Energy Efficiency Metrics for Reinforcement Learning on the Travelling Salesman Problem: An Experimental Approach in Green Computing

Celestino Simon
Dept. Electronic Engineering
Maynooth University
Maynooth, Ireland
celestino.simon.2024@mumail.ie

Samara Santos
Hamilton Institute
Maynooth University
Maynooth, Ireland
samara.santos.2024@mumail.ie

Thalita Nazaré
Centre for Ocean Energy Research
Maynooth University
Maynooth, Ireland
thalita.nazare.2023@mumail.ie

André L. C. Ottoni
Dept of Computer Science
Federal University of Ouro Preto
Ouro Preto, Brazil
andre.ottoni@ufop.edu.br

Danton D. Ferreira
Dept of Automatic
Federal University of Lavras
Lavras, Brazil
danton@ufla.br

Erivelton Nepomuceno
Hamilton Institute
Centre for Ocean Energy Research
Maynooth University
Maynooth, Ireland
erivelton.nepomuceno@mu.ie

***Abstract*—Artificial intelligence models continue to grow in complexity, but so does their energy demand—an issue gaining attention as these models expand into energy-sensitive domains such as Reinforcement Learning. This paper examines the energy usage of Q-learning in solving the Travelling Salesman Problem. To carry out this analysis, both hardware and software tools were used to measure energy: a PMD-USB power meter for direct readings and CodeCarbon for software-based estimation. By comparing a standard version of Q-learning with one that includes ϵ-decay, the results showed clear improvements. The optimised version completed tasks faster and used noticeably less energy in some cases, up to 68% less. These results suggest that with minor adjustments, it is possible to make Machine Learning models more energy-efficient. The approach used here could be beneficial in other areas that rely on AI but require energy efficiency, such as logistics or embedded systems. Beyond its core findings, the study also underscores the importance of validating software-based estimates with real-time hardware monitoring. The dual platform comparison provides a methodological basis for benchmarking future RL models under energy constraints. This contribution supports a growing interest in evaluating Machine Learning not only by performance, but also by sustainability.**

***Index Terms*—Artificial Intelligence (AI), Green Machine Learning, Energy Efficiency, Green Computing, Sustainable AI, Travelling Salesman Problem(TSP), Reinforcement Learning (RL), Power Measurement**

I. Introduction

In recent years, AI has driven remarkable advances across various fields, including logistics and climate modeling [1]. However, this progress has come with increasing environmental costs. Forecasts indicate that the Information and Communication Technology (ICT) sector may contribute between 3% and 5% of global carbon emissions by 2024, with data centres alone expected to consume nearly 3,000 TWh by 2030 [2], [3]. This upward trend has been acknowledged by the European Commission [4].

Despite being long-standing, concerns about ICT energy use are becoming more pressing. Past analyses warned that efficiency gains are often outpaced by the growing scale of computation [5]. These warnings now carry more weight, given the expanding footprint of AI across sectors.

In parallel, major tech companies such as Google and Microsoft[1] have faced scrutiny over the emissions generated by their AI infrastructure. Consequently, the pressure to adopt sustainable practices is intensifying [6], [7].

To address this, the field of green computing promotes energy-efficient methods throughout the technology lifecycle [1], [8]. Encouragingly, advances such as optimised attention mechanisms in transformers suggest that AI sustainability remains achievable [9]. One promising approach within this space is to improve the energy efficiency of learning algorithms themselves.

Among these algorithms, Q-learning has emerged as a robust technique for solving optimisation problems [10]. Notably, it has been applied effectively to TSP [11], where the objective is to identify the shortest route visiting a set of cities exactly once. As interest grows in reducing the environmental cost of such methods, energy-aware algorithm design is gaining attention[2] [3].

Among many optimisation challenges, TSP remains one of the most extensively studied NP-hard problems, with practical applications in logistics, robotics, and circuit design [11]. While methods like branch-and-bound, genetic algorithms, and

[1]NPR, AI Brings Soaring Emissions for Google and Microsoft, 12 July 2024

[2]International Energy Agency (IEA), Data Centres and Data Transmission Networks, 2024

ant colony optimisation remain effective, they often demand extensive computational effort or tailored heuristics. Reinforcement learning, by contrast, offers online adaptability and has gained traction in dynamic settings [12].

Building on this, recent studies show that Q-learning can approximate optimal routes in TSP without requiring labelled data [13], [14]. This positions it as a competitive option compared to classical metaheuristics. Several works have also explored the application of Reinforcement Learning to dynamic or stochastic variants of TSP, confirming its potential beyond deterministic instances [15], [16]. However, despite its growing use, there is limited understanding of its energy demands.

To mitigate this gap, researchers have examined strategies for speeding up Q-learning and lowering its computational footprint [10], [11]. Tools like CodeCarbon provide software-based energy estimates, while hardware meters such as the PMD-USB offer real-time measurements. Nevertheless, reliance on runtime or software tools can be misleading. Jay et al. [17] and Koomey et al. [18] report that such tools may overestimate usage by including irrelevant components. Even hardware-based tools face challenges with consistency [19].

