A performance model for deadline-aware off-policy reinforcement learning with vectorized environments

1st Klavdiya Bochenina
Computer Science Department
University of Helsinki
Helsinki, Finland
klavdiia.bochenina@helsinki.fi

2nd Volodymyr Beimuk
Computer Science Department
University of Helsinki
Helsinki, Finland
volodymyr.beimuk@helsinki.fi

3rd Laura Ruotsalainen
Computer Science Department
University of Helsinki
Helsinki, Finland
laura.ruotsalainen@helsinki.fi

***Abstract*—Vectorized environments in modern reinforcement learning (RL) frameworks are widely used to accelerate training for computationally expensive simulators. In many real-world applications, RL policies require periodic retraining under strict operational constraints. To ensure timely deployment, it becomes essential to accurately predict RL training time under limited computational resources.**

In this work, we propose a performance model for off-policy RL with vectorized environments, that decomposes training time into learner, overhead, and simulation components, while supporting parameterized decision frequencies. Using the ride-pooling case study evaluated on the MAHTI and LUMI supercomputers, we demonstrate that the model generalizes across multiple problem configurations with MAPE$<$15% for all experimental settings. We further show how the calibrated model enables deadline-aware configuration selection by predicting the maximum feasible number of training steps to meet a prescribed training-time deadline.

***Index Terms*—performance model, reinforcement learning, parallel training, off-policy learning, vectorized environments**

I. Introduction

Reinforcement learning (RL) has gained significant popularity for a variety of optimization problems in dynamic environments [1]. Training an RL algorithm requires a large number of interactions between the RL agent and the environment during the sampling process. One of the ways to accelerate the training process is to perform sampling in multiple independent environments in parallel. This approach is implemented in modern RL frameworks such as StableBaselines and Gym under the name of vectorized environments.

To enable systematic analysis of training time and principled experiment planning, we propose a performance model for off-policy RL with vectorized environments (Section III). The model relates training time to the number of environments, training steps, and gradient updates, while explicitly accounting for tunable decision-step granularity. Once calibrated, it enables deadline-aware configuration selection by predicting achievable training budgets under fixed time constraints. While we demonstrate our approach using ride-pooling with the SUMO traffic simulator, the performance model is application-agnostic and applicable to any off-policy RL algorithm with vectorized environments.

Our main contributions include: (i) an analytical performance model decomposing training time into learner, overhead, and simulation components, (ii) a continuous scaling formulation that interpolates between episode-invariant and step-invariant training budgets, (iii) closed-form expressions for deadline-aware parameter selection, (iv) experimental validation at MAHTI and LUMI supercomputers, achieving prediction errors below 15% MAPE for the selected benchmark.

The code and processed experimental data used in this study are available via an online repository [2].

This work was supported by the Academy of Finland Flagship program: Finnish Center for Artificial Intelligence FCAI and the Academy of Finland project 365406 AI-based optimization tool for sustainable urban planning (AIOSUT), and the Department of Computer Science, University of Helsinki. The authors acknowledge the research environment provided by ELLIS Institute Finland.

II. Method

A. Problem formulation

Training RL algorithms for real-world applications often requires meeting strict time constraints while exploring different training configurations. In such settings, it is essential to understand how training time depends on the structure of the training pipeline, the degree of parallelism, and the decision frequency.

The training pipeline considered in this study is illustrated in Figure 1. It uses n environments per vectorized step. At each step, the RL agent receives a vector of n observations and outputs a vector of n actions, which are sent to n independent environment instances. As a result, each vectorized step produces n transitions instead of one in a sequential version, significantly increasing the throughput of RL sampling.

In this study, we assume that the RL task requires experimenting with different decision frequencies. For example, actions may be computed and applied every second, minute, or several minutes. We formalize it using a parameter Δ, defined as the number of simulation iterations between consecutive RL decision steps. During sampling (Figure 1), an action is applied and simulation is performed for Δ iterations, resulting in a simulation time equal to $\Delta \cdot t_{sim}$. After this, rewards and observations for the next decision step are calculated, and gradient steps are performed (t_{rl} denotes the time of learning updates). Sampling also introduces overhead t_{over} arising from

