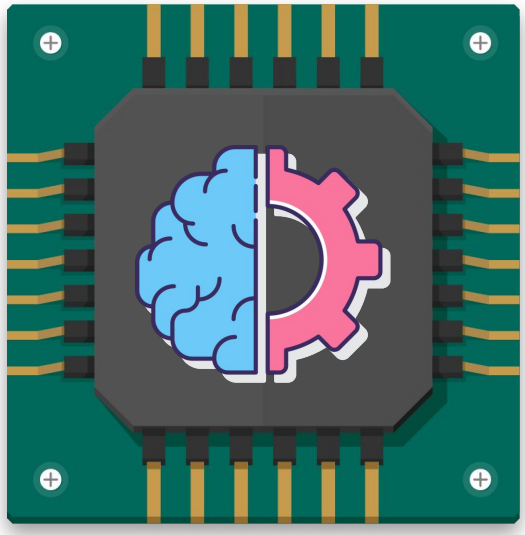
Deep Reinforcement Learning for Real-time Scheduling

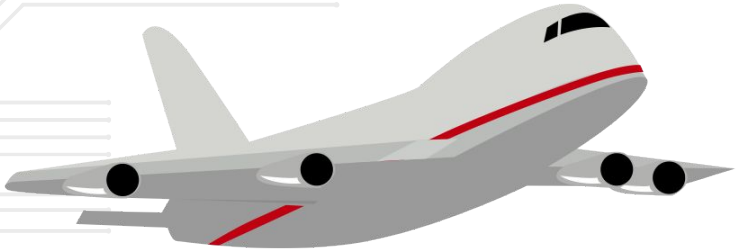


Dissertation

Bruno Daniel Durães Pereira Mendes

Master in Informatics and Computing Engineering
Faculty of Engineering of the University of Porto

Table of contents



01 Context and Motivation

Real-time systems are everywhere in society

- In this **IoT era**, embedded systems gather and transmit data in industrial and healthcare settings, improving processes
- Some embedded systems have **real-time requirements**, as timeliness is crucial for the correct operating of the system and the safety of humans
- Modern systems leverage **mixed-criticality** jobs (e.g. aircrafts' traffics control systems)

Avionics system from GE Aerospace [1]. Visualising the topology of the ground is less critical compared to sensing nearby objects with a flight radar.

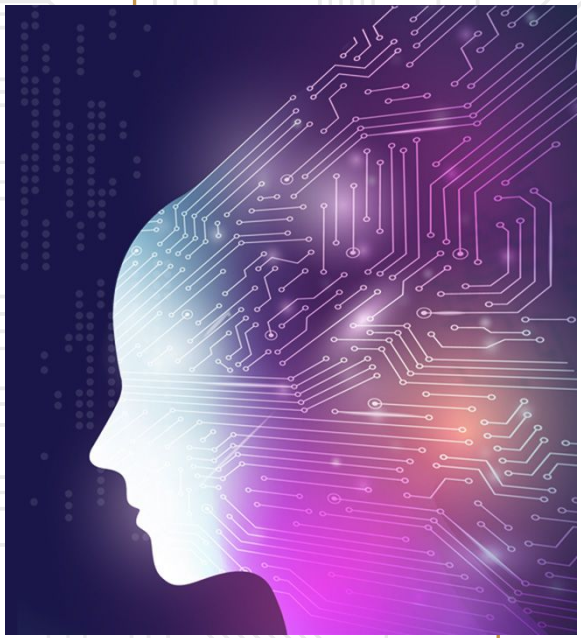
Could we use mixed-criticality as a testbed for AI-based scheduling optimisation?

Subject to design philosophy

- Contrary to traditional real-time systems, **not all tasks need** to finish execution before their deadline with **100% certainty**
- **Being optimistic** when estimating non-critical tasks' worst-case execution times may actually lead to **better resource utilisation**

Derive from observed behaviour

- Non-critical tasks WCET **do not need** need to be obtained with the **same level of assurance** and thus are usually derived from **experimentation** plus subjective scaling factor
- In the **era of Deep (Reinforcement) Learning**, we can manage resources of mixed-criticality RT systems more efficiently

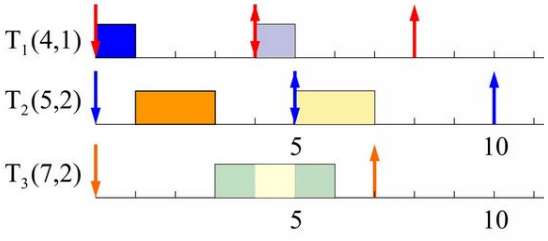


02

Background

Real-time systems

- **Correctness** depends not only on its logical correctness but also on **timeliness**
- Comprised of mostly **periodic tasks** with **deadlines** by which they must be invoked and executed
- Usually scheduled with **fixed-priority algorithms** such as RM or dynamic ones such as EDF
- Schedules may be guaranteed feasible using known **schedulability analysis** methods
- Nowadays **RTOS** such as FreeRTOS or ZephyrOS provide out-of-the-box real-time infrastructure for tiny devices