These issues hinder the accurate quantification of Machine Learning energy use. Incomplete or biased reporting may misguide optimisation strategies. Most current evaluations stop at runtime or simulations, rarely validating claims through physical measurement. A more reliable framework is therefore needed—one that integrates multiple energy assessment modalities.

This study aims to fill that need by comparing both hardware and software tools for energy measurement. A dual-platform methodology is proposed to assess the energy profile of a Q-learning algorithm applied to TSP. One version follows a standard design, while the other incorporates an ϵ-decay strategy to speed up convergence and reduce energy usage.

While the focus of this work is on Reinforcement Learning, it is important to situate Q-learning within the broader landscape of TSP optimisation methods. Classic metaheuristics such as Ant Colony Optimisation (ACO) [20], Genetic Algorithms (GA) [21], and Simulated Annealing have long been used to solve TSP efficiently. These approaches are often effective for smaller problem instances and can achieve high-quality solutions. However, their scalability and energy efficiency vary significantly depending on implementation. In contrast, Reinforcement Learning techniques, when optimised, can adapt to dynamic environments and may offer better energy profiles under specific constraints. This study builds on that premise by evaluating the energy cost of Q-learning under different configurations.

To examine this potential, the study offers the following key contributions:

- A dual-platform method combining real-time hardware readings and software-based energy estimation.
- A comparison of baseline and optimised Q-learning implementations in terms of energy and performance.
- A discussion of CodeCarbon and PMD-USB trade-offs for energy-conscious AI research.

The rest of the paper is organised as follows. Section II covers background. Section III describes the methodology. Section IV presents results. Section V concludes the study.

II. Background

The concept of energy proportional computing remains highly relevant when evaluating algorithmic workloads [22]. To meaningfully improve energy efficiency in AI algorithms, it is first necessary to understand how power consumption is measured and how learning processes operate computationally. This section outlines the core concepts supporting the experiments in this work: CPU power monitoring principles, a brief overview of Q-learning, and a comparison of software and hardware-based energy tracking tools. These foundations set the stage for the methodology that follows.

A. CPU and Power Measurement

When running RL tasks, especially in optimisation problems, it's common for the CPU to handle a large computational load. Because of this, it becomes important to monitor how much power is being used. There are generally two ways to do this: using software-based estimations or hardware devices.

1) Software-Based Methods: Software tools estimate energy consumption based on system activity and sensor data. On Windows, for instance, CPU load is accessible via Task Manager [23]. A widely used library in Machine Learning is CodeCarbon, which estimates energy (E) as:

$$E = P \times T \times \frac{\text{C\_U}}{100}, \tag{1}$$

where P is power (kW), T is time (hours), and C_U is CPU usage (%).

Carbon emissions are then computed by:

$$\text{CO}_2\text{e} = E \times \text{C\_I}, \tag{2}$$

where C_I denotes carbon intensity in kg CO_2e per kWh.

Though convenient, software estimators are often imprecise [18]. CodeCarbon may capture energy from unrelated components, such as memory or fans, leading to 10–20% overestimations [17], [18]. Alternatives like Running Average Power Limit (RAPL) offer CPU-specific readings but may behave inconsistently under dynamic loads [24].

Despite such limitations, software tools remain widely adopted due to their ease of deployment and platform independence. However, their abstraction complicates the isolation of algorithm-specific energy consumption, posing challenges for optimisation-oriented studies.

2) Hardware-Based Methods: Hardware-based tools offer physical measurements by capturing voltage and current directly from the system, using:

$$P = V \times I, \tag{3}$$

where V is voltage and I is current.

This study employs the ElmorLabs PMD-USB, a real-time power monitor supporting up to 1680 watts via USB. By integrating this device during training runs, it becomes possible to capture actual CPU power usage throughout learning [25].

These tools offer greater accuracy than software counterparts but come with constraints. They require dedicated hardware, proper calibration, and may not align perfectly with system-level reports. Nonetheless, they serve as a critical baseline to validate software-based estimations. Hardware methods allow finer attribution of energy consumption to specific processes, enabling better benchmarking and validation of energy-aware modifications [19]. As AI workloads intensify, such precision becomes essential for advancing sustainable computing.

B. Reinforcement Learning Foundations

Reinforcement learning enables agents to learn optimal behaviours through trial and error interactions with an environment, modelled as a Markov Decision Process (MDP) [10]. In this setting, the agent seeks to maximise cumulative future rewards by selecting actions based on the current state.

Q-learning is a widely used off-policy RL algorithm suited for discrete action spaces [26]. It maintains a Q-table in which each entry $Q(s, a)$ estimates the expected return of taking action a in state s and following the optimal policy thereafter. The table is updated iteratively via the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right], \quad (4)$$

where α is the learning rate, γ the discount factor, r the immediate reward, and s' the next state.

Action selection typically follows an ϵ-greedy strategy, which balances exploitation of known rewards with exploration of alternative paths [27]. In the context of TSP, states correspond to cities, actions denote transitions to unvisited cities, and rewards are often defined as the negative edge distance. The Q-learning agent incrementally constructs a valid tour while optimising path efficiency, making it a competitive metaheuristic when enhanced with convergence and energy-saving mechanisms [14].