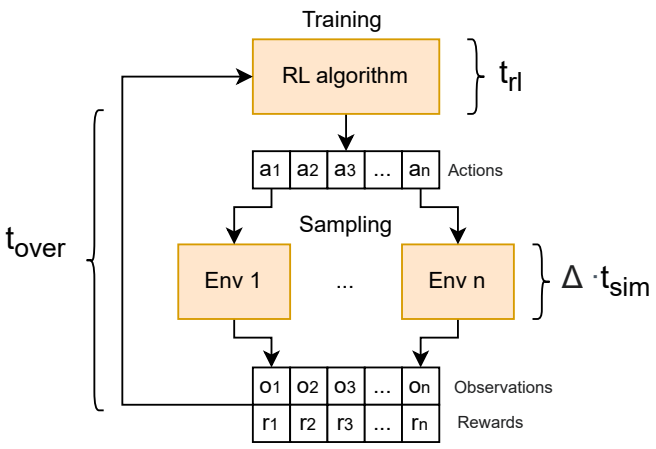

Fig. 1. Scheme of a vectorized algorithm

initialization, communication, and coordination of n parallel environments.

With vectorization and tunable decision frequency, estimating training time becomes challenging yet essential for deadline-aware deployment. We therefore formulate the following problem. Given a configuration (n, D, Δ, p_{RL}), where D is a training-time deadline and p_{RL} are RL training parameters (e.g., learning rate and batch size), determine the maximum number of training (decision) steps that can be completed within the deadline:

$$K^* = \max\{K : t_{train}(K, n, \Delta, p_{RL}) \leq D\}.$$

III. Parallel performance model

In this study, we focus on modeling training-time feasibility rather than learning dynamics. We therefore adopt a throughput-based view in which n parallel environments generate n transitions per vectorized step, and analyze the time required to execute a fixed number K training steps of the RL algorithm. Each training step consists of Δ simulation iterations, so the total number of simulation iterations for training is equal to $K \cdot \Delta$. One simulation run (one episode) consists of a fixed number of iterations I. If we denote the number of episodes as E, the following expression holds[1]:

$$K \cdot \Delta = E \cdot I. \tag{1}$$

While E is often used as a proxy for training budget, a fixed number of episodes corresponds to a different number of decision steps for different Δ. Then, for the problems with a tunable granularity we use the number of decision steps K to compare training budgets across different Δ.

If we denote the baseline number of episodes for $\Delta = 1$ as E_1, then for $\Delta > 1$ two boundary cases can be distinguished: (i) the number of episodes remains fixed at E_1, leading to a

[1]For example, if one episode represents 1 hour ($I = 3600$ seconds), we apply new action every 5 minutes ($\Delta = 300$ seconds), and train RL agent for $E = 500$ episodes, the number of decision steps K will be equal to $\frac{500 \cdot 3600}{300} = 6000$, with 12 different actions per episode.

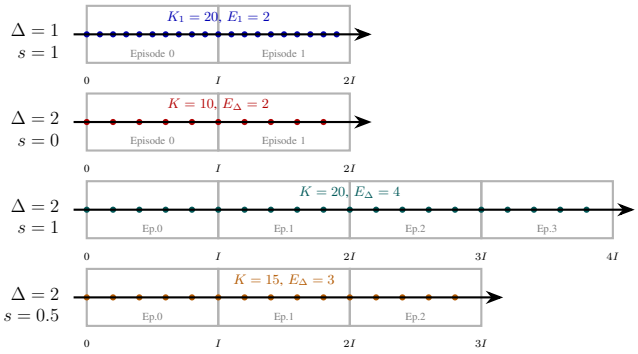

Fig. 2. Effect of scaling coefficient on training duration for different Δ

reduction of K by a factor of Δ, (ii) the number of episodes is increased to fully compensate for this reduction, preserving the total number of decision steps. To systematically interpolate between these cases and enable continuous tradeoff between training time and decision resolution, we introduce scaling coefficient $s \in [0; 1]$ which defines the number of decision steps for $\Delta > 1$ as:

$$K(\Delta, s) = K_1 \cdot \left(\frac{1-s}{\Delta} + s\right), \tag{2}$$

where K_1 is the number of steps for $\Delta = 1$.

Introducing s allows us to express deadline feasibility in a way that is independent of absolute episode counts and highlights how much additional decision resolution can be afforded for coarser temporal granularities. When $s = 0$, the total number of episodes $E_\Delta = E_1$ will be kept during the training across Δ, and when $s = 1$, the total number of decision steps $K_\Delta = K_1$ will be kept (Figure 2).