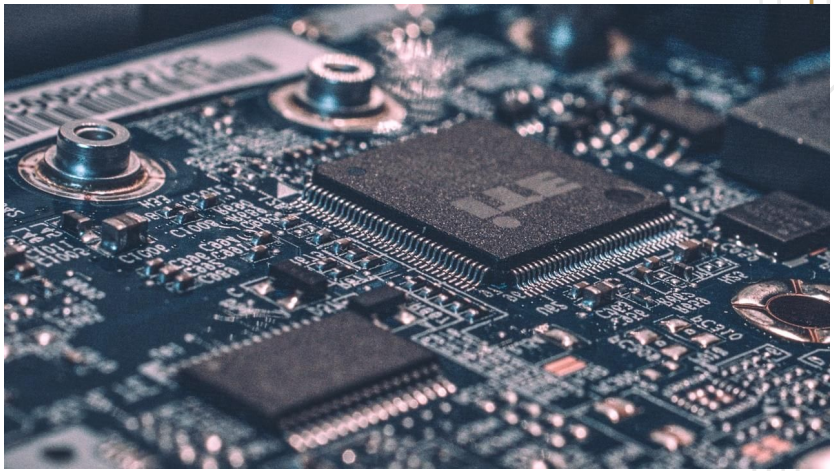
A RM schedule. [9]

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

Sufficient but not necessary schedulability test for task sets scheduled using fixed-priority. [10]

Mixed-critical systems

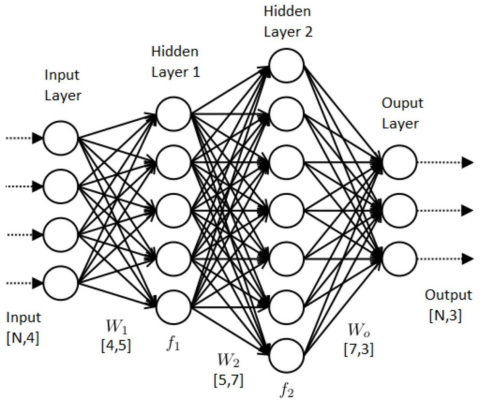
- Modelled with **Vestal's model** from 2007: each task provided with **WCET estimated** with an assurance level matching the respective criticality [4]
- Some research focuses on **deriving tasks' WCET based on architectural details** such as utilization of caches [2]
- Other approaches **use task servers** to achieve isolation and **change their time budget** upon a mode change according to **handcrafted heuristics** [3]



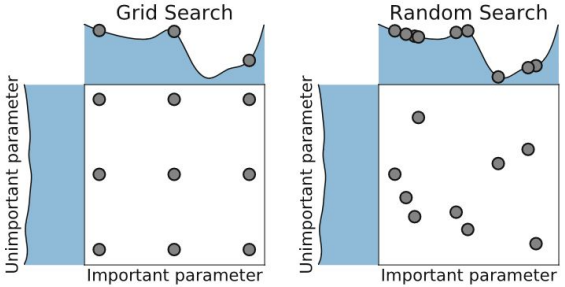
Even microcontrollers typically include caches. A cache hit significantly speeds up memory access. Locking σ pages in the cache for each task reduces its timing upper bound.

Machine Learning

- Concerned with detecting data patterns to predict future data or **decide under uncertainty**
- ML algorithms output a **model** that is **trained** by **minimising a loss function** e.g. via stochastic gradient descent
- **Hyperparameters** are intrinsic to the algorithms and must be provided *a priori*
- Modern ML relies heavily on architecturally-complex **NNs** to perform inference



A simple feedforward NN. [11]

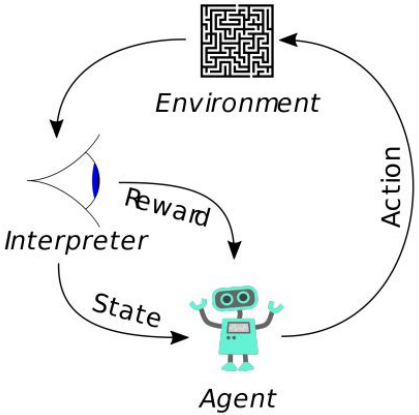


Ways of performing hyperparameter tuning. [12]

Reinforcement Learning

- Focuses on finding the **optimal behaviour** for an **agent** in an **environment**
- The classical RL algorithm is **Q-learning**, where a table maps state-action pairs to a **perceived value**
- The Q-values are updated as the agent receives **rewards** for its actions and **computes losses**
- The best parameterizable policy is the one that **maximizes the expected return over many trajectories**, considering their probability: [6]

