5 REINFORCEMENT LEARNING FOR THE DETERMINATION OF THE BRIDGE TIME STEP IN CLUSTER DYNAMICS SIMULATIONS

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

ABSTRACT

Astrophysical simulations often deal with multi-scale and multi-physics cases, which entails the need for methods which can handle those. A common approach involves separating the system into multiple parts where each is integrated using problem-specific methods. Then, those parts 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 (small) numbers of bodies and find that the method remains robust for multiple cases and across multiple physical domains. We test the effect of changing the integrators used for the different parts of the system and demonstrate 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 is also applicable in this case. For long-term integration, where energy errors tend to accumulate, 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 jumps in energy error and prevent them by recursively reducing the time-step size at a given instance. The case with the hybrid implementation performs orders of magnitude better compared to our baselines, without significantly increasing the computation time. This method ensures the robustness of

simulations that use reinforcement learning.

Our method eliminates the need for expert knowledge and balances computation time and accuracy while adapting to the needs of the simulation at each step. We show that it can be directly extrapolated to a large range of astrophysical simulations.

5.1 Introduction

As our knowledge of the universe grows, astrophysics simulations become more complex to adapt to increasingly-advanced studies. This complexity can appear in a simulation in multiple forms. For example, by increasing the number of bodies involved or by including multiple physical phenomena, 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 take the computational limitations of the problem studied into consideration.

In a simulation of a system of *N* bodies moving due to their mutual gravitation, numerical errors appear mainly due to discretization approximations and round-off (Boekholt & Portegies Zwart (2015b)). These errors can accumulate and lead to a deviation of the results with respect to the physics. The accuracy of a simulation can therefore be measured in terms of the numerical error present. Newton's equation of motion can be integrated accurately with 4th-order direct integration methods. However, the computational effort of direct methods scales with N^2, with N being the number of bodies in the system. This can lead to large computation times being required to obtain accurate solutions. For example, simulating a star cluster with $N = 100$[1] using `Brutus` integrator can take 43 days when run on an 8-core modern workstation. To allow for faster integration of large systems, several methods have been designed to speed up this computation at the cost of accuracy. For example, the hierarchical tree algorithm from Barnes-Hut (Barnes & Hut (1986)) groups far-away bodies, whereas nearby interactions 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 part to speed up the calculation while missing the subtleties of the smallest one, or to integrate the system using a time-step size consistent with the scale of the smallest part, leading to impractically-large computation times. This problem can be effectively addressed by using methods that separate these systems into smaller parts and integrate them independently. Examples of these methods are `Lonely planets`, `Nemesis` (Portegies 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, the different parts of the system need to communicate with each other and ensure that each part includes the effect of its environment. This coupling is dealt with differently in each method. A common challenge is finding the time scale at which those

[1] Initialized using a Plummer sphere, equal-mass stars, and run for 20 *N*-body time units.

systems need to interact, i.e., communicate with each other, to ensure that the simulation is accurate enough for the purpose of the study while keeping the computational cost low. The choice of this coupling time-step size is normally made manually based on expert knowledge and kept fixed throughout the simulation. As explained in Saz Ulibarrena & Portegies Zwart (2024), systems that experience dynamical changes in their behavior benefit from a variable time-step size that adapts to the relevant scales of the system at a given time (Heggie & Hut (2003); Aarseth (2003)). Indeed, many integration codes 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 avoiding expensive calculations (Cai et al. (2021b); Greydanus et al. (2019a); Saz Ulibarrena et al. (2024)). Additionally, an interesting application is using ML methods to replace choices that are otherwise made manually. When setting up a simulation, there are certain 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. 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 an 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 times, and the number of stars in the cluster. We then simulate a system of 1,000 stars with a different star cluster integrator to further demonstrate the generalization capabilities of our method. Lastly, in Section 5.6 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`.

5.2 Methodology

We couple different parts of a multi-scale system and incorporate 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 of the system into multiple parts.

We take as our integration framework `Bridge` as defined in `AMUSE` (Portegies Zwart & McMillan (2018); Portegies Zwart et al. (2009); Portegies Zwart et al. (2013)). It is a method that separates a system into a parent and a child. Each of them can then be integrated using a different code and integration settings. This allows the use of the most adequate methods in cases where there are fundamental differences in the behavior of the parts that form a system. We apply this method to a multi-scale problem, but it can also be used for cases 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. A more detailed description of `Bridge` can be found in Fujii et al. (2007); Portegies Zwart et al. (2013).

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

Although `Bridge` can be useful for multiple applications, in many cases we find that one particle should be included in the integration of both parts. This application is not implemented in the classic `Bridge` method. To solve that, we develop an `inclusive Bridge`, or `iBridge`, in which one particle is common to both the parent and child. In this case, in addition to exchanging information about the potential between parts, we use `Bridge` to update the state of the common particle. In Figure 5.1 we see the `iBridge` method for a cluster of stars in which one star has a planetary system. We divide the system into a parent and a child: the star cluster and the star with 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 the 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. 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 a final time or maximum number of steps is reached.

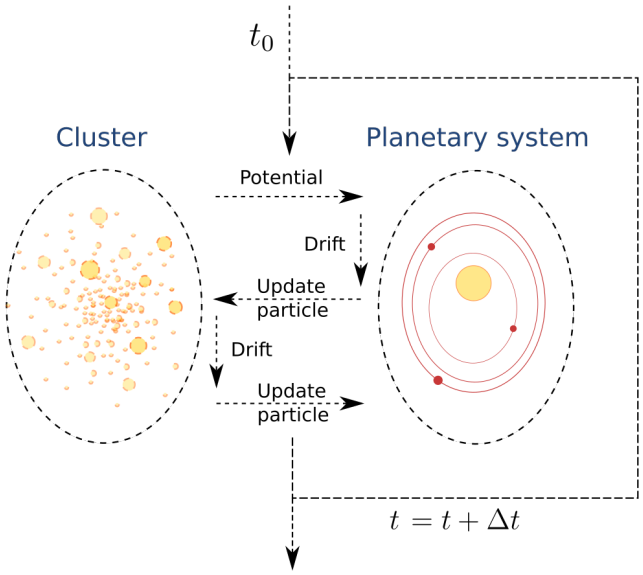


Figure 5.1: Schematic of the `inclusive Bridge` method as developed for this application. The system is divided into two parts: a star cluster and a planetary system, with one star being common for both parts. The acceleration caused by the cluster on the planetary system is calculated and used to update the velocities of the bodies in the planetary system. Then the planets and central star are evolved and the state of the central star in the cluster is updated with that of the planetary system. Afterwards, the cluster 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 at every step.

