MCSched: Memory-Controller-Aware Scheduling for Embodied LLM Workloads on NVIDIA Jetson

Teng Mei[1*] Chengxuan Pei[1*] Marco Canini[2] Yunming Xiao[1]

[1] The Chinese University of Hong Kong, Shenzhen [2] KAUST

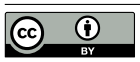
Abstract

Embodied LLM systems increasingly co-locate latency-critical robotics pipelines with compute- and memory-intensive language-model inference on edge platforms such as NVIDIA Jetson. This co-location avoids cloud round trips and enables privacy-preserving, low-latency interaction, but it also creates a new source of resource interference. Our experiments show that the deadline miss rate of a real-time task can reach nearly 100% under co-running LLM inference. The root cause is that, although robotics workloads and LLM inference typically execute on the CPU and GPU, respectively, they share the same unified-memory subsystem on edge platforms. As a result, Memory-Controller (MC) capacity can become a hidden bottleneck even when conventional compute resources do not appear saturated. Based on this finding, we propose *MCSched*, a lightweight MC-aware runtime scheduler for embodied LLM systems. *MCSched* combines coarse MC-pressure signals with foreground timing feedback and temporarily gates background LLM execution when robotics tasks approach deadline risk. In our prototype experiments, *MCSched* successfully reduced the deadline miss rate of the camera task and PCL task by up to 83.35% and 74.11%, respectively, on the ROS-LLM framework while maintaining LLM inference speed. These observations provide preliminary but compelling evidence that MC-aware scheduling is a practical operating system/runtime direction for deployable embodied AI on unified memory edge devices.

CCS Concepts

• **Computer systems organization → Real-time systems**; *Embedded systems*; *Heterogeneous (hybrid) systems*; Robotics.

*Teng Mei and Chengxuan Pei contributed equally to this research.



APSys '26, Bangkok, Thailand

ACM ISBN 979-8-4007-2912-6/26/09

https://doi.org/10.1145/3838177.3841726

Keywords

Memory-Controller contention, unified memory, real-time scheduling, embodied LLM systems, edge AI, ROS 2, NVIDIA Jetson, resource interference

ACM Reference Format:

Teng Mei, Chengxuan Pei, Marco Canini, and Yunming Xiao. 2026. *MCSched*: Memory-Controller-Aware Scheduling for Embodied LLM Workloads on NVIDIA Jetson. In *ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '26), September 17–18, 2026, Bangkok, Thailand*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3838177.3841726

1 Introduction

Recent embodied AI systems increasingly deploy VLA and LLM components on edge devices to support perception, instruction understanding, and command analysis [17, 28]. Edge deployment reduces cloud dependence and can improve privacy, power efficiency, and real-time responsiveness. NVIDIA Jetson platforms are a natural target for this deployment model because they combine CPU cores, integrated GPUs, accelerators, and unified memory in a compact power envelope. In these systems, however, LLM inference is not an isolated service. It co-runs with latency-sensitive robotics pipelines such as perception, localization, planning, and control, often implemented as Robot Operating System 2 (ROS 2) callbacks and executors [16]. Unlike latency in chatbot services, delays in sensor feedback or motor control can directly affect physical behavior, causing a robot to stall, drift, collide, or violate safety margins [5].

The same integration that makes Jetson attractive also creates a subtle interference channel. Unified memory simplifies data sharing between CPU and GPU work, but it couples those workloads through a shared subsystem [3]. LLM inference, particularly its prefill, decoding, and attention-heavy stages, can generate sustained memory traffic [7, 14]. In the meantime, robotics pipelines perform camera ingestion, image preprocessing, object detection, cost-map updates, control and real-time SLAM [33], all of which rely on bounded latency for correctness. Even when these tasks do not saturate CPU or GPU compute, they can miss deadlines if a co-running workload delays service in the unified-memory subsystem.

