The Tree Reconstruction Game: Phylogenetic Reconstruction Using Reinforcement Learning

Dana Azouri [1,2,†] Oz Granit [3,†] Michael Alburquerque [2,†] Yishay Mansour [3,*] Tal Pupko [2,*] and Itay Mayrose [1,*]

[1]School of Plant Sciences and Food Security, Tel Aviv University, Ramat Aviv, Tel Aviv 69978, Israel
[2]The Shmunis School of Biomedicine and Cancer Research, Tel Aviv University, Ramat Aviv, Tel Aviv 69978, Israel
[3]Balvatnik School of Computer Science, Tel Aviv University, Ramat Aviv, Tel Aviv 69978, Israel
[†]These author contributed equally.

***Corresponding authors:** E-mails: itaymay@tauex.tau.ac.il; talp@tauex.tau.ac.il; mansour@tauex.tau.ac.il.

Associate editor: Andrey Rzhetsky

Abstract

The computational search for the maximum-likelihood phylogenetic tree is an NP-hard problem. As such, current tree search algorithms might result in a tree that is the local optima, not the global one. Here, we introduce a paradigm shift for predicting the maximum-likelihood tree, by approximating long-term gains of likelihood rather than maximizing likelihood gain at each step of the search. Our proposed approach harnesses the power of reinforcement learning to learn an optimal search strategy, aiming at the global optimum of the search space. We show that when analyzing empirical data containing dozens of sequences, the log-likelihood improvement from the starting tree obtained by the reinforcement learning–based agent was 0.969 or higher compared to that achieved by current state-of-the-art techniques. Notably, this performance is attained without the need to perform costly likelihood optimizations apart from the training process, thus potentially allowing for an exponential increase in runtime. We exemplify this for data sets containing 15 sequences of length 18,000 bp and demonstrate that the reinforcement learning–based method is roughly three times faster than the state-of-the-art software. This study illustrates the potential of reinforcement learning in addressing the challenges of phylogenetic tree reconstruction.

***Key words:* phylogenetics, reinforcement learning, machine learning, artificial intelligence, evolution, molecular biology.**

Introduction

A phylogenetic tree is a hypothesis regarding the evolutionary relations among the studied sequences or organisms. Reconstructing a phylogenetic tree for a group of organisms has been a fundamental challenge in evolutionary research since Darwin's time. Inferred phylogenies hold a great amount of information regarding the underlying evolutionary process, and their accurate inference is critical for numerous downstream analyses spanning molecular evolution, ecology, and genomics. Leading approaches for phylogeny reconstruction rely on probabilistic evolutionary models that describe the stochastic processes of nucleotide, amino acid, and codon substitutions (Yang 2007). Under the maximum-likelihood paradigm of phylogeny reconstruction, the tree topology, its associated branch lengths, and parameters that dictate the substitution rates and the site-specific evolutionary rates are optimized for a given multiple sequence alignment (MSA). Notably, the number of possible tree topologies increases super-exponentially with the number of sequences. When only a few dozen of sequences are analyzed, there are already billions of alternative phylogenetic tree topologies that could potentially describe their evolutionary relationships, rendering the search for the best tree algorithmically challenging.

The computational search for the maximum-likelihood tree topology was previously shown to be NP-hard (Chor and Tuller 2005). Thus, tree search methodologies rely on a specified heuristic strategy, which must balance accuracy and running time. At present, heuristics employed by the community depend on the intuition of the algorithm developers and their expert knowledge regarding the phylogenetic search space. These are usually based on a hill-climbing rearrangement algorithm that defines neighboring trees (Whelan 2007). Typically, a search algorithm begins from an initial tree and iteratively replaces the current one via rearrangement to a neighbor with a higher likelihood, until no better neighbor can be found.

Received: July 02, 2023. **Revised:** May 17, 2024. **Accepted:** May 28, 2024

This procedure results in a tree that is locally better than all its neighbors, but this tree might not be the global optimum. In order to increase the probability of finding the global optimum tree, several techniques have been considered, e.g. initiating the search from multiple starting points, applying simulated annealing (that accepts suboptimal moves with a certain probability) (Stamatakis 2005), and employing genetic algorithms (in which different areas of the search space are being explored through the use of crossover, mutation, and selection operators on a population of candidate solutions) (Lewis 1998). Our aim here is not to devise a specific novel search strategy, but rather, devise an artificial intelligence (AI) framework, which automatically searches among alternative strategies (policies hereafter) for an optimal one. This framework views the strategy as an evolving entity, which is continuously optimized based on experience, i.e. training data.

Machine learning has been applied to multiple tasks in biology, including molecular biology, evolutionary, and ecology research (Tarca et al. 2007; Schrider and Kern 2018; Suvorov et al. 2020; Zou et al. 2020; Azouri et al. 2021; Haag et al. 2022; Zaharias et al. 2022). Reinforcement learning (RL) is a subfield of machine learning that is focused on learning to optimize long-term goals. Over the years, RL has had many successes, from playing backgammon (in the 1990s) to playing Go and Atari games (in recent years). RL applications are usually modeled as a Markov decision process, in which an *agent* (an RL learner) interacts with its *environment* (the representation of the problem space) in discrete time steps. In each step, the agent chooses an *action* (a move) to be taken given the current *state*. The transition between the current state and the next state is influenced by the agent's action, which can be deterministic or stochastic. Another component of RL is a numeric feedback termed "the reward," which depends on the current state and on the action taken. Given a sequence of rewards, the return function aggregates multiple rewards to one objective criterion, which implicitly defines the goal of the learner (to maximize "the return"). Popular *returns* include the finite horizon (H) return, which considers only the first H returns, for some parameter H, and the discounted return, which weights the reward at time t by γ^t for some discount factor $\gamma < 1$.

The learner's main task is to select actions that would maximize the return. This is done through a *policy*, which is a mapping from states to actions, such that given a current state the policy selects an action. For any given policy, a *value function* can be computed, which is the expected return from each state. Our task is to identify the optimal policy, which induces an optimal value function. It is well known that there always exists a deterministic optimal policy. We will consider *terminating environments* in which the interaction terminates eventually. The sequence of states, actions, and rewards from the start state until termination is called an episode.

The RL characteristics of exploratory search and delayed reward, together with the capability of optimizing a sequence of actions (i.e. a policy) for the task of interest, distinguish RL from other domains of machine learning (Sutton and Barto 1998; Szepesvári 2010). Accordingly, RL is beneficial in environments where the task is aimed at reaching a winning state at the end of a procedure, rather than aimed at optimizing some myopic (or immediate) reward. When applied to the task of phylogeny inference, RL should allow taking nongreedy steps, which nevertheless, should allow reaching the optimal tree in the fewest number of steps. This is equivalent, for example, to allowing a chess player to make apparently suboptimal moves, such as sacrificing the queen, in order to win the game in the following several moves. Generally, a low (immediate) reward may still lead to an optimal terminal state. Thus, to develop and characterize a full RL algorithm for phylogenetic tree inference, it is necessary to design the algorithm based on a long-term plan by taking into account not only the immediate rewards but more importantly the future ones.

Setting an RL representation of the phylogenetic tree search dynamics requires a tailored representation of both the tree topology with branch length estimates and the possible actions to be taken, as well as deriving a meaningful immediate reward function. In the phylogeny context, the reward is based on the likelihood change resulting from a local modification of the tree, i.e. the log-likelihood difference between the next and current state (termed hereafter "likelihood score"). Likewise, a value function considers the entire search path, namely the estimated likelihood scores of subsequent moves.

