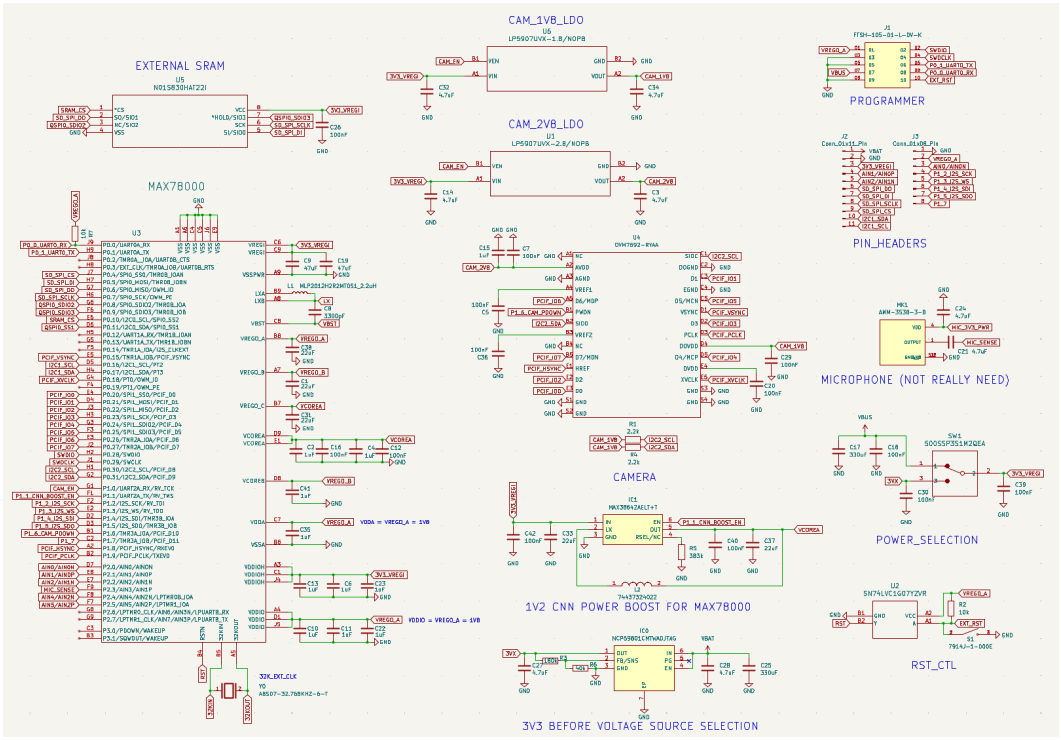


Figure 11: Schematic Design. The schematic comprises two primary blocks: a power module and a functional module. The functional module integrates the MAX78000 SoC, the OVM7692_RYAA camera, external SRAM, and a SWD interface for programming and debugging.

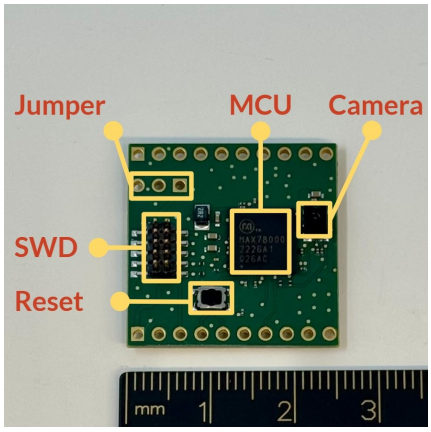


Figure 12: MAX78000 PCB Platform for wildlife monitoring.

H.2 Camera

The GC0308 camera module is interfaced with the MAX78000 via the Parallel Camera Interface (PCIF). The camera's power consumption and capture time depend on three key parameters: image resolution, PCIF frequency, and MCU system clock frequency. Frame rate can be set via duty cycle based on the application.

Resolution, however, affects both camera power consumption and vision model accuracy. Figure 13a-c shows frame acquisition time across different resolutions and PCIF frequencies. Notably, all resolutions we evaluated use a hardware-built-in downsampling from the camera's native output, which is an unavoidable feature given by the MSDK driver. The acquisition time is determined primarily by the native sensor resolution rather than the downsampled output size. For example, a 100×100 image downsampled from 400×400 takes longer than a 180×180 image downsampled from 360×360, because the camera sensor must still read and process the full array before downsampling. Average current consumption per frame (Fig. 13d-f) follows similar patterns.

Lower native resolutions reduce both capture time and total power.