In Figure 5.2, we show a comparison of `iBridge` against `Bridge` as it is implemented in `AMUSE`, and direct integration, i.e., integrating the entire system with a dedicated numerical integrator. `iBridge` outperforms the regular `Bridge` in accuracy by orders of magnitude. Nevertheless, both generate energy errors orders of magnitude larger than direct integration. The error in both `Bridge` methods compared to direct integration partly results from them being 2nd-order coupling strategies against a 4th-order direct integrator. Additionally, the `Bridge` methods assume that the system is perfectly separable at all times. We will discuss the advantages and shortcomings of using this method compared to some more complex ones in Section 5.6. The error difference is also partially derived from the `Bridge` time step not adapting to the needs of the problem, whereas the direct integrator includes an adaptable time-stepping strategy. We aim to implement a similar strategy for `iBridge` using RL.

The advantage of `Bridge` and `iBridge` with respect to direct integration is the possibility of using several integrators independently for each part. `Bridge` is a strategy aimed at reducing the computation time in systems with different scales or multiple physical processes. This advantage is mostly relevant in systems with large N. As a consequence, in Figure 5.2 it is not possible to see an improvement of the computation time for the `Bridge` methods as the number of bodies is not large enough. This results in better computation times with direct integration than with both `Bridge` implementations (see Portegies Zwart et al. (2020)). In Figure 5.20 from Section 5.6, we perform a more detailed study of the performance of `iBridge` compared to direct integration for different numbers of bodies. Although the goal of using approximate methods is to simulate systems with large N, for the purpose of this work, we limit ourselves to N between 5 and 15 to speed up the calculations. Additionally, this comparison is not fair as the code for the `Bridge` methods has not been optimized for speed (they are implemented in `Python`), in contrast to the code for the direct integrator.

5.2.2 Experimental 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 the parent and the child, and a planetary system is initialized around it assuming an oligarchical planetary growth (Tremaine (2015); Kokubo & Ida (2002)).

The child and parent are evolved with different integrators. `Hermite` (Makino & Aarseth (1992)), `Ph4`, and `Brutus` (Boekholt & Portegies Zwart (2015b)) are integrators that are commonly used in these cases. For the integration of the equations of motion of the stars in the cluster we use `Ph4` as it is implemented in `AMUSE` (Portegies Zwart & McMillan (2018)). This is a direct integration code with internal calculation of the time step for each particle. For the planetary system, we use `Huayno` (Jänes et al. (2014)), an integrator derived from 2nd order Hamiltonian splitting for N-body dynamics, which makes it well suited for the integration of planetary systems. Each code is initialized with a time-step parameter (η) as indicated in Table 5.1, which scales the internally-calculated

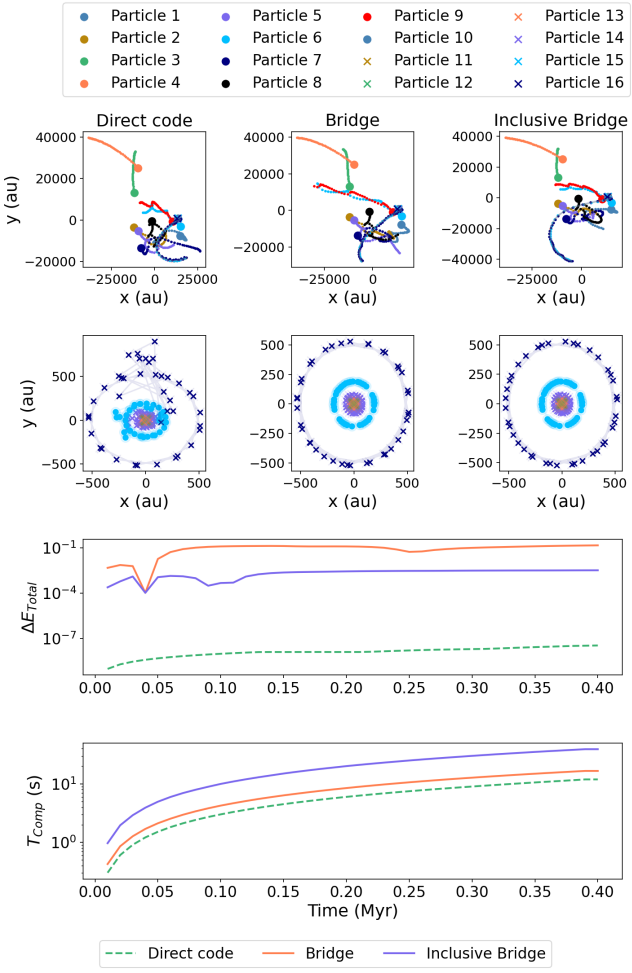


Figure 5.2: Comparison of our `(inclusive) Bridge` against the `Bridge` method as implemented in `AMUSE` and direct integration.

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
FRACTAL DIMENSION	1.6

INTEGRATION	
CLUSTER CODE	PH4
CLUSTER CODE η_C	10^{-2}
PLANETARY SYSTEM CODE	HUAYNO
PLANETARY SYSTEM CODE η_P	10^{-2}
CHECK STEP SIZE	10^{-2} Myr

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

time-step sizes. The state of the system is saved and the bridge time step re-evaluated with a frequency determined by the check step size.

There are four main time steps involved in this setup:

- **Time-step size of the cluster integration**: 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 (η_C).

- **Time-step size of the planetary system integration**: 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 (η_P) which might be different from η_C used for the cluster.

- **Bridge time step** Δt_B: this is the time scale on which the parent and child exchange information. This means, how often the method calculates the acceleration caused on one system by the other and updates the common particle. This time-step size has to be selected at the start of the simulation. There is currently no implementation that automatically selects an optimum value for Δt_B; this selection is done based on expert knowledge. This is the time step that we will determine using RL.

- **Check step size**: time scale used to save the state of the system and apply the reinforcement learning method. This means, after how much time the choice of the bridge time step is re-evaluated.

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 bodies 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 experiences large differences in its evolution depending on the initial conditions and time step used for its integration. As a result, the optimal bridge time-step size is different depending on the initial realization. Additionally, since the conditions change quickly in time, the optimal time-step size will also vary during the simulation.

