PHYLOGENETIC RECONSTRUCTION USING DEEP REINFORCEMENT LEARNING

Daniël Zee **Kevin Bretz**

ABSTRACT

Phylogenetic reconstruction—the estimation of evolutionary relationships between biological sequences—is a fundamental task in bioinformatics. Traditional methods rely on greedy heuristics that often converge to local optima. In this work, we investigate the application of deep reinforcement learning to guide the tree search process. We re-implement a DQN-based framework and extend it by introducing Soft Q-Learning (SQL) to improve policy stability and exploration. Furthermore, we explore two distinct state representations: an extended set of hand-crafted features and a learned graph-based representation utilizing Graph Attention Networks (GATv2). Our results demonstrate that SQL consistently outperforms DQN in both success rate and variance. We also show that while hand-crafted features remain highly effective, our optimized feature extraction pipeline reduces computational complexity from cubic to quadratic time. Finally, we provide a proof-of-concept for graph-based representations, showing their potential to learn search dynamics directly from tree topology.

1 INTRODUCTION

Phylogenetic reconstruction is a fundamental task in bioinformatics that involves estimating the evolutionary relationships between biological sequences. Under the Maximum Likelihood (ML) framework, the goal is to identify the tree topology that best explains the observed data. However, the space of possible trees grows super-exponentially with the number of taxa, making an exhaustive search impossible. Practical tools like RAxML-NG instead rely on heuristic search strategies, such as Subtree Pruning and Regrafting (SPR) moves, to navigate the likelihood landscape.

While effective, these heuristics are typically greedy and do not adapt to the specific structure of a given dataset. Recent work has demonstrated that tree search can be framed as a sequential decision-making problem, allowing Reinforcement Learning (RL) agents to learn optimized search policies. Azouri et al. (2024) recently showed that Deep Q-Networks (DQN) can guide SPR moves more effectively than standard heuristics. However, issues regarding training stability and the reliance on hand-crafted features remain.

In this work, we extend this RL framework by introducing Soft Q-Learning (SQL). By using an entropy-regularized objective, SQL improves exploration and provides more stable training compared to the standard DQN baseline. We further investigate the impact of state representation by comparing an expanded set of hand-crafted features against a learned graph-based representation using Graph Attention Networks (GATv2). This allows the agent to learn structural dependencies directly from the tree topology.

To address the computational cost of evaluating many moves per step, we implement an optimized feature extraction pipeline. By utilizing Euler tours and sparse tables, we reduce the complexity of move evaluation from $O(n^3)$ to $O(n^2)$. We evaluate our approach on a dataset of DNA sequences from the fungal order *Russulales*. Our results indicate that SQL combined with efficient feature extraction offers a more robust and scalable approach for phylogenetic tree search.

2 BACKGROUND

This section introduces the core concepts from computational phylogenetics and reinforcement learning that are required to frame phylogenetic tree search as a sequential decision problem.

2.1 COMPUTATIONAL PHYLOGENETICS

Computational phylogenetics concerns the reconstruction of evolutionary relationships between biological sequences using algorithmic and statistical methods (Warnow, 2017). Given a set of homologous sequences, the goal is to infer a phylogenetic tree, a hypothesis that represents their shared evolutionary history. Phylogenetic trees are typically modeled as unrooted binary trees. In this representation, the leaves correspond to the observed taxa (e.g. DNA or protein sequences), while internal nodes represent hypothetical most recent common ancestors. The primary input to phylogenetic inference is a multiple sequence alignment (MSA), in which the sequences are aligned to identify regions of similarity, revealing conserved areas, mutations, and insertions/deletions across taxa.

Phylogenetic methods can broadly be divided into distance-based and character-based approaches (Zou et al., 2024). Character-based approaches model the evolutionary process at the level of individual alignment columns. Among these, maximum likelihood (ML) methods are widely regarded as the most accurate. In the ML framework, the likelihood of observing an MSA is modeled as a function of four components: the tree topology, branch lengths, evolutionary model parameters (e.g. substitution rates), and the alignment itself. The optimal phylogenetic tree is defined as the one that maximizes this likelihood.

While branch lengths and model parameters are continuous and can be optimized efficiently, the space of tree topologies is discrete and grows super-exponentially with the number of taxa. The computational search for the maximum-likelihood tree topology was previously shown to be NP-hard (Chor & Tuller, 2005). As a result, exhaustive search is infeasible for all but very small datasets, and practical methods rely on heuristic search strategies. These strategies explore tree space using local rearrangements such as subtree pruning and regrafting (SPR), navigating the likelihood landscape through a sequence of incremental topology modifications (Wooding, 2004).

2.2 DEEP REINFORCEMENT LEARNING

Reinforcement learning (RL) studies how an agent can learn to make sequential decisions through interaction with an environment (Sutton & Barto, 2018). Such environments are formalized as Markov decision processes (MDPs), defined by a tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ the set of actions, $p(s'|s, a)$ the state transition probability function for $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$, $r(s, a, s')$ a reward function, and $\gamma \in [0, 1]$ a discount factor. At each time step, the agent observes a state, selects an action, receives a reward, and transitions to a new state according to the environment dynamics.

The goal of RL is to learn the optimal policy π^*, a mapping from states to actions that maximizes the expected cumulative discounted rewards over time:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$$

A key assumption underlying MDPs is the Markov property, which states that the future transition dynamics of the environment depend only on the current state and action, and not on the full history of past states and actions (Sutton & Barto, 2018). Formally, the current state is assumed to be a sufficient summary of all relevant information from the past needed to predict future rewards and transitions.

Deep reinforcement learning extends classical RL by using deep neural networks as function approximators, enabling RL methods to scale to high-dimensional state and action spaces (Terven, 2025).

3 Related Work

RAxML-NG (Kozlov et al., 2019) is the successor of the widely used maximum likelihood-based phylogenetic inference tool RAxML. Starting from a set of initial trees, the algorithm iteratively evaluates SPR neighbors and performs hill-climbing by selecting the move that yields the highest immediate likelihood improvement, terminating once no improving neighbor can be found. This method represents the current standard for likelihood-based phylogenetic inference. While effective in practice, this greedy strategy runs the risk of converging to suboptimal local maxima. Furthermore, it requires calculating the approximate likelihood improvement of each SPR neighbor before acting, which becomes computationally expensive as the number or length of the sequences grows.

