fReeLoaders: An IoT Ecosystem for Real-Time Deadline-Driven Task Scheduling using Reinforcement Learning

Marshall Clyburn*
marshallc@virginia.edu
University of Virginia

Nabeel Nasir*
nabeeln@ucsb.edu
UC Santa Barbara

Md Fazlay Rabbi Masum Billah
frmasum@amazon.com
Amazon.com

Victor Ariel Leal Sobral
sobral@virginia.edu
University of Virginia

Jiechao Gao
jiechao@stanford.edu
Stanford University

Fateme Nikseresht
fn5an@virginia.edu
University of Virginia

Brad Campbell
bradjc@virginia.edu
University of Virginia

Abstract

As latency-sensitive IoT applications proliferate, edge computing becomes critical to sustaining real-time performance. Yet, limited edge infrastructure and reliance on costly static task profiling constrain its potential. This paper presents fReeLoaders, an IoT ecosystem addressing both challenges through opportunistic offloading and adaptive scheduling. fReeLoaders uses nearby idle smart devices to augment edge availability and uses a deadline-driven reinforcement learning scheduler to learn task behavior on the fly, eliminating expensive a priori profiling. Evaluation on real hardware shows it improves quality of service by 11.4% over a state-of-the-art profiling scheduler and adapts to dynamic workloads.

CCS Concepts

• **Software and its engineering** → **Scheduling**; • **Computer systems organization** → *Real-time systems*; • **Networks** → **Cyber-physical networks**; • **Computing methodologies** → **Reinforcement learning**.

Keywords

Task scheduling, Edge computing, Internet of Things, Reinforcement learning, Realtime systems, Task offloading

SEC '25, December 3–6, 2025, Arlington, VA, USA

ACM ISBN 979-8-4007-2238-7/2025/12
https://doi.org/10.1145/3769102.3770617

ACM Reference Format:
Marshall Clyburn*, Nabeel Nasir*, Md Fazlay Rabbi Masum Billah, Victor Ariel Leal Sobral, Jiechao Gao, Fateme Nikseresht, and Brad Campbell. 2025. fReeLoaders: An IoT Ecosystem for Real-Time Deadline-Driven Task Scheduling using Reinforcement Learning. In *The Tenth ACM/IEEE Symposium on Edge Computing (SEC '25), December 3–6, 2025, Arlington, VA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3769102.3770617

1 Introduction

Advanced Internet of Things (IoT) applications are often stymied by a lack of accessible compute near end devices. Applications, such as AR-based cognitive assistance [16] and smartwatch-based hand-wash detection [25], require low-latency processing that nearby edge resources, such as Cloudlets [29], can better provide than more distant cloud servers [34]. As IoT and AI deployments proliferate, edge computing becomes essential for sustaining real-time performance [5, 26, 32]. The key challenges lie in *scheduling*: selecting optimal edge resources based on QoS and network conditions [24, 35], and in *availability*: ensuring accessible edge services near devices [18, 35]. Inefficiencies in either dimension can significantly degrade performance and user experience.

These challenges remain largely unresolved. Existing work like HeteroEdge [37] and Kim et al.'s scheduler [19], use profiling to predict execution time and energy on edge machines. However, the heterogeneity of the IoT—diverse devices, workloads, and QoS requirements—makes profiling infeasible at scale. Each new or modified task necessitates re-profiling

*Both authors contributed equally to this paper.

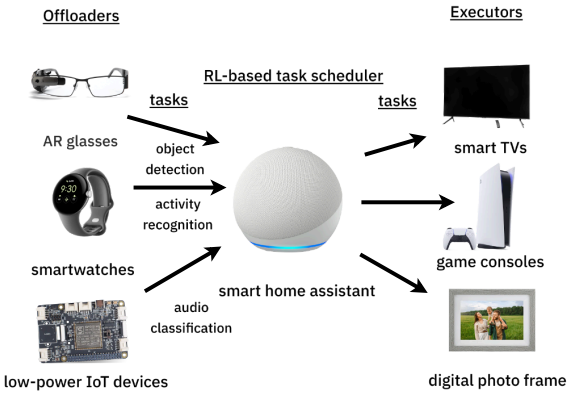


Figure 1: An overview of the fReeLoaders ecosystem.

across all edge nodes, incurring time and energy costs that hinder responsiveness. While such expensive profiling may be acceptable in static real-time systems like avionics, IoT workloads evolve rapidly, with tasks frequently added, updated, or migrated, rendering traditional schedulers ineffective. Furthermore, edge service deployment remains sparse. Many providers have scaled back [9, 22] or discontinued operations [12, 31], limiting coverage and accessibility [20, 21].