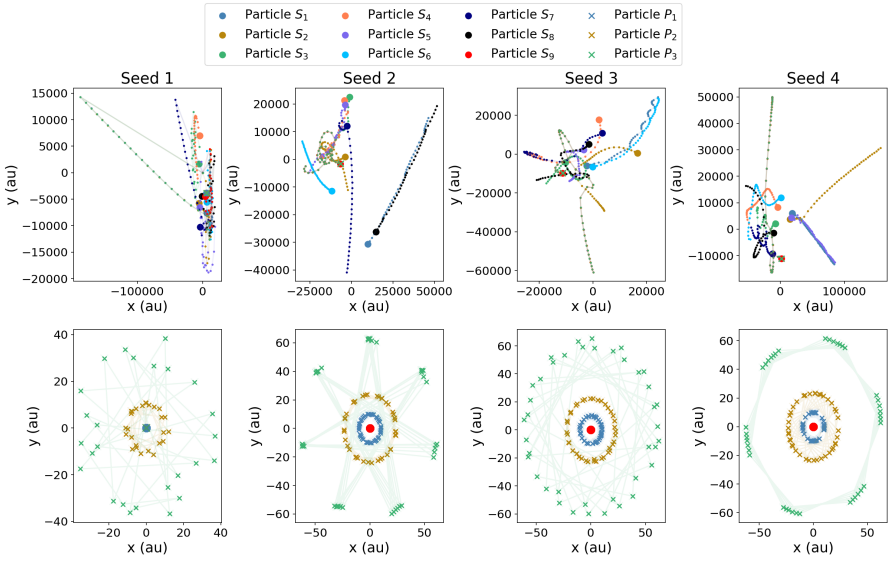


Figure 5.3: Initializations for Seeds 1 to 4 run for 40 steps (0.4 Myr) with a `Bridge` time-step of 5×10^{-5} Myr. The setup is formed by 9 stars and 3 planets.

5.2.3 Energy error

This system is fundamentally chaotic. As a consequence, small changes in the simulation may lead to large differences in the evolution of the system. In many experiments, it is common to use an accurate simulation as a baseline to compare new methods based on the state of the system at different moments. However, for the long-term evolution of chaotic systems, this becomes unfeasible; the dynamical evolution will unavoidably diverge from the baseline. In these cases, 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 5.6.

The total energy of the system is not perfectly conserved during the simulation. This means that the use of numerical integrators leads to energy errors. The total energy is the sum of the kinetic and potential energy of the system. We define the energy error as the relative difference of the energy at time step i and the initial time-step,

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

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, i.e., the validity of our simulation.

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 $\boldsymbol{R}$

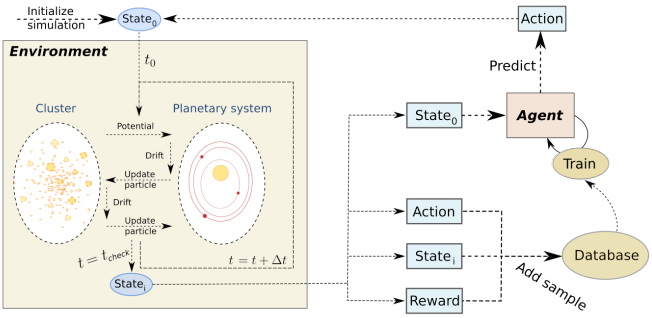


Figure 5.4: Schematic of the interaction between the Environment and the Agent.

(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 using random seeds during the training.

- **Agent**: the agent is the reinforcement learning algorithm that is trained to select the actions. It is composed of 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 ($\boldsymbol{S}$) 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 is evaluated with values from the environment and is used as the loss function to train the network.

  To set up the agent, we have to select 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 environment 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 ($\boldsymbol{S}$) 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 costs. A more detailed discussion on this remark can be found in Saz Ulibarrena & Portegies Zwart (2024).

- **Actions**: the actions ($\boldsymbol{A}$) are the possible values of a decision variable. The reinforcement learning algorithm is trained to select 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 simulation. The number of actions selected 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 at hand, we want to optimize accuracy and computation time. Therefore, we design 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. $W_{1,2}$ are the weights used to balance these two terms. They are a design choice and the values used 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.

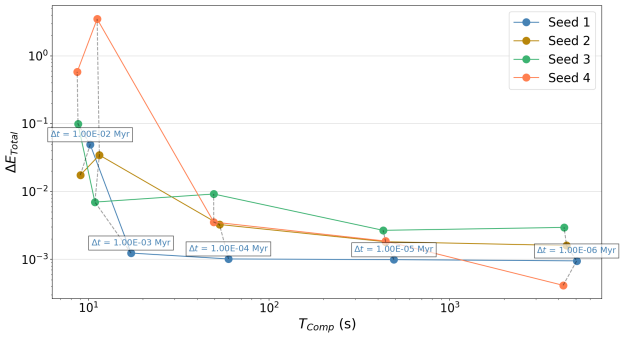


Figure 5.5: Energy error and computation time for the simulation of initializations with seeds 1 to 4 run for 40 steps (0.4 Myr) with different fixed `iBridge` time-step sizes. The time-step sizes are indicated in the figure.

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. Also, this problem 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) may lead to radically different outcomes.

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 initialized with seeds 1 to 4 (see Figure 5.3) for 40 steps using different values of Δt_B. We perform a convergence study to find the value of Δt_B for which the energy error does not improve further if being reduced but instead results in larger computation times. This definition of convergence is different from that used for example in Boekholt & Portegies Zwart (2015b) [2]. There, a converged solution is achieved with arbitrary precision codes such as `Brutus`.

We show the results in Figure 5.5. The value of Δt_B for which the simulation converges depends on the initial realization. Some cases such as the one with seed 4 result in the minimum value of Δt_B 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. We can see that depending on the simulation a time step size of 10^{-2} Myr may yield large energy errors. We choose

[2] From Boekholt et al.: "A converged solution is a solution for which the first specified number of decimal places of every phase-space coordinate in our final configuration in the N-body experiment becomes independent of the length of the mantissa and the Bulirsch-Stoer tolerance".

Table 5.2: Training and simulation parameters.

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]
IBRIDGE η_B	1.0

this value as the upper limit for the RL actions.

5.3.2 Training results

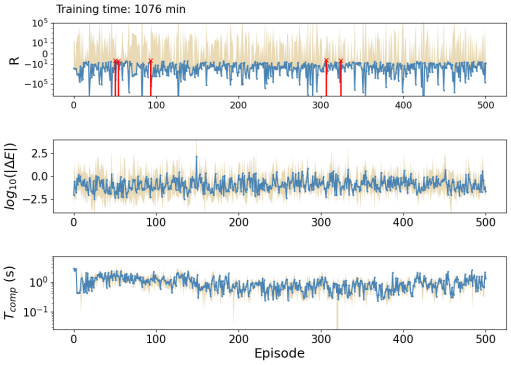
We have defined the environment in Subsection 5.2.2 and the RL method in Subsection 5.2.4. With that information and the settings in Table 5.2, we obtain the trained models. We set the maximum number of integration steps to 40, which with a check step size of 10^{-2} Myr corresponds to a final time of 0.4 Myr. The number of planets depends on the mass of the central star, and will therefore vary depending on the other settings.

In Table 5.2 we show the network architecture. Due to the large differences between initial realizations, 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 (Saz Ulibarrena & Portegies Zwart (2024)). 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 small. Note that the model will be evaluated on a larger number of test cases after the training is completed. 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 η_B for the `iBridge` 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 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 orange the standard deviation. We also show the total training time in the top left corner. We mark in red the five episodes with the largest reward and choose the best-performing model among those.