Existing embodied systems on edge devices tend to emphasize LLM-centric metrics such as token throughput and inference latency [27], while paying less attention to the deadline behavior of co-running robotics pipelines. These

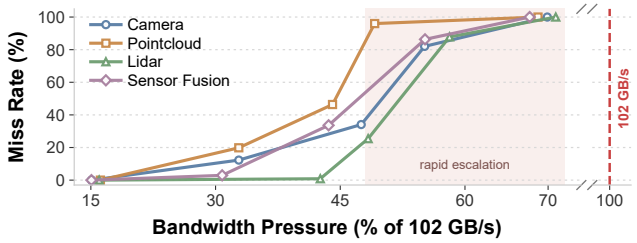


Figure 1: ROS 2 sensor tasks deadline miss rate under increasing MC bandwidth pressure with LLM inference workload.

metrics can indicate whether the LLM runtime is healthy, but they do not necessarily reveal whether the embodied system as a whole remains safe and responsive. To study this gap, we evaluate a representative ROS-LLM system on a Jetson platform [17] and measure the deadline miss rate of foreground real-time tasks, as shown in Figure 1. We evaluated four typical sensor tasks by controlling the inference speed of LLM (token throughput 50 %-100 %). Their deadline miss rates also increased when bandwidth pressure increased due to LLM inference, and they started to rise sharply at 45 % bandwidth utilization. Our measurements show that background LLM inference can raise MC pressure enough to sharply increase deadline misses in ROS sensor tasks. This effect is most visible for bandwidth-sensitive foreground tasks. And it suggests that the MC is not merely an implementation detail: it is a shared resource whose pressure must be considered in runtime scheduling.

Based on these findings, we propose *MCSched*, an MC-aware runtime prototype for scheduling embodied LLM workloads on Jetson-class platforms. The main contributions are as follows:

- We identify and characterize MC contention as a hidden failure mode for embodied LLM systems on unified-memory edge platforms. Across both ROS-LLM and non-ROS workloads on Jetson-class hardware, we show that LLM co-running can substantially inflate real-time task latency and deadline misses while leaving CPU/GPU utilization and LLM throughput metrics largely intact.
- We design and prototype *MCSched*, a lightweight MC-aware runtime scheduler that combines coarse MC-pressure signals with task-level timing feedback to detect foreground deadline risk and gate background LLM execution. The prototype illustrates a practical path toward deployable scheduling support for embodied LLMs on unified-memory edge platforms.

2 Background & Challenges

2.1 Background

Co-located Robotics and LLM Workloads. Due to strict size, weight, and power constraints and the need to avoid unpredictable cloud-offloading latency, modern mobile robots increasingly consolidate their software stacks onto a single edge platform [10]. These platforms must host latency-critical workloads such as sensor drivers, perception, localization, motion planning, and control. ROS 2 [22] provides the common communication and execution framework for such systems, where applications are organized as callback chains managed by executors; prior real-time studies have analyzed their response time, scheduling, and priority assignment [5, 23]. For these workloads, correctness depends not only on output quality but also on timeliness: a delayed perception result or control update can be equivalent to a failure and have serious consequences. Beyond conventional robotics callbacks, latency-sensitive processing on edge and end systems may also include CPU-side neural analytics; RET-Net, for example, uses a lightweight CNN for real-time network-flow classification and reports 0.201 ms per flow under serial inference on a general-purpose CPU [15].

Recent embodied LLM applications introduce a second workload class into the robotics stack. When grounded in robot skills or sensory observations, LLMs can support semantic planning, instruction following, and action generation [1, 4, 8]. Unlike low-level control loops with hard timing constraints, these LLM workloads are more temporally elastic: fast generation improves responsiveness, but modest delays in semantic reasoning or planning usually do not threaten immediate physical safety [32]. Depending on the application, the LLM may run continuously in the background, be triggered by user interaction, or be invoked by a high-level planner. Because both workloads are forced to co-locate on the same hardware, the LLM interacts with robotics workloads even when it is not directly inside the low-level control loop: LLM inference may consume GPU time, launch CPU runtime threads, allocate memory, and transfer data while ROS callbacks continue to process sensor and control events.

Unified Memory and MC Contention. Unified memory turns this software-level co-location into hardware-level contention. Instead of treating CPU memory and GPU memory as fully separate address spaces, Jetson-class SoCs allow CPU cores, the integrated GPU, and other accelerators to physically access the same unified-memory subsystem [20], as illustrated in Figure 2.