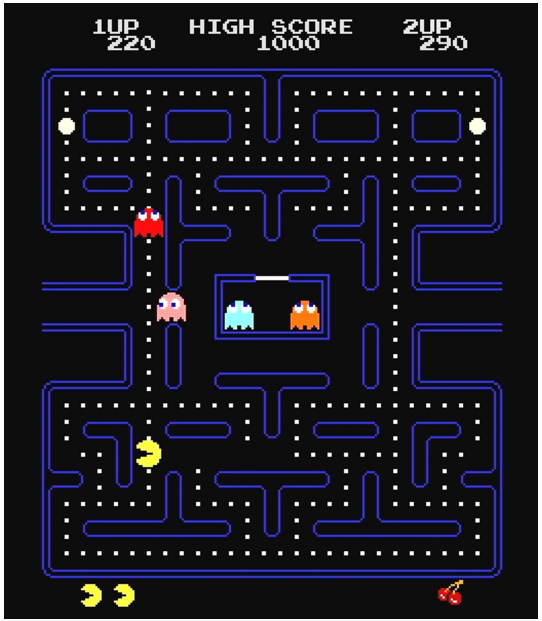
$$\theta = \underset{\theta}{\operatorname{argmax}}[\mathbb{E}_{\tau}[r[\tau]]] = \underset{\theta}{\operatorname{argmax}}\left[\int Pr(\tau|\theta)\cdot r(\tau)\,d\tau\right]$$



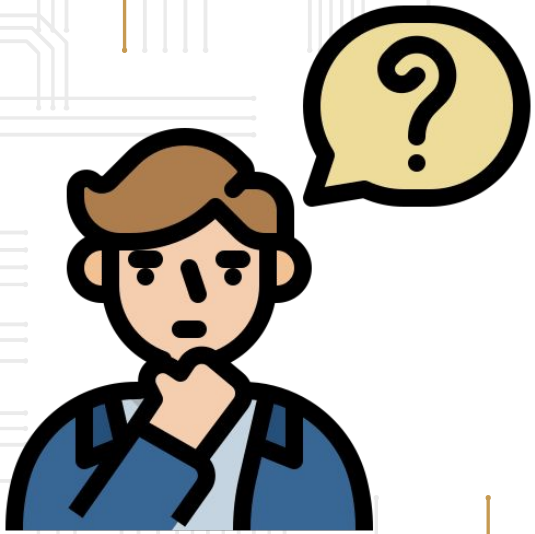
Agent-environment interaction loop. [14]

Deep Reinforcement Learning

- **DQN** from 2013 improved on traditional Q-learning by feeding **large state spaces** to a deep neural network [5]
- Modern **actor-critic** algorithms such as **DDPG** [7] or **PPO** [8] compute a policy via gradient methods, leading to possibly more stable learning and enabling continuous action spaces



Pacman, one of the Atari games used as a proof of concept in the DQN paper. Only the pixels are fed to the network as the state representation.



03

Methodology

Problem Statement

Platform

- Single-core platform
- CPU interrupted by HW timer to run a scheduler

Task Model

- Vestal's model with two criticality modes
- Switches to H-mode when a H-task overruns their budget in L-mode
- Switching from L-mode to H-mode kills all L-tasks
- H-mode dispenses all L-tasks
- Slow L-tasks are cancelled

Problem

It is not trivial to assign time budgets to tasks in a mixed-critical, real-time system due to the trade-off between perceived safety and potential resource utilisation.

Hypothesis

Using Deep Reinforcement Learning in a mixed-critical real-time system to orches-trate the scheduling process can improve the overall schedulability of the system.

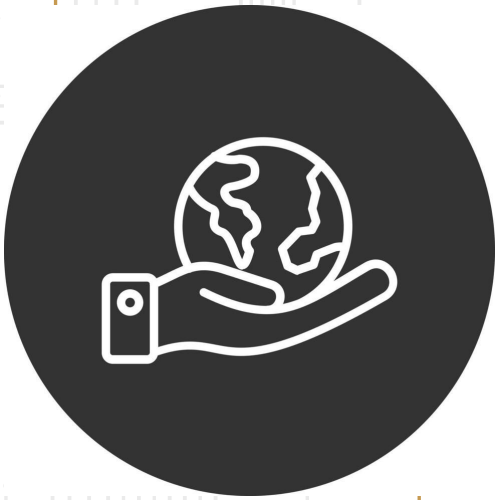
Methodological Approach

Environment Setup

- Simulated, **proof-of-concept real-time environment** using a mixed-critical scheduling schema - the *simulator*
- Realistic **task sets** generated
- **DRL agent** able to **mutate L-mode time budgets** of tasks in the system at runtime
- Understand how to model the system as a **MDP**