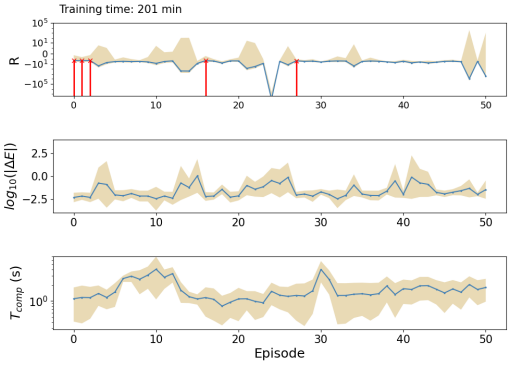
After the global training, the results are still not satisfactory and the average of the reward oscillates. We therefore perform a local search (Figure 5.7) starting from the

best performing model (model at episode 50 in Figure 5.6) 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. The top five performing models are shown in the top row in red. A table is shown with the corresponding training and simulation parameters.



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. The top five performing models are shown in the top row in red. A table is shown with the corresponding training and simulation parameters.

We evaluate the performance of the chosen model. To do that, we run multiple simulations using the trained model and compare the results with those without RL. Figure 5.8 shows a schematic representation of the plot that will be used for the statistical 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 small 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 performance that can be achieved with fixed time-step sizes. Results below the curve represent better-performing cases compared to the fixed Δt_B cases. We aim to obtain a method that is located on the Pareto front (to eliminate the expert knowledge) or below (to also obtain better-performing methods).

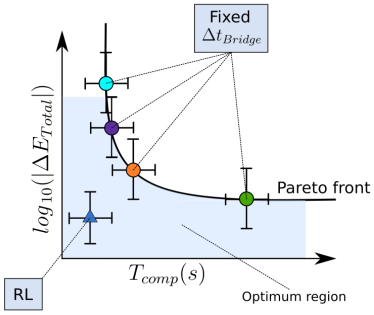


Figure 5.8: Schematic representation of the comparison of fixed Δt_B with the RL method.

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 the RL model at episode 27 (RL-27) to 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 points but for simplicity, the computation time associated is ignored in the plot. An optimum value balances energy error (y-axis) and computation time (x-axis). We show the Pareto front as a line joining the mean value of the fixed time-step cases. We observe that the trained model does not present an advantage over the fixed time-step cases for $N = 5, 15$.

Some samples in the plots are filled, whereas others are not. The unfilled samples represent those cases in which the planets escaped the planetary system. In that situation, `Bridge` methods are no longer valid. An in-depth discussion is presented in Subsection 5.3.3.

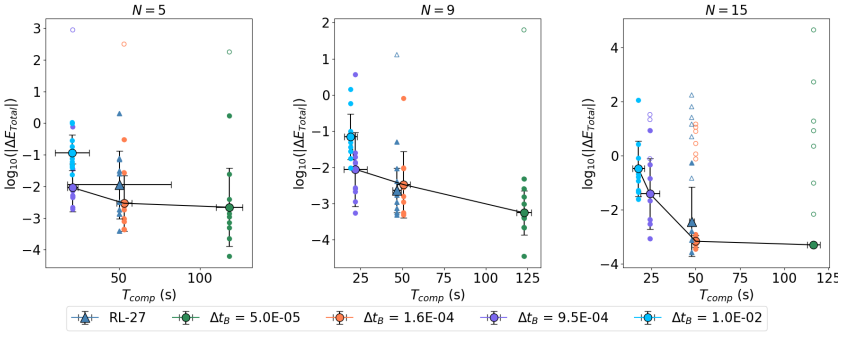
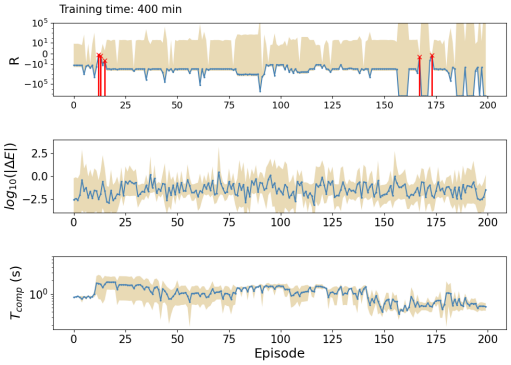


Figure 5.9: Average and standard deviation of the energy error and computation time for 10 different initializations run for 0.4 Myr. The results of the RL-27 model are compared to those of fixed Δt_B. The results of the RL model are unsatisfactory.

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 $N = 5, 9$, 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 for some cases with fixed Δt_B, most samples involve escaped planets (represented by unfilled markers). Nevertheless, the algorithm achieves a comparable performance to the cases with fixed Δt_B and results in fewer cases with escaped planets.



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. The top five performing models are shown in the top row in red. A table is shown with the corresponding training and simulation parameters.

We have shown that our model performs as well, or better, than the best-performing fixed Δt_B case. When initializing a simulation, it is common to use the default values 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 expert knowledge or a convergence study.

5.3.3 Integration results

To better understand the performance and extrapolation capabilities of model RL-173, we perform multiple experiments.

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 represents 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 presents 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

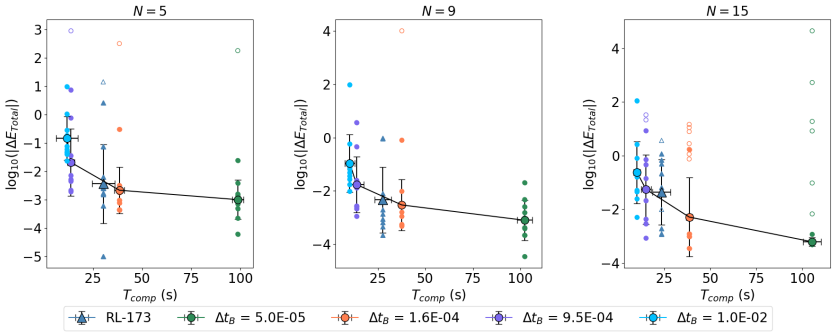


Figure 5.11: Average and standard deviation of the energy error and computation time for 10 different initializations run for 0.4 Myr. The results of the RL-173 model are compared to those of fixed Δt_B.