Prior to the use of reinforcement learning, machine learning techniques were explored to guide phylogenetic tree search at the level of individual topology modifications. In earlier work, the same authors of the paper we reproduced proposed a supervised learning approach based on random forests to predict the immediate log-likelihood improvement of candidate SPR moves using hand-crafted topological features, which proved effective for ranking moves (Azouri et al., 2021).

Beyond this line of work, only a limited number of studies have explored learning-based approaches to phylogenetic reconstruction. Lipták and Kiss proposed a reinforcement learning method that constructs unrooted phylogenetic trees from distance matrices using a neighbor joining formulation, framing tree reconstruction as a sequential decision process but without optimizing a likelihood objective (Lipták & Kiss, 2021). In contrast, other work has applied supervised neural networks to directly predict phylogenetic tree structures from distance matrices, without explicitly modeling tree search (Zhu & Cai, 2021).

4 Methodology

In this section, we describe our reinforcement learning framework for phylogenetic tree search. We re-implement the method of Azouri et al. (2024) as a baseline and introduce several modifications and extensions aimed at improving interpretability and performance. Unless stated otherwise, all components described below correspond to our implementation.

4.1 Environment

We formulate phylogenetic tree search as a Markov Decision Process (MDP) and implement it as a Gym-like environment.

A state $s \in \mathcal{S}$ is defined as $s = (M, T^*)$, where M denotes a multiple sequence alignment (MSA) and T^* an unrooted binary tree topology with branch lengths optimized under evolutionary model parameters fixed for each MSA. The action space consists of all valid subtree pruning and regrafting (SPR) moves applicable to the current tree, resulting in a fixed-size discrete action space. State transitions are deterministic: applying an SPR move yields a new topology, after which branch lengths are re-optimized. The reward is defined as the change in log likelihood between consecutive states,

$$r_t = \mathrm{LL}(s_{t+1}) - \mathrm{LL}(s_t),$$

where $\mathrm{LL}(\cdot)$ denotes the log likelihood of the MSA given the tree.

An environment instance may contain one or more MSAs with an equal number of sequences. For a fixed number of sequences, all MSAs induce the same discrete tree topology search space, while differing in their associated likelihood landscapes. At the beginning of each episode, a single MSA and an associated starting tree are sampled uniformly at random. Episodes terminate after a fixed number of environment steps.

Likelihood evaluation and branch length optimization are performed using RAxML-NG (Kozlov et al., 2019), which is treated as a black-box likelihood oracle. For each MSA, evolutionary model parameters are optimized once and then fixed for the duration of environment interaction. During training and evaluation, applying an action corresponds to performing an SPR move followed by likelihood evaluation with branch length optimization under the fixed model parameters.

RAxML-NG is additionally used offline to obtain reference optima for evaluation starting trees. Further implementation details and performance optimizations are provided in Appendix A.

4.2 State and Action Representations

Reinforcement learning agents cannot operate directly on tree topologies and therefore require a suitable representation of state–action pairs. In this work, we consider two classes of representations: hand-crafted feature vectors and learned graph-based encodings.

Feature-based representation. Following the original paper (Azouri et al., 2024), each candidate SPR move a in state s is represented by a fixed-dimensional feature vector $\phi(s, a)$. These features are derived from global properties of the current tree and from local structural statistics of the four subtrees induced by the SPR move. The feature representation was originally designed to predict the immediate log-likelihood improvement of a single SPR move using a random forest model (Azouri et al., 2021). A complete list of features, together with an illustration showing the four subtrees induced by SPR moves are provided in Appendix B. While we adopt the same feature definitions, we implement feature extraction more efficiently. Implementation details on the feature extraction is provided in Appendix C.

Extended feature set. While the underlying MDP state (M, T^*) is Markov, the feature representation $\phi(s, a)$ provides only a partial observation of this state. In particular, the original features focus on local move-specific structure and do not explicitly encode global search context or progress through the optimization landscape. This can result in distinct states being mapped to similar feature vectors, potentially violating the Markov property required for accurate value estimation. To address this limitation, we extend the original feature set with additional features, including new global tree statistics, normalized variants of existing features, and features encoding the current phase of the search. These extensions are intended to provide a more informative approximation of the underlying Markov state. The full extended feature set is described in Appendix D.

Graph-based representation. As an alternative to hand-crafted features, we investigate a graph-based representation using a Graph Attention Network v2 (GATv2) Brody et al. (2022). This architecture was selected specifically for its ability to incorporate edge attributes into the attention mechanism. The message-passing process follows the general framework described by Gilmer et al. (2017), where node and edge information are updated iteratively to produce a global embedding of the tree topology. This allows the agent to learn structural dependencies directly from the data without relying on domain-specific feature engineering, at the cost of increased computational complexity. Action representations are constructed by combining local embeddings of the graph components involved in an SPR move. Architectural details are provided in Appendix E.

4.3 Reinforcement Learning Algorithms

We investigate two value-based deep reinforcement learning algorithms that learn an action-value function $Q(s, a)$ over state–action pairs. In our implementation, the Q-network operates on the feature representation $\phi(s, a)$ described in Section 4.2, and actions during training are sampled from a Boltzmann policy:

$$\pi(a \mid s) = \frac{\exp(Q(s, a)/\alpha)}{\sum_{a'} \exp(Q(s, a')/\alpha)},$$

where α controls the exploration–exploitation trade-off.

Deep Q-Network (DQN). To mirror the original paper, we implement Deep Q-Network (Mnih et al., 2015), which learns an approximation of the optimal action-value function by minimizing the temporal-difference error. Our DQN implementation additionally incorporates Prioritized Experience Replay and Double Q-Learning (Hessel et al., 2017). While DQN learns off-policy, its update rule relies on a hard maximization over actions, which can lead to instability.

Soft Q-Learning. To address this limitation, we also consider Soft Q-Learning (Haarnoja et al., 2017), which augments the standard objective with an entropy term and replaces the hard maximization with a soft alternative. By encouraging stochastic policies that assign probability mass to multiple high-value actions, Soft Q-Learning can provide more robust exploration and improved convergence behavior.

The Bellman targets used by the two algorithms are given by:

$$y_t^{\text{DQN}} = r_t + \gamma \max_{a'} Q_{\theta'}(s_{t+1}, a'), \qquad y_t^{\text{SQL}} = r_t + \gamma\, \alpha \log \sum_{a'} \exp\left(\frac{Q_{\theta'}(s_{t+1}, a')}{\alpha}\right),$$

where θ' denotes the parameters of a target network.