Given that K steps (each of Δ iterations) are distributed between n workers, the training time for n workers can be calculated as (see also Figure 1):

$$t_n(\Delta, s) = \\ = \underbrace{\frac{K(\Delta, s)}{n}}_{\text{\# of vec steps}} \cdot \left(\underbrace{g_{step} \cdot t_{rl}}_{\text{learner updates}} + \underbrace{t_{over}}_{\text{overhead}} + \underbrace{\Delta \cdot t_{sim}}_{\text{parallel sim}}\right). \tag{3}$$

Here, t_{rl} is the time of one gradient update, g_{step} is the number of gradient updates per one vectorized step, t_{over} is the total overhead per vectorized step, t_{sim} is a simulation time per one iteration (thus, $\Delta \cdot t_{sim}$ is a simulation time per one vectorized step). g_{step} depends on RL training parameters and can be calculated as:

$$g_{step} = \frac{gs}{tf}, \tag{4}$$

where gs is the number of gradient updates performed every tf vectorized steps. The model assumes constant per-step times t_{sim} and t_{rl}, which represent average execution times.

For single-node vectorized environments, we neglect communication time, so t_{over} captures environment initialization and processes' synchronization times. At the end of each

vectorized step, processes wait at the barrier, so waiting time scales with n. Moreover, longer per-process simulation time increases the variability of completion times controlled by Δ. Thus, we approximate t_{over} as a linear function of n and Δ. Note that with this approximation, $\frac{K}{n} t_{over} \approx \frac{K}{n}\alpha_1 \Delta n = \alpha_1 K \Delta = \alpha_1 E I$ (by (1)), i.e., the overhead term scales proportionally to the number of episodes E for fixed episode length I. This allows us to aggregate episode-level environment initialization (over E resets) and per-step synchronization overhead into a single effective term.

Assuming that t_{rl} and t_{sim} are constants for a fixed problem size, we formulate the performance model as:

$$t_n(\Delta, s) = \frac{K(\Delta, s)}{n} \left(\alpha_0 \cdot g_{step} + \alpha_1 \cdot \Delta \cdot n + \alpha_2 \cdot \Delta\right). \quad (5)$$

The coefficients α are estimated for fixed problem size across varying Δ and n.

For a given deadline D, using Eq. (5), we can calculate $s^*(\Delta, D, n)$ – the maximum scaling coefficient so that the training time will not exceed the deadline:

$$s^*(\Delta, D, n) = \frac{\Delta n D - K_1 Z}{K_1 Z (\Delta - 1)}, \quad (6)$$

where $Z = \alpha_0 g_{step} + \alpha_1 \Delta n + \alpha_2 \Delta$. If $s^* < 0$, the deadline is unfeasible. If $s^* > 1$, the deadline is larger than the necessary time to keep K_1 decision steps, and we can set $s = 1$. For the special case, when we want to keep training time the same as for $\Delta = 1$, the following expression holds:

$$s^*(\Delta, D_{\Delta=1}, n) = \frac{\alpha_0 g_{step}}{\alpha_0 g_{step} + \alpha_1\ \Delta n + \alpha_2 \Delta}. \quad (7)$$

IV. Experimental study

A. On-demand ride-pooling case study

As a case study to evaluate the performance model for off-policy vectorized RL, we use an on-demand ride-pooling method proposed in [3]. Ride-pooling refers to the problem of serving a set of passengers using a fleet of taxis where passengers may share a ride. In an on-demand setting, passenger requests are not known in advance, requiring the taxi fleet to be controlled through a sequence of decision steps.

In the considered approach, reinforcement learning selects the parameters $\mathbf{p}$ (e.g. maximum capacity, maximum pick-up distance) of a centralized dispatching heuristic. These parameters are applied to the entire taxi fleet over the interval $(t, t + \Delta)$. After Δ simulation iterations, observations are collected and provided to the RL algorithm together with the reward from the previous step.

In this study, we used six observation variables representing the global state of demand and supply, a one-dimensional action (setting the maximum occupancy of a taxi fleet), and a two-component reward defined as a weighted sum of passenger satisfaction and taxi fleet utilization.

Ride-pooling is particularly suitable for evaluating the proposed performance model because: (i) it relies on computationally expensive SUMO traffic simulation, (ii) it naturally supports tunable decision frequency Δ, and (iii) deployment requires periodic retraining under training-time deadlines.