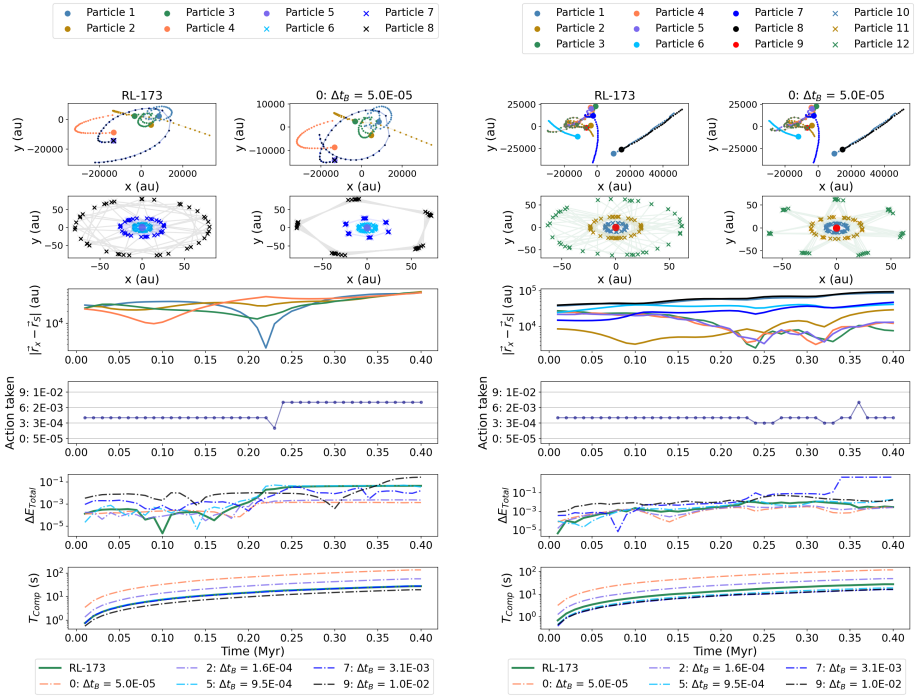
taken by the RL model at each step. Finally, the last two rows represent the energy error and computation time of each integration case (RL model and several fixed Δt_B).

We see in Figure 5.12 (a) that there is only one close encounter, and that the RL model recognizes it and selects a more restrictive action (smaller time step). After the close encounter, the stars move further away 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 compared to the fixed Δt_B cases without incurring large computation times.

In Figure 5.13, we see two other examples. Figure 5.13 (a), shows an example with 15 bodies without any close encounters. In this case, 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 smaller than most of the other fixed Δt_B 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. To understand these jumps, we plot in Figure 5.14 the evolution of the distance between each planet and the central star. 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 correspond to the distance of a planet to the central star increasing radically. Similarly, it can be observed in these cases that the eccentricity of the orbit of the planet also increases. The most common scenario is the outermost planet (Planet 3) suddenly changing its orbit when a nearby star perturbs it.

We show in Figure 5.14 that the jumps in energy error correspond to the planets moving further away from their central star. It is unclear whether the jump in energy error is caused by the change in the orbit of the planets or vice-versa. We observe that the use of reinforcement learning helps to prevent the planets from escaping (see unfilled markers in Figure 5.11). This may indicate that the planets escaping are a consequence of a large energy error during close encounters. An interesting discussion can be derived from this

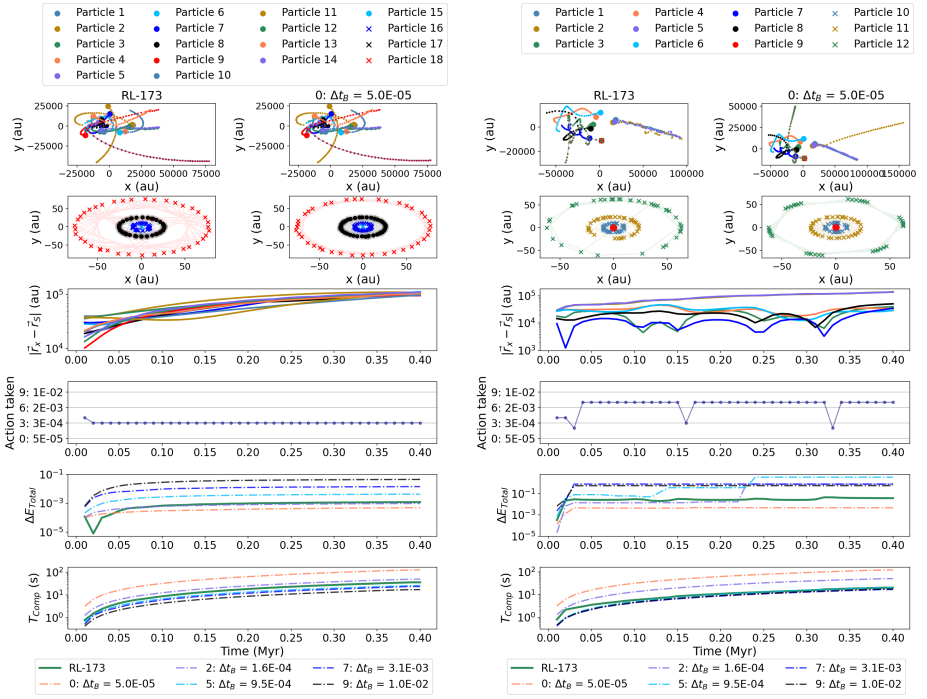


(a) **(b)**

Figure 5.12: Comparison of fixed Δt_B to 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).

observation. The `Bridge` methods assume that the system can be separated into different parts. However, this assumption may not hold true at certain moments of the simulation. When one planet becomes unbound or increases the distance to the central star, to keep the energy error bound, the planet should be integrated as part of the same N-body code as the star cluster. Also, if a star from the cluster moves close to the planetary system, those should be integrated together and `Bridge` becomes no longer valid. More complex coupling algorithms such as `Nemesis` (Portegies Zwart et al. (2020)) include this option in their implementation. We leave this addition to future works.

This situation can also be mitigated when using `Bridge` methods by further reducing the bridge time step to adapt to the needs of the problem. This may lead to impractically small time-step sizes and large computation times. In our method, we set lower and upper limits for the values of the bridge time step (i.e., for the actions) to avoid extreme values of Δt_B.