III. Methodology

This section describes the dual platform setup used to measure the energy consumption of Q-learning applied to TSP. Two complementary tools were employed: the PMD-USB for hardware-based measurements on Windows, and the CodeCarbon library for software-based estimations on Linux. This setup enabled simultaneous analysis of real-time hardware data and software-based energy estimates, allowing a more logical comparison of energy performance across environments.

To ensure consistency, both Q-learning variants, baseline and optimised, were implemented and executed independently on each platform using the same algorithmic structure. Power consumption and execution data were recorded for all cases. The experimental flow is illustrated in Fig. 1.

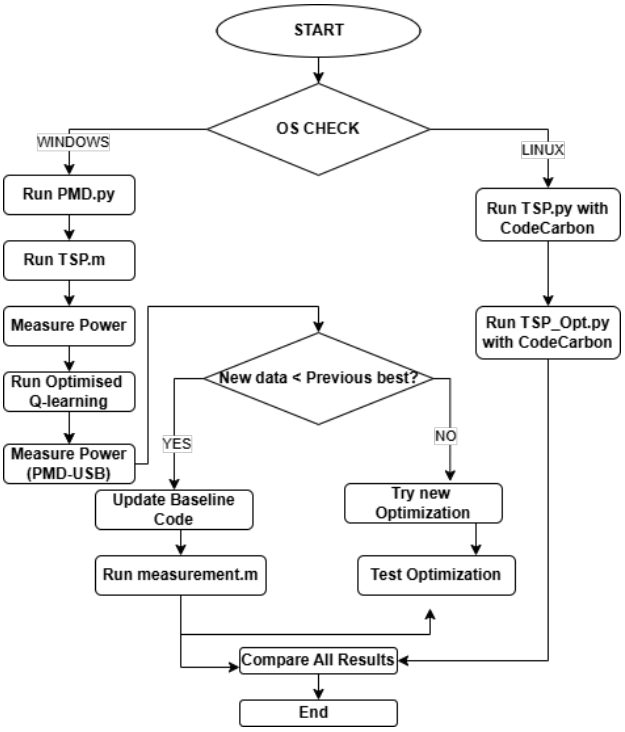

Fig. 1. Dual-platform workflow used to evaluate the energy efficiency of Q-learning applied to TSP. The left branch shows the Windows-based measurement process using the PMD-USB device and MATLAB implementation. Power is measured for both baseline and optimised variants. The right branch corresponds to the Linux-based setup where Python implementations are monitored via CodeCarbon. Both workflows generate comparable datasets for analysis.

A. Hardware Settings

To ensure fair comparison, experiments were conducted on systems with clearly defined hardware and software configurations. CodeCarbon executions ran on a Linux desktop with an Intel(R) Core(TM) i5-4590 CPU at 3.30 GHz, 16 GB RAM, and no dedicated GPU. The system used Python 3.10.12 and CodeCarbon 2.6.0. Background processes were minimised to prevent interference with energy tracking.

Figure 2 shows the system metadata automatically captured by CodeCarbon before initiating the measurement process.

B. Optimisation Techniques: Epsilon Decay and Early Stopping

To reduce the energy consumption and convergence time of the Q-learning algorithm, two optimisation techniques were applied: epsilon decay and early stopping. These modifications aimed to improve learning efficiency while preserving solution quality in the TSP scenario.

The epsilon parameter (ϵ) governs the trade-off between exploration and exploitation in RL. Instead of using a constant value throughout training, the optimised implementation applied exponential decay. Starting at $\epsilon = 0.9$, the exploration rate decreased gradually at each episode using the update rule:

$$\epsilon_{t+1} = \max(\epsilon_{\text{min}}, \epsilon_t \cdot \epsilon_{\text{decay}}), \quad (5)$$

```
CPU Model on constant consumption mode: Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz
>>> Tracker's metadata:
  Platform system: Linux-6.5.0-44-generic-x86_64-with-glibc2.35
  Python version: 3.10.12
  CodeCarbon version: 2.6.0
  Available RAM : 15.518 GB
  CPU count: 4
  CPU model: Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz
  GPU count: None
  GPU model: None
```

Fig. 2. System specifications captured by CodeCarbon on a 4-core Intel i5-4590 CPU with 15.5 GB RAM and no GPU.

where $\epsilon_{\text{decay}} = 0.9999$ and $\epsilon_{\text{min}} = 0.1$. This dynamic strategy allowed the agent to explore broadly in the early stages while focusing more on exploitation as training progressed.

Early stopping was introduced to eliminate redundant computation once a satisfactory solution was found. A 9,000 km threshold—based on the Berlin52 dataset—was used to terminate training as soon as a shorter tour was discovered, even if the 100,000 episode cap had not been reached. This reduced unnecessary iterations and lowered overall energy use.

These enhancements were exclusive to the footprint-aware variant. The baseline version, in contrast, executed the full number of episodes without optimisation, enabling a fair comparison of energy consumption and learning performance.

