ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΗΣ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΠΑΡΑΓΩΓΗΣ
http://www.eee.uniwa.gr
http://www.idpe.uniwa.gr
Θηβών 250, Αθήνα-Αιγάλεω 12241
Τηλ: +30 210 538-1614
Διατμηματικό Πρόγραμμα Μεταπτυχιακών Σπουδών
Τεχνητή Νοημοσύνη και Βαθιά Μάθηση
https://aidl.uniwa.gr

UNIVERSITY OF WEST ATTICA
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING
DEPARTMENT OF INDUSTRIAL DESIGN AND PRODUCTION ENGINEERING
http://www.eee.uniwa.gr
http://www.idpe.uniwa.gr
250, Thivon Str., Athens, GR-12241, Greece
Tel: +30 210 538-1614
Master of Science in
Artificial Intelligence and Deep Learning
https://aidl.uniwa.gr

Master of Science Thesis

Solving Multiple Sequence Alignment using Deep Reinforcement Learning

Student : Eirini Kotzia

Registration Number: AIDL-0022

Supervisor : Dr. Panagiotis Kasnesis

ATHENS-EGALEO , February 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΗΣ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΠΑΡΑΓΩΓΗΣ
http://www.eee.uniwa.gr
http://www.idpe.uniwa.gr
Θηβών 250, Αθήνα-Αιγάλεω 12241
Τηλ: +30 210 538-1614
Διατμηματικό Πρόγραμμα Μεταπτυχιακών Σπουδών
Τεχνητή Νοημοσύνη και Βαθιά Μάθηση
https://aidl.uniwa.gr

UNIVERSITY OF WEST ATTICA
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING
DEPARTMENT OF INDUSTRIAL DESIGN AND PRODUCTION ENGINEERING
http://www.eee.uniwa.gr
http://www.idpe.uniwa.gr
250, Thivon Str., Athens, GR-12241, Greece
Tel: +30 210 538-1614
Master of Science in
Artificial Intelligence and Deep Learning
https://aidl.uniwa.gr

Μεταπτυχιακή Διπλωματική Εργασία

Επίλυση Ευθυγράμμισης Πολλαπλών Ακολουθιών χρησιμοποιώντας Βαθιά Ενισχυτική Μάθηση

Φοιτήτρια : Ειρήνη Κοτζιά
Αριθμός Μητρώου: AIDL-0022

Επιβλέπων : Δρ. Παναγιώτης Κασνέσης

ΑΘΗΝΑ ΑΙΓΑΛΕΩ, Φεβρουάριος 2024

This MSc Thesis has been accepted, evaluated, and graded by the following committee:

Supervisor	Member	Member
Panagioti s Kasnesis Digitally signed by Panagiotis Kasnesis Date: 2024.03.08 14:25:42 +02'00'	[illegible] Digitally signed by Charalampos Patrikakis Date: 2024.03.07 12:27:18 +02'00'	[illegible]
Kasnesis Panagiotis	Patrikakis Charalampos	Dominik Grimm
Lecturer	Professor	Professor
Electrical & Electronic Engineering	Electrical & Electronic Engineering	TUM Campus Straubing for Biotechnology and Sustainability
University of West Attica	University of West Attica	Technical University of Munich & Weihenstephan-Triesdorf University of Applied Sciences

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Η κάτωθι υπογεγραμμένη Ειρήνη Κοτζιά. του Νικολάου, με αριθμό μητρώου 0022 μεταπτυχιακή φοιτητήτρια του ΔΠΜΣ «Τεχνητή Νοημοσύνη και Βαθιά Μάθηση» του Τμήματος Ηλεκτρολόγων και Ηλεκτρονικών Μηχανικών και του Τμήματος Μηχανικών Βιομηχανικής Σχεδίασης και Παραγωγής, της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής,
δηλώνω υπεύθυνα ότι:
«Είμαι συγγραφέας αυτής της μεταπτυχιακής διπλωματικής εργασίας και κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος. Η εργασία δεν έχει κατατεθεί στο πλαίσιο των απαιτήσεων για τη λήψη άλλου τίτλου σπουδών ή επαγγελματικής πιστοποίησης πλην του παρόντος. Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του διπλώματός μου.
Επιθυμώ την απαγόρευση πρόσβασης στο πλήρες κείμενο της εργασίας μου μέχρι 29 Φεβρουαρίου 2024 και έπειτα από αίτησή μου στη Βιβλιοθήκη και έγκριση του επιβλέποντος καθηγητή.»

Η Δηλούσα
Ειρήνη Κοτζιά

(Υπογραφή φοιτητή/ήτριας)

[illegible]

Declaration of the author of this MSc thesis

I, Eirini Kotzia, Nikolaos, with the following student registration number: 0022, postgraduate student of the MSc program in "Artificial Intelligence and Deep Learning", which is organized by the Department of Electrical and Electronic Engineering and the Department of Industrial Design and Production Engineering of the Faculty of Engineering of the University of West Attica, hereby declare that: ,
I am the author of this MSc thesis and any help I may have received is clearly mentioned in the thesis. Additionally, all the sources I have used (e.g., to extract data, ideas, words or phrases) are cited with full reference to the corresponding authors, the publishing house or the journal; this also applies to the Internet sources that I have used. I also confirm that I have personally written this thesis and the intellectual property rights belong to myself and to the University of West Attica. This work has not been submitted for any other degree or professional qualification except as specified in it.
Any violations of my academic responsibilities, as stated above, constitutes substantial reason for the cancellation of the conferred MSc degree. I wish to deny access to the full text of my MSc thesis until 29th of February 2024, following my application to the Library of UNIWA and the approval from my supervisor.

The author
Eirini Kotzia

(Signature)

[illegible]

Στην αγαπημένη μου γιαγιά Ευδοξία

Acknowledgements

At this point, I would like to thank the Department of Biotechnologie und Nachhaltigkeit (TUMCS) at Campus Straubing for hosting me as a Practical Student. I would also like to thank Jonathan Pirnay, whose expertise and guidance have been instrumental, and Prof. Dr. Dominik Grimm, for his academic prowess and mentorship.
I would also like to acknowledge the support and knowledge from Panagiotis Kasnesis from the University of West Attica and Prof. Charalampos Z. Patrikakis, who contributed significantly through this cooperation between the two universities. Your collective guidance and expertise have been the cornerstone of my academic success. Furthermore, I appreciate the University of Alberta for the 'Fundamentals of Reinforcement Learning' course, which significantly enhanced my understanding of different Reinforcement Learning concepts. My sincere appreciation to NCBI for providing Entrez a molecular biology database system, a fundamental resource in advancing my research and scientific pursuits. Finally, I want to thank Jonathan Pirnay for acquiring policy-based-self-competition, which served as a foundational resource for the development of this project.

Abstract

Multiple Sequence Alignment (MSA) is a fundamental task in Bioinformatics, essential for understanding evolutionary relationships, genetic adaptations, drug design, and other applications. In general, MSA is a Nondeterministic Polynomially complete problem with many heuristic solvers approaching over the years.
Machine Learning (ML) can handle combinatorial optimization problems by learning models that generalize in various instances. Recent years have witnessed a surge of interest in applying Deep Reinforcement Learning (DRL) techniques for training agents to approach MSAs.
In this work, we introduce IntellAlign, a novel methodology for aligning sequences using DRL for training and Natural Language Processing (NLP) approach. We build strong policies with only a few simulations, no previous knowledge, and self-play using the Gumbel AlphaZero algorithm. Our goal is maximizing a multi-objective reward based on well-known MSA quality metrics. We also propose attention-based networks for encoding MSAs. We contribute by achieving full flexibility of the sequence shape and by allowing the agent with a stop move to finish the alignment. We utilize the Glimpse-Pointer Network for pointing a series of positions sequentially to add a gap in the MSA. Finally, we utilize positional encodings for passing the sequence structural information to the network.
Our algorithm is compared with three well-established aligners (Clustal Omega, MAFFT, and MUSCLE) on DNA MSAs. The results are promising, showing that IntellAlign outperforms MAFFT and MUSCLE tools. Moreover, it tends to score close to the Clustal Omega tool.

Keywords

Reinforcement Learning, Multiple Sequence Alignment, Bioinformatics, Deep Learning, Gumbel AlphaZero, Transformers, NLP

Περίληψη

Η Ευθυγράμμιση Πολλαπλών Ακολουθιών (MSA) είναι μια θεμελιώδης εργασία στον τομεα της Βιοπληροφορικής, απαραίτητη για την κατανόηση των εξελικτικών σχέσεων, των γενετικών προσαρμογών, του σχεδιασμού φαρμάκων και άλλων εφαρμογών. Παραδοσιακά, χρησιμοποιούνται ευρετικές μέθοδοι λόγω του ότι το (MSA) θεωρείται ένα πλήρες μη ντετερμινιστικό πρόβλημα πολυωνυμικού χρόνου (NP-complete), αλλά θα πρέπει να εισαχθούν πιο προηγμένες λύσεις για αυτό το πρόβλημα συνδυαστικής βελτιστοποίησης. Τα τελευταία χρόνια έχει παρατηρηθεί ένα κύμα ενδιαφέροντος για την εφαρμογή τεχνικών Βαθιάς Ενισχυτικής Μάθησης (DRL) για την εκπαίδευση μοντέλων (MSA).
Σε αυτήν την εργασία, παρουσιάζουμε το IntellAlign, μια νέα μεθοδολογία για την ευθυγράμμιση ακολουθιών χρησιμοποιώντας τεχνικές Βαθιάς Ενισχυτικής Μάθησης (DRL) για την εκπαίδευση και προσεγγίζοντας το πρόβλημα με τεχνικές Επεξεργασίας Φυσικής Γλώσσας (NLP). Κατασκευάζουμε ισχυρές στρατηγικές με λίγες μόνο προσομοιώσεις, χωρίς προηγούμενη εμπειρία και αυτο-παιχνίδι χρησιμοποιώντας τον αλγόριθμο Gumbel AlphaZero. Στόχος μας είναι να μεγιστοποιήσουμε μια πολλαπληανταμοιβή με βάση διαδεδομένες μετρικές ποιότητας MSA. Προτείνουμε επίσης δίκτυα που βασίζονται σε αρχιτεκτονική Attentionγια την κωδικοποίηση MSA. Συμβάλλουμε επιτυγχάνοντας πλήρη ευελιξία του μεγέθους τών ακολουθιών και επιτρέποντας στον πράκτορα με μια κίνηση να ολοκληρώσει την ευθυγράμμιση. Χρησιμοποιούμε το Δίκτυο Glimpse-Pointer προκειμένου να επιδείξουμε διαδοχικά μια σειρά από θέσεις στις οποίες ένα κενό θα προστεθεί στο MSA. Τελικά, χρησιμοποιούμε κωδικοποιήσεις θέσεων για τη μεταφορά των δομικών πληροφοριών των ακολουθιών στο δίκτυο
Ο αλγόριθμός μας συγκρίνεται με καθιερωμένα εργαλεία ευθυγραμμιστών (Clustal Omega, MAFFT και MUSCLE) σε MSAsπου περιέχουν DNAακολουθίες. Τα αποτελέσματα είναι ενθαρρυντικά, δείχνοντας ότι ο IntellAlignμπορεί να ξεπεράσει στις περισσότερες περιπτώσεις το MAFFT και το MUSCLE . Επιπλέον, τείνει να επιτυγχάνει παρόμοια αποτελέσματα με το Clustal Omega.

Λέξεις-Κλειδιά

Βαθιά Ενισχυτική Μάθηση, Πολλαπλή Σειριακή Συντονισμένη Ακολουθία , Βιοπληροφορική, Βαθιά Μάθηση, Επεξεργασία Φυσικής Γλώσσας, AlphaZero, Gumbel AlphaZero, Transformers

Contents

List of Tables

List of Figures

Glossary

A3C Asynchronous Advantage Actor Critic. 52

AI Artificial Intelligence. 22, 23, 25, 34, 42, 51

APDB Analyze alignments with PDB. 113

BALiBASE Benchmark Alignment dataBASE. 112–114

BLAST Basic Local Alignment Search Tool. 21

BLOSUM Blocks Substitution Matrix. 58

CCG Complete Column Gaps. 63

CNN Convolutional Neural Network. 37, 38, 44, 97

COP Combinatorial Optimization Problem. 20

DNN Deep Neural Network. 23, 32, 35

DRL Deep Reinforcement Learning. 5, 7, 24, 32, 52, 72, 96

EI Evolutionary Information. 22

FF Feed Forward. 70

FFT Fast Furier Transform. 50

GAZ Gumbel Alpha Zero. 73, 75, 76

GELU Gaussian Error Linear Unit. 68, 70

KL divergence Kullback–Leibler divergence. 42, 75

LSTM Long Short-Term Memory. 52

MAFFT Multiple Alignment using Fast Fourier Transform. 21, 51, 53, 56, 72, 74, 77, 90, 97, 112, 114

MCTS Monte Carlo Tree Search. 23, 32, 33, 35–37, 39, 51, 52, 75, 77, 79, 90, 96

MDP Markov Decision Process. 28–30, 32, 52, 72, 73, 96

MHA Multi Head Attention. 45, 69, 70, 96

ML Machine Learning. 5, 22, 25, 42, 73

MLP MultiLayer perceptron. 68, 70

MRP Markov Reward Process. 28

MSA Multiple Sequence Alignment. 5, 7, 15, 20–24, 47, 49, 51–58, 60, 62–65, 72–74, 77, 81, 96, 97, 114

MSE Mean Squared Error. 36, 38, 42, 75

MUSCLE Multiple Sequence Comparison by Log- Expectation . 50, 51, 53, 56, 72, 74, 77, 79, 90, 97, 112, 114

NCBI National Center for Biotechnology Information. 73

NLP Natural Language Processing. 5, 7, 23, 42–44, 60, 72, 73

NP-complete Nondeterministic polynomial-time complete. 7, 20

RL Reinforcement Learning. 23, 25, 27, 28, 30, 32, 34, 35, 37, 51–53, 73, 109

RNN Recurrent Neural Network. 43, 44

Seq2Seq Sequence to Sequence. 43, 44

SGD Stohastic Gradient Descent. 38

SL Supervised Learning. 25, 35

SP Sum of Pairs. 16, 52, 53, 56, 57, 64, 73, 77, 79–82, 90, 92, 95–97

T-Coffee Tree-based Consistency Objective Function for alignment Evaluation. 50

TC Totally Conserved or Column Score. 16, 53, 59, 73, 77, 79–81, 90, 94–97

TSP Traveling Salesman Problem. 69, 70

UCT Upper Confidence bounds applied to Trees. 33, 39, 52

WSP Weighted Sum of Pairs. 57

Chapter 1

Introduction

1.1 Overview

MSA has been a prominent research topic since the late 1960s. MSA refers to the alignment of more than two DNA, RNA, or protein sequences aiming to extract valuable information concerning functional, structural, and evolutionary relationships. During the alignment process, gaps are introduced within the sequences, resulting in a new set of modified sequences. MSA lies in the family of combinatorial optimization problems (COPs), with the "goal" being finding the optimal solution as a path over a finite set of possible choices. Moreover, it is defined, in general, as an NP-complete problem (Wang & Jiang, 1994) with many recognizable heuristic solvers, each with unique strengths and limitations. Despite decades of research, MSA remains a challenging computational problem without a universally accepted global solution. This is primarily due to multiple factors, including the ambiguity of the problem, making it difficult to define a single "correct" alignment. Such ambiguity further complicates the evaluation and definition of scoring and objective functions. Moreover, one of the foremost challenges in MSA is the complexity of the alignment process. To illustrate this, let's consider a simple example. Imagine having a set of three sequences, each of length three and containing only four distinct letters. For example, in Figure 1.1, if our objective is maximizing vertical matches between the sequences, shifting the first sequence one cell to the right would yield more matches than the initial alignment. In this scenario, it is relatively straightforward for a human to manually align these sequences for a given objective. However, this

simplicity drastically increases when dealing with larger alignments. Dynamic programming has been common and efficient for approaching MSA problem. Still, the computational cost grows rapidly with the number N of sequences to be compared ($\mathcal{O}(l^N)$, with l being the mean length of sequences) (Carrillo & Lipman, 1988).
While the significance of MSA may not be immediately apparent, its impact extends across diverse domains. MSA is beneficial for analyzing the taxonomy of sequences, drug discovery, establishing genotype or phenotype correlations, unraveling evolutionary forces, etc.(Aniba et al., 2010). The following subsection will highlight the importance of MSA with recent usage examples, aiming to generate some interest and understanding of the topic. Finally, the introduction will be wrapped up by discussing this thesis's motivation and goals.

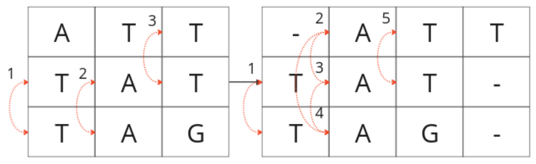


Figure 1.1: Example alignment. In this alignment, red arrows correspond to one-to-one matching counting from left to right in a column-wise manner.

1.1.1 Significance

Delving into MSA's practical applications becomes crucial to understanding its importance. This broader perspective extends beyond focusing only on the MSAs computational challenges, and it holds particular relevance for research and applications in various fields.
MSA is a valuable tool to identify similarities among genetic sequences, aiding in the systematic categorization of species. By aligning and comparing DNA or protein sequences from different organisms, MSA enables researchers to discern shared genetic patterns and evolutionary relationships, facilitating the accurate classification of species. Modha et al. (2018) claims that the frequency of genetic sequence submission in databases escalates. They propose the ViCTree tool for taxonomic clustering, which applies MSA and then BLAST (Altschul et al., 1990) MSA tool to filter relevant sequences with some predefined criteria. Another great application example is Read2Tree by Dylus et al. (2023), which utilizes the MAFFT(Katoh et

al., 2002) MSA tool for building phylogenetic trees [1].
Next, Genotype-Phenotype Correlations is a way of finding the mutations within a given genotype responsible for the presence of a physical trait (Frew et al., 2019). This process can provide information regarding disease pathogenesis, future disease progression, severity, etc. SigniSite tool is proposed by Jessen et al. (2013) for finding genotype-phenotype correlations by employing as inputs protein MSAs. More specifically, given MSAs as inputs accompanied with a real number that quantifies the presence of a phenotype in a sequence, they predict the probabilities of amino acids being related to the given phenotype.
Finally, protein structure prediction is a crucial problem for understanding protein functionalities, which can be vital for making hypotheses on ways to affect, control, or modify them (Darnell, 2020). Most state-of-the-art ML approaches for this problem employ evolutionary information (EI) derived from MSAs as inputs (Heinzinger et al., 2022). In the groundbreaking AlphaFold created by DeepMind (Jumper et al., 2021), MSAs are a key point since they serve as inputs to the network for passing co-evolution information from the homologous sequences (Fang et al., 2023). CopulaNet uses directly MSAs as inputs for learning residues co-evolution (Ju et al., 2021). Another exciting application is related to vaccine development. Langya henipavirus, a virus of increasing concern since 2018, has captured researchers' attention due to its relation to deadly viruses. Researchers claim that if studying proteins is challenging, it becomes much more difficult when attempting to create them in a laboratory (Callaway, 2023). Viral mutation and recombination are the main hardships in vaccine development since they provoke high genetic diversity. These genetic changes may be intractable and offer limited vaccines and therapeutic development time. EVEscape (Thadani et al., 2023) Artificial Intelligence (AI) tool predicts the evolution of SARS-CoV-2, giving warnings of possible mutation crucial for vaccine development earlier than lab-based methods. Moreover, EVEscape can generalize to other viruses, such as HIV. MSAs are utilized as part of the input sets for training their networks.

1.2 Personal Motivation - Research Aims

At this point, we have established a robust foundation for understanding that MSA:

[1] They present an interesting example of classification SARS-CoV-2 and clustering of its variances (Figure A.5)

- is a scientific combinatorial optimization problem that raises significant challenges as it is computationally demanding and requires strong decision-making abilities.
- is a problem with a notable lack of a global solver and a global quality measure,
- can highly contribute to applications that try to solve important problems facing humanity, such as a pandemic

AI has recently become a prominent tool in addressing challenges in biology and genetics, with major research groups such as DeepMind showing interest in their research activity. Genomic data, characterized by their complexity and continuously increasing volume, can benefit from advancements in the AI field. Researchers have successfully harnessed AI techniques to extract valuable insights from genomic data. Furthermore, Reinforcement Learning (RL), a subfield of AI, can be helpful in complex decision-making tasks, and MSA is one such domain. By employing Deep Neural Networks (DNN) to create policies, RL can help discover novel strategies and alignments, whereas traditional rule-based, heuristic approaches may not yield innovative results.
We propose a novel approach, IntellAlign, an RL agent that can apply strong MSAs without any previous knowledge. Our method incorporates NLP techniques, aiming to design a flexible pipeline compatible with DNA, RNA, and protein structures of variable lengths and number of sequences to align. To achieve this, we utilize Transformer architectures with a Glimpse-Pointer Mechanism. We also employ Positional Encodings to retain structural information about nucleotide positions. The main objective is to maximize standard used score metrics for MSAs and to compete with some of the most known current MSA solvers. For maximizing the objectives without any knowledge of good strategies for solving an MSA, we train our agent using Gumbel AlphaZero (Danihelka et al., 2022), a recent advancement of the AlphaZero (Silver et al., 2018) algorithm. AlphaZero allows the agent to learn by playing and competing with himself, giving no information about a good or bad alignment but only the game's rules. Gumbel expansion allows the agent to learn better strategies in a shorter amount of simulations. Our approach introduces the application of network-guided MCTS to MSA. While MCTS has been previously applied to MSA (Edelkamp & Tang, 2015), to our knowledge, there is no substantial work exploring the integration of network-guided MCTS in the context of MSA.

1.3 Thesis Organization

The rest of the thesis is organized as follows:

- Chapter 2 contains essential background considering information on the terminology, notation, algorithms, and network architectures. It equips the reader with all the required elements for understanding the contents of the subsequent Chapters. Additionally, this chapter provides a comprehensive literature review on current MSA-solving approaches, including Deep Reinforcement Learning (DRL) works.
- Chapter 3 holds a clear mathematical definition of the MSA problem and introduces common score and evaluation metrics.
- Chapter 4 delves into materials, methods, and the chosen model architecture.
- Chapter 5 outlines the experimental goals and supplies general details about the experimental setup. Next, it presents the experimental results and provides a detailed comparison with other alignment tools. This chapter also analyzes the results and discusses key findings.
- Chapter 6 summarizes conclusions, discusses limitations, and outlines the future scope of research.

Chapter 2

Background and Literature Review

2.1 Background

In the following subsection, certain key concepts, theories, and terminology utilized in the subsequent sections are clarified and discussed. Subsections 2.1.1, 2.1.2 are built upon Sutton and Barto (2018)'s book and Silver (2015)'s lectures.

2.1.1 Reinforcement Learning

Supervised learning (SL) has made remarkable progress in the AI field but relies heavily on high-quality data to learn effectively, which can be impractical, costly, or even impossible to have in certain scenarios. On the other hand, RL is a branch of ML that does not require labeled examples. Instead, it draws inspiration from how animals and humans learn through interactions with their environment. RL, introduces an *agent* interacting with an *environment*. Similarly to animals or humans, the agent tries to learn how to behave using the feedback from the environment given as rewards or penalties. The agent is responsible for learning the problem dynamics and acting accordingly to maximize the rewards. This is a circular process where the environment contains all the information, and the agent follows a series

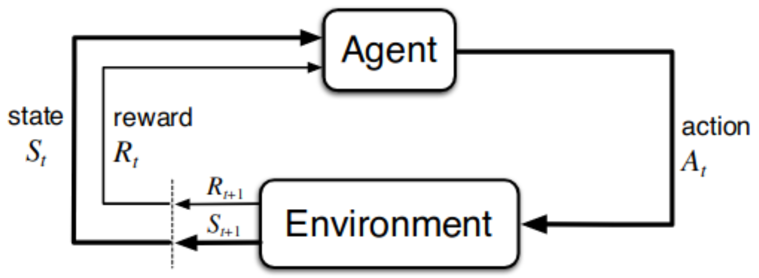


Figure 2.1: Interaction of the agent with the environment (Sutton & Barto, 2018)

of decisions.
Let $\mathcal{S}$ be the set of all possible states and $\mathcal{A}$ the set of all possible actions. At each timestep, t, the environment reveals the current situation, namely the state, $S_t \in \mathcal{S}$, to the agent. Next, the agent selects an action, $A_t \in \mathcal{A}(S_t)$, where $\mathcal{A}(S_t)$ is the set of available actions in the state S_t. The action A_t will be applied to the environment, resulting in a new state S_{t+1}. Finally, the agent receives a numerical reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ as a result of the chosen action (Figure 2.1). The rewards can be either positive or negative due to previous actions.

