Optimization of code for scalability of deep reinforcement learning agents in slow and stochastic industrial process patterns

Guilherme Peniche Cordeiro[1*], Eduardo Mansur Ferreira Bittencourt Júnior[2], Michell Thompson Ferreira Santiago[3]

[1] *Jorge Amado University Center, Computer Science, Camaçari, Bahia, Brazil; guipeniche@hotmail.com*
[2] *Federal University of Bahia, Control and Process Automation Engineering, Lauro de Freitas, Bahia, Brazil; educof33@gmail.com*
[3] *Federal University of Bahia, Electrical and Computer Engineering, Salvador, Bahia, Brazil; michell.thompson@ufba.br*

Abstract: This work investigates the impact of code optimization techniques on the scalability and training time of deep reinforcement learning (DRL) agents applied to stochastic and slow-dynamic industrial process control environments. DRL combines deep learning and reinforcement learning to address high-dimensional decision-making problems, but it often demands significant computational resources and time, especially in complex industrial scenarios. To address these challenges, this research explores the use of vectorization, parallelism, and hyperparameter optimization via Optuna to improve performance and training efficiency. Experiments were conducted using the "TempControl-v0" environment from the PIDGym library, simulating thermal inertia processes akin to industrial furnaces. Key metrics analyzed include total training time, average episode reward, and CPU/GPU utilization. The results demonstrate that strategic code optimization can significantly enhance the performance and scalability of DRL agents, making their application more feasible in real-world industrial contexts.

Keywords: Code Optimization. Deep Reinforcement Learning. Hyperparameter Tuning. Industrial Process Control. Parallel Computing.

1.Introduction

Artificial Intelligence is a branch of computer science that seeks to build mechanisms, physical or digital, that simulate the human ability to think and make decisions. (Barbosa, Portes, 2019, p.17). These mechanisms generally depend on training and testing to build their knowledge and application in different areas.

In this context, within the field of Artificial Intelligence, a specific training approach known as Deep Reinforcement Learning (DRL) stands out. According to Arulkumaran et al. (2017), this technique has made it possible to solve problems that were previously intractable due to the large number of possible states and actions.

However, Reinforcement Learning (RL) combined with Deep Learning (DL) can become costly as the problem size increases. First, as argued by Bianchi and Costa (2004), unfortunately, the convergence of RL algorithms can only be achieved after extensive exploration of the state-action space, which is generally time-consuming. Moreover, DL algorithms are highlighted by Araújo (2024) as "one of the most resource- and energy-intensive tasks," further warning that training without hardware accelerators can take several days.

With this, performance bottlenecks can be critical when combining both methods to generate deep reinforcement learning. Especially when applied to slow and stochastic industrial

process routines. According to Silva (2024), a stochastic process is characterized by uncertainty and the description of the future in terms of probabilities that model the presence of randomness in the system. Subsequently, Cendron (2022) corroborates the idea that manufacturing industries face dynamic and stochastic production environments. That is, with random industrial environments, deep reinforcement learning may face long training periods due to the high rate of variables or states in process routines to be learned, promoting a slow learning dynamic to achieve a small error.

Parallel to the previous context, code optimization stands out. Passos et al. (2022) highlight in their study that in various applications, the need emerges to build a solution that is fast or that makes maximum use of available hardware, thus demanding performance optimization techniques. From this, the idea of optimizing code and/or system algorithms stands out, such as those of deep reinforcement learning in industrial processes, to improve performance, speed, and accuracy.

Therefore, the general objective of this research is to analyze the impact of code optimization on the scalability and training time of deep reinforcement learning agents applied to the control of slow and stochastic industrial processes. Additionally, the following specific objectives are highlighted for the development of this research: (i) to investigate vectorization, parallelism, and parameter tuning applicable to deep reinforcement learning algorithms, (ii) to evaluate the impact of these strategies on training time and agent performance through experiments, and (iii) to analyze the scalability of the optimized algorithms in different simulated industrial scenarios with varying levels of complexity.

2.Theoretical Foundation

2.1.Code Optimization Techniques

Among the aforementioned code optimization strategies, parallelism stands out as a primary approach. According to Wilkinson and Allen (2005), parallel programming, within the computational context, aims at executing multiple tasks or different parts of the same task, concurrently, leveraging multiple processing cores to enhance program execution and solve complex problems more efficiently in terms of time.