C. PMD-USB Measurement

Following the application of the optimisation strategies, this study employed the PMD-USB device to capture real-time power consumption. The device connects via USB to a Windows system and records instantaneous voltage and current from the power source, logging the data to a CSV file. This approach enables high-resolution sampling and facilitates the detection of energy fluctuations, especially during dynamic workloads.

The measurement process began with a Python script (*PMD.py*) that established the connection with the PMD-USB device. This script operated concurrently with the MATLAB-based Q-learning execution (*TSP.m* for the baseline and *TSP_Opt.m* for the optimised variant). During runtime, an auxiliary Python function monitored CPU usage at regular intervals using the `getProcessTimes()` method from the `psutil` library. The final power estimation incorporated this data using the following formula:

$$P = \frac{V \times I \times \text{C\_U}}{100}, \tag{6}$$

where V is the voltage measured by PMD-USB, I is the current also captured by the device, and C_U is the CPU usage expressed as a percentage. This integration of real sensor data and system metrics enabled the construction of a time-aligned power dataset stored in *TSP.csv* and *Optimised TSP.csv*.

Post-processing of the measurement logs was performed using the custom MATLAB script *measurement.m*. This tool filtered the dataset based on execution time windows and

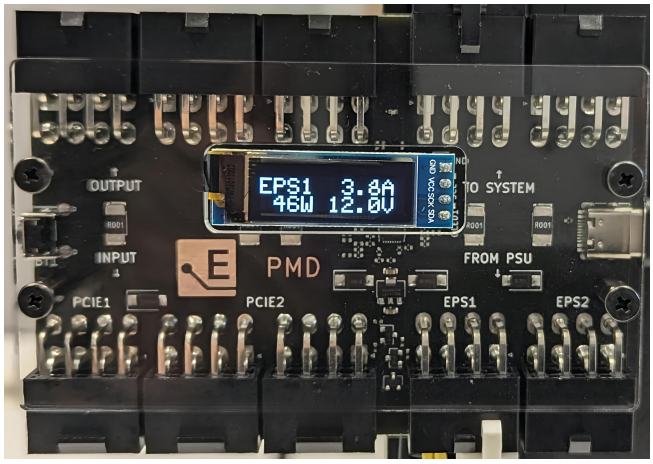

Fig. 3. Photograph of the PMD-USB device connected to the test platform during the measurement process.

applied trapezoidal numerical integration to estimate energy in Joules:

$$E = \int_{t_0}^{t_f} P(t)\, dt \approx \sum_{i=1}^{n-1} \frac{(P_i + P_{i+1})}{2} \cdot \Delta t, \tag{7}$$

where P_i denotes the power at timestamp i, and Δt is the interval between samples. Beyond total energy, the script also calculated statistical indicators such as peak power, average power, standard deviation, and cost estimation using a fixed electricity rate of $0.4327/kWh$, corresponding to the average residential tariff in Ireland during 2024. These metrics enabled a detailed analysis of the energy footprint for each Q-learning variant.

This measurement pipeline proved particularly effective for capturing short energy bursts triggered by algorithmic transitions, fluctuations often masked by software-only estimation tools. Moreover, incorporating real-time CPU usage via `psutil` ensured that recorded power reflected actual computational activity rather than passive electrical draw. All scripts used in this process are publicly available to support reproducibility and further research [28].

D. CodeCarbon Estimation Setup

Following the application of the optimisation techniques ϵ decay and early stopping, a complementary software-based

Algorithm 1 Energy Measurement Procedure Using CodeCarbon

1: **Initialise** CodeCarbon tracker with sampling rate
2: **Start** tracker
3: **Record** start time
4: **Execute** Q-learning training loop
5: **Record** end time and compute elapsed time
6: **Stop** tracker and retrieve emissions rate
7: **Convert** emissions to total energy in joules
8: **Print** results

estimation was performed using the CodeCarbon library. This tool is designed to track energy usage and estimate CO_2 emissions for Python workloads. Tests were run on a Linux machine using a Python implementation of Q-learning applied to the same TSP instance, mirroring the MATLAB-based setup to ensure comparability between measurement methods.

The scripts *TSP.py* and *TSP_Opt.py* were modified to initialise and terminate CodeCarbon tracking at execution boundaries. The emissions tracker was configured with a one-second sampling rate and used in both the baseline and footprint-aware variants. Algorithm 1 summarises the core process, while the full implementation is provided in Listing 1 for reproducibility.

```
from codecarbon import EmissionsTracker
import time

tracker = EmissionsTracker(measure_power_secs=1)
tracker.start()
start = time.time()

# --- Q-learning training loop happens here ---

elapsed = time.time() - start
emissions = tracker.stop()

energy_joules = (emissions / 1000) * 3.6e6 * elapsed

print(f'Emissions rate: {emissions:.6f} g.CO2eq/s')
print(f'Total energy consumed: {energy_joules:.2f} J
    ')
```

Listing 1. Compact CodeCarbon tracking implementation used in Q-learning.