Policy

At any given timestep t, the agent will choose an action A_t according to a policy π_t. This policy can be considered as the agent's strategy. The policy can be stochastic, meaning that it may produce probabilities of each action for a given state, or it could be deterministic in cases where the policy maps a state to a specific action. A policy specifies probabilities over actions A_t given S_t in stochastic cases. The policy changes while the agent gets more knowledge of the environment.

Reward

The reward can be considered as a signal from the environment due to the agent's behavior. Rewards can be provided at each timestep (intermediate rewards) or the end of an episode (episodic rewards). The agent is trying to maximize the reward; thus, the reward affects the policy. Moreover, the agent can influence the reward through its actions. The reward defines the desired and undesired actions and is the

primary way of learning the optimal way to handle a given task.
Although the reward signal gives feedback to the agent on how to behave, the agent's goal is not to maximize the immediate reward. An RL aims to maximize the cumulative reward in the long run.

Returns

As mentioned earlier, the agent's goal is to maximize the cumulative reward in the long run. This is formally defined as the expected return, G_t, and is the total discounted reward from timestep t. It can be expressed mathematically using the following geometric formula :

$$G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \tag{2.1}$$

$$G_t := \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

where R_{t+i} is the reward obtained at timestep $t+i$, and $0 \leq \gamma \leq 1$ is the discount factor. If the γ value is one, there will be no discounting, leading to an infinite sum in continuing tasks. So γ is a discount factor that can make G_t always be finite. For γ close to zero, the agent will pay attention to the immediate rewards. On the other hand, when γ is closer to 1, the agent is more far-sighted, meaning future rewards are considered more strongly than immediate rewards.

Model of the Environment

Another element for some RL systems is the model of the environment. In some cases, models are utilized for planning, leading to model-based methods. In contrast, in model-free methods, the agent learns solely from experience, and it is hard to plan. Precisely, in model-based approaches, the model can foresee state transitions and rewards and thus offer a significant advantage: the ability to explore various possibilities, plan ahead, and make informed decisions. This process is usually mentioned as planning in the context of RL.

2.1.2 Markov Decision Process

Almost all RL problems can be mathematically formalized as a Markov Decision Process (MDP). Based on Markov Property, the future is determined only by the present state and not by the past states. Roughly, the current state includes information on the historical states so that the current state is sufficient input to understand the future. When a RL task satisfies the Markov Property, it is a Markov Reward Process (MRP). Finite MDPs are the ones that share finite spaces of states and actions. The dynamics of such an MDP are defined as :

$$p\left(s', r \mid s, a\right) = \Pr\left\{S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a\right\}, \tag{2.3}$$

Similarly, we can define policy π as $\pi(a|s) = \Pr\{A_t = a \mid S_t = s\}$, which specifies a distribution over actions A_t given S_t. The policy is assumed to be Markovian, which depends on the current state (not historical states) and time.
Consider a RL task being a finite MDP with $\mathcal{S}$ being a finite set of states and $\mathcal{A}$ being a finite set of actions. Using Equation 2.3 the following can be defined:

- State transition probabilities (or simply transition probabilities),

$$p(s'|s,a) = \Pr\left\{S_{t+1} = s' \mid S_t = s, A_t = a\right\} = \sum_{r \in \mathcal{R}} p\left(s', r \mid s, a\right)$$

- Expected Rewards for state-action pairs,

$$r(s,a) = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p\left(s', r \mid s, a\right)$$

- Expected Rewards for state-action-next-state triples,

$$r\left(s, a, s'\right) = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s'\right] = \frac{\sum_{r \in \mathcal{R}} r p\left(s', r \mid s, a\right)}{p\left(s' \mid s, a\right)}$$

Finally, we can summarize a finite MDP as a tuple of $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, r, \gamma)$.

Value function

The value function expresses the expected return when following a policy π and starting from state s. Its importance lies in allowing an agent to assess the quality of its present situation without waiting for the extended-term consequences to unfold. The value function (or state-value function) will assert a value for a given state in long-term consideration. For MDPs, the state-value function can be formally defined as :

$$V_\pi(s) := \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \tag{2.4}$$

Another function we need to define is the state-action-value function, also known as the quality function. The state-action-value function describes how well the choice of a particular action is in a given state. More formally, it represents the expected return when following a policy π from state s after performing an action a and can be defined as :

$$Q_\pi(s,a) := \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] \tag{2.5}$$

Bellman equations are incorporated to simplify the state-value and the state-action-value estimation. The Bellman equation for the state-value function defines a relationship between the value of a state and the value of its possible successor states. Taking advantage of the recursive nature of $V_\pi(s)$, the Equation 2.4 can be expanded into :

$$V_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p\left(s', r \mid s, a\right) \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} \mid S_{t+1} = s' \right] \right] \forall s \in \mathcal{S} \tag{2.6}$$

using the sum of all possible actions, the sum of all possible rewards, and the next states. Similarly to the definition, it can be considered as a sum of possible outcomes weighted by the probabilities of their occurrence, $\pi(a \mid s) p(s', r \mid s, a)$. In Equation 2.4 we can substitute the expected return of s' for timestep $t+1$ with the state-value V_π for s' owing to their equivalence, as follows:

$$V_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p\left(s', r \mid s, a\right) \left[r + \gamma V_\pi\left(s'\right) \right] \tag{2.7}$$

$$V_\pi(s) = \sum_a \pi(a \mid s) \sum_{s', r} p\left(s', r \mid s, a\right) \left[r + \gamma V_\pi\left(s'\right) \right] \tag{2.8}$$

Note that in Equation 2.8, the sum over possible states and rewards is merged for notation simplicity into one sum over both. This is the Bellman equation for the state-value function and captures the relationship between the value of the state and the values of the possible successor states. The Bellman Equation for the state-action-value function will be formed similarly but without including the policy π since the action is already selected. The expected return for the next state s' needs to be expressed as a weighted sum over all the possible actions using $\pi\left(a' \mid s'\right)$:

$$Q_\pi(s,a) = \sum_{s',r} p\left(s',r \mid s,a\right)\left[r+\gamma\sum_{a'}\pi\left(a' \mid s'\right)\mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1}=s', A_{t+1}=a'\right]\right] \tag{2.9}$$

Again, we can replace the expected return of s' , a' for timestep $t+1$ using Equation 2.5 resulting to :

$$Q_\pi(s,a) = \sum_{s',r} p\left(s',r \mid s,a\right)\left[r+\gamma\sum_{a'}\pi\left(a' \mid s'\right)Q_\pi\left(s',a'\right)\right] \tag{2.10}$$

Optimal Policies

The primary goal in any RL task is to find a policy that obtains as much reward as possible in the long run. Consider policies π_1 and π_2, we can say that π_1 is as good as or better than π_2, $\pi_1 \geq \pi_2$, if and only the $V_{\pi_1}(s) \geq V_{\pi_2}(s)$ for every state $s \in \mathcal{S}$. For any MDP, there exists at least one optimal policy π_* which is as good as or better than any other policy, $\pi_* \geq \pi, \forall\pi$. In this case, the optimal policy π_* will get the greatest value in any given state. All optimal policies share the same state-value function, called the optimal state-value function, which can be defined as:

$$V_*(s) := \mathbb{E}_{\pi_*}\left[G_t \mid S_t = s\right] = \max_\pi V_\pi(s) \;, \forall s \in \mathcal{S} \tag{2.11}$$

Optimal policies also share the same state-action value function, the optimal action-value function, denoted as Q_*, which is defined as:

$$Q_*(s) := \mathbb{E}_{\pi_*}\left[G_t \mid S_t = s, A_t = a\right] = \max_\pi Q_\pi(s) \;, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s) \tag{2.12}$$

Since V_* is the optimal value, intuitively, the expected return over an optimal policy should be the expected return for the best action given a state. This can be expressed as:

$$
\begin{aligned}
V_*(s) &= \max_{a \in \mathcal{A}(s)} Q_{\pi_*}(s,a) && (2.13)\\
&= \max_a \mathbb{E}_{\pi^*}\left[G_t \mid S_t = s, A_t = a\right] && (2.14)
\end{aligned}
$$

Or similarly to Equation 2.8

$$
V_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma V_*\left(s'\right)\right] \tag{2.15}
$$

where policy reference is replaced by the best action. Finally, the Bellman optimality equation for the optimal state-action value function, Q_*, is defined as:

$$
Q_*(s,a) = \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma \max_{a'} Q_*\left(s', a'\right)\right] \tag{2.16}
$$

Policy Improvement Theorem

Let π and π' be any pair of deterministic policies such that:

$$
Q_\pi\left(s, \pi'(s)\right) \geq V_\pi(s) \ , \forall s \in \mathcal{S} \tag{2.17}
$$

In this case, policy π' is as good as or better than π. That is, it must obtain greater or equal expected return:

$$
V_{\pi'}(s) \geq V_\pi(s) \ , \forall s \in \mathcal{S} \tag{2.18}
$$

The policy π' is an improvement over π since it improves its value in all states. Note that in case of a strict inequality of Equation 2.17 at any state, then there must be strict inequality of Equation 2.18 at that state.
We can define a new greedy policy, π' as the one that selects actions that are believed to be the best according to $q_\pi(s,a)$:

$$
\begin{aligned}
\pi'(s) &\doteq \arg\max_a q_\pi(s,a)\\
&= \arg\max_a \mathbb{E}\left[R_{t+1} + \gamma V_\pi\left(S_{t+1}\right) \mid S_t = s, A_t = a\right]\\
&= \arg\max_a \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma V_\pi\left(s'\right)\right]
\end{aligned} \tag{2.19}
$$

Policy improvement is the process of improving an original policy with a new policy by making it greedy with respect to the value function of the original policy. If the new greedy policy is as good as but not better than the original π, then value functions should equal $V_{\pi'} = V_{\pi}$. In this case, using Equation 2.19, we will fall into the Bellman optimality equation for the value function Equation 2.15.

$$\begin{aligned} V_{\pi'}(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma V_{\pi'}(S_{t+1}) \mid S_t = s, A_t = a\right] \\ &= \max_a \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma V_{\pi'}\left(s'\right)\right] \quad , \forall s \in \mathcal{S} \end{aligned} \tag{2.20}$$

This leads to the conclusion that the new, improved policy is strictly better unless policy π is already optimal.

2.1.3 Deep Reinforcement Learning

In the previous section, we discussed what an RL is, its components, and how it can be formulated as an MDP. We also defined value and optimal value functions. Let us briefly define the Deep Reinforcement Learning DRL domain. DRL is a family of solution methods for RL which uses powerful representations derived from DNN to represent different elements of the RL. These could be, for example, the value function, the policy, or the model. The introduction of deep learning is motivated by its capability to enable RL to scale and address decision-making problems that were previously considered intractable (Arulkumaran et al., 2017).

2.1.4 Monte Carlo Tree Search

In sequential decision problems and particularly within the context of model-based methods, planning algorithms are often employed to explore a sequence of possible choices and their outcomes. An RL agent could make more sophisticated decisions using search trees. The search tree can be exhaustively explored when the problem has a small number of possible cases, but with larger problems, it becomes prohibitively slow and impractical.
Monte Carlo Tree Search (MCTS) is an iterative tree algorithm that balances exploration and exploitation [1]. In such trees, nodes denote states, whereas edges represent

[1]see section A.1

transitions (actions) from one state to another. This method was first proposed by Coulom (2006) and Kocsis and Szepesvári (2006) for making computers play the Go game, as reported by Świechowski et al. (2023). MCTS, consists of four main steps: selection, expansion, simulation, and backpropagation.

- **selection**: The algorithm selects, at each level, a node starting from the root node and terminates when reaching a leaf node. The selection method is called tree policy and is the rule that is used for selecting a child node among a given node's available children.
- **expansion**: The algorithm expands the current node into at least one child node. This could include all the possible transitions from the current node to the next node. This step happens only when the current node is not representing a terminal state.
- **simulation**: Involves the simulation of playing an episode until a terminal state is reached. At this point, the algorithm fetches the payoffs as results from the game. This step is also mentioned as a rollout.
- **backpropagation**: Updates information kept the nodes from the last visited node back to the root node. This information generally refers to each node's value and derives from the simulation step. The value accuracy improves with the number of simulations (Silver et al., 2016).

We summarize these steps in the following pseudocode in Algorithm 1 for a better understanding. In pseudocode, $\mathcal{T}$ represents the set of all nodes. As mentioned in the selection step, defining a tree policy is crucial for an agent to determine which action to select for the next move. This policy allows the agent to explore unseen actions while selecting promising actions in a moderate ratio. The most commonly employed function for this is called Upper Confidence bounds applied to Trees (UCT) by Kocsis and Szepesvári (2006) is the following:

$$a^* = \arg\max_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{\ln[N(s)]}{N(s,a)}} \right\}, \tag{2.21}$$

where $Q(s,a)$ is the average return for choosing action a for state s, $N(s)$ holds the total number of times state s has been visited so far, and $N(s,a)$ holds the total number of times that action a has been sampled from state s. Alternative methods for the selection step include Objective Monte-Carlo (OMC) (Chaslot et al., 2006), Probability to be Better than Best Move (PBBM) (Coulom, 2006) etc.

Algorithm 1 Monte Carlo Tree Search

```
**Input:** $root\_node, computational\_budget$
**Output:** $best\_action$
$current\_node \leftarrow root\_node$
while $computational\_budget$ do
    while $current\_node \in \mathcal{T}$ do
        $last\_node \leftarrow current\_node$
        $current\_node \leftarrow select(current\_node)$
    end while
    $last\_node \leftarrow expand(last\_node)$                ▷ Leaf Node is expanded
    $R \leftarrow simulate(last\_node)$
    $current\_node \leftarrow last\_node$
    while $current\_node \in \mathcal{T}$ do                ▷ Backpropagation step
        $backpropagate(current\_node, R)$
        $current\_node \leftarrow current\_node.parent$
    end while
    $computational\_budget \leftarrow computational\_budget - 1$
end while
return $best\_action = argmax(root\_node)$
```

2.1.5 From AlphaGo to Gumbel AlphaZero

AlphaZero (Silver et al., 2018) is one major algorithm accomplishment in the RL field, developed by the DeepMind team owned by Google. AlphaZero is considered a significant advancement over its predecessor, AlphaGo (Silver et al., 2016). Before we dive into AlphaZero, let's revisit the genesis of AlphaGo.

2.1.5.1 AlphaGo

AlphaGo was designed to master the Go game, which has been considered as one of the most challenging games in the AI field (Silver et al., 2016). Go is a traditional board game of Chinese origin, characterized by its simple rules compared to chess. Despite its apparent simplicity, the possible moves are countless, surpassing the total count of atoms in the known universe, as reported in "From AI to protein folding: Our Breakthrough runners-up" (2016). Another challenge has been evaluating the board positions and the actions (Silver et al., 2016). In 2016, the match between AlphaGo and Lee Sedol, a professional South Korean Go player often regarded as one

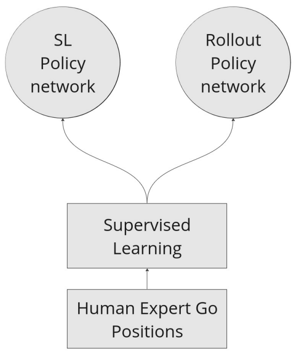


Figure 2.2: Rollout and SL Policy Networks

of the best in his generation, was marked as a monumental moment. AlphaGo won in a five-game Go match and was later honored with the highest Go grandmaster rank by the Korea Baduk Association. AlphaGo was initially published in Nature and later recognized and featured in Science as 'one of the most significant scientific breakthroughs of 2016' ("From AI to protein folding: Our Breakthrough runners-up", 2016). For detailing the methods employed by AlphaGo, we rely on the original paper by Silver et al. (2016).

Methods AlphaGo uses SL over a corpus of human expert player moves combined with RL self-play games for training policy and value DNNs. The algorithm includes MCTS that combines neural network evaluations with Monte Carlo rollouts. Specifically, the pipeline is divided into three stages.
The first stage includes training a policy network $\pi^{\sigma}(a|s)$ [2] on numerous randomly selected expert moves encapsulated as state-action pairs using SL. Moreover, a faster but weaker rollout policy network $\pi^{\phi}(a|s)$ is also trained. They apply stochastic gradient ascend to both policies to maximize the log-likelihood of selecting move a for a given state s. The first network is the SL policy network, and the second one is the Rollout policy network (as shown in Figure 2.2). A policy network π^{ρ} with the same architecture as the SL policy network is then introduced. This network is the RL policy network, and its weights are initialized with the same values, such as $\rho = \sigma$. The RL policy network is improved using Policy Gradient RL (Sutton et al.,

[2]We have violated the notation for weights, transitioning from subscript to superscript for the weight letter. This change is implemented to prevent confusion with the subscript π used to denote policy in Chapter 2 for the value function.

2000; Williams, 1992), including self-play and weight update by stochastic gradient ascend for maximizing the expected game outcome.
Last, a value network is trained $V_\pi^\theta(s)$ to predict the game's outcome at a given moment. Since the optimal value function $V_*(s)$ is unknown and pursued, they estimate it using the RL-policy network as the strongest one. The weights of the value network are updated using stochastic gradient ascend to minimize the Mean Squared Error (MSE) between the network output $V_\pi^\theta(s)$ and the actual game outcome, z. The architecture of the value network closely resembles that of the RL policy network but with a single numerical output and is trained using regression methods.
One interesting point is that training using successive game frames led to over-fitting due to their high similarity. To overcome this challenge, 30 million games are gathered. From this dataset, a single random game moment (one frame) is sampled for training, aiding the value network in generalizing more effectively. Previously (in Section 2.1.4), we have discussed the main components of a MCTS. AlphaGo combines policy and value networks within a MCTS.
For action selection from the root node of MCTS, a variant of PUCT algorithm (Rosin, 2011) is utilized :

$$a_t^* = \underset{a \in A(s_t)}{\operatorname{argmax}} \left(Q(s_t, a) + U(s_t, a)\right) \tag{2.22}$$

where $U(s,a)$ is added for encouraging exploration :

$$U(s,a) = c_{\text{puct}} P(s,a) \frac{\sqrt{N(s)}}{1 + N(s,a)} \tag{2.23}$$

where $P(s,a)$ is the prior probability derived by the SL policy network, $N(s)$ is the visit count of the parent node, and parameter c_{puct} is a constant determining the level of exploration.
Starting from the root node, the tree policy is utilized for action selection, transitioning to successive nodes until reaching a leaf node. When reaching a leaf node, the node is expanded, and the probabilities of selecting available actions from that state are calculated once using the $\pi^\sigma(a|s)$ from the SL policy network and stored as prior probabilities, $P(s,a)$:

$$P(s,a) = \pi^\sigma(a|s) \tag{2.24}$$

Next, random rollouts are applied as part of the simulation step from the leaf node up to the terminal node (where the game finishes) using the fast rollout policy network, $\pi^\pi(a|s)$. This leaf node is evaluated using a combination of the actual game outcome,

z_L, from the rollout play with the fast rollout policy network and the value networks $V_\pi^\theta(s_L)$ prediction. This combined leaf node evaluation is formulated as follows:

$$V(s_L) = (1-\lambda)V_\theta(s_L) + \lambda z_L \tag{2.25}$$

where λ is a weighting parameter, and s_L represents the leaf state nodes. Finally, at the backpropagation step, the action values and the visit count are calculated and stored (for the first time or updated) for all the edges in the tree path. The action values Q(s, a) are calculated as an average of $V(s_L)$ values over all simulations. $N(s,a)$ is the total number of visits for the edges in all simulations. After applying many simulations, AlphaGo selects the moves according to the highest number of visits.

2.1.5.2 AlphaGo Zero - AlphaZero

AlphaGo Zero, introduced by Silver et al. (2017), serves as the successor to AlphaGo. A year later, AlphaZero (Silver et al., 2018) was published, extending the principles of AlphaGo Zero to Chess and Shogi games. The goal was to increase performance and generality. In AlphaGo Zero, Deep Mind drops any expert human knowledge of the game and relies exclusively on learning using self-play games. They also switch to one single network for policy and value prediction. The input representation of the game is simplified. Finally, they apply modifications to the search tree.

AlphaGo Zero Methods AlphaGo Zero, is trained using RL self-play games. In contrast with AlphaGo, there is only one CNN network, $f^\theta(s)$, with θ parameters, that combines the value mentioned above V and policy π networks. The networks' parameters θ are randomly initialized. In each state s, a MCTS is applied. Inside the MCTS, similarly to AlphaGo, the Equation 2.22 is employed for selecting an action. Same as AlphaGo, the node is expanded after reaching a leaf node. The Monte Carlo step is dropped after the expansion and follows the evaluation step. Like AlphaGo, the leaf node is evaluated once, but this time using the aforementioned single network $f^\theta(s)$, producing a probability distribution over actions and a state value representing the agent's probability of winning the game from state s.

$$(\pi, V) = f^\theta(s) \tag{2.26}$$

Similarly, both are stored as $P(s,a)$, prior probability for each edge, and $V(s_L)$ in the leaf node. Additional exploration is achieved here by adding Dirichlet noise to the prior probabilities :

$$P(s,a) = (1-\varepsilon)\pi(a|s) + \varepsilon\eta_a \tag{2.27}$$

where $\eta \sim \text{Dir}(0.03)$ and $\varepsilon = 0.25$. Finally, the backpropagation step is the same as AlphaGo, where the action values and the visit counts are calculated and updated. After completing the search, the AlphaGo Zero selects an action from the current root node state s_0 according to the improved policy:

$$\pi'(a \mid s_0) = \frac{N(s_0,a)^{1/\tau}}{N(s_0)^{1/\tau}} \tag{2.28}$$

Here, τ is a temperature parameter that controls the exploration-exploitation trade-off. This parameter varies during self-play, initiating with $\tau = 1$ for the first 30 moves per game to allow exploration and continuing to values close to zero for selecting the most visited actions.
Self-play is executed asynchronously in parallel. The parameters θ of the network are continuously optimized by evaluating the agents, with the best-performing agent being retained. Moreover, they are optimized using stochastic gradient descent SGD with momentum set to 0.9 and learning rate annealing. Finally, the network parameters are adjusted to minimize MSE between game outcome and value prediction and cross-entropy loss between the network probabilities for policy and the search tree probabilities:

$$(\pi, V) = f_\theta(s) \text{ and } l = (z-V)^2 - (\pi')^{\mathrm{T}} \log \pi + c\|\theta\|^2 \tag{2.29}$$

where c is the parameter of L2 weight regularization. This regularization serves the purpose of minimizing overfitting. Considering the network architecture, it generally comprises multiple CNN blocks, followed by a policy head that outputs logits for probabilities and a value head that outputs a scalar value ranging from -1 to 1.

AlphaZero Methods AlphaZero builds upon AlphaGo Zero with a few adjustments to make it applicable to Chess and Shogi. The algorithm and network architecture remain the same across several games, such as Go, Chess, and Shogi by AlphaZero, demonstrating a more general solution. The content of AlphaZero is supplementary to AlphaGo Zero, highlighting three modifications while maintaining overall similarity.
One key difference between AlphaGo Zero and AlphaZero is that the game outcome is not formed as a binary result of win or loss but tries to predict and optimize the expected result z. This adjustment is important since, unlike the game of Go, where

outcomes are limited to win or loss, Chess and Shogi introduce the potential for a draw outcome.
The game of Go is symmetric, meaning that rotation and reflection of the board do not impact the game's rules. On the other hand, Chess and Shogi do not have this property on their board. Therefore, AlphaZero, unlike AlphaGo Zero, abstains from applying any state augmentations. The board is oriented to the perspective of the current player. Another distinction is that instead of waiting for an iteration to complete and updating the network parameters in case there is a better player, the weights are continuously updated. When an agent initiates self-play, the latest network checkpoint is fetched.

2.1.5.3 Gumbel Alpha Zero Policy Improvement

AlphaZero can fail to improve its policy network if the number of simulations is insufficient to explore the whole action space. AlphaZero applies Dirichlet noise (Silver et al., 2017) to the prior probabilities in the root node (see Equation 2.27) to encourage exploring uncertain actions, but this strategy may harm a potential optimal policy network (Danihelka et al., 2022). In Gumbel AlphaZero and Gumbel MuZero (Danihelka et al., 2022), an improved version of AlphaZero and MuZero (Schrittwieser et al., 2020) algorithms is designed by alternating some steps of the initial algorithms. These alternations involve:

1. the selection of the actions to search on the tree's root utilizing the Gumbel-Top-k trick for sampling a subset of the available actions and then using the same Gumbel distribution for directing the best action selection in the search,
2. the selection of actions at the root node by utilizing the Sequential Halving algorithm as proposed by Karnin et al. (2013),
3. the action selection from the environment (after the search simulations) with the final action resulting from the Sequential Halving search process,
4. the update of weights for the policy network in a way that secures an improvement of the policy, based on the root actions values derived from the search, and finally,
5. the action selection inside the search tree, which instead of the PUCT (Rosin, 2011) algorithm for AlphaZero or UCT as mentioned in a common MCTS, a deterministic action selection method is proposed. The aforementioned method

uses an improved policy based on the completed Q values. Then, the action is not selected directly from the improved policy but from aligning the empirical visit counts with the intended policy enhancement.