(a) **(b)**

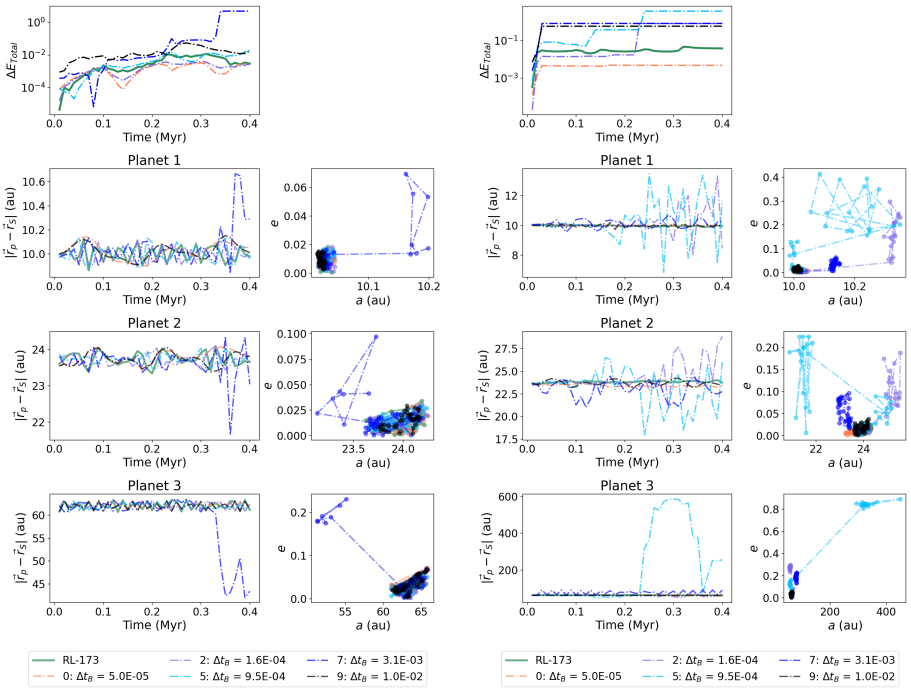
Figure 5.13: Comparison of fixed Δt_B to our RL model for 40 time steps (0.4 Myr). We present the trajectory in Cartesian coordinates of the star cluster (top-row panels) and the planetary system (second-row panels), the distance between each star to the one containing the planetary system (third row), 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 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 carry out experiments in which we modify the final integration time, the integrators used, and the bridge time-step parameter.

5.4.1 Number of bodies

In Figure 5.9, we showed a comparison of the performance of the RL model for 10 different initializations for three cases of N. We use this plot as a baseline with which to compare the other experiments.



(a) **(b)**

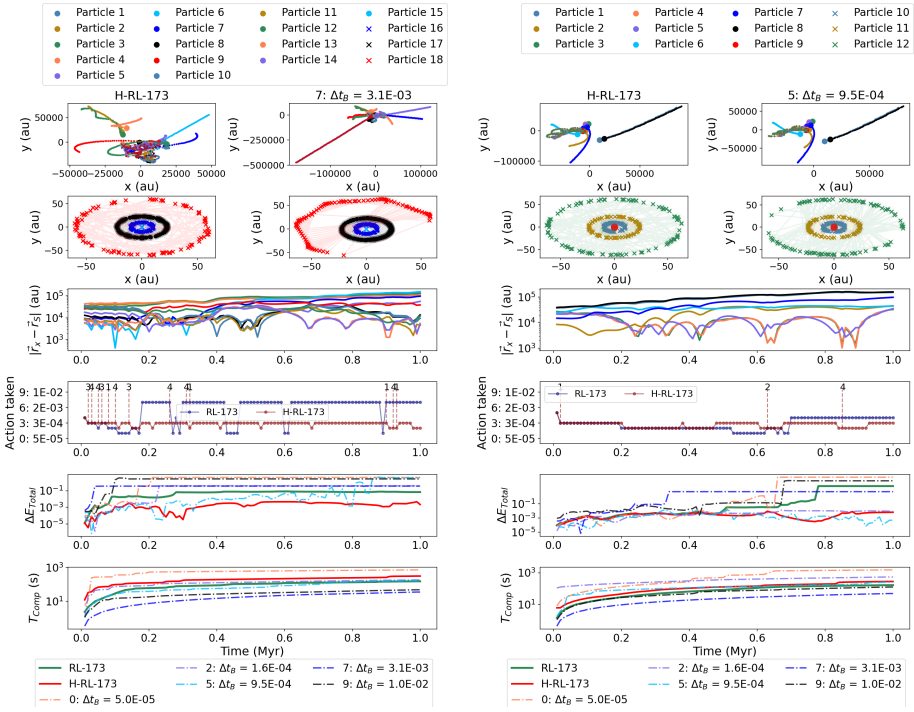
Figure 5.14: Comparison of fixed Δt_B to our RL model for 40 time steps (0.4 Myr). We present the energy error (top row), the time evolution of the distance of each planet to their central star (left panels), and the evolution of the semi-major axis (a) against the eccentricity (e) (right panels) for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

5.4.2 Long term integration

The model is trained on simulations that were run for 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 still able to identify close encounters and chooses more restrictive actions to keep the energy constant and small. We can see in Figure 5.15 (a) that the energy error achieved is systematically smaller than with the most accurate fixed Δt_B case while the computation cost remains small. In Figure 5.15 (b), we see a case without pronounced close encounters, but where some cases experience a jump in energy error at $t \approx 0.7$ Myr. Making a mistake in the choice of time-step size can lead to sudden changes in energy error. The RL algorithm can keep the energy error small for longer than other cases shown, but at $t \approx 0.8$ it still experiences a jump.

For long-term integration, it is essential to avoid mistakes in the choice of the time step. A wrong choice in the action by the RL model can result in a long simulation being



(a) **(b)**

Figure 5.15: Comparison of fixed Δt_B to our RL and H-RL models 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.

inaccurate and therefore unusable. To prevent this, we propose a method that can identify jumps in energy error and adapt the action accordingly to correct for a wrong choice of time step. A similar idea was shown in Saz Ulibarrena et al. (2024). We evaluate the energy error with respect to the previous step. If this relative error is larger than a predetermined value, we repeat the integration step with a time-step size that corresponds to a smaller action or, if we were already at the smaller action, reduce the time-step to half its size. This process can be repeated 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 jumps in energy error. In 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 below the threshold. The H-RL method results in energy errors orders of magnitude smaller than the best result without incurring much additional computation time (Figure 5.15 (a)). For the cases where the RL method experienced a 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 the statistical analysis for the integration up to 1 Myr, including the RL and the H-RL results. We find that for $N = 5, 9$, and 15, the RL model results in a similar performance to that of the fixed time-step cases. The mean value appears over the Pareto front as the use of RL helps to prevent the cases in which planets escape. In most cases of fixed Δt_B, the values with a larger energy error have been discarded as they involve escaping planets. This results in the mean energy error displayed in the plots being smaller than with RL. The H-RL case further reduces the final energy error at the cost of some computation time and leads to unequivocally better performance for $N = 9$. For $N = 5, 15$, the results with H-RL are comparable to those with fixed time-step size but with a larger standard deviation in computation time. This represents an improvement in computation time with respect to the fixed-step cases for similar values of the energy error.