When considering the context of Deep Reinforcement Learning (DRL), there is clear evidence of success in solving problems that were once considered intractable. However, solving such problems imposes a high computational cost due to the large number of possible variables. According to Arulkumaran et al. (2017), “Deep learning enables RL to scale to decision-making problems that were previously intractable, i.e., settings with high-dimensional state and action spaces.” The training of these deep learning models not only demands computational and energy resources, but can also take a long time without proper hardware support, as previously mentioned by Araújo et

al. (2024). This highlights the importance of using GPUs and TPUs as strategies for optimizing time and resource consumption, as they are designed for performing massive operations simultaneously.

Considering the issues mentioned above, it is evident that the performance bottleneck can become even more critical when implementing DRL in the context of industrial processes characterized as stochastic and slow in dynamics. These processes involve uncertainty and high variability, requiring extensive exploration of the state and action space, which results in long training times. Therefore, code optimization through parallelism becomes essential for the scalability and feasibility of these agents in complex industrial environments. From this perspective, vectorization can be addressed. Allen and Kennedy (1984) approach vectorization as a transformation of serial code reordering, adapting it to be executed more efficiently on architectures that support vector operations, allowing a single processor instruction to operate on multiple data simultaneously, thus enabling the processor to process a set of elements in a single step, offering significant performance gains. Araújo et al. (2024) address the idea that DRL models, especially those with deep neural networks, execute a massive volume of numerical operations, making the importance of vectorization in optimizing this process notable, particularly based on observations made by Nuzman and Henderson (2005), who understand the effectiveness of vectorization when applied to repetitive operations, such as mathematical calculations on arrays and matrices.

Finally, parameter adjustments are evidenced through Optuna, which Akiba et al. (2019) break down its dynamic structure (define-by-run): "Optuna formulates the hyperparameter optimization as a process of minimizing/maximizing an objective function that takes a set of hyperparameters as an input and returns its (validation) score". This approach is essential for dealing with complex and dynamic environments, allowing the construction of search spaces that evolve according to the experiment. Another point to be highlighted is its efficient optimization with "pruning", which enables scalability, since it drastically reduces the time spent on unpromising episodes, as it prematurely interrupts experiments that present unsatisfactory performance, saving computational resources and accelerating agent convergence.

2.2.Deep reinforcement learning (DRL)

It is considered that deep reinforcement learning is the union of reinforcement learning techniques with deep learning. In this sense, Kaelbling, Littman and Moore (1996) state that reinforcement learning (RL) is "the problem faced by an agent that must learn behavior through trial-and-error interactions with a dynamic environment," that is, a behavior learning training based on trial-and-error iterations in a dynamic environment. In the

study on the use of RL in mobile robots, Faria and Romero (2002) highlight that this learning is based on the idea that if an action is followed by satisfactory states, or by an improvement in the state, then the tendency to produce this action is increased, that is, reinforced.

The action of a satisfactory state is increased or decreased from the numerous trial-and-error iterations within the state space and, sometimes, propagated through time. However, one of the necessary conditions in which RL algorithms can find an action policy is the complete exploration of the state space, normally impossible in practical situations (Faria and Romero, 2002). Practical situations such as in random industrial processes or those with slow dynamics can become challenges for RL, considering that the random state space is large and dynamic with high risk in industries with large-scale routines.

Regarding the second object of the DRL union, deep learning is described by Bochie et al. (2020) as an area of machine learning characterized by the extraction of complex representations from simpler representations. Pereira (2017) corroborates the concept by stating that this area can be considered as an intersection between the areas of neural networks, graphical modeling, optimization, pattern recognition, and signal processing.

As stated by LeCun, Bengio, and Hinton (2015), deep learning “is making major advances in solving problems that have resisted the best attempts of the artificial intelligence community for many years.” The authors explain that, ten years ago, this learning model began solving problems that had long resisted all attempts by the artificial intelligence community, highlighting the vast potential for applications in current problems that are still in the discovery phase.

However, regarding the challenges of deep learning, Bochie et al. (2020) state that typically, deep learning requires models that capture the non-linearities of the problem, making the modeling relatively more complex. This means that despite the advantages of deep learning, its application demands more complex models and longer training processes, resulting in higher computational requirements and additional difficulties in development.

Deep reinforcement learning is presented in the study by Arulkumaran et al. (2017) as "represents a step toward building autonomous systems with a higher-level understanding of the visual world". This statement emphasizes that DRL goes beyond simple visual data processing, developing a broader understanding of the environment that allows autonomous systems to make more effective decisions in complex situations.

