5 REINFORCEMENT LEARNING FOR ADAPTIVE TIME-STEPPING OF BRIDGED CLUSTER DYNAMICS SIMULATIONS

Work in preparation by Veronica Saz Ulibarrena, Simon Portegies Zwart. Reprinted here in its entirety.

ABSTRACT

Astrophysical simulations often involve multi-scale and multi-physics scenarios, which entails the need for methods that can handle those. A common approach involves separating the system into multiple subsystems according to their characteristics, with each subsystem being integrated using problem-specific methods. Then, the systems are coupled on a given time scale. The coupling time scale, or coupling integration time, is determined manually and remains fixed throughout the simulation. In this work, we introduce a novel approach that leverages reinforcement learning techniques to automatically select the coupling time step during simulations. Our method effectively balances computation time and accuracy by adapting the time-step size to the characteristics of the simulation at each step. We test our method on a multi-scale problem: a star cluster in which one star contains a planetary system. We perform multiple tests on clusters with different (low) numbers of bodies and find that the method remains robust for multiple cases. Additionally, we test the effect of changing the integrators for the subsystems and find that our method is independent of this choice. Similarly, we implement a parameter that scales the time steps to tune the accuracy requirements of the problem and find that our method remains valid in this case. For long-term integration, where energy errors tend to accumulate due to the chaotic nature of the problem, we find that our reinforcement learning method can achieve better results than the methods with fixed-time steps, but all cases lead to large energy errors in time. We develop a hybrid strategy that can detect sudden jumps in energy error and recursively reduce the time-step size at a given step to prevent them. The hybrid method achieves performances orders of magnitude better compared to standard methods, without significantly increasing the computation time. This method en-

sures the robustness of simulations that use reinforcement learning techniques. With our method, we eliminate the need for expert knowledge and achieve simulations that balance computation time and accuracy while adapting to the needs of the simulation at each step. This method can be directly extrapolated to a large range of astrophysical simulations.

5.1 Introduction

Astrophysics simulations are becoming more complex with time. By increasing the number of bodies involved, or the physical phenomena taken into consideration, the results obtained are more true to the physical reality and therefore easier to compare with observational results. This increase in complexity leads to the need for more efficient methods that can take into consideration the computational limitations of the problem to be studied.

Whereas 4th order direct integration methods through such as `Hermite` Makino & Aarseth (1992), `Ph4`, and `Brutus` Boekholt & Portegies Zwart (2015b) can result in very accurate simulations (with the right choice of simulation parameters), they are not suitable for simulating large systems. The computational effort of direct integration methods scales with N^2, with N being the number of bodies in the system. For example, the computational effort to integrate a star cluster, which can contain from dozens to millions of stars, can lead to simulations that last several months depending on the time scale over which it is integrated. To circumvent this problem, some methods have been designed to speed up this computation at the cost of accuracy. For example, the hierarchical tree algorithm from `Barnes-Hut Tree Code` Barnes & Hut (1986) groups bodies that are far away, whereas interactions from bodies close to each other are calculated directly.

Another problem arises when a system is composed of multiple hierarchical systems. For example a planetary system with moons, and a star cluster with planetary systems, among others. In these cases, the difference in scales leads to a decision problem: to integrate the system on the scale of the largest subsystem to speed up the calculation while missing the subtleties of the smallest subsystem, or to integrate the system using a time-step size consistent with the scale of the smallest subsystem, leading to impractically-large computation times. This problem can be effectively addressed by using methods that can separate these subsystems to allow the integration of each of those using the right integration settings. Examples of these methods are `Lonely planets` [cite], `Nemesis` Zwart et al. (2020), and `Bridge`, a method designed by Fujii et al. (2007), and which we will further look into in Subsection 5.2.1. Similarly, many of these methods can be used when there are many physical phenomena involved in the problem, such as gravitational interactions and hydrodynamical processes, radiation processes, stellar evolution, etc.

These methods provide a solution that allows to numerically integrate systems “separately". However, those still need to communicate with each other and ensure that each subsystem includes the effect of its environment. This coupling is dealt with differently in various methods. A common challenge is finding the time scale at which those systems need to interact (or communicate with each other) to keep our simulation accurate enough for our purpose, depending on the scientific question, while keeping the computational cost low. The choice of this coupling time step is normally made manually based on expert knowledge and kept fixed throughout the simulation. As explained in Saz Ulibarrena

& Portegies Zwart (2024), most chaotic systems benefit from a variable time-step size Heggie & Hut (2003); Aarseth (2003). Indeed, most integration codes nowadays include some internal calculation of the time-step size to ensure that it adapts to the conditions of the problem. Examples are found in Aarseth & Lecar (1975); Pelupessy et al. (2012), where the free-fall time of pairs of particles is used as a measure of how closely particles are interacting and used to estimate an adequate step size. However, this type of solution is not available for coupling codes such as `Bridge` yet.

Machine Learning (ML) is being studied as a method to speed up simulations in many fields. It is common to use ML methods as a proxy to replace the integrator to improve efficiency by skipping expensive calculations Cai et al. (2021b); Greydanus et al. (2019a); Ulibarrena et al. (2024). Additionally, an interesting application consists of using ML methods to replace choices that are otherwise made manually. When setting up a simulation, there are many choices to be made; from the initial conditions, to the integrator, to the time-step size. Many of these choices are made based on expert knowledge, whereas other parameters are often left as their default value. This can lead to poor results or inefficient simulations. Additionally, expert knowledge is not always available for every experiment.

Reinforcement Learning (RL) methods are used to make choices automatically based on some values to be optimized. One of the most important parameters to be chosen in a simulation is the time-step size. As mentioned before, many integrators include the automatic calculation of the optimum time-step size at each step, but there is no current solution to obtain an estimation for coupling systems. We develop a Reinforcement Learning algorithm that automatically chooses this time-step size. In addition to adapting to the conditions of the problem at each step and providing a solution that optimizes accuracy and computation time, our method reduces the expert knowledge needed to run these simulations.

We focus on the case of a star cluster in which some stars have planetary systems orbiting them. Using the `Bridge` method, we train a reinforcement learning algorithm to choose the optimum time-step size at each simulation step.

In Section 5.2, we explain in detail our implementation of the `Bridge` method, the initial conditions chosen for the problem, the integration settings, and the main parameters involved in the training of a RL algorithm. In Section 5.3, we show the training of the RL algorithm and the results of applying the trained method to our simulations. We compare the performance of the trained model for a variety of initializations, integration scales, and the number of stars in the cluster. We also study the use of our trained RL algorithm in a multi-physics scenario: a star surrounded by a proto-planetary disk. Finally in Section 5.5, we discuss the possible uses and limitations of the method created and summarize the goals and results of our work.

The code is publicly available at `https://github.com/veronicasaz/RL_bridgedCluster`github.com/veronicasaz/RL_bridgedCluster.

5.2 Methodology

In our method, we couple different subsystems and integrate a reinforcement learning algorithm into the simulation.

5.2.1 System setup

We look into systems in which different scales are involved. Specifically, a cluster of stars where some have planets orbiting them. Integrating such a system becomes quadratically more expensive as the number of bodies increases. Therefore, optimizing the computation time becomes an essential part of astronomical simulations. Additionally, the choice of time-step size should be based on the fundamental scales of the system. This scale can be radically different for the star cluster and the individual planetary systems. Hence, it is relevant to look into methods that allow the separation into multiple subsystems.

We take as our integration method the `Bridge` as defined in `AMUSE` Portegies Zwart & McMillan (2018); Portegies Zwart et al. (2009); Zwart et al. (2013). It is a method that separates a system into different subsystems. Each of them can then be integrated using a different code and integration settings. This allows the use of the most adequate methods in systems where there are fundamental differences in the behavior of the subsystems that form it. For example, it can be used when there are different physical processes involved. An example is a cluster in which some stars have a disk of material around them, where gravity and hydrodynamics have to be coupled. Another example in which a `Bridge` results useful is when the system consists of different scales. Integrating a large system separately from a smaller one allows for a better choice of integration parameters. A more detailed description of the `Bridge` can be found in Fujii et al. (2007); Zwart et al. (2013).

The fundamental idea of `Bridge` is that a system is divided into different subsystems, which are then integrated separately. After a certain time (Δt_B), the acceleration caused by one subsystem onto the other is calculated and used to modify the velocity of each particle. Then the subsystems are evolved again individually and the process is repeated until a final time is reached.