In our implementation of Soft Q-Learning, the temperature parameter α is learned automatically to maintain a target policy entropy, which is set as a fraction of the entropy corresponding to the uniform distribution over the discrete action space.

4.4 Evaluation

We evaluate trained agents by executing deterministic evaluation episodes and comparing their performance against RAxML-NG baselines obtained from the same starting trees.

Evaluation Protocol Evaluation episodes are run for all MSAs and their associated evaluation starting trees. At each step, actions are selected greedily with respect to the learned Q-values, unless stated otherwise. The episode horizon matches the horizon used during training. To prevent cyclic behavior, SPR moves that would result in revisiting a tree topology already encountered within the same episode are disallowed.

In addition to greedy evaluation, we consider a top-k evaluation strategy that bridges value-based policies and classical hill-climbing. At each step, the k actions with the highest Q-values are selected, their true likelihood improvements are evaluated, and the action yielding the largest immediate improvement is applied. This increases the number of likelihood evaluations from $\mathcal{O}(H)$ to $\mathcal{O}(kH)$ per episode, where H denotes the episode horizon.

Performance Metrics The original paper evaluates agents using a normalized improvement score relative to the RAxML-NG optimum. We do not adopt this metric, as it depends strongly on the likelihood landscape induced by the starting tree and does not directly measure convergence to the reference solution.

Instead, we report the success rate, defined as the fraction of evaluation starting trees for which the best tree encountered during the episode has a log-likelihood within $\varepsilon = 0.01$ of the corresponding RAxML-NG optimum.

5 Experimental Setup

This section describes the experimental configuration and outlines the experiments conducted to evaluate the proposed methods. Unless stated otherwise, complete hyperparameter configurations are reported in the Appendix F.

5.1 Problem Configuration

All experiments are conducted on phylogenetic inference problems with MSAs containing $n = 9$ sequences. For this setting, the discrete tree space consists of $(2n-5)!! = 135{,}135$ unrooted binary tree topologies, and each state admits a discrete action space of $4(n-3)(n-2) = 168$ valid SPR moves.

To construct training and evaluation environments, we generate MSAs by uniformly sampling sequences from a large reference dataset containing 856 ITS barcode DNA sequences from the order *Russulales* (fungi), obtained from the UNITE database (Abarenkov et al., 2023), and align them using MAFFT (Katoh et al., 2002). Compared to the original paper, which trains on a set of emprical MSAs from diverse genes and organisms, this approach reduces biological heterogeneity, and allows us to easily control both the number of MSAs to use in an environment, together with the number of sequences in each MSA.

For each MSA, a fixed set of starting trees is generated and partitioned into 100 training starting trees and 20 evaluation starting trees. RAxML-NG baselines are computed only for the evaluation starting trees.

5.2 Experiments

We conduct a sequence of experiments designed to study algorithmic choices, representation quality, and generalization behavior. For each experimental setting, five independent agents are trained, and all reported results are averaged across these runs.

Experiment 1: Algorithmic comparison. Using a single-MSA environment, we compare DQN and Soft Q-Learning using the hyperparameters reported in the original paper. Training is extended to at least 30,000 episodes to assess long-term stability.

Experiment 2: Generalization across MSAs. We train Soft Q-Learning agents in an environment containing 100 MSAs and evaluate them on a separate validation environment with 100 different MSAs. To study the robustness on unseen MSAs, we investigate the introduction of a dropout layer and the use of the AdamW optimizer.

Experiment 3: Discount factor and feature representation. We study the effect of temporal credit assignment by training Soft Q-Learning agents with discount factors $\gamma \in \{0.99, 0.9, 0.5, 0.0\}$. These experiments are performed using both the original and extended feature representations to assess their interaction with bootstrapping.

Experiment 4: Graph-based representation. Finally, we assess the performance of the graph neural network–based representation individually under the same evaluation protocol. Due to increased computational and architectural complexity, we only perform a select few runs, serving as proof of concept. We make logical inferences on hyperparameter choice and algorithm design where sensible.

6 Results

This section reports the empirical performance of the proposed methods under the experimental settings described in Section 5.

6.1 Algorithmic Comparison

Figure 1 shows the success rate achieved by DQN and Soft Q-Learning during training in a single-MSA environment. We observe that both configurations of Soft Q-Learning clearly outperform DQN, with and without the stability extensions. Not only does Soft Q-Learning achieve a higher success rate, it also shows a stark decrease in the variance between the individual repetitions. This indicates that the soft Bellman backup achieves better performance on our environment, and that the combination of Prioritized Experience Replay and Double Q-Learning was not able to counteract the inherent instability caused by off-policy hard updates.

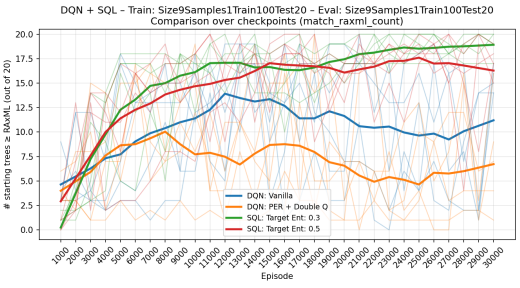


Figure 1: Success rate comparison for Experiment 1 in a single-MSA environment. We observe Soft Q-Learning outperforming DQN with and without stability extensions.

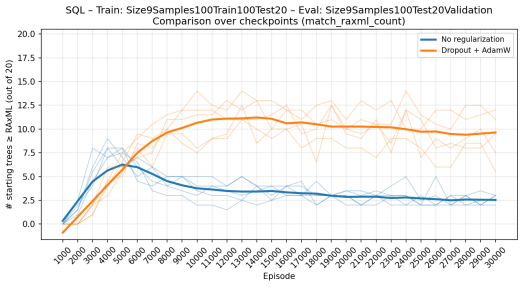
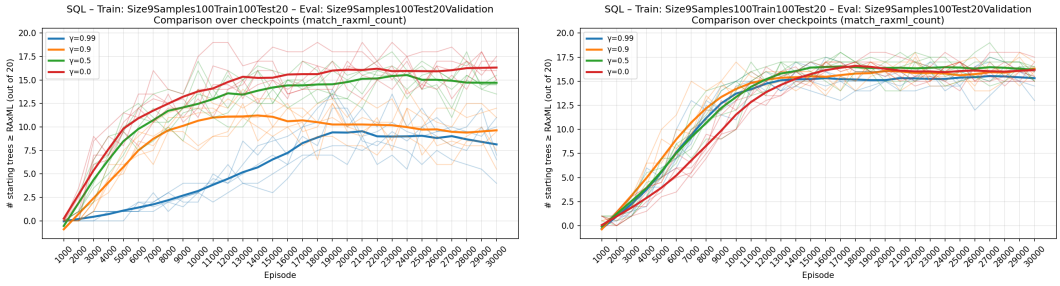


Figure 2: Success rate comparison for Experiment 2, with Soft Q-Learning trained and evaluated on separate environments of 100 MSAs. We observe that network regularization has a substantial effect on generalization across unseen MSAs..