After tracking, the tool outputs an emissions estimate in grams of CO_2 equivalent per second. Although CodeCarbon internally estimates energy based on CPU activity and runtime, as described earlier in Equation 1, in this study, the tool's reported carbon emissions were used to back-calculate energy consumption using the following conversion:

$$E = \text{gCO}_2\text{eq/s} \times t \times \frac{3.6 \times 10^6}{1000}, \tag{8}$$

where t denotes the measured execution time in seconds. This enabled consistent reporting across platforms. The resulting energy values for both Q-learning variants—baseline and footprint aware were stored as CSV files and later compared with the hardware-based measurements from the PMD-USB.

To ensure fair testing conditions, background services were minimised, and both scripts were executed under consistent settings. The baseline script ran for 100,000 episodes, while the optimised version incorporated ϵ decay and early stopping triggered at a distance threshold of $9{,}000km$, as described in Section III-B. These measures ensured compatibility between the two platforms and enabled a reliable evaluation of energy efficiency.

With both pipelines completed, hardware readings on Windows via PMD-USB and software estimates on Linux via CodeCarbon, the datasets were aligned for evaluation. Each tool assessed two algorithm variants: a standard implementation and a footprint-aware version using ϵ decay with early stopping.

IV. Results

This section presents the experimental results from evaluating the energy efficiency of Q-learning applied to TSP. Two measurement platforms were used: the PMD-USB device on Windows and the CodeCarbon library on Linux. Each platform assessed both a baseline and an energy-aware version of the algorithm, incorporating ϵ-decay and early stopping in the latter.

The results are organised into a hardware-based analysis using PMD-USB, followed by the software-based estimates from CodeCarbon, and concluding with a comparison of measurement accuracy and algorithmic behaviour across platforms.

A. Results from PMD-USB

The first setup used the PMD-USB device to measure power during Q-learning runs in MATLAB. Two variants were tested: a baseline with 100,000 episodes, and an optimised version using ϵ-decay and early stopping at a 9,000 km tour threshold.

The optimised model converged in 25,683 episodes—far fewer than the baseline and slightly outperformed it in tour quality, while significantly reducing energy use and execution time.

Figures 4 and 5 illustrate the power consumption profiles for both variants. The baseline maintained a steady draw throughout its extended training duration, whereas the footprint-aware version exhibited an initial power surge followed by rapid convergence and reduced activity.

The evolution of the tour distance across episodes is shown in Figures 6 and 7. These plots reveal that the optimised algorithm converged much faster to a viable solution while avoiding excessive computation.

B. Results from CodeCarbon Platform

The energy estimation carried out on the Linux platform was based on CodeCarbon's software tracking functionality, which provided detailed measurements regarding power usage and CO_2 emissions during the execution of both Q-learning variants.

Figure 8 and Figure 9 present the raw output logs obtained from CodeCarbon for the baseline and footprint-aware implementations, respectively. These logs indicate energy consumed by RAM, CPU, and the total system, expressed in kilowatt-hours and later converted into Joules. For the baseline run, the

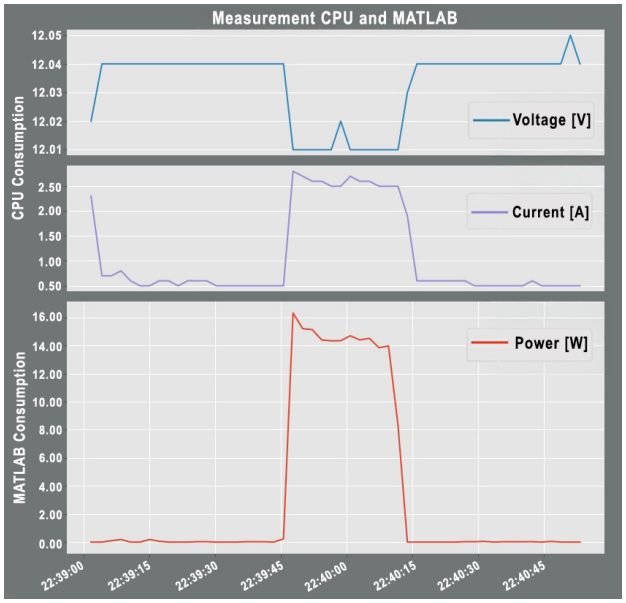

Fig. 4. Power usage over time during execution of the baseline Q-learning algorithm (no early stopping), measured using PMD-USB.