Although the bridge can be used in many cases: such as when coupling multi-physics phenomena, in many cases, we find that one particle should be included in the integration of both subsystems. This application is not implemented in `Bridge`. To solve that, we develop a "`inclusive Bridge`" in which one particle is common to both subsystems. In this case, in addition to exchanging information about the potential between subsystems, we use the `Bridge` to update the state of the common particle, as seen in Figure 5.1. For the case of a cluster of stars in which one star has a planetary system, we divide the system into two subsystems: the star cluster and the planetary system, respectively. Then, we calculate the potential of the cluster on the planetary system and use it to update the velocities of the planets and central star. We then evolve the planetary system using the adopted integrator for a time Δt_B and update the state of the common particle in the cluster subsystem. For the next step, we ignore the effect of the planets on the star cluster, and proceed to evolve (drift) the start cluster for a time Δt_B. Finally, we use the latest state of the star cluster to update the particles in the planetary system by drifting its center of mass. This process is repeated until the final time or maximum number of steps is reached. From this point onward, every mention of the `Bridge` will refer to our `inclusive Bridge`.

In Figure 5.2, we show a comparison of our `inclusive Bridge` against the `Bridge` method as implemented in `AMUSE` and with direct integration, i.e., integrating the whole system together. We can see that our method achieves orders of magnitude better accuracy than the regular `Bridge`. Nevertheless, both lead to energy errors orders of magnitude

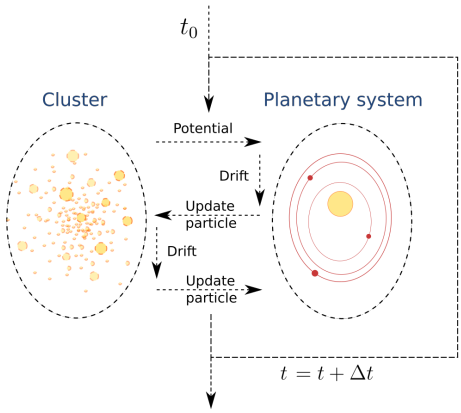


Figure 5.1: Schematic of the `(inclusive) Bridge` method as developed for this application. The system is divided into two subsystems: a star cluster and a planetary system, with one star being common for both. The potential of the cluster on the planetary system is calculated and used to update the velocities of the particles in the planetary system. Then the planets and central star are evolved and the state of the central star in the cluster subsystem is updated with that of the planetary system. Afterwards, the cluster subsystem is evolved and the state of the center of mass of the planetary system is updated using the latest cluster information. This process is repeated for a number of steps.

larger than direct integration. The error in both `Bridge` methods compared to direct integration results from them being very simple coupling strategies. We will discuss the advantages and shortcomings of using this method compared to some more complex ones in Section 5.5. This error is also derived from the fact that the `Bridge` time step is not adapting to the needs of the problem, whereas the direct integration case implements an adaptable time-stepping strategy. We will aim to improve on this using RL. The advantage of the `Bridge` with respect to direct integration is the possibility of using several integrators. Specially, in the case when different physical processes are present it is not possible to use direct integration. Secondly, the `Bridge` method is an approximation made to reduce the computation time in systems with large N. This advantage cannot be seen in Figure 5.2 as the number of bodies is not large enough. Additionally, it is also not a fair comparison as the direct integrator is developed in a high-performing language, whereas the `Bridge` methods are `Python` implementations. This results in better computation times with direct integration than with both `Bridge` implementations. In Section 5.5, Figure 5.19 we show a comparison of the performance (accuracy and computation time) for our `inclusive Bridge` method compared to direct integration and the RL implementation for different numbers of bodies. Although the advantage of using approximate methods appears for a larger N, for the purpose of this work, we limit ourselves to N between 5 and 15 to simplify the analysis.

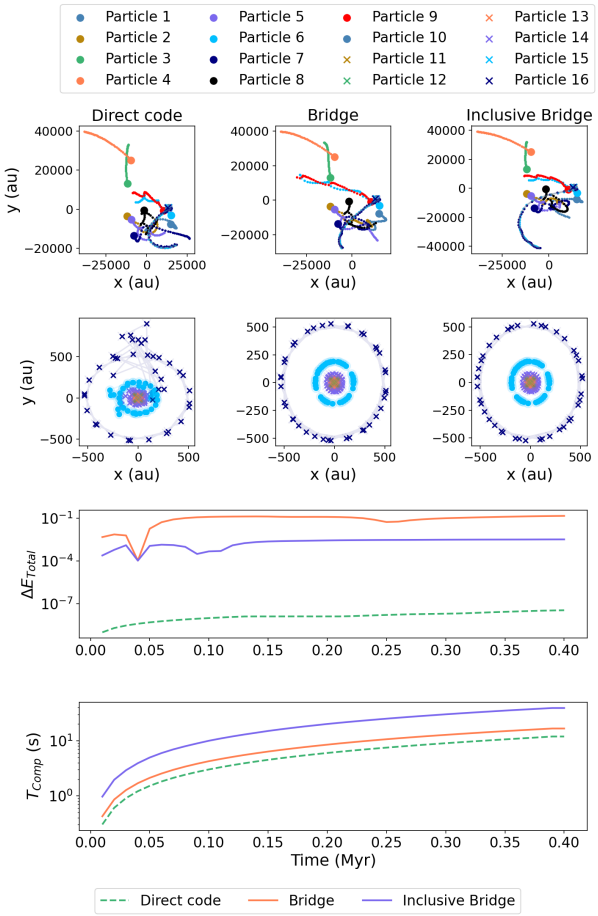


Figure 5.2: Comparison of our `(inclusive) Bridge` against the `Bridge` method as implemented in `AMUSE` and with direct integration.

Table 5.1: Initial conditions and integration settings of a cluster with one planetary system orbiting one of the stars.

STAR CLUSTER	
NUMBER OF STARS	[5 - 20]
MASS RANGE OF THE STARS	[1, 100] M_{Sun}
RADIUS OF THE CLUSTER	0.1 PARSEC
VIRIAL RATIO	0.5

PLANETARY SYSTEM	
INNER DISK RADIUS	10 AU
OUTER DISK RADIUS	100 AU
DISK MASS	0.02 M_{Sun}

INTEGRATION	
CLUSTER CODE	PH4
CLUSTER CODE ε	10^{-2}
PLANETARY SYSTEM CODE	HUAYNO
PLANETARY SYSTEM CODE ε	10^{-2}
CHECK STEP SIZE	$10^{-2}\ Myr$

5.2.2 Scientific setup

We take a simplified example of a star cluster with a small number of stars (between 5 and 20) and place a planetary system around one of those stars. Note that only the effect of Newtonian gravity is taken into consideration.

The masses of the stars are chosen following a power-law (Salpeter (1955)) distribution and the cluster is initialized using a fractal cluster model Goodwin & Whitworth (2004) with the values indicated in Table 5.1. Then, the last star is chosen as the common one between subsystems, and a planetary system is created around it assuming an oligarchical planetary growth Tremaine (2015); Kokubo & Ida (2002).

Each subsystem is evolved with a different integrator. For the star cluster we use `Ph4` integrator as implemented in `AMUSE` Portegies Zwart & McMillan (2018). This code is a direct integration code with internal calculation of the time step for each particle. For the planetary system, we use the `AMUSE` implementation of `Huayno` Jänes et al. (2014), which is well suited for the integration of planetary systems. Each code is initialized with a time-step parameter (ε) as indicated in Table 5.1, which scales the internally-calculated time-step sizes. The check step size refers to the frequency at which the state of the system is saved and the time-step size of the `Bridge` re-evaluated. This step has no relation with the actual integration steps of either subsystem or the `Bridge`.

There are many time steps involved in this setup:

- **Time-step size of the cluster subsystem**: this is the time-step used for the integration of the star cluster. When using `Ph4`, this time step is calculated internally and multiplied by a scaling parameter (ε).

- **Time-step size of the planetary subsystem**: this is the time-step used for the integration of the planetary system. When using `Huayno`, this time step is calculated internally and multiplied by a scaling parameter (ε) which might be different from the one for the cluster.

- **Bridge time step** Δt_B: this is the time scale on which the two subsystems exchange information. This means, how often the method calculates the potential of one system on the other and updates the common particle. This time-step size has to be chosen at the start of the simulation and there is no current solution to choose

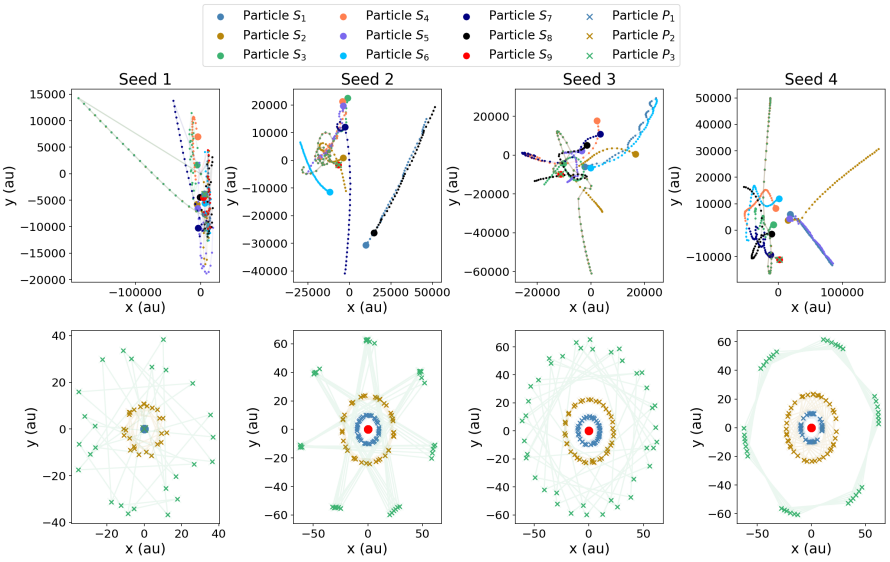


Figure 5.3: Initializations for Seeds 1 to 4 run for 40 steps (0.4 Myr) with a `Bridge` time-step of 5×10^{-5} Myr. The setup is formed by 9 stars and 3 planets.

an optimal value. This is the time-step size that we will determine using the RL algorithm.