6.2 Generalization Across MSAs

Continuing with only Soft Q-learning, figure 2 reports the success rate on a validation set of 100 sampled MSAs, for agents trained on 100 different MSAs, evaluating generalization beyond the training likelihood landscapes. We observe that in the default configuration, performance peaks early in training to a success rate of only 6, dropping afterwards. Introducing the network regularization techniques however appears to double the success rate at its peak. This increase in performance can be attributed to the dropout layer forcing the Q-network to become more robust, while also increasing explorations due to the noisy prediction early on in training.



(a) Original feature set

(b) Extended feature set

Figure 3: Success rate for Experiment 3 as a function of the discount factor γ. Agents are trained using Soft Q-Learning with discount factors $\gamma \in \{0.99, 0.9, 0.5, 0.0\}$ and evaluated using either the original or the extended feature representation.

6.3 Discount Factor and Feature Representation

Figure 3 shows the success rate obtained for different discount factors γ using the original and extended feature representations, using the same training and evaluation environment used in 6.2. The results using the original feature set show a monotonic decrease in performance as γ increases. This is a strong indication that the features do not capture enough context to be able to predict future rewards, violating the required Markov property and therefore resulting in bootstrapping only introducing noise. This result is not surprising given that the feature set was originally designed to only predict immediate likelihood improvement of SPR moves.

After augmenting the representation with the new features, future rewards appear to have become predictable. All configurations appear to converge to the same success rate of 16, with only $\gamma = 0.99$ performing slightly worse. We notice that $\gamma > 0$ learns faster than $\gamma = 0.0$ now, due to the successful backward propagation of values, making it so that actions that set up future improvements are recognized earlier. The final noteworthy observation is that final success rates never exceed $\gamma = 0.0$, indicating that a shortsighted policy is still globally sufficient to reach good optima.

For the best-performing configuration (Soft Q-Learning with extended features, dropout, AdamW, and $\gamma = 0.9$), we additionally compare greedy action selection to a top-5 evaluation strategy. The results in figure 4 show that the top-5 evaluation approach is effective enough to achieve a perfect success rate, showing that besides the algorithmic choices made for training, the evaluation method of the derived Q-network is just as important for assessing its practical usability in phylogenetic tree search.

6.4 Graph-Based Representation

Finally, for Experiment 4, we present two distinct results. Figure 5 shows the average success rate of all agents in our first run, using graph embeddings computed by a graph neural network to replace the hand-crafted feature representation. We start by copying all experimental design choices not related to the graph embeddings from Experiment 2, as a baseline. There are also additional GNN-specific hyperparameters. The trained agents clearly display learning behavior, reaching a maximum success rate of 9 just before the end of training, with higher performance not out of the question given a longer training duration.

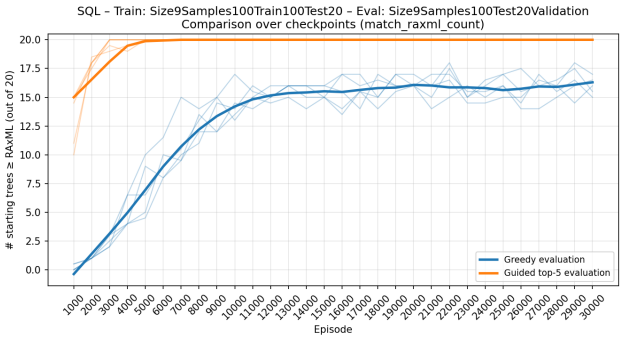


Figure 4: Comparison of evaluation strategies for the best-performing configuration (Soft Q-Learning with extended features, dropout, AdamW, and $\gamma = 0.9$). The success rate is shown for greedy evaluation, where the action with the highest predicted Q-value is selected at each step, and for top-5 evaluation, where the action with the highest immediate log-likelihood improvement among the five highest-valued actions is chosen.

Figure 6 shows the average success rate of an additional training run using an experimental setup. It incorporates a decaying Target Entropy, as well as a Replay β annealing schedule that lasts longer than the entropy decay. Furthermore, the training continues to optimize the agents' policies even after both the Target Entropy and Replay β have reached their final values. It must also be noted that for this run a different, larger sample pool was used, making the results less comparable. The exact hyperparameter overrides are listed in the Appendix F, all other values are taken from Experiment 2. Learning is slow for the first 13,000 episodes, and even stagnates for a while, hinting at a too high starting value for the target entropy. However, the agents do reach an average success rate of 10 (again with uncertainty whether more episodes would be beneficial), outperforming the previous setup.

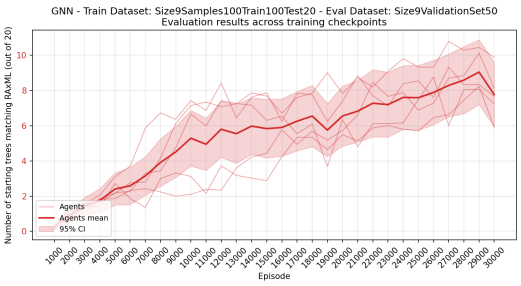


Figure 5: Success rate for Experiment 4 with identical setup to Experiment 2, but with graph embeddings successfully replacing the hand-crafted feature representation.

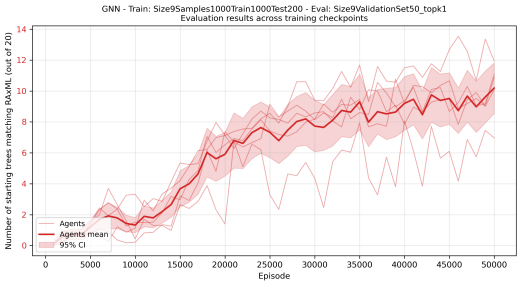


Figure 6: Success rate for Experiment 4 using an alternative setup to observe differences in learning behavior.