The RL-based algorithm introduced in this study is based on optimizing a policy for phylogenetic tree search, with the aim to identify the optimal tree for a given MSA in terms of its likelihood score, relying on previous AI techniques to estimate the likelihood function without actually calculating it (Azouri et al. 2021). We modeled the phylogenetic tree search problem in a similar way as RL navigation to the highest point on a grid. Defining the grid (environment) as all possible topologies, the state of the agent is the current location, and the action is the move to a new location. The transitions in the environment are deterministic. An optimal policy of an agent would therefore be to reach the topology with the maximum-likelihood score, while taking the minimal number of steps. To account for the length of the path the agent takes till reaching the optimal topology, a discount factor γ is set (see Equation 2 in Materials and Methods). The discount factor in RL determines the weight the agent assigns to rewards in the distant future relative to those in the immediate future. If $\gamma = 0$, the agent will be completely myopic and only optimize an immediate reward. If $\gamma = 1$, the agent will evaluate each of its actions equally, based on the sum of all its future rewards, thus aiming to reach the optimal configuration but disregarding the number of steps. In the implementation described here, a discount factor $0 < \gamma < 1$ was used as a hyperparameter, which provides an algorithmic incentive to reach the higher likelihood topologies earlier in the trajectory, such that the optimal policy (given

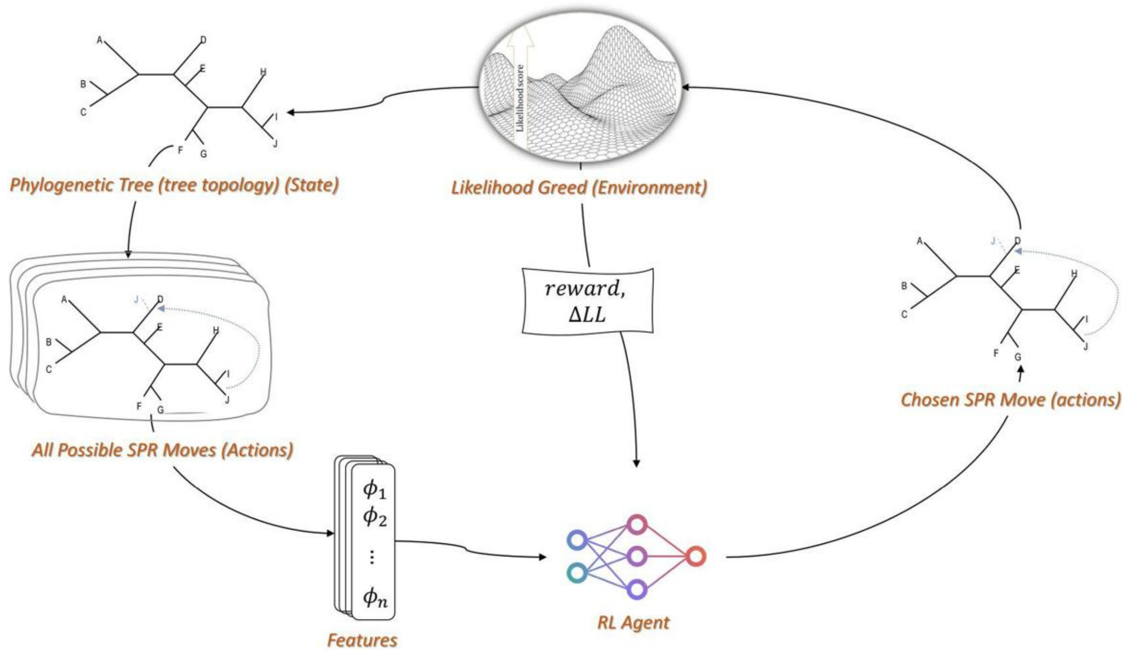
Phylogenetic Tree (tree topology) (State)
Likelihood Greed (Environment)
Likelihood score
reward, ΔLL
All Possible SPR Moves (Actions)
Chosen SPR Move (actions)
ϕ_1 ϕ_2 ⋮ ϕ_n
Features
RL Agent

Fig. 1. Modeling phylogenetic tree search as RL framework. A schematic flowchart of the RL framework applied in this study. Given an empirical sequence data set, the environment represents all phylogenetic tree topologies (states), their possible single-step SPR moves (actions), and the scaled log-likelihood difference between phylogenetic trees (rewards). We first extracted feature vectors that represent a state with its actions. These were then fed into the agent's neural network, which outputs a prediction for the best action to be taken in the agent's state, accounting for both immediate and future rewards. The reward (ΔLL; the scaled log-likelihood change) obtained following the action conducted was then stored, as part of the transition data, in the agent's memory buffer, to be later sampled during the agent's training.

a certain γ value) corresponds to finding the shortest path from the initial topology to the final topology. The discounted cumulative reward is updated recursively according to Bellman's equation (Puterman 1994) and is estimated using a Q-network (Mnih et al. 2015).

The Q-network, and thus the proposed RL framework, should receive as input a vectorized representation of the states and actions in order to estimate the value function. The representation we designed to suit these requirements is based on a set of tree features that were previously shown to effectively characterize a state–action pair in the context of a likelihood-based phylogeny reconstruction (Azouri et al. 2021). Specifically, we represented each state–action pair the agent came across during training and testing by calculating 27 features (supplementary table S1, Supplementary Material online) that are based on the current tree topology, its branch-length estimates, and a certain subtree pruning and regrafting (SPR) move (Wooding 2004) to a neighboring phylogenetic tree, by pruning a subtree from the current tree and regrafting it to the remaining tree (see Materials and Methods). After each move, i.e. an SPR modification to the current tree, the agent arrives to a new location in the tree space until reaching a predefined end of an episode (see Fig. 1 for a schematic flowchart of the RL framework applied in this study).

The goal of an RL agent is to learn a policy that would make optimal decisions in any given state of the environment. The optimization is performed during a training phase in which an agent plays numerous episodes, allowing it to collect relevant observations (i.e. transitions). That is, by exploring the dynamics of the environment the agent learns the optimal mapping between states and actions for maximizing the long-term reward signal (Sutton and Barto 1998). An important issue that needs to be tackled when developing an RL algorithm, as opposed to other types of learning, is how to balance the known trade-off between exploration and exploitation during the training phase. That is, in order to reach beneficial surfaces of high likelihood, the agent has to exploit the good transitions in the tree space it had already experienced. At the same time, it also has to explore unseen transitions, perhaps some that decrease the immediate likelihood gain, in order to make better selection of actions in the future. This was tackled using a known RL technique to sample an action based on its predicted benefit, while allowing some exploration.

Here, we developed an RL strategy for the task of searching for the maximum-likelihood phylogeny. Our method introduces novel approaches for tackling the NP-hard problem of maximum-likelihood tree search by optimizing the exploration strategy itself, which inherently considers suboptimal steps to be taken if they are expected to be beneficial in the long run. Additionally, our method does not require the direct time-consuming calculation of the likelihood function in order to predict an optimal tree. Furthermore, the computational resources needed for using this approach for phylogeny prediction are hardly influenced by the input sequence length. In the following, we first study the potential benefit of looking beyond a single step when using the classic hill-climbing optimization strategy and demonstrate that taking suboptimal moves can regularly lead to better trees in a subsequent step. Then, a framework based on deep-Q-learning (Mnih et al. 2015) for predicting the optimal tree for a given MSA is introduced. We demonstrate the application of the developed method on a set of unseen data, i.e. on unseen RL environments defined by nucleotide MSAs of up to 20 sequences. Importantly, both training and testing rely on empirical data, which were previously shown to be more challenging for phylogeny reconstruction compared to simulated data (Edwards 1995; Huelsenbeck 1995; Abdo et al. 2005; Abadi et al. 2019). Our results show that for this search space, the likelihood scores of the inferred phylogenies are comparable to those obtained from widely used methods. We then explore the feasibility of applying an agent that was trained on a certain data size on different sizes of the search spaces.

Results

The Potential Benefit of a Nongreedy Search Strategy for Phylogenetic Reconstruction

The strength of RL lies in its ability to take actions that are suboptimal in the short term for optimizing a long-term reward. To assess the potential benefit of RL in the context of phylogeny-tree search, we examined a large number of two-step trajectories. We computed the percentage of moves in which choosing two consecutive greedy moves (i.e. choosing the best single-step action available at each move) would lead to lower likelihood score than choosing a nongreedy move, followed by a greedy one. To this end, for a set of 13,200 randomly chosen starting trees, we generated all possible 1,082,400,000 (13,200 × 13,200) two-step trajectories. For each starting tree, we located the best tree (i.e. the best two-step neighbor) in terms of the likelihood score. We then quantified the fraction of starting trees for which the greedy approach was not optimal. This analysis revealed that the greedy approach was suboptimal in 33% and 41% of the cases for data sets of size 7 and 12 sequences, respectively. Interestingly, some of the intermediate moves that led to trees with higher likelihood than the greedy approach were among the worst possible first moves (Fig. 2). Although the analysis was not prolonged for more than two steps ahead, this result implies that the strict stepwise greedy optimization is not necessarily the best strategy to traverse the tree topology space, even when the search space is rather limited.