- **Check step size**: we apply this denomination to the time scale we use to save the state of the system and to apply the reinforcement learning method. This means, after how much time we want to re-evaluate the choice of `Bridge` time step.

Using the aforementioned initial conditions, we plot in Figure 5.3 four examples of the integration of this system with 9 stars for seeds 1 to 4. The top row represents the evolution of the positions of the particles in the cluster. The second row shows the evolution of the planetary system. The stars are represented by a circle and the planets by an "x". This system is fundamentally chaotic, which will lead to large differences in its evolution depending on the initial conditions and time-step sizes. As a result of the system being chaotic, the optimal `Bridge` time-step size is different depending on the initialization. Additionally, since the conditions vary quickly in time, the optimal time-step size might also vary along the simulation.

5.2.3 Energy error

Since the dynamics is chaotic, the only method to evaluate the accuracy of a simulation is based on conservation laws. We use the energy error as an indication of accuracy. A discussion on the limitations of this metric will follow in Section 5.5.

The use of numerical integrators leads to energy errors. A large value of the energy error is an indication of unphysical solutions. We can therefore understand the energy error as a measurement of accuracy, or the validity of our simulation. The total energy error

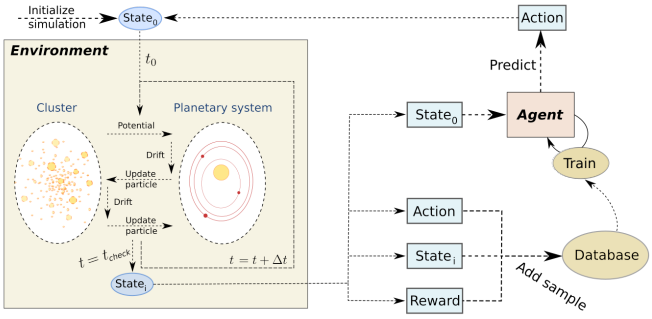


Figure 5.4: Schematic of the interaction between the Environment and the Agent.

is the sum of the kinetic and potential energy of the system. In this work, we denominate energy error to the relative difference of the energy at time step i and the initial time-step as

$$\Delta E_i = \frac{(E_{k,i} + E_{p,i}) - (E_{k,0} + E_{p,0})}{E_{k,0} + E_{p,0}} = \frac{E_i - E_0}{E_0}. \tag{5.1}$$

5.2.4 Reinforcement Learning

We train a reinforcement learning (RL) method to estimate the optimal `Bridge` time-step size. We choose Deep Q-learning as our RL algorithm to maximize a reward value ***R*** Sutton & Barto (2018). By supplying the algorithm with information from different astronomical simulations, it learns to take actions that maximize the reward. We adopt the specific implementation of Q-learning called Deep Q-networks (DQN) to allow for a continuous observation space Mnih et al. (2015). More information about Deep Q-learning and the reasoning behind this choice can be found in Saz Ulibarrena & Portegies Zwart (2024).

There are different elements interacting in the DQN method, as seen in Figure 5.4:

- **Environment**: the environment is composed of the astronomy simulations. The data obtained from them is used to create a dataset of the states, rewards, and actions. The composition of the environment is as explained in Subsection 5.2.2 and the astronomical simulations are initialized randomly during the training.

- **Agent**: the agent is the reinforcement learning algorithm that is trained to choose the actions. It is composed by two neural networks; namely the Q-net and the Target net. The weights of the Q-net are updated at each training step, whereas the Target net is only updated after an arbitrary number of steps. Both networks receive as input the State (S) of the system generated by the environment and produce the denominated Q-values associated with each possible action. Then, the action with the largest Q-value is selected and used for the next step of the simulation. The reward function obtained from the environment is used as the loss value to train the networks.

To create the agent we have to choose the hyperparameters of the neural networks as well as other training parameters. The specific values chosen will be mentioned in Subsection 5.3.2.

- **State**: the state is the representation of the environment that is used as an input to the neural networks in the agent. It must be formed by values that are representative of the physical state of the problem at a given time.

  In Saz Ulibarrena & Portegies Zwart (2024), as in many studies dealing with neural networks in the gravitational *N*-body problem, the state is chosen to be the cartesian coordinates representing the positions and velocities of each particle of the system. However, this leads to a fundamental problem: the input size is dependent on *N*, leading to limited extrapolation capabilities. To circumvent this problem, we define the state (***S***) as

$$\mathbf{S} = \left[\sum_{i}^{N-1} V_{n_i \to n_c} \quad , \quad -\log_{10}(\Delta E) \right] \tag{5.2}$$

  where $\sum_i^{N-1} V_{n_i \to n_c}$ is the gravitational potential of every star in the cluster (n_i) at the position of the common star (n_c). The second term is the current energy error of the simulation. This term is included to take into account that once the error is at a certain value, it is not likely to be reduced by large amounts. It is therefore important for the RL algorithm to consider this to avoid incurring unnecessary computational cost. A more detailed discussion on this remark can be found in Saz Ulibarrena & Portegies Zwart (2024).

- **Actions**: the actions (***A***) are the possible values of a decision variable. The reinforcement learning algorithm is trained to choose between these values to optimize a reward function. The actions are taken from a finite-size array which contains the value of the control variable associated with each action. Our decision variable is the `Bridge` time-step size. At each step, an action is chosen to determine the value of Δt_B to be taken for the next steps of the system simulation. The number of actions allowed as well as the minimum and maximum values are shown for each case in Subsection 5.3.2.

- **Reward function**: the reward is the function to be optimized by reinforcement learning. For the study in hand, we want to balance accuracy and computation time. Therefore, we take a function that balances both the energy error and the computation time. We write this function as in Saz Ulibarrena & Portegies Zwart (2024)

$$\mathbf{R} = -W_1 \frac{\log_{10}\left(|\Delta E|/10^{-10}\right)}{|\log_{10}(|\Delta E|)|^3} + W_2 \frac{1}{\log_{10}(A)}. \tag{5.3}$$

  The first term corresponds to the energy error at a given step normalized by 10^{-10} and divided by the cube of the energy error. The logarithm is used to linearize the range of values in this term. The second term corresponds to the computation time represented by the inverse of the time-step size taken. $W_{1,2}$ are the weights used to balance these two terms. They are a design choice and the used values can be found in Subsection 5.3.2. This reward function is specifically designed for the problem

of the simulation of a number of bodies interacting via their gravitational forces Saz Ulibarrena & Portegies Zwart (2024).

5.3 Results

We show the results obtained from training the RL algorithm and its application to different cases of the start cluster simulation.

5.3.1 Validation of the results

There is no analytical solution to the problem of a group of bodies ($N > 2$) interacting via Newtonian gravity. Additionally, it is chaotic. This means that finding a baseline with which to compare the results becomes challenging. Once the RL algorithm is trained, its results can be compared with those obtained without the use of RL. However, different values of the time step(s) will lead to radically different outcomes. To simplify this problem, we fix the values of the time-step parameters of the integrators involved (see Table 5.1) and study the effect of the `Bridge` time-step (Δt_B) on the outcome of the simulations.

We perform a convergence study to further understand the effect of Δt_B on the accuracy and computation time. Additionally, we want to understand which range of values should be used for the actions ($\boldsymbol{A}$). To do so, we simulate the systems with initializations with seeds 1 to 4 (see Figure 5.3) for 40 steps using different values of Δt_B. We expect to find a value of Δt_B for which the energy error does not improve further if being reduced, but instead just results in larger computation times. Then we can say that we have found a converged solution. This definition is not rigorous, as in order to find a truly converged solution in a chaotic system we would need to use an algorithm with arbitrary precision such as `Brutus` Boekholt & Portegies Zwart (2015b).

We show the results in Figure 5.5. The value of Δt_B for which the simulation converges depends on the initialization. Some cases such as the one with seed 4 are more chaotic, which results in the minimum value of Δt_B considered not being small enough to find convergence in the energy error. However, for a case such as the one with seed 1, convergence is reached for relatively large values of Δt_B. This study supports the idea that the optimal choice of time step largely depends on the initial conditions.