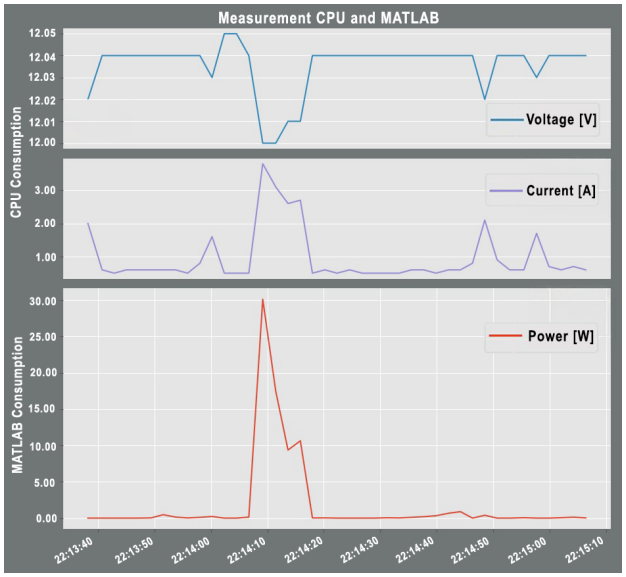

Fig. 5. Power usage over time during execution of the footprint-aware Q-learning algorithm, measured using PMD-USB.

process consumed approximately 901.69 J over 257.42 seconds, whereas the footprint-aware version consumed 289.31 J over 191.33 seconds.

In terms of learning behaviour, the evolution of the total travelled distance per episode was also recorded. Figure 10 shows the gradual improvement across 100,000 episodes in the baseline variant, while Figure 11 reflects how early stopping was triggered after approximately 52,000 episodes in the footprint-aware version, reaching a best distance of 8,793.33 km.

These results indicate that applying early stopping based on

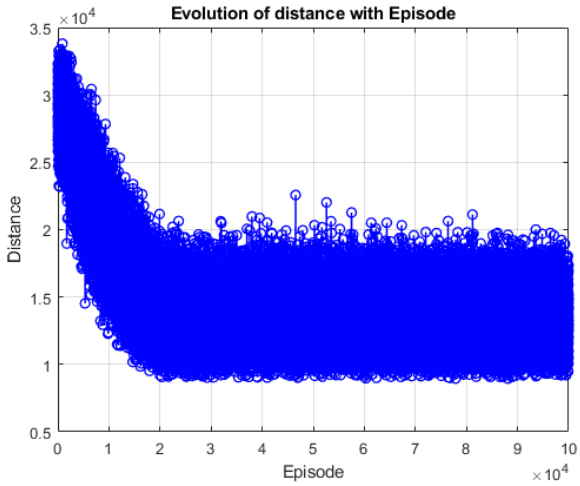

Fig. 6. Tour distance per episode for the baseline Q-learning algorithm using PMD-USB.

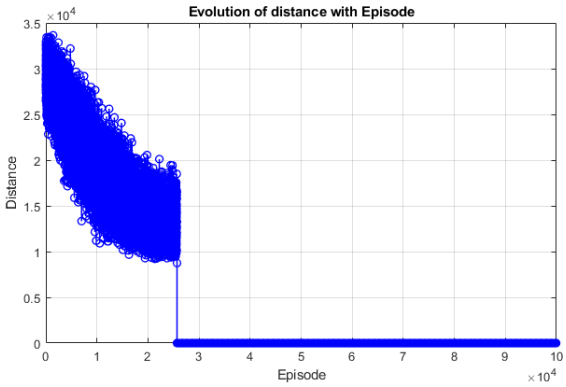

Fig. 7. Tour distance per episode for the footprint-aware Q-learning algorithm using PMD-USB.

distance threshold and ϵ-decay leads to a significant reduction in energy consumption (over 68%) while maintaining comparable solution quality. These insights confirm the viability of using CodeCarbon to detect efficiency gains derived from algorithmic modifications.

C. Comparison

A final comparison was performed to analyse the energy efficiency, execution time, and solution quality between the PMD-USB and CodeCarbon platforms. Table I consolidates the main performance indicators obtained across both tools for the baseline and footprint-aware variants of the Q-learning algorithm.

The footprint-aware variant outperformed the baseline in energy efficiency on both platforms. PMD-USB measurements showed a reduction from 192 J to 170 J (11.5%) and runtime from 28 s to 6 s. CodeCarbon estimates indicated a 68% drop (902 J to 289 J) and a runtime cut of over 25%.

Despite differing methodologies, hardware sampling vs. software inference both confirmed the benefits of ϵ-decay and early stopping. Yet, discrepancies (e.g., 289 J vs. 170 J)

```
Episode: 100000, Best Distance: 8886.19, Current Tour Distance: 10431.28
[codecarbon INFO @ 12:23:11] Energy consumed for RAM : 0.000415 kWh. RAM Power : 5.822549343109131 W
[codecarbon INFO @ 12:23:11] Energy consumed for all CPUs : 0.000220 kWh. Total CPU Power : 42.0 W
[codecarbon INFO @ 12:23:11] 0.000635 kWh of electricity used since the beginning.
Total energy consumed: 0.000183 kg CO2
CPU time used: 257.42 seconds
Estimated energy consumed by the process: 901.69 joules
```

Fig. 8. CodeCarbon terminal output for baseline Q-learning (100,000 episodes).

```
Episode: 52147, Best Distance: 8793.33, Current Tour Distance: 10349.81
[codecarbon INFO @ 12:54:39] Energy consumed for RAM : 0.000309 kWh. RAM Power : 5.822549343109131 W
[codecarbon INFO @ 12:54:39] Energy consumed for all CPUs : 0.000063 kWh. Total CPU Power : 42.0 W
[codecarbon INFO @ 12:54:39] 0.000372 kWh of electricity used since the beginning.
Total energy consumed: 0.000040 kg CO2
CPU time used: 191.33 seconds
Estimated energy consumed by the process: 289.31 joules
```

Fig. 9. CodeCarbon terminal output for footprint-aware Q-learning (target: 9,000 km).