Performance Evaluation of the Proposed RL Framework

We developed a tree search framework that is entirely based on RL and tested its performance. For training the RL model, we assembled a large collection of transition data from a database of empirical MSAs (see Materials and Methods). We define transition data as all the observed shifts from one state and action combination (referred to as a "state–action pair") to a neighboring state and action combination, together with the corresponding scaled log-likelihood difference between the states. We first focused on relatively small data sets (MSAs), containing at most 12 sequences (i.e. a space size of up to ca. 10^9 topologies). For these data sets, for each state the agent came across, we explored all possible immediate SPR moves during training and testing. For larger data sets that contained 15 and 20 sequences (i.e. a space size of up to ca. 10^{20} topologies), we restricted the range of possible actions from each state in order to make the training of agents feasible for the scope of this proof-of-concept study (see Materials and Methods).

The RL algorithm aims to optimize the entire search path from the starting tree to the global maximum. Therefore, throughout this study, we measured the agent's performance at the end of an episode according to the improvement the agent achieved relative to the maximal observed improvement, i.e. the improvement obtained by RaxML-NG (Kozlov et al. 2019) from the same starting tree (see Materials and Methods; "The Performance Metric"). Thus, an agent that achieved the maximal observed improvement received a score of 1, while an agent that achieved an improvement of 150 likelihood points relative to the starting tree, but 50 likelihood points less than the estimated global maximum, received a score of 0.75. Under this definition, the performance obtained by RaxML is set to 1.

Typical examples of the likelihood improvement as the agent progresses in the search space are shown in Fig. 3. In these examples, we compared the trajectory of a trained agent to two alternative strategies: (i) a hill-climbing fully greedy strategy (i.e. evaluating in each move the log-likelihood of all possible single-step neighbors using RaxML-NG) and (ii) maximum-likelihood search obtained by running RaxML-NG (i.e. the final likelihood only); all three searches were initiated from the same random tree. Three different examples are presented: (i) the RL agent did not reach the best-known tree; (ii) the RL agent discovered the optimal tree, while taking fewer moves than the fully greedy procedure (five compared to six moves) by taking suboptimal moves; and (iii) the RL agent converged to a better tree than the greedy search.

Accuracy for Data Sets of Relatively Small Size

We evaluated the performance of the trained RL model on unseen test data. First, we examined data composed of seven sequences. For this challenge of searching in a space of size 10^3, both the hill-climbing fully greedy strategy and our RL model converged to the optimal tree with an average accuracy of 1 and 0.99999 (95% confidence interval of 0.999 to 1), respectively. We next evaluated the performance on a much larger search space, such as that defined by MSAs containing 12 sequences (i.e. search space of size 654,729,075 topologies). The average accuracy score of the fully greedy strategy and the trained model was

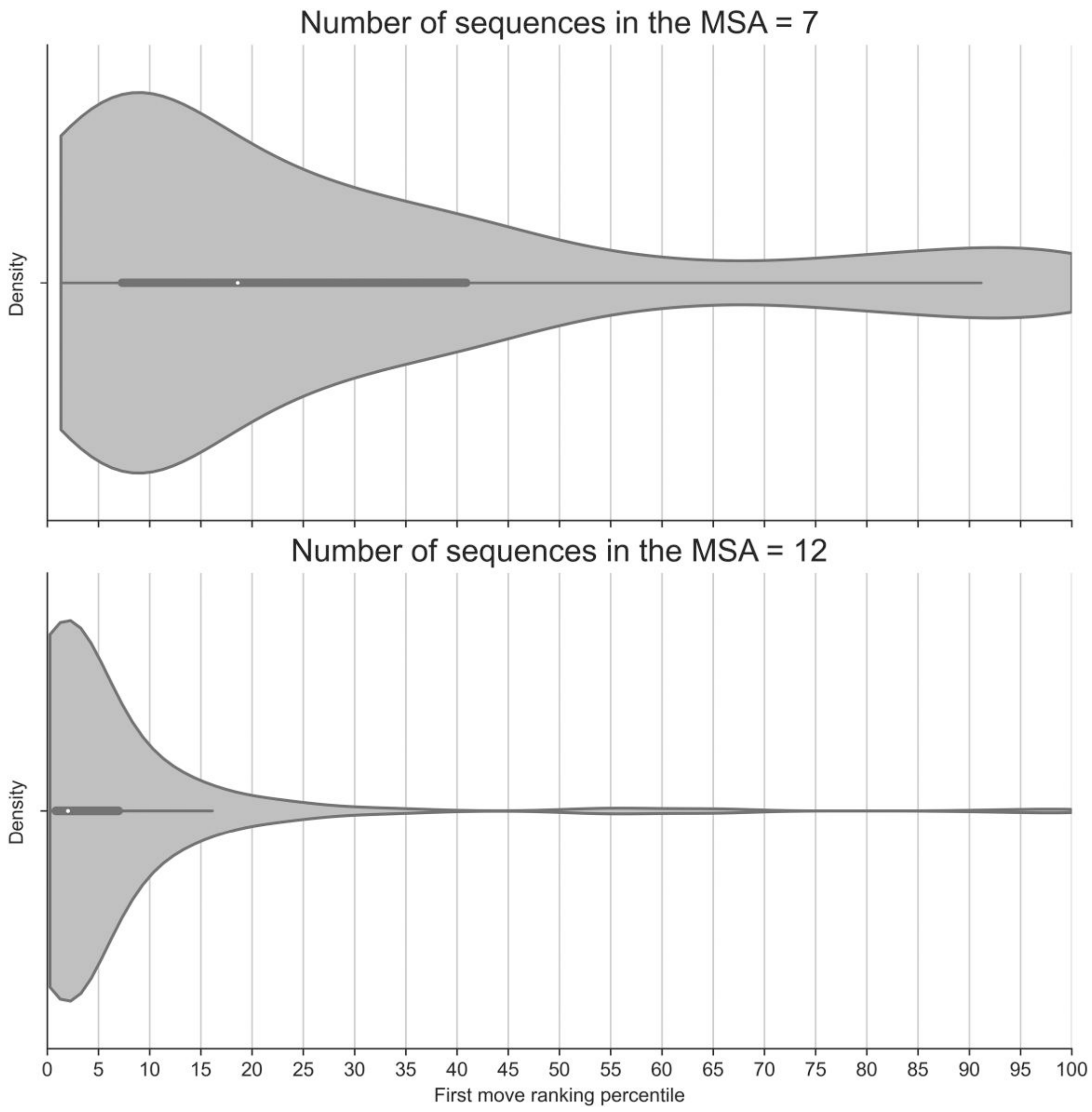

Fig. 2. The ranking percentiles of beneficial suboptimal first moves. The distribution of the percentiles of first moves that led to better trees than two-step greedy moves. The top panel presents the outcomes for data sets containing seven species while the bottom panel presents these for data sets of 12 sequences. When ranking the moves from the move that improves the likelihood score the best (rank 1) to the worst, the *x* axis denotes the ranking percentile of first moves that in the subsequent move led to better trees than two consecutive greedy moves (i.e. higher percentiles imply worst moves). The box inside each violin shows the quartiles of the data set with the white dot being the median, while the whiskers extend to show the 1.5 × interquartile range past the low and high quartiles.

0.99999 and 0.969, respectively (95% confidence interval of 0.9999 to 1 and 0.945 to 0.993). This indicates that the trained RL agent successfully learns a search strategy that can be well generalized for empirical data sets of various sources.