LLM inference can generate sustained memory traffic throughout both prefill and decoding. During prefill, transformer attention processes the input context; during decoding, inference repeatedly accesses model weights and KV-cache state, both of which are known to be memory-sensitive components in modern LLM serving [7, 14]. These accesses are arbitrated by the shared MC, whose activity is exposed through External Memory Controller (EMC) metrics such as bandwidth utilization and operating frequency [20, 21]. Consequently, contention is not limited to CPU or GPU compute resources alone. A robotics callback may miss its timing

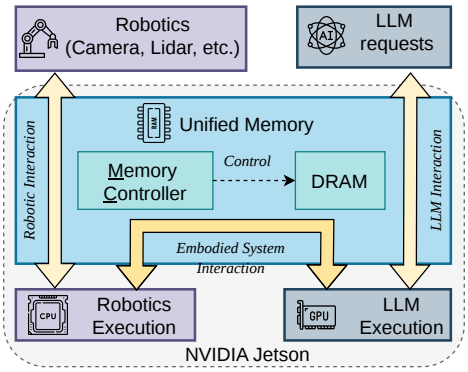


Figure 2: Unified-memory interference on Jetson. CPU-side ROS callbacks and GPU-side LLM inference issue memory requests to the same unified-memory subsystem. The memory controller can therefore become a shared bottleneck even when CPU and GPU compute resources are not saturated.

Table 1: Comparison of CPU stream-task latency under low and high GPU-side MC pressure. EMC denotes MC bandwidth utilization, and GFPO denotes GPU compute utilization. Latency is the elapsed time of one CPU task iteration.

Category	Metric	None	Low	High
Utilization	EMC	9.5%	10.0%	**64.3%**
	GFPO	0%	**76.3%**	0.3%
95% Latency	p95	20.2ms	20.4ms	**25.3ms**
	Increase	0.0%	0.8%	**25.0%**
99% Latency	p99	20.4ms	20.5ms	**25.6ms**
	Increase	0.0%	0.8%	**25.8%**

target because memory requests are delayed in the unified-memory subsystem, even while the LLM continues to achieve high GPU utilization and token throughput.

To isolate this effect, we measure the latency of a simple CPU stream task, representative of streaming ROS callbacks, while co-running GPU workloads with different memory behaviors. Table 1 shows that high GPU compute utilization alone has little impact on CPU-task tail latency: when GPU floating-point operations per second (GFPO) rises to 76.3% of the theoretical limit, EMC utilization remains near the idle level, and p95 latency increases by only 0.8%. In contrast, when EMC utilization rises to 64.3%, p95 and p99 latency increase by 25.0% and 25.8%, respectively. These results indicate that MC pressure, rather than GPU compute occupancy alone, can become the bottleneck for CPU-side robotics tasks. We therefore treat MC pressure as a cause-side signal that must be interpreted together with victim-side timing signals such as callback latency, jitter and deadline misses.

2.2 Challenges

The measurements above indicate that MC contention is an important contributor to deadline degradation in co-running ROS-LLM workloads. An intuitive response is to ease MC contention through scheduling: when foreground robotics tasks approach deadline risk, the runtime should reduce or defer memory-intensive LLM execution. However, scheduling around MC behavior is harder than scheduling CPU cores. The runtime must observe MC pressure without adding jitter, translate a non-partitionable hardware signal into manageable practical control actions, and decide when contention is actually harmful to bursty robotics SLOs. We elaborate on these challenges below.

Challenge #1: Memory-Controller Bandwidth Is Not Explicitly Schedulable. Existing real-time and robotic scheduling systems primarily reason about CPU-side resources, such as task priorities, executor behavior, core assignment, and processor utilization. Unified-memory bandwidth has been recognized as an important source of temporal interference in real-time systems [31], but it is rarely exposed as a first-class control knob in commodity robotic platforms. Unlike CPU cores, the MC cannot be directly partitioned or assigned to processes through standard scheduling interfaces. Similar resource-control limitations have also been observed in datacenter GPU fabrics and RDMA NICs [6, 12]. This creates a gap between observation and control: even when the runtime detects excessive MC pressure, it must translate that signal into indirect actions available on the platform.

Challenge #2: Robotics SLOs Are Bursty and Context-Dependent. Robotics SLOs capture end-to-end timing requirements for perception–control pipelines [25]. A ROS pipeline contains heterogeneous tasks with different timing sensitivities. Sensor processing is often latency-critical because it maintains the freshness of observations consumed by downstream planning and control stages [2, 30]; delayed sensor results can lead to stale observations and incorrect control decisions. However, not all robotics tasks are equally critical at all times: map refinement or global environment updates may tolerate moderate delay, while camera and radar perception can be highly sensitive to transient latency inflation caused by co-running LLM inference.