7 Discussion and Conclusion

In this work, we reproduced and extended a reinforcement learning approach to maximum likelihood phylogenetic tree search. Across all experiments, Soft Q-Learning consistently outperformed DQN, exhibiting both higher success rates and substantially lower variance across independent runs. Moreover, the introduction of network regularization through dropout and weight decay had a pronounced effect on generalization across unseen MSAs, albeit at the cost of longer training times. Importantly, these longer training regimes were made feasible by a more efficient feature extraction pipeline, which reduced the computational complexity of evaluating all candidate SPR moves from cubic to quadratic time.

The learned Q-network successfully enables a reduction in the number of likelihood evaluations performed during tree search. Classical SPR-based heuristics evaluate the likelihood of all valid SPR moves at each step, resulting in a quadratic number of likelihood computations in the number of sequences per step. When using a learned Q-function, likelihood evaluation can instead be restricted to a small subset of candidate moves selected based on their predicted Q-values. In the extreme case, only the highest-valued action is evaluated, while a more conservative top-k strategy evaluates the likelihood of only the k most promising moves. This filtering approach is conceptually similar to the radius-restricted SPR strategy employed by RAxML-NG and, assuming Q-value prediction is cheaper than likelihood computation, can either accelerate the search process or allow greater exploration under a fixed computational budget.

Our experiments further demonstrate that, when using the original feature representation, performance deteriorates as the discount factor increases, indicating that future rewards are not reliably predictable from the feature vectors alone. This observation raises the question of whether the gains reported in the original paper can be fully attributed to reinforcement learning, given the absence of experiments exploring the role of temporal credit assignment. The fact that a discount factor of $\gamma = 0.0$ achieves comparable final performance after extending the feature set reinforces the well-established effectiveness of greedy hill-climbing strategies in phylogenetic tree search. Furthermore, despite results in the original paper suggesting that multi-step strategies may yield more efficient search trajectories, we do not observe evidence that the learned Q-functions consistently exploit such strategies in practice.

The graph-based representation offers an alternative to hand-crafted features and demonstrates that learning directly from tree topology is possible. Due to the low rate at which experiments using a GNN could be conducted, there is still large ambiguity over which specific design elements have the greatest impact on performance. It stands to reason that, particularly when scaled up to MSAs with more sequences, graph based data could prove more informative than hand-crafted heuristic values. The foundations for this were laid in this project. However, with current GNN implementations running on CPUs, they are significantly less efficient to compute than simple scalar heuristics, which, at least for now, prove to be more effective at our current experimental scale.

Several directions for future work emerge from this study. A key next step is scaling the framework to MSAs with larger numbers of sequences, as well as evaluating zero-shot generalization to MSAs with different numbers of sequences, which would provide insight into the robustness of learned policies. Additionally, further graph-based experiments should be pursued, particularly combining our temporal features from Experiment 3 with the learned graph embedding, as well as attempting to reduce computational complexity. Finally, integrating the learned Q-network directly into RAxML-NG as a search heuristic would enable a direct comparison of runtime and inference quality against existing approaches, offering a more definitive assessment of the practical benefits of guiding phylogenetic tree search using reinforcement learning.

In conclusion, our results suggest that while reinforcement learning can successfully guide phylogenetic tree search, its effectiveness critically depends on representation quality, regularization, and practical implementation choices.

8 Code availability

The complete source code, including datasets, environment integration and evaluation scripts, is publicly available at `https://github.com/Dzee97/SADRL_PhyloRL`. Although we take inspiration from the work of Azouri et al. (2024), all provided code is our own independent self-implementation and scientific contribution.

REFERENCES

Shiran Abadi, Dana Azouri, Tal Pupko, and Itay Mayrose. Model selection may not be a mandatory step for phylogeny reconstruction. *Nature Communications*, 10, 02 2019. doi: 10.1038/s41467-019-08822-w.

Kessy Abarenkov, R Henrik Nilsson, Karl-Henrik Larsson, Andy F S Taylor, Tom W May, Tobias Guldberg Frøslev, Julia Pawlowska, Björn Lindahl, Kadri Põldmaa, Camille Truong, Duong Vu, Tsuyoshi Hosoya, Tuula Niskanen, Timo Piirmann, Filipp Ivanov, Allan Zirk, Marko Peterson, Tanya E Cheeke, Yui Ishigami, Arnold Tobias Jansson, Thomas Stjernegaard Jeppesen, Erik Kristiansson, Vladimir Mikryukov, Joseph T Miller, Ryoko Oono, Francisco J Ossandon, Joana Paupério, Irja Saar, Dmitry Schigel, Ave Suija, Leho Tedersoo, and Urmas Kõljalg. The unite database for molecular identification and taxonomic communication of fungi and other eukaryotes: sequences, taxa and classifications reconsidered. *Nucleic Acids Research*, 52(D1):D791–D797, 11 2023. ISSN 0305-1048. doi: 10.1093/nar/gkad1039. URL https://doi.org/10.1093/nar/gkad1039.

Dana Azouri, Shiran Abadi, Yishay Mansour, Itay Mayrose, and Tal Pupko. Harnessing machine learning to guide phylogenetic-tree search algorithms. *Nature Communications*, 12, 03 2021. doi: 10.1038/s41467-021-22073-8.

Dana Azouri, Oz Granit, Michael Alburquerque, Yishay Mansour, Tal Pupko, and Itay Mayrose. The tree reconstruction game: Phylogenetic reconstruction using reinforcement learning. *Molecular Biology and Evolution*, 41(6):msae105, 06 2024. ISSN 1537-1719. doi: 10.1093/molbev/msae105. URL https://doi.org/10.1093/molbev/msae105.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks?, 2022. URL https://arxiv.org/abs/2105.14491.

Benny Chor and Tamir Tuller. Maximum likelihood of evolutionary trees: Hardness and approximation. *Bioinformatics (Oxford, England)*, 21 Suppl 1:i97–106, 07 2005. doi: 10.1093/bioinformatics/bti1027.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML)*, 2017.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies, 2017. URL https://arxiv.org/abs/1702.08165.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning, 2017. URL https://arxiv.org/abs/1710.02298.