Methods As mentioned, AlphaZero's original version utilizes Dirichlet noise as an addition to the policy network to enhance exploration. In contrast, Gumbel AlphaZero uses the Gumbel-Top-k trick to sample actions without a replacement at the root node.
Gumbel-Top-k trick (Kool et al., 2019, 2020; Vieira, 2014; Yellott, 1977), is an extension of the Gumbel-Max trick (Gumbel, 1954; Luce, 1959). Gumbel-Max trick is a method for sampling from categorical distributions. In the Gumbel-Top-k trick, a fixed number of actions are sampled from a Gumbel distribution and added to the prior logits, $logits^{\pi}$. Prior logits are derived from the logits generated by the policy network π, representing an unnormalized categorical distribution. The prior logits are calculated and stored once during the expansion step of each node. Each action is sampled sequentially without a replacement to reduce the variance. From this collection, the top m actions will represent the sampled set $A_{top_m} = \{A_1, A_2, \ldots, A_m\}$:

$$\left(g \in \mathbb{R}^k\right) \sim \text{Gumbel}(0)$$
$$A_{top_m} = \text{argtop}(g + \text{logits}^{\pi}, m))$$

Next, given a tree node, the same Gumbel distribution g is utilized to find the action from A_{top_m} that maximizes Equation 2.30. This action is considered the best, A_{n+1}.

$$A_{n+1} = \text{argmax}_{a \in A_{top_m}}(g(a) + \text{logits}^{\pi}(a) + \sigma(\hat{Q}(a))) \tag{2.30}$$

Note that σ can be any monotonically increasing function, but the proposed is formulated as:

$$\sigma(\hat{Q}(a)) = \left(c_{\text{visit}} + \max_b N(b)\right) c_{\text{scale}}\, \hat{Q}(a) \tag{2.31}$$

where $\max_b N(b)$ is the maximum number of visits found for the most visited action, $c_{visit} = 50$, and $c_{scale} = 1.0$.
The Sequential Halving algorithm (Karnin et al., 2013) is added on top to minimize simple regret in the root node. Given several simulations, n of Sequential Halving, the search of A_{n+1} is divided into several phases. The number of phases is calculated using the $\log_2(m)$ where m is the number of sampled actions to consider. Half of the considered actions are dropped in each phase, with all actions being equally visited. At each phase, a set of top actions that maximize Equation 2.30 is kept, leading to the final single action, A_{n+1}. To allocate the number of simulations per phase, the

total simulation budget is divided by the number of phases to determine the number of simulations allocated per phase. This result is then further divided by m_{phase} to calculate the number of simulations assigned per action for the current phase. The number of visits for each action for a given phase can be calculated:

$$visits = \left\lfloor \frac{n}{\log_2(m) m_{phase}} \right\rfloor \tag{2.32}$$

which divides the total budget of simulations by the number of phases, resulting in the number of simulations per phase. Then, it is divided by m_{phase} to find the number of simulations per action for the current phase. The higher the simulation number, the more *visits* for each action, thereby leading to a tree with greater depth and more comprehensive information.
After completing the search, the Sequential Halving algorithm is also utilized for selecting actions by the agent in the environment. For learning an improved policy, instead of comparing the policy network with a categorical distribution that relies on the visit counts of the root actions, Gumbel AlphaZero proposes an improved policy π'. For the construction of the improved policy π', a vector of completed Q-values is built in a way that ensures zero advantage to the unvisited actions:

$$\text{completedQ } (a) = \begin{cases} Q(a) & \text{if } N(a) > 0 \\ \hat{V}, & \text{otherwise} \end{cases} \tag{2.33}$$

where $Q(a)$ (similar to AlphaZero) represents the estimated q values for the actions and is calculated as the mean action value over the simulations. For constructing the $\hat{V}$, they propose to either replace with $V = \sum_a \pi(a)q(a)$ or approximate using the value network or using a mixed value approximation (Equation 2.34) that is especially recommended for off-policy cases.

$$\hat{V} := \frac{1}{1 + \sum_b N(b)} \left(V + \frac{\sum_b N(b)}{\sum_{b \in \{b : N(b) > 0\}} \pi(b)} \sum_{a \in \{a : N(a) > 0\}} \pi(a) Q(a) \right). \tag{2.34}$$

where V is the value approximation from the value network and b represents the children (i.e. the actions). For the mixed value approximation in the case of zero visits,

$$\sum_b N(b) = 0 \overset{(2.34)}{\Longleftrightarrow} \hat{V} = \frac{1}{1} V. \tag{2.35}$$

so the prediction of the value network is used as $\hat{V} == V$. Finally, the improved policy may be defined as :

$$\pi' = \text{softmax}(\text{logits}^{\pi} + \sigma(\text{completed}\, Q)) \tag{2.36}$$

Considering the action selection in non-root nodes in the search tree, improved policy π' is utilized, and a deterministic action selection approach is applied that seeks to minimize the MSE between π' and the normalized visit counts. This action selection strategy can be formulated as:

$$\arg\max_{a} \left(\pi'(a) - \frac{N(a)}{1 + \sum_b N(b)} \right) \tag{2.37}$$

Finally, to encourage network π to improve towards π', they utilized the Kullback-Leibler divergence (KL divergence) between the network's current policy π and the improved policy π'. This constructs the completed loss l for policy π :

$$L_{completed}(\pi) = \text{KL}\left(\pi' \| \pi\right) \tag{2.38}$$

KL divergence replaces cross-entropy loss in Equation 2.29 that is utilized in AlphaGo Zero and AlphaZero.

2.1.6 Natural Language Processing

Computational linguistics or Natural Language Processing (NLP) is a sub-field of AI that has its roots in the late 1940s after World War II, as reported by Hirschberg and Manning (2019) and Nadkarni et al. (2011). They also report that early NLP systems used rule-based systems with computer scientists manually reporting language rules and vocabulary to enable computers to process text. This approach was very limited since human language is incredibly complex and nuanced. Subsequently, researchers explored statistical and probabilistic methods, developing sophisticated ML techniques that brought about significant advancements in the field. There are many applications of NLP using ML, namely, machine translation, dialogue systems, sentiment analysis, generation of speaker state, etc.

2.1.6.1 Sequence-to-Sequence

Sequence-to-Sequence (Seq2Seq) is a specific type of neural network commonly used in NLP tasks where an input sequence $(x_1, .., x_n)$ is mapped into an output sequence $(y_1, .., y_m)$. The length of the input and output sequence may not be the same, $n \neq m$. Seq2Seq models are usually based on Encoder-Decoder architecture, which includes:

1. Encoder: Processes the input sequence to summarize all the information into an encoded representation and emits a context vector C, of fixed length. Context vector encapsulates the input sequence details.
2. Decoder: Consumes the context vector C of the input sequence and generates an output sequence y by computing the probability over the given output sequence $(y_1, .., y_m)$.

Attention Bahdanau et al. (2014) observed one limitation in previous Seq2Seq architectures. A context vector C of a fixed length occasionally struggled to effectively summarize information from long sequences [3]. To address this issue, Bahdanau et al. (2014) introduced an attention mechanism to focus on the most important parts of the sequences combined with a RNN architecture for both the encoder and the decoder. The core idea of Attention is to identify the relevant parts of a sequence and allow the model to pay attention to them [4]. In this work, the encoder converts the input sequence $(x_1, .., x_n)$ into a sequence of annotations $(h_1, .., h_n)$. The context vector, then, is computed as a weighted average over all input feature vectors:

$$c_t = \sum_{j=1}^{n} \alpha_{tj} h_j \tag{2.39}$$

where a is the attention vector for each timestep t. The attention vector is calculated using an attention function that produces normalized weights. These weights are

[3] In a Neural Machine Translation task

[4] Attention can be applied to non-sequential data as well, but this is not of interest to this thesis

normalized into probabilities using softmax (Bridle, 1990).

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})} \tag{2.40}$$

where,

$$e_{ij} = a(s_{i-1}, h_j) \tag{2.41}$$

is the score of alignment, between inputs around position j and the output at position i, where s_{i-1} is the hidden state from the RNN. There have been several proposed attention functions, such as Cosine (Graves et al., 2014), Additive (Bridle, 1990), General (Luong et al., 2015) and Dot-product (Luong et al., 2015).

2.1.6.2 Transformers

Among the most notable breakthroughs in the NLP field is the advent of Transformers (Vaswani et al., 2017), an architectural innovation by Google Brain, widely incorporated in Seq2Seq models. Transformers aimed to overcome the limitations of previously established methods for sequence modeling, such as RNNs and CNNs, in terms of both efficiency and performance. This was achieved with parallelization and capturing long-range dependencies within sequences. They outperformed all previous works in two machine translation tasks while requiring notably less training time.
To gain a deeper understanding of Transformers, we will dive into some key innovations presented by Vaswani et al. (2017), including scaled-dot product attention, self-attention, and multi-head attention mechanism. Next, we will discuss how these elements are utilized to form a transformer block. Finally, we will provide a summary of the overall transformer architecture.

Scaled Dot Product Attention In Transformers, a novel attention mechanism called ”Scaled Dot-Product Attention” is introduced. Dot-Product attention (Luong et al., 2015) presents a weakness of vanishing gradients, while the dot products are getting high values, resulting in extremely small gradients for the softmax function (Vaswani et al., 2017). To overcome this effect, they scale the dot products by the root of the encoder output dimension.
They also introduce query, key, and value terms and update the definition of the attention function as the one that maps a query and a set of key-value pairs to an

output. Let Q, K, V be the matrices of query, key, and value, respectively, the scaled dot-product can be defined as:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.42}$$

where d_k is the dimension of queries and keys.

Self-Attention The general attention mechanism works between two sequences. However, in Transformers, a novel attention mechanism known as Self-Attention is introduced to capture relationships among elements within a single given sequence. This practically means that the matrices that depict query, keys, and values are the same matrix ($Q = K = V$). This approach holds less computational complexity while being able to capture dependencies in long sequences. This is particularly useful for creating a representation of a sequence.

Multihead-Attention In Transformers, a novel attention layer Multi-Head Attention layer (MHA) (Equation 2.43) is introduced, which stacks multiple Scaled Dot-Product Attention layers (Equation 2.42) for h head times that may run in parallel. Moreover, each query, key, and value is multiplied by a projection weight matrix, namely, W^Q, W^K, and W^V. The result of this multiplication is split into the number of heads to which a Self-Attention is applied. Finally, the result is concatenated and transformed using a square weight matrix W^O:

$$\text{MultiHead}(Q,K,V) = \text{Concat}\begin{pmatrix} \text{Attention}\left(QW_1^Q, KW_1^K, VW_1^V\right) \\ \ldots, \\ \text{Attention}\left(QW_h^Q, KW_h^K, VW_h^V\right) \end{pmatrix} W^O \tag{2.43}$$

where $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ are the projection weight matrices. This technique allows for a stronger and more complex output representation while each head has the ability to attend to different segments of the input.

Transformer Block The transformer blocks are the basic component of the Transformer architecture. The input sequence can not be raw text, so a preparation step creates the actual input of the Transformer Block. The input sequence is employed for producing embeddings and positional encodings. Since the Transformers abandoned the usage of recurrence, the model inherently lacks an understanding of the position of each element in a sequence. Utilizing Positional Encodings (PE), they successfully capture positional information inside the sequence, helping the model track the word order. The summation of the embeddings and the positional encodings is the encoder's final input. Specifically, a block consists of a Multihead Self-Attention layer and a Fully Connected Feed-Forward layer. Both layers are followed by residual connections (He et al., 2016) around the previous layers and a Normalization layer (Ba et al., 2016). The whole architecture is presented in Figure 2.3.

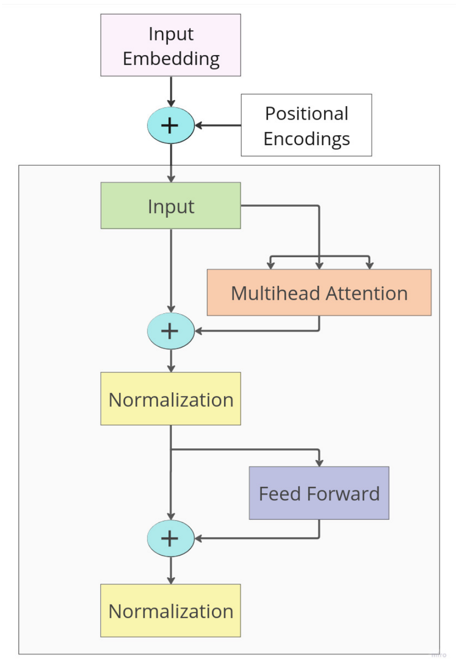


Figure 2.3: Transformer Block by Vaswani et al. (2017)

General Architecture The original Transformer, as proposed by Vaswani et al. (2017), consists of an encoder-decoder architecture. The encoder part comprises a stack of 6 identical Transformer Blocks. The decoder also comprises a stack of 6 identical Transformer Blocks with slight modifications. The Transformer is auto-regressive, while at each timestep, it consumes its previous predictions. The decoding process is initialized with a Masked Multi-head attention step, which prevents positions from attending to subsequent positions. Subsequently, the decoder incorporates a cross-attention step, which attends to the encoder output. The cross-attention is calculated following the formula in Equation 2.43, but in contrast with self-attention, it employs the encoder output as both queries and keys, with the decoder's previous layer output serving as values. This mechanism introduces information from the input sequence to the layers of the decoder. Finally, the final layer is a fully connected feed-forward layer. Similarly to the encoder, all three layers are followed by residual connections around the previous layers and a normalization layer. The complete transformer architecture can be shown in Figure 2.4.

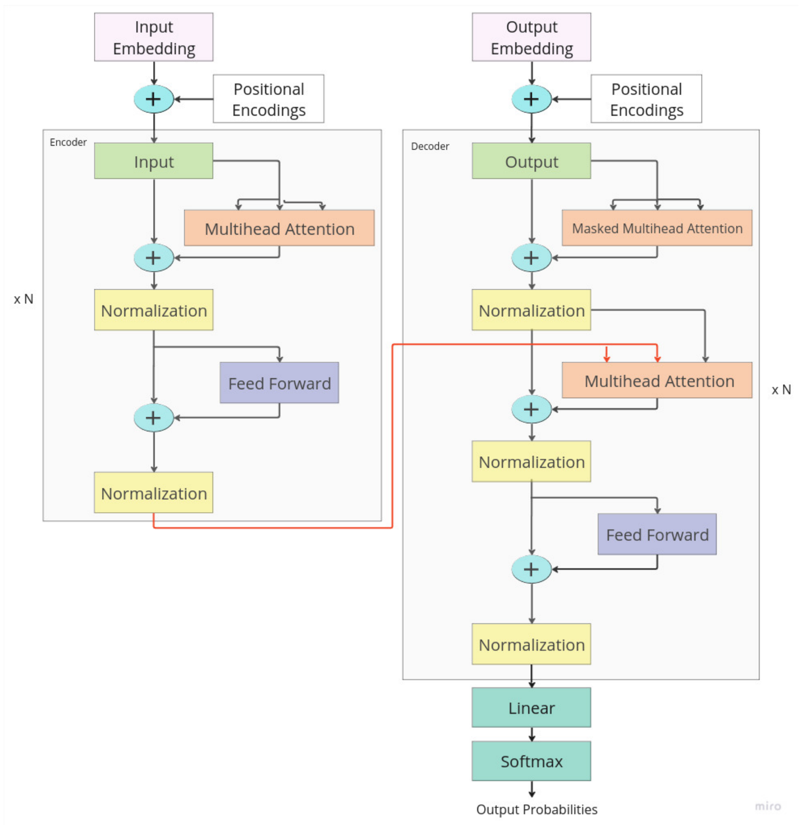


Figure 2.4: Transformer Architecture by Vaswani et al. (2017)

2.2 Literature Review

In the past decades, numerous approaches have been introduced for addressing the MSA problem. Pairwise Alignment is considered the alignment between a pair of two sequences in contrast with MSA, which involves the alignment of more than two sequences. Pairwise Alignment can be considered the precursor of MSA. Dynamic programming has proven very effective for pairwise alignment, but it can be

time-consuming when more sequences are requested to be aligned. On the other hand, pairwise alignment may be insufficient in bioinformatics and computational biology since there is often a need for aligning more than two sequences simultaneously. This has led to extensive research on developing more efficient and flexible algorithms that can handle multiple sequences.
There have been two main concepts in the Sequence alignment topic-related techniques, namely, global and local alignment.

- Global alignment aims to find the best alignment over the entire length of two sequences.
- Local alignment, on the other hand, focuses on specific regions. As proposed by Smith and Waterman (1981), segments of high similarity can be identified inside sequences. Local alignment is considered beneficial when there are significant differences in the lengths of individual sequences.

Another categorization of the traditional problem approaches can be:

- Exact, which guarantees finding optimal solutions and is usually based on dynamic programming algorithms,
- Progressive approaches,
- Iterative approaches,
- Consistency-based approaches,
- and others.

2.2.1 Exact Alignment

One of the most famous and historic dynamic programming algorithms for solving pairwise alignment is proposed by Needleman and Wunsch (1970) and is an exact alignment. The complexity of this method comes to $O(2^k * n^k)$ where k is the number of sequences and n is the length of the sequences. This algorithm consists of three steps. First, a matrix is initialized, then filled using matching and penalty values, and finally, the sequence alignment is constructed by finding the best path

using traceback [5]. Needlman-Wunsch algorithm belongs to the pairwise global alignment methods family and was later extended by Murata et al. (1985) to achieve the alignment of three sequences.
In the following year, Bacon and Anderson (1986) claimed that functional and chemical characteristics of amino acids could be valuable for calculating similarities in sequence alignments. They propose a non-binary similarity measure. A set of 8 amino acid attributes is selected to pass this information. These are structural preference with subcategories α-helix, β-strand, and β-turn, amphiphilicity with subcategories hydrophobic and polar, size, and finally, charge with positive and negative subcategories. Each property is set as an axis in a Cartesian coordinate system. Finally, the similarity score is calculated by scaling the rounded value of the Euclidean distance minus a positive predefined scaled and rounded constant in the range $[\![0, 9]\!]$.
Carrillo and Lipman (1988) introduced an approach for reducing the search space. They claimed that every MSA inherently imposes pairwise alignments on all sequence pairs, akin to projecting a path in two-dimensional space. Their strategy also employs a bounded score for MSAs pairwise combinations (Konagurthu & Stuckey, 2006).

2.2.2 Progressive Alignment

Progressive alignment is a step-by-step hierarchical alignment technique. Sequences are initially pairwise aligned, and using these intermediate alignments, and then gradually, more sequences are incorporated into the alignment until all sequences are aligned into a final alignment. The progressive alignment method is introduced by Feng and Doolittle (1987). This method involves the computation of a distance matrix using the Needleman and Wunsch algorithm (Needleman & Wunsch, 1970) between pairwise alignments and then producing a guided tree to control the alignment order. This way, the algorithm prioritizes aligning the most similar sequences first before proceeding to the less similar ones.
A significant tool, ClustalW, was released by Thompson et al. (1994) and later became the tenth most cited scientific paper of all time by 2014 (Noorden et al., 2014). This tool's initial goal was to upgrade the performance of the progressive multiple alignment method (Feng & Doolittle, 1987) without sacrificing efficiency and speed. In ClustalW, it is claimed that the selection of gap penalties is essential, especially in divergent sequences. This is because, in closely related sequences, any scoring method works reasonably well due to the dominance of identical residues. To over-

[5]A detailed example of Needlman-Wunsch algorithm can be found in section A.2.

come this difficulty, they propose a dynamic gap strategy where the gap penalty is determined by residue and position. At the same time, they also apply Affine Gap Penalization (see subsection 3.2.2).
The main drawback of progressive alignment can be summarized by the expression "once a gap, always a gap". Progressive alignment methods lack the option for refinement and follow a one-way procedure, meaning that prior decisions cannot be undone. This inflexibility may lead to misalignments due to greediness.

2.2.3 Iterative Alignment

Iterative methods use progressive alignment to compute a sub-optimal intermediate solution and then optimize this solution by modifying the alignment (Pevsner, 2009). These methods do not guarantee an optimal solution but are less sensitive and more flexible than progressive alignment. They are generally an extension of progressive alignment and can overcome its limitation, which is that once sequences are aligned, the algorithm cannot revisit or correct earlier alignment decisions. MAFFT tool by Katoh et al. (2002) includes both progressive alignments using a guide tree and iterative refinement methods for the entire process, with adjustments to the positions of gaps and insertions to improve the alignment. To achieve faster alignments while maintaining high accuracy, it utilizes Fast Fourier Transform (FFT) to rapidly find homologous segments. They apply modifications in the weight similarity matrix and gap penalties. Other examples of tree refinement include MUSCLE tool (Edgar, 2004), Dialign tool (Morgenstern, 2004), Praline tool (Simossis & Heringa, 2005), PRIME tool (Yamada et al., 2006) PRRP (Gotoh, 1996), etc.

2.2.4 Consistency-Based Alignment

Consistency-based methods utilize information about the entire alignment as it is being generated to guide the pairwise alignments. This information refers to the posterior probability of a residue aligning with another one. A consistency-based method was introduced by Notredame et al. (2000), which offers slower but more accurate alignments with the goal of overcoming the significant pitfalls arising from the inherent greediness of this algorithm. This approach, named Tree-based Consistency Objective Function for alignment Evaluation (T-Coffee), also generates intermediate alignments but considers information from all of the sequences during each alignment step, not just the ones currently being aligned. T-Coffee assigns weights in each pair of aligned residues, indicating the consistency of alignment with residues from all the other sequences. Other approaches in this category are Probcons (Do et al., 2005), Probalign (Roshan & Livesay, 2006), L-INS-i variant of MAFFT as reported by Llados et al. (2021), etc.

2.2.5 Other Approaches

Another interesting approach is the Hybrid Multi-objective Artificial Bee Colony (HMOABC) by Rubio-Largo et al. (2016). Two objective functions are employed to cover the quality and consistency of the alignment: Weighted Sum of Pairs with affine gap penalties and Total Column score. This algorithm is based on a swarm-based evolutionary algorithm, the Artificial Bee Colony (ABC) algorithm, created by Karaboğa (2005), which imitates the behavior of bees. It is worth noting that they utilize Kalign2 (Lassmann et al., 2009) as a part of their algorithm to partially align the sequences. Similarly, Aniba et al. (2010) employs existing aligners as a foundational element in their algorithm using a traditional AI method, Decision Trees. They claim that widely used aligners such as ClustalW, Dialign, MAFFT, MUSCLE, Kalign, and ProbCons present variable strengths and weaknesses based on the input characteristics. To overcome this difficulty, they developed AlexSys, a decision tree-based tool that aims to predict the best aligner tool depending on the input raw alignment.
Edelkamp and Tang (2015) uses MCTS for approaching the MSA task. This time, in contrast with progressive alignment trees, the root node is empty, and the other nodes sequentially represent a column in alignment from left to right. Specifically, the children nodes of the root node are all the possible combinations of the first column. After each expansion, nodes are created that contain all the possible combinations for the next column to the right (Figure 2.5).