The experimental study was conducted on a representative urban area (5 × 3.5 km) of Helsinki using realistic passenger demand extracted from a previously developed SUMO traffic model for the morning rush hour [4], using LUMI and MAHTI supercomputers. Experiments were performed by sampling a given fraction r of the original demand; the fractions of taxi passengers and taxis were fixed as 0.3 and 0.1, respectively. We use DQN with MLPPolicy as a basic RL algorithm for metaheuristic method [3] (learning rate 0.0001, exploration fraction 0.3, buffer size 200000).

B. Deadline-aware training planning

In this section, we evaluate the prediction accuracy of the proposed performance model, analyze the contribution of its components across different numbers of environments, and demonstrate its use for deadline-aware training-time planning. While results are presented for the selected case study, the performance model is application-agnostic, because it captures fundamental computational components which are presented in any off-policy RL training pipeline regardless of domain.

The parameters of the experiments[2] are presented in Table I. Four considered values of (tf, gs) represent different degrees of learning aggressiveness: $(1, n)$ and $(32/n, 32)$ perform one gradient update per transition with a batch size of 1 and 32, $(2, 1)$ and $(4, 1)$ represent more conservative schemes with one gradient update every 2 and 4 vectorized steps.

TABLE I
Deadline-aware planning: experiment parameters

Parameter	Description	Values
n	Number of environments	1, 2, 4, 8, 16, 32
r	Sampled fraction of the original demand	0.2
Δ	Number of simulation iterations per decision step	1, 3, 9, 30
I	Number of simulation iterations per episode	3000
s	Scaling coefficient	Δ = 3: s = 0, 0.25, ..., 1 Δ = 9: s = 0, 0.0625, 0.125, 0.1875, 0.25 Δ = 30; s = 0, 0.017, 0.034, 0.051, 0.068.
E_1	Number of episodes for $\Delta = 1$	128, 256, 1024
(tf, gs)	(training frequency, gradient steps per update)	(1,n), (32/n, 32), (2,1), (4,1)

Once calibrated, the performance model (5) can be used to predict training time for varying number of decision steps K (and scaling coefficient s). To demonstrate its applicability for deadline-aware planning, we fit the coefficient vector α for $s = 0$ and other parameters given in Table I, and evaluate the predictive accuracy for $s > 0$.

We compare the proposed model against a linear speedup baseline, which assumes perfect parallelization and predicts

[2]Values of s are selected such that the maximum number of episodes for $\Delta \in 3, 9, 30$ equals to $E_1 \cdot c$, where $c = 3$ (thus, $s_{max} = 1$ for $\Delta = 3$). In this setting, $s_{max} = (c - 1)/(\Delta - 1)$.

training time as the measured time for $n = 1$ divided by n. Our model consistently yields substantially lower prediction errors (Table II), highlighting the importance of modeling gradient computation t_{rl} and overhead t_{over}. Table II and Figure 3 show that the model (5) generalizes across different n, Δ, and s, with MAPE below 15% and MAE below 3 hours across all configurations, corresponding to only a few hours for 35–40 hour runs, which is sufficient for deadline-aware planning.

TABLE II
PREDICTION ACCURACY OF THE PERFORMANCE MODEL

Model	(tf,gs)	MAPE	MAE (hrs)	RMSE (hrs)
Our model	$(1, n)$	7.05%	1.21	1.74
Baseline	$(1, n)$	38.51%	4.69	6.26
Our model	$(32/n, 32)$	9.75%	1.01	2.16
Baseline	$(32/n, 32)$	78.19%	7.35	10.70
Our model	$(2, 1)$	10.84%	2.44	3.34
Baseline	$(2, 1)$	52.50%	8.95	10.72
Our model	$(4, 1)$	10.73%	2.48	3.62
Baseline	$(4, 1)$	53.48%	9.17	10.76