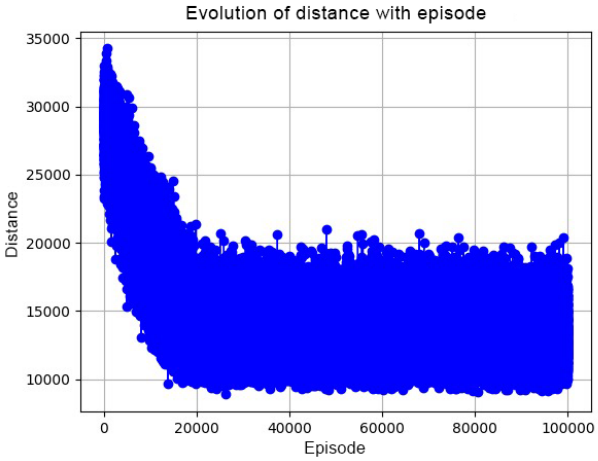

Fig. 10. Evolution of distance over 100,000 episodes for baseline Q-learning, measured using CodeCarbon.

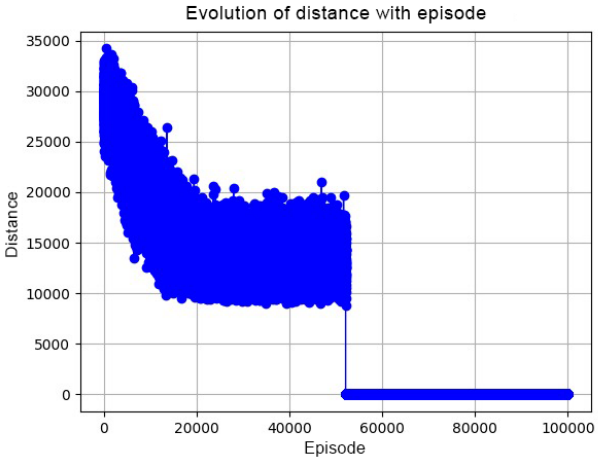

Fig. 11. Evolution of distance for footprint-aware Q-learning with ϵ-decay, measured using CodeCarbon.

TABLE I
CROSS-PLATFORM COMPARISON OF Q-LEARNING VARIANTS USING PMD-USB AND CODECARBON.

Metric	**Baseline**		**Footprint-Aware**	
	PMD	CC	PMD	CC
Episodes	100,000	100,000	25,683	52,147
Best Distance (km)	8,948	8,886	8,751	8,793
Time (s)	28	257	6	191
Energy (J)	192	902	170	289
Max Power (W)	16	NA	30	NA

suggest CodeCarbon may overestimate energy due to reliance on CPU metrics alone, ignoring voltage spikes and transient effects.

Several factors contribute to this gap. CodeCarbon uses average CPU load and lacks access to current and voltage data, limiting granularity. It may also include background activity or system overhead unrelated to the Q-learning process, with accuracy varying by environment.

In addition, the Linux implementation may have underperformed due to less efficient code structure or library use, compared to MATLAB's JIT-optimised execution. This could inflate software estimates.

Lastly, hardware-level tools like PMD-USB mitigate issues such as thermal throttling and task switching by capturing direct electrical signals, making them more reliable.

In summary, early stopping had the greatest impact by reducing training length, while ϵ-decay accelerated convergence. These strategies enabled substantial energy savings without degrading tour quality. The findings support validating software estimates with hardware tools and show that well-chosen parameters can enhance RL sustainability, an aspect often overlooked in prior TSP studies.

V. CONCLUSIONS

This work set out to explore the energy implications of Reinforcement Learning by focusing on Q-learning applied to

the Travelling Salesman Problem. Rather than relying solely on theoretical projections or abstract software estimates, a dual platform methodology was proposed, combining physical power measurements via the PMD-USB with software-level estimations using CodeCarbon. This approach enabled a clearer and more practical view of how different algorithmic strategies impact energy consumption in real-world settings.

The results showed that integrating ϵ-decay and early stopping led to significant energy savings: approximately 11% using the PMD-USB hardware setup, and up to 68% in the CodeCarbon environment. These reductions were achieved without degrading solution quality, reinforcing the idea that thoughtful algorithmic design can support more sustainable computing. To support transparency and further research, all scripts and configuration files have been made publicly available.

At the same time, some limitations must be recognised. The analysis focused on CPU-based systems and did not explore GPU acceleration or distributed training, both of which are common in modern ML workloads. Background processes and platform-specific variations may also have influenced the energy readings. Looking ahead, this framework could be extended to cover more advanced Reinforcement Learning models, such as DQN or actor-critic, and applied to broader hardware setups. Embedding energy-aware feedback mechanisms directly into training routines may also pave the way for more adaptive and efficient AI systems.