Our approach tackles the challenges of limited edge infrastructure and inadequate scheduling in heterogeneous, dynamic IoT environments through a two-pronged design. First, to reduce reliance on unreliable edge services, we propose offloading tasks from resource-constrained IoT devices (e.g., AR/VR headsets, smartwatches) to idle, wall-powered smart devices in the vicinity (e.g., smart TVs, gaming consoles). Such devices offer superior compute, long idle periods, and stable power sources, making them ideal for opportunistic task execution. Second, we introduce a reinforcement learning (RL)–based scheduler that learns task performance on available devices *on the fly*, eliminating the need for costly *a priori* profiling and enabling adaptation to runtime variability. We implement these ideas in fReeLoaders, an IoT ecosystem where constrained "offloader" devices send tasks to nearby "executor" devices, with a central "controller" (e.g., a Google Home) orchestrating the process (Figure 1). For example, a smartwatch performing hand-wash detection can leverage the GPU of a nearby PlayStation 5 [8]. Leveraging compute in executors eliminates the need for dedicated infrastructure to handle offloaded tasks.

While prior work explores everyday devices as offloading targets [11, 19, 33, 37], our contribution lies in *eliminating the need for costly task profiling* through a "deadline-driven", RL–based scheduler. Traditional real-time schedulers estimate worst-case execution time (WCET) or CPU demand via profiling [15, 19, 37], incurring significant time and energy overhead—an issue we quantify in this paper. We replace *a priori* profiling with a scheduler that learns task behavior *on the fly*, aiming for deadline satisfaction rather than explicit WCET estimation. We leverage the soft real-time nature of IoT systems [13], allowing the scheduler to tolerate early deadline misses and learn from repeated task executions. The approach converges quickly and surpasses static profiling-based methods on deadline satisfaction and energy usage while also adapting to changing devices, workloads, and network conditions.

We demonstrate a prototype implementation and evaluate its performance, adaptability, and overhead with a testbed consisting of hardware matching the compute of existing smart devices. We compare fReeLoaders against a profiling scheduler, an oracle offline scheduler, and common scheduling strategies. fReeLoaders improves QoS by 11.4% over the state-of-the-art *a priori* scheduler, adapts to changing executor host loads and the availability of new hosts, and can run on hardware matching a commodity smart device. These results show that fReeLoaders eliminates profiling overhead and harnesses idle smart device compute to enable low-latency, energy-efficient applications while addressing core scheduling and availability challenges.

2 Related Work

Real-time Scheduling. There are existing schedulers designed to minimize energy usage and meet QoS requirements. CoGTA [36] allocates social sensing tasks to cooperative edge nodes, HeteroEdge [37] addresses edge device heterogeneity with a uniform resource management interface, and HEROS [15] is a heuristic-based scheduler for heterogeneous resources assigning workloads to capable yet energy-efficient nodes. Kim et al. [19] provide a cooperative scheduler that integrates with the OS to assign tasks to idle IoT devices. However, these approaches rely on *a priori* task profiling. fReeLoaders overcomes this with a deadline-driven RL-based scheduler that learns task performance over time and enables adaptive, heterogeneity-aware scheduling.

RL-based Scheduling for the Edge. RL-based scheduling has emerged as a solution to edge computing workload and hardware heterogeneity. Tang et al. [30], Chen et al. [7], Lu et al. [23], and Alfaikh et al. [6] use variations of RL (e.g., deep Q-learning, MADDPG, RL-SARSA) to offload to MEC servers or cloud data centers. However, these approaches require *a priori* profiling, thus not adapting well to new workloads. They also assume constant computing capacity and evaluate in simulated environments, making it difficult to account for uncertainty in real-world scenarios. Contrastingly, fReeLoaders dynamically estimates workloads and computing capacity without a priori profiling using deep Q-learning. We also validate the system on a real-world testbed.

3 System Design

We design fReeLoaders to leverage heterogeneous compute. The ecosystem benefits *offloaders* which send workloads to *executors*. A *controller* mediates with an RL-based scheduler.

3.1 Reinforcement Learning Scheduler

The scheduler must navigate uncertainties affecting workload execution time and energy usage. There are many kinds of workloads it must handle, as there are many kinds of devices that may offload. Also, the static and dynamic characteristics of the compute available to run workloads is wide-ranging. This overwhelming diversity alongside ever-changing compute load state, the effect of concurrent workloads on performance, updates introducing new and changing workloads, and more, make profiling intractable. Instead, because we expect these workloads to repeat, we leverage RL to address these challenges. An RL agent will have ample opportunity to explore how each workload runs on the ecosystem's compute, and continually learn to meet deadlines and optimize energy usage.