The above results were obtained with ten data sets for generating the training observations and 2,000 training episodes. These values were selected by analyzing the dependence between the prediction accuracy on the validation set and the number of MSAs used to generate the training data, focusing on data sets with 12 sequences. To this end, we increased the number of different empirical data sets based on which we generated the training observations from 1 to 10, 20, and 30 (but keeping the total number of episodes and transitions constant) and compared the performances (supplementary fig. S1, Supplementary Material online; see supplementary data S1, Supplementary Material online for detailed attributes of the sampled data sets). This analysis indicated that using only a single data set for learning is significantly inferior to all other sizes ($P < 0.03$ for one-way ANOVA test for the means), but using more than ten data sets does not significantly improve the performance ($P > 0.64$ for one-way ANOVA test for the means when comparing 10 to 20 and 30 data sets). Consequently, ten different empirical data sets were used to collect the training data. To further investigate the main factors affecting the performance of the RL agent, we sought to investigate the impact of the number of episodes in the training

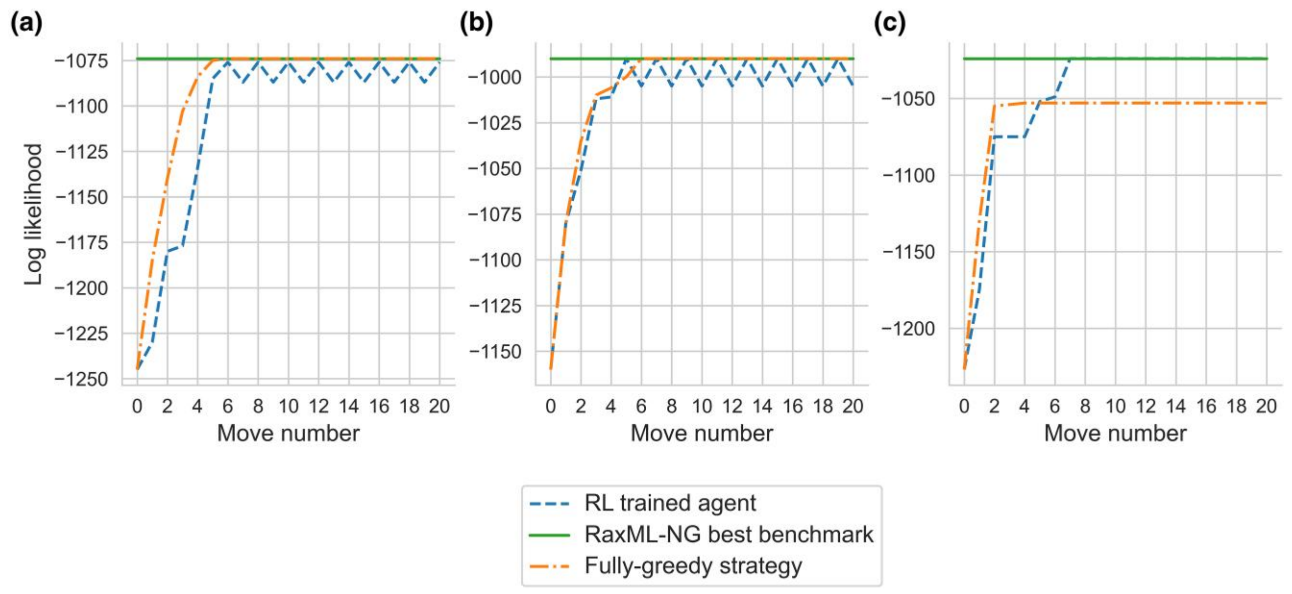

Fig. 3. Typical examples of the likelihood gain as the search progresses. a) The case where RL agent did not reach the best-known tree. b) The case where the RL agent discovered the optimal tree, while taking fewer moves than the fully greedy procedure. c) The case where the RL agent converged to a better tree than the greedy search. The *x* axis represents the SPR move number, while the *y* axis represents the log-likelihood achieved following each move of a trained agent (dashed line), a hill-climbing-fully greedy strategy (dash-dotted line), and the maximum-likelihood score obtained by RaxML-NG (solid line). Different panels represent different tests, on data sets containing 12 sequences a, b) and 15 sequences c).

phase on the validation accuracy. The accuracy increased as a function of the number of episodes ($P < 0.004$; Pearson correlation coefficient for testing noncorrelation of the means), reaching a plateau at around 2,000 episodes. Although the increase in accuracy was statistically nonsignificant when increasing the number of episodes in the range between 1,500 and 5,000 (supplementary fig. S2, Supplementary Material online; $P > 0.37$ for one-way ANOVA test for the means), the best accuracy was obtained when 2,000 episodes were used during test. To balance runtime and accuracy, the results across the entire analyses are presented using 2,000 episodes and ten distinct empirical MSAs (data sets) to generate the training data.

RL for Large Search Space

Search spaces of data sets containing 15 and 20 sequences are of size 7.9×10^{12} and 2.2×10^{20} topologies, respectively. For these data sets, feature extraction of all possible neighbors of a given tree, either at the learning stage or when searching for the best tree, is computationally demanding. Thus, we limited the number of considered neighbors of a given state by applying a restriction on the SPR moves, considering only local changes in the tree topology as commonly performed in various tree search heuristics (Stewart et al. 2001; Stamatakis et al. 2005) (see Materials and Methods). When applying this procedure in both training and testing, the average performance of the trained models for 15 sequences was 0.999 (95% confidence interval of 0.998 to 1.001). This suggests that narrowing the range of possible neighbors should be considered as a technique for training RL agents and inferring the phylogenies for data sets with large phylogenetic search spaces. When data sets with 20 sequences were considered, the test accuracy was lower; i.e. the average test performance was 0.89. We speculate that this performance could be improved using alternative, more exhaustive, data collection methodologies (see Discussion).

Table 1 Accuracy scores of zero-shot experiments

	7	12	15	20
7	*0.999*	…	…	…
12	0.999	*0.969*	…	…
15	0.998	0.973	*0.993*	…
20	0.999	0.93	0.993	*0.892*

The table details the performance of each pretrained agent of a certain data set size (row) to each other smaller data set size (column), while the main diagonal (italic font) shows the performance values on test data of the same size. Each cell shows the accuracy score of the trained model, averaged over the test data sets.

Employing Pretrained Agents across Data Sizes

Our learning so far concentrated on RL training on data sets of specified size. To assess the potential of using agents that were trained on a specific data set size to solve the phylogenetic search problem for varied number of sequences, we sought to apply zero-shot testing (Higgins et al. 2017). Specifically, we investigated the predictive power when testing pretrained agents of up to 20 sequences on data sets with fewer sequences and found comparable performance (Table 1). For example, the performance of a zero-shot agent trained on data sets containing 15 sequences on unseen environments of data sets containing 12 sequences obtained an averaged accuracy score of 0.973, which is slightly

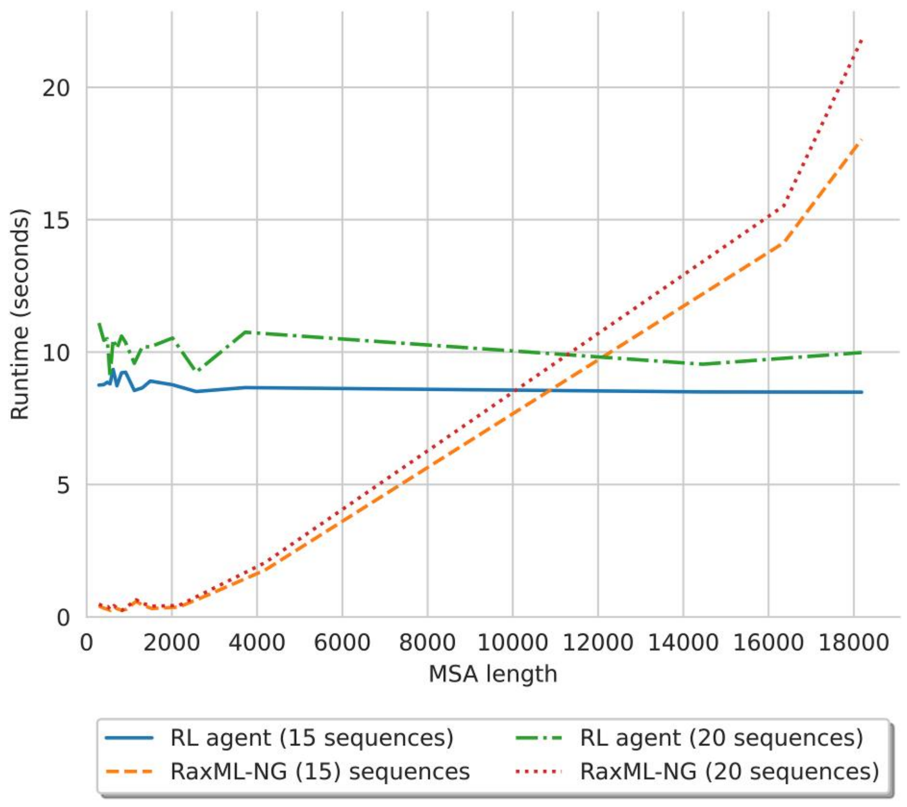

Fig. 4. Running time. The average inference running time in seconds (*y* axis) relative to the length of the sequences analyzed (*x* axis; 100 data points binned to 17 groups). In solid line and dashed line are the average running times of inferring the optimal tree for data sets with 15 sequences using the RL trained agent and RaxML-NG (with the same single-random-starting point), respectively. Similarly, in dash-dotted line and dotted line are the running times for data sets containing 20 sequences, of the RL agent and RaxML-NG, respectively.

better than that obtained by an agent that was trained and tested on an environment of 12 sequences (average accuracy of 0.969). Overall, this analysis indicates that a transfer between environments of different sizes does exist and that this approach could potentially assist in solving varied phylogenetic search space environments.