Evaluation Approach

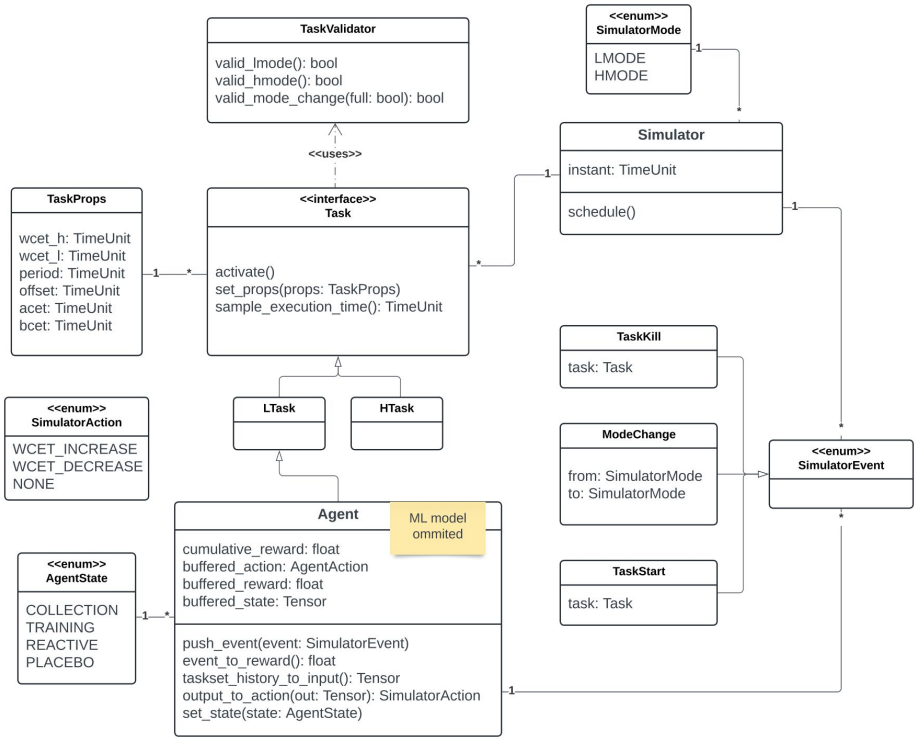
- Perform simulations with **different hyperparameters** and select the best ones for each task set
- Test whether there is a **relevant difference** in the number of mode changes and task cancellations, with and without the agent
- Evaluate the agent overhead, in terms of **runtime** and **memory usage**



04

Environment and Constraints

System Architecture



The Simulator

Scheduling

- Generates the **arrival of jobs of tasks** with a set of parameters according to a **fixed-priority scheduling scheme** during a desired time interval
- Uses mostly **incremental time progression**
- Yields control to the agent only when the **system is idle**: changing tasks' L-mode time budgets while they are running invalidates the analysis

Event Recording

- **Records relevant events** during simulation, such as task starts and mode changes
- **Signals these events** to the agent as they happen

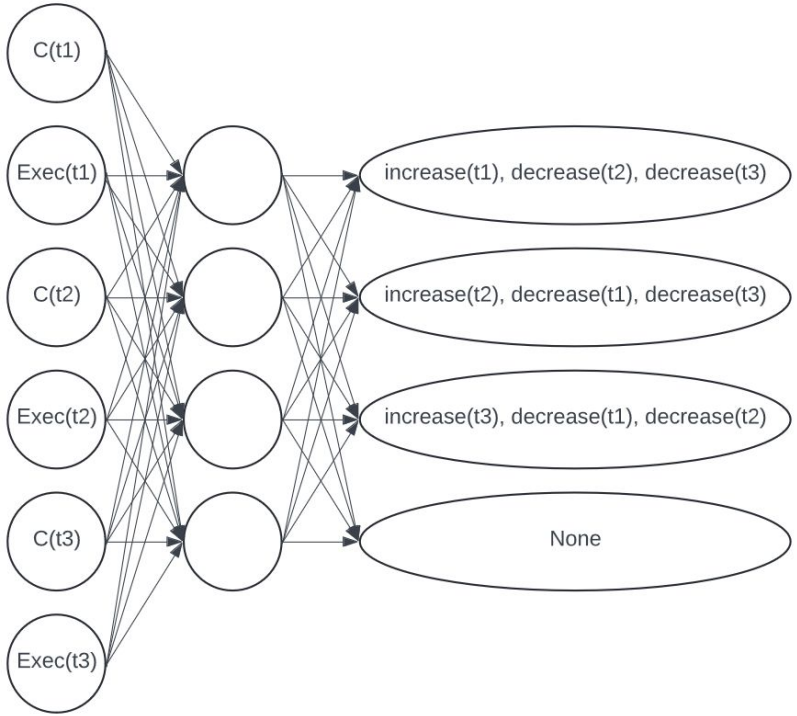
The Agent

Environment Representation

- The agent **pushes environment events to a queue** as it is notified and **consumes them when it is *activated***
- Events are **converted to an environment representation** that is fed to a NN

Action Space

- Each activation may trigger the **increase in 1 budget and decrease in 2 others**, or nothing
- The action space is **discrete**: there are $n \cdot C((n-1),2)+1$ actions available



The policy network of an agent in a system with tasks t1, t2 and t3.

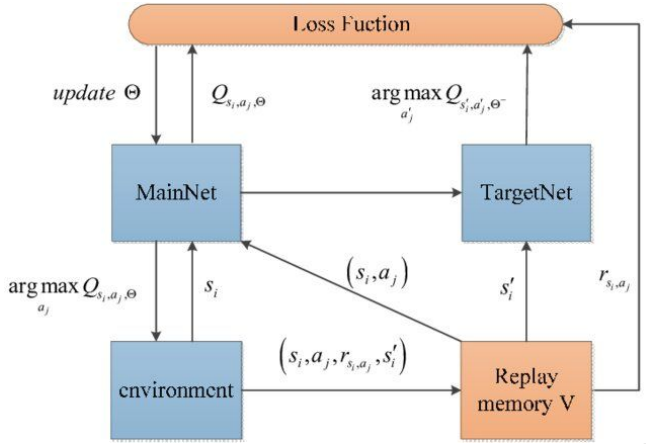
The Agent

DQN as the engine

- DQN is chosen as the DRL algorithm due to **ease of implementation** with ***libtorch*** as a tool