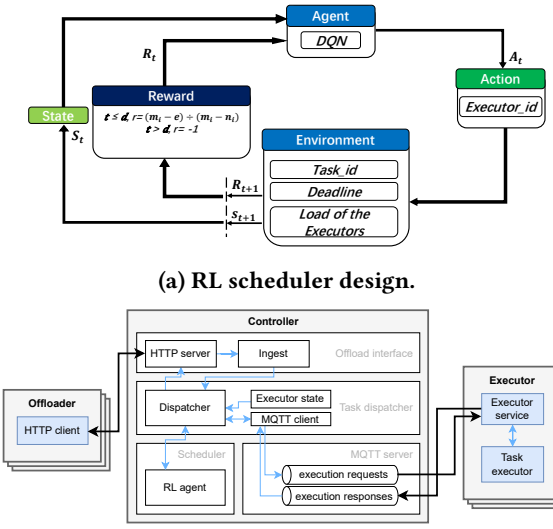
Meeting deadlines requires knowing one type of workload from others, current compute resource load, and workload deadlines. Each type of workload will have different compute requirements, necessitating an identifier. The deadline reflects the workload's urgency. The load state of each compute resource informs which resource has computing capacity. We provide this information to the scheduler, and it selects a compute resource to execute the workload.

The RL agent gets feedback based on deadline satisfaction and energy usage (Figure 2a). We prioritize deadline satisfaction to meet application QoS expectations and secondarily aim for low energy usage. Owners pay the cost of energy usage, making energy efficiency a priority. The reward function yields a positive reward (r) between 0 and 1 when meeting the deadline ($t \leq d$) and a -1 reward otherwise. The reward is non-stationary. It depends on where the energy usage of the latest instance of workload i's execution (e) compared to the most energy used (m_i) and the least energy used (n_i) executing the same workload. Because tasks can exhibit a wide range of energy usage based on their execution time, we track this on a per-task basis. Mathematically:

$$r = \begin{cases} (m_i - e) \div (m_i - n_i) & t \leq d \\ -1 & t > d \end{cases}$$

3.2 The fReeLoaders Ecosystem

The fReeLoaders ecosystem consists of *offloaders* which send *tasks* to *executors* through a mediating *controller*. **Tasks** are the unit of offloaded work in the ecosystem. Their code exists on both offloading devices and executing hosts. **Offloaders**



(a) RL scheduler design.

(b) The fReeLoaders ecosystem implementation.

Figure 2: An overview of fReeLoaders.

are constrained devices that benefit from offloading tasks. Such devices include smartwatches processing fitness data or sound systems doing speech recognition. **Executors** run tasks on behalf of offloaders. They are more computationally capable than offloaders and have stable energy sources. For example, smart home assistants, game consoles, and smart TVs. The **controller** accepts tasks from offloaders and uses the RL-based scheduler to assign them to executors.

4 Implementation

The controller and executors run programs to enable task execution. The controller exposes an server accepting tasks and assigns them to executors using the scheduler. The controller has a pluggable design, making it easy to swap the scheduler in use. Executors provide load state (via psutil [28]) and execute tasks as child processes with low priority.

Once an offloader submits a task, the controller and executor handle the rest of the process. The controller accepts the task, fetches state from executors, and uses the scheduler to decide which executor will run the task. The controller sends the task to the executor via MQTT, and the executor executes the task immediately. Upon completion or failure, the executor responds to the controller, providing execution time and energy usage of the task, as well as its post-execution load state. Figure 2b gives an overview of the implementation.

We use Chainer [1] to implement the Deep Q-Learning RL model. It consists of 43 inputs (32 for the vectorized task ID, one for the deadline, and ten for executor loads), three hidden layers, and ten outputs, each corresponding to an

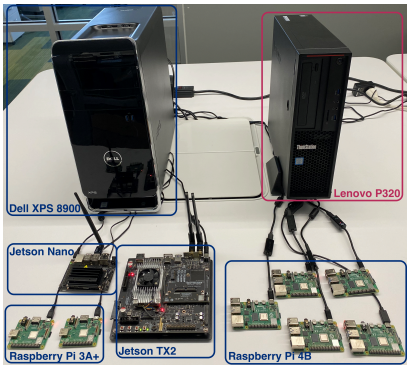


Figure 3: Evaluation devices (controller in red).