Clock Frequency. Higher PCIF frequencies reduce frame transfer time from the camera to the MCU memory. With 60 MHz MCU system clock, at 10 MHz PCIF, a 180×180 frame requires 135.5 ms to capture and transfer, while at 5 MHz PCIF this doubles to 267.7 ms (Fig. 13a). However, higher PCIF frequencies slightly increase instantaneous current draw by 1.19 mA due to increased switching activity in the PCIF peripheral and I/O buffers.

Compared to the optimal 60 MHz system clock we found, reducing to 30 MHz decreases average current by 14.4% but doubles the capture and total cycle time, leading to higher overall energy consumption. At 7.37 MHz, cycle time balloons to over 1 second, a 7× increase, making real-time applications infeasible despite slightly lower instantaneous power. We note that the system clock affects DMA throughput and memory access latency during frame buffer transfers. Lower clocks create memory access bottlenecks that dramatically extend capture time, overwhelming any current reduction benefits.

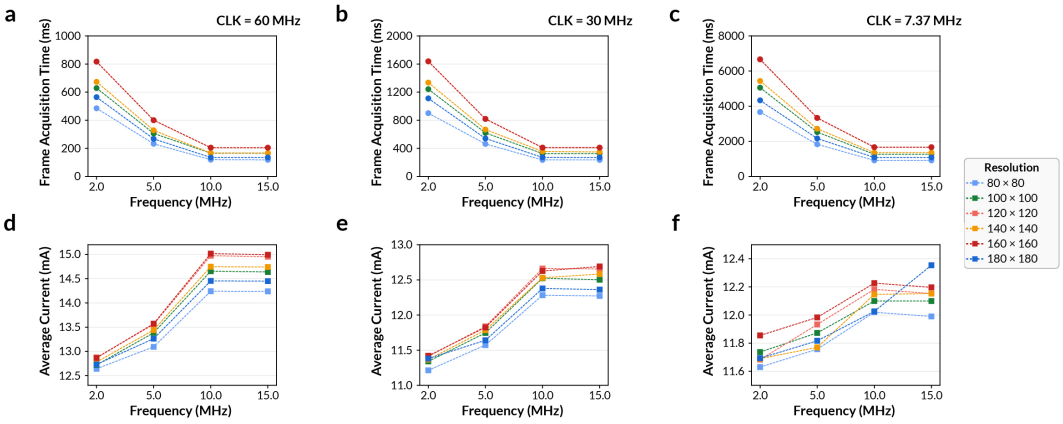
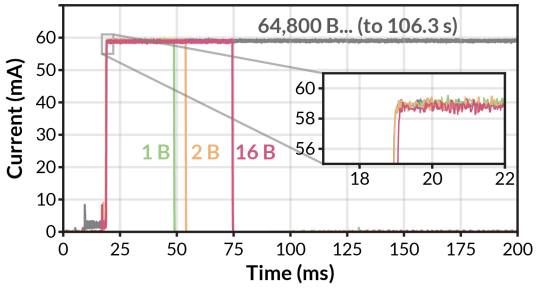


Figure 13: **Camera acquisition performance on MAX78000 across system clocks, PCIF frequencies, and image resolutions.** (a-c) Frame acquisition time (including camera capture and wake-up overhead) versus PCIF frequency. (d-f) Average current consumption per frame (N=10). Resolutions are hardware-downsampled from the camera's native output, demonstrating that metrics mainly depend on the native sensor resolution rather than the final downsampled output.

H.3 LoRa Communication

An SX1279 LoRa transceiver operated at 3.3 V is interfaced with the MAX78000 MCU via SPI.



Packet size	Tx time (ms)	$I_{tx,avg}$ (mA)	E_{tx} (mJ)
1 B	30.122 ± 0.013	58.968 ± 0.027	5.862
2 B	35.500 ± 0.012	59.148 ± 0.025	6.929
16 B	56.836 ± 0.011	59.302 ± 0.026	11.123
64,800 B	$106{,}321.2 \pm 0.837$	57.656 ± 0.027	20229.182

Figure 14: Current consumption profile during LoRA transmission for payload sizes of 1 B, 2 B, 16 B, and 64,800 B (N=10). The 64,800 B case represents a large data transfer segmented into multiple LoRA packets rather than a single-packet transmission. Peak transmission current remains below 60 mA across all payload sizes. Small payloads complete within 60 ms, while the 64,800 B transmission continues to 106.3 s (axis truncated for visualization).