Running Times

In this study, we focused on developing the conceptual aspects of RL phylogenetics, and as part of this, we developed a prototype implementation. This prototype did not undergo cycles of optimization; e.g. a large portion of the computational runtime is devoted to feature extraction, which in the current version is implemented inefficiently in Python. For 15 sequences, for example, the training of an RL agent took 600 CPU hours. However, once the agent is trained, the time required to predict the optimal tree takes a few seconds. Specifically, we compared the runtime required to reconstruct the optimal tree to that of RaxML-NG. For data sets with 15 sequences, running the trained agent took 8.7 s on average (ranged between 8.4 and 9.3 s), of which, 7 s for extracting the features, and 1.7 s for all other computational tasks, e.g. estimating the *Q* function. Noticeably, the running time does not depend on MSA length (Fig. 4). For the same data sets, the likelihood computation of RaxML-NG took 8.7 s on average, but these varied widely from less than half a second for short MSAs (up to 800 bp) to 18 s for very long ones (more than 16,500 bp). The same trend was observed when data sets with 20 sequences were considered (Fig. 4).

Discussion

For many biological domains, spanning diverse fields such as ecology, genomics, systematics, and epidemiology research, an accurate inference of the underlying phylogeny is indispensable. As such, the development of more accurate phylogeny reconstruction techniques is an ongoing effort that continuously progressed with the type and size of data analyzed, the computational resources available, and algorithmic developments. Numerous computational techniques were imported from the fields of statistics and computer science to improve phylogenetic tree reconstruction. These include treating character evolution as a Markov process (Felsenstein 1981), Branch-and-Bound (Hendy and Penny 1982), Markov chain Monte-Carlo (Yang and Rannala 1997), genetic algorithms (Lewis 1998), simulated annealing (Stamatakis 2005), and more recently, machine learning (Suvorov et al. 2020; Zou et al. 2020; Azouri et al. 2021; Zhicheng et al. 2023). Despite these improvements, commonly used algorithms still lack the ability to provide an optimal solution. In this study, we propose an out-of-the-box AI approach for phylogenetic reconstruction, namely, RL.

The idea of introducing RL algorithms to the task of finding the optimal phylogenetic tree is based on the concept of optimizing a strategy for the tree search, rather than incrementally optimizing the likelihood gain within a series of steps. RL includes several aspects that together could prove particularly beneficial to phylogeny inference. First, similar to simulated annealing, it allows taking suboptimal steps as part of the search strategy. Our results above demonstrate that this often enables more efficient convergence to optimal trees. Second, and unlike any other existing approach, our algorithm directly optimizes a policy based on empirical training data, without the need of predetermined heuristics. This means that an agent can decide to be greedy or to take suboptimal moves according to the specific characteristics of the data and the specific position in the tree space. Third, our agent moves without optimizing the likelihood directly, potentially reducing running time, especially for long sequences.

When a phylogenetic tree is provided as input to machine learning algorithms, it must be represented as a vector. In a recent study, we represented a tree and its SPR neighbors as a vector of 19 features (shown in bold in supplementary table S1, Supplementary Material online), and showed that we could predict optimal SPR moves without computing the likelihood function (Azouri et al. 2021). In this work, we exploited such tree representation technique for training an RL agent, which can successfully traverse previously unseen phylogenetic spaces of empirical data sets. This study could thus serve as a benchmark for different representations of a phylogenetic search space, which reportedly until current days, was missing. We expect that additional improvement in representing trees and alignments would further improve RL-based tree search.

Two recent studies employed RL to phylogeny (Liptak and Kiss 2021; Zhu and Cai 2021). In these studies, RL was used in the context of distance-based methods, which are known to be faster, albeit less accurate than likelihood-based methods (Saitou and Nei 1987; Ogden and Rosenberg 2006). These studies and ours showed the potential of using RL for the complicated tree search problem and emphasized the challenge of training an agent on topological spaces of more than 20 sequences. While our algorithm and representation are not theoretically limited by the number of sequences in the data, at present, accuracy was not satisfactory for data sets with more than 20 sequences. We believe that promising future directions toward the application of RL to large data sets should concentrate on the following:

(i) Improving the training data. This includes the size of the training observations, as well as its quality. Increasing the number of training examples necessitates training the agent longer, which depends on the availability of computational resources. In this regard, transfer learning should enable repeatedly using previous models trained on small data sets as the starting point for learning larger ones (Karimpanal and Bouffanais 2019). As for the data quality, it requires developing means to collect training observations of those cases that would maximize the learning of the agent, for example by collecting more observations from regions of high likelihood, which could provide valuable information for traversing these important parts of the likelihood surface.

(ii) Improving code efficiency. Although the running time of the proposed methodology is hardly affected by the input sequences lengths and thus is suitable for large-scale data, a more efficient implementation with regard to feature extraction could enable better usage of the computing resources available, particularly during training.

(iii) Using an alternative, automatic, representation of the tree search space. For example, it has been recently proposed to represent tree topologies with embedded node features based on graph neural networks (Cheng 2023). This direction of extracting learnable topological features can potentially better capture the complexity of empirical phylogenetic environments, without requiring to handcraft additional features.

Another possible direction for improving the effectiveness of RL for phylogenetics could be considering alternative immediate reward functions, e.g. directly calculating the likelihood function as the reward during inference instead of estimating the likelihood change. Additionally, while, in this work, we considered SPR actions only, the combination of complementary neighborhood definitions for local search phylogenetic reconstruction algorithms, such as nearest-neighbor interchange (Robinson 1971) and tree bisection and regrafting (Allen and Steel 2001), could be considered when modeling the tree search dynamics. Expanding the range of possible actions could thus help the agent fine-tune the search strategy when it is in low or high likelihood regions. An additional variation to the current implementation could be extracting additional features that are not topology specific. For instance, the nucleotide frequencies, the entropy score of the alignment, the number of gap blocks in it, and the average gap block length. Yet, when considering large data sets, this kind of modification could come at the cost of runtime, which will become dependent on the alignment size. Lastly, there are a large number of variants of RL algorithms. As part of the development of the current implementation, we have examined the applicability of alternative RL-based schemes, e.g. policy-based algorithms. This procedure demands more computation resources and was attempted for relatively small data sizes of up to 12 sequences. However, other existing RL frameworks could prove beneficial for the task of phylogenetic reconstruction.

The main conceptual novelty of our approach is to view phylogenetic tree reconstruction as a dynamic game, in which the rules are specified, but the winning strategy is unknown and difficult to optimize. In such a case, better inference is obtained following numerous games generated in silico. We expect that, with time, RL will be introduced for additional evolutionary genomics optimization problems, including MSA, synteny inference, and elucidating complex patterns of population dynamics.

Materials and Methods

A RL Algorithm for Predicting the Maximum-Likelihood Phylogeny

The Environment

We defined the environment using (S, A, R), where S is the state space, i.e. all possible trees given a set of aligned sequences, and A is the set of possible actions, i.e. all possible SPR moves given a tree topology. R is the immediate reward function following a transition from state s to s'. In our setting, (s, a) deterministically determines the next state s'. $R(s, s')$ is defined as the log-likelihood difference, scaled by LL_{NJ} (the log-likelihood of the reconstructed BioNJ [Gascuel 1997] tree as implemented in PhyML 3.0 [Guindon et al. 2010]) so that the reward function would have the same magnitude across different data sets:

$$R(s, s') = \frac{LL_{s'} - LL_s}{LL_{\text{NJ}}}. \tag{1}$$

The Features. Each state–action pair (s, a) is represented by a set of 27 phylogenetically informative features from an input tree (supplementary table S1, Supplementary Material online). The feature vector, $\phi(s, a)$, captures properties of the current state (the topology and its branch lengths) and the action (one possible SPR move). Of these, 19 (bolded in supplementary table S1, Supplementary Material online) were previously developed in the context of predicting the optimal neighbor as part of a tree search (Azouri et al. 2021) and capturing, for example, features related to the topological differences between the starting and resulting trees and properties related to their branch lengths. Eight additional features were implemented in this work and are based on nonparametric bootstrap computations. While features 20 to 23 were extracted based on the UPGMA algorithm (Michener and Sokal 1957), features 24 to 27 were based on the bioNJ algorithm.