Device	CPU	RAM
Lenovo ThinkStation P320	Intel Core i7-7700	16 GB
Dell XPS 8900	Intel Core i7-6700K	16 GB
NVIDIA Jetson TX2	ARM Cortex-A57	8 GB
NVIDIA Jetson Nano	ARM Cortex-A57	4 GB
Raspberry Pi 4B (×5)	ARM Cortex-A72	4 GB
Raspberry Pi 3A+ (×2)	ARM Cortex-A53	4 GB

Table 1: Specifications of evaluation devices.

Task Information			Met Deadline on Executors?				
Task Type	ID	Deadline	Nano	Pi3	Pi4	TX2	PC
Loop	0	890 ms	✓	✓	✓	✓	✓
	9	5,112 ms	✓	✓	✓	✓	✓
Matrix Mult.	10	1,614 ms	✗	✗	✓	✓	✓
	19	9,859 ms	✗	✗	✓	✓	✓
FFT	20	1,189 ms	✗	✓	✓	✗	✓
	29	7,411 ms	✗	✗	✓	✓	✓
Hum. Act. Rec.	30	807 ms	✗	✗	✓	✗	✓
	39	3,873 ms	✗	✗	✗	✗	✓
Obj. Det.	40	5,709 ms	✗	✗	✗	✗	✓
	49	6,763 ms	✗	✗	✗	✗	✓
Room Class.	50	4,829 ms	✓	✓	✓	✓	✓
	59	7,949 ms	✓	✓	✓	✓	✓

Table 2: Evaluation tasks (only lightest and heaviest).

executor. We configure the model to train for five epochs using a batch of ten samples randomly sampled from a replay memory of the past 50 experiences and a discount rate of 0.1. Action selection uses a softmax temperature of 0.25.

5 Evaluation

We compare fReeLoaders with a variety of other schedulers and also test its adaptability and overheads.

5.1 Evaluation Setup

The evaluation uses a real-world testbed. The controller runs on a desktop PC. We submit tasks with a script to control experiment conditions. For each evaluation, we randomly offload the same sequence of tasks at a rate of 15 tasks per minute. All executors connect via Wi-Fi (Figure 3, Table 1). While using commercial smart devices is infeasible, this hardware provides a good match and presents heterogeneity in CPU architecture, computing power, and network speeds.

We group the schedulers in our evaluation into three categories. Static, non-profiling schedulers: **random** randomly assigns tasks, **load balancer** assigns a task to the least-loaded executor, and **performance-weighted round-robin** sequentially assigns a fixed number of tasks to executors relative to their Geekbench [14] score. Feedback-based, non-profiling schedulers: **energy-priority** starts with the most efficient executor and then tries more powerful ones, **performance-priority** starts with the most capable executor and then tries more energy-efficient ones, and **fReeLoaders** which uses the RL model to schedule, starting with an untrained model for each experiment. The profiling **HEROS** [15] scheduler uses the authors' parameter recommendations ($\alpha = 110$, $\beta = 0.9$, and $\gamma = 1.2$) and profiling data collected with perf [2]. We also compare against the profiling **HeteroEdge** [37] scheduler. We exclude the scheduler by Kim et al. [19], as it must run as an OS scheduler.

We use six types of tasks (Table 2) with ten variants, for a total of 60 tasks based on real workloads. **Loop** computes a sum in a loop, **matrix multiplication** multiplies two matrices, **Fast Fourier Transform** computes an FFT, **human activity recognition** uses a neural network for gesture recognition, **object detection** uses YOLO [27] to classify objects in an image, and **room classification** uses SqueezeNet [17] to classify a room. We classify the tasks into categories: "lax" (loop, room classification), "average" (FFT, matrix multiplication), and "strict" (human activity recognition, object detection), depending on the context the task would run in and use this to set contextually-appropriate deadlines.

We track energy usage with per-device power curves derived from measurements at several load values. Executors track usage during each experiment to model energy usage.

