Apprentissage automatique pour l'évolution moléculaire

Luca Nesterenko

Université Claude Bernard Lyon 1

THESE de DOCTORAT DE

L'UNIVERSITE CLAUDE BERNARD LYON 1

Ecole Doctorale 341

E2M2 - Evolution Ecosystèmes Microbiologie Modélisation

Discipline : Apprentissage profond

Soutenue publiquement le 18/11/2024, par :

Luca Nesterenko

Apprentissage automatique pour l'evolution moléculaire

Devant le jury composé de :

Cocco, Simona	Directrice de recherche, CNRS Paris	Présidente
Scornavacca, Celine	Directrice de recherche, CNRS Montpellier	Rapporteure
Robin, Stéphane	Professeur des universités, Sorbonne Université Paris	Rapporteur
Pupko, Tal	Professeur, Université Tel Aviv Israël	Rapporteur
Gueguen, Laurent	Maître de conférences, Université Lyon 1	Examinateur
Jay, Flora	Chargée de recherche, CNRS Gif-sur-Yvette	Examinatrice
Boussau, Bastien	Directeur de recherche, CNRS Lyon	Directeur de thèse
Jacob, Laurent	Directeur de recherche, CNRS Paris	Co-directeur de thèse

Luca Nesterenko

Deep learning for molecular evolution

PhD Thesis

Université Claude Bernard Lyon 1

November 18th, 2024

"*Phylogenetics illuminates the tapestry of life, weaving threads from the past into the fabric of the present, offering us a glimpse into the evolutionary dance that connects all living beings.*"

GPT 3.5

Contents

Resumé

Comprendre l'histoire évolutive d'un groupe d'organismes est une tâche centrale en biologie. En particulier, étant donné un ensemble de séquences codant pour la même protéine chez plusieurs espèces, un objectif important est de reconstruire l'arbre décrivant leur évolution à partir d'un ancêtre commun. Bien qu'étant une étape cruciale dans plusieurs pipelines bioinformatiques, cela représente un problème difficile en soi, car le nombre d'arbres possibles augmente de manière superexponentielle avec le nombre d'espèces.

Les méthodes de pointe reposent sur des modèles probabilistes de l'évolution des séquences et cherchent à maximiser la vraisemblance de l'arbre correspondant. Cette stratégie n'est réalisable qu'avec des modèles très simplifiés. De plus, elle est extrêmement coûteuse en termes de calculs et conduit parfois à des estimations imprécises, notamment dans le cas de mauvaise specification du modèle.

D'un autre côté, les simulations permettent de générer facilement de grandes quantités de données à partir de ces modèles. L'objectif principal de cette thèse a été d'explorer une approche d'apprentissage supervisé pour résoudre ce problème dans un cadre d'inférence basé sur la simulation, sans recours à la vraisemblance. Au lieu de maximiser la vraisemblance d'un modèle d'évolution des séquences, nous avons généré des arbres phylogénétiques ainsi que des séquences ayant évolué selon ces modèles, et les avons utilisés pour apprendre une fonction, paramétrée par un réseau de neurones profond, qui transforme un ensemble de séquences homologues en un ensemble de distances évolutives.

L'arbre lui-même peut ensuite être reconstruit à partir de ces distances via les méthodes dites basées sur la distance. Bien que des progrès notables aient été réalisés ces dernières décennies pour améliorer ces méthodes, l'estimation des distances est encore généralement effectuée dans le cadre du maximum de vraisemblance par de simples comparaisons par paires, ce qui ne permet pas d'exploiter pleinement l'information contenue dans l'alignement multiple des séquences en entrée et qui conduit finalement à des précisions de reconstruction inférieures par rapport à une approche de maximum de vraisemblance complète. Le présent travail vise donc à combler cette lacune en proposant une prédiction conjointe de toutes les distances évolutives, en tirant parti des développements récents et des succès de l'apprentissage profond dans le traitement de données à haute dimension et de séquences.

Nous montrons que ce nouveau paradigme peut améliorer les méthodes de reconstruction phylogénétique existantes ou aboutir à des précisions similaires pour de grands ensembles d'espèces pour lesquelles les méthodes actuelles

seraient trop coûteuses en ressources. Cette approche ouvre également la voie à l'adoption de modèles d'évolution plus complexes et réalistes, pour lesquels l'inférence, avec les méthodes basées sur la vraisemblance, serait intractable.