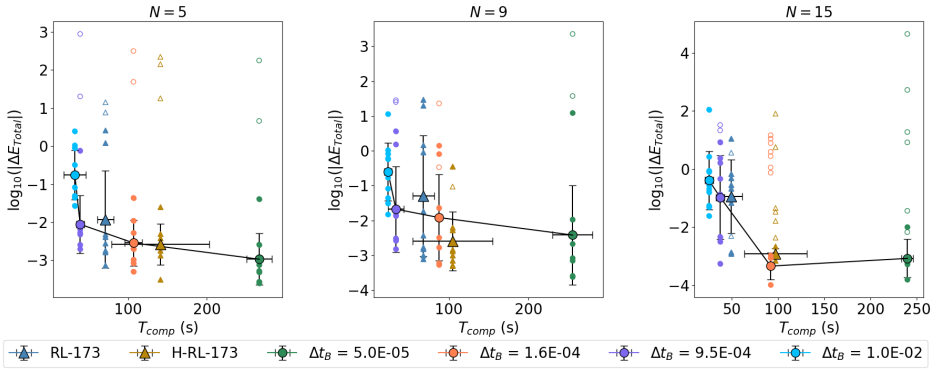


Figure 5.16: Average and standard deviation of the energy error and computation time for 10 different initializations run for 1 Myr. The results of the RL-173 and the H-RL-173 models are compared to those of fixed Δt_B.

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 Application of a time-step parameter

A time-step parameter (η) 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 η) or more accurate (small η). We implement a similar feature to scale the values of Δt_B. In Figure 5.17, 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. We observe here that scaling Δt_B also results in fewer samples with escaping planets.

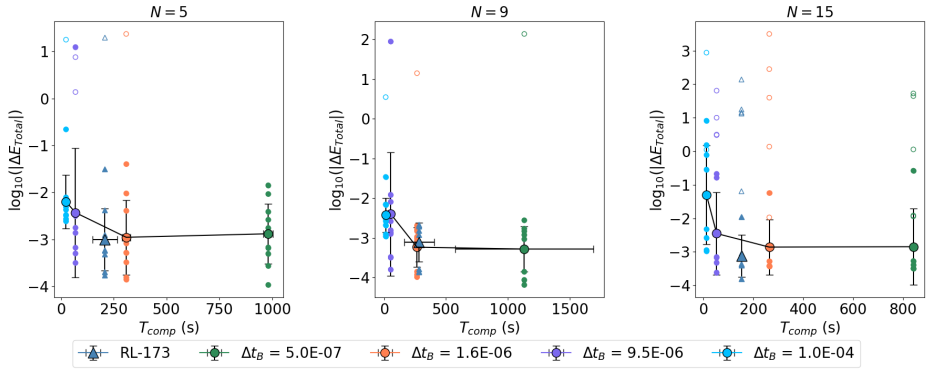


Figure 5.17: Average and standard deviation of the energy error and computation time for 10 different initializations run for 0.4 Myr. The results of the RL-173 model are compared to those of fixed Δt_B. The time-step parameter is changed to 10^{-2}.

5.4.4 Numerical integrators

One of the main limitations of reinforcement learning methods is their difficulty extrapolating to different setups. Therefore, we want to understand whether our trained model is independent of the choice of integrators for the parent and child. We replace the cluster integrator with `Hermite` and the planetary system integrator with `Ph4`.

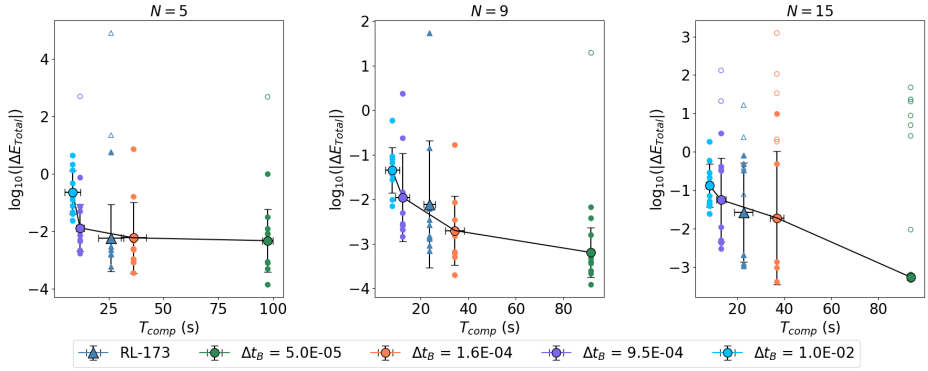


Figure 5.18: Average and standard deviation of the energy error and computation time for 10 different initializations run for 0.4 Myr. The results of the RL-173 model are compared to those of fixed Δt_B. The numerical integrators used in this case are different from those used for training.

We can see in Figure 5.18 how the performance of the RL model remains comparable to the baseline case. It achieves better results than the fixed cases for $N = 5, 15$. For $N = 9$, the average for the RL method is located approximately on the Pareto front,

which means that the results perform similarly to the fixed time-step cases. For $N = 15$ we again encounter a large number of samples with escaping planets, which were not included in the calculation of the mean and standard deviations.

5.5 Star cluster simulation using Tree codes

We have tested our RL and H-RL methods on simulations using direct integrators for the parent and the child and a small *N*. To further test the general applicability of the RL-`iBridge` methods, we simulate a system with $N = 1,000$ and a Barnes-Hut Tree integrator (Barnes & Hut (1986)) for the integration of the star cluster. We integrate the system for 1 Myr and initial seed 2.

We find that RL-173 is capable of identifying close encounters and adapting the actions accordingly. The H-RL method further reduces the energy error at the cost of computation time. Both methods achieve results that balance accuracy and computation time without the need for expert knowledge.

5.6 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 planetary system in a star cluster. In 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 Δt_B. Additionally, our method automatically varies the value of Δt_B dynamically to adapt to the needs of the simulation, which is essential in problems with fast-changing conditions.

The performance of our RL method is better than with any of the fixed Δt_B cases. For long integration times, none of the cases with fixed time-step achieved good accuracy. At some point during the integration, due to a close encounter, the time-step size would need to be smaller than the lower limit selected for the bridge time step, which led to sudden increases in the energy error. Adapting the time step dynamically using RL helped to maintain the energy error constant for longer periods of time. However, in some cases, the jumps in energy error also appeared with the RL method. Therefore, we implemented a hybrid method that identifies sudden changes in energy error and recursively reduces the time-step size at a given step until the change in energy error is below a predefined limit. 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 without a major increase in computation time. In many cases (see Figure 5.15), 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 represents a robust solution for long-term integration and can be used for astronomical simulations to improve their performance and accuracy without requiring 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 small energy error is an indication of the method