2.2.6 Deep Reinforcement Learning Approaches

RL has been first introduced by Mircea et al. (2014) into the field of MSA. Their algorithm utilizes Q-learning, an off-policy method, combined with the Needleman and Wunsch (1970) algorithm. They experience both linear and affine gap penalization. Their method combines RL with progressive alignment by training an agent to choose the order of the sequences to align. To overcome the rigidness of aligning a sequence to a set of already aligned sequences, the alignment happens between the

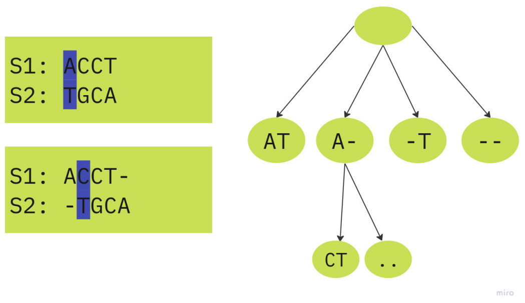


Figure 2.5: MCTS for MSA by Edelkamp and Tang (2015)

sequence to be aligned and a profile for the set. In this MDP, an action represents the next state to be selected, and the final goal is to find a path from an initial to a final state, building a tree. The agent can pick more than once the same sequence, but in such cases, it receives a high penalty to avoid exploring invalid paths. Again, Q learning is applied, and for action selection, the ϵ-Greedy mechanism is utilized with a look-ahead step.
Next, Jafari et al. (2019) with the same MDP as Mircea et al. (2014) designs a Long short-term memory (LSTM) network for Q Values and Asynchronous-Advantage-Actor-Critic (A3C) algorithm instead of Q learning.
Joeres (2021) utilizes the same MDP as Mircea et al. (2014) and Jafari et al. (2019) and implements the SARSA algorithm, DQN algorithm (Mnih et al., 2013), A3C, and UCT algorithm, which represents an enhancement of the MCTS method.
RALIGN, in (Ramakrishnan et al., 2018), is based on a different MDP than the previous RL approaches, where the agent moves the nucleotides left or right at each step. Moving a nucleotide involves introducing a gap at its location and prompting the current nucleotide and all following nucleotides in the designated direction to shift one position forward. Each state stands to an alignment, and the agent's reward is calculated using the Sum Of Pairs metric, SP (see subsection 3.2.1). Every episode contains a fixed number of moves, and the final state selected is the one where the agent achieved the highest score. This algorithm is trained using A3C algorithm. Additionally, for sequences with greater length than the fixed model expected length, a sliding-window heuristic is introduced, applying the model to all windows within the initial alignment.
DPAMSA, a DRL sequence alignment tool, was introduced by Liu et al. (2023) recently. In this approach, the agent applies gaps at each timestep per column. The state is divided into an aligned part for all the previously aligned parts and an unaligned one for the current and all the successive columns. The agent is trained using the Q network. The state is flattened into one embedded sequence accompanied by positional encodings. They also use Self-Attention followed by Multilayer

perceptron for calculating the Q values.
None of the aforementioned RL approaches apply a comprehensive comparative analysis of widely recognized MSA tools (Clustal Omega, MAFFT, MUSCLE) using the specific SP score parameters that these tools employ for their self-evaluation. This may lead to an increased performance in the SP score of RL agents as seen in (Joeres, 2021) compared to the other scores as Totally Conserved or Column (TC) score where no special parameters differentiate the objective.

Chapter 3

MSA Definition and Score Metrics

3.1 Problem Definition

This section presents a clear definition of the MSA problem. The problem is defined for DNA sequence MSAs but can be easily adapted to RNA and protein sequences. DNA is a double-stranded, helical molecule constructed from nucleotides, also known as mononucleotides. These nucleotides comprise three essential components: a phosphate group, a pentose sugar, and a nitrogenous base. The nitrogenous bases can be categorized into two groups: purines and pyrimidines. There are two purines, adenine and guanine, and three pyrimidines, cytosine, thymine, and uracil, that are present in nucleic acids. In DNA, there are four distinct types of nucleotides, namely adenine (A), thymine (T), guanine (G), and cytosine (C) (Klug et al., 2012) .
Let Σ_{DNA}= $\{A,T,G,C\}$ be the set of available letters that can be found in DNA sequences.
An updated set will be defined, including the hyphen symbol $-$, corresponding to a gap between two letters. Set Σ'_{DNA} can be defined as the union of two sets :

$$\Sigma'_{DNA} = \{-\} \cup \{A,T,G,C\}$$

Alternatively, we can rewrite it as:

$$\Sigma'_{DNA} = \{-\} \cup \Sigma_{DNA}$$

Let m be the number of sequences requested to be aligned : $Seq_1, Seq_2, Seq_3, \ldots, Seq_m$ where $m > 2$. Each sequence Seq_i has an arbitrary length l_i and is represented by a tuple of letters from set Σ_{DNA} so that each element of tuples $Seq_i \in \Sigma_{DNA}$. Moreover, each element within a sequence will be denoted by the symbol c [1].

$$
\begin{aligned}
Seq_1 &: \left(c_1^1, c_2^1, c_3^1, \ldots, c_{l_1}^1\right) \\
Seq_2 &: \left(c_1^2, c_2^2, c_3^2, \ldots, c_{l_2}^2\right) \\
&\vdots \\
Seq_m &: \left(c_1^m, s_2^m, c_3^m, \ldots, c_{l_m}^m\right)
\end{aligned}
$$

For achieving standardization of sequence lengths, the following method is employed. Given a set of sequences $Seq_1, Seq_2, ...Seq_m$, maximum length l_{max} is utilized for padding all sequences to it. Sequences are padded to the left using the hyphen gap symbol $-$.

$$
\begin{aligned}
l_{max} &= max(l_1, l_2, ..lm) \\
Seq_1 &: \left(c_1^1, c_2^1, c_3^1, \ldots, c_{l_{max}}^1\right) \\
Seq_2 &: \left(c_1^2, c_2^2, c_3^2, \ldots, c_{l_{max}}^2\right) \\
&\vdots \\
Seq_m &: \left(c_1^m, c_2^m, c_3^m, \ldots, c_{l_{max}}^m\right)
\end{aligned}
$$

The output of the MSA will be an updated set of $Seq'_1, Seq'_2, ..., Seq'_m$ tuples, derived from adding gaps in selected positions of the initial $Seq_1, Seq_2, ..., Seq_m$. In this case, each sequence Seq'_i will be a tuple of elements that belong to the updated set Σ'_{DNA}. It is essential to find a sophisticated way to select the positions to add gaps to get an optimal MSA. The output sequences always have a greater or equal length with the initial $Seqs$, $|Seqs| \leq |Seqs'|$.

3.2 Score Metrics

Metrics are essential for assessing the quality of MSAs. In an ideal scenario, a higher score signifies a stronger biological relevance. One of the significant challenges in

[1] This notation is chosen to prevent any possible confusion with the state notation introduced earlier in Section 2.1.2

addressing the MSA problem is that it is missing a global function for quantifying the quality of an alignment. Over the years, researchers have proposed different score measures or even a combination of those. Defining a score metric for MSA that can be accurate and easy to calculate can be tricky. The current section will define three of the most important score metrics for MSA and provide descriptions of some variations of those.

3.2.1 Sum of Pairs Score

The Sum of Pairs (SP) score, introduced by Carrillo and Lipman (1988), stands out as one of the most commonly used metrics for quality alignment calculation. This scoring method is widely adopted in the context of MSA and is utilized by popular alignment tools such as Clustal Omega (Sievers et al., 2011), MAFFT (Katoh et al., 2002), MUSCLE (Edgar, 2004), etc. The score can be formulated as:

$$\mathcal{SP} = \sum_{i=1}^{l_{max}} \sum_{j=1}^{m-1} \sum_{k=1}^{m} \mathrm{sp}(c_i^j, c_i^k) \quad \text{with} \quad \mathrm{sp}(c_i^j, c_i^k) = \begin{cases} gg, & \text{if } c_i^j = c_i^k = - \\ ll, & \text{if } c_i^j = c_i^k \\ lg, & \text{if } c_i^j = - \text{ or } c_i^k = - \\ ldl & \text{if } c_i^j \neq c_i^k \end{cases} \tag{3.1}$$

where l_{max} is the total number of columns, m is the number of sequences, $\mathrm{sp}(c_i^j, c_i^k)$ represents the score for pair (c_i^j, c_i^k) (Jafari et al., 2019). Moreover, the constants gg, ll, lg, and $ldl \in \mathbb{R}$ are parameters that represent different conditions.
It's important to note that these scoring parameters may vary in the literature, and different studies use different values for these conditions. For example, in Thompson et al. (1994), each residue matching (ll) contributes 1 point, and for any other case, it is 0. Lall and Tallur (2023), on the other hand, for each residue matching, the output is 1 point, but for mismatching, there is a penalty of 0.6. Jafari et al. (2019) and Joeres (2021) propose 2 points for each residue match, -2 points for each residue mismatch (ldl), -1 point for each residue-gap occurrence (lg), and 0 points in any other case. Finally, Liu et al. (2023) proposes 4 points for each residue match and -4 points for every mismatch.
In the evaluation process, particularly when comparing with other studies-aligners, it's essential to be aware of those scoring parameters. Ignoring them can result in inconsistent comparisons based on different criteria.

3.2.1.1 Weighted Sum of Pairs

In some cases, SP score can be expanded into a Weighted Sum of Pairs (WSP) (Rubio-Largo et al., 2016) where each pair (c_i^j, c_i^k) can be multiplied with a weight if it is believed to show more interest. In general, WSP allows inserting specific criteria or preferences into the scoring of residue pairs in an MSA. For example, it may be more exciting or biologically relevant to catch a match after 20 continuous pair matches compared to finding a match after only three matches.

3.2.2 Gap Penalty

Gap Penalty is a negative score and is a way for scoring MSAs. In general, adding a gap while applying an MSA is expected and can indicate an insertion or a deletion of a residue, and by adding gaps, an optimal alignment can be found. Based on the hypothesis that excessively fragmenting the sequences in an MSA can diminish the information contributed by the sequences, the use of gap penalization is intended to reduce the excessive accumulation of gaps within an alignment. This cost is typically subtracted from the $\mathcal{SP}$ score and cannot serve as an objective on its own. To adjust the proportion of gaps added, inserting gaps is penalized using four methods, namely, constant, linear, convex, affine, and profile-based.

Constant Gap Penalization represents the simplest way of gap penalization. In this case, a single gap penalty constant is applied each time a sequence is split using a gap. We define this cost as the gap opening (denoted as go) cost.

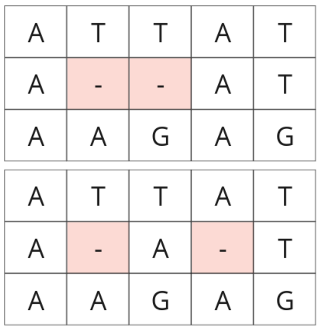
$$\mathcal{GC} := d_o go \tag{3.2}$$

where d_o is the total number of splits of the sequences within the MSA.

Linear Gap Penalization increases linearly with respect to the number of gaps added in the MSA. We may define ga as the cost of any gap. Note that in this method, the total penalty will be the same for a single large gap with many small gaps. For example, in Figure 3.1, both options of gap insertions will lead to the same gap penalty.

$$\mathcal{GC} := d_a ga \tag{3.3}$$

where d_a is the total number of applied gaps within the MSA.



A	T	T	A	T
A	-	-	A	T
A	A	G	A	G

A	T	T	A	T
A	-	A	-	T
A	A	G	A	G

Figure 3.1: Example of two different gap insertions that will be equally penalized in linear gap penalization

Convex Gap Penalization utilizes a concave gap penalty function as a more advanced version of Linear Gap Penalization. This method is not widely employed since it is slower to calculate, and there is uncertainty regarding its potential to improve the accuracy of an alignment (Ranwez & Chantret, 2020).

Affine Gap Penalization is the most widely used method among the previous ones. (Rasmussen & Krink, 2003). Konagurthu and Stuckey (2006) claims that affine gap costs have been shown to be more accurate than linear gap penalties. This approach diverges gaps into two groups, the opening and extending gaps. For every instance where a sequence in the alignment is split, indicating the presence of one or more gaps between two letters, a cost of $go + (\lambda - 1)ge$ is calculated, where go is the gap opening cost and ge is the gap extension cost, and λ the gap length. Based on this logic, we define the $\mathcal{GC}$ for being applicable to the whole MSA such as:

$$\mathcal{GC} := \sum_{i=1}^{l_{max}} \sum_{j=1}^{m} gc(c_i^j) \quad \text{with} \quad \text{gc}(c_i^j) = \begin{cases} 0 & \text{if } c_i^j \neq - \\ ge, & \text{else if } i > 1 \text{ and } c_{i-1}^j = - \\ go, & \text{else} \end{cases} \tag{3.4}$$

Similarly to $\mathcal{SP}$ parameters go and ge vary in literature. For example, in (Rubio-Largo et al., 2016), following the Blocks substitution matrix (BLOSUM) 62, gap opening is proposed to be six and gap extension 0.85. In the work of Lall and Tallur (2023), the gap opening cost is 1, while the gap extension cost is 0.4.

3.2.3 Totally Conserved Score

The Totally Conserved or Column (TC) Score refers to the sum of the columns of a sequence alignment where all the residues (nucleotides or amino acids) match precisely. It can be formulated as:

$$\mathcal{TC} := \sum_{i=1}^{l_{max}} tc_i \quad \text{with} \quad tc_i = \begin{cases} 1 & \text{if all residues match} \\ 0 & \text{else} \end{cases} \tag{3.5}$$

where l_{max} is the total number of columns (see section 3.1).

Chapter 4

IntellAlign Materials and Methods

This section presents and extensively discusses the materials and methods for IntellAlign, a novel approach for solving MSA.

4.1 RL Environment Construction

4.1.1 State Definition

A matrix of letters initially represents an MSA. In an MSA matrix, each row corresponds to one sequence, and each column represents a position in the alignment. For this reason, a proper data preprocessing stage is required to utilize them in any deep learning application. The representation is based on traditional NLP encoding methods where each letter corresponds to a token in a predefined dictionary of words. These letters are then mapped to unique integers. In our architecture, the sequences are concatenated into a single flattened sequence with an end token between sequences to distinguish them. Let $< END >$ be the end token. Each sequence is followed by $< END >$ at the end. Moreover, all sequences are padded to the same length l_{max}, which will be the maximum sequence length found in the alignment, using $< GAP >$ tokens. Next, we add a $< STOP >$ token at the beginning of the alignment. The purpose of the $< STOP >$ token is to provide the agent with the option to terminate the alignment at any point rather than being

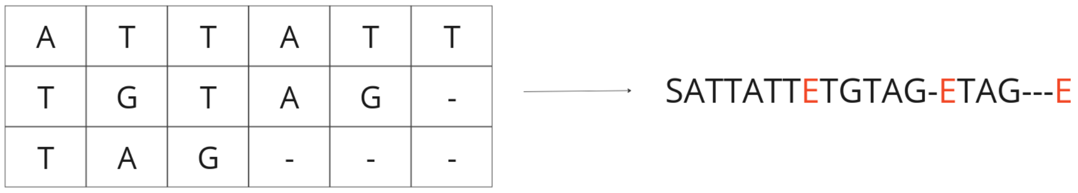


Figure 4.1: Encoding Scheme

limited to a predefined number of steps. $< STOP >$ token utility will be further explained in the next subsection. The resulting sequence represents the state of the environment at timestep t and is defined as:

$$\begin{aligned} S_t : \Big(& STOP, c_1^1, c_2^1, \ldots, c_{l_{max_t}}^1, END, \\ & \ldots, \\ & c_1^m, c_2^m, \ldots, c_{l_{max_t}}^m, END \Big) \end{aligned} \tag{4.1}$$

The maximum sequence length l_{max_t} may also increase during different timesteps. The final representation is a flat sequence consisting of all sub-sequences, as shown in the example in Figure 4.1 [1].

4.1.2 Action Space

In the given scenario, available actions may categorized roughly into two categories: first involves inserting a gap, which entails selecting a specific position to introduce a gap, causing the remainder of the sequence to shift to the right. The second one allows the agent to finalize the alignment by stopping the alignment process. The action space $\mathcal{A}_t$ (Equation 4.2) is represented as a tuple of an arbitrary length at each step t, precisely matched with the length n of S_t.
Let $\mathcal{A}_t$ (equation 4.2) be the action space, represented as a tuple of integers ranging from 0 to $n-1$, of an arbitrary length at each step t, precisely matched with the length n of S_t. At each timestep t, the agent picks an action from $\mathcal{A}_t$ where action 0 maps to the $< STOP >$ token indicating the end of the game while all other integers

[1] To enhance the figures' visual appeal and comprehensibility, these conventions are applied to the MSA representation, where 'S' signifies the stop token $< STOP >$, '-' denotes the gap token $< GAP >$, and 'E' is equivalent to $< END >$ token, highlighted in red color

indicate the position for adding a gap in S_t. See section A.3 for an example case.

$$\begin{aligned} n &= (l_{max}+1)\cdot m+1 \\ \mathcal{A}_t &:= (0,1,\ldots,n-1) \end{aligned} \tag{4.2}$$

4.1.3 Transition

For a given action A_t, at timestep t, the state S_t transitions to $S_{t+1} = \mathrm{F}(S_t, A_t)$ where $\mathrm{F} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, in a deterministic manner following the state transition function.

$$Insert(S_t, i) := S_t[0:i] + \mathrm{GAP} + S_t[i+1:n] \tag{4.3}$$

$$S_{t+1} = \mathrm{F}(S_t, A_t) := \begin{cases} S_t & \text{if } A_t = 0 \\ Insert(S_t, A_t) & \text{if } A_t \neq 0 \end{cases} \tag{4.4}$$

At each timestep t, it is essential to ensure that all sub-sequences maintain the same length inside S_t while adding new gaps. A specific strategy is followed to extend Equation 4.4 to achieve this. After inserting a gap to a chosen point on a sequence, all sequences are padded with a *GAP* at the end, except for the one requested to add a gap. Through this process, all sequences maintain the same length. If, as a side effect of the previous strategy, the last index of all sequences results being a $< GAP >$, the last column is dropped, and: $|S_{t+1}| = |S_t|$, $l_{max,t+1} = l_{max,t}$. We present the complete transition strategy in Algorithm 2 [2].

4.1.4 Terminal Gaps

Terminal Gaps are defined as the ones that are attempted to be added at the end of the alignment. Adding gaps at the end of a sequence is considered undesired and useless for the alignment. However, drawing inspiration from Schrittwieser et al. (2020), we opt not to restrict this action to pass domain knowledge of MSA to the agent. Therefore, for the sake of flexibility and for maintaining loose rules for the agent, we allow the agent to add gaps at the end of the MSA, and this will transition the state S_t into a state S_{t+1} with a new column at the end full of gaps. The game will automatically end if the agent applies a terminal gap with a high penalty value.

[2] see section A.4 for a visual example of state transition

4.1.5 Complete Column Gaps

Complete Column Gaps (CCG) are the ones that result in a column where all elements are gaps. Columns where all elements are gaps are considered undesired and offer no insight for the MSA. Similarly to Terminal Gaps, we do not restrict this movement, but the game automatically ends after this transition accompanied by a high penalty value. This functionality operates as a boolean hyperparameter in our algorithm and thus is not utilized in all the experiments.

Algorithm 2 State Transition

```
Input: $A_t \in [0,n] \cap \mathbb{Z}$
Input: $m \geq 3$                                   ▷ number of sequences
Input: $S_t$
Input: $l_{max}$
$S_{t+1} \leftarrow S_t$
if $A_t \neq 0$ then                                ▷ if the agent doesn't decide to stop
    $picked\_seq \leftarrow \lfloor \frac{A_t - 1}{l_{max}} \rfloor$          ▷ sequence that the agent wants to modify
    $last\_e \leftarrow S_t[l_{max}(picked\_seq + 1) - 1]$     ▷ last element of that sequence
    $S_{t+1} \leftarrow \text{Insert}(S_{t+1}, A_t)$                  ▷ Insert gap
    if $last\_e = \text{GAP}$ then
        $S_{t+1} \leftarrow \text{Remove}(S_{t+1}, l_{\max}(picked\_seq + 1))$
    else
        for $i = 0, ..., m-1$ do
            if $i \neq picked\_seq$ then                  ▷ Padding Step
                $S_{t+1} \leftarrow \text{Insert}(S_{t+1}, l_{\max}(i+1) + i)$   ▷ Insert gap
            end if
        end for
    end if
end if
```

4.1.6 Reward

We design an episodic reward, $\mathcal{R}_s^a$ so that the agent would get a final reward at the end of each episode. The reward depends only on the final state of the episode. Selecting the reward function is challenging, as we aim to compete with other aligners. This challenge is compounded by the lack of a universally accepted objective function for addressing the MSA problem. We propose a multi-objective function compounded by the two main scoring schemes, namely, $\mathcal{SP}$ and $\mathcal{TC}$. The SP score can be extended with affine gap penalization by setting $ge \in \mathbb{R}, ge > 0$, $go \in \mathbb{R}, go > 0$.

$$\mathcal{R}_s^a := \mathcal{SP}_s - \mathcal{GC}_s + \alpha \mathcal{TC}_s \tag{4.5}$$

that can be expanded as :

$$\begin{aligned}
\mathcal{R}_s^a &:= \sum_{i=1}^{l_{max}} \left(\sum_{j=1}^{m-1} \sum_{k=1}^{m} \mathrm{sp}(c_i^j, c_i^k) - \sum_{j=1}^{m} \mathrm{gc}(c_i^j) + \alpha tc_i \right) \\
\text{with } \mathrm{sp}(c_i^j, c_i^k) &= \begin{cases} 1, & \text{if } c_i^j = c_i^k \\ 0, & \text{else} \end{cases} \\
\text{with } \mathrm{gc}(c_i^j) &= \begin{cases} 0, & \text{if } c_i^j \neq - \\ ge, & \text{else if } i > 1 \text{ and } c_{i-1}^j = - \\ go, & \text{else} \end{cases} \\
\text{with } tc_i &= \begin{cases} 1, & \text{if all residues match} \\ 0, & \text{else} \end{cases}
\end{aligned} \tag{4.6}$$

In Equation 4.6, a multi-objective reward function is presented where *ge* is the gap extension penalty and *go* is the gap opening penalty. Moreover, we employ a scaling factor α for controlling the magnitude of $\mathcal{TC}$. For *sp* and *tc*, we select the same parameters as proposed by Thompson et al. (1999) where for *sp*, each residue match counts as one point while zero points are returned in any other case, and *tc* gets one point for a complete column match.

Note that we want to help the agent learn that adding terminal gaps (see subsection 4.1.4) and creating columns full of gaps (see subsection 4.1.5) is undesired. To achieve this, the state will not pass through the multi-objective $\mathcal{R}$, but a high penalty of -100000 will be returned on the spot as a reward. Another technique would be masking those actions to restrict the agent from selecting them. Although this could be a solution in our case, calculating masks at each timestep might be computationally expensive. Consider that for generating masks at each timestep, we would have needed to calculate the outcome of each possible action selection for a given budget of simulations and a given maximum number of moves per simulation.

4.1.7 Game Steps

Considering the action space previously discussed, solving an MSA can involve varying numbers of steps to reach an optimal alignment. Determining the exact number of moves required is challenging due to several factors, such as the number of sequences, their lengths, and their similarities. We consider MSA as a game with a dynamically changing initial setup, where the required number of moves to win varies each time. Moreover, the winning state is unknown.
To determine when to terminate the game, we employ two strategies. The first one gives the agent the option to stop the game at any point by selecting action 0 (see subsection 4.1.2), and the second one involves using the parameter $steps_{ratio}$. In the second strategy, we set the maximum number of steps to $steps_{ratio} \cdot |S_0|$ where $steps_{ratio} \in (0, 1]$.

4.2 Model Architecture

Our method uses a State Encoder Network f for encoding the sequences. A policy p and a value h network are also designed, sharing the same encoder part f in a stacked fashion: $\pi_\theta(s) = p(f(s))$ and $V_\nu(s) = h(f(s))$. Parameters θ and ν are the initialization parameters of the networks. This network receives the MSA representation, aka state, and predicts the policy logits and a value. In AlphaZero, the predicted value corresponded to a scalar estimation of the likelihood of winning the game from the current position s_t. In contrast with games like chess, shogi, and Go, where the goal is to win, draw, or lose the game, the outcome of MSA is determined by accumulating points, so in our problem, the value estimates the expected number of points instead of the probability of winning. The number of available actions for each game moment maps to the length of the current state. One of our main challenges for the network is that probabilities' output shape depends on the input state. Furthermore, an additional challenge for the network to address is that the input state has a variable length in each timestep since the player can add a gap at each timestep, which will ascend the length of the MSA by one.
Inspired by Kool et al. (2018), the encoder-decoder architecture is based on Attention and a Glimpse-Pointer mechanism. The policy and value networks share the shame encoding part but differ in the decoder. In the following subsections, the network components are described in detail.