5.2 Comparison to a Profiling Scheduler

We compare the performance of fReeLoaders to HEROS using profiling data collected under different scenarios. We submit the same random sequence of 4,000 tasks. Figure 4 show DSR for the first 500 offloads, after which DSR plateaus. We provide HEROS with profiling data from several representative profiling scenarios. **TX2/PC task profiles** uses task profiling data from a single executor, matching an application developer profiling their application but not having access to all

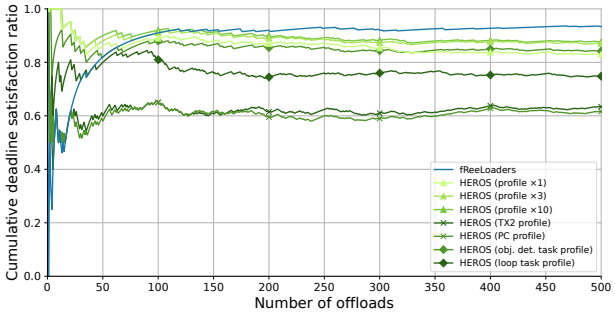
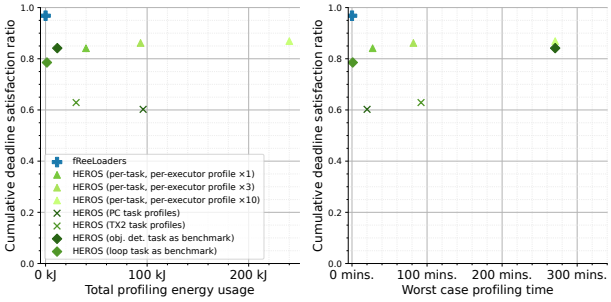


Figure 4: DSR of fReeLoaders and HEROS. fReeLoaders learns over time how to assign the 60 task types.



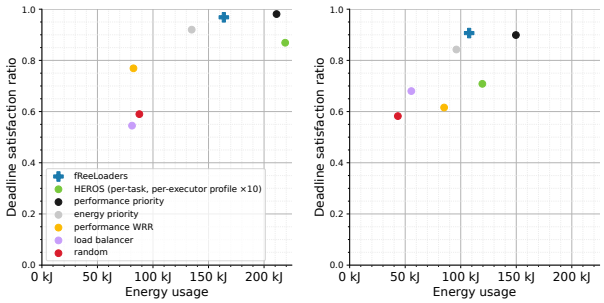
(a) Profiling energy vs. DSR **(b) Profiling time vs. DSR**

Figure 5: Energy and time costs of profiling approaches.

possible hardware. **Loop/obj. det. task as HW benchmark** benchmarks executor performance with a single task, matching a hardware designer providing performance data on a fixed workload. **Per-task, per-executor profile ×N** uses per-task, per-executor profiling data collected over N runs of each task on each executor, matching in-situ profiling.

The fReeLoaders scheduler learns and improves over all profiling variants by 150 offloads, showing that it does adapt to meet task deadlines. It also optimizes for energy usage, Only "PC profile" and "TX2 profile" consume less energy. However, their DSRs (60.24%, 62.83%) are not competitive with fReeLoaders (96.82%). The best-performing HEROS variants achieve competitive DSRs at the expense of higher runtime energy (obj. det. benchmark, 84.15%) or higher profiling energy cost (profile ×10, 86.9%).

Using a deadline-driven RL-based scheduler rather than devising evermore complex profiling is effective. HEROS' profiling data lacks important sources of variability, such as the effect of concurrent workloads. MIPS on the TX2 running the matrix task falls 13.7% when run with the object detection task. Such impacts cause its DSR to suffer. The fReeLoaders scheduler learns on-the-fly and outperforms an existing state-of-the-art scheduler in DSR while also optimizing energy usage (Figure 5). This makes it suitable for handling IoT workloads in an ecosystem subject to changing workloads.



(a) Dedicated compute. **(b) Non-dedicated compute.**

Figure 6: Energy usage vs. DSR of schedulers.

device	role	load profile
Nano	smart doorbell	ephemeral, 4 / 75 offloads
Pi 3	air quality sensor	base load, 25%
Pi 3	digital frame	base load, 5%
Pi 4	smart TV	2 sessions, 166 offloads, 70%
Pi 4	robot vacuum	1 session, 125 offloads, 100%
Pi 4	router	base load, 75%
Pi 4	smart camera	ephemeral, 8 / 240 offloads
Pi 4	set-top box	2 sessions, 167 offloads, 70%
TX2	voice assistant	ephemeral, 8 / 240 offloads
PC	game console	1 session, 100%

Table 3: Executor dynamic load profiles.

5.3 Performance with Dedicated Executors

Next, we compare the fReeLoaders scheduler and an ensemble of common schedulers. Executors only run the assigned tasks. We offload the same random sequence of 4,000 tasks to each scheduler and show results in Figure 6a. The fReeLoaders scheduler (96.8%), performance-priority (98.1%), and energy-priority (92.0%) attain the best DSR. Performance-priority performs slightly better but consumes much more energy than the fReeLoaders scheduler. While energy-priority uses less energy, it is slow to improve and underperforms the fReeLoaders scheduler.