Acknowledgment

This publication has emanated from research conducted with the financial support of Taighde Éireann – Research Ireland under Grant numbers 18/CRT/6049 and 21/FFP-P/10065. Erivelton Nepomuceno was supported by the Brazilian Research Agency CNPq (Grant No. 444154/2024-8). Thalita Nazaré was supported by the John and Pat Hume Doctoral Award (WISH). André Ottoni was funded by Edital PROPPI 18/2024 - UFOP. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission. The authors are grateful to the Maynooth SPUR programme.

References

[1] T. Saheb, M. Dehghani, and T. Saheb, "Artificial intelligence for sustainable energy: A contextual topic modeling and content analysis," *Sustainable Computing: Informatics and Systems*, vol. 35, p. 100699, 2022.

[2] L. Belkhir and A. Elmeligi, "Assessing ict global emissions footprint: Trends to 2040 & recommendations," *Journal of Cleaner Production*, vol. 177, pp. 448–463, 2018.

[3] A. S. G. Andrae and T. Edler, "On global electricity usage of communication technology: Trends to 2030," *Challenges*, vol. 6, no. 1, pp. 117–157, 2015.

[4] S. T. Tzeiranaki, P. Bertoldi, L. Castellazzi, M. G. Torres, E. Clementi, and D. Paci, "Energy consumption and energy efficiency trends in the eu, 2000–2020," Joint Research Centre, European Commission, Luxembourg, Tech. Rep., 2022.

[5] J. G. Koomey, "Growth in data center electricity use 2005 to 2010," Analytics Press, Stanford, CA, USA, Tech. Rep., 2011.

[6] M. Avgerinou, P. Bertoldi, and L. Castellazzi, "Trends in data centre energy consumption under the european code of conduct for data centre energy efficiency," *Energies*, vol. 10, no. 10, p. 1470, 2017.

[7] T. Nazaré, J. Gadelha, E. Nepomuceno, and R. Lozi, "Green computing for energy transition: A survey," *IEEE Latin America Transactions*, vol. 21, no. 9, pp. 937–948, 2023.

[8] G. Raffin and D. Trystram, "Dissecting the software-based measurement of cpu energy consumption: A comparative analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 36, no. 1, pp. 96–107, 2025.

[9] Y. Wang *et al.*, "An energy-efficient transformer processor exploiting dynamic weak relevances in global attention," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, pp. 227–242, January 2023.

[10] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018.

[11] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*. Princeton, NJ, USA: Princeton University Press, 2007.

[12] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[13] T. Thein, M. M. Myo, S. Parvin, and A. Gawanmeh, "Reinforcement learning based methodology for energy-efficient resource allocation in cloud data centers," *Journal of King Saud University - Computer and Information Sciences*, vol. 32, no. 10, pp. 1127–1139, 2020.

[14] M. K. Moori, H. M. G. de A. Rocha, J. Schwarzrock, A. F. Lorenzon, and A. C. S. Beck, "Improving the efficiency of graph algorithm executions on high-performance computing," *Concurrency and Computation: Practice and Experience*, vol. 35, no. 18, p. e7419, 2023.

[15] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" 2019.

[16] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," 2018.

[17] M. Jay, V. Ostapenco, L. Lefevre, D. Trystram, A.-C. Orgerie, and B. Fichel, "An experimental comparison of software-based power meters: Focus on cpu and gpu," in *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2023, pp. 106–118.

[18] J. G. Koomey, "Estimating total power consumption by servers in the us and the world," Lawrence Berkeley National Laboratory, Berkeley, CA, USA, Tech. Rep., March 2007.

[19] V. M. Weaver, D. Terpstra, and S. V. Moore, "Non-determinism and overcount on modern hardware performance counter implementations," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 215–224.

[20] M. Dorigo and L. M. Gambardella, "Ant colonies for the traveling salesman problem," *Biosystems*, vol. 43, no. 2, pp. 73–81, 1997.

[21] J. H. Holland, *Adaptation in Natural and Artificial Systems*, 2nd ed. Cambridge, MA: MIT Press, 1992.

[22] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *IEEE Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[23] W. Yaqi, F. Baochuan, W. Zhengtian, and G. Shuang, "A review on energy-efficient technology in large data center," in *2018 Chinese Control And Decision Conference (CCDC)*. Shenyang, China: IEEE, June 2018, pp. 5109–5114.

[24] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, "Rapl in action: Experiences in using rapl for power measurements," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 3, no. 2, 2018.

[25] J. Browne, T. Nazaré, and E. Nepomuceno, "Experimental evaluation of horner's method for cpu energy reduction in nonlinear modelling," in *2024 35th Irish Signals and Systems Conference (ISSC)*, 2024, pp. 1–6.

[26] C. Watkins and P. Dayan, "Technical note: Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 05 1992.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–33, 02 2015.

[28] S. Santos, "Powertrack: Real-time process energy analysis," 2024. [Online]. Available: https://doi.org/10.17605/OSF.IO/ZR2U4