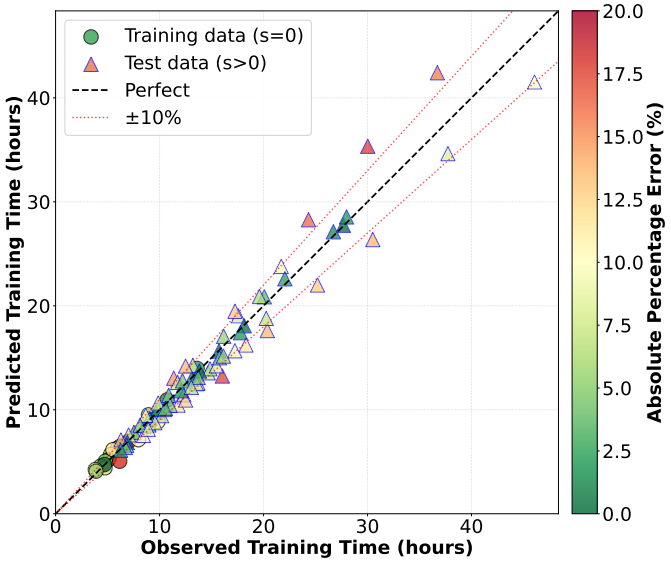

Fig. 3. Comparison of predicted and observed training times, $(tf, gs) = (1, n)$. Fitted coefficients: $\alpha_0 = 0.002091, \alpha_1 = 0.003616, \alpha_2 = 0.009960$. For test data: $R^2 = 0.9531$, MAPE = 7.05%.

Figure 4 shows component contributions for $\Delta = 9$. As predicted by the model, gradient computation (α_0) and overhead (α_1) remain constant with respect to n, while simulation (α_2) scales as $1/n$. The inset demonstrates that gradient contribution scales as $1/\Delta$: from about 29% at $\Delta = 1$ to about 1% at $\Delta = 30$, validating Eq. (5).

Thus, using Eq. (6), we can calculate $s^*(\Delta, D, n)$ – the maximum scaling coefficient s for a given decision granularity Δ such that training completes before a given deadline D for the selected number of environments n. Figure 5 illustrates this procedure for fixed D and n. The value of s^* shows where the feasible number of decision steps lies within the interval $[K_1/\Delta, K_1]$ for the given Δ and D. For example, for $\Delta = 3$

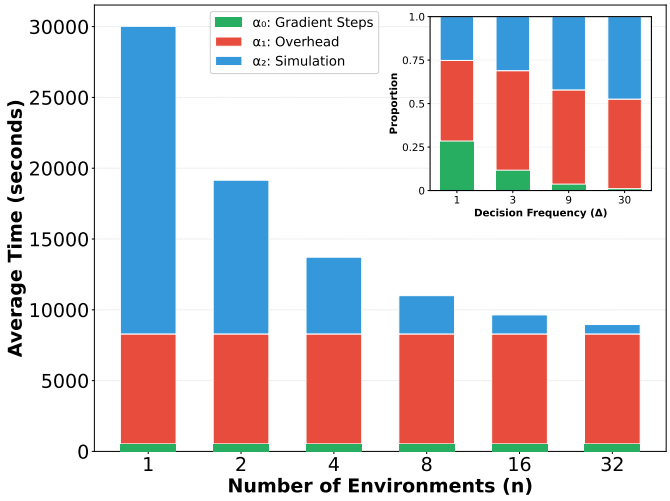

Fig. 4. Contribution of gradient computation (α_0), overhead (α_1), simulation (α_2) components to the overall training time for $\Delta = 9$, $(tf, gs) = (32/n, 32)$. The values for every Δ are averaged for different s. In-plot shows relative contributions for $\Delta = 1, 3, 9, 30$.

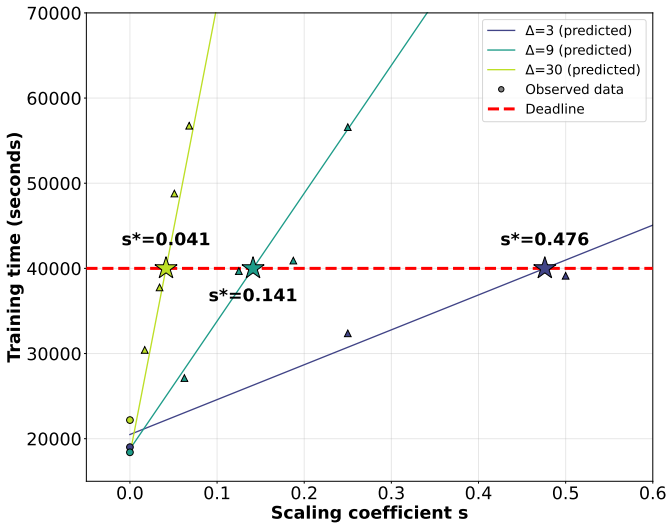

Fig. 5. Deadline-aware planning example ($E_1 = 1024, n = 8, D = 40000$, $\Delta = 3, 9, 30$). s^* are calculated with Eq. (6) from the fitted performance model. Markers show observed runtimes.