4.2.1 Encoder

4.2.1.1 Encodings-LookUp Embeddings

We design our encoder similar to the architecture proposed by Vaswani et al. (2017), which utilizes positional encodings such that the resulting embeddings are variant to the order. We utilize two types of positional encodings to inject some positional information into the model. Our encodings include information regarding which sub-sequence that the letter belongs to pos_{seq} and its position within the sub-sequence by pos_{subseq}. The output of the encodings is a triple of tuples. We may define them as:

$$\mathcal{E} = (e_{\text{token}}, pos_{\text{seq}}, pos_{\text{subseq}}) \tag{4.7}$$

For example, for a given flattened set of sequences at a given timestep t, $S_t = (STOP, A, T, T, A, T, END, T, G, T, A, G, END, T, A, G, -, -, END)$ the resulting encoding will be

$$\begin{aligned}
&((0,1,4,4,1,4,5,4,3,4,1,3,5,4,1,3,6,6,5),\\
&(0,1,2,3,4,5,6,1,2,3,4,5,6,1,2,3,4,5,6),\\
&(0,1,1,1,1,1,1,2,2,2,2,2,2,3,3,3,3,3,3))
\end{aligned}$$

Let $g_*(e_*)$ be the embedding function with parameters W_* and argument e_*. For example, $g_{token}(e_{token})$ has parameters W_{token}. Each trainable matrix has dimensions as:

$$\begin{aligned}
W_{\text{token}} &\in \mathbb{R}^{v \times d}\\
W_{\text{seq}} &\in \mathbb{R}^{m \times d}\\
W_{\text{subseq}} &\in \mathbb{R}^{l \times d}
\end{aligned}$$

where v is the token vocabulary length, m is the maximum number of sequences, l is the maximum sequence length length and d is the embedding dimension.
Encoded tuples are passed from g resulting on a triple of simple lookup embedding vectors : $(Emb_{\text{token}}, Emb_{\text{seq}}, Emb_{\text{subseq}})$ of d dimensional $Emb \in \mathbb{R}$ weights for each token.

$$\begin{aligned}
Emb_{\text{token}} &= g_{token}(e_{\text{token}})\\
Emb_{\text{seq}} &= g_{seq}(pos_{\text{seq}})\\
Emb_{\text{subseq}} &= g_{subseq}(pos_{\text{subseq}})
\end{aligned}$$

We aggregate those *Emp* vectors using element-wise addition into one vector, denoted as star embedding, $E*$. The learnable star, $E* \in \mathbb{R}^{\tilde{d}}$ (equation 4.8), where $\tilde{d}$ is the latent space dimension. Finally, each token of the input sequence is represented by a $1 \times \tilde{d}$ vector.

$$E* = Emb_{\text{token}} + Emb_{\text{seq}} + Emb_{\text{subseq}} \tag{4.8}$$

4.2.1.2 State Encoder Network

Let $f : \mathcal{S} \in \mathbb{R}^{\tilde{d}}$ be the State Encoder Network. The state encoder network f consists of Transformer Blocks. Using encoder structure logic as proposed in Vaswani et al. (2017), the blocks comprise a stack of five identical layers. We use circumflex to represent the output of each block as a new estimation of our embeddings.
The aggregated embedding $E*$ serves as the input of network t. Each Transformer Block consists of several layers. First $E*$ passes a Normalization layer (LN) (Ba et al., 2016), which outputs the normalized $E*_{norm}$ (see equation 4.9). The output $E*_{norm}$ is passed from a Multihead Self-Attention of which the output is aggregated with $E*$ and passed through another LN (see 4.10).

$$E*_{norm} = LN(E*) \tag{4.9}$$

$$\widehat{E} = LN(E * + \text{MultiHead}(Q = E*_{norm}, K = E*_{norm}, V = E*_{norm})) \tag{4.10}$$

where Multihead Self-Attention consists of 8 heads. We use a dropout for preventing overfitting with a probability of randomly omitted units of 0.01 (Hinton et al., 2012). Then $\widehat{E}$ is fed into a Feed-Forward network FF, added to itself such as:

$$E = \widehat{E} + FF(\widehat{E}) \tag{4.11}$$

After the Transformer Blocks, there is one last step, where the encoder computes the state embedding C (, denoted as AVG Context Embedding in Figure 4.2). Here, we want to encapsulate the entire state of the environment with an embedding of a fixed shape, and to achieve this, we average E and retrieve a unified vector C, as defined in equation 4.12. Note that $|C| = d$ ensures the context vector maintains a consistent and unchanging shape, regardless of the dynamic input states.

$$C \in \mathbb{R}^{d} = \frac{1}{L} \sum_{i=1}^{L} E_i \quad \text{where } L \text{ is the sequence length} \tag{4.12}$$

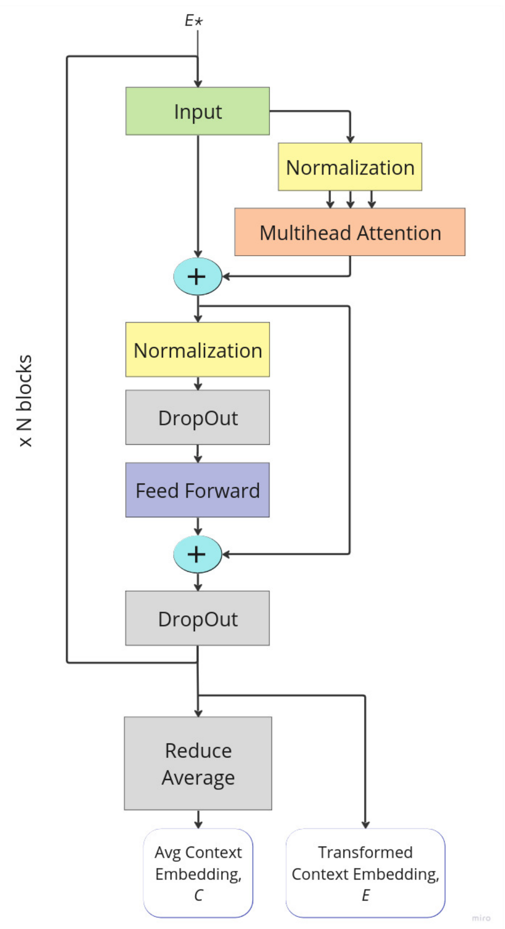


Figure 4.2: State Encoder Network Architecture

For the State Encoder Network, FF is a type of multilayer perceptron (MLP) with the same input and output dimensions along with a hidden layer that has a size four times the input dimension. This network employs the GELU activation function within its architecture (Hendrycks & Gimpel, 2016). Figure 4.2 summarizes the complete State Encoder Network architecture.

4.2.2 Decoder

For a given state s, the encoder generates E_s and C_s, with C_s, capturing the characteristics of the entire state. Since the training algorithm is based on the Gumbel AlphaZero approach, a value head and a policy head are introduced. The value

and policy networks share the same encoding part t but are followed by value head $h_\nu : \mathbb{R}^{\tilde{d}}, \mathbb{R}^L \times \mathbb{R}^{\tilde{d}} \to \mathbb{R}$ and policy head $p_\theta : \mathbb{R}^{\tilde{d}}, \mathbb{R}^L \times \mathbb{R}^{\tilde{d}} \to \mathbb{R}^L$.
The policy network in our problem shares similarities with rooting problems that Kool et al. (2018) focused on. In their work on the Traveling Salesman Problem (TSP), the decoder points to specific nodes to visit next and generates predictions sequentially using node and context embeddings while masking previously visited nodes. Similarly, we aim to point out positions to add a gap, but the input state size varies at each timestep.
Decoding, in our case, occurs sequentially with dynamic embeddings updated at each timestep. Unlike the TSP algorithm by Kool et al. (2018), our proposed MSA (policy) decoder does not employ any masks. The policy vector augments at each timestep while selecting a point at the previous timestep is equivalent to inserting a gap token into the sequence.

4.2.2.1 Policy head

The policy head, p, aims to generate a probability vector that indicates the most suitable index within our input sequence to select. Following the model's position selection, a gap is introduced, causing the sequence to shift to the right starting from that point. A Glimpse-Pointing Mechanism is utilized similar to Kool et al. (2019).

Glimpse mechanism Vinyals et al. (2016) introduces an attention step before the pointer mechanism called glimpse. Similarly, we utilize a MHA layer (Equation 4.13) as a glimpse, producing a new context embedding C'. The Attention takes as input a query vector $Q = C \in \mathbb{R}^{\tilde{d}}$ and E as a set of reference vectors $E = \{E_1, \dots, E_L\}$ where $E_i \in \mathbb{R}^{\tilde{d}}$.

$$C' = \text{MultiHead}(Q = C, K = E, V = E) \tag{4.13}$$

where Multihead Self-Attention consists of 8 heads. As reported by Bello et al. (2016), glimpsing more than once may not help the model improve when using the same parameters, so we use one attention glimpse step before the pointing mechanism.

Pointer mechanism A final Singlehead Attention layer is introduced to our decoder policy head, p, where the state representation C_s serves the query vector and the key-value vectors are represented by E_s. Similarly to Bello et al. (2016), the pointing mechanism produces a probability distribution over the next index in the sequence to which an action should be applied.
Singlehead weights, u_i are computed similarly to Bello et al. (2016) and Kool et al. (2018):

$$u_i = F \cdot tanh(\frac{(W^q C')^T (W^k E)}{\sqrt{\tilde{d}}}) \quad \text{where } u_1, \ldots, u_L \in [-F, F] \subseteq \{R\} \tag{4.14}$$

F is a constant that controls the range of the policy logits set to 10. Generally, u_i represents the logit of the probability of adding a gap in a specified position in the input sequence or ending the alignment (for zero index). This serves the policy vector $\pi(a|s)$. This final layer enables variable length input-output and is particularly effective when the output length matches the input length. The main characteristic we share with TSP as seen in Kool et al. (2018) is that each timestep input length matches the output length.

4.2.2.2 Value head

The value head, h, is structured much like the policy head, but instead of producing a vector, it predicts a single real number as the prediction. Initially, it uses a MHA layer and then a simple FF network (see equation 4.15).

FeedForward The FF network will applied a MHA layer for value head. This FF is implemented as a multilayer perceptron (MLP) featuring a series of linear transformation layers followed by non-linear activations (GELU) (Hendrycks & Gimpel, 2016) introducing non-linearity into the network in a stacked fashion. This FF network initially expands the output dimension four times the input dimension, then maintains this intermediate dimension, and finally reduces the output dimension to a scalar value:

$$h = \text{FF}(\text{MultiHead}(Q = C, K = E, V = E)) \tag{4.15}$$

where Multihead Self-Attention consists of 8 heads.
The complete design of the decoder's value and policy head architecture is illustrated in Figure 4.3.

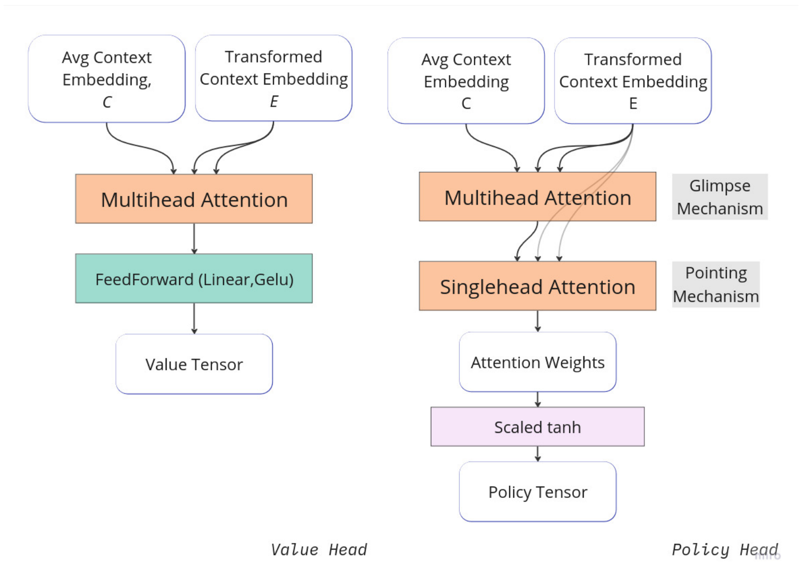


Figure 4.3: Value head and Policy Head Structure

Chapter 5

Experimental Results

5.1 Experimental Goal

Various solutions with distinct strategies have been proposed for solving MSA. Most solvers [1], are based on heuristic approaches for addressing the problem. Notably, DRL is a less commonly used approach in this domain. Many existing DRL approaches (Jafari et al., 2019; Joeres, 2021; Mircea et al., 2014) utilize the same MDP where the agent's task is selecting the order of sequences to align rather than adding direct gaps. In contrast, in RLALIGN (Ramakrishnan et al., 2018), the agent actually applies direct modifications as actions to the sequences. However, RLALIGN uses an architecture of a fixed input shape and applies it as a sliding window in sub-parts of the MSA for generating candidate actions. Even though the chosen action is one that maximizes a global score, this strategy raises the limitation of relying on localized information to generate policies. Another limitation is that RLALIGN allows the agent to perform a fixed number of steps, in contrast with our introduced strategy, which allows the agent to stop the game at any time. Additionally, none of the aforementioned and other DRL approaches, such as DPAMSA (Liu et al., 2023) utilize the same parameters for the evaluation metrics as some major MSA tools (Clustal Omega, MAFFT, MUSCLE) to create a fair comparison.
We aim to leverage NLP techniques to achieve flexibility in handling variable input

[1]for example MAFFT by Katoh et al. (2002), MUSCLE by Edgar (2004), T-Coffee by Notredame et al. (2000), ClustalW by Thompson et al. (1994), Clustal Omega by Sievers et al. (2011), DIALIGN by Morgenstern (2004), etc.

sizes and use an MDP to enable an agent to build robust policies for addressing MSA by strategically introducing gaps.
Our primary goal is to train an RL Agent using GAZ algorithm and NLP techniques to learn using self-play without any previous knowledge. We also aim to create a comparison with some widely known heuristic MSA solvers. We utilize a multiobjective reward function where we want to explore how variations in reward function parameters impact the performance of the RL agent. Note that we are not interested in maximizing only one metric (SP or TC) since our competitors present both metrics in their evaluation. Additionally, we investigate the impact of concepts like punishing complete column gaps or adjusting the ratio of the maximum number of steps per episode. Furthermore, we want to evaluate the influence of introducing synthetic data as training examples on the generalization capabilities of our agent. Finally, an in-depth comparison is complicated for several reasons, primarily due to variations in dataset selection, especially when working with DNA sequences and the chosen optimization objectives. Different objectives lead to different solutions, making direct comparisons more complex. To ensure a fair comparison, we select some of the most widely recognized aligner tools and apply the same evaluation metrics proposed for comparison by Thompson et al. (1999). We also apply a qualitative analysis of the MSAs to understand our aligning strengths and weaknesses or common strategies.

5.1.1 Data Acquisition

We utilize the Biopython tool (Chapman & Chang, 2000) to acquire DNA sequence data. Biopython tool provides code to access the National Center for Biotechnology Information (NCBI) using the Entrez package. Entrez is a molecular biology database system that provides access to nucleotide and protein sequence data (Schuler et al., 1996). From a total number of 38 databases for DNA and protein sequences, we utilize “nuccore” to download data in FASTA format (Pearson & Lipman, 1988). Only the “sequence length” criterion is applied for querying DNA sequence data from NCBI to ensure unbiased data collection. A local collection of sequences of length 10 is subsequently created in batched FASTA files. DNA sequences can generally contain 18 possible letters since they may also contain special characters apart from the four bases: adenine, guanine, cytosine, and thymine. In the acquired sequences, we apply a filter to retain only the sequences that are composed explicitly of the four nucleotide letters. Last, the remaining number of sequences of length 10 after filtering is 118825.
The ML pipeline involves generating MSAs by both sequences constructed by ran-

domly generated letters and authentic sequences from the local collection. This process employs an adjustable data ratio $ratio_{real/fake} \in (0, 1]$ to control the balance between real and synthetic sequences. For the generation of validation and test datasets, the ratio is set to one, as we aim to apply a fair evaluation exclusively on authentic MSAs. During the training stage, $ratio_{real/fake}$ is considered a hyperparameter, and we experiment with variable values.
All samples contain from 4 to 5 sequences with a length of 10. The training data are generated on the fly. The validation set is used as a baseline for keeping the best model. Both validation and test sets comprise of 256 number of sample MSAs. For the creation of validation and test sets, an initial set of 512 samples is generated randomly. Subsequently, half of these samples are sampled randomly for the validation set, and the remaining 256 are used for constructing the test set.

5.2 Competitors

We compare our approach using three well-established aligners. We utilize the *application* subpackage of Biotite by Kunzmann and Hamacher (2018), which offers interfaces to external software from which we utilize MUSCLE (Edgar, 2004) (version 3.8.31), MAFFT (Katoh et al., 2002), and Clustal Omega (Sievers & Higgins, 2018) using default options [2].
To compare validly, the methodology that those tools use to evaluate themselves is studied and produced similarly to make a fair comparison. In pursuit of a meaningful and equitable comparison, we analyze the datasets each aligner tool uses for self-evaluation and the metrics it seeks to maximize. Our investigation reveals that they all utilized $\mathcal{SPR}$ and $\mathcal{TC}$ scores. The $\mathcal{SPR}$ score is defined in Equation A.2, and it is essentially the ratio of the $\mathcal{SP}$ score achieved by the tool to the $\mathcal{SP}_r$ score obtained from a reference alignment. Regarding the $\mathcal{SP}$ score, all aligners apply one value for matching and 0 for all other cases. All competitors are evaluated in proteins and utilized protein reference alignment datasets [3].

[2]For MAFFT tool, the –auto flag is enabled, which facilitates automated selection over a set of programs. This flag chooses a consistency-based program (L-INS-i) in a small number and lengths of sequences (Katoh et al., 2002).

[3]For additional details, we refer to section A.6.

5.3 Setup

5.3.1 Training Loop

We train the agent using an improved version of AlphaZero, GAZ, to obtain maximum performance using a relatively small number of simulations. We utilize multiple playing processes for playing games with periodically updated network checkpoints. We update our network checkpoints every 600 training steps using the 256 fixed initial states from the validation step to play. Moreover, we continuously update network weights through the learning process of Gumbel MCTS. A visual representation of the training loop can be found in section A.7.

5.3.2 Reward

For the multiobjective reward function $\mathcal{R}_s^a$ (as described in Equation 4.5), we explore different parameters for α, go and ge. When setting go and ge other than zero, we introduce an affine gap penalization with $go > ge$.

5.3.3 Replay Buffer

At each episode, the agent generates experience, which is stored in the Replay Buffer when the episode finishes. The elements stored include tuples of states and the improved policies from Gumbel MCTS at each step, along with the actual game outcomes such as $(s_t, \pi'(s_t)), (s_t, z_t)$.

5.3.4 Loss Functions

The policy π_θ and value V_v networks are trained using the historic states that occurred from the interaction with the environment sampled from the replay buffer. Equivalent to Gumbel AlphaZero, the loss function for policy network π_θ is the KL divergence, and the loss function for the value network is the MSE.

5.3.5 Gumbel Sequential Halving

For the Gumbel trick, a maximum of $m = 16$ actions are sampled without replacement at the root of GAZ's search tree. A simulation budget of 200 in the tree search is utilized in almost all the experiments [4]. These are the default values as proposed by Danihelka et al. (2022).

5.3.6 Hyperparameters

For all the conducted experiments, the replay buffer retains information from the most recent 1500 episodes for almost all our experiments. We also employed a replay buffer of the 2000 most recent episodes for certain experiments. We employ the Adam optimizer as described by Kingma and Ba (2014), with a fixed learning rate of 0.0001. During each training step, we draw batches of 512 samples. Finally, similarly to AlphaZero (Equation 2.29), gradients are clipped to have a maximum $L_2 - norm$ of 1. The agent plays 80k episodes, maintaining the ratio of approximately two optimizer steps for every played episode. For the network architecture (as described in section 4.2), we use a latent dimension of 256 in all experiments.

5.4 Evaluation metrics

During training, we explore different parameters for formulating the reward function. As previously discussed, all of the aforementioned aligners use $\mathcal{SPR}$ and $\mathcal{TC}$ as proposed by Thompson et al. (1999) for evaluation. $\mathcal{SPR}$ is a fraction since it is calculated relatively to a target-aligned reference set. Since we do not use any target or reference dataset, we drop the scaling part and count raw $\mathcal{SP}$ with the same parameters. On the other hand, $\mathcal{TC}$ score is not a relative metric, and it is also included in the evaluation.

[4] We conducted one experiment with a higher simulation budget of 300.

5.5 Results

In this section, we present the results from the experiments in comparison with three established aligners. Our experimental configurations encompass several elements, including tuning parameters such as α, go, and ge in the scoring function. Specifically, go and ge are employed for gap penalization. For the α parameter, we use the value of one in settings 9 and 10 and multiples of ten (10, 20) in all other settings. A consistent batch size of 512 is generally employed, with one experiment using a batch size of 1024. All experiments employ 200 simulations in the MCTS, except for setting 10, with increased simulations of 300. Furthermore, we explore training the agent with and without CCG. We also vary the maximum number of allowed steps by tuning $steps_{ratio}$ based on the observed behavior of Clustal Omega on the validation set. We find a rough average of added gaps in each alignment is about 20 percent of the flattened MSA initial length. Subsequently, we select $steps_{ratio}$ from a range spanning 0.25 to 0.5. Finally, we fine-tune the $ratio_{real/fake}$ parameter, adjusting it within the range of 0.8 to 1 to manage the balance between real and synthetic training examples. We add only a small proportion of synthetic data to boost the diversity of training examples, aiming for a better model generalization. However, in setting 8, we deliberately reduce this proportion to half of the samples to observe the impact on the learning process.
A summary of our experimental findings can be found in Table 5.1. The competitor aligner tools are ranked according to their scores, with Clustal Omega leading, then MAFFT, and MUSCLE ranking last. The second half of the table displays various experiment settings. The last four columns include the scoring metrics, namely, SP and TC averaged and summed across the test set.
The results indicate that settings 9, 5, and 4 delivered the best outcomes regarding average and total scores for SP. Next, considering the TC scores, the highest scores are observed in settings 6, 8, and 5, with setting 5 exhibiting nearly identical scores to setting 4. Consequently, settings 4 and 5 are in the top three for both SP and TC objectives, with setting 5 having a little precedence, especially for SP score over setting 4.
Even though the speed for execution is not within the scope of this thesis, we want to provide a rough estimate of the inference time for each episode. The evaluation on the test set took approximately 2.7 minutes in total and around 0.6 seconds per example alignment[5].

[5]Since available resources influence the time metrics, we refer to section 6.1 for details about the hardware configuration.

Method	α	batch	rb	simulations	CCG	go	ge	$steps_{ratio}$	$ratio_{real/fake}$	Avg SP	Avg TC	Sum SP	Sum TC
ClustalOmega	-	-	-	-	-	-	-	-	-	26.54	0.66	6795	169
MAFFT	-	-	-	-	-	-	-	-	-	23.54	0.50	6025	128
MUSCLE	-	-	-	-	-	-	-	-	-	23.20	0.38	5938	97
Setting 1	20	512	2000	200	True	0	0	0.4	0.8	21.26	0.34	5442	88
Setting 2	20	1024	2000	200	True	0	0	0.4	0.8	23.55	0.61	6028	156
Setting 3	10	512	1500	200	True	0	0	0.4	0.9	24.02	0.60	6148	154
Setting 4	**10**	**512**	**1500**	**200**	**False**	**0**	**0**	**0.4**	**0.9**	**24.45**	**0.64**	**6260**	**164**
Setting 5	**10**	**512**	**1500**	**200**	**False**	**0**	**0**	**0.4**	**1**	**24.68**	**0.65**	**6318**	**165**
Setting 6	10	512	1500	200	False	0.1	0.01	0.4	0.9	23.85	0.77	6105	198
Setting 7	10	512	1500	200	True	0.1	0.01	0.25	1	22.34	0.49	5718	126
Setting 8	10	512	1500	200	False	0	0	0.3	0.5	24.34	0.68	6230	175
Setting 9	1	512	1500	200	True	0	0	0.3	0.8	24.70	0.52	6324	132
Setting 10	1	512	1500	300	True	0	0	0.4	0.8	24.40	0.47	6245	121

Table 5.1: Summary of results

5.5.1 Quantitative analysis

In this section, we discuss performance across different settings based on the various parameters we have utilized. The discussed results refer to the Table 5.1. The experiments with the best performance among all settings, for both SP and TC objectives, are settings 4 and 5. Setting 9 exhibits superior performance in the SP score, while setting 6 excels in TC, surpassing all other configurations. Even though both managed to be the first in the ranking for one metric, they fall short of achieving both objectives, as evidenced (in Table 5.2) by the 7th position in the ranking for the SP metric for setting 6 and the 6th position for the TC metric for setting 9. In settings 9 and 10, there is a notable discrepancy between their high SP rankings and comparatively lower TC rankings. This observation may be attributed to a low α. Specifically, Clustal Omega (in Table 5.1) yields an average of 26.54 matches for SP score per alignment, with only 0.66 for TC score. Subsequently, in cases where α is one, SP and TC scores do not contribute equally to the reward function, which might lead to a dominance of SP score. Moreover, even though setting 10 has more simulations in the MCTS it does not perform better than setting 9, which could be attributed to the longer horizon of the game, where setting 9 performs more steps for each episode.
Furthermore, settings 4 and 5 have similar performance, with 5 reporting slightly better results. A plausible explanation for this difference could be due to the 10% synthetic data in the training examples of setting 4. This isn't necessarily a drawback, and its impact may vary across different test sets or with longer training episodes.
Another interesting observation is that setting 4 and setting 6 share the same parameters except for affine gap penalization applied in setting 6. Affine gap penalization tends to reduce the range of the reward function. Since TC typically gets low values, this difference might help it stand out more in the multiobjective reward function and potentially make the algorithm more sensitive to variations in TC.
Moreover, the options for setting 4 are the same as setting 3, except that setting 3 incorporates *CCG*. Interestingly, setting 3 returns worse results in both SP and TC. This observation raises the possibility that including *CCG* may be confusing or potentially not beneficial for the agent.
The findings are encouraging, with settings 2, 3, 4, 5, 6, 8, and 9 exceeding MAFFT and MUSCLE competitors. None of our models exceeds Clustal Omega in both SP and TC at the same time. Notably, setting 5 has an average negative difference with Clustal Omega of 1.85 in the SP score and 0.01 in the TC score.