The Algorithm

The developed RL algorithm is based on a Deep Q-network (DQN) (Mnih et al. 2015), a model-free and off-policy RL algorithm. In the DQN setting, the agent learns a value function, named the quality function $Q(s, a)$, which represents the estimated benefit of a specified action in gaining some future reward, given a certain state. More specifically, we implemented a neural network Q_θ (with weight parameters θ), which represents the agent, that predicts the quality function of a state–action pair, given the feature vector $\phi(s, a)$. This predicted value is termed $Q(\phi(s, a))$ and is explained in more details below. Starting from state s, we estimated $Q(\phi(s, a))$ for all possible SPR actions and chose the action with maximal $Q(\phi(s, a))$, which defines the next state (Algorithm 1). In more detail, for each state s, the feature function ϕ extracts a feature vector for the tree (state s) and SPR move (action a). The feature vector $\phi(s, a)$ is the input for the neural network. The returned scalar is the agent's evaluation of the tree and SPR move, written as follows:

$$Q_\theta(\phi(s, a)). \tag{2}$$

The action that received the maximal evaluation by the agent is then selected, marked here as a':

$$a' = \text{argmax}_a Q_\theta(\phi(s, a)). \tag{3}$$

Finally, the final tree (after a predefined number or SPR moves) is returned. The number of unique state–action

Algorithm 1 Inference

Algorithm 1 Deep Q-Learning of Tree Reconstruction - Inference

```
Given a Feature-Extractor function $\phi$
and an action-value function $Q_\theta$:
Sample starting tree $s_0$

for $h = 0...H$ do
    for all possible SPR-moves $a$ in $s_h$ do
        Evaluate $Q_\theta$ for input $\phi(s_h, a)$ as in Exp. $(2)$
    end for

    select $a_h$ using Eq. $(3)$
    Execute SPR-move $a_h$ and observe next tree $s_{h+1}$

end for

return $s_h$
```

Algorithm 2 Training

```
Algorithm 2 Deep Q-Learning of Tree Reconstruction - Training
  Given a Feature-Extractor function $\phi$:
  Initialize action-value function $Q_\theta$ with a random set of weights $\theta$
  Initialize replay buffer $\mathcal{D}$

  for $episode = 1...M$ do

    Sample starting tree $s_0$
    for $h = 0...H$ do

      for all possible SPR-moves $a$ in $s_h$ do
        Evaluate $Q_\theta$ for input $\phi(s_h, a)$ as in Exp. $(2)$
      end for

      if $h > 0$ then
        select $a'$ using Eq. $(3)$
        Store the last transition $(\phi(s_{h-1}, a_{h-1}), r_{h-1}, \phi(s_h, a'))$ in $\mathcal{D}$
      end if

      Sample $a_h$ from exploration policy using Eq. $(5)$
      Execute SPR-move $a_h$
      Collect reward $r_h$ using Eq. $(1)$
      Observe next tree $s_{h+1}$
    end for

    for $t = 1...times - to - learn$ do
      Sample mini-batch of transitions from $\mathcal{D}$
      Perform a gradient descent step using Eq. $(4)$
    end for

  end for
```

pairs to be computed when conducting a move is $2(n-3)(2n-7)$, which is $O(n^2)$, where n is the number of sequences in the input MSA (Allen and Steel 2001). The starting state for each episode, s_0, was randomly sampled (using RaxML-NG [Kozlov et al. 2019] random tree generator), such that the agent could start the trajectory from anywhere in the tree space.

The Model

The main strength of Q-learning lies in its ability to construct a policy that maximizes the cumulative reward (Sutton and Barto 1998). In deep Q-learning, neural networks are trained to estimate the value of the Q function for unseen states and thus it combines Q-learning with a deep artificial neural network (ANN). The recursive form of the optimal return function, known as the Bellman equation (Puterman 1994), is:

$$Q^*(\phi(s_t,\ a_t)) = r_t + \ \gamma \max_a Q^*(\phi(s_{t+1},\ a)), \qquad (4)$$

where Q^* is the optimal state–action value function and r_t is the immediate reward obtained at time step t. The γ hyperparameter is the discount rate, a constant $0 \leq \ \gamma \leq 1$, which weights rewards from the uncertain far future less than the ones in the fairly confident near future. That is, a reward received k time steps in the future is worth only γ^{k-1} times what it would be worth if it were received immediately. Of note, there exists a $\gamma = 1 - \epsilon, \quad \epsilon > 0$, for which the optimal policy corresponds to the shortest path from the starting topology to the topology with the highest likelihood (up to H SPR moves away).

We would like the agent to learn a policy that does not involve taking an unlimited number of actions; i.e. we would like to balance the runtime with the improvement in likelihood. This balance is controlled by the horizon hyperparameter, denoted as H, which specifies the number of actions taken from the starting tree (Algorithm 2). Of note, each search stops after a predefined number of H steps. This is termed an episode. Importantly, the specific γ value inspires a different optimal policy (see supplementary note S1, Supplementary Material online).

Choosing an appropriate value for the discount rate is crucial for balancing short-term and long-term rewards. Setting the discount rate to 0 or 1 leads to extreme cases: $\gamma = 0$ would prioritize only the immediate rewards, while $\gamma = 1$ would treat all future rewards as equally important. The length of the horizon should be considered when choosing a discount rate. In short-term planning problems, a lower discount rate might suffice, while for long-term planning problems, a higher discount rate could be more suitable. In more detail, we attempt to choose γ such that the optimal policy it inspires is the same policy that leads to the highest likelihood tree up to H SPR moves away from the starting tree. Notice, in our setting the agent does not know about the H moves limitation, nor does it have any indication of how many moves are left in the episode. To tackle this issue, we chose a rather low γ, to incentivize collecting rewards (reaching better topologies) sooner rather than later.

Our usual setting was $\gamma = 0.9$, $H = 20$. In that setting, the agent values the reward $H + 1$ moves away at ~11% of its original value. Compared to a standard $\gamma = 0.99$, which is considered standard in many RL applications, the agent values the reward $H + 1$ moves away at ~81% of its original value. Since for our problem set, there are no rewards after move H, we kept the estimation of moves after H low, while still considering long-term rewards.

As stated above, the total number of episodes during learning is also a hyperparameter. Following each episode, the network weights are updated. Specifically, as in standard DQNs, we used the experience replay method to hold the agent's training trajectories, i.e. a buffer of a predetermined size containing transition observations (state–action pairs together with their rewards and next state–actions). At the end of each episode, H new memories are added to the buffer (and the H oldest memories are discarded), and the ANN is trained based on a batch of trajectories sampled (with replacement) from the memory buffer "time-to-learn" times (Algorithm 2). The sizes of the memory buffer and the batch, as well as the "time-to-learn," are hyperparameters of the algorithm.

Additional hyperparameters are related to the deep network architecture and the learning dynamics. These include the number of fully connected hidden layers, the number of neurons in each layer, the activation function, the loss function, the optimization algorithm, and the learning rate (supplementary table S1, Supplementary Material online).

To control the exploration–exploitation trade-off during training, we allowed the agent to take suboptimal moves with respect to the Q function. This is commonly known as an exploration policy. We used the SoftMax exploration policy that selects an action $a \in A$, where A is the set of all possible next moves, based on the following probability:

$$P(a \mid s) = \frac{e^{Q(s,a)/T}}{\sum_{x \in A} e^{Q(s,x)/T}}. \tag{5}$$

This allows greater value actions to be selected with greater chance, yet permitting some randomness. T is a hyperparameter that controls the level of exploration. Running a SoftMax function to determine the next move might cause some divergence from the optimum Q-function. These inaccuracies in the estimation of the Q function could result in the collection of more transitions in low regions of the search space during the initial training phase (when the phylogenies are quite far from the maximum-likelihood solution), as well as more transitions where the phylogenetic tree state is better at the later phases of training.

RL Agent Architecture

We implemented the ANN in Python using PyTorch (Paszke et al. 2019). The above hyperparameters were optimized via Optuna framework (Akiba et al. 2019), which is an automatic hyperparameter optimization package, particularly designed for machine learning (summary of the model hyperparameter values and further details are described in Table 2). The models were trained using an Intel(R) Xeon(R) Gold 6148 CPU @ 2.40 GHz, with 40 CPUs, 10 GB of DDR2 RAM and an X86_64 instruction set.