This bursty and context-dependent criticality makes static partitioning unattractive. Always reserving unified-memory bandwidth for robotics tasks unnecessarily suppresses LLM throughput, while reacting only to average MC utilization can miss short harmful contention windows. As a result, the scheduling problem is not simply to minimize MC pressure, but to decide when MC pressure is actually harming foreground timing.

3 Memory-Controller-Aware Runtime

MCSched is a lightweight runtime scheduler for embodied LLM systems that co-run foreground robotics tasks with background LLM inference on unified-memory edge platforms. Its design follows directly from the two challenges

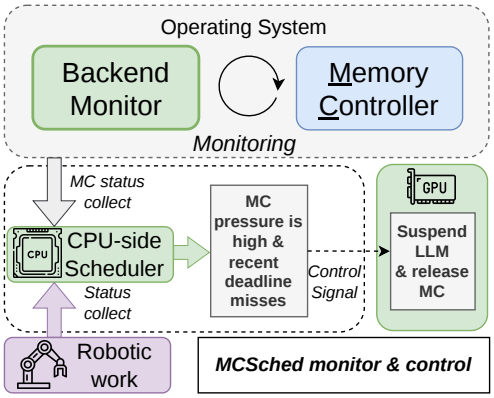


Figure 3: Architecture overview of *MCSched*

in § 2.2: MC bandwidth is difficult to schedule directly, and robotics SLOs are bursty and context-dependent. Rather than partitioning memory bandwidth statically or throttling the LLM whenever MC pressure is high, *MCSched* regulates the more elastic LLM workload only when foreground robotics tasks approach deadline risk.

At a high level, *MCSched* is built around two pieces: an MC-aware scheduling policy for foreground real-time tasks, and a minimal ROS-LLM integration layer that can slow or pause background inference during foreground-critical windows. *MCSched* does not assign an MC quota to a ROS callback or reserve a fixed fraction of unified-memory bandwidth for a control loop. Instead, it relies on coarse but practical control actions applied to LLM inference. These actions match the priority structure of embodied systems: foreground robotics tasks have tight timing constraints, whereas background LLM generation can tolerate short deferrals.

Use of AI tools. OpenAI Codex with was used to assist in writing benchmarking and plotting scripts for the experiments reported in this paper. The authors reviewed and tested all AI-assisted code and independently verified the experimental data, plots, and reported results.

3.1 Runtime Overview

The runtime follows three principles. First, the online control path must have low perturbation: it should not continuously run heavyweight profilers or require precise attribution of every memory transaction. Second, it should protect robotics tasks at callback or release boundaries, where deadlines are naturally defined. Third, the LLM should remain unmodified when possible, because deployed robots may use existing ROS and LLM runtimes.

As shown by Figure 3, *MCSched* separates scheduling policy from the source of measurement. Rather than introducing a new measurement subsystem, it consumes lightweight runtime signals that are already available to the robotics and LLM stacks: foreground task timing, recent deadline behavior, and coarse MC-pressure summaries. For each protected task, *MCSched* maintains a recent window summary over callback executions, including response time, waiting time, execution time, minimum slack, and deadline misses. In a ROS 2 deployment, these values can be collected by wrapping timer, subscription, or action callbacks at the executor or callback-dispatch layer, without modifying ROS 2 nodes. MC pressure is used only as a coarse cause-side signal, which can be obtained from a hardware-level real-time Activity Monitor, see § 3.3 for details. We evaluated this method of MC stress acquisition, and found that using this method only increases the tail latency of ROS tasks by 0.03-0.05%, which can be considered minimal interference. The scheduling decision is driven by whether foreground timing indicates that a robotics task is becoming vulnerable.

3.2 Scheduling Policy

Let P_W be a coarse MC-pressure signal in window W, M_W the recent miss rate of the protected task, and S_W its minimum observed slack. *MCSched* raises a risk signal only when the protected task shows victim-side timing stress: it has begun missing deadlines, its slack is nearly exhausted, or high MC pressure coincides with low slack. This separates a possible cause from an actual victim. MC pressure alone only indicates that the memory subsystem is busy; it does not imply that a robotics task is being delayed. Slack and deadline misses provide this victim-side evidence. Therefore, *MCSched* throttles LLM inference only when memory pressure coincides with signs that the protected robotics task is approaching, or has already entered, deadline risk.