Nous discutons des avantages et de la flexibilité offerts par l'architecture de réseau de neurones développée, qui peut facilement être adaptée pour traiter différentes tâches d'inférence biologique connexes, démontrant ainsi son efficacité dans l'analyse des données de séquences moléculaires.

Abstract

Understanding the evolutionary history of a group of organisms is a central task in biology. In particular, given a set of sequences encoding the same protein in multiple species, an important objective is to reconstruct the tree describing their evolution from a common ancestor. While being an important step in several bioinformatic pipelines this is a hard problem in itself given that with more and more species the number of possible trees grows superexponentially.

The state-of-the-art relies on probabilistic models of sequence evolution and seeks the tree that maximizes the corresponding likelihood. This strategy is only feasible with very simplified models. Moreover, it is very computationally expensive and sometimes leads to imprecise estimates, notably in the case of model misspecification.

However, via simulations, it is simple to sample large quantities of data from these models. The main objective of this thesis has been to explore a supervised learning approach to the problem in a likelihood-free, simulation based inference framework. Instead of maximizing the likelihood of a sequence evolution model, we generated phylogenetic trees as well as sequences having evolved according to these models, and used them to learn a function, parameterized by a deep neural network, that transforms a set of homologous sequences into a set of evolutionary distances. The tree itself can be reconstructed then from these distances via so called distance-based methods. While in the last decades noticeable progress has been made in improving these methods, the distance estimation in itself is typically still performed in the maximum likelihood framework via simple pairwise comparisons. These fail to fully exploit the information contained in the input multiple sequence alignment, eventually leading to worse reconstruction accuracies with respect to a full scale maximum likelihood approach. The present work then aims to fill this gap with a joint prediction of all evolutionary distances leveraging the recent developments and successes of deep learning in dealing with high dimensional and sequence data.

We show that this new paradigm can improve existing phylogenetic reconstruction methods or lead to similar accuracies on large sets of species for which existing methods would be too resource intensive. The approach also paves the way to the adoption of more complex and realistic evolution models under which inference, with existing likelihood-based methods, would be intractable. We discuss the advantages and flexibility provided by the developed neural network architecture which can easily be adapted to deal with different related biological inference tasks showcasing its effectiveness in dealing with molecular sequence data.

Outline of the thesis

Chapter 1, the introduction, aims at presenting the concepts and methods used throughout the thesis in a concise yet self-contained way, providing the context and the statistical framework the work is built upon. I firstly introduce the problem of phylogenetic reconstruction outlining the phylogenetician's framework with the commonly employed models of evolution and tree reconstruction methods discussing at once their advantages and limitations. A brief introduction of deep learning follows, with a particular emphasis on its applications in the fields related to molecular evolution, and on the basic building blocks of the neural network architectures that will be considered throughout this work. Finally a discussion of the recent advances that the deep learning revolution has brought on in the field of simulation-based inference, provides the main framework for our approach to phylogenetic reconstruction and ohter problems in molecular evolution.

In chapter 2 I will present and discuss the main work that I carried out throughout my thesis, the development of a neural network-based approach to the problem of phylogenetic reconstruction via the joint estimation of evolutionary distances between sequences inside a multiple alignment.

Chapter 3 stems from a project on which Estelle Bergiron, an M1 student, has worked on for her internship, cotutored by me, Bastien Boussau and Julien Barnier. In this work we adapted the developed neural network architecture for a different task, namely the detection of sites subject to a selection shift associated with a given phenotype.

Chapter 4 results from a joint project with Charlotte West, a PhD student at EBI in Cambridge. Again, we adapted our network architecture to tackle the main problem she's been working on, namely the gene-wise detection of positive selection. This allows us to once again showcase the flexibility of the developed approach working this time with nucleotide instead of amino acid data and allowing a direct comparison with a convolutional network trained for the same task.