The Performance Metric

To measure the performance at the end of an episode, we computed the following metric: let LL_{gain} be the log-likelihood difference between the final tree and the starting tree. Let LL_{rax} be the log-likelihood difference between the maximum-likelihood tree and the starting tree, as obtained by executing RAxML-NG from the same starting tree. Both resulting topologies were subject to branch-length optimization. The ratio between these two terms $\left(\frac{LL_{\text{gain}}}{LL_{\text{rax}}}\right)$ is a number that reflects the improvement in likelihood score achieved by an agent and can be used to compare the performance between different data sets.

Empirical Data Preparation

We selected all empirical data sets with 7, 12, 15, and 20 sequences from the training data collected in Azouri et al. (2021). These represent nucleotide coding alignments (Moretti et al. 2014), user-submitted phylogenies from TreeBase (Vos et al. 2012), plant phylogenies reconstructed using the OneTwoTree pipeline (Drori et al. 2018), and genomic sequences that were aligned according to the tertiary structure of its encoded proteins (Carroll et al. 2007). For each data set size (i.e. the number of sequences in the alignment), 30 MSAs were randomly fixed as validation data, ten MSAs were randomly fixed as test data, and from the rest, ten data sets (unless otherwise specified) were randomly sampled to generate the training samples (see supplementary data S1, Supplementary Material online for detailed attributes of the sampled data sets).

Transition Data Collection

To apply the memory buffer method, the transition observations (state–action pairs together with their corresponding scaled log-likelihood differences and the corresponding next state–action pair) need to be collected

Table 2 Details of the RL configuration and the hyperparameter values

Parameter name	Value in the trained model	Additional details
NN architecture	Five fully connected hidden layers, in addition to the input layer (containing 27 neurons) and the output layer (containing a single node)	Number of neurons within each layer: {1: 4,096; 2: 4,096; 3: 2,048; 4: 128; 5: 32}
Input layer	27 neurons	
Output layer	1 neuron	For regression output
Loss function	Mean square error (MSE)	
Activation function	Leaky ReLU	
Optimizer	Adam	
Discount factor (γ)	0.9	
Replay buffer size	10,000	The maximal size of transitions collected during training
Times-to-learn	50	The number of times we sampled a batch to train the ANN
Horizon *H*	20 (for data of 7, 12, and 15 sequences), 30 (for 20 sequences)	The number of SPR moves in each episode. This hyperparameter was reoptimized when we considered different number of sequences in the analysis
Batch size	128	
Learning rate	10^{-5}	
Exploratory policy	SoftMax	With *T* parameter = 1
Episodes	2,000	Number of episodes in training

through many training episodes. To this end, each episode was initiated from a random tree, which was generated using RaxML-NG "complete random starting tree" generator, and the obtained reward was calculated (using RaxML-NG) and stored for each transition taken. Precisely, a model that is trained for 2,500 episodes of 20 SPR moves each is essentially trained over 25, 000 $\times 20 =$ 50, 000 training observations. The substitution rate parameters of a GTR + I + G model (Abadi et al. 2019) were optimized once for each data set, based on a reconstructed BioNJ tree as implemented in PhyML 3.0 (Guindon et al. 2010), and were then fixed for the following likelihood calculations of the respective data set.

When data sets of size 15 and 20 sequences were considered, the computational resource required to compute the features for all neighbors was beyond the scope of the study conducted here. Therefore, we limited the space of possible neighbors by applying a restriction on the range of SPR moves, allowing each pruned subtree to be regrafted up to a predefined radius. This radius defines the number of branches in the path between the pruned and regrafted branches, not including these branches (supplementary fig. S3, Supplementary Material online). Setting a radius of 4 narrowed the neighborhood space to a feasible task. Additionally, when data sets with 20 sequences were considered, we applied an alternative approach to collect experiences, which proved superior to the one used for smaller data sizes (see supplementary note S2, Supplementary Material online).

Data Collection for the Two-Step Preanalysis

We collected from the training data all MSAs containing 7 and 12 sequences (i.e. 51 and 81 data sets, respectively). Next, 100 random starting trees were reconstructed for each data set using RAxML-NG. We then obtained all their respective 32,640,000 and 1,049,760,000 possible two-step trajectories. Precisely, we (i) obtained all single-step SPR neighbors for each starting tree and recorded all likelihoods and (ii) recorded all likelihood scores of the single-step SPR neighbors of each of the latter trees. This allowed us to identify the best two-step neighbor of each starting tree, as well as the best neighbor reached by applying the single-greedy step twice sequentially. The likelihoods throughout this analysis were computed using RAxML-NG, allowing for branch-length optimization. The substitution rate parameters were optimized once for each data set, based on a reconstructed tree as implemented in PhyML 3.0 (Guindon et al. 2010) and were then fixed for the following likelihood calculations of the respective data set, assuming the GTR + I + G model (Abadi et al. 2019).

Supplementary Material

Supplementary material is available at *Molecular Biology and Evolution* online.

Acknowledgements

We acknowledge the Data Science & AI Center at TAU for supporting this study. D.A. was supported by The Council for Higher Education program for excellent PhD students in Data Sciences and by a fellowship from the Fast and Direct PhD Program at Tel Aviv University. M.A. was supported by a fellowship from the Edmond J. Safra Center for Bioinformatics at Tel Aviv University. O.G. and Y.M. were supported by a grant from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 882396), by the Israel Science Foundation (grant number 993/17), the Yandex Initiative for Machine Learning at Tel Aviv University, and a grant from the Tel Aviv University Center for AI and Data Science (TAD). T.P. was supported by an

Israel Science Foundation grant 2818/21. I.M. was supported by an Israel Science Foundation grant 1843/21. We would like to express our gratitude to Dr. Shiran Abadi for providing insightful feedback on the draft and to Professor Jeffrey L. Thorne for the meaningful remarks during the review process.

Author Contributions

D.A., O.G., and M.A. jointly designed and conducted the work including programming the algorithm, performing the analyses, and drafting the manuscript. Y.M., T.P., and I.M. supervised this work and revised the manuscript.

Conflict of Interest

The authors declare no competing interests.

Data Availability

The data sets contained within the empirical set have been deposited in GitHub with the identifier https://github.com/michaelalb/ThePhylogeneticGame. The code that supports the findings of this study was written in Python version 3.9.7 has been deposited in GitHub with the identifier https://github.com/michaelalb/ThePhylogeneticGame. Computation of likelihoods was executed using the following application versions: PhyML 3.0 (Guindon et al. 2010) and RAxML-NG 0.9.0 (Kozlov et al. 2019). The ANN was implemented in PyTorch (Paszke et al. 2019) version 1.13.1.

References

Abadi S, Azouri D, Pupko T, Mayrose I. Model selection may not be a mandatory step for phylogeny reconstruction. *Nat Commun*. 2019:**10**(1):934. https://doi.org/10.1038/s41467-019-08822-w.

Abdo Z, Minin VN, Joyce P, Sullivan J. Accounting for uncertainty in the tree topology has little effect on the decision-theoretic approach to model selection in phylogeny estimation. *Mol Biol Evol*. 2005:**22**(3): 691–703. https://doi.org/10.1093/molbev/msi050.

Akiba T, Sano S, Yanase T, Ohta T, Koyama M. Optuna: a next-generation hyperparameter optimization framework. *Proc 25th ACM SIGKDD Int Conf Knowl Discov Data Min*. 2019:2623–2631.

Allen BL, Steel M. Subtree transfer operations and their induced metrics on evolutionary trees. *Ann Comb*. 2001:**5**(1):1–15. https://doi.org/10.1007/s00026-001-8006-8.

Azouri D, Abadi S, Mansour Y, Mayrose I, Pupko T. Harnessing machine learning to guide phylogenetic-tree search algorithms. *Nat Commun*. 2021:**12**(1):1983. https://doi.org/10.1038/s41467-021-22073-8.

Carroll H, Beckstead W, O'Connor T, Ebbert M, Clement M, Snell Q, Mcclellan D. DNA reference alignment benchmarks based on tertiary structure of encoded proteins. *Bioinformatics* 2007:**23**(19): 2648–2649. https://doi.org/10.1093/bioinformatics/btm389.