MCSched supports two scheduling modes. ***Release-guard*** pauses the LLM shortly before a protected task release and resumes it after a configured protected interval. This mode is simple and effective when the task period or release time is known, such as for periodic sensing and control callbacks. ***Sensor-adaptive*** is more selective: it gates the LLM only when coarse MC pressure is high and recent foreground timing indicates low slack or deadline misses, making it suitable for event-driven pipelines whose criticality varies with sensor input and scene complexity. To avoid over-throttling, both modes use a compact release-window parameterization instead of separate budget counters. When throttling is triggered, Algorithm 1 suspends the LLM from $t_{\text{release}} - G$ to $t_{\text{release}} + H$, giving each throttling action a bounded pause quantum of $G + H$. Since the controller issues at most one suspension per task period T, the per-period pause budget is also bounded by $G + H$, and the LLM is guaranteed to run for at least $T - G - H$ before the next possible suspension, assuming $G + H < T$.

Algorithm 1 summarizes the sensor-adaptive mode used by *MCSched*; release-guard is its special case that omits the MC/risk condition. The controller first aligns with the next protected task release and enters a guard interval before

Algorithm 1 Phase-Aware Adaptive LLM Throttling

Input: Period T, guard G, hold H, tasks $\mathcal{P}$, thresholds θ_{mc}, θ_{slack}

1: $t_{release} \leftarrow$ Now() + T
2: **while** scheduler is active **do**
3: Sleep until $t_{release} - G$
4: $stage \leftarrow$ GetLLMStage()
5: $U_{mc} \leftarrow$ GetRecentMCUtilization()
6: $S_{min}, M_{rate} \leftarrow$ GetROSTelemetry($\mathcal{P}$)
7: $Risk \leftarrow (S_{min} < \theta_{slack}) \vee (M_{rate} > 0)$
8: **if** $stage \in$ ActiveStages && $U_{mc} > \theta_{mc}$ && $Risk$ **then**
9: SendSignal(LLM_PID, SIGSTOP)
10: Sleep until $t_{release}$
11: **if** $H > 0$ **then**
12: Sleep for H
13: SendSignal(LLM_PID, SIGCONT)
14: $t_{release} \leftarrow t_{release} + T$

the foreground-critical window (Lines 1–3). It then samples the LLM stage, recent MC pressure, and ROS-side timing summary to determine whether the foreground task is at risk (Lines 4–7). If the LLM is active and high MC pressure coincides with foreground timing risk, *MCSched* suspends background LLM execution and resumes it after the protected interval and optional hold duration (Lines 8–13). The controller then advances to the next release and repeats the procedure (Line 14). This policy combines a cause-side signal with victim-side timing feedback, avoiding unnecessary throttling when MC pressure is high but foreground tasks remain schedulable.

3.3 Implementation

Our current *MCSched* prototype follows the design constraints: it uses lightweight cause-side and victim-side signals, and integrates with existing ROS 2 and local LLM runtimes with minimal modification. On the robotics side, *MCSched* collects task-level timing metadata at callback boundaries, including release time, dispatch time, completion time, and deadline. In ROS 2, timer, subscription, and action callbacks already pass through executor-controlled dispatch paths; wrapping these boundaries is sufficient to estimate response time, slack, and deadline misses [22, 26]. This exposes the victim-side timing feedback needed by the scheduler without modifying perception, planning, or control kernels.

On the LLM side, the prototype regulates background inference through external process-group control. When *MCSched* detects foreground deadline risk, it suspends the LLM runtime with `SIGSTOP` and later resumes it with `SIGCONT`. This mechanism is coarse-grained, but portable across existing runtimes such as `llama.cpp`, and requires no changes to GPU kernels, model code, or inference libraries. The cause-side MC-pressure signal is obtained from Jetson's lightweight kernel-level interface, `ACTMON` [19], and summarized over a recent window. This avoids heavyweight profiling or hardware instrumentation in the online control path.

4 Preliminary Evaluation

Our preliminary evaluation investigates whether MC-aware control can reduce foreground deadline misses without fully suspending background LLM inference. Experiments are conducted on a Jetson Orin Nano Super [18].