5.4 Comparison to an Oracle Scheduler

We evaluate fReeLoaders against an oracle scheduler based on HeteroEdge [37], which uses *a priori* profiling and offline optimization. HeteroEdge models dependent task sets, requiring a separate evaluation. We use a Personalized Fitness Coaching (PFC) app with three sequential tasks (HAR, FFT, Object detection) executing across ten devices. The oracle collects 300 profiling samples (10 offloads × 10 executors × 3 tasks). We test fReeLoaders with i) FL(0): no training, ii) FL(30): 30 samples, iii) FL(150): 150 samples, and iv) FL(300): full profiling equivalent. We measure DSR for parallel PFC

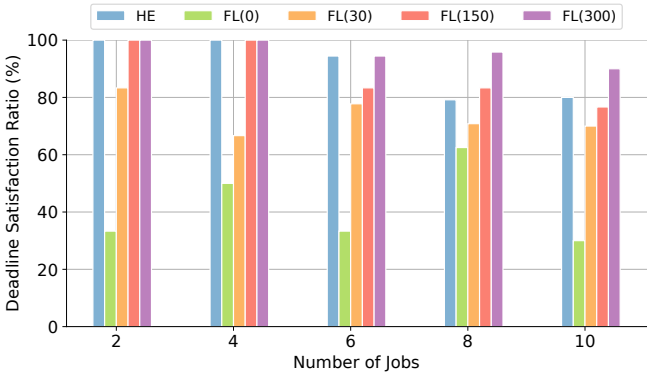


Figure 7: DSR for jobs of the PFC application. HeteroEdge has 300 measurements of profiling fReeLoaders is trained with 0, 30, 150, and 300 measurements.

jobs (Figure 7) and find that fReeLoaders quickly learns and outperforms HeteroEdge, adapting well under higher load.

5.5 Adaptability of the RL Scheduler

We evaluate fReeLoaders in scenarios of change to gauge its suitability in leveraging inconstant, hyper-local resources.

5.5.1 Dynamically loaded executors. We evaluate fReeLoaders's adaptability to dynamic loads of non-dedicated compute by configuring executors to exhibit the load of common devices (Table 3). We perform 4,000 offloads and show the results in Figure 6b. All executors exhibit poor performance at points during the experiment, so statically relying on a single executor is not possible. The best possible DSR is 93.3%.

The fReeLoaders scheduler achieves 90.67%, outperforming performance-priority (89.9%) and using 28.1% less energy. It outperforms energy-priority (84.6%) by finding sufficient executors more quickly. This evaluation shows that the fReeLoaders is able to adapt to these dynamics and still optimize energy usage.

5.5.2 Adding executors. Over time, the user will add devices. The scheduler must leverage these additions to improve performance. We test the fReeLoaders scheduler's adaptability by incrementally adding executors. We offload the same sequence of 2,000 tasks in four configurations, totaling to 8,000 offloads. The first configuration is a single Raspberry Pi 4; then we add four more Raspberry Pi 4s; then we add the Nano, TX2, and Raspberry Pi 3s; then we add the PC. We represent a non-existent executor by fixing the load input to 1 and failing a task if the scheduler assigns a task to it.

The fReeLoaders scheduler improves DSR with each change. Adding four Raspberry Pi 4s allows improvement from 45.5% to 61.2%, having all devices except the PC gives a small improvement to 63.8%, and adding the PC allows improvement to 93.5%. This shows that the fReeLoaders scheduler can adapt to new compute resources to improve its performance.

5.6 Controller Resource Overhead

We run the fReeLoaders controller on the Nano to observe the feasibility of running the controller on commodity hardware. After ten offloads we record the controller consistently uses 236 MB of memory. During the offload session, which includes a single round of training, CPU usage averages 11.10%. This demonstrates that this unoptimized prototype would run on devices with 512 MB of RAM, though many home assistants have 1 GB or more [10].

6 Discussion

Applicability. Building cross-vendor device ecosystems is difficult due to vendor restrictions and interoperability issues. But, single-vendor ecosystems (e.g., Samsung, Apple) make this feasible, enhancing user experience as each device adds computing power. Also, Thread [4] and Matter [3] standards can advance cross-vendor interoperability and adoption.

Security and Privacy. fReeLoaders lets users keep data on devices they control, but further controls would enable users to limit the executors and offloaders they trust. Linking devices would allow them to authenticate each other and limit which devices handle the user's data. Input data can remain private from the controller in this way. Multi-tenant setups would require a revised model or multiple models.