$s = 0$ corresponds to an episode-invariant budget with $K = K_1/3$ decision steps, while $s = 1$ corresponds to $K = K_1$. For intermediate values, the relative decision-step budget is given by $K/K_1 = s + (1 - s)/\Delta$, see Eq. (2). Under the selected deadline, it yields feasible budgets of 65%, 24%, and 7% of the $\Delta = 1$ decision steps for $\Delta = 3, 9$ and 30, respectively.

Figure 6 illustrates the special case in which the deadline is equal to the training time for $\Delta = 1$. In this setting, Eq. (7) is used to perform time-invariant granularity tuning, i.e., to determine s^* for each Δ such that the resulting training time matches the baseline budget for $\Delta = 1$. The value of s^* increases as Δ decreases, since for larger Δ the number of decision steps compensated via s constitutes a smaller fraction of the total number of compensated steps. Figure 6 also shows

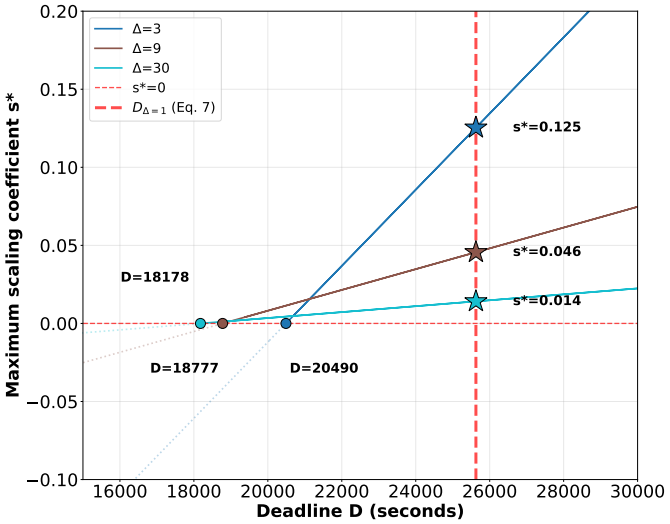

Fig. 6. Time-invariant granularity tuning example ($n = 8$, $\Delta = 3, 9, 30$). Values of s^* are calculated using Eq. (7) from the fitted performance model. Circle markers show predicted training times for $s = 0$ (where the number of episodes equals that of $\Delta = 1$ baseline, $E_\Delta = E_1$).

that, despite the fact that for $s = 0$ the number of episodes is identical for all Δ, the predicted training times differ and increase as Δ decreases. This is explained by the fact that, while the α_1 (overhead) and α_2 (simulation) terms remain constant for $s = 0$, the learner term α_0 scales as $1/\Delta$.

Vectorized environments increase RL training throughput by generating more transitions per unit time. To study this effect at scale, we conducted an additional experiment on the LUMI supercomputer with up to $n = 128$ environments. Rather than analyzing speedup for a fixed training budget, we scale the number of episodes with the number of environments, $E(\Delta, n) = nE(\Delta, 1)$, to characterize data-generation throughput. The fraction of sampled demand r was varied from 0.1 to 0.5 to represent different environment complexities. Using the fitted model (5), we compute observed and predicted average time per episode for different n. Figure 7 shows that, in contrast to a fixed-budget setting, increasing n improves throughput by amortizing per-episode overhead, until saturation is reached when simulation cost dominates (e.g., $n = 64$ for $r = 0.1$ and $n = 128$ for $r = 0.2$).

C. Training quality and stability

While the proposed model focuses on training-time feasibility rather than learning quality, it is necessary to verify that feasible plans correspond to non-trivial policy learning for the considered case study. Figure 8 shows a representative training and evaluation curve for a selected configuration ($\Delta = 9$, $E_1 = 1024$, $tf = 4$, $gs = 1$).

To investigate the dependency of the learning quality and stability on scaling coefficient s, we divided the values of s from 0 to 0.5 at three bins of equal length (Low, Mid and High) and calculate the following per-bin metrics: (i) evaluation mean reward for last 10% of evaluation points, EMR10%; (ii) advantage of EMR10% over the random baseline, ARB; (iii) coefficient of variation (ratio of the standard deviation