Chapter 1

Introduction

1.1 Phylogenetic inference

The principle underlying phylogenetic reconstruction is that all living organisms share a common ancestor and, as species diverge over time, they accumulate differences. By analyzing these differences, researchers can infer a "family tree", or phylogeny, that reflects the underlying evolutionary history via successive branching patterns. While the idea of a phylogeny to represent the evolutionary relationships between different organism dates back to the mid 19th century (figure 1.1), it is the advent of high-throughput sequencing technologies that has revolutionized the field with the development of *molecular phylogenetics*. Working with sequence data instead of morphological characters provides a more fine-grained trace of the evolutionary history of a set of organisms and a more objective way to assess the differences they accumulated over time. The starting point of such an approach is typically a multiple sequence alignment (MSA), consisting in the retrieval from known databases of supposedly *homologous sequences*, that is evolutionary related through a common ancestor, and their subsequent arrangement in a matrix having as rows all the sequences. This is done in a way which maximizes the overall score of the alignment, specifically defined to ensure that each column corresponds to a particular site in the ancestor sequence, possibly subject to different mutations or insertion/deletion events through the course of evolution. Nowadays phylogenetic inference has widespread applications across numerous fields of biology with phylogenies being used for instance to trace the origins and track the spread of pathogens, to study the evolution of cancer cells, to prioritize species with unique evolutionary histories in conservation biology or to assist the discovery of novel enzymes in biotechnology.

While the massive amount of data provided by sequencing technologies promises to offer unprecedented insight into the evolutionary relationships among species and the mechanisms of evolution, it comes with its challenges, as the need for efficient computational methods to process and analyze large datasets. Indeed whereas complex models that aim at accurately capturing the evolutionary processes underlying data have been developed, their applicability remains limited, given the computational bottleneck impeding their use for large scale datasets. On the other hand the last decades have seen the rise of machine learning and in particular of its subfield of deep learning, which with its capabilities of leveraging massive amounts of high-dimensional data

to unravel their complex underlying relationships promises to revolutionize many fields, including phylogenetics.

The subject of the present manuscript is situated at the intersection of these two fields, molecular phylogenetics and deep learning, the goal of the following introduction will then be to provide the context and framework in which the thesis work has been carried on.

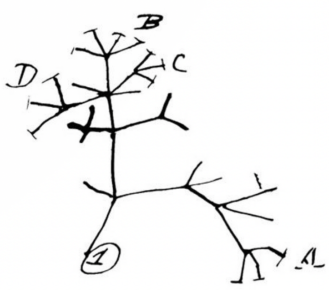


FIGURE 1.1: A phylogenetic tree drawn by Charles Darwin, in his first notebook on Transmutation of Species, 1837.

1.1.1 Phylogenetic trees

In phylogenetics one typically takes into account only binary trees as evolutionary events leading to the simultaneous divergence into more than two species are assumed to be highly unlikely and can be approximated by several bifurcations along a binary tree with arbitrarily small branches. *Polytomies*, nodes with more than two descendants can still be considered in the literature but this is typically done in order to represent uncertainty in the evolutionary relationship between different lineages, whereas *fully resolved* trees are always represented as binary, this will always be the case throughout this work.

Notation

Given a phylogenetic tree τ, a binary tree with each edge labelled with its length, a non-negative real number, we shall denote by $E(\tau)$ the set of its edges, which will often be referred to as branches, with $l(e)$ the length of a branch e and with $V(\tau)$ the set of the vertices (nodes) of τ. We will refer to branches connecting leaves to their parent nodes as terminal or external while denoting as internal the others, we shall also sometimes refer to the leaves as tips or taxa and to their number as the size of the tree.

Interpretation of branch lenghts

Although it would be natural to assume so, most commonly in phylogenetics the tree's branch lengths do not represent the time passed between one branching in the tree and the next (or the moment when the sequences are observed if the next node is a leaf) but rather the amount of evolutionary change that a sequence evolving along a branch has undergone, as we shall soon see, this allows allows typically employed models of molecular evolution to take into account different rates of evolution, i.e. different speeds at which sequences evolve, along different branches. In such a framework trees in which the rate of evolution is constant in each branch, referred to as *ultrametric*, are said to follow the *molecular clock hypothesis*, according to which the rate of evolutionary change is constant over time and across different lineages. This assumption does allow to interpret the branch lenghts as the time (eventually

multiplied by the constant rate) passed between successive branching events, but such a condition is rarely met in empirical data.