H.3.1 SPI Interface Configuration and Debugging

The SPI connection was configured in Mode 0 at 1 MHz. A minimal custom driver was developed because initial integration attempts using the RadioLib library failed to establish communication. For connectivity verification, RegVersion (address 0x42; datasheet default 0x12) was read, but the initial readout deviated from expectation. Oscilloscope measurements of the clock (SCLK), chip select (CS), master output (MOSI), and master input (MISO) signals confirmed correct timing and frame formatting from the microcontroller while capturing the transceiver responses. Analysis revealed that the first byte returned after power-up or upon entering LoRa mode, the first byte returned by the first read transaction was consistently invalid, whereas all subsequent bytes were correct (0x12 for RegVersion). To address this hardware quirk, a warm-up procedure was added to the initialization sequence: a dummy read of RegVersion is performed immediately after reset or mode switching, with the first returned byte discarded before normal register operations begin. With this step in place, the register I/O and the TX/RX data path operated reliably and reproducibly.

H.3.2 LoRa Driver Architecture

The custom driver implemented four primary operational phases:

Initialization. Following a hardware reset, the transceiver entered LoRa Sleep mode, then transitioned to Standby. A warm-up dummy SPI read was performed with the first byte discarded.

Configuration. The radio was configured with a center frequency of 915 MHz. Modem parameters were set as follows: bandwidth (BW) = 125 kHz, coding rate (CR) = 4/5, spreading factor (SF) = 7, explicit header mode enabled, cyclic redundancy check (CRC) enabled, and preamble length = 12 symbols.

Transmission. From Standby state, the interrupt flags were cleared, the FIFO pointer was configured, payload data and length were written to the buffer, and the radio transitioned to TX mode. The system awaited the TxDone interrupt (with timeout protection), cleared status flags, and returned to Standby or Sleep mode.

Reception. Reception follows a similar state machine: flags are cleared, the receive pointer is reset, continuous or single receive mode is entered, the system waits for reception completion with timeout, payload integrity is verified via cyclic redundancy check, data is read from the FIFO buffer, flags are cleared, and the radio optionally enters Sleep mode.

H.3.3 Power Minimization for LoRa Driver

To balance energy consumption with long-range communication reliability, SF7 with BW = 125 kHz, CR = 4/5, and 12-symbol preamble was selected. At 125 kHz bandwidth, SF7 yielded a symbol duration of 1.024 ms, reducing time-on-air and per-message energy consumption compared to higher spreading factors. The CR = 4/5 configuration provided the minimum forward error correction overhead while maintaining robustness against typical outdoor channel impairments. The 12-symbol preamble ensured reliable automatic gain control, synchronization, and frequency offset compensation for multi-kilometer links with a fixed overhead of approximately $(12 + 4.25) \times 1.024\,\text{ms} \approx 16.6\,\text{ms}$ at SF7.

During operation, the transceiver remained in Sleep mode and transitioned through Standby to TX mode only when data transmission was required. Following each transmission, the radio immediately returned to Sleep mode. Average system current consumption outside transmission windows was measured at $<65\,\mu$A. The DIO0 pin was mapped to the TxDone interrupt, enabling interrupt-driven completion: the MCU entered sleep mode during transmission, the interrupt service routine set a completion flag upon TxDone assertion, and the main control loop subsequently cleared flags and returned the radio to Sleep mode. This approach eliminated polling overhead and minimized CPU active time.

The first-byte SPI read anomaly mitigation (single warm-up dummy read) is performed only once during initialization and not repeated in each transmission cycle, avoiding repetitive overhead on a per-packet basis.

Payload (B)	Driver	$I_{\text{tx,avg}}$ (mA)	I_{idle} (mA)	I_{cycle} (mA)	Tx time (ms)	E_{tx} (mJ)	E_{cycle} (mJ)
1	Ours	**58.968 ± 0.027**	**0.065± 0.004**	**1.768 ± 0.004**	**30.122 ± 0.013**	**5.862**	**5.994**
	Standard	59.866 ± 0.010 **(+1.5%)**	1.960 ± 0.000 **(+2938.8%)**	3.610 ± 0.000 **(+104.2%)**	29.950 ± 0.025 **(-0.6%)**	5.917 **(+0.9%)**	12.241 **(+104.2%)**
2	Ours	**59.148 ± 0.025**	**0.061 ± 0.003**	**2.050 ± 0.000**	**35.500 ± 0.012**	**6.929**	**6.984**
	Standard	59.870 ± 0.021 **(+1.2%)**	1.952 ± 0.004 **(+3079.2%)**	3.878 ± 0.007 **(+89.2%)**	35.050 ± 0.042 **(-1.3%)**	6.925 **(-0.1%)**	13.218 **(+89.2%)**
16	Ours	**59.302 ± 0.026**	**0.060± 0.004**	**3.210 ± 0.000**	**56.836 ± 0.011**	**11.123**	**11.158**
	Standard	59.938 ± 0.012 **(+1.1%)**	1.956 ± 0.005 **(+3160.0%)**	5.020 ± 0.006 **(+56.4%)**	55.258 ± 0.075 **(-2.8%)**	10.930 **(-1.7%)**	17.451 **(+56.4%)**