Although it is difficult to get a clear conclusion about the range of values that should be used for the given simulations, we can observe that in most cases convergence is reached for time step sizes between 10^{-4} and 10^{-5} Myr. We therefore choose an intermediate value of 5×10^{-5} as the lowest limit for the RL actions. Regarding the higher limit, we can see that depending on the simulation a time step size of 10^{-2} Myr may yield large energy errors. We choose this value as the upper limit for the RL actions.

5.3.2 Training results

We defined the environment in Subsection 5.2.2 and the RL method in Subsection 5.2.4. With that information, and the settings in Table 5.2, we obtain the trained models.

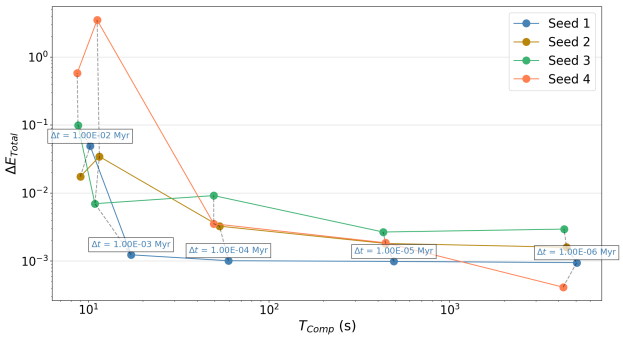


Figure 5.5: Energy error and computation time for the simulation of initializations with seeds 1 to 4 run for 40 steps (0.4 Myr) with different fixed `Bridge` time-step sizes. The time-step sizes are indicated in the figure.

Table 5.2: Training and simulation parameters.

COMMON	
NUMBER OF PLANETS	VARIABLE
MAX STEPS PER EPISODE	40
ΔE TOLERANCE	1×10^0
HIDDEN LAYERS	5
NEURONS PER LAYER	200
BATCH SIZE	125
TEST DATA SIZE	3
NUMBER OF ACTIONS	10
RANGE OF ACTIONS	$[5 \times 10^{-5}, 10^{-2}]$ MYR
$W_{1,2}$	[50, 1]
BRIDGE ε	1.0

First of all, the number of planets depends on the mass of the central star, and will therefore vary depending on the other settings. We set the maximum number of integration steps to 40, which with a check step size of 10^{-2} corresponds to 0.4 Myr.

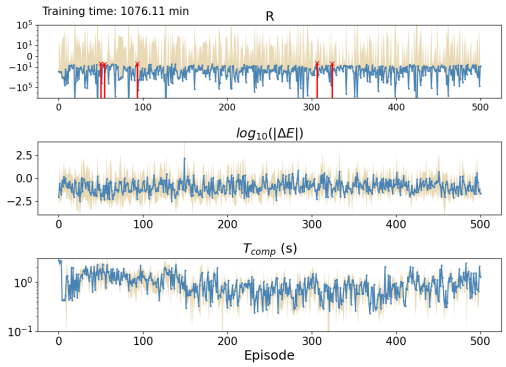
In Table 5.2 we show the network architecture. Due to the chaotic nature of the problem, we had to find a baseline on which to calculate the reward during the training. Therefore, we test the model at each episode on a fixed set of test cases. We use 3 test cases. Ideally, this number should be increased for a better indication of the performance of the model at each episode, but due to computational limitations, we choose to keep this number low. Note that the model will be evaluated on a larger number of test cases after the training is finished.

The actions is a discrete array of length 10 with values of Δt_B that range from 5×10^{-5} to 10^{-2} Myr. Similarly to the implementation in `Hermite` and `Huayno`, we define a time-step parameter ε for the `Bridge` that multiplies the value in the actions. This value is set by default to 1. Finally, the weights for the reward function are shown in Table 5.2 and chosen so that the first term in Equation 5.3 is 50 times larger than the second term.

We start the training with a global search. In this case, we limit the number of stars

to 5 and keep a large value of the learning rate (as seen in the table in Figure 5.6) to keep the computation cost low. We train for 500 episodes and evaluate the performance of the models using the average reward at each episode. In Figure 5.6, we present the results of this global training. The rows in the Figure represent the reward value, energy error, and computation time for each episode. In blue we show the average value obtained from the test cases, and in yellow the standard deviation. We also show the total training time at the top left corner. We mark in red the five episodes with the largest reward and choose the best-performing model from them.

After the global training, the results are still not satisfactory and the reward oscillates. We therefore perform a local search (Figure 5.7) starting from the best performing model (model at episode 50) for 50 episodes with a lower learning rate. Then, we choose the best-performing model; i.e., the one at episode 27.

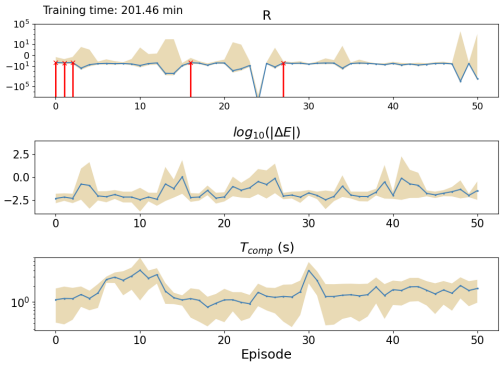


Global search	
Number of stars	5
Max episodes	500
Learning rate	1×10^{-3}
Model chosen	50

Figure 5.6: Evolution of the average (blue) and standard deviation (orange) of different metrics of the test dataset per episode of: the reward value (first row), the energy error (second row), and the computation time (third row) for the global training. With the corresponding training and simulation parameters.

We want to evaluate the performance of this best model. To do that, we run multiple simulations using the trained model and compare the results with those without the RL model. Figure 5.8 shows a schematic representation of the plot we will be using for the comparison of the performance. The runs with different fixed Δt_B ideally form a Pareto front that ranges from the cases with large computation time requirements and low energy errors (right of the plot) to the ones with small computation times and large errors (left of the plot). This Pareto front represents the best accuracy that can be achieved with fixed time-step sizes. Below this curve is the optimum region. Anything below the curve represents a better-performing case. We aim to obtain a method that is located on the Pareto front (to eliminate the expert knowledge) or below (to obtain better-performing methods).

In Figure 5.9, we show the average and standard deviation in computation time and energy error for 10 initializations run for 0.4 Myr. We compare the results of RL model 27 (RL-27) with those with fixed Δt_B. We do that for 5, 9, and 15 stars. The actual energy errors obtained for each of the runs are shown as dots but for simplicity, the computation



Local search 1	
Number of stars	5
Max episodes	50
Learning rate	1×10^{-4}
Model chosen	27

Figure 5.7: Evolution of the average (blue) and standard deviation (orange) of different metrics of the test dataset per episode of: the reward value (first row), the energy error (second row), and the computation time (third row) for the local training performed after the global one. With the corresponding training and simulation parameters.

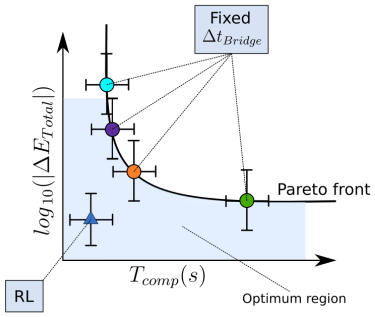


Figure 5.8: Schematic representation of the comparison of fixed Δt_B with the RL method.

time associated is ignored in the plot. An optimum value balances energy error (y-axis) and computation time (x-axis). We observe that the trained model does not present an advantage over the fixed time-step cases.

To solve this performance issue, we further train the network including simulations with variable N and a lower learning rate. The results of the training are shown in Figure 5.10. We choose the model at episode 173 and compare the results in Figure 5.11. We see the improvement in performance achieved by the RL-173 model. For $N = 5$, the model performs similarly to the one with $\mu = 1.6 \times 10^{-4}$, but with a shorter average computation time and smaller spread in energy error. For $N = 9$ our model results in both an improvement in energy error and computation time with respect to the fixed times-step cases. For $N = 15$, the results become harder to interpret as the fixed-step methods do not behave as expected (see Figure 5.8). Nevertheless, the algorithm achieves a better performance than the cases with fixed Δt_B.