Kazutaka Katoh, Kazuharu Misawa, Kei-ichi Kuma, and Takashi Miyata. Mafft: a novel method for rapid multiple sequence alignment based on fast fourier transform. *Nucleic Acids Research*, 30 (14):3059–3066, 07 2002. ISSN 0305-1048. doi: 10.1093/nar/gkf436. URL https://doi.org/10.1093/nar/gkf436.

Alexey M Kozlov, Diego Darriba, Tomáš Flouri, Benoit Morel, and Alexandros Stamatakis. Raxml-ng: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference. *Bioinformatics*, 35(21):4453–4455, 05 2019. ISSN 1367-4803. doi: 10.1093/bioinformatics/btz305. URL https://doi.org/10.1093/bioinformatics/btz305.

Panna Lipták and Attila Kiss. Constructing unrooted phylogenetic trees with reinforcement learning. *Stud. Univ. Babeș-Bolyai Inform.*, 66(1), 2021.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Kirkeby Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015. URL https://api.semanticscholar.org/CorpusID:205242740.

Vincent Moulton, Andreas Spillner, and Taoyang Wu. Upgma and the normalized equidistant minimum evolution problem, 2017. URL https://arxiv.org/abs/1704.00497.

Naruya Saitou and Masatoshi Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4 4:406–25, 1987. URL https://api.semanticscholar.org/CorpusID:12287470.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.

Juan Terven. Deep reinforcement learning: A chronological overview and methods. *AI*, 6(3), 2025. ISSN 2673-2688. doi: 10.3390/ai6030046. URL https://www.mdpi.com/2673-2688/6/3/46.

Tandy Warnow. *Computational Phylogenetics: An Introduction to Designing Methods for Phylogeny Estimation*. Cambridge University Press, 2017.

Stephen Wooding. Inferring phylogenies. *The American Journal of Human Genetics*, 74, 05 2004. doi: 10.1086/383584.

Tujin Zhu and Yunpeng Cai. Applying neural network to reconstruction of phylogenetic tree. pp. 146–152, 02 2021. doi: 10.1145/3457682.3457704.

Yue Zou, Zixuan Zhang, Yujie Zeng, Hanyue Hu, Youjin Hao, Sheng Huang, and Bo Li. Common methods for phylogenetic tree construction and their implementation in r. *Bioengineering*, 11(5), 2024. ISSN 2306-5354. doi: 10.3390/bioengineering11050480. URL https://www.mdpi.com/2306-5354/11/5/480.

A Environment Details

This appendix provides details related to the environment implementation.

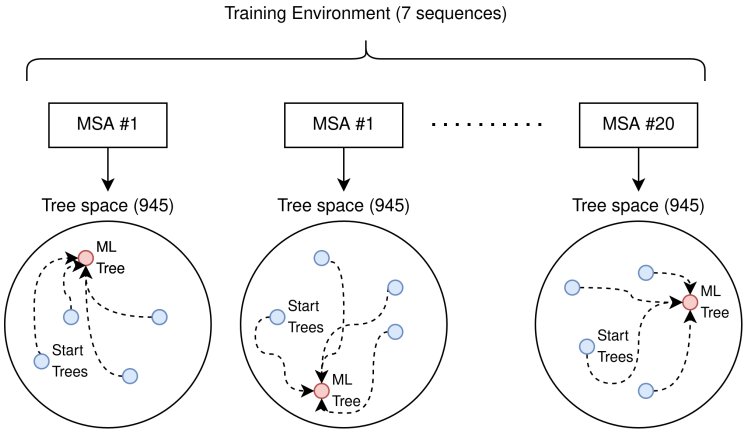


Figure 7: An Illustration of an environment instance consisting of 20 MSAs, each with 7 sequences. As the number of sequences are equal, each MSA induces the same tree space, albeit with a different likelihood landscape.

A.1 RAxML-NG Integration

Likelihood evaluation and branch length optimization within the environment are performed using the RAxML-NG binary (Kozlov et al., 2019). Rather than re-implementing phylogenetic likelihood computations, RAxML-NG is treated as a black-box backend that provides likelihood values and optimized branch lengths for a given MSA and tree topology.

For each MSA, RAxML-NG is used in a multi-stage preprocessing pipeline to generate starting trees, determine fixed evolutionary model parameters, and establish baseline optimization results. First, the `--start` command is used to generate two sets of starting trees, one for training and one for evaluation, and 10 parsimony starting trees. The parsimony trees are then evaluated using the `--evaluate` command, during which evolutionary model parameters are optimized for the GTR+I+G model (Abadi et al., 2019). The model parameters corresponding to the highest-likelihood parsimony tree are selected and fixed for that MSA for the remainder of the environment interaction.

Using these fixed model parameters, the `--search` command is executed for each of the evaluation starting trees. The resulting likelihood values define reference optima used by the evaluation metrics described in Section 5.4.

During environment interaction, taking an action performs an SPR move on the current tree topology T_t^*. After every action, the resulting topology T_{t+1} is evaluated using the `--evaluate` command with the fixed evolutionary model parameters, optimizing only branch lengths, resulting in the new log likelihood $\mathrm{LL}(s_{t+1})$ used for the reward signal, together with new optimized tree topology T_{t+1}^*.

A.2 Performance Optimizations

To reduce computational overhead during environment interaction, we implement a caching mechanism for likelihood evaluations. Tree topologies are identified by hashing their Newick representations with branch lengths removed. For each MSA, the cache stores optimized branch lengths and the corresponding log likelihood for previously encountered topologies. When a topology is revisited, cached results are reused, avoiding redundant calls to RAxML-NG.

In addition, all intermediate RAxML-NG output files are written to a temporary RAM-based filesystem, minimizing disk I/O during training and evaluation, significantly improves runtime performance.