Rewards and Transition Assembly

- RL requires the generation of rewards, while training DQN requires storing transitions in the replay memory
- Transitions are built in the **activation *after* actions** are taken, since we **need time to pass to build new state**
- **Rewards are intuitive** considering our goals



The principle of the DQN algorithm. [13]

$$\text{event\_to\_reward(event)} = \begin{cases} 0.1 & \text{if event} = \textit{Start} \\ -1.0 & \text{if event} = \textit{TaskKill} \\ -2.0 & \text{if event} = \textit{ModeChange(LMode to HMode)} \\ 0.0 & \text{otherwise} \end{cases}$$

The Agent

Action Restrictions

- In traditional RL, the environment may not be deterministic, in that an **action may not always lead to the expected outcome**
- We **discard the agent's actions if AMC-rtb is violated** as a consequence
- To speed up the AMC-rtb computation time during an agent activation, we **replace the recurrence parcel** of the equation with task deadlines

$$C_i^H + \sum_{\tau_j \in hp(i) \cap \mathrm{LO}} \left\lceil \frac{R_i^{LO}}{T_j} \right\rceil C_j^L + \sum_{\tau_j \in hp(i) \cap \mathrm{HI}} \left\lceil \frac{R_i^*}{T_j} \right\rceil C_j^H \leq D_i$$

$$C_i^H + \sum_{\tau_j \in hp(i) \cap \mathrm{LO}} \left\lceil \frac{R_i^{LO}}{T_j} \right\rceil C_j^L + \sum_{\tau_j \in hp(i) \cap \mathrm{HI}} \left\lceil \frac{D_i}{T_j} \right\rceil C_j^H \leq D_i$$

05

Experimental Evaluation

Tasks Dataset Generation

Automotive Taskset Features

- An article from Robert Bosch GmbH unveils:
  - **realistic ACETs** for runnables in a car
  - **factors to calculate BCETs and WCETs from the ACET**
  - **period shares**

- A task is a **composition of runnables** with the same period

Period	Best		Worst	
	f_{min}	f_{max}	f_{min}	f_{max}
1 ms	0.19	0.92	1.30	29.11
2 ms	0.12	0.89	1.54	19.04
5 ms	0.17	0.94	1.13	18.44
10 ms	0.05	0.99	1.06	30.03
20 ms	0.11	0.98	1.06	15.61
50 ms	0.32	0.95	1.13	7.76
100 ms	0.09	0.99	1.02	8.88
200 ms	0.45	0.98	1.03	4.90
1000 ms	0.68	0.80	1.84	4.75

Number of Runnables	Share
2	30 %
3	40 %
4	20 %
5	10 %

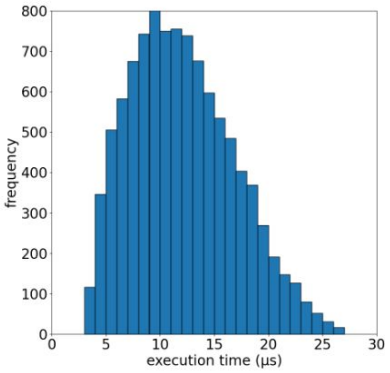
Period	Average Execution Times (μs)		
	Minimum	Average	Maximum
1 ms	0.34	5.00	30.11
2 ms	0.32	4.20	40.69
5 ms	0.36	11.04	83.38
10 ms	0.21	10.09	309.87
20 ms	0.25	8.74	291.42
50 ms	0.29	17.56	92.98
100 ms	0.21	10.53	420.43
200 ms	0.22	2.56	21.95
1000 ms	0.37	0.43	0.46

Period	Share
1 ms	3 %
2 ms	2 %
5 ms	2 %
10 ms	25 %
20 ms	25 %
50 ms	3 %
100 ms	20 %
200 ms	1 %
1000 ms	4 %

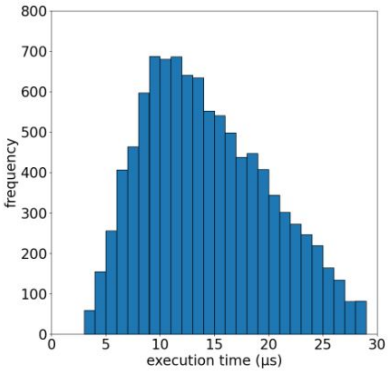
Tasks Dataset Generation

Generating Execution Times of Simulated Tasks

- We have **no research insight** on how to simulate architectural-related patterns in computing systems
- We resort to a **PERT distribution** for sampling execution times during simulation, due to a strong analogy with the timing uncertainty in project management