Scalability. Executor loads are inputs to the RL model. It is possible to pre-allocate inputs and outputs (Section 5.5.2) but increasing the inputs and outputs requires a new model. The system architecture and the expectation that all executors install task code makes scaling to large numbers of executors or tasks difficult. Further research is necessary to make scaling sustainable; for example, determining at install-time a subset of executors to install task code.

7 Conclusion

The IoT is replete with applications demanding low latency. As the IoT expands, devices supporting these applications will find their way into new environments, making cloud or edge computing support less practical and requiring that we find ways to support them regardless of location. fReeLoaders shows how we can leverage hyper-local compute to support constrained devices. The scheduler adapts to inconstant computing resources and workloads, forgoing profiling overheads. It also sets the foundation for building ecosystems of devices that interoperate to enable new applications across the domains that the IoT supports.

Acknowledgments

We thank Ethan Blaser for his review of the RL model and thank Marco Brocanelli for his shepherding assistance. This work is supported by the National Science Foundation under Grant No. 2144940.

References

[1] 2023. *Chainer*. Retrieved Jan 20, 2023 from https://chainer.org/

[2] 2024. *perf: Linux profiling with performance counters*. Retrieved December 5, 2024 from https://perfwiki.github.io/main/

[3] 2025. *Build with Matter (Connectivity Standards Alliance)*. Retrieved October 19, 2025 from https://csa-iot.org/all-solutions/matter/

[4] 2025. *Thread*. Retrieved October 19, 2025 from https://threadgroup.org/

[5] 3GPP. 2022. *5G System Overview*. Retrieved Dec 1, 2024 from https://www.3gpp.org/technologies/5g-system-overview

[6] Taha Alfakih, Mohammad Mehedi Hassan, Abdu Gumaei, Claudio Savaglio, and Giancarlo Fortino. 2020. Task Offloading and Resource Allocation for Mobile Edge Computing by Deep Reinforcement Learning Based on SARSA. *IEEE Access* 8 (2020), 54074–54084. doi:10.1109/ACCESS.2020.2981434

[7] Xing Chen and Guizhong Liu. 2021. Energy-Efficient Task Offloading and Resource Allocation via Deep Reinforcement Learning for Augmented Reality in Mobile Edge Networks. *IEEE Internet of Things Journal* 8, 13 (Jul 2021), 10843–10856. doi:10.1109/JIOT.2021.3050804

[8] Sony Corporation. 2023. *PlayStation 5*. https://www.playstation.com/en-us/ps5/

[9] Data Centre Dynamics Ltd (DCD). 2024. *Crown Castle scales back small cell build out, outlines plans to connect metro data center markets*. Retrieved Dec 1, 2024 from https://www.datacenterdynamics.com/en/news/crown-castle-scales-back-small-cell-build-out-outlines-plans-to-connect-metro-data-center-markets/

[10] Electronic360 News Desk. 2021. *Teardown: Google Nest Hub (2nd gen)*. https://electronics360.globalspec.com/article/17053/teardown-google-nest-hub-2nd-gen

[11] Peiran Dong, Jingyi Ge, Xiaojie Wang, and Song Guo. 2021. Collaborative edge computing for social Internet of Things: Applications, solutions, and challenges. *IEEE Transactions on Computational Social Systems* 9, 1 (2021), 291–301.

[12] EdgeIR.com. 2022. *EdgeMicro*. Retrieved Dec 1, 2024 from https://www.edgeir.com/companies/edgemicro

[13] Jeremy P Erickson and James H Anderson. 2022. Soft real-time scheduling. In *Handbook of Real-Time Computing*. Springer, 233–267.

[14] Geekbench. 2023. *Geekbench 6 - Cross-Platform Benchmark*. Retrieved March 16, 2023 from https://www.st.com/en/evaluation-tools/nucleo-f446re.html

[15] Mateusz Guzek, Dzmitry Kliazovich, and Pascal Bouvry. 2015. HEROS: Energy-efficient load balancing for heterogeneous data centers. In *2015 IEEE 8th International Conference on Cloud Computing*. IEEE, 742–749.

[16] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. 2014. Towards wearable cognitive assistance. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*. 68–81.

[17] Forrest N Iandola. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).

[18] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. 2019. Edge computing: A survey. *Future Generation Computer Systems* 97 (2019), 219–235. doi:10.1016/j.future.2019.02.050

[19] Youngjin Kim, Chiwon Song, Hyuck Han, Hyungsoo Jung, and Sooyong Kang. 2020. Collaborative task scheduling for IoT-assisted edge computing. *IEEE Access* 8 (2020), 216593–216606.