Number of topologies

A single topology is possible for an unrooted binary tree having 3 leaves, namely the one with a single internal node of degree 3 connected to each of them via a single branch. We can then notice that one can construct an unrooted binary tree (UBT) with n leaves from one with $n-1$, by choosing a branch on which a new node is placed along with the branch connecting it to the new leaf, and it is straightforward to see that every UBT on n leaves can be constructed in such a way (Fig. 1.2).

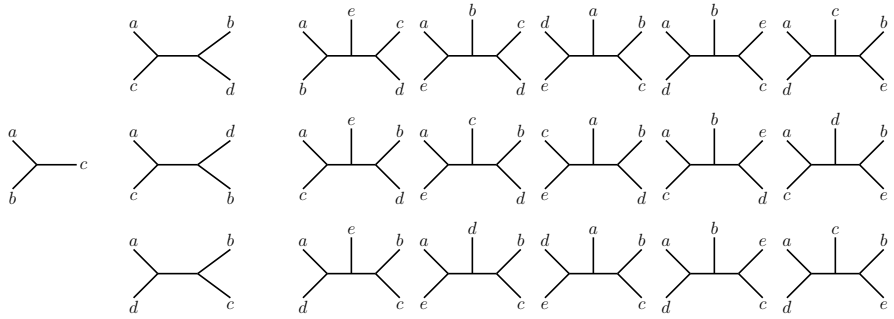


FIGURE 1.2: Enumeration via step-wise addition of all unrooted binary trees for increasing numbers of leaves.

Given that both the numbers of nodes and branches are increased by 2 in this way it easily follows by induction that an UBT with n leaves has $2n-2$ nodes and $2n-3$ branches. The same reasoning now allows us to enumerate all the UBTs with n leaves: Given that there are at each step $2(n-1)-3 = 2n-5$ choices for the branch along which to place the new node the number of topologies on n leaves is given by $\#\mathcal{T}_n = 3 \cdot 5 \cdots (2n-5) = (2n-5)!!$ (Figure 1.3). Given that rooting an UBT simply amounts to choosing a branch along which the root is placed, a rooted binary tree with n leaves has $2n-1$ nodes and $2n-2$ branches and the number of possible rooted binary topologies is given by $\#\hat{\mathcal{T}}_n = (2n-3)!!$.

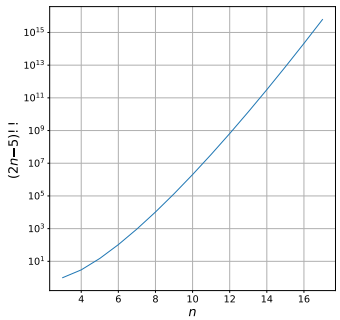


FIGURE 1.3: Super-exponential growth of the number of unrooted binary topologies as a function of n. One can see for instance that there are already more then a million possible topologies for a tree with 10 leaves.

Tree space exploration

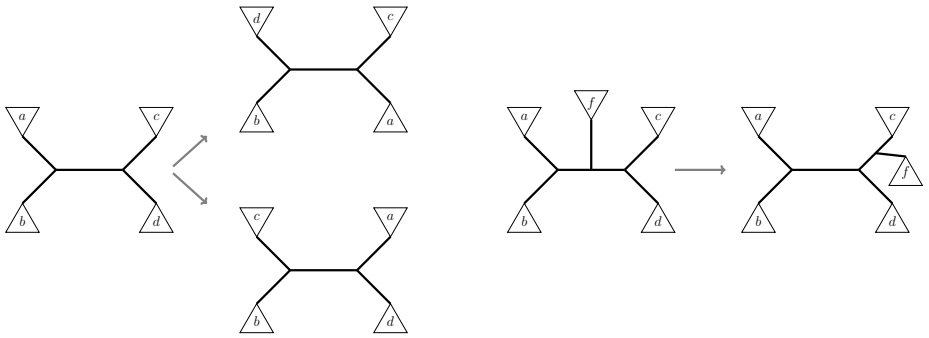


FIGURE 1.4: Two NNIs on the left and one SPR move on the right.