We have shown that our model performs as well, or better, than the best-performing

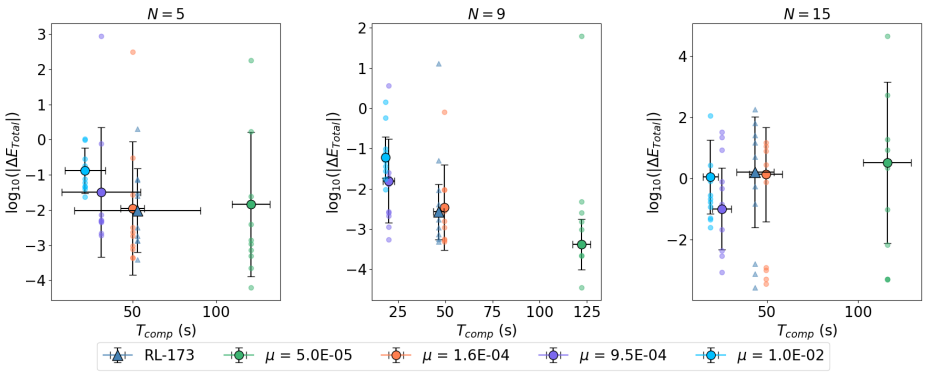
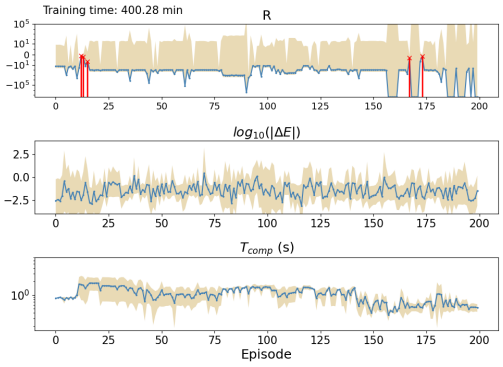


Figure 5.9: Average and standard deviation of the energy error and computation time for 10 different initializations run for 0.4 Myr. The results of the RL-27 model are compared to those of fixed Δt_B. The results of the RL model are unsatisfactory.



Local search 2	
Number of stars	[5-20]
Max episodes	200
Learning rate	1×10^{-4}
Model chosen	173

Figure 5.10: Evolution of the average (blue) and standard deviation (orange) of different metrics of the test dataset per episode of: the reward value (first row), the energy error (second row), and the computation time (third row) for the local training for different bodies. With the corresponding training and simulation parameters.

fixed-size case. When initializing a simulation, it is common to keep the default values used for Δt_B which generally leads to suboptimal results. Our method allows the achievement of optimal performance in terms of computational time and accuracy without the need for any expert knowledge or convergence study.

5.3.3 Integration results

We perform multiple experiments with model RL-173 to better understand its performance and extrapolation capabilities.

We show the individual behavior of the model for initializations with seed 4 (Figure 5.12 (a)) and seed 2 (Figure 5.12 (b)) with different numbers of stars. The top row rep-

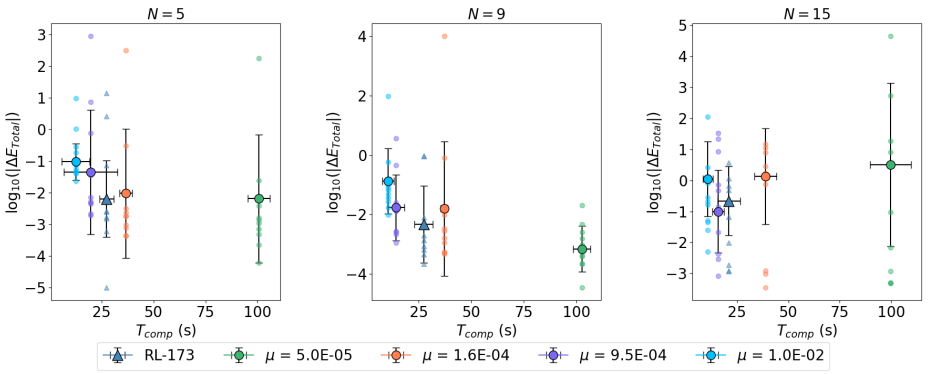


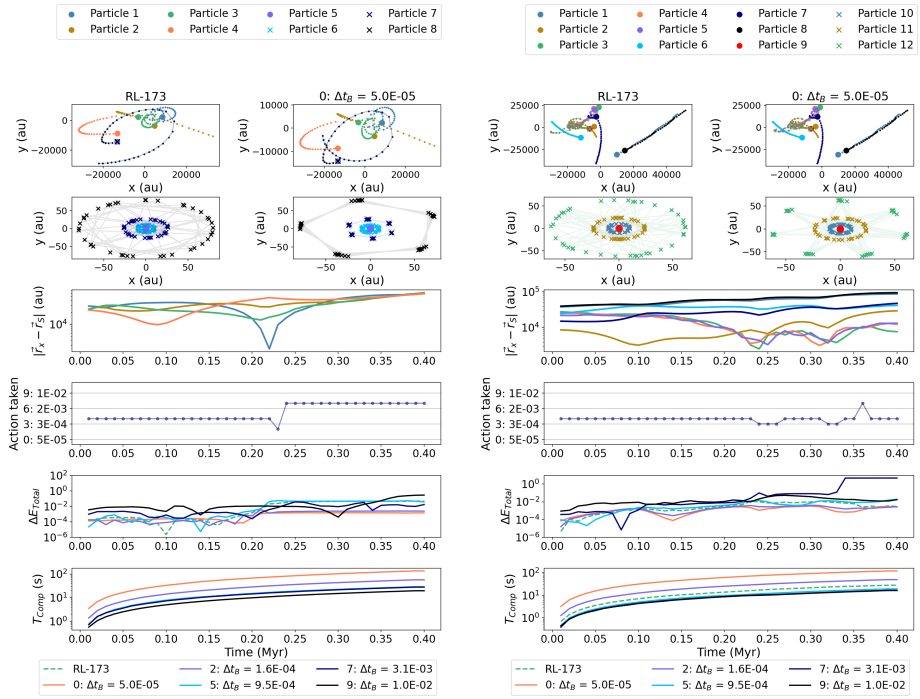
Figure 5.11: Average and standard deviation of the energy error and computation time for 10 different initializations run for 0.4 Myr. The results of the RL-173 model are compared to those of fixed Δt_B.

resents the position of the bodies in the star cluster. The left column shows the evolution using the RL model and the right one the results with the best-performing model with fixed Δt_B. The second row shows the evolution of the planetary system around the central star. The third row is the distance from each star to the one with the planetary system, which contains information about close encounters. The fourth row is the actions taken by the RL model at each step. Finally, the last two rows represent the energy error and computation time of each integration case (RL model and various fixed Δt_B).

We see in Figure 5.12 (a) that there is only one close encounter, and that the RL model recognizes it and chooses a more restrictive action (smaller time-step size). After the close encounter, the stars get further from each other, and the model chooses a less restrictive action, saving computation time. In Figure 5.12 (b), we see a case for 9 stars. We recognize two or three close encounters and see that the RL model adapts accordingly. Finally, it achieves an energy error on the lowest range without incurring large computation times.

In Figure 5.13, we see two other examples. Figure 5.13 (a), shows an example with 15 bodies. In this case, we do not see any close encounters and the algorithm learns to keep a constant action which represents a balance between energy error and computation time. In contrast, in Figure 5.13 (b), the model recognizes close encounters and adapts the otherwise large actions, obtaining an energy error that is lower than most of the other fixed-size cases with a very low computation time.

In Figures 5.12 and 5.13, we observe sudden jumps in the energy error for certain cases. Additionally, the planets may escape their central star or move to more eccentric orbits. To understand these jumps in energy error, we plot in Figure 5.14 the evolution of the distance between each planet and the central star, to better understand these cases. We do this for the same scenarios as in Figures 5.12 (b) and 5.13 (b). Additionally, we present the semi-major axis and eccentricity values for a better understanding of the dynamical evolution of the planetary system. We observe how the jumps in energy error correlate with the distance of a planet to the central star increasing radically. Similarly, it can be observed in these cases that the eccentricity also increases. The most common scenario is the outermost planet (Planet 3) suddenly changing its orbit when a nearby star perturbs it.



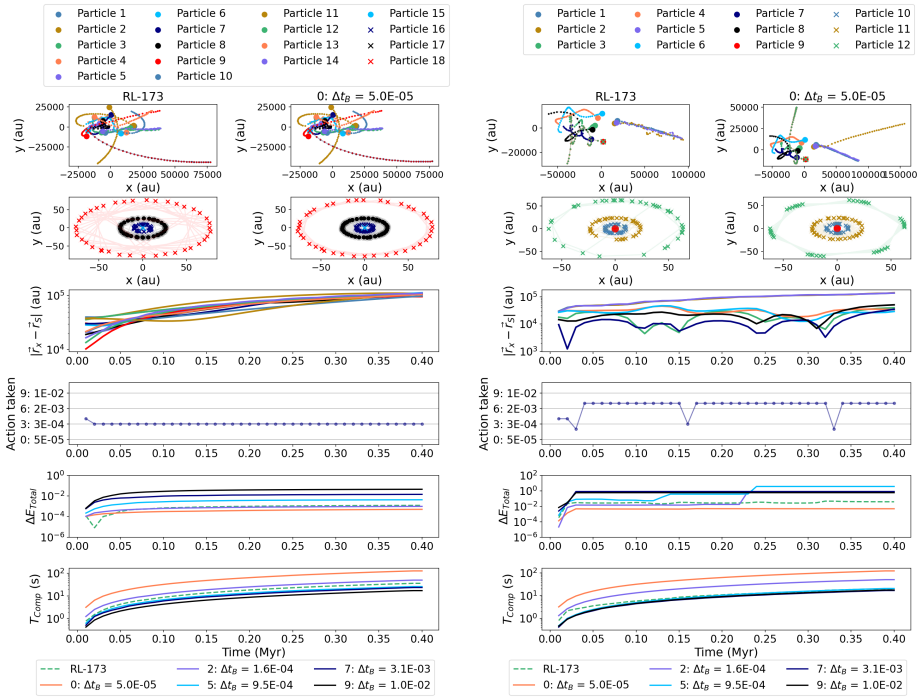
(a) **(b)**

Figure 5.12: Comparison of fixed-size time-step parameters with our RL model for 40 time steps (0.4 Myr). We present the trajectory in Cartesian coordinates of the star cluster (top-row panels) and the planetary system (second-row panels), the distance between each star to the one containing the planetary system (third row), the actions taken by the RL algorithm (fourth row), the energy error at each time step for each study case (fifth row), and the computation time for each study case (last row), for initializations with seed 4 (a) and seed 2 (b).