Rank	SP	TC
1	9	6
2	5	8
3	4	4, 5
4	10	2
5	8	3
6	3	9
7	6	7
8	2	10
9	7	1
10	1	

Table 5.2: Settings Ranking (descending order)

5.5.2 Qualitative analysis

Metrics are indicators of alignment quality, offering quantitative assessments. However, it is also important to inspect the original alignment outputs and compare them with those generated by competitor methods. To understand some qualitative differences, a presentation of four example algorithm outputs from the test set is intended. This will help us understand some algorithm strategies and how they handle specific alignment challenges. Furthermore, manual inspection helps us detect any undesired or biologically irrelevant behaviors in the alignment, a crucial step for future optimizations. In the examples, the top table represents the initial alignment, while the middle tables hold the output alignment for different settings, and the Bottom table is the output alignment of Clustal Omega. We selected Clustal Omega because it stands out as the strongest aligner among our competitors. From our list of trained models, we select settings 4, 5, 6, and 8 for qualitative analysis.
In the first example (Table 5.3), setting 6 outperforms Clustal Omega with one additional match and has the same TC score. Setting 6 decides on longer gaps than all other settings. The output of setting 6 differs only on one element from the one from Clustal Omega. Clustal Omega moves the first letter (T) of the third sequence, adding two gaps before the sequence. On the other hand, setting 6 decides to add a gap before and after the same element (T), which yields a higher SP score.
Next, in the second example (Table 5.4), all our models successfully yielded a TC match, while setting 6 achieved two matches. Specifically, setting 6 achieves the same TC match as Clustal Omega and an additional one, accomplishing this by inserting a gap on the fourth element of the second sequence instead of at the beginning, as performed by Clustal Omega. Settings 4 and 5 share a common strategy of

moving the second sequence to the right to create the TC match on the 9th column. For sliding the second sequence, both split the sequence into two parts by introducing a gap. Note that at this point, setting 5 chooses a wiser position to insert a gap, resulting in an additional two points in the SP score. In this test example, all our presented models exceed Clustal Omega in SP score. In example 3 (Table 5.5), all settings exhibited similar behavior by introducing gaps in the second and fourth sequences. Interestingly, settings 5 and 8 perform identical alignments and score the highest. This is an example case where Clustal Omega excels over all our settings in SP score without splitting any sequence by sacrificing one TC match. Similarly to Clustal Omega, our settings perform the same number of steps (of adding two gaps).

In example 4 (Table 5.6), Clustal Omega stands out by introducing numerous gaps, in contrast to our settings, which exhibit a more conservative behavior. Notably, settings 8 and 5, by adding two gaps, achieve the highest SP score among our settings. Setting 4 stops the alignment one step before setting 5, while setting 6 promptly ends the alignment without adding any gaps, resulting in the lowest score. This example can be explained as follows: Sequences 1, 2, 3, and 5 exhibit high similarity in the right part, whereas the fourth sequence is irrelevant in the right part but shares some similarities, particularly with sequence 2 in the left part. To achieve Clustal Omega output, our algorithm should perform 15 gaps while passing through some less favorable states. This is one hard MSA case example that would need a more explorative and far-sighted agent to reach the Clustal Omega score.

We include example 5 (Table 5.7) as a case where Clustal Omega introduces a gap within a sequence, a behavior that is not commonly observed. Two of our settings, 4 and 6, decided to drop the alignment without adding any gap. On the other hand, setting 8 adds a single gap, causing a shift in the second sequence and resulting in an additional match. Notably, setting 5 demonstrates superior performance, surpassing all other settings and even outperforming Clustal Omega itself, achieving 21 matches through a series of three steps. This serves as a counterexample to the general norm observed in the previous examples, where Clustal Omega tends to avoid splitting sequences and tries to add gaps at the beginning or end of sequences. Here, Clustal Omega splits 2 sequences with two consecutive gaps each, whereas setting 5 adds fewer gaps and achieves 4 extra matches.

Example 6 (Table 5.8) is a typical example, where setting 5 gets the best score compared to the other settings. All other settings split sequences into at least one point, whereas setting 5 does not, which is quite surprising considering that it was not trained with any gap penalization strategy.

In example 7 (Table 5.9), both Clustal Omega and setting 5 slide sequences in the same way, leading sequences 1, 2, 4, and 5 align similarly. However, Clustal Omega deviates by shifting sequence 3 to the left, resulting in a worse alignment for 1 match.

This is the third example of setting 6 stopping the alignment without introducing any gaps. The conservative behavior of setting 6 may contribute to its absence from the top-ranking positions, particularly when evaluating the SP score.
Analyzing all the examples, we observe a common pattern in our settings, which tends to introduce a few gaps and often concludes the alignment process early. This behavior might be beneficial, where the agent can align fine with limited gaps. However, we also want to achieve a more explorative agent that tries alternative strategies, especially in future cases involving larger sequences. Moreover, we frequently observe a common intention in the aligners regarding the intended direction for shifting each sequence (either right or left).

G	C	C	A	T	C	C	G	G	T
G	T	T	T	C	C	T	T	T	C
T	A	A	C	G	T	C	G	G	C
A	T	C	T	A	G	A	T	C	C

G	C	C	A	T	C	C	G	G	T	-	-
G	T	-	T	T	C	C	T	T	T	C	-
T	A	A	C	G	T	C	G	G	-	C	-
A	T	-	-	C	T	A	G	A	T	C	C

SP-Score = 18, TC-Score = 0, Setting 4

G	C	C	A	T	C	C	-	-	G	G	T
G	T	T	T	C	C	T	T	T	C	-	-
-	T	-	A	A	C	G	T	C	G	G	C
A	T	C	T	A	G	A	T	C	C	-	-

SP-Score = 18, TC-Score = 0, Setting 5.

-	-	-	G	C	C	A	T	C	C	G	G	T
G	T	T	T	C	C	T	T	T	C	-	-	-
-	T	-	A	A	C	G	T	C	G	G	C	-
A	T	C	T	A	G	A	T	C	C	-	-	-

SP-Score = 23, TC-Score = 1, Setting 6.

G	C	C	A	T	C	C	G	G	T	-	-
G	T	T	-	T	C	C	T	T	T	C	-
-	T	-	A	A	C	G	T	C	G	G	C
A	T	C	T	A	G	A	T	C	C	-	-

SP-Score = 17, TC-Score = 0, Setting 8.

-	-	-	G	C	C	A	T	C	C	G	G	T
G	T	T	T	C	C	T	T	T	C	-	-	-
-	-	T	A	A	C	G	T	C	G	G	C	-
A	T	C	T	A	G	A	T	C	C	-	-	-

SP-Score = 22 TC-Score = 1 ClustalOmega

Table 5.3: Example Test 1

A	A	C	C	G	A	A	C	G	T
G	A	C	G	A	T	C	G	T	C
G	C	T	T	A	G	A	T	G	T
G	C	T	G	A	T	G	C	G	A

A	A	C	C	G	A	A	C	G	T	-
G	A	-	C	G	A	T	C	G	T	C
G	C	T	T	A	G	A	T	G	T	-
G	C	T	G	A	T	G	C	G	A	-

SP-Score = 23, TC-Score = 1, Setting 4

A	A	C	C	G	A	A	C	G	T	-
G	A	C	G	A	T	-	C	G	T	C
G	C	T	T	A	G	A	T	G	T	-
G	C	T	G	A	T	G	C	G	A	-

SP-Score = 25, TC-Score = 1, Setting 5

A	A	C	C	G	A	A	C	G	T	-
G	A	C	-	G	A	T	C	G	T	C
-	G	C	T	T	A	G	A	T	G	T
-	G	C	T	G	A	T	G	C	G	A

SP-Score = 23, TC-Score = 2, Setting 6

A	A	C	C	G	A	A	C	G	T	-
G	A	C	-	G	A	T	C	G	T	C
G	C	T	T	A	G	A	T	G	T	-
G	C	T	G	A	T	G	C	G	A	-

SP-Score = 23, TC-Score = 1, Setting 8

A	A	C	C	G	A	A	C	G	T	-
-	G	A	C	G	A	T	C	G	T	C
-	G	C	T	T	A	G	A	T	G	T
-	G	C	T	G	A	T	G	C	G	A

SP-Score = 22, TC-Score = 1, ClustalOmega

Table 5.4: Example Test 2

G	C	A	A	C	C	A	C	A	G
G	A	A	C	T	C	C	A	C	A
T	A	A	A	G	T	T	A	A	T
T	A	C	T	A	T	C	A	T	C
A	C	A	T	T	C	A	C	A	A

G	C	A	A	C	C	A	C	A	G	-
G	A	-	A	C	T	C	C	A	C	A
T	A	A	A	G	T	T	A	A	T	-
T	A	-	C	T	A	T	C	A	T	C
A	C	A	T	T	C	A	C	A	A	-

SP-Score = 35, TC-Score = 1, Setting 4

G	C	A	A	C	C	A	C	A	G	-
G	-	A	A	C	T	C	C	A	C	A
T	A	A	A	G	T	T	A	A	T	-
T	-	A	C	T	A	T	C	A	T	C
A	C	A	T	T	C	A	C	A	A	-

SP-Score = 39, TC-Score = 2, Setting 5

G	C	A	A	C	C	A	C	A	G	-
-	G	A	A	C	T	C	C	A	C	A
T	A	A	A	G	T	T	A	A	T	-
T	-	A	C	T	A	T	C	A	T	C
A	C	A	T	T	C	A	C	A	A	-

SP-Score = 38, TC-Score = 2, Setting 6

G	C	A	A	C	C	A	C	A	G	-
G	-	A	A	C	T	C	C	A	C	A
T	A	A	A	G	T	T	A	A	T	-
T	-	A	C	T	A	T	C	A	T	C
A	C	A	T	T	C	A	C	A	A	-

SP-Score = 39, TC-Score = 2, Setting 8

-	G	C	A	A	C	C	A	C	A	G
G	A	A	C	T	C	C	A	C	A	-
T	A	A	A	G	T	T	A	A	T	-
T	A	C	T	A	T	C	A	T	C	-
-	A	C	A	T	T	C	A	C	A	A

SP-Score = 42, TC-Score = 1, ClustalOmega

Table 5.5: Example Test 3

T	C	T	A	T	A	A	C	C	T
A	G	G	G	A	T	A	A	A	A
C	T	T	T	C	A	A	A	A	C
G	G	A	G	G	G	G	G	C	T
T	G	A	T	C	C	A	A	A	G

-	T	C	T	A	T	A	A	C	C	T
A	G	G	G	A	T	A	A	A	A	-
C	T	T	T	C	A	A	A	A	C	-
G	G	A	G	G	G	G	G	C	T	-
T	G	A	T	C	C	A	A	A	G	-

SP-Score = 29, TC-Score = 0, Setting 4

-	T	C	T	A	T	A	A	C	C	T
A	G	G	G	A	T	A	A	A	A	-
C	T	T	T	C	A	A	A	A	C	-
-	G	G	A	G	G	G	G	G	C	T
T	G	A	T	C	C	A	A	A	G	-

SP-Score = 30, TC-Score = 0 Setting 5

T	C	T	A	T	A	A	C	C	T
A	G	G	G	A	T	A	A	A	A
C	T	T	T	C	A	A	A	A	C
G	G	A	G	G	G	G	G	C	T
T	G	A	T	C	C	A	A	A	G

SP-Score = 24, TC-Score = 0 Setting 6

T	-	C	T	A	T	A	A	C	C	T
A	G	G	G	A	T	A	A	A	A	-
C	T	T	T	C	A	A	A	A	C	-
-	G	G	A	G	G	G	G	G	C	T
T	G	A	T	C	C	A	A	A	G	-

SP-Score = 30, TC-Score = 0 Setting 8

-	-	-	-	-	T	C	T	A	T	A	A	C	C	T
-	-	A	G	G	G	A	T	A	A	A	A	-	-	-
-	-	-	-	C	T	T	T	C	A	A	A	A	C	-
G	G	A	G	G	G	G	G	C	T	-	-	-	-	-
-	-	-	-	T	G	A	T	C	C	A	A	A	G	-

SP-Score = 34, TC-Score = 0, ClustalOmega

Table 5.6: Example Test 4

T	T	T	T	A	T	T	T	G	C
T	T	G	C	C	G	A	A	T	C
G	G	G	G	G	A	G	A	T	G
G	G	T	G	A	T	G	G	G	C

T	T	T	T	A	T	T	T	G	C
T	T	G	C	C	G	A	A	T	C
G	G	G	G	G	A	G	A	T	G
G	G	T	G	A	T	G	G	G	C

SP-Score = 16, TC-Score = 0, Setting 4

T	T	T	T	-	A	T	T	T	G	C
-	T	T	G	C	C	G	A	A	T	C
G	G	G	G	G	A	G	A	T	G	-
G	G	T	G	A	T	G	-	G	G	C

SP-Score = 21, TC-Score = 0 Setting 5

T	T	T	T	A	T	T	T	G	C
T	T	G	C	C	G	A	A	T	C
G	G	G	G	G	A	G	A	T	G
G	G	T	G	A	T	G	G	G	C

SP-Score = 16, TC-Score = 0 Setting 6

T	T	T	T	A	T	T	T	G	C	-
-	T	T	G	C	C	G	A	A	T	C
G	G	G	G	G	A	G	A	T	G	-
G	G	T	G	A	T	G	G	G	C	-

SP-Score = 17, TC-Score = 0 Setting 8

-	T	T	-	-	T	T	A	T	T	T	G	C
-	T	T	-	-	G	C	C	G	A	A	T	C
G	G	G	G	G	A	G	A	T	G	-	-	-
-	-	-	G	G	T	G	A	T	G	G	G	C

SP-Score = 17, TC-Score = 0, ClustalOmega

Table 5.7: Example Test 5

G	A	C	T	T	G	T	A	T	A
A	A	A	A	A	T	G	G	A	T
C	C	T	G	T	A	A	T	C	C
A	G	A	T	A	A	A	T	T	C
G	T	A	T	G	G	G	C	C	C

G	A	C	T	T	G	T	A	T	A	-
A	A	A	A	A	T	G	G	A	T	-
-	C	C	T	G	T	A	A	T	C	C
A	G	A	T	A	-	A	A	T	T	C
G	T	A	T	G	G	G	C	C	C	-

SP-Score = 28, TC-Score = 0, Setting 4

-	G	A	C	T	T	G	T	A	T	A
A	A	A	A	A	T	G	G	A	T	-
C	C	T	G	T	A	A	T	C	C	-
A	G	A	T	A	A	A	T	T	C	-
-	G	T	A	T	G	G	G	C	C	C

SP-Score = 29, TC-Score = 0 Setting 5

G	-	A	C	T	T	G	T	A	T	A
A	A	A	A	A	T	G	G	A	T	-
C	C	T	G	T	A	A	T	C	C	-
A	G	A	T	A	A	A	T	T	C	-
G	T	A	T	G	G	G	C	C	C	-

SP-Score = 26, TC-Score = 0 Setting 6

G	A	C	T	T	G	T	A	T	A	-
A	A	A	A	A	T	G	G	A	T	-
C	-	C	T	G	T	A	A	T	C	C
A	G	A	T	A	A	A	T	T	C	-
G	T	A	T	G	G	G	C	C	C	-

SP-Score = 26, TC-Score = 0, Setting 8

-	G	A	C	T	T	G	T	A	T	A	-
A	A	A	A	A	T	G	G	A	T	-	-
-	-	C	C	T	G	T	A	A	T	C	C
-	-	A	G	A	T	A	A	A	T	T	C
-	G	T	A	T	G	G	G	C	C	C	-

SP-Score = 33, TC-Score = 0, ClustalOmega

Table 5.8: Example Test 6

A	G	G	G	C	C	G	G	C	G			
A	T	A	T	A	T	A	T	A	T			
C	A	A	A	A	C	A	A	C	A			
T	A	G	T	C	A	T	G	C	G			
T	T	G	A	T	T	T	T	A	A			
A	G	G	G	C	C	G	G	C	G	-		
-	A	T	A	T	A	T	A	T	A	T		*SP-Score = 26,*
C	A	A	A	A	C	A	A	C	A	-		*TC-Score = 0,*
T	A	G	T	C	A	T	G	C	G	-		*Setting 4*
T	T	G	A	T	T	T	T	A	A	-		
-	A	G	G	G	C	C	G	G	C	G		
A	T	A	T	A	T	A	T	A	T	-		*SP-Score = 27,*
C	A	A	A	A	C	A	A	C	A	-		*TC-Score = 0*
-	T	A	G	T	C	A	T	G	C	G		*Setting 5*
T	T	G	A	T	T	T	T	A	A	-		
A	G	G	G	C	C	G	G	C	G			
A	T	A	T	A	T	A	T	A	T			*SP-Score = 24,*
C	A	A	A	A	C	A	A	C	A			*TC-Score = 0*
T	A	G	T	C	A	T	G	C	G			*Setting 6*
T	T	G	A	T	T	T	T	A	A			
A	-	G	G	G	C	C	G	G	C	G		
A	T	A	T	A	T	A	T	A	T	-		*SP-Score = 26,*
C	A	A	A	A	C	A	A	C	A	-		*TC-Score = 0,*
T	-	A	G	T	C	A	T	G	C	G		*Setting 8*
T	T	G	A	T	T	T	T	A	A	-		
-	-	A	G	G	G	C	C	G	G	C	G	
-	A	T	A	T	A	T	A	T	A	T	-	*SP-Score = 26,*
C	A	A	A	A	C	A	A	C	A	-	-	*TC-Score = 0,*
-	-	T	A	G	T	C	A	T	G	C	G	*ClustalOmega*
-	T	T	G	A	T	T	T	T	A	A	-	

Table 5.9: Example Test 7

5.5.3 Further Results

Additional plots while training are presented for setting 5. In Figure 5.1, we present the average objective that the agent scores every 128 episodes. It's important to note that the observed high turbulence is expected, as each input alignment can yield a different score not only due to the agent's behavior but also due to the inherent variability in the setup. We observe that after 70000 episodes, there are some points where the average objective function gets higher reward values, which might indicate that the agent may improve over longer training episodes. However, the early stagnation observed on the average objective curve prompts consideration of parameters that might influence the model's learning process.
In Figure 5.2, we present the ratio of the agent stopping the episode immediately without adding any gap in every 128 episodes. In an ideal scenario controlled by an optimal policy, stopping the game without playing suggests that there is no alternative action that could yield a greater score in the long run. Notably, the agent tends to stop immediately more frequently at the beginning of the training. This might be due to the agents' lack of confidence and understanding of the game dynamics. At the end of the training, the agent seems to stop the game immediately less often, which is considered a positive outcome.
In Figure 5.3, the evaluation of Gumbel MCTS in the validation set during training is presented. In the current plot, each point corresponds to 600 training steps and is the average objective that the agent achieves in the evaluation steps.
Considering the loss function while optimizing the model, Figure 5.4 presents the loss curve on a log scale (in the vertical axis). By employing the logarithmic scale, we provide a more detailed view, especially for lower ranges of loss values. Moreover, the loss curve is averaged every 128 steps.
Previously, we presented the average and the sum of scores for two different metrics in Table 5.1. While these metrics provide an overall understanding of how our models compare with the aligners, they lack the context of performance in individual games. To address this, we compare our model against other aligners in a win-draw-lose fashion. Considering the 256 test examples, we present the ratio of times that our algorithm achieves a score better than other aligners, which we termed as a 'win'. Similarly, we present the ratio of times that our model achieved the same score as other aligners, labeled as a 'draw.' and the ratio of times that our model reported a lower score than other aligners, labeled as 'lose'. The result of this comparison for setting 5 is depicted in Figure 5.5 for SP score and Figure 5.6 for TC score, respectively. The results show that the agent derived from setting 5 scores equal or better in most of the cases for all the aligners in TC score and for MAFFT and MUSCLE aligners in SP score. On the other hand, our model loses from Clustal Omega in SP score around 60% of the times.