Table 20: Comparison of our optimized low-power versus standard LoRa driver implementations per transmission across different payload sizes (N=10). Percentages indicate relative difference from the low-power implementation.

I Early Language Development and Phoneme Perception

Understanding how young children acquire language is a central question in developmental science. To study early language and cognitive development, developmental scientists rely largely on longitudinal naturalistic data, including speech, vocalizations, gestures, and social interactions between children and caregivers. These data are often collected continuously and passively in real-world environments, enabling the capture of rich and spontaneous behaviors that are difficult to observe in laboratory settings. However, such continuous in-the-wild data collection introduces significant design challenges, requiring the joint optimization of battery life, storage footprint, and on-device analysis capabilities for specialized applications, while maintaining a lightweight, portable, and minimally obtrusive form factor during daily activities.

We collaborated with developmental researchers to understand real-world data collection needs and developed a lightweight, comfortable, and accessible head-worn device for capturing children's development data from an egocentric perspective. By jointly optimizing for power consumption, storage capacity, and thermal constraints, we enable continuous real-world data collection in naturalistic settings. This supports applications such as longitudinal observation of infant–caregiver interactions, phoneme perception in child-focused speech, and early intervention for delayed language acquisition.

As a preliminary demonstration, we present a 4-second audio snippet containing speech from a toddler and a caregiver, recorded with the wearable amid background noise from several children playing with toys. The audio snippet is de-identified, unpublished data provided by an external collaborator and collected under a university IRB-approved study. The toddler says *"I can do it for him"*, pauses briefly, and the caregiver responds *"good job"*. Fig. 16 compares the phonetic transcriptions produced by the full-size XLSR model (1153.75 MB) and our compressed model (2.80 MB, $412\times$ smaller). Both models correctly localize the two speech regions and the intervening pause. On the caregiver's adult speech, the compressed model performs comparably to XLSR. Both transcribe "job" as /tʃɑ/, and the compressed model's /kɔ/ for "good" is arguably closer to the canonical /gʊ/ than XLSR's /pʊ/, since /k/ shares the velar place of articulation of /g/. On the toddler's utterance, the gap widens considerably. XLSR recovers most of the sentence as /aɪ sʌ ju ɪ t fɔ m/, and its errors are attributable to child articulation: the onset of "can" becomes /s/ rather than /k/, as child stop consonants are often weakly released, and "do" becomes /ju/, reflecting a palatalized production in which the alveolar stop approaches a palatal glide. The compressed model, by contrast, detects only four phonemes (/aɪ ʌ sʌ/). It identifies "I" correctly but omits most of the remaining segments and substitutes /sʌ/ for "for."

The errors on the toddler's utterance share a common cause: the acoustics of a child's vocal tract differ from those of the adult speech on which both models were trained. Future work can therefore focus on fine-tuning with real child recordings collected by our deployed wearables. Future improvements can focus on fine-tuning using real child recordings collected by our deployed wearables. Such adaptation could support applications such as real-time pronunciation feedback, in which caregivers receive alerts about mispronounced phonemes and can provide immediate correction, an important capability for early language learning.

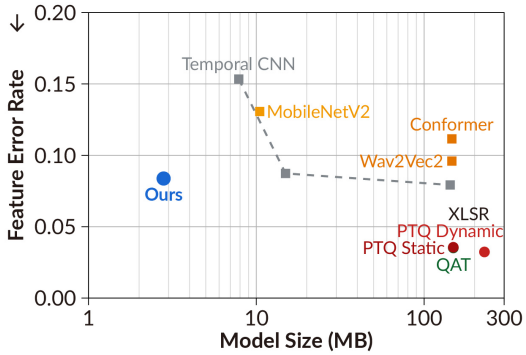


Figure 15: Comparison of our compressed models with standard compression baselines for speech-to-IPA transcription.