Given the vastness of the space of tree topologies, algorithms which seek the best phylogenetic tree optimizing a given criterion generally cannot perform an exhaustive search, testing each possible topology, except when working with very small datasets. Criterion-based phylogenetic reconstruction algorithms have then to resort to heuristics to partially explore the tree space. The most popular techniques used by these hill climbing algorithms to go from a tree to another are Nearest Neighbor Interchange (NNI) and Subtree Pruning and Regrafting (SPR) (Figure 1.4). The first consists in selecting an interior edge of the tree and swapping one of the two rooted subtrees stemming from one end of the edge with one of the two stemming from the other. Given that a tree $\tau \in \mathcal{T}_n$ has $n-3$ internal branches, the number of its NNI neighbors is equal to $2(n-3)$. An SPR tree rearrangement operation consists instead in selecting an edge of the tree τ, detaching it from the latter at only one end thus obtaining a pruned subtree, the latter is then reattached to τ, introducing a new node, along one the remaining edges. It is easy then to see that NNI moves are a special case of SPRs in which the pruned subtree is attached along the branch joining the considered edge to one of the subtrees which are on the other end of it. The number of SPR neighbors a tree $\tau \in \mathcal{T}_n$ has is $2(n-3)2(n-7)$ [1] therefore such moves allow a more thorough local tree space exploration, potentially allowing to escape local NNI minima.

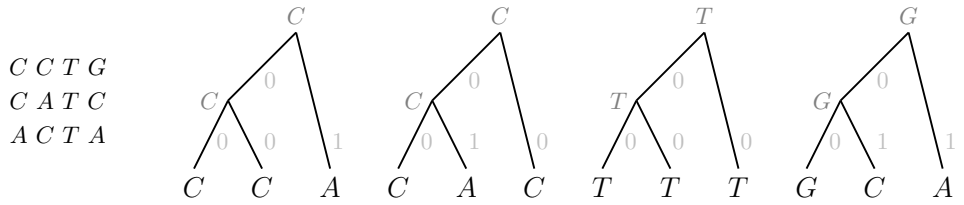
Tree rooting

Under the molecular clock hypothesis, assuming that all lineages have evolved with the same speed from their common ancestor, it is straightforward to place the latter, the root of the tree, as the node equidistant from all the leaves. We shall see that in the more general case, in which different rates of evolution are allowed across different branches, the common assumption of *time reversibility* made by the vast majority of evolution models implies that a priori the placement of the root cannot be identified. A common method to root a phylogenetic tree is then *outgroup rooting* which consists in including in the analysis a group of sequences known for a fact to be only distantly related to those under study, this entails that the root will necessarily find itself on the branch joining this outgroup to the other sequences. In this work we will use rooted trees to simulate the evolution of sequences along given phylogenies, on the other hand we will not consider the problem of the root placement when inferring trees: Whenever we will compare phylogenetic trees

these will always be considered as unrooted as all the metrics used to compare trees throughout the manuscript are independent of their root placement.

1.1.2 Parsimony

In order to find a tree that best fits the data we have at hand one needs first a criterion to choose among different trees. One of the first from an historical point of view, and most intuitive among such criteria is loosely based on the general principle of *Occam's razor* which can be stated in very simplistic terms as "The simplest explanation is usually the best one". In the context of phylogenetics this becomes "The preferred tree is the one that involves the minimum amount of evolutionary changes". This leads to the definition of a *parsimony score* [2] for a given phylogenetic tree topology where inferred ancestral states have been assigned to internal nodes, as the total number of observable character changes having occurred along the branches of the tree. In this framework the problem of phylogenetic reconstruction then becomes the quest for the tree topology and assignment of ancestral sequences that minimize this score. Beyond the limitations of such a rationale, which shall be further discussed, such an approach still suffers from the vastness of the tree topology space $\mathcal{T}$. Indeed, while for a given topology the assignment of the sequences to internal nodes which minimises the score (known as the small parsimony problem) can be computed efficiently ($\mathcal{O}(nL)$ time complexity) via dynamic programming with Fitch's algorithm [2], the minimization across all possible topologies is known to be an NP-hard problem [3][4] and in practice one has to recur to heuristics in order to find a tree that is at least a local optimum according to this criterion. Furthermore it is worth noting that the global optimum need not be unique as several trees can be equally parsimonious.



$$p(\tau) = 1 + 1 + 0 + 2$$

FIGURE 1.5: Example of parsimony score of a tree for a toy alignment.