Cheng Z. 2023. Learnable topological features for phylogenetic inference via graph neural networks, arXiv 2302.08840, preprint: not peer reviewed.

Chor B, Tuller T. Maximum likelihood of evolutionary trees: hardness and approximation. *Bioinformatics* 2005:**21**(Suppl 1):i97–106. https://doi.org/10.1093/bioinformatics/bti1027.

Drori M, Rice A, Einhorn M, Chay O, Glick L, Mayrose I. OneTwoTree: an online tool for phylogeny reconstruction. *Mol Ecol Resour*. 2018: **18**(6):1492–1499. https://doi.org/10.1111/1755-0998.12927.

Edwards AWF. Assessing molecular phylogenies. *Science* 1995: **267**(5195):253–253. https://doi.org/10.1126/science.7809633.

Felsenstein J. Evolutionary trees from gene frequencies and quantitative characters: finding Maximum likelihood estimates. *Evolution* 1981:**35**(6):1229. https://doi.org/10.2307/2408134.

Gascuel O. BIONJ: an improved version of the NJ algorithm based on a simple model of sequence data. *Mol Biol Evol*. 1997:**14**(7):685–695. https://doi.org/10.1093/oxfordjournals.molbev.a025808.

Guindon S, Dufayard J-F, Lefort V, Anisimova M, Hordijk W, Gascuel O. New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of PhyML 3.0. *Syst Biol*. 2010:**59**(3):307–321. https://doi.org/10.1093/sysbio/syq010.

Haag J, Höhler D, Bettisworth B, Stamatakis A. From easy to hopeless-predicting the difficulty of phylogenetic analyses. *Mol Biol Evol*. 2022:**39**(12):msac254. https://doi.org/10.1093/molbev/msac254.

Hendy MD, Penny D. Branch and bound algorithms to determine minimal evolutionary trees. *Math Biosci*. 1982:**59**(2):277–290. https://doi.org/10.1016/0025-5564(82)90027-X.

Higgins I, Pal A, Rusu A, Matthey L, Burgess C, Pritzel A, Botvinick M, Blundell C, Lerchner L. 2017. DARLA: improving zero-shot transfer in reinforcement learning. In: 34th International Conference on Machine Learning (ICML). 3:2335–2350.

Huelsenbeck JP. Performance of phylogenetic methods in simulation. *Syst Biol*. 1995:**44**(1):17–48. https://doi.org/10.2307/2413481.

Karimpanal TG, Bouffanais R.. Self-organizing maps for storage and transfer of knowledge in reinforcement learning. *Adapt Behav*. 2019:**27**(2):111–126. https://doi.org/10.1177/105971231881.

Kozlov AM, Darriba D, Flouri T, Morel B, Stamatakis A. RAxML-NG: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference. *Bioinformatics* 2019:**35**(21):4453–4455. https://doi.org/10.1093/bioinformatics/btz305.

Lewis PO. A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data. *Mol Biol Evol*. 1998:**3**(3):277–283. https://doi.org/10.1093/oxfordjournals.molbev.a025924.

Liptak P, Kiss A. Constructing unrooted phylogenetic trees with reinforcement learning. *Stud Univ Babeș-Bolyai Inform*. 2021:**66**(1): 37. https://doi.org/10.24193/subbi.2021.1.03.

Michener CD, Sokal RR. A quantitative approach to a problem of classification. *Evolution* 1957:**11**(2):490–499. https://doi.org/10.2307/2406046.

Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, *et al.* Human-level control through deep reinforcement learning. *Nature* 2015:**518**(7540):529–533. https://doi.org/10.1038/nature14236.

Moretti S, Laurenczy B, Gharib WH, Castella B, Kuzniar A, Schabauer H, Studer RA, Valle M, Salamin N, Stockinger H, *et al.* Selectome update: quality control and computational improvements to a database of positive selection. *Nucleic Acids Res*. 2014:**42**(D1): D917–D921. https://doi.org/10.1093/nar/gkt1065.

Ogden TH, Rosenberg MS. Multiple sequence alignment accuracy and phylogenetic inference. *Syst Biol*. 2006:**55**(2):314–328. https://doi.org/10.1080/10635150500541730.

Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, *et al.* Proceedings of the 33rd international conference on neural information processing systems. Red Hook (NY): Curran Associates Inc.; 2019. Article 721. p. 8026–8037.

Puterman ML. *Markov decision processes: discrete stochastic dynamic programming*. Wiley; 1994.

Robinson DF. Comparison of labeled trees with valency three. *J Comb Theory, Ser B*. 1971:**11**(2):105–119. https://doi.org/10.1016/0095-8956(71)90020-7.

Saitou N, Nei M. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol Biol Evol*. 1987:**4**(4):406–425. https://doi.org/10.1093/oxfordjournals.molbev.a040454.

Schrider DR, Kern AD. Supervised machine learning for population genetics: a new paradigm. *Trends Genet*. 2018:**34**(4):301–312. https://doi.org/10.1016/j.tig.2017.12.005.

Stamatakis A, Ludwig T, Meier H. RAxML-III: a fast program for maximum likelihood-based inference of large phylogenetic trees. *Bioinformatics* 2005:**21**(4):456–463. https://doi.org/10.1093/bioinformatics/bti191.

Stamatakis A. 2005. An efficient program for phylogenetic inference using simulated annealing. In: Proceedings - 19th IEEE International Parallel and Distributed Processing Symposium.

Stewart CA, Hart D, Berry DK, Olsen GJ, Wernert EA, Fischer W. 2001. Parallel implementation and performance of fastDNAml. Proceedings of the ACM/IEEE conference on supercomputing.

Sutton RS, Barto AG. *Reinforcement learning: an introduction*. MA: MIT Press; 1998.

Suvorov A, Hochuli J, Schrider DR. Accurate inference of tree topologies from multiple sequence alignments using deep learning. *Syst Biol*. 2020:**69**(2):221–233. https://doi.org/10.1093/sysbio/syz060.

Szepesvári C. Algorithms for reinforcement learning. *Synth Lect Artif Intell Mach Learn*. 2010:**4**:1–103.

Tarca AL, Carey VJ, Chen X, Romero R, Drăghici S. Machine learning and its applications to biology. *PLoS Comput Biol*. 2007:**3**(6):e116. https://doi.org/10.1371/journal.pcbi.0030116.

Vos RA, Balhoff JP, Caravas JA, Holder MT, Lapp H, Maddison WP, Midford PE, Priyam A, Sukumaran J, Xia X, *et al.* NeXML: rich, extensible, and verifiable representation of comparative data and metadata. *Syst Biol*. 2012:**61**(4):675–689. https://doi.org/10.1093/sysbio/sys025.

Whelan S. New approaches to phylogenetic tree search and their application to large numbers of protein alignments. *Syst Biol*. 2007:**5**(5):727–740. https://doi.org/10.1080/10635150701611134.

Wooding S. Inferring phylogenies. *Am J Hum Genet*. 2004:**74**(5):1074. https://doi.org/10.1086/383584.

Yang Z, Rannala B. Bayesian phylogenetic inference using DNA sequences: a Markov chain Monte Carlo method. *Mol Biol Evol*. 1997:**14**(7):717–724. https://doi.org/10.1093/oxfordjournals.molbev.a025811.

Yang Z. PAML 4: phylogenetic analysis by maximum likelihood. *Mol Biol Evol*. 2007:**24**(8):1586–1591. https://doi.org/10.1093/molbev/msm088.

Zaharias P, Grosshauser M, Warnow T. Re-evaluating deep neural networks for phylogeny estimation: the issue of taxon sampling. *J Comput Biol*. 2022:**29**(1):74–89. https://doi.org/10.1089/cmb.2021.0383.

Zhicheng W, Sun J, Yuan G, Yongwei X, Zha Y, Kuan L, Wei Z, Chi Z, Jian Z, Li Z. Fusang: a framework for phylogenetic tree inference via deep learning. *Nucleic Acids Res*. 2023:**51**(20):10909–10923. https://doi.org/10.1093/nar/gkad805.

Zhu T, Cai Y. 2021. Applying neural network to reconstruction of phylogenetic tree. In: 13th International Conference on Machine Learning and Computing. 146–152.

Zou Z, Zhang H, Guan Y, Zhang J, Liu L. Deep residual neural networks resolve quartet molecular phylogenies. *Mol Biol Evol*. 2020:**37**(5):1495–1507. https://doi.org/10.1093/molbev/msz307.