(a) PERT Distribution

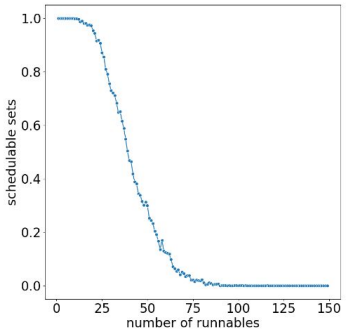


(b) Triangular Distribution

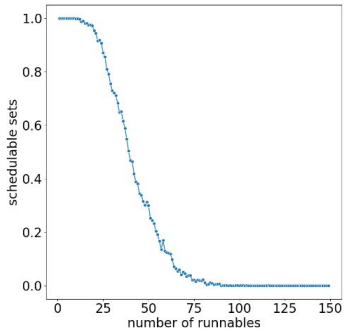
Tasks Dataset Generation

Generating Task Set Parameters

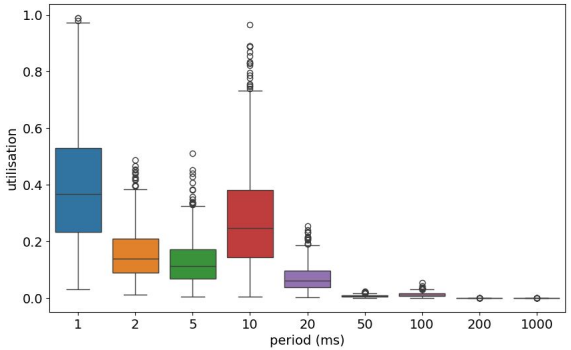
- We use **PERT** for generating ACETs for runnables, with the average ACET as mode
- We multiply these WCETs by a random value between *fmax* and *fmin* to get the WCET_Hs and BCETs
- The WCET_Ls are a random value between the WCET_H and the ACET
- We **can't match the number of runnables** in the system stated in the article due to our single-core, mixed-critical platform



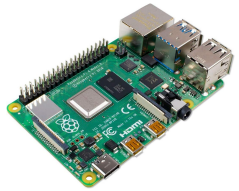
(a) 30% L-tasks



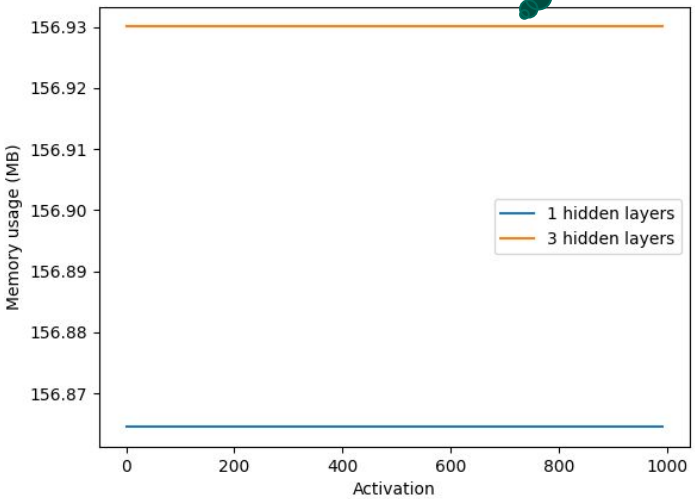
(b) 60% L-tasks



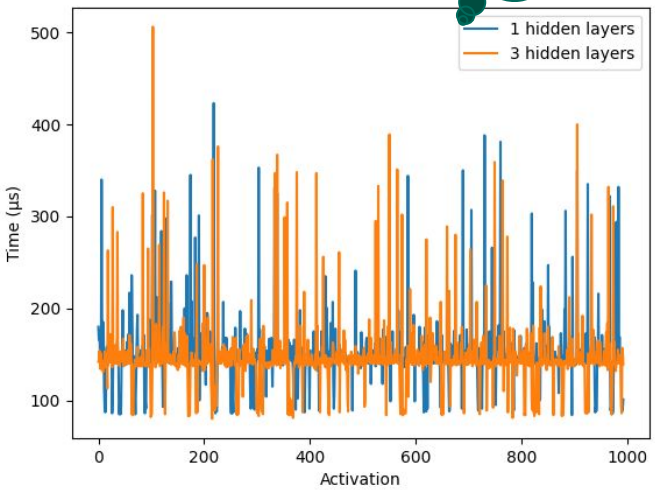
Computational Requirements of Different Models



Resident Set Size (includes shared libraries)