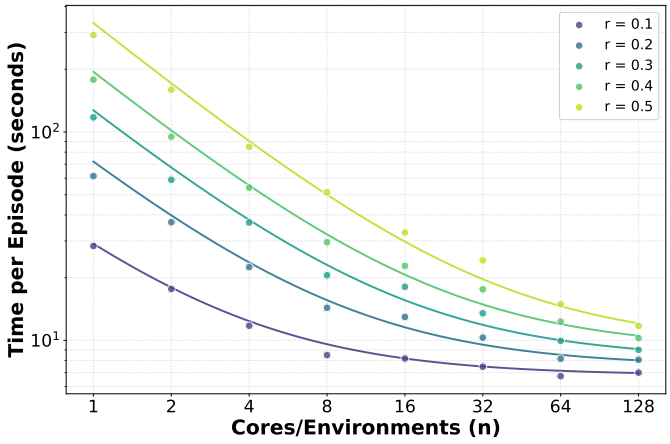

Fig. 7. Average time per episode, fitted versus observed, for varying n and fraction of the sampled demand r, $\Delta = 1$.

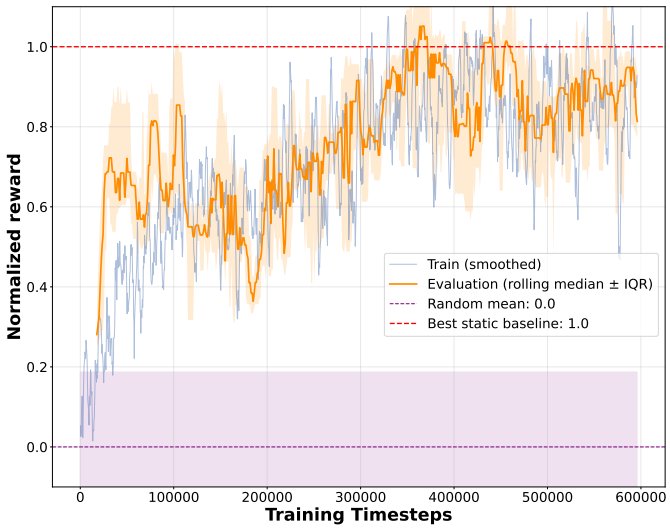

Fig. 8. Normalized training and evaluation performance for $\Delta = 9$, $E_1 = 1024$, $n = 4$, $s = 0.09375$, $tf = 4$, $gs = 1$. Rewards are normalized with respect to a random policy (0) and the best static baseline (1), corresponding to near-optimal solution (see [3]). Smoothing window length is equal to 10.

to the absolute value of its mean) of EMR10%; (iv) robustness score (ratio of mean reward to standard deviation) of EMR10%, ROB; (v) success rate, SR. CV measures relative reward variability, while ROB additionally preserves the sign of the mean reward. A run is marked as successful if ARB > 0 and CV < 1.5.

Table III presents the metrics per scaling coefficient bins. The highest final quality (EMR10%) is consistently achieved for the Mid s bin, indicating a non-monotonic dependence on s, while stability metrics improve for the High s bin. This example shows that for the selected configuration different values of s represent different tradeoffs between the quality, stability and speed of training. Thus, the proposed training time prediction model can be further used with configuration-dependent quality/stability model to increase the efficiency of resource usage.

TABLE III
LEARNING QUALITY AND STABILITY METRICS

s bin	N	**EMR**	**ARB**	**CV**	**ROB**	**SR**
Low	23	1489.0±103.3	684.8±134.3	0.487	2.90	100%
Mid	20	1530.6±50.3	725.0±112.1	0.418	3.07	100%
High	18	1480.8±82.5	648.0±110.5	0.346	3.28	95.8%

V. RELATED WORK

Distributed training acceleration for deep RL (DRL) is achieved via data, model or pipeline parallelism [5]. High-throughput DRL methods mainly rely on data parallelism, using multiple RL environments (actors) with synchronous (e.g., GA2C [6]) or asynchronous (e.g., IMPALA [7]) interaction. Distributed off-policy methods (such as Ape-X [8] and R2D2 [9]) further decouple data collection and learning via replay buffers in multi-node settings. In contrast, off-policy learning with vectorized environments, considered in this work, typically operates in a single-node synchronous setting without explicit actor/learner separation. Although widely supported in frameworks such as Stable-Baselines and Gym, their training-time scalability has not been studied from a performance modeling perspective. Existing RL system works report empirical throughput but do not provide predictive models of training time, while evaluation-focused studies (e.g., [10], [11]) emphasize benchmarking and reproducibility rather than computational efficiency.