In our method, we have not considered cases in which one planet becomes unbound or increases the distance to the central star. Once this happens, to keep the energy error bound, the planet should be integrated in the same N-body code as the star cluster, instead of remaining within the planetary system code. More complex bridging algorithms such as `Nemesis` Zwart et al. (2020) include this option in their implementation. We leave this addition to future works.

5.4 Experiments

We have seen that the trained RL model manages to achieve results that are better than those with fixed Δt_B. Those results were obtained with conditions similar to those used to train the model. Therefore, we want to understand its performance when applied to different scenarios. To do that, we perform experiments in which we modify the final integration time, the integrators used, and the time-step parameter.



(a) **(b)**

Figure 5.13: Comparison of fixed-size time-step parameters with our RL model for 40 time steps (0.4 Myr). We present the trajectory in Cartesian coordinates of the star cluster (top-row panels) and the planetary system (second-row panels), the distance between each star to the one containing the planetary system (third row), the actions taken by the RL algorithm (fourth row), the energy error at each time step for each study case (fifth row), and the computation time for each study case (last row), for two initializations with seeds 3 (a) and 4 (b).

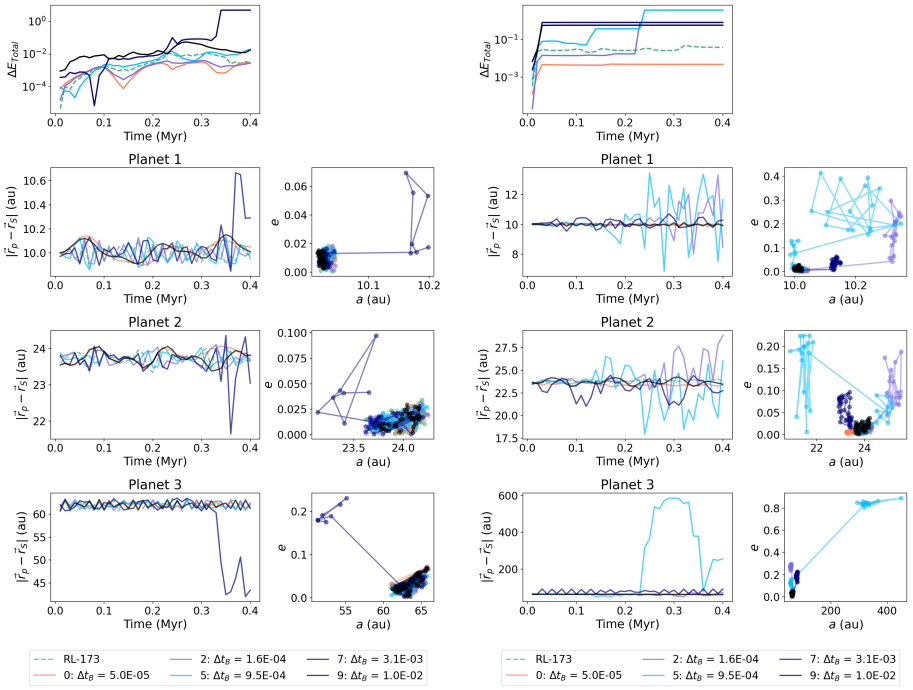
5.4.1 Number of bodies

In Figure 5.9, we show a comparison of the performance of the RL model for 10 different initializations for three cases of N. We use this plot as the baseline with which to compare the other experiments. These systems are chaotic, and the results depend on the initializations. We discuss this further in Section 5.5.

5.4.2 Long term integration

The model is trained on simulations that were run until a final time of 0.4 Myr. We study the performance of the trained model on longer integration times to understand the possible use of our method for long-term simulations.

In Figure 5.15, we show two examples of the simulation with seeds 1 (a) and 2 (b) with different numbers of bodies. We observe here that the model is also able to identify close encounters and chooses more restrictive actions to keep the energy constant and low.

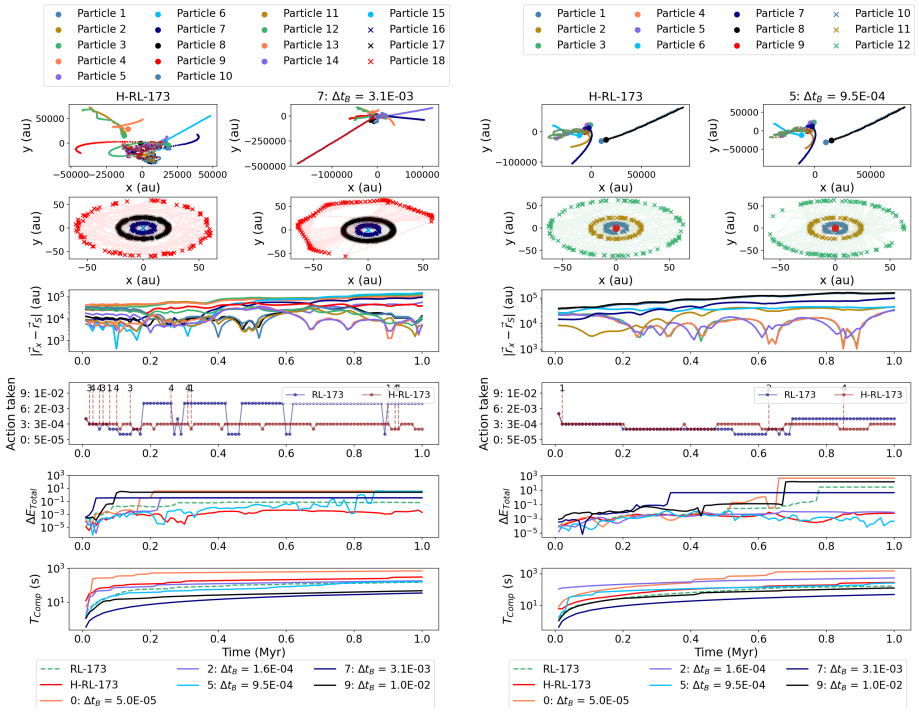


(a) **(b)**

Figure 5.14: Comparison of fixed-size time-step parameters with our RL model for 40 time steps (0.4 Myr). We present the energy error (top row), the time evolution of the distance of each planet to their central star (left panels), and the evolution of the semi-major axis (a) against the eccentricity (e) (right panels) for each planet with seeds 2 (a) and 4 (b).

We can see in Figure 5.15 (a) that the energy error achieved is smaller than with the most accurate fixed-step size while the computation cost remains small. In Figure 5.15 (b), we see a case without pronounced close encounters, but some bodies escape the system at $t \approx 0.7$ Myr due to a sudden increase in the energy error. We observe this behavior in many cases. Making a mistake in the choice of time-step size can lead to sudden jumps in energy error. The RL algorithm can keep the energy error low for longer than other cases shown, but at $t \approx 0.8$ it still experiences a sudden increase.