It features a 6-core Arm CPU and an NVIDIA Ampere-architecture GPU, both integrated onto a single SoC. It has a theoretical bandwidth ceiling of 102 GB/s. The CPU and GPU share 8 GB of 128-bit LPDDR5 unified memory. We construct a mixed ROS-LLM workload comprising four representative robotics tasks and three LLMs. The workload captures diverse memory-access behaviors and evaluates whether *MCSched* can preserve the timing of bandwidth-sensitive sensor tasks (camera and PCL) without unnecessarily throttling LLM inference when mixed with less bandwidth-sensitive control and planner tasks. Our primary focus is on the sensor tasks. The camera task emulates high-resolution image preprocessing callbacks operating on full-frame floating-point tensors exceeding one million pixels [24]. The PCL task emulates irregular memory accesses in point-cloud filtering pipelines, performing approximately 1.6 million indexed gather operations over point attributes per callback [9].

Although the absolute deadline values are representative rather than derived from a certified robot safety case, they are configured according to the target sensor frequency (i.e., task period). For each job k of task τ_i, we define the expected release time $r_{i,k}$ and completion time $c_{i,k}$. The end-to-end response time is measured as $R_{i,k} = c_{i,k} - r_{i,k}$, which includes both release jitter and execution delay. A deadline miss occurs when $R_{i,k} > D_i$. Therefore, the reported miss rates capture the overall timing inflation experienced by latency-critical ROS tasks under co-running LLM interference.

Table 2 shows that the baseline ROS workload is schedulable, with zero deadline misses for both camera and PCL tasks. When co-running with LLM inference, however, uncontrolled execution causes severe interference: camera miss rates rise to 74.30–85.56%, while PCL reaches 85.71–96.61%, with p95 latency inflated to about 62–97 ms. *MCSched* significantly reduces this impact. Camera miss rates drop to 1.66–2.76%, with p95 latency restored to 28–30 ms; PCL miss rates decrease to 22.50–35.00%, with p95 latency reduced to about 46–47 ms. These improvements come at only 1.20–1.97% LLM throughput loss, showing that *MCSched* protects latency-critical sensor tasks while maintaining inference throughput as much as possible.

5 Discussion

Our current study has two main limitations. First, the evaluation is conducted on a single NVIDIA Jetson Orin Nano Super platform. Although unified-memory CPU–GPU architectures are common across edge devices, we still lack testing on non-NVIDIA Jetson platforms. The nature of memory controller contention may vary across hardware vendors;

Table 2: ROS sensor tasks deadline miss rates and tail latencies (p95) across different LLM workloads. *MCSched* effectively mitigates MC contention, achieving significant latency reductions with minimal LLM throughput loss.

Model	Sensor	✗ Uncontrolled		✓ *MCSched*		LLM tput drop
		Miss Rate	p95	Miss Rate	p95	
Baseline	Cam.	0.00%	25.44ms	–	–	–
	PCL	0.00%	39.86ms	–	–	
Qwen2.5 [29]	Cam.	85.56%	62.14ms	**2.21%**	**29.31ms**	1.20%
	PCL	85.71%	97.08ms	**35.00%**	**47.03ms**	
DeepSeek-R1 [11]	Cam.	84.44%	96.47ms	**2.76%**	**29.47ms**	1.97%
	PCL	86.44%	95.07ms	**33.61%**	**46.93ms**	
TinyMixtral -MOE [13]	Cam.	74.30%	94.72ms	**1.66%**	**28.46ms**	1.41%
	PCL	96.61%	93.87ms	**22.50%**	**45.90ms**	

while architectural analysis suggests such contention exists, further quantitative measurement is required. Furthermore, the system design does not account for other hardware platforms—for instance, many edge devices utilize NPUs for additional neural network acceleration, yet the invocation of NPUs is more difficult to intercept and control.

Second, MCSched is primarily intended for foreground workloads whose timing behavior is sensitive to memory-bandwidth contention. Our experiments focus on bandwidth-sensitive sensor-processing tasks, such as camera preprocessing and point-cloud processing, for which increased MC pressure can substantially inflate latency and cause deadline misses. For compute-bound, cache-resident, or otherwise bandwidth-insensitive foreground tasks, MC contention may contribute little to their execution latency, and reducing background LLM memory traffic would therefore provide limited benefit. We additionally evaluate four representative control workloads, including PID, LQR, computed-torque control, and MPC. In contrast to the bandwidth-sensitive camera and point-cloud workloads, these control tasks exhibit little degradation under the co-running GPU-heavy workload. Their execution-time p95 changes by at most 2.95 % in magnitude, while response-time p95 changes range from -2.08% to +5.89 %. Deadline misses remain negligible: three workloads observe no increase in miss rate, while computed-torque control reaches only 0.1 %. Moreover, the absolute response times remain far below their corresponding deadlines. These results indicate that high MC pressure does not uniformly translate into timing degradation; its impact strongly depends on the memory sensitivity of the foreground workload.