Finally, Arulkumaran et al. (2017) highlight that "DRL can deal efficiently with the curse of dimensionality, unlike tabular and traditional nonparametric methods". The "curse of dimensionality" represents one of the main obstacles in traditional reinforcement learning problems, where the number of possible states

grows exponentially with the dimensionality of the problem. DRL overcomes this limitation through the ability of deep neural networks to approximate complex functions in high-dimensional spaces, enabling applications in real-world problems that would be intractable with conventional methods.

3.Methodology

3.1.Environment Structure

To investigate the impact of code optimization on the scalability of training deep reinforcement learning agents, a custom environment named "TempControlEnv" was developed using the "Gymnasium" library. The environment models the evolution of temperature over time based on the action applied by the agent, with variations that include efficient heating, thermal loss, random noise, and external disturbances. Aiming to incorporate high-precision computational bottlenecks, the environment assigns a quadratic term (temp_diff**2) to heat loss and a temperature update using the 4th-order Runge-Kutta method, which increases the complexity and computational cost of execution. Additionally, at each simulation step, matrix multiplication and singular value decomposition are performed to create a heavy and constant CPU workload.

The action space has one dimension and is continuous, represented by a value between -1 and 1, thus corresponding to the intensity of cooling or heating. It can vary according to the enabled configurations, having in its basic form two main variables: the temperature error relative to the setpoint and the derivative of the error, allowing the modulation of the environment's complexity, making it suitable for comparative performance testing of algorithms.

The environment features a reward function composed of a penalty proportional to the absolute error, an additional penalty for large variations between sequential actions, extra reward when the error is below a precision threshold, and an additional penalty for oscillation, based on the standard deviation of the error history.

3.2.Comparison Metrics

The metrics used for analysis within this environment are the training time to demonstrate the effectiveness of optimizations and the average reward per episode, mentioned by Kayal et al. (2025) as fundamental for measuring the agent's convergence and effectiveness by summing the total rewards received by the agent throughout an episode, along with the reward standard deviation to measure control stability and penalize oscillations.

Araújo et al. (2024) address another analysis point that involves the variation in computational performance based on GPU/CPU usage, since the chosen optimization techniques aim to result in better CPU core utilization and better hardware bandwidth usage, which should directly impact CPU usage rate and reduce the average load on the GPU/CPU.

To develop the evaluation of the impact based on training time and performance, timing

functions were used to record the total training time. Regarding the reward per episode, as mentioned by Kayal et al. (2025), the rewards are accumulated from each episode for all parallel actors and then averaged across the actors.

As for GPU/CPU usage metrics, performance analysis is conducted using the Python libraries "psutil" and "pynvml," with "psutil" responsible for monitoring the average CPU usage throughout agent training and execution, and "pynvml" enabling access to GPU utilization metrics. These tools allow for a correlation between the applied optimizations and the computational resources used, enabling a detailed analysis of the performance of the applied techniques.

3.3.Models for Testing

For this study, four models were selected: the "Base" model without any optimizations; the "Optuna" model using the tool of the same name along with simple vectorization via "DummyVecEnv"; the "Parallel" model using the aforementioned parallelism technique in combination with vectorization implemented through the "SubprocVecEnv" (Subprocess Vectorized Environment) tool; and finally, the "Parallel + Optuna" model combining both methods.

4.Results

Table 1 - Results for each method

Configuration	Average Reward	Average time(s)	Average CPU(%)
Base	-36298.78	310,9	36,7
Optuna	-35823.4	282,4	45,3
Paralelo	-37111.97	153,7	5,3
Paralelo + Optuna	-37174.18	120,1	21,7

Standard deviation of rewards was also calculated across configurations. The Base model had the highest deviation with 919.69, while Optuna had the lowest with 661.35. The Parallel and Parallel + Optuna models had deviations close to the base, with 898.89 and 907.11 respectively.

From the table, the impact of optimization strategies on training efficiency can be observed. It shows that the version using "Optuna" achieved a higher average reward and lower variability in standard deviation. On the other hand, "Parallelism + Optuna" was the fastest in training time, while the "Parallelism" test had the lowest CPU usage (5.34%).

5.Discussion

5.1. Impact of Optimizations on Training Time

Based on the results obtained, a drastic reduction in training time is evident in the approaches using parallelism, clearly demonstrating the influence of parallel environments on training time reduction, with a clear decrease compared

to the baseline. This approach proves essential for scaling DRL agents in complex environments.

5.2.Impact of Optimizations on Agent Performance

Another noteworthy point is the performance gain through hyperparameter optimization (Optuna test), resulting in more effective control and outperforming the baseline as indicated by the average reward and standard deviation values.