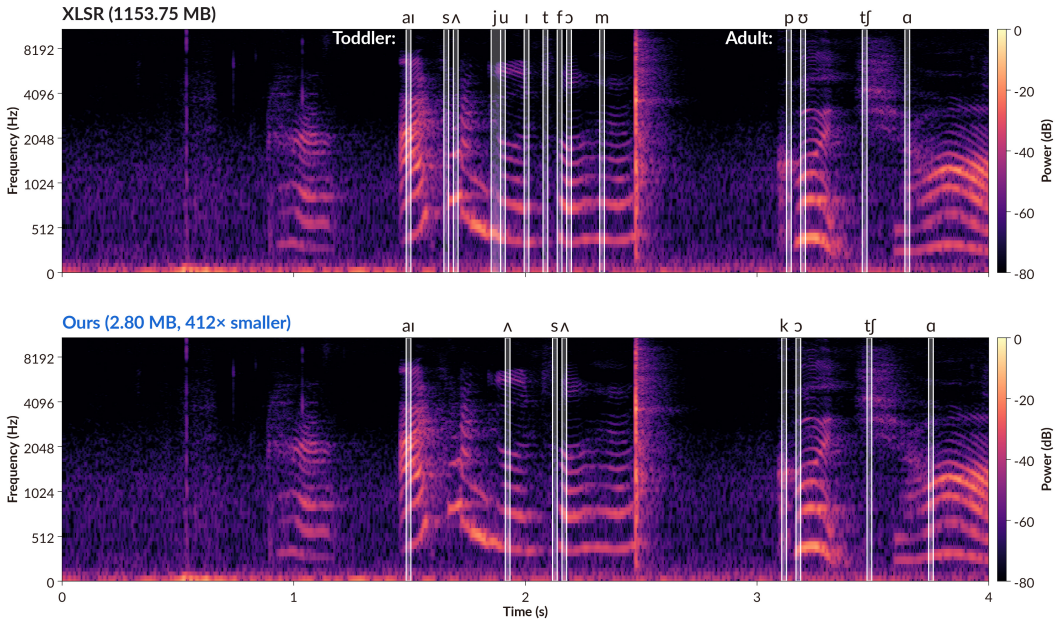


Figure 16: Log-mel spectrograms of a 4-second audio snippet of toddler and caregiver speech, overlaid with timestamped phonetic transcriptions from the full-size XLSR model and our compressed model. The toddler says *"I can do it for him"*, pauses briefly, and the caregiver responds *"good job"*.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification:

   Justification: Each claim in the abstract and introduction is supported in the paper: the arena formalization and platform in Section 2 and Fig. 2; iterative improvement under hardware feedback in Fig. 3; compression, power, and thermal results in Section 3 and the Appendix; and real-world deployments in Section 4 and the Appendix.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss limitations in Section 5.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: [N/A]

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We provide full experimental details. However, full reproduction requires physical hardware (MAX78000, STM32N6, ESP32-S3) and measurement instruments (Nordic PPK2, thermal camera), as results depend on real device behavior.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code is available at the link in the abstract, with instructions for reproducing the experiments.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Experimental details are provided in Section 4, along with additional detail throughout Appendix sections. Real world deployment details are discussed at length in Appendix F and G. Custom hardware details are provided in H.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We did not perform experiments of multiple random seeds due to resource constraints.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix C contains information on model training using A100s, the measurement setup and hardware in the loop devices are described in Section 4. Cloud compute used public API calls for Claude, GPT-5 and Gemini.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This work does not pose safety risks. This work did not collect new human subjects data, children audio data was provided by an external collaborator and de-identified before sharing. Data collect study was approved by university IRB.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Broader impacts are discussed briefly in section 6

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: [N/A]

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The work is original, uncompressed models input to the agent pipeline and datasets (e.g. YOLO) are cited

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: A hardware platform is demonstrated, documentation of the hardware is in Appendix H

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: We note that the children speech data was de-identified unpublished data collected through a university IRB approved study provided by an external collaborator.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [Yes]

    Justification: The child speech data used in this work are de-identified, unpublished recordings collected under a protocol approved by the UC San Diego Institutional Review Board (Protocol #811556).

    Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLM agents are the core subject of this work. Section 2 formalizes the agent loop, and Section 3 evaluates LLMs on model compression, power minimization, and thermal management.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.