6 Conclusion & Future Work

Embodied LLM systems make edge robots more capable, but they also intensify contention on unified-memory platforms. This paper argues that MC pressure is a key source of real-time interference on Jetson devices and should be treated as a schedulable runtime concern. *MCSched* takes a deliberately lightweight approach: it combines coarse MC-pressure signals with foreground task timing, and gates background LLM execution only when robotics tasks are at risk. In our preliminary ROS-LLM evaluation, *MCSched* reduces camera tasks miss rate from 74–86% to 1.6–3% and PCL tasks miss rate from 85–97% to 22–35%, while retaining about 98% of unmanaged LLM throughput. These results suggest that memory-controller-aware scheduling is a worthwhile and practical direction for deployable embodied AI on unified-memory edge platforms.

We aim to explore three directions in the future. First, *MCSched* currently controls LLM execution through process-level suspend/resume, which is portable but coarse. Integrating cooperative runtime hooks, such as request-stage or token-group boundaries, could reduce MC pressure with finer granularity and lower throughput loss. Second, *MCSched* uses a threshold-based policy; future controllers should tune pause budgets online using foreground slack, recent miss history, MC-pressure trends, and LLM phase information. Finally, our evaluation is preliminary and limited to representative ROS-style tasks on one Jetson-class platform. Future work should study full robot applications, additional edge SoCs, different model sizes, and closed-loop quality metrics such as perception freshness, planning stability, control smoothness, and mission success.

7 Acknowledgment

We sincerely thank the anonymous reviewers for their insightful comments. Yunming Xiao is the corresponding author.

References

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[2] Rodrigo Aldana-López, Rosario Aragüés, and Carlos Sagüés. Latency vs precision: Stability preserving perception scheduling. *Automatica*, 155:111123, 2023.

[3] Waqar Ali and Heechul Yun. Protecting real-time gpu kernels on integrated cpu-gpu soc platforms. *arXiv preprint arXiv:1712.08738*, 2017.

[4] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023.

[5] Daniel Casini, Tobias Blaß, Ingo Lütkebohle, and Björn Brandenburg. Response-time analysis of ros 2 processing chains under reservation-based scheduling. In *31st Euromicro Conference on Real-Time Systems*, pages 1–23. Schloss Dagstuhl, 2019.

[6] Danyang Chen, Yufeng Gu, Yibo Huang, Chengxuan Pei, Peichun Hua, Yang Zhou, and Yunming Xiao. Unlocking software-defined gpu fabric scheduling in the llm era. In *Proceedings of the 17th ACM SIGOPS Asia-Pacific Workshop on Systems, APSys 2026, Bangkok, Thailand, September 17-18, 2026*. ACM, 2026.

[7] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

[8] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. PaLM-E: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

[9] Yu Feng, Gunnar Hammonds, Yiming Gan, and Yuhao Zhu. Crescent: taming memory irregularities for accelerating deep point cloud analytics. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 962–977, 2022.

[10] Miguel Á González-Santamarta, Francisco J Rodríguez-Lera, David Sobrín-Hidalgo, Ángel Manuel Guerrero-Higueras, and Vicente MatellÁn-Olivera. Integrating quantized llms into robotics systems as edge ai to leverage their natural language processing capabilities. *arXiv preprint arXiv:2506.09581*, 2025.

[11] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

[12] Yibo Huang, Yiming Qiu, Yunming Xiao, Archit Bhatnagar, Sylvia Ratnasamy, and Ang Chen. Exposing RDMA NIC resources for software-defined scheduling. In *Proceedings of the 9th Asia-Pacific Workshop on Networking, APNet 2025, Shanghai, China, August 7-8, 2025*, pages 1–8. ACM, 2025.

[13] Isotonic. Tinymixtral-4x248m-moe. Hugging Face model card, 2024. Accessed: 2026-05-18.