B ORIGINAL FEATURE REPRESENTATION

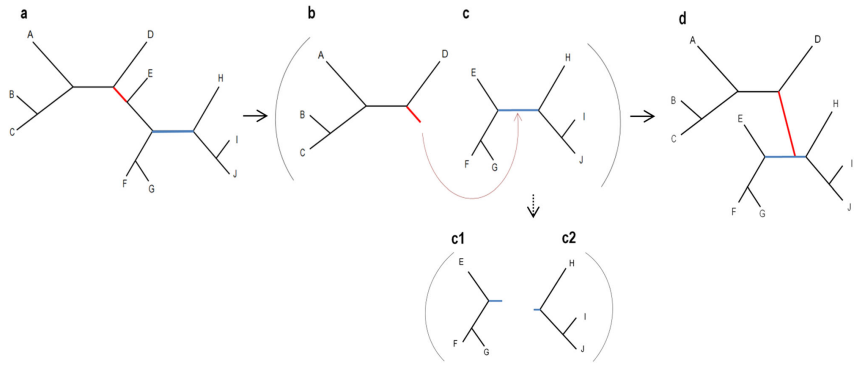


Figure 8: The trees defined by an SPR move. For feature calculations, two trees and four subtrees were considered: **a** an example of a starting tree; **b**, **c** the two subtrees induced by the pruned branch of a tree *a* (in red), where *b* is the pruned subtree and *c* the remaining subtree; **c1**, **c2** the regrafted branch (in blue) also induces two subtrees, from both sides of the regrafted branch of subtree *c*; **d** the resulting tree following the SPR move (Azouri et al., 2021).

Table 1 lists the features used in the original feature-based representation.

Feature	Feature name	Description
1	Total branch lengths	The sum of branch lengths in the starting tree
2	Longest branch	The length of the longest branch in the starting tree
3-4	Branch length	The length of the branch that was being pruned or regrafted
5	Topology distance from the pruned node	The number of branches in the path between the regrafting and the pruning branches, not including these branches
6	Branch length distance from the pruned node	The sum of branches in the path between the regrafting and the pruning branches, not including these branches
7	New branch length	The approximated length of the new branch formed due to pruning
8-11	Number of species	The number of leaves in the four subtrees
12-15	Total branch lengths	The sum of branch lengths in the four subtrees
16-19	Longest branch	The length of the longest branch in the four subtrees
20-23	Bootstrap support (UPGMA)	The approximated bootstrap support of the Pruning internal branch (that defined a split) that was being pruned or regrafted, calculated on both the starting tree and the resulting tree following the SPR move
24-27	Bootstrap support (NJ)	Same procedure, but using NJ (Saitou & Nei, 1987) bootstrap trees instead of UPGMA (Moulton et al., 2017)

Table 1: Original feature set used to represent candidate SPR moves (Azouri et al., 2024).

C Efficient Feature Extraction for SPR Moves

In the original implementation, feature extraction is performed independently for each candidate SPR move. Since each move induces a different partitioning of the tree, this approach requires repeated traversals of the tree to compute subtree-specific statistics, resulting in significant computational overhead when evaluating all possible moves in a state.

To reduce this cost, we implement a preprocessing-based approach that allows features for all SPR moves to be computed in constant time from a single traversal of the starting tree.

C.1 Tree Preprocessing

For each tree topology, we perform a single depth-first traversal to compute and cache structural metadata for every node. During this traversal, we record:

- subtree sizes (number of leaves)
- total branch length per subtree
- maximum branch length per subtree

In addition, we construct an Euler tour representation of the tree and build a sparse table for range minimum queries (RMQ), enabling constant-time lowest common ancestor (LCA) queries between any pair of nodes. This allows distances between prune and regraft locations to be computed efficiently.

C.2 Feature Assembly for SPR Moves

Each SPR move can be characterized by a prune edge, a regraft edge, and their lowest common ancestor. Using the precomputed metadata, all features required by the original representation can be derived by combining cached values associated with the affected nodes. As a result, feature vectors for candidate SPR moves can be assembled in constant time per move, without requiring additional tree traversals.

C.3 Computational Complexity

For a phylogenetic tree with n leaves, a full tree traversal requires $\mathcal{O}(n)$ time. The number of valid SPR moves for such a tree scales as $4(n-3)(n-2) = \mathcal{O}(n^2)$.

In the original approach, feature extraction is performed independently for each SPR move, requiring a separate tree traversal per move. This results in an overall time complexity of $\mathcal{O}(n) \times \mathcal{O}(n^2) = \mathcal{O}(n^3)$ per environment step.

In contrast, our approach performs a single preprocessing traversal of the starting tree in $\mathcal{O}(n)$ time. Feature vectors for all SPR moves are then assembled in constant time per move, yielding a total complexity of $\mathcal{O}(n) + \mathcal{O}(n^2) = \mathcal{O}(n^2)$

This reduction from cubic to quadratic time complexity significantly lowers the computational cost of feature extraction and enables longer training horizons and larger experimental settings.

D EXTENDED FEATURE SET

Table 2 lists the additional features introduced to extend the original feature-based representation. These features provide global context, normalized quantities, and information about the current phase of the search, and are intended to yield a more informative approximation of the underlying Markov state.

Feature	Feature name	Description
101	Mean branch length	The average branch length in the starting tree
102	Normalized longest branch	The length of the longest branch normalized by the mean branch length
103	Branch length diameter	The maximum branch-length distance between any pair of leaves in the tree
104	Topology diameter	The maximum number of branches between any pair of leaves in the tree
105	Normalized topology diameter	The topology diameter normalized by the maximum possible diameter for the given number of species
106–107	Normalized prune and regraft branch lengths	The prune and regraft branch lengths normalized by the mean branch length
108	Normalized topology distance	The topology distance between prune and regraft locations normalized by the topology diameter
109	Normalized branch-length distance	The branch-length distance between prune and regraft locations normalized by the branch-length diameter
110	Normalized new branch length	The approximated new branch length normalized by the mean branch length
111–114	Normalized number of species	The number of leaves in each of the four subtrees, normalized by the total number of species
115–118	Normalized subtree branch lengths	The sum of branch lengths in each of the four subtrees, normalized by the total branch length
119–122	Normalized longest subtree branch	The longest branch length in each of the four subtrees, normalized by the mean branch length
123	Log-likelihood distance to starting tree	The difference in log likelihood between the current tree and the initial starting tree
124	Log-likelihood distance to parsimony tree	The difference in log likelihood between the current tree and the corresponding parsimony starting tree
125	Step number	The current step index within the episode
126	Recent mean reward	The mean of the rewards obtained during the previous three steps, computed as $\tanh(r/10)$ per step

Table 2: Extended feature set used to augment the original SPR move representation.

E GRAPH-BASED STATE–ACTION REPRESENTATION

This appendix describes the graph neural network (GNN) based state–action representation used as an alternative to hand-crafted feature vectors.