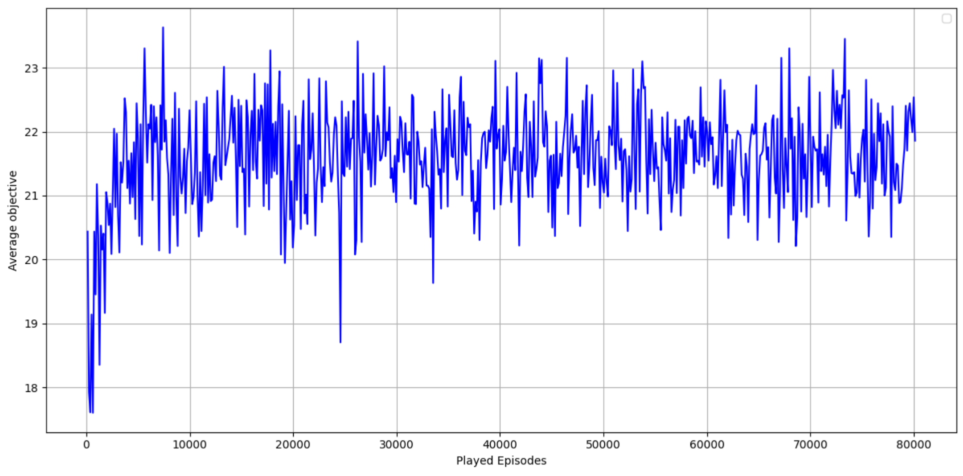


Figure 5.1: Average Objective. Setting 5

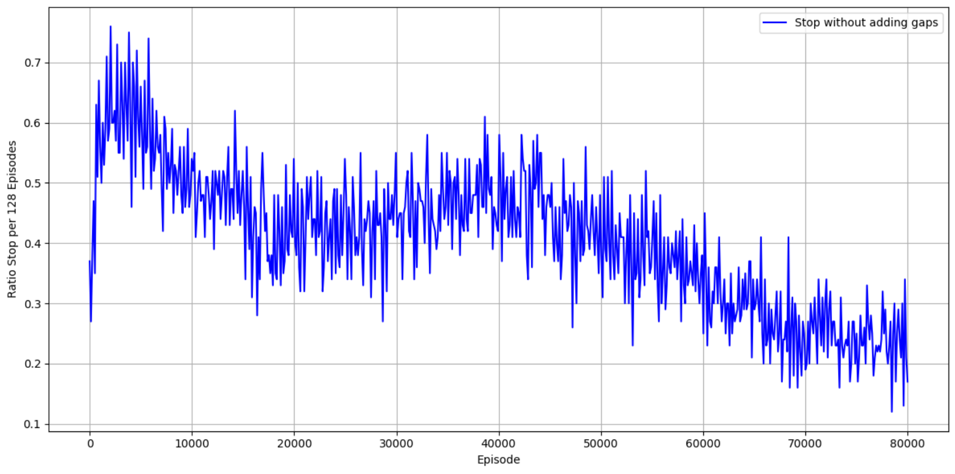


Figure 5.2: Ratio of stopping Setting 5

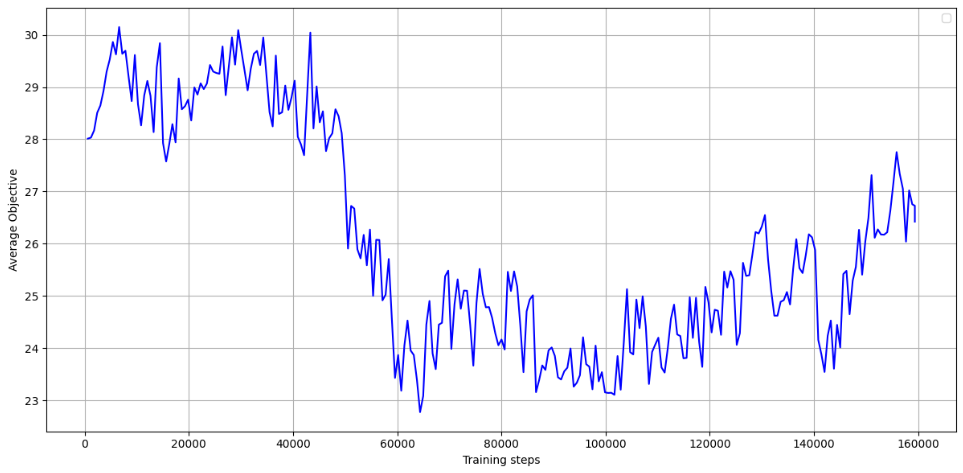


Figure 5.3: Evaluation every 600 steps. Setting 5

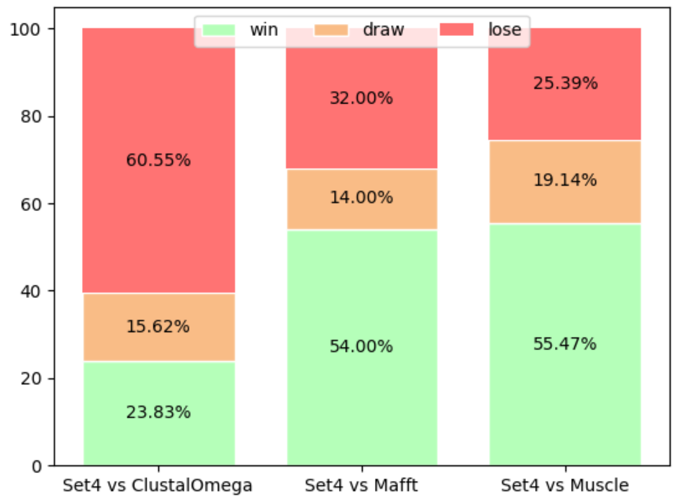


Figure 5.5: Win-Draw-Lose SP Setting 5

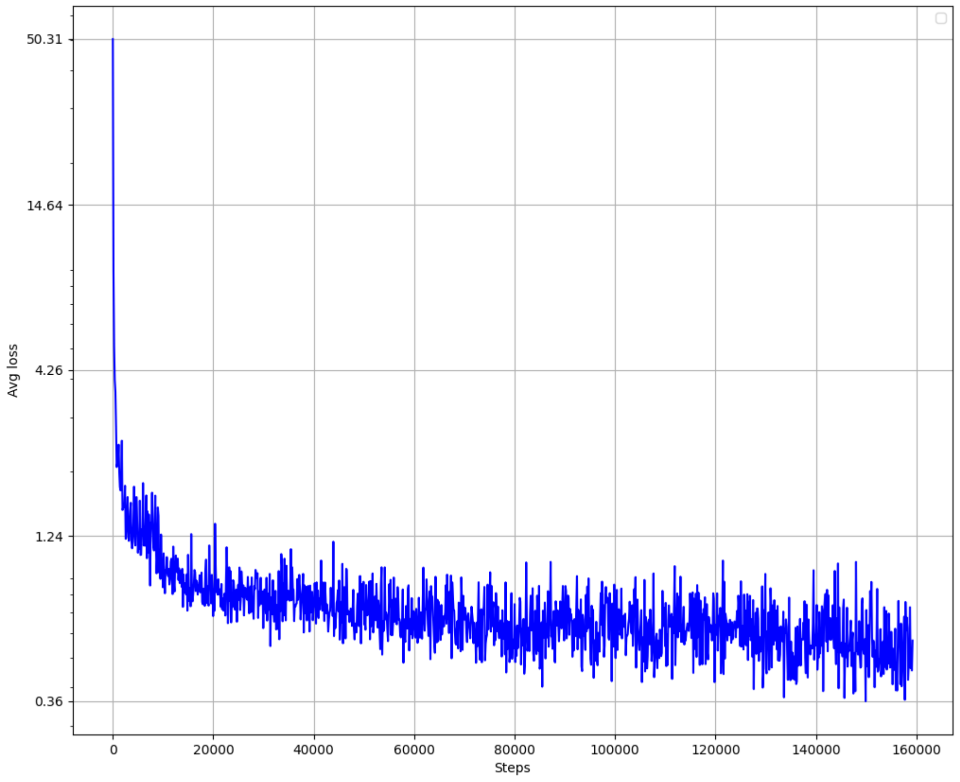


Figure 5.4: Average loss (log scale). Setting 5

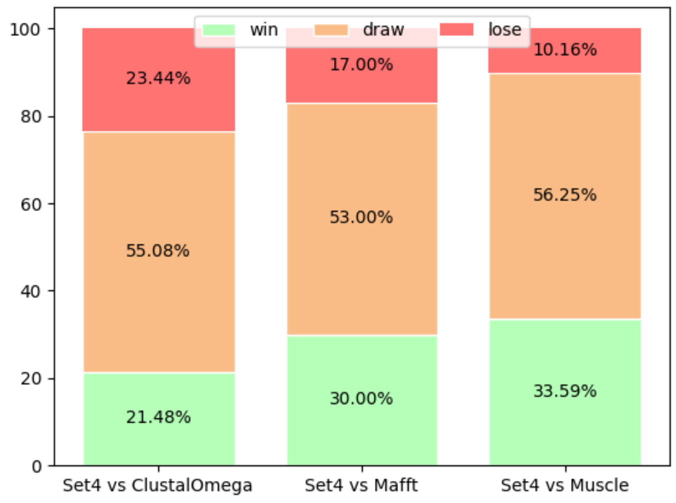


Figure 5.6: Win-Draw-Lose TC Setting 5

5.5.4 Reproducibility

We conduct a comprehensive evaluation of the performance and robustness of our algorithm across three different seeds in Figure 5.7 and Figure 5.8. Additionally, we present the results in the evaluation step of the test set in Table 5.10. We utilize seeds 42, 43 and 44 for setting 4 [6]. The different seeds do not present notable variations in performance.

[6]Initially chosen for its promising performance, setting 4 was later surpassed by setting 5

Avg SP	Avg TC	Sum SP	Sum TC
24.41 ± 0.25	0.64 ± 0.02	6235.00 ± 81.55	163.67 ± 4.50

Table 5.10: Setting 4, performance across seeds ∈ 42, 43, 44, averaged ± standard deviation

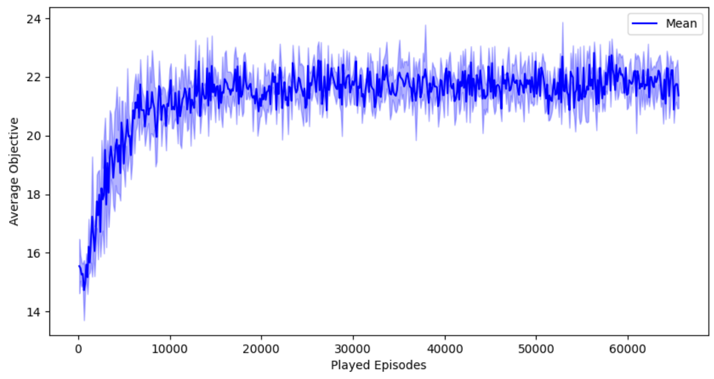


Figure 5.7: Mean training performance, run with 3 distinct seeds ∈ 42, 43, 44. Shades denote standard errors.

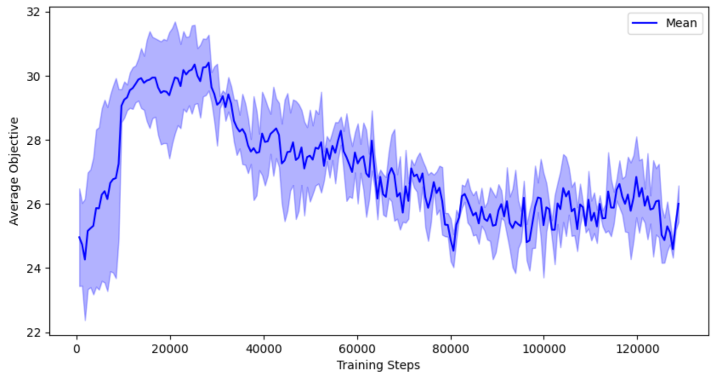


Figure 5.8: Mean evaluation performance, run with 3 distinct seeds ∈ 42, 43, 44. Shades denote standard errors.

Chapter 6

Conclusion

In this thesis, we introduce a novel strategy for addressing the MSA problem. Our approach relies on Gumbel AlphaZero and a transformer-based architecture. We believe that our method is a powerful starting point for solving this problem using no previous knowledge or heuristics in general.
Compared to previous works, we use DRL and introduce a MDP in which the agent directly affects the sequences by adding gaps. We train an agent using Gumbel AlphaZero, which reports major performance improvements in games like Go, chess, and Atari with only a few simulations in the MCTS. Diverging from recent transformer-based approaches (Dotan et al., n.d.), our utilization of AlphaZero allows the agent exclusively learn through self-play games without relying on input-target datasets. This is advantageous in our scenario, where the exact optimal alignment output is unknown, and an agent can discover novel solutions. A multiobjective reward function is introduced to maximize two scores simultaneously (SP, TC). Given the potential performance challenges arising from a rigid definition of allowed steps per episode, we introduce two strategies. These strategies allow the agent to stop the alignment at any point over a maximum number of allowed steps relative to the size of each input shape. Finally, we introduce variability in the training data by including a proportion of randomly generated sequences. Our strategy can potentially scale up to bigger MSAs with larger action spaces and longer episode step horizons. We enable MHA to enhance our model's ability to learn complex representations and understand the inherent sequential patterns. We take advantage of a Glimpse-Pointing mechanism based on Cross-Multihead Attention and Singlehead Attention. The Glimpse step introduces extra depth to our decoder. The Pointing mechanism produces the probability distribution over the MSA. Positional encodings are uti-

lized to introduce positional information to the model. This architecture provides flexibility and compatibility with variable MSA shapes, addressing a challenge that CNN-based approaches, as seen in related work like Ramakrishnan et al. (2018), may encounter.
In our evaluation, we compare with three well-known aligner tools. We rely on SP and TC scores, which are found to be common practice in many research studies. The results are promising, with our algorithm exhibiting the same or even better performance in TC scores of at least 76% in all cases and SP scores in 39%, 68%, and 75% of the cases for Clustal Omega, MAFFT, and MUSCLE respectively. Moreover, we exceed MAFFT and MUSCLE in the average and sum SP score, and we are close to Clustal Omega with an average difference of approximately 2 matching points considering SP.
While our approach is promising, it is essential to report certain limitations. This step creates space for future considerations and refinements that may enhance the overall performance.

6.1 Limitations and Future Considerations

One major limitation of this project is the absence of examining reference datasets. Unlike our competitors, who include well-known reference alignments in their evaluation process, we did not employ such a strategy. The primary reason is that those datasets comprise protein sequences rather than DNA sequences. While our algorithm is suitable for training using protein sequences, we restricted training in DNA sequences to control the experiment plan. To explore diverse settings within a constrained timeline and conduct multiple trainings, we limited the MSAs to 4-5 sequences of length 10. In future work, we suggest gradually increasing these sizes, especially the length of the sequences, to investigate how effective our approach is in more and bigger sequences. This exploration would provide a valuable indicator of the extent to which the attention mechanism can effectively capture long dependencies. Our algorithm is designed with extensive parameterization, allowing for easy experimentation with different input sizes.
Next, given the complexity of our approach, which involves multiple adjustable parameters, it is hard to make solid conclusions about the impact of the parameters on the algorithm. As this work represents a master's thesis, training on numerous sets of configurations was not feasible. Future work should involve training on a more diverse set of configurations to gain a better understanding of the parameters' effect on the performance. Moreover, longer training episodes could be beneficial for

revealing patterns that might not be evident in 80k episodes.
Further research could investigate alternative strategies for positional encodings, such as sine and cosine functions of different frequencies. Finally, we could explore a masking strategy for the undesired actions instead of our invalid action penalty strategy.

Additional Information

Hardware and Training Time

One NVIDIA RTX A5000 Graphics Card and a set of 47 CPUs were utilized for the experiments. These CPUs have a minimum of 1500.000 MHz and a maximum of 3737.890 MHz. The model has 7122433 parameters Each experiment setting required approximately two days to complete.

Data-Code Availability

This project was developed using Python 3.10 and Pytorch (Paszke et al., 2017). For training purposes, Docker containers were employed, utilizing the 'pytorch/pytorch:2.0.1-cuda11.7-cudnn8-devel' image from the docker-hub PyTorch images repository. The project's required libraries can be found listed in the 'requirements.txt' file, accessible on GitHub. The whole project can be found on Github at the following: URL.

Bibliography

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, *215*, 403–410. https://doi.org/10.1016/S0022-2836(05)80360-2

Aniba, M. R., Poch, O., Marchler-bauer, A., & Thompson, J. D. (2010). Alexsys: A knowledge-based expert system for multiple sequence alignment construction and analysis. *Nucleic Acids Research*, *38*, 6338. https://doi.org/10.1093/NAR/GKQ526

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, *34*, 26–38. https://api.semanticscholar.org/CorpusID:4884302

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. https://arxiv.org/abs/1607.06450v1

Bacon, D. J., & Anderson, W. F. (1986). Multiple sequence alignment. *Journal of Molecular Biology*, *191*, 153–161. https://doi.org/10.1016/0022-2836(86)90252-4

Bahdanau, D., Cho, K. H., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. https://arxiv.org/abs/1409.0473v7

Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *5th International Conference on Learning Representations, ICLR 2017 - Workshop Track Proceedings*. https://arxiv.org/abs/1611.09940v3

Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. *Neurocomputing*, 227–236. https://doi.org/10.1007/978-3-642-76153-9_28

Callaway, E. (2023). How alphafold and other ai tools could help us prepare for the next pandemic. *Nature*, *622*, 440–441. https://doi.org/10.1038/D41586-023-03201-4

Carrillo, H., & Lipman, D. (1988). The multiple sequence alignment problem in biology. *SIAM Journal on Applied Mathematics*, *48*, 1073–1082. https://doi.org/10.1137/0148063

Chapman, B., & Chang, J. (2000). Biopython: Python tools for computation biology. http://www.bris.ac.uk/Depts/Chemistry/MOTM/

Chaslot, G., Uiterwijk, J., Bouzy, B., & Herik, H. (2006). Monte-carlo strategies for computer go. *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence.*

Coulom, R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. *5th International Conference on Computer and Games,* 72–83. https://doi.org/10.1007/978-3-540-75538-8_7/COVER

Danihelka, I., Guez, A., Schrittwieser, J., & Silver, D. (2022). Policy improvement by planning with gumbel.

Darnell, S. (2020). Why structure prediction matters. https://www.dnastar.com/blog/protein-analysis-modeling/why-structure-prediction-matters/

Do, C. B., Mahabhashyam, M. S., Brudno, M., & Batzoglou, S. (2005). Probcons: Probabilistic consistency-based multiple sequence alignment. *Genome Research*, *15*, 330. https://doi.org/10.1101/GR.2821705

Dotan, E., Belinkov, Y., Avram, O., Wygoda, E., Ecker, N., Alburquerque, M., Keren, O., Loewenthal, G., & Pupko, T. (n.d.). Multiple sequence alignment as a sequence-to-sequence learning problem.

Dylus, D., Altenhoff, A., Majidian, S., Sedlazeck, F. J., & Dessimoz, C. (2023). Inference of phylogenetic trees directly from raw sequencing reads using read2tree. *Nature Biotechnology 2023*, 1–9. https://doi.org/10.1038/s41587-023-01753-4

Edelkamp, S., & Tang, Z. (2015). Monte-carlo tree search for the multiple sequence alignment problem. *Proceedings of the International Symposium on Combinatorial Search*, *6*, 9–17. https://doi.org/10.1609/SOCS.V6I1.18359

Edgar, R. C. (2004). Muscle: Multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, *32*, 1792–1797. https://doi.org/10.1093/NAR/GKH340

Fang, X., Wang, F., Liu, L., He, J., Lin, D., Xiang, Y., Zhu, K., Zhang, X., Wu, H., Li, H., & Song, L. (2023). A method for multiple-sequence-alignment-free protein structure prediction using a protein language model. *Nature Machine Intelligence 2023 5:10*, *5*, 1087–1096. https://doi.org/10.1038/s42256-023-00721-6

Feng, D. F., & Doolittle, R. F. (1987). Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *25*, 351–360. https://doi.org/10.1007/BF02603120

Frew, J. W., Hawkes, J. E., Sullivan-Whalen, M., Gilleaudeau, P., & Krueger, J. G. (2019). Inter-rater reliability of phenotypes and exploratory genotype-phenotype analysis in inherited hidradenitis suppurativa. *The British journal of dermatology*, *181*, 566–571. https://doi.org/10.1111/BJD.17695

From AI to protein folding: Our Breakthrough runners-up. (2016). *Science*. https://www.science.org/content/article/ai-protein-folding-our-breakthrough-runners

Gotoh, O. (1996). Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments. *Journal of molecular biology*, *264*, 823–838. https://doi.org/10.1006/JMBI.1996.0679

Graves, A., Wayne, G., & Danihelka, I. (2014). Neural turing machines. https://arxiv.org/abs/1410.5401v2

Gumbel, E. J. (1954). Statistical theory of extreme values and some practical applications : A series of lectures. https://api.semanticscholar.org/CorpusID:125881359

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-December*, 770–778. https://doi.org/10.1109/CVPR.2016.90

Heinzinger, M., Rost, B., & Steinegger, M. (2022). *How to speak protein? - representation learning for protein prediction*. Universitätsbibliothek der TU München. https://books.google.de/books?id=Zr3QzwEACAAJ

Hendrycks, D., & Gimpel, K. (2016). Gaussian error linear units (gelus). https://arxiv.org/abs/1606.08415v5

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hirschberg, J., & Manning, C. D. (2019). Advances in natural language processing. *Science*. http://science.sciencemag.org/

Ivo Van Walle, L. W., Ignace Lasters. (2005). Sabmark—a benchmark for sequence alignment that covers the entire known fold space. *Bioinformatics*, *21*(7), 1267–1268.

Jafari, R., Javidi, M. M., & Rafsanjani, M. K. (2019). Using deep reinforcement learning approach for solving the multiple sequence alignment problem. *SN Applied Sciences*, *1*. https://doi.org/10.1007/s42452-019-0611-4

Jessen, L. E., Hoof, I., Lund, O., & Nielsen, M. (2013). Signisite: Identification of residue-level genotype-phenotype correlations in protein multiple sequence alignments. *Nucleic acids research*, *41*(W1), W286–W291.

Joeres, R. (2021). Multiple sequence alignment using deep reinforcement learning.

Ju, F., Zhu, J., Shao, B., Kong, L., Liu, T. Y., Zheng, W. M., & Bu, D. (2021). Copulanet: Learning residue co-evolution directly from multiple sequence alignment for protein structure prediction. *Nature Communications 2021 12:1*, *12*, 1–9. https://doi.org/10.1038/s41467-021-22869-8

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., … Hassabis, D. (2021). Highly accurate protein structure prediction with alphafold. *Nature 2021 596:7873*, *596*, 583–589. https://doi.org/10.1038/s41586-021-03819-2

Karaboğa, D. (2005). An idea based on honey bee swarm for numerical optimization. *ournal of Computer and Communications, Vol.2 No.4, March 18, 2014*.

Karnin, Z., Koren, T., & Somekh, O. (2013, June). Almost optimal exploration in multi-armed bandits. In S. Dasgupta & D. McAllester (Eds.), *Proceedings of the 30th international conference on machine learning* (pp. 1238–1246, Vol. 28). PMLR. https://proceedings.mlr.press/v28/karnin13.html

Katoh, K., Misawa, K., Kuma, K. I., & Miyata, T. (2002). Mafft: A novel method for rapid multiple sequence alignment based on fast fourier transform. *Nucleic acids research*, *30*, 3059–3066. https://doi.org/10.1093/NAR/GKF436

Kingma, D. P., & Ba, J. L. (2014). Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. https://arxiv.org/abs/1412.6980v9

Klug, W. S., Spencer, C. A., Cummings, M. R., & Palladino, M. A. (2012). *Essentials of genetics, global edition*. Pearson UK.

Kocsis, L., & Szepesvári, C. (2006). Bandit based monte-carlo planning. *Machine Learning: ECML 2006*, 282–293.

Konagurthu, A. S., & Stuckey, P. J. (2006). Optimal sum-of-pairs multiple sequence alignment using incremental carrillo and lipman bounds. *JOURNAL OF COMPUTATIONAL BIOLOGY*, *13*.

Kool, W., van Hoof, H., & Welling, M. (2018). Attention, learn to solve routing problems! *7th International Conference on Learning Representations, ICLR 2019*. https://arxiv.org/abs/1803.08475v3

Kool, W., van Hoof, H., & Welling, M. (2019). Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement.

Kool, W., van Hoof, H., & Welling, M. (2020). Ancestral gumbel-top-k sampling for sampling without replacement. *Journal of Machine Learning Research*, *21*(47), 1–36. http://jmlr.org/papers/v21/19-985.html

Kunzmann, P., & Hamacher, K. (2018). Biotite: A unifying open source computational biology framework in python. *BMC Bioinformatics*, *19*, 1–8. https://doi.org/10.1186/S12859-018-2367-Z/FIGURES/6

Lall, A., & Tallur, S. (2023). Deep reinforcement learning-based pairwise dna sequence alignment method compatible with embedded edge devices. *Scientific Reports 2023 13:1*, *13*, 1–10. https://doi.org/10.1038/s41598-023-29277-6

Lassmann, T., Frings, O., & Sonnhammer, E. L. (2009). Kalign2: High-performance multiple alignment of protein and nucleotide sequences allowing external features. *Nucleic Acids Research*, *37*, 858. https://doi.org/10.1093/NAR/GKN1006

Liu, Y., Yuan, H., Zhang, Q., Wang, Z., Xiong, S., Wen, N., & Zhang, Y. (2023). Multiple sequence alignment based on deep reinforcement learning with self-attention and positional encoding (P. Robinson, Ed.). *Bioinformatics*, *39*. https://doi.org/10.1093/BIOINFORMATICS/BTAD636

Llados, J., Cores, F., Guirado, F., & Lérida, J. L. (2021). Accurate consistency-based msa reducing the memory footprint. *Computer Methods and Programs in Biomedicine*, *208*, 106237. https://doi.org/10.1016/J.CMPB.2021.106237

Luce, R. D. (1959). Individual choice behavior: A theoretical analysis. *Journal of the Royal Statistical Society. Series A (General)*, *123*. https://doi.org/10.2307/2343282

Luong, M. T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing*, 1412–1421. https://doi.org/10.18653/v1/d15-1166

Mircea, I. G., Bocicor, M. I., & Dîincu, A. (2014). On reinforcement learning based multiple sequence alignment. *Studia Universitatis Babes-Bolyai, Informatica.* https://www.cs.ubbcluj.ro/~studia-i/contents/2014-2/04-MirceaBocicorDincu.pdf

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. https://arxiv.org/abs/1312.5602v1

Modha, S., Thanki, A. S., Cotmore, S. F., Davison, A. J., & Hughes, J. (2018). Victree: An automated framework for taxonomic classification from protein sequences. *Bioinformatics*, *34*, 2195. https://doi.org/10.1093/BIOINFORMATICS/BTY099

Morgenstern, B. (2004). Dialign: Multiple dna and protein sequence alignment at bibiserv. *Nucleic Acids Research*, *32*, W33. https://doi.org/10.1093/NAR/GKH373

Murata, M., Richardson, J. S., & Sussman, J. L. (1985). Simultaneous comparison of three protein sequences. *Proceedings of the National Academy of Sciences of the United States of America*, *82*, 3073–3077. https://doi.org/10.1073/PNAS.82.10.3073

Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011). Natural language processing: An introduction. https://doi.org/10.1136/amiajnl-2011-000464

Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, *48*, 443–453. https://doi.org/10.1016/0022-2836(70)90057-4

Noorden, R. V., Maher, B., & Nuzzo, R. (2014). The top 100 papers. *Nature*, *514*, 550–553. https://doi.org/10.1038/514550a,pmid:25355343

Notredame, C., Higgins, D. G., & Heringa, J. (2000). T-coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of molecular biology*, *302*, 205–217. https://doi.org/10.1006/JMBI.2000.4042

O'Sullivan, O., Zehnder, M., Higgins, D., Bucher, P., Grosdidier, A., & Notredame, C. (2003). Apdb: A novel measure for benchmarking sequence alignment methods without reference alignments. *Bioinformatics*, *19*, i215–i221. https://doi.org/10.1093/BIOINFORMATICS/BTG1029

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., Facebook, Z. D., Research, A. I., Lin, Z., Desmaison, A., Antiga, L., Srl, O., & Lerer, A. (2017, October). Automatic differentiation in pytorch.