Finally, the problem of planning RL training under strict time constraints has received limited attention in the literature. There exist performance modeling techniques that allow prediction of parallel program runtime based on the statistics of the runs, program time tracking or infrastructure simulators [12] which are widely used e.g. for deadline-aware scheduling of parallel scientific workflows [13]. However, these ideas have not been systematically applied to DRL training pipelines. In particular, studies introducing novel parallel and distributed RL frameworks (see e.g. [14]) usually report the observed training times or throughput for fixed configurations but do not address the inverse problem of selecting training budgets (e.g. number of decision steps and environments) to meet a prescribed deadline. The present work addresses this gap by providing an analytical parallel performance model that enables deadline-aware planning for off-policy RL with vectorized environments.

VI. CONCLUSION

In this study, we formulate and investigate the problem of deadline-aware planning for off-policy RL with vectorized environments and tunable decision frequency. We propose an analytical parallel performance model that enables prediction of training time and its decomposition into learner, overhead, and simulation components across problem configurations and number of environments. The calibrated model achieves good validation accuracy (below 15% MAPE for the ride-pooling use case across all experimental settings) and provides closed-form expressions for deadline-aware parameter selection. The proposed framework is particularly relevant for scenarios when retraining must be performed repeatedly under strict operational deadlines. For practitioners, the recommended workflow is: (i) calibrate model on small-scale experiments, (ii) use Eq. (6) to select feasible (n, Δ, s), (iii) validate quality on selected configuration.

As discussed in Section IV-C, training quality and stability can exhibit non-monotonic and highly configuration-dependent behavior with respect to the scaling coefficient s. This observation highlights the need to complement training time models with separate quality or stability surrogates in order to further increase the efficiency of computational resource usage. Further work may evaluate the model for the different off-policy algorithms (such as DDPG, SAC, and TD3) as well as extend the proposed model to multi-node settings with heterogeneous communication costs.

REFERENCES

[1] S. Padakandla, "A survey of reinforcement learning algorithms for dynamically varying environments," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–25, 2021.

[2] K. Bochenina, V. Beimuk, and L. Routsalainen, "A performance model for deadline-aware off-policy reinforcement learning with vectorized environments," https://github.com/helsinki-sda-group/pfm-dl-rl, 2026.

[3] K. Bochenina and L. Ruotsalainen, "A reinforcement learning-based metaheuristic algorithm for on-demand ride-pooling," in *2024 International Conference on Intelligent Environments (IE)*, 2024, pp. 117–123.

[4] K. Bochenina, A. Taleiko, and L. Ruotsalainen, "Simulation-based origin-destination matrix reduction: a case study of helsinki city area," in *SUMO Conference Proceedings*, vol. 4, 2023, pp. 1–13.

[5] Z. Liu, X. Xu, P. Qiao, and D. Li, "Acceleration for deep reinforcement learning using parallel and distributed computing: A survey," *ACM Computing Surveys*, vol. 57, no. 4, pp. 1–35, 2024.

[6] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, "Ga3c: Gpu-based a3c for deep reinforcement learning," *CoRR abs/1611.06256*, 2016.

[7] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," in *International conference on machine learning*. PMLR, 2018, pp. 1407–1416.

[8] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver, "Distributed prioritized experience replay," *arXiv preprint arXiv:1803.00933*, 2018.

[9] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney, "Recurrent experience replay in distributed reinforcement learning," in *International conference on learning representations*, 2018.

[10] R. Gorsane, O. Mahjoub, R. J. de Kock, R. Dubb, S. Singh, and A. Pretorius, "Towards a standardised performance evaluation protocol for cooperative marl," *Advances in Neural Information Processing Systems*, vol. 35, pp. 5510–5521, 2022.

[11] S. Jordan, Y. Chandak, D. Cohen, M. Zhang, and P. Thomas, "Evaluating the performance of reinforcement learning algorithms," in *International Conference on Machine Learning*. PMLR, 2020, pp. 4962–4973.

[12] B. Gu, L. Zhao, C. Li, and X. Chi, "Review and analysis of performance prediction methods and tools for heterogeneous parallel programs," *CCF Transactions on High Performance Computing*, pp. 1–21, 2026.

[13] E. Saeedizade and M. Ashtiani, "Scientific workflow scheduling algorithms in cloud environments: a comprehensive taxonomy, survey, and future directions," *Journal of Scheduling*, vol. 28, no. 1, pp. 1–63, 2025.

[14] J. Kwok, M. Lohstroh, and E. A. Lee, "Efficient parallel reinforcement learning framework using the reactor model," in *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures*, 2024, pp. 41–51.