E.1 TREE ENCODING

Phylogenetic trees are modeled as graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where nodes correspond to tree nodes and edges correspond to branches. Although phylogenetic trees are inherently unrooted, standard GNN layers operate on directed graphs. We therefore represent each tree as a bidirectional graph: for every branch connecting nodes u and v with branch length l, we introduce two directed edges (u, v) and (v, u) with identical edge attributes.

To avoid injecting domain-specific heuristics, we use minimal atomic features. Node features consist of a single binary indicator denoting whether a node is a leaf or an internal node. Edge features consist of a single continuous value representing the branch length.

E.2 GNN ARCHITECTURE

Tree structure is encoded using a Graph Attention Network v2 (GATv2) (Brody et al., 2022), which explicitly incorporates edge features into the message-passing process. The architecture begins with a linear projection of node features to a hidden dimension d_{model}, followed by three GATv2 convolutional layers with multi-head attention (four heads per layer). Residual connections are applied after each layer.

To obtain a global embedding of the tree, final node embeddings are aggregated using a concatenation of sum, mean, and max pooling:

$$H_{\text{tree}} = \left[\sum_{v \in \mathcal{V}} h_v \;\Big\|\; \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} h_v \;\Big\|\; \max_{v \in \mathcal{V}} h_v \right].$$

This aggregation captures global scale, average structure, and extreme values simultaneously.

E.3 STATE–ACTION FUSION

Each valid SPR move is defined by a prune edge $(u_{\text{in}}, u_{\text{out}})$ and a regraft edge $(v_{\text{in}}, v_{\text{out}})$. The corresponding action representation is constructed by concatenating the learned node embeddings $h_{u_{\text{in}}}, h_{u_{\text{out}}}, h_{v_{\text{in}}}, h_{v_{\text{out}}}$ with three scalar metadata features: the topological distance between prune and regraft locations, and the branch lengths of the prune and regraft edges.

This vector is processed by an action encoder multilayer perceptron to produce an action embedding H_{action}. The Q-value is then computed by a head network operating on the concatenation of the global tree embedding and the action embedding:

$$Q(s, a) = \text{MLP}_{\text{head}}([H_{\text{tree}} \,\|\, H_{\text{action}}]).$$

F HYPERPARAMETERS

This appendix lists default hyperparameters used across experiments and specifies experiment-specific deviations. Unless stated otherwise, all runs use the defaults in Tables 3–5.

F.1 DEFAULT HYPERPARAMETERS

COMMON (SHARED ACROSS AGENTS)

Parameter	Default	Description
Training episodes	30,000	Number of training episodes
Episode horizon H	20	Maximum environment steps per episode
Feature set	Original	Original or extended feature representation
Checkpoint frequency	1000	Save model checkpoint every N episodes
Update frequency	1	Gradient updates per environment step
Batch size	128	Mini-batch size
Hidden dimension	256	Q-network hidden layer width
Network architecture	3-layer MLP	Fully connected network with ReLU activations
Dropout probability	0.0	Dropout after first layer
Replay buffer size	10,000	Maximum stored transitions
Warm-up steps	1000	Minimum transitions before training begins
Learning rate	1×10^{-5}	Optimizer learning rate
Optimizer	Adam	AdamW used when weight decay is enabled
Discount factor γ	0.9	Reward discounting
Target update rate τ	0.005	Polyak averaging coefficient

Table 3: Default training hyperparameters shared across all agents.

DQN-SPECIFIC DEFAULTS

Parameter	Default	Description
Boltzmann temperature α	1.0	Temperature used for action sampling
Double Q-learning	False	If enabled, decouples action selection and evaluation
Prioritized replay	Disabled	Enabled when replay priority exponent > 0

Table 4: Default hyperparameters specific to DQN.

SOFT Q-LEARNING-SPECIFIC DEFAULTS

Parameter	Default	Description
Initial temperature α_0	4.0	Initial value for the learned temperature
Entropy schedule frames	400,000	Steps over which entropy target is scheduled
Target entropy start	0.3	Fraction of uniform-policy entropy at start
Target entropy end	0.3	Fraction of uniform-policy entropy at end

Table 5: Default hyperparameters specific to Soft Q-Learning.

F.2 Experiment-Specific Overrides

For each experiment, Table 6–10 list only hyperparameters that differ from the defaults above.

Experiment 1: Algorithmic Comparison

Parameter	Value	Note
Double Q-learning (DQN)	Enabled	Decoupled action selection and evaluation
Replay priority exponent	0.6	Enables Prioritized Experience Replay
Replay β start	0.4	Importance sampling correction
Replay β frames	400,000	Linear annealing schedule
Entropy target (SQL)	$\{0.3, 0.5\}$	Compared values

Table 6: Hyperparameter overrides for Experiment 1.

Experiment 2: Generalization Across MSAs

Parameter	Value	Note
Dropout probability	0.2	Applied after first hidden layer
Optimizer	AdamW	Weight decay $= 0.01$

Table 7: Hyperparameter overrides for Experiment 2.

Experiment 3: Discount Factor and Feature Set Analysis

Parameter	Value	Note
Dropout probability	0.2	Applied after first hidden layer
Optimizer	AdamW	Weight decay $= 0.01$
Discount factor γ	$\{0.99, 0.9, 0.5, 0.0\}$	Compared values
Feature set	Original / Extended	Representation comparison

Table 8: Hyperparameter overrides for Experiment 3.

EXPERIMENT 4: GRAPH-BASED REPRESENTATION

Parameter	Value	Note
GAT layers	4	Number of message passing layers
Attention heads	8	Number of attention heads
Action layers	3	Adequate model complexity
Q layers	4	Adequate model complexity

Table 9: Default hyperparameters specific to Graph Neural Networks

Parameter	Value	Note
Training episodes	50,000	Number of training episodes
Hidden dimension	512	Q-network hidden layer width
Action layers	3	Adequate model complexity
Q layers	4	Adequate model complexity
Replay β frames	800,000	Linear annealing schedule
Entropy schedule frames	600,000	Steps over which entropy target is scheduled
Target entropy start	0.9	Fraction of uniform-policy entropy at start
Target entropy end	0.07	Fraction of uniform-policy entropy at end
Sample pool size	1000	Number of samples trained on

Table 10: Hyperparameters specific to second, experimental GNN experiment