5.3.Speed vs. Quality

Given the results presented, there is an evident trade-off between training time efficiency and the final performance of the agent. The hyperparameter optimization approach achieved the highest reward and lowest standard deviation, while the approaches using parallelism did not reach this performance despite being faster. This is likely due to the way experiences are collected. In sequential training, although slower, exploration is more focused since the gradient used to update the neural network is based on experiences from a more consistent policy, favoring higher-quality convergence. In parallel training, experiences are collected simultaneously from different trajectories and time steps, which speeds up data collection but may result in poorer gradients, causing the agent to converge to a good solution, but not necessarily the best possible one.

6.Conclusion

In conclusion, this work achieves its main objective of investigating the impact of code optimization techniques—specifically parallelism, hyperparameter tuning, and vectorization—on the scalability and performance of deep reinforcement learning agents when applied to industrial environments with slow and stochastic dynamics. Hyperparameter optimization via Optuna, combined with simple vectorization, proved to be crucial for improving the agent's final performance, resulting in a more effective and stable control policy, as evidenced by the average reward and standard deviation.

Parallelism, in turn, implemented through a complex vectorized environment, proved to be an indispensable tool for scalability, drastically reducing training time. Another key point is the identification of a trade-off between training speed and solution quality, where fast, parallelism-based approaches did not produce agents with satisfactory final rewards.

It is therefore understood that the choice of optimization strategies is contextual to the project’s priorities. When rapid prototyping and agile interaction are desired, parallelism is the superior approach; however, for maximum performance extraction, hyperparameter optimization becomes more advantageous. These findings have direct implications for the practical application of DRL, where development time and final control effectiveness are critical factors.

References

[1] AKAIWA, Takuya et al. Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining. 2019. p. 2623-2631.

[2] ALLEN, John R.; KENNEDY, Ken. Automatic loop interchange. In: Proceedings of the 1984 SIGPLAN symposium on Compiler construction. 1984. p. 233-246.

[3] ARAÚJO, Thiago et al. Otimizando Aplicações de Aprendizado Profundo na Cloud. In: Escola Regional de Alto Desempenho da Região Sul (ERAD-RS). SBC; 2024. p. 131-132.

[4] ARULKUMARAN, K.; DEISENROTH, M. P.; BRUNDAGE, M.; BHARATH, A. A. Deep reinforcement learning: a brief survey. IEEE Signal Processing Magazine. 2017 Nov;34(6):26-38. ISSN 1053-5888.

[5] BARBOSA, Lucia Martins; PORTES, Luiza Alves Ferreira. A inteligência artificial. Revista Tecnologia Educacional [on line]. 2023;236:16-27.

[6] BIANCHI, Reinaldo A. C.; COSTA, Anna H. R. Uso de heurísticas para a aceleração do aprendizado por reforço. In: XXV Congresso da Sociedade Brasileira de Computação. São Paulo; 2005. p. 130-139.

[7] BOCHIE, Kaylani et al. Aprendizado profundo em redes desafiadoras: Conceitos e aplicações. Sociedade Brasileira de Computação; 2020.

[8] CENDRON, Jean Carlos et al. Algoritmo genético e simulação de eventos discretos: uma abordagem híbrida aplicada na otimização de layout e processos industriais. 2022.

[9] FARIA, Gedson; ROMERO, Roseli A. Francelin. Navegação de robôs móveis utilizando aprendizado por reforço e lógica fuzzi. Sba: Controle & Automação Sociedade Brasileira de Automatica. 2002;13:219-230.

[10] KAELBLING, Leslie Pack; LITTMAN, Michael L.; MOORE, Andrew W. Reinforcement learning: A survey. Journal of artificial intelligence research. 1996;4:237-285.

[11] KAYAL, Aya; PIGNATELLI, Eduardo; TONI, Laura. The impact of intrinsic rewards on exploration in Reinforcement Learning. Neural Computing and Applications. 2025:1-35.

[12] LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. Deep learning. Nature. 2015;521(7553):436-444.

[13] PASSOS, Arthur M. et al. Análise e aplicação de técnicas de otimização de código. 2022.

[14] PEREIRA, Matheus de Mattos. Aprendizado profundo: redes LSTM. Dourados, MS: Universidade Federal da Grande Dourados; 2017. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação).

[15] SILVA, Júlio Cesar Martins Oliveira da et al. Cadeias de Markov na engenharia de produção um panorama das Aplicações. 2024.

[16] WILKINSON, Barry; ALLEN, Michael. Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers. 2. ed. Upper Saddle River, NJ: Pearson Prentice Hall; 2005.