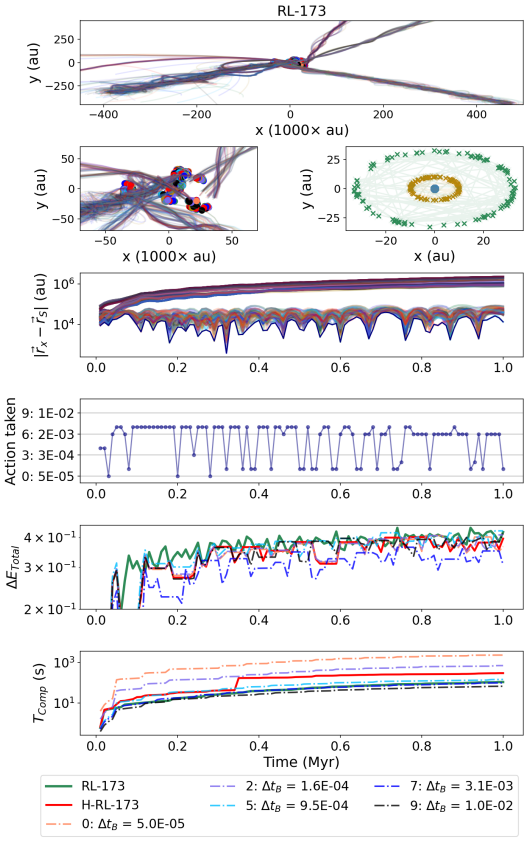


Figure 5.19: Comparison of fixed Δt_B to our RL model for 100 time steps (1 Myr). We present the trajectory in Cartesian coordinates of the star cluster (top-row panel), a close-up view of the star cluster evolution (second row, left panel), and the trajectory of the planetary system (second-row, right panel). The distance between each star to the one containing the planetary system is shown in the third row, the actions taken by the RL algorithm in the fourth row, the energy error at each time step for each study case in the fifth row, and the computation time for each study case in the last row. The simulation is run for Seed 1 for two initializations with Seeds 1 and 2 and `BHTree` integrator for the star cluster evolution.

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 (Boekholt & Portegies Zwart (2015b)): changes in the time step can lead to a different realization (Trani et al. (2024)). It is however important to take into consideration that the energy error becomes less reliable as a measurement of accuracy as the number of bodies increases.

We have shown how the performance of our method compares to that of the fixed Δt_B cases using the mean and standard deviation of the accuracy and computation time for 10 initializations. We observe that different initial realizations result in large ranges

of performance even with the same choice of integration settings. In chaotic problems, a change in the time-step size can lead to divergence in the dynamical behavior of a system. It is therefore important to focus the analysis of the performance of a method on statistical results, rather than on a direct comparison of the dynamical evolution. For the same reason, it is not trivial to find a baseline with which to compare the results. There is currently no implementation to choose or dynamically adapt the time-step size of the `Bridge` method. We 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.

The `Bridge` methods are valid as long as the system simulated is separable. We see for example in Figures 5.11, 5.17, and 5.18, that simulations with large energy errors appear for all cases of Δt_B. We identify those as situations in which at least one planet has escaped the planetary system ($e \geq 1$). When planets become unbound or a star from the cluster crosses the planetary system, the bodies involved should be integrated together to avoid large energy errors. More advanced coupling methods such as `Nemesis` (Portegies Zwart et al. (2020)) allow the bodies to be integrated as part of the parent or the child at different moments of the integration to adapt to the needs of the problem. This solution is not available for the `Bridge` methods. When using `Bridge` and `iBridge`, this problem can be mitigated by using smaller bridge time steps during close encounters. However, correctly capturing this interaction may require smaller time-step sizes than the minimum limit chosen for this problem (see Table 5.2). Therefore, we see a slight improvement in these cases when using the hybrid RL method. In this study, we have chosen a maximum of 4 hybrid-method iterations when reducing the time-step size at each step. Increasing the number of iterations will lead to better accuracies, at the cost of computation time. This means that the H-RL method (assuming an infinite number of maximum iterations) has the capability of finding the right time-step size even in cases with very close encounters between a star and the planetary system.

We have performed our studies for a small number of bodies (5 to 15 stars). To better understand the advantages of the `Bridge` methods compared to direct integration, we plot in Figure 5.20 the accuracy (final energy error) and computation time as a function of the number of stars in the cluster. The computation cost of direct integration scales quadratically with the number of bodies. For $N \leq 200$, direct integration performs better than the `Bridge` methods both in terms of energy error and computation time. However, for $N \geq 200$, the energy error grows and the computation time is almost twice as expensive as with `iBridge`. We plot four cases of `iBridge`. The first one is `iBridge` with fixed time-step size. Secondly, we use the hybrid strategy in this fixed time-step size case. Then, we plot the `iBridge` method with the RL implementation that selects the time-step size. Finally, we show the hybrid RL `iBridge` method. We observe how as N increases, the hybrid method tends to reduce the energy error compared to the non-hybrid case. RL also leads to smaller energy errors than with fixed Δt_B. Regarding computation time, we see that the fixed Δt_B case of `iBridge` requires larger computation times than with RL for $N = 200$. However, all cases of the `iBridge` led to lower computation times than direct integration as the number of bodies increased. The results shown in Figure 5.20 are based on one initialization with seed 3 instead of statistical results based on initializations with different seeds. We aim to observe the difference between direct integration and the `iBridge` methods, but a more detailed comparison between them

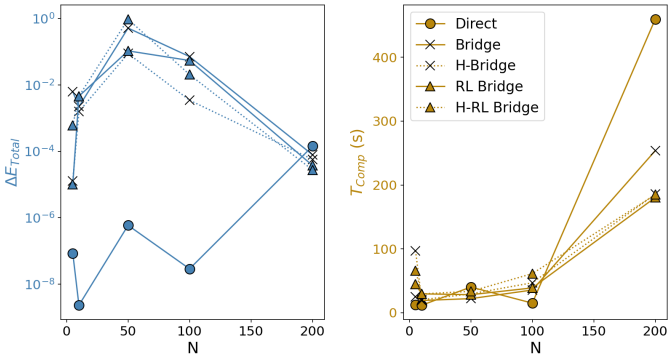


Figure 5.20: Comparison of the total energy error and computation time for an initialization with seed 3 run 40 steps with 9 stars. We compare the results with direct integration, with our `iBridge`, a hybrid implementation of the `iBridge`, and the cases with RL and H-RL.

would benefit from a larger sample size.

We have currently performed experiments for $N = 5, 9, 15$ and trained the RL algorithm with clusters with up to 20 stars. Our trained method is independent of the number of stars in the cluster, which we demonstrate in Section 5.5 with an experiment for $N = 1,000$.

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 without the need for retraining, although the performance might decrease as the setup diverges from the one used to train the RL method. Future work should focus on understanding the extrapolation capabilities of the trained model to a larger variety of star clusters. Additionally, we have not performed a hyperparameter optimization, which we believe could also improve the performances shown here.

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 problems with quick changes in their dynamics. 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. We demonstrated that the performance remains robust despite these changes. We have implemented a hybrid method that makes the RL solution robust against mistakes and results in more accurate simulations while optimizing computation time. This method can be applied to a variety of astrophysics simulations without major changes to improve their performance.

5.7 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).