[20] LightReading. 2021. *Mapping out edge computing: How dense is it?* Retrieved Dec 1, 2024 from https://www.lightreading.com/the-edge-network/mapping-out-edge-computing-how-dense-is-it-

[21] Lightreading. 2022. *Demand for edge computing is taking longer than expected to develop*. Retrieved Dec 1, 2024 from https://www.lightreading.com/the-edge-network/demand-for-edge-computing-is-taking-longer-than-expected-to-develop

[22] LightReading. 2023. *Verizon admits to miscalculations on 5G, edge computing and private networks*. Retrieved Dec 1, 2024 from https://www.lightreading.com/the-edge-network/verizon-admits-to-miscalculations-on-5g-edge-computing-and-private-networks

[23] Haifeng Lu, Chunhua Gu, Fei Luo, Weichao Ding, and Xinping Liu. 2020. Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning. *Future Generation Computer Systems* 102 (Jan 2020), 847–861. doi:10.1016/j.future.2019.07.019

[24] Quyuan Luo, Shihong Hu, Changle Li, Guanghui Li, and Weisong Shi. 2021. Resource scheduling in edge computing: A survey. *IEEE Communications Surveys & Tutorials* 23, 4 (2021), 2131–2165.

[25] Md Abu Sayeed Mondol and John A Stankovic. 2015. Harmony: A hand wash monitoring and reminder system using smart watches. In *proceedings of the 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services on 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. 11–20.

[26] Arm Limited (or its affiliates). 2023. *White Paper: The economics of a trillion connected devices*. https://community.arm.com/arm-community-blogs/b/internet-of-things-blog/posts/white-paper-the-route-to-a-trillion-devices

[27] J Redmon. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

[28] Giampaolo Rodola. 2023. *psutil 5.9.4*. Retrieved Jan 20, 2023 from https://pypi.org/project/psutil/

[29] Mahadev Satyanarayanan, Paramvir Bahl, Ramon Caceres, and Nigel Davies. 2009. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing* 8, 4 (Oct. 2009), 14–23. doi:10.1109/MPRV.2009.82 Conference Name: IEEE Pervasive Computing.

[30] Ming Tang and Vincent W.S. Wong. 2022. Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing Systems. *IEEE Transactions on Mobile Computing* 21, 6 (Jun 2022), 1985–1997. doi:10.1109/TMC.2020.3036871

[31] Telecom.TV. 2020. *Ericsson's Edge Gravity drops out of favour*. Retrieved Dec 1, 2024 from https://www.telecomtv.com/content/edge/ericssons-edge-gravity-drops-out-of-favour-38931/

[32] Xiaofei Wang, Yiwen Han, Victor CM Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. 2020. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys & Tutorials* 22, 2 (2020), 869–904.

[33] Dale Willis, Arkodeb Dasgupta, and Suman Banerjee. 2014. ParaDrop: a multi-tenant platform to dynamically install third party services on wireless gateways. In *Proceedings of the 9th ACM Workshop on Mobility in the Evolving Internet Architecture* (Maui, Hawaii, USA) *(MobiArch '14)*. Association for Computing Machinery, New York, NY, USA, 43–48. doi:10.1145/2645892.2645901

[34] Mengwei Xu, Zhe Fu, Xiao Ma, Li Zhang, Yanan Li, Feng Qian, Shangguang Wang, Ke Li, Jingyu Yang, and Xuanzhe Liu. 2021. From cloud to edge: a first look at public edge platforms. In *Proceedings of the 21st ACM internet measurement conference*. 37–53.

[35] Wei Yu, Fan Liang, Xiaofei He, William Grant Hatcher, Chao Lu, Jie Lin, and Xinyu Yang. 2017. A survey on the edge computing for the Internet of Things. *IEEE access* 6 (2017), 6900–6919.

[36] Daniel Zhang, Yue Ma, Chao Zheng, Yang Zhang, X Sharon Hu, and Dong Wang. 2018. Cooperative-competitive task allocation in edge

computing for delay-sensitive social sensing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 243–259.

[37] Daniel (Yue) Zhang, Tahmid Rashid, Xukun Li, Nathan Vance, and Dong Wang. 2019. HeteroEdge: Taming the Heterogeneity of Edge Computing System in Social Sensing *(IoTDI '19)*. ACM, New York, NY, USA. http://doi.acm.org/10.1145/3302505.3310067