Time per activation



Impact on the Quality of Service

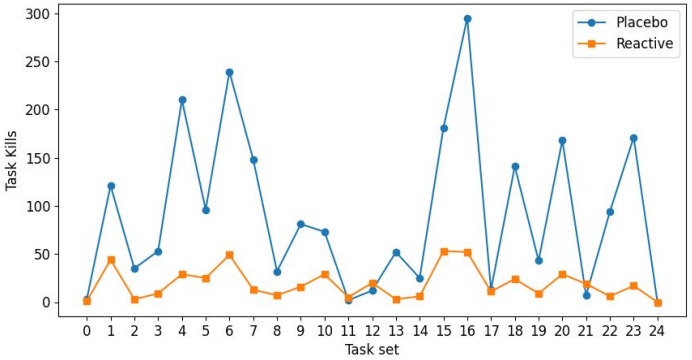
Test pipeline

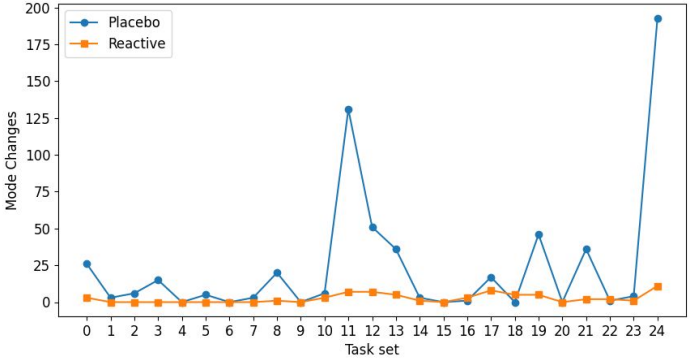
- Generate **feasible task set**
- Choose a set of hyperparameters and train the agent in a 10-second simulation
- Run the trained agent in a new 2-second simulation
- Select the set of hyperparameters that resulted in a better agent, i.e. the one that received the **most reward**

Hyperparameter	**Values Space**
maximum memory size	200
minimum memory size	20
gamma	0.99
network update frequency	5, 10, 20
learning rate	5×10^{-5}
hidden layers sizes	$[n/2], [n, n/2], [n, n/2, n/4]$
sample batch size	3, 6, 12
network activation function	sigmoid, relu, tanh

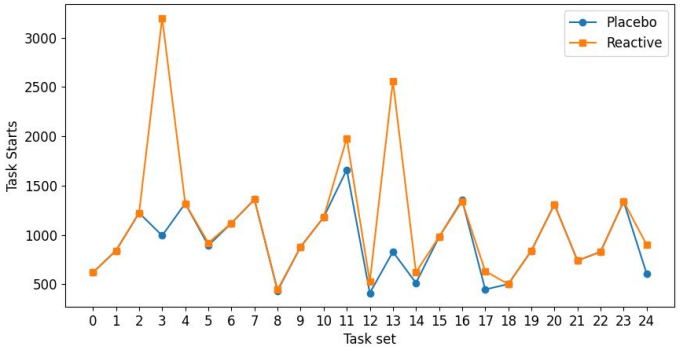
The chosen set of hyperparameters.

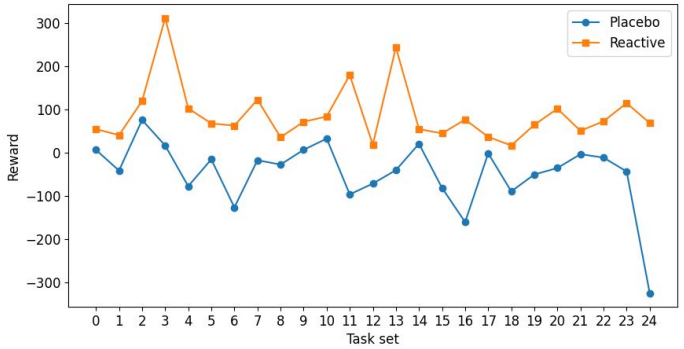
Impact on the Quality of Service





Impact on the Quality of Service





06

Conclusions

Conclusions

Main Contributions

- Review of **state-of-the-art RT scheduling approaches** and **DRL algorithms**
- Study on the **constraints required for applying RL in an AMC mixed-critical system** to allow for certification
- Novel approach to **convert asynchronous environment signals to state** in the real-time scheduling domain
- Development of a **realistic task set generator**, based on industry insights
- **Evaluation of the agent's impact** on the schedulability of the real-time system

Future Work

- More through evaluation with **multiple simulations and longer simulation times**
- Review scheduling analysis to find whether we can **run the agent periodically**
- Study **other DRL algorithms** and compare the performance
- Apply DRL optimisation ideas to **other RT scheduling schemas** than AMC

Hypothesis

Using Deep Reinforcement Learning in a mixed-critical real-time system to orchestrate the scheduling process can improve the overall schedulability of the system.

Questions?

Bibliography

1. Avionics Systems | GE Aerospace. Accessed: Jan. 29, 2024. [Online]. Available: https://www.geaerospace.com/systems/avionics

2. Muhammad Ali Awan et al. Mixed-criticality Scheduling with Dynamic Redistribution of Shared Cache. Apr. 28, 2017. arXiv: 1704.08876[cs]. URL: http://arxiv.org/abs/1704.08876 (visited on 01/22/2024).

3. Muhammad Ali Awan et al. “Techniques and Analysis for Mixed-criticality Scheduling with Mode-dependent Server Execution Budgets”. In: ACM Transactions on Embedded Computing Systems 18.5 (Oct. 8, 2019), 109:1–109:23. ISSN: 1539-9087. DOI: 10.1145/3358234. URL: https://dl.acm.org/doi/10.1145/3358234 (visited on 01/23/2024).

4. Steve Vestal. “Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance”. In: 28th IEEE International Real-Time Systems Symposium (RTSS 2007). 28th IEEE International Real-Time Systems Symposium (RTSS 2007). ISSN: 1052-8725. Dec. 2007, pp. 239–243. DOI: 10.1109/RTSS.2007.47. URL: https://ieeexplore.ieee.org/document/4408308 (visited on 01/18/2024).