For long-term integration, it is essential to avoid mistakes in the choice of time-step size. A wrong choice in the action by the RL model can result in a long simulation being inaccurate, and therefore unusable. To prevent this, we propose the implementation of a method that can identify sudden jumps in energy error and adapt the action accordingly to correct for a wrong choice of time step. A similar idea was shown in Ulibarrena et al. (2024). We evaluate the energy error with respect to the previous step. If this relative error is larger than a certain choice value, we repeat the integration step with a time-step size that corresponds to an action lower or, in the case that we were already at the lowest action, reduces the time-step to half its size. This process can be recursively repeated



(a) **(b)**

Figure 5.15: Comparison of fixed-size time-step parameters with our RL model for 100 time steps (1 Myr). We present the trajectory in Cartesian coordinates of the star cluster (top-row panels) and the planetary system (second-row panels), the distance between each star to the one containing the planetary system (third row), the actions taken by the RL algorithm (fourth row), the energy error at each time step for each study case (fifth row), and the computation time for each study case (last row), for two initializations with Seeds 1 and 2.

until the energy error falls within the desired limits. We choose a maximum number of 4 iterations to avoid incurring too large computation time. The error tolerance is defined as

$$\Delta t_B = \frac{\Delta t_B}{2} \qquad \text{for} \qquad \log_{10}(\Delta E_i) - \log_{10}(\Delta E_{i-1}) > 0.3. \tag{5.4}$$

The results obtained with this method, denominated as H-RL (for Hybrid-RL) are included in Figure 5.15. The hybrid implementation manages to prevent sudden jumps in energy error. On the fourth row, we show the actions taken by the H-RL method, and also at which steps the hybrid method was activated (according to Equation 5.4) and the number of iterations it performed to lower the energy error to below the threshold.

The H-RL method results in energy errors orders of magnitude lower than the best result without incurring a much larger additional computation time (Figure 5.15 (a)). For the cases where the RL method experienced a sudden jump in energy error (5.15 (b)), the hybrid method prevents this jump, leading to a final energy error that is on the order of the most accurate results of the fixed time-step cases.

In Figure 5.16, we present a similar analysis for the integration up to 1 Myr, including the RL and the H-RL results. We find that for $N = 9$ and $N = 15$, the RL model results in a similar performance to that of the fixed time-step cases. However, the H-RL further reduces the final energy error at the cost of some computation time. This leads to better-performing results for $N = 9, 15$. For $N = 5$ we find that the RL method is comparable in performance to those with fixed-size time-step. The H-RL model results in a large standard deviation in the energy error for different runs. This is the result of the chaotic behavior in the system, which we discuss further in Section 5.5.

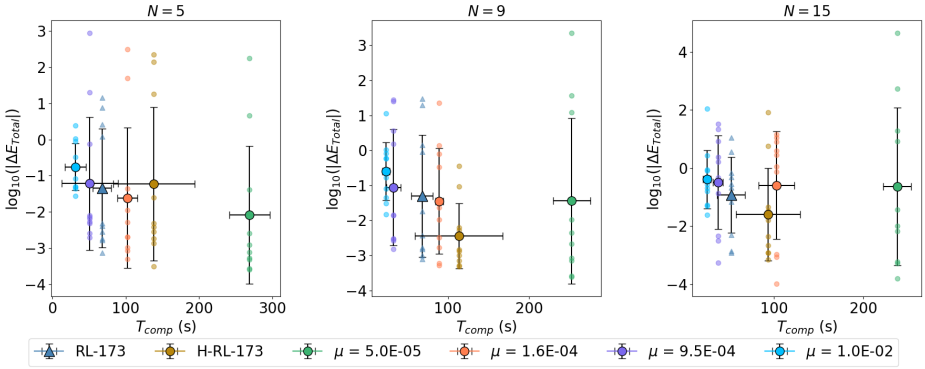


Figure 5.16: Average and standard deviation of the energy error and computation time for 10 different initializations run for 1 Myr. The results of the RL-173 and the H-RL-173 models are compared to those of fixed Δt_B.

This hybrid method is especially relevant for long-term integration as an increase in the energy error is rarely reversible. For the purpose of simplicity, we will not include the hybrid integrator results in the following experiments.

5.4.3 Numerical integrators

In this case, we want to understand whether our trained model is independent of the choice of integrator for the subsystems involved. We therefore replace the Cluster integrator with `Hermite` and the planetary system integrator with `Ph4`.

We can see in this case (Figure 5.17) that the performance of the RL model remains comparable to the baseline case. It achieves better results in terms of a smaller standard deviation than the fixed cases for $N = 5$. For $N = 9$, the average for the RL method is located on the Pareto front, which means that the results perform similarly than the fixed time-step cases. For $N = 15$ we again encounter results that are surprising, such as the most accurate case of fixed time-step resulting in a larger average energy error than some of the larger step-size cases.

5.4.4 Application of a time-step parameter

A time-step parameter (ε) is used in integrators such as `Hermite` and `Huayno` to scale the size of the time steps to allow to make the simulations faster (large ε) or more accurate

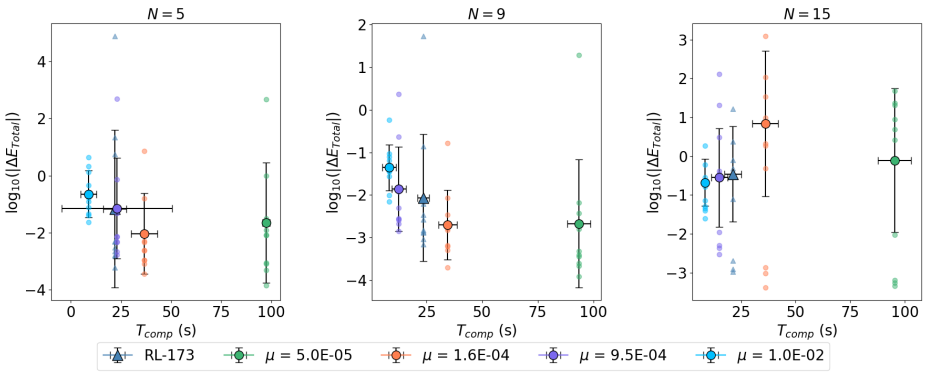


Figure 5.17: Average and standard deviation of the energy error and computation time for 10 different initializations run for 0.4 Myr. The results of the RL-173 model are compared to those of fixed Δt_B. The numerical integrators used in this case are different from those used for training.

(small ε). We implement a similar feature to scale the values of Δt_B. In Figure 5.18, we show that the performance does not decrease by scaling the actions by 10^{-2}. For all cases, the RL method performs better, or similarly to the fixed-size cases. Additionally, it can be seen that for $N = 15$ all cases result in similar average energy errors. This is an indication that the values chosen are unnecessarily small (see Subsection 5.3.1). For $N = 5$ and $N = 9$, the results obtained with the RL model are unequivocally better than those with fixed-step size.

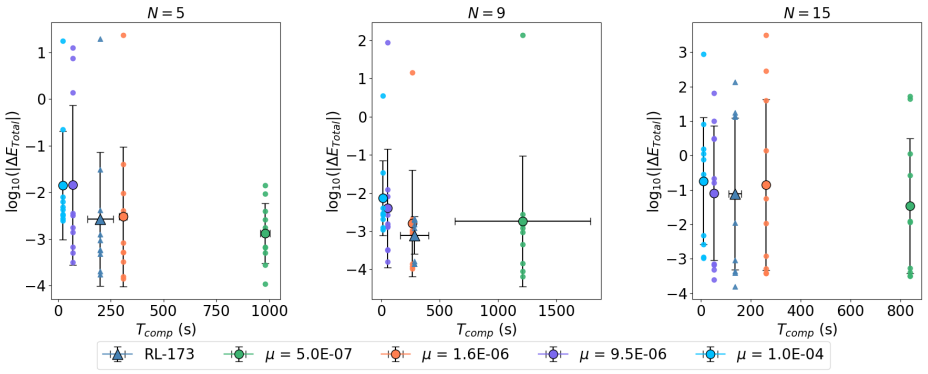


Figure 5.18: Average and standard deviation of the energy error and computation time for 10 different initializations run for 0.4 Myr. The results of the RL-173 model are compared to those of fixed Δt_B. The time-step parameter is changed to 10^{-2}.

5.5 Discussion and conclusions

We have trained a reinforcement learning algorithm to automatically find an optimum value for the coupling integration time-step size of a star cluster. By doing so, we have

eliminated the need for expert knowledge and experimentation needed to set up a simulation with an adequate value of Δt_B, therefore saving time and computational resources. Our method balances energy error (i.e., accuracy) and computation time, and finds results that are better, or in the worst cases similar, to the best-performing cases of fixed time step. Additionally, our method allows to vary the value of Δt_B dynamically to adapt to the needs of the simulation, which is essential in chaotic problems due to their fast-changing conditions.

The performance of our RL method was better than that of all the other options. However, for long integration times, none of the fixed time-step sizes achieved good accuracies. At some point during the integration, due to a close encounter, the time-step size would need to be smaller than what is allowed, which led to steep jumps in the energy error. Although our method managed to keep the energy error constant for longer periods of time than the fixed step-size cases, we want to create a method that is robust for longer periods of time. Therefore, we implemented a hybrid method that identifies sudden changes in energy error and recursively reduces the time-step size at a given step to prevent the increase in energy error. The results are very similar to those of the RL when there was no sudden increase in energy error, but the hybrid implementation prevents those in the necessary cases without a major increase in computation time. In many cases, this hybrid method led to an accuracy that was orders of magnitude better than the best-performing option of fixed time-step size. This method can be used for astronomical simulations to improve their performance and accuracy without adding expert knowledge.