[14] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

[15] Zhe Li, Chengxuan Pei, Yanyue Xu, Sifan Hou, Onur Barut, Kun Qiu, and Jin Zhao. Ret-net: A cnn framework for real-time traffic classification using key-byte mechanism. *IEEE Transactions on Network and Service Management*, 2026.

[16] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science robotics*, 7(66):eabm6074, 2022.

[17] Christopher E Mower, Yuhui Wan, Hongzhan Yu, Antoine Grosnit, Jonas Gonzalez-Billandon, Matthieu Zimmer, Puze Liu, Daniel Palenicek, Davide Tateo, Jan Peters, et al. A robot operating system framework for using large language models in embodied ai. *Nature Machine Intelligence*, pages 1–13, 2026.

[18] NVIDIA. Jetson orin nano super developer kit. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/nano-super-developer-kit/. Accessed: 2026-05-18.

[19] NVIDIA. Jetson Orin Series: Platform Power and Performance. https://docs.nvidia.com/jetson/archives/r36.3/DeveloperGuide/SD/PlatformPowerAndPerformance/JetsonOrinNanoSeriesJetsonOrinNxSeriesAndJetsonAgxOrinSeries.html. Accessed 2026-05-19.

[20] NVIDIA. NVIDIA Jetson Orin Series. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/. Accessed: 2026-05-12.

[21] NVIDIA. Tegrastats Utility. https://docs.nvidia.com/jetson/archives/r36.5/DeveloperGuide/AT/JetsonLinuxDevelopmentTools/TegrastatsUtility.html. Accessed: 2026-05-12.

[22] Open Robotics. ROS 2 Documentation: Executors. https://docs.ros.org/en/humble/Concepts/Intermediate/About-Executors.html. Accessed: 2026-05-17.

[23] Yue Tang, Zhiwei Feng, Nan Guan, Xu Jiang, Mingsong Lv, Qingxu Deng, and Wang Yi. Response time analysis and priority assignment of processing chains on ros2 executors. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 231–243. IEEE, 2020.

[24] Zachary Teed and Jia Deng. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. *Advances in neural information processing systems*, 34:16558–16569, 2021.

[25] Harun Teper, Tobias Betz, Mario Günzel, Dominic Ebner, Georg Von Der Brüggen, Johannes Betz, and Jian-Jia Chen. End-to-end timing analysis and optimization of multi-executor ros 2 systems. In *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 212–224. IEEE, 2024.

[26] Harun Teper, Mario Günzel, Niklas Ueter, Georg von der Brüggen, and Jian-Jia Chen. End-to-end timing analysis in ROS2. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 53–65, 2022.

[27] Zhaode Wang, Jingbang Yang, Xinyu Qian, Shiwen Xing, Xiaotang Jiang, Chengfei Lv, and Shengyu Zhang. Mnn-llm: A generic inference engine for fast large language model deployment on mobile devices. In *Proceedings of the 6th ACM International Conference on Multimedia in Asia Workshops*, MMAsia '24, page 1–7. ACM, December 2024.

[28] Justin Williams, Kishor Datta Gupta, Roy George, and Mrinmoy Sarkar. Litevla-edge: Quantized on-device multimodal control for embedded robotics. *arXiv preprint arXiv:2603.03380*, 2026.

[29] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 Technical Report, 2025.

[30] Yanlei Ye, Zhenguo Nie, Xinjun Liu, Fugui Xie, Zihao Li, and Peng Li. Ros2 real-time performance optimization and evaluation. *Chinese journal of mechanical engineering*, 36(1):144, 2023.

[31] Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 55–64. IEEE, 2013.

[32] Wei Zhang, Zhiyu Wu, Yi Mu, Rui Ning, Banruo Liu, Nikhil Sarda, Myungjin Lee, and Fan Lai. JITServe: SLO-aware LLM serving with imprecise request information. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pages 825–848, Renton, WA, May 2026. USENIX Association.

[33] Fan Zhu, Yifan Zhao, Ziyu Chen, Peichen Liu, Hui Zhu, Chunmao Jiang, and Juyong Zhang. FGO-SLAM++: Real-time geometry-aware gaussian slam with continuous opacity field. *IEEE Transactions on Visualization and Computer Graphics*, 32(9):7982–7997, 2026.