5. Volodymyr Mnih et al. Playing Atari with Deep Reinforcement Learning. Dec. 19, 2013. arXiv: 1312.5602[cs]. URL: http://arxiv.org/abs/1312.5602 (visited on 01/06/2024).

6. Simon J.D. Prince. Understanding Deep Learning. MIT Press, 2023. URL: http://udlbook.com.

7. Timothy P. Lillicrap et al. Continuous control with deep reinforcement learning. 2016. arXiv: 1509.02971[cs, stat]. URL: http://arxiv.org/abs/1509.02971 (visited on 01/18/2024).

8. John Schulman et al. Proximal Policy Optimization Algorithms. Aug. 28, 2017. arXiv: 1707.06347[cs]. URL: http://arxiv.org/abs/1707.06347 (visited on 01/07/2024).

Bibliography

9. Real-time Scheduling | CIS 700. Accessed: Jul. 10, 2024. [Online]. Available: https://www.slideshare.net/slideshow/realtime-scheduling/102581763

10. C. Liu and James Layland. “Scheduling Algorithms for Multiprogramming in a Hard- Real-Time Environment”. In: Journal of the Association for Computing Machinery 20.1 (1973)

11. Extreme Learning Machines I. Accessed: Jul. 12, 2024. [Online]. Available: https://medium.datadriveninvestor.com/extreme-learning-machines-82095ee198ce

12. Matthias Feurer and Frank Hutter. “Hyperparameter Optimization”. In: Automated Machine Learning. Ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Series Title: The Springer Series on Challenges in Machine Learning. Cham: Springer International Publishing, 2019, pp. 3–33. DOI: 10.1007/978-3-030-05318-5_1. URL: http://link.springer.com/10.1007/978-3-030-05318-5_1 (visited on 05/29/2024).

13. Deep Q-Learning-Based Content Caching With Update Strategy for Fog Radio Access Networks. Accessed: Jul. 12, 2024. [Online]. Available: https://www.researchgate.net/publication/334387468_Deep_Q-Learning-Based_Content_Caching_With_Update_Strategy_for_Fog_Radio_Access_Networks/figures?lo=1

14. Megajuice. English: Diagram showing the components in a typical Reinforcement Learning (RL) system. An agent takes actions in an environment which is interpreted into a reward and a representation of the state which is fed back into the agent. Apr. 2017. URL: https://commons.wikimedia.org/wiki/File:Reinforcement_learning_diagram.svg (visited on 12/24/2023).

The Simulator

Algorithm 1: Simulator

```
1  current_mode ← LMode;
2  current_running_jobs ← [];
3  task_start_events ← collect_start_events(self.tasks);
4  time ← 0;
5  while time < duration do
6      new_jobs ← determine_new_jobs(task_start_events, current_mode, time);
7      current_running_jobs.append(new_jobs);
8      current_running_jobs.sort_by_priority();
9      if current_running_jobs.is_not_empty() then
10         job ← current_running_jobs.first();
11         if job.is_starting() then
12             next_start_event ← find_start_event(task_start_events, job);
13             update_start_event(next_start_event, job);
14             push_event(agent, job, time, START, job.id);
15         if job.has_finished() then
16             remove_job(current_running_jobs, job);
17             push_event(self.agent, job, time, END, job.id);
18         else if current_mode == LMode ∧ job.running_for ≥ job.task.wcet_l then
19             if job.task == HTask then
20                 current_mode ← HMode;
21                 retain_htasks(current_running_jobs);
22                 push_event(self.agent, job, time, MODE_CHANGE, HMODE);
23             else
24                 kill_task(current_running_jobs, job, time);
25                 push_event(agent, job, time, TASK_KILL, job.id);
26     else
27         if current_mode ≠ LMode then
28             current_mode ← LMode;
29             push_event(self.agent, job, time, MODE_CHANGE, LMODE);
30         run_agent(agent, time);
31     time ← next_start_event(task_start_events, time);
```

The Agent

Algorithm 2: Agent activation

```
1  state ← history_to_input(events_history, simulator);
2  action ← None;
3  if self.stage ≠ Placebo then
4      action ← epsilon_greedy(memory_policy, policy_network, epsilon, state, simulator);
5  action.apply(tasks);
6  if action ≠ None then
7      if ¬ feasible_schedule_online(tasks) then
8          action.reverse().apply(tasks);
9  if self.buffered_action ≠ None then
10     reward ← calculate_reward(events_history, simulator);
11     push_transition_replay_mem(buffered_state, buffered_action, reward, state);
12     if replay_memory.is_filled() then
13         stage ← Training;
14 buffered_action ← action;
15 buffered_state ← state;
16 if self.stage == Training then
17     sample_batch ← replay_memory.sample_batch(sample_batch_size);
18     qvalues ← policy_network.forward(memory_policy, sample_batch.state).gather();
19     target_values ← target_network.forward(memory_target, sample_batch.next_state);
20     expected_values ← sample_batch.reward + gamma × max_target_values;
21     loss ← mean_squared_error(qvalues, expected_values);
22     backward(loss);
23     memory_policy.apply_grads_adam(learning_rate);
24     if reward_history_len % update_freq == 0 then
25         target_network.update(memory_policy);
26         update_epsilon();
```