In this work, we have made some assumptions. First of all, we are using energy error as the main measurement of accuracy. A low value of the energy error is an indication of the method adhering to the physical laws but does not ensure that the dynamics of individual bodies are correct. We show a convergence study based on the energy error because convergence based on the dynamics is not possible in a chaotic problem. Any small change in time-step size would lead to a different realization Irani et al. (2024).

We have shown how the performance of our method compares to that of the fixed-step size using the mean and standard deviation of the accuracy and computation time for 10 initializations. Many variations in performance are due to the fact that the problem is largely chaotic. Different initializations result in large ranges of performances even with the same choice of integration settings. It becomes challenging to find a straightforward method to measure the performance of any method. When interpreting these plots, the chaotic nature of the problem needs to be taken into consideration, as many show non-intuitive results. For example, the performance might be worse for low time-step sizes than for larger ones in some cases.

Also due to the chaotic nature of the problem, it is not trivial to find a baseline with which to compare the results. Every change in time-step size (and initial conditions) will lead to dramatic changes in the dynamical evolution of the system. There is no current method to choose or dynamically adapt the time-step size of the `Bridge` method, and therefore we do not have a baseline to compare our new results with. Additionally, we many times compare our results with the most accurate case of fixed-time step size, but this does not always result in a good performance itself. Actually, our method is currently the best-performing solution.

For a low number of bodies, we can compare the `Bridge` method with a direct integrator such as `Ph4`. In Figure 5.19, we plot the accuracy (final energy error) and com-

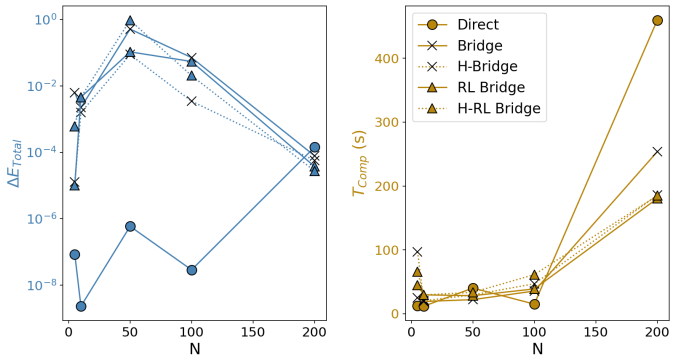


Figure 5.19: Comparison of the total energy error and computation time for an initialization with seed 3 run 40 steps with 9 stars. We compare the results with direct integration, with our `Bridge`, a hybrid implementation of the `Bridge`, and the cases with RL and H-RL.

putation time as a function of the number of stars in the cluster. We can see how direct integration is not suitable for the integration of a large number of bodies as the computation cost increases quadratically with N. The energy error with the `Bridge` method is orders of magnitude larger for a small number of bodies but becomes comparable as N increases. Additionally, we show the results with our RL method and with our hybrid implementation. This comparison is however not completely fair as the direct code has been optimized for speed, whereas the `Bridge` codes in all their variations are simple `Python` implementations. The gap in computation time could be reduced by optimizing the implementation. Similarly, as mentioned in Subsection 5.2.1 Figure 5.2, the `Bridge` methods are simple coupling methods which accuracy could be improved with a more complex implementation.

We have currently performed experiments for $N = 5, 9, 15$ and trained the RL algorithm with clusters with up to 20 stars. Future work should focus on increasing this number in order to allow the use of this method for a larger variety of star clusters. Additionally, we have not performed a hyperparameter optimization, which we believe could also slightly improve the performances shown here.

The general nature of the method allows us to extrapolate to a large variety of systems. The planets could be replaced with a central star surrounded by a protoplanetary disk or a cloud of gas and the method would still be applicable without any changes in the method. Also, the number of bodies can be scaled largely without the need for retraining, although the performance might decrease as the setup diverges from the one used to train the RL method.

We have tested the method for different integrators, for the inclusion of a time-step parameter to adapt the level of accuracy desired, and for long-term integration including our hybrid method. More tests can be performed. However, we show here that the performance remains robust with these changes and only varies slightly.

In short, we have created a method that eliminates the need for the manual choice of the coupling time-step size and changes it dynamically to adapt to chaotic problems. We have built a hybrid method that makes the RL solution robust to mistakes and results

in more accurate simulations for an optimized computation time. This method can be applied to a large variety of astrophysics simulations without major changes to improve their performance.

5.6 Acknowledgments

This publication is funded by the Dutch Research Council (NWO) with project number OCENW.GROOT.2019.044 of the research programme NWO XL. It is part of the project “Unraveling Neural Networks with Structure-Preserving Computing". In addition, part of this publication is funded by the Nederlandse Onderzoekschool Voor Astronomie (NOVA).

Appendix

5.A Summary of experiments

We show a summary of the experiments performed with our RL and H-RL methods. We show the averages and standard deviations of multiple runs for each of the `Bridge` time-step cases.

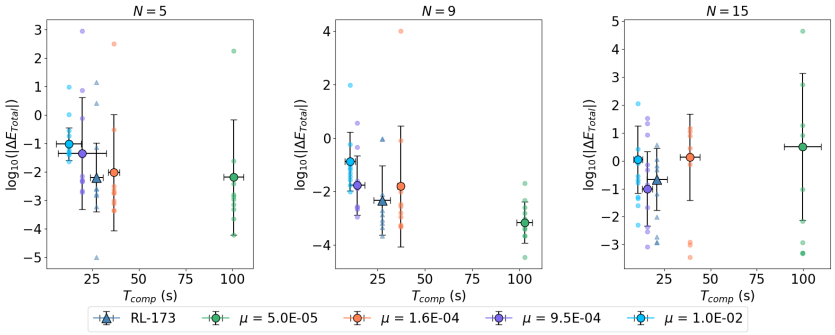


Figure 5.20: Results of the simulations run for 40 steps with the baseline settings.

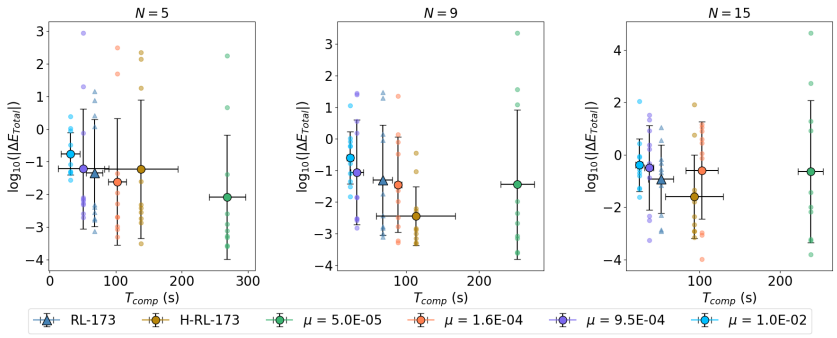


Figure 5.21: Results of the simulations run for 100 steps with the baseline settings. The solutions with the hybrid method are also displayed here.

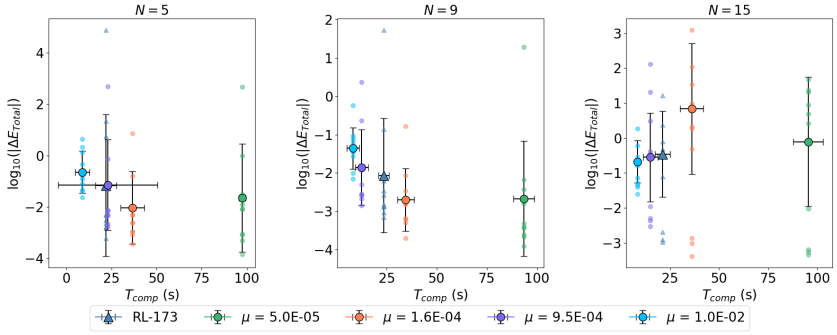


Figure 5.22: Results of the simulations run for 40 steps with a different choice of integrators.

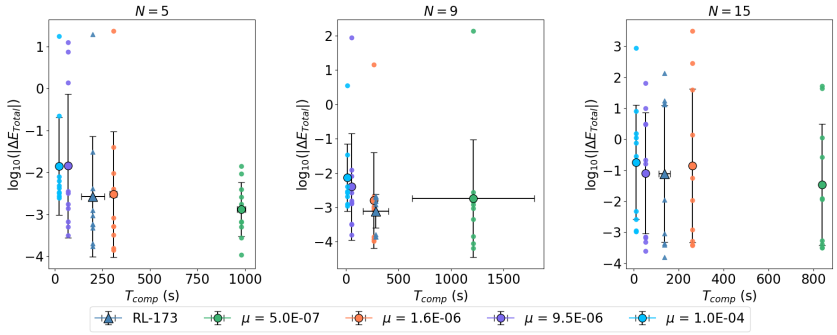


Figure 5.23: Results of the simulations run for 40 steps with a time-step parameter of 10^{-2}.