Pearson, W. R., & Lipman, D. J. (1988). Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences of the United States of America*, *85 8*, 2444–8. https://api.semanticscholar.org/CorpusID:15966318

Pevsner, J. (2009). Bioinformatics and functional genomics, second edition. *Bioinformatics and Functional Genomics, Second Edition*, 190–192. https://doi.org/10.1002/9780470451496

Ramakrishnan, R. K., Singh, J., & Blanchette, M. (2018). Rlalign: A reinforcement learning approach for multiple sequence alignment. *Proceedings - 2018 IEEE 18th International Conference on Bioinformatics and Bioengineering, BIBE 2018*, 61–66. https://doi.org/10.1109/BIBE.2018.00019

Ranwez, V., & Chantret, N. N. (2020). Strengths and limits of multiple sequence alignment and filtering methods. In C. Scornavacca, F. Delsuc, & N. Galtier (Eds.), *Phylogenetics in the genomic era* (2.2:1–2.2:36). No commercial publisher — Authors open access book. https://hal.science/hal-02535389

Rasmussen, T. K., & Krink, T. (2003). Improved hidden markov model training for multiple sequence alignment by a particle swarm optimization - evolutionary algorithm hybrid. *BioSystems*, *72*, 5–17. https://doi.org/10.1016/S0303-2647(03)00131-X

Roshan, U., & Livesay, D. R. (2006). Probalign: Multiple sequence alignment using partition function posterior probabilities. *Bioinformatics*, *22*, 2715–2721. https://doi.org/10.1093/BIOINFORMATICS/BTL472

Rosin, C. D. (2011). Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, *61*(3), 203–230. https://doi.org/10.1007/s10472-011-9258-6

Rubio-Largo, Á., Vega-Rodríguez, M. A., & González-Álvarez, D. L. (2016). Hybrid multiobjective artificial bee colony for multiple sequence alignment. *Applied Soft Computing*, *41*, 157–168. https://doi.org/10.1016/J.ASOC.2015.12.034

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., & Silver, D. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature 2020 588:7839*, *588*, 604–609. https://doi.org/10.1038/s41586-020-03051-4

Schuler, G. D., Epstein, J. A., Ohkawa, H., & Kans, J. A. (1996). [10] entrez: Molecular biology database and retrieval system. *Methods in Enzymology*, *266*, 141–162. https://doi.org/10.1016/S0076-6879(96)66012-1

Sievers, F., & Higgins, D. G. (2018). Clustal omega for making accurate alignments of many protein sequences. *Protein science : a publication of the Protein Society*, *27*, 135–145. https://doi.org/10.1002/PRO.3290

Sievers, F., Wilm, A., Dineen, D., Gibson, T. J., Karplus, K., Li, W., Lopez, R., McWilliam, H., Remmert, M., Söding, J., Thompson, J. D., & Higgins, D. G. (2011). Fast, scalable generation of high-quality protein multiple sequence alignments using clustal omega. *Molecular systems biology*, *7*. https://doi.org/10.1038/MSB.2011.75

Silver, D. (2015). Lectures on reinforcement learning.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, *529*(7587), 484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, *362*, 1140–1144. https://doi.org/10.1126/SCIENCE.AAR6404/SUPPL_FILE/AAR6404_DATAS1.ZIP

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Driessche, G. V. D., Graepel, T., & Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature 2017 550:7676*, *550*, 354–359. https://doi.org/10.1038/nature24270

Simossis, V. A., & Heringa, J. (2005). Praline: A multiple sequence alignment toolbox that integrates homology-extended and secondary structure information.

Nucleic Acids Research, *33*, W289–W294. https://doi.org/10.1093/nar/gki390

Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of molecular biology*, *147*, 195–197. https://doi.org/10.1016/0022-2836(81)90087-5

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (Second). MIT press.

Sutton, R. S., Mcallester, D., Singh, S., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, *12*, 1057–1063.

Świechowski, M., Godlewski, K., Sawicki, B., & Mańdziuk, J. (2023). Monte carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, *56*, 2497–2562. https://doi.org/10.1007/s10462-022-10228-y

Thadani, N. N., Gurev, S., Notin, P., Youssef, N., Rollins, N. J., Ritter, D., Sander, C., Gal, Y., & Marks, D. S. (2023). Learning from prepandemic data to forecast viral escape. *Nature 2023 622:7984*, *622*, 818–825. https://doi.org/10.1038/s41586-023-06617-0

Thompson, J. D., Higgins, D. G., & Gibson, T. J. (1994). Clustal w: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, *22*, 4673–4680. https://doi.org/10.1093/NAR/22.22.4673

Thompson, J. D., Plewniak, F., & Poch, O. (1999). A comprehensive comparison of multiple sequence alignment programs. *Nucleic acids research*, *27*(13), 2682–2690.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *2017-December*, 5999–6009. https://arxiv.org/abs/1706.03762v7

Vieira, T. (2014). Gumbel-max trick and weighted reservoir sampling — graduate descent. https://timvieira.github.io/blog/post/2014/08/01/gumbel-max-trick-and-weighted-reservoir-sampling/

Vinyals, O., Bengio, S., & Kudlur, M. (2016). Order matters: Sequence to sequence for sets. *4th International Conference on Learning Representations, ICLR 2016,San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings.*

Wang, L., & Jiang, T. (1994). On the complexity of multiple sequence alignment. *Journal of computational biology : a journal of computational molecular cell biology*, *1*, 337–348. https://doi.org/10.1089/CMB.1994.1.337

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning 1992 8:3*, *8*, 229–256. https://doi.org/10.1007/BF00992696

Yamada, S., Gotoh, O., & Yamana, H. (2006). Improvement in accuracy of multiple sequence alignment using novel group-to-group sequence alignment algorithm with piecewise linear gap cost. *BMC Bioinformatics*, *7*, 524. https://doi.org/10.1186/1471-2105-7-524

Yellott, J. I. (1977). The relationship between luce's choice axiom, thurstone's theory of comparative judgment, and the double exponential distribution. *Journal of Mathematical Psychology*, *15*, 109–144. https://doi.org/10.1016/0022-2496(77)90026-8

Appendix A

Appendices

A.1 Exploration - Exploitation

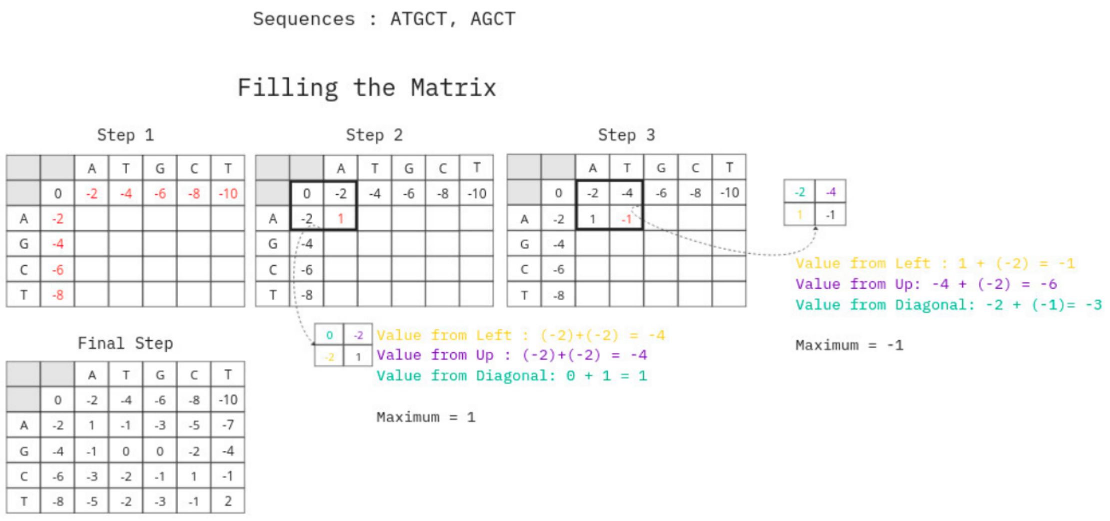
A common dilemma when it comes to RL is the exploration-exploitation dilemma. Imagine trying to win an advanced opponent in a cardboard game. In this case, the expected behavior would be to stick with familiar, reliable moves for a better chance of winning. This tactic, known as exploitation, involves exploiting known strategies to maximize the probability of winning. However, occasionally, trying new, untested moves or strategies is beneficial to improve in a game. Similarly, in RL, for maximizing their rewards, agents should pick actions that have already been experienced and found promising. On the other hand, if an agent is overly conservative and avoids exploration, it might converge to suboptimal solutions and fail to discover the best strategies. Exploration may produce greater cumulative rewards in the long run, so there is a need for balance between those two.

A.2 Example of Needlman-Wunsch algorithm usage

Here, an example of Needleman-Wunsch for pairwise alignment is illustrated in Figures A.1 and A.2. First, a matrix D is initialized. In our example, sequences have lengths 4 and 5, respectively, resulting in a matrix of shape $(4+1) \times (5+1) = 5 \times 6$. A scoring schema is introduced where the gap penalty equals -2, a match of letters is 1, and a mismatch is -1. The first row and column of the matrix D are filled (in red) with a gap score added to the previous cell of the row or column, respectively. The next step involves the filling of the matrix with a direction starting from the upper left. The strategy formulated in Equation A.1 is followed for filling each matrix cell (i, j) starting from (2,2).

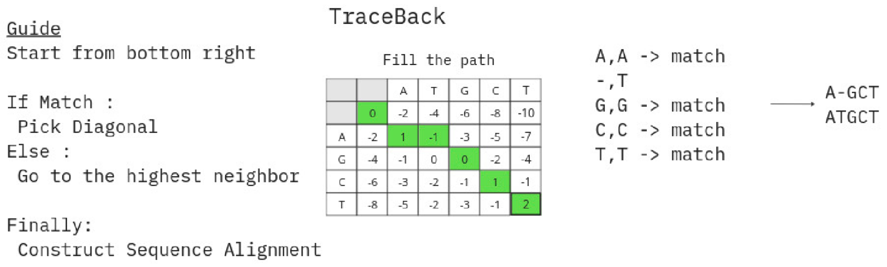
$$D(i,j) = \max \begin{cases} D(i-1, j-1) + sub_{score}(i,j) \\ D(i-1, j) - 2 \\ D(i, j-1) - 2 \end{cases} \tag{A.1}$$

where sub_{score} is the substitution score for letters i and j.
The final step, known as the traceback step, begins from the bottom-right corner of the matrix. During this step, a path is created according to the following rules: In case of a match between letters such as $M(i,0) = M(0,j)$ for cell i, j, a diagonal step is made. In any other case, we follow the highest neighbor.


Sequences : ATGCT, AGCT
Filling the Matrix
Step 1
Step 2
Step 3
Final Step
Value from Left : (-2)+(-2) = -4
Value from Up : (-2)+(-2) = -4
Value from Diagonal: 0 + 1 = 1
Maximum = 1
Value from Left : 1 + (-2) = -1
Value from Up: -4 + (-2) = -6
Value from Diagonal: -2 + (-1)= -3
Maximum = -1

		A	T	G	C	T
	0	-2	-4	-6	-8	-10
A	-2	1	-1	-3	-5	-7
G	-4	-1	0	0	-2	-4
C	-6	-3	-2	-1	1	-1
T	-8	-5	-2	-3	-1	2

Figure A.1: Example dynamic pairwise global alignment approach by Needleman and Wunsch (1970). Step 1-2



```
Guide
Start from bottom right

If Match :
 Pick Diagonal
Else :
 Go to the highest neighbor

Finally:
 Construct Sequence Alignment
```

TraceBack

Fill the path

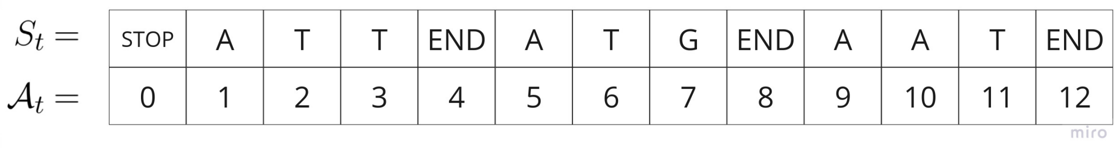
		A	T	G	C	T
	🟩 0	-2	-4	-6	-8	-10
A	-2	🟩 1	🟩 -1	-3	-5	-7
G	-4	-1	0	🟩 0	-2	-4
C	-6	-3	-2	-1	🟩 1	-1
T	-8	-5	-2	-3	-1	🟩 2

```
A,A -> match
-,T
G,G -> match        ──→   A-GCT
C,C -> match              ATGCT
T,T -> match
```

Figure A.2: Example dynamic pairwise global alignment approach by Needleman and Wunsch (1970). Step 3

A.3 Example of Available Actions

In the example shown in Figure A.3, a state of three sequences to be aligned is given. The agent in the initial state can select an action for 0 to 12, where each action corresponds to a position in the state, indicating where a gap can be added. If the chosen action is 0, though, the agent selects the $< STOP >$ token, which will indicate the completion of the alignment process. The episode will then finish.

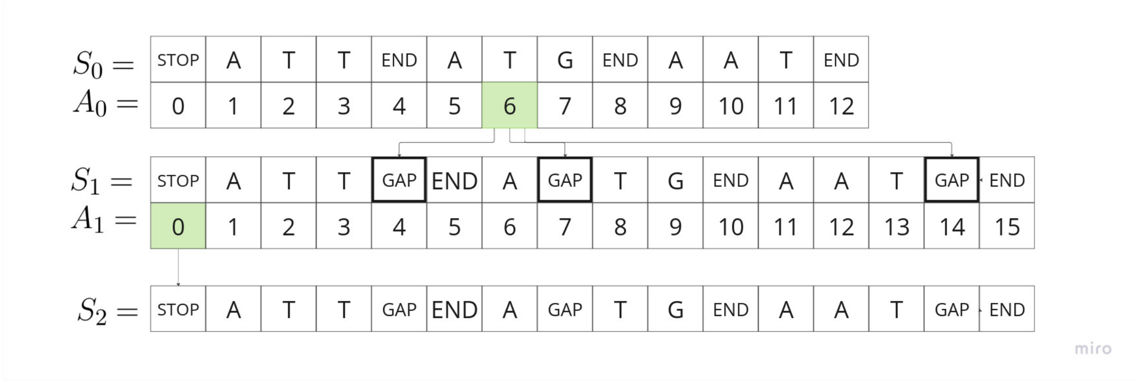


$S_t =$	STOP	A	T	T	END	A	T	G	END	A	A	T	END
$\mathcal{A}_t =$	0	1	2	3	4	5	6	7	8	9	10	11	12

Figure A.3: Example State - Action Space

A.4 Example of Transitions

The agent plays two steps in the example episode shown in Figure A.4. In the first step, the agent selects action six. This results in a gap insertion in position six in the state and a gap insertion at the end of all other sequences to match the maximum length. In the next timestep, the agent selects to stop the game with $A_1 = 0$. The episode ends, and final state is S_2.



$S_0 =$
$A_0 =$
STOP A T T END A T G END A A T END
0 1 2 3 4 5 6 7 8 9 10 11 12
$S_1 =$
$A_1 =$
STOP A T T GAP END A GAP T G END A A T GAP END
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
$S_2 =$
STOP A T T GAP END A GAP T G END A A T GAP END
miro

Figure A.4: Example Transitions

A.5 Example of usage of Read2Tree

A.6 Competitors evaluation strategies

A.6.1 MUSCLE

MUSCLE (Edgar, 2004) conducts a performance evaluation using four datasets: BALiBASE, SABmark (Ivo Van Walle, 2005), SMART, and PREFAB.
More specifically, version 2 of the BALiBASE benchmark is employed with reference sets Ref 1 - Ref 5. They also report that they selected version 1.63 of the SABmark alignments, with sets containing sequences of three (3) to twenty-five (25) sequences with an average length of 179 (Edgar, 2004). SABmark includes solely sequences with mid-low similarity (Ivo Van Walle, 2005).
Next, SMART is also utilized for evaluation, retaining a subset of 267 alignments. Each of these alignments contains an average of 31 sequences, with an average sequence length of 175.
Finally, PREFAB version 3.0, the dataset contains 1932 alignments averaging 49 sequences of length 240 (Edgar, 2004). To summarize the characteristics of the test sets used to evaluate the MUSCLE tool, we can observe that the average number of sequences per alignment is less than 49, with an average sequence length of around 200.
They compare their results with three alignment tools: ClustalW (Thompson et al., 1994), T-Coffee (Notredame et al., 2000), and MAFFT (Katoh et al., 2002). They

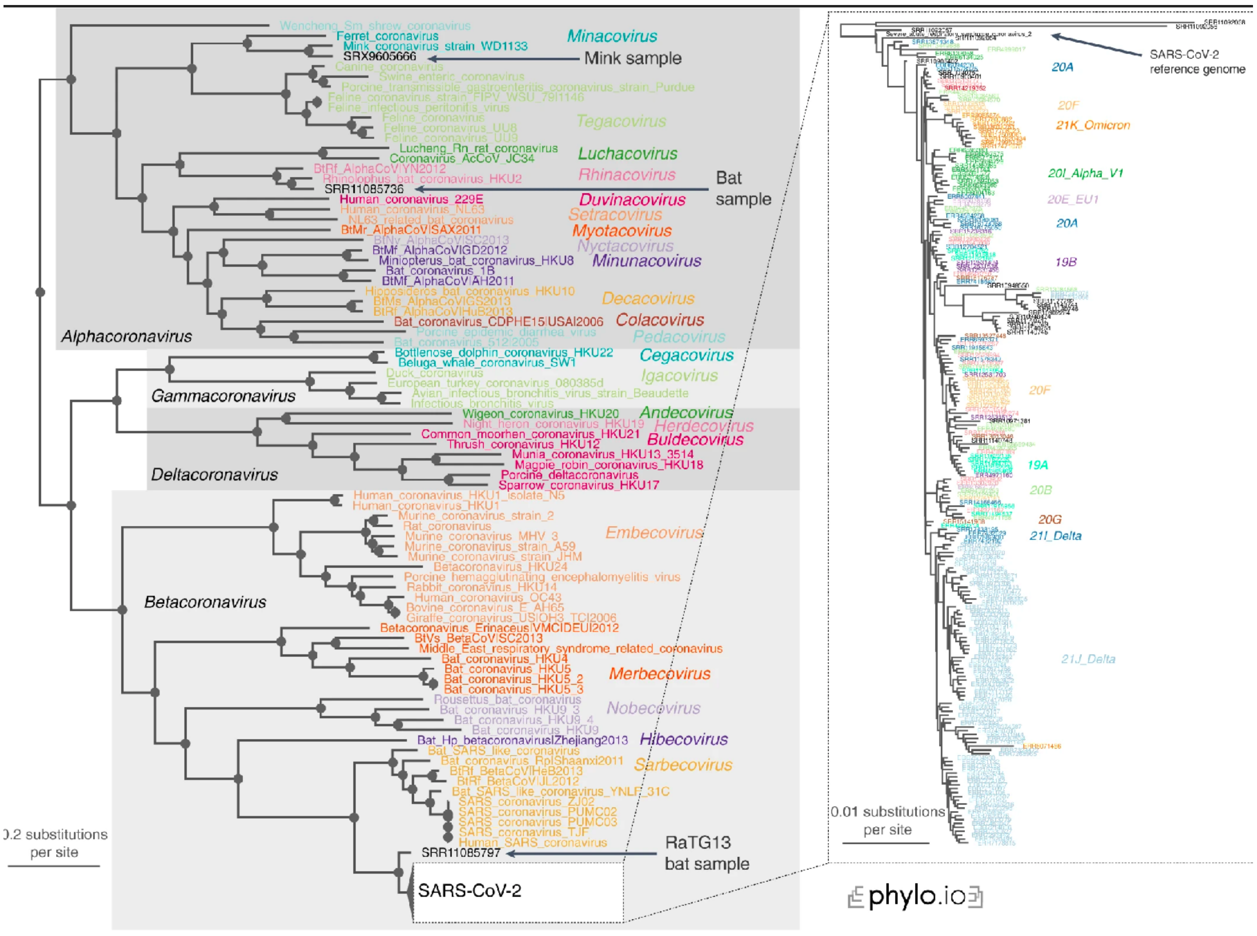


Figure A.5: Read2Tree correctly classifies the recent SARS-CoV-2 sequences, (Dylus et al., 2023)

also employ three specific metrics to measure accuracy: $\mathcal{SP}$ score, $\mathcal{TC}$ score, and APDB (O'Sullivan et al., 2003) . It's worth noting that APDB is unique in that it relies solely on structural information and does not require a reference alignment but is designed for protein sequences. SP and TC scores are calculated either on specific parts of the alignment (in BALiBASE) or in all columns (in SMART). Edgar (2004) does not define $\mathcal{SP}$ and $\mathcal{TC}$ but refers to Thompson et al. (1994) for both of them. For this $\mathcal{SP}$, residues matching gives 1 point, and in all other cases, it gives no points. $\mathcal{SP}$ is also slightly differently utilized, dividing it by the reference score alignment. More specifically, it can be defined as:

$$\mathcal{SPR} = \frac{\mathcal{SP}}{\mathcal{SP}_r} \tag{A.2}$$

where r refers to the reference alignment.

A.6.2 Clustal Omega

Clustal Omega is another widely known MSA tool that was released after Clustal W and Clustal X (Sievers & Higgins, 2018). With Clustal Omega, researchers aim to achieve scalability for large MSAs without sacrificing accuracy. For evaluating the tool's performance, they employ three benchmark datasets, namely BALiBASE, Prefab, and QuanTest, for testing large alignments. BALiBASE version 3, which contains 218 protein alignments of an average number of sequences per alignment 21 and sequence length from 88 to 8481. They also use Prefab, which is a collection of 1682 reference sequence pairs. They utilize $\mathcal{SPR}$ as seen in Equation A.2 and $\mathcal{TC}$ scores for evaluation.

A.6.3 MAFFT

MAFFT tool employs BALiBASE as a benchmark to calculate the accuracy of their proposed method. In their evaluation, they present $\mathcal{SP}$ and $\mathcal{TC}$ scores as metrics to measure performance. Likewise, as in MUSCLE, the author refers to (Thompson et al., 1999) for $\mathcal{SP}$ where residues match gives 1 point, and in all other cases, it gives no points. $\mathcal{SP}$ is also measured in comparison with the reference $\mathcal{SP}_r$ as a fraction, and it is the same as $\mathcal{SPR}$ (Equation A.2).

A.7 Training Loop

In Figure A.6, the high level of the project's flow is depicted. Multiple workers are running in parallel on different machines. Each worker is continuously playing games, updating weights, and evaluating the latest model. All workers have access to the same network storage and replay buffer for saving samples, updating model checkpoints, and fetching the latest checkpoints. Moving to Figure A.7, we aim to capture all the functionality that occurs inside a single worker.

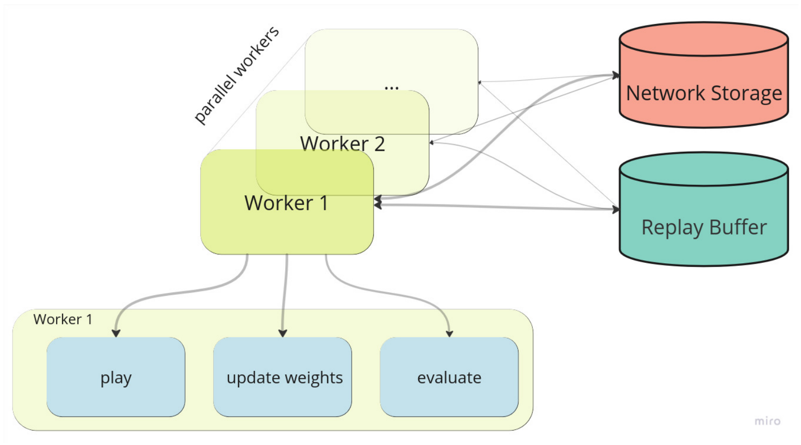


Figure A.6: Level 1. Workers running on parallel

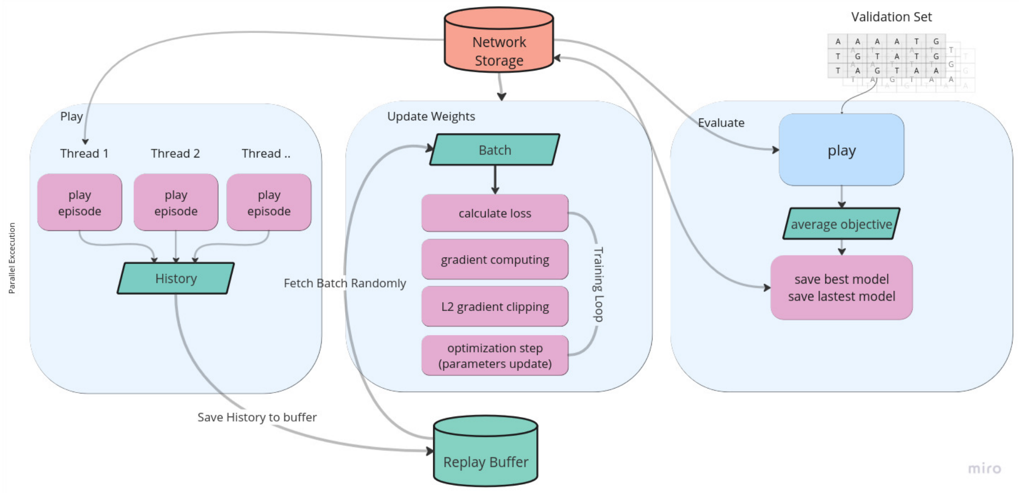


Figure A.7: Level 2. Experience Worker High-Level Architecture