The parsimony criterion is not statistically consistent, that is even with infinite amount of data (infinitely long sequences), one will not necessarily obtain the correct tree. The standard example for this behavior is the phenomenon known under the term *long branch attraction* (LBA): To show the inconsistency of parsimony Felsenstein [5] considered a tree such as that on the left in Figure 1.6 and showed that under a simple model of evolution, denoting by $P(xywz)$ the probability of observing the character pattern $xywz$ at any given site, if the two long branches are sufficiently longer than the three short ones, then $P(xyxy) > P(xxyy)$ where x and y are distinct characters. The former pattern induces maximum parsimony to prefer the wrong topology on the right of Figure 1.6, leading therefore, in the limit of infinite sequences, the method to wrongly infer the latter topology as the one underlying the

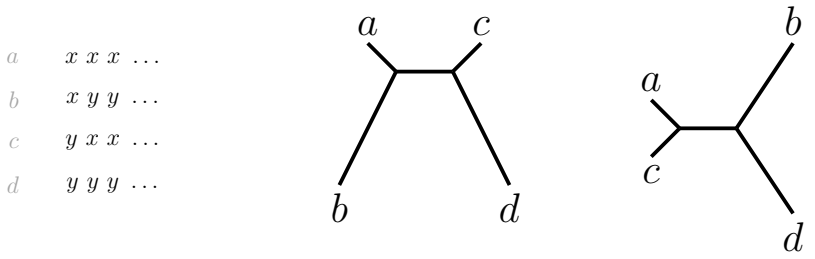


FIGURE 1.6: Example of long branch attraction, true tree $((a, b), (c, d))$; on the left and wrong tree $((a, c), (b, d))$; reconstructed by maximum parsimony in the limit of infinite sequence length on the right.

observed sequence data.

The faulty assumption underlying parsimony is that each site, assumed to be evolving independently, can undergo at most a single character change in the process of evolution of a sequence along a branch. Indeed as illustrated in figure 1.5 one can see the parsimony score of a tree in which the internal node have been annotated with ancestral states as the sum of the per-site tree lengths, where the length of a tree is defined as

$$l(\tau) = \sum_{e \in E(\tau)} l(e) \tag{1.1}$$

having made the assumption that $l(e) \in \{0, 1\}\ \forall e \in E(\tau)$. Equivalently this amounts to consider the length of the tree regarding all sites at once, assuming the length of an edge to be the *Hamming distance* between the sequences at its ends, that is the number of sites at which the two sequences differ. We will see that a more sophisticated method, minimum evolution 1.4.4, works by minimizing the same tree length function, in that case however, the use of an unbiased distance measure, which does take into account the possibility of multiple substitutions and back-mutations along the branch, will guarantee the statistical consistency of the method.

1.2 Probabilistic models of molecular evolution

As the process of evolution is generally assumed to be memoryless most often biological sequence evolution is modeled as a continuous-time Markov chain [6][1]. Furthermore, a widespread assumption is that the evolution process is *independent* and *identically distributed* (*i.i.d.*) at each site of the sequence so that the evolution of a protein sequence of length L is modeled by L i.i.d. Markov processes running along the branches of a common underlying phylogenetic tree with the twenty amino acids alphabet as state space $\mathcal{A} = \{A, R, N, D, \ldots, Y, V\}$.[1]

Let then the column vector of state probabilities be

$$X(t) = (p_A(t), p_R(t), p_N(t), \ldots, p_V(t))^T, \tag{1.2}$$

where $p_x(t)$ is the probability of finding the character $x \in \mathcal{A}$ at the considered site at time t. Given the *transition rate matrix* $Q = (q_{ij})$, where q_{xy} for $x, y \in \mathcal{A}$ is the transition rate from amino acid x to amino acid y when $x \neq y$ and $q_{ii} = -\sum_{j \neq i} q_{ij}$, the evolution process is governed by the differential equation

$$X(t)' = QX(t), \tag{1.3}$$

from which immediately follows

$$X(t) = e^{tQ}X(0). \tag{1.4}$$

The elements $P_{xy}(t)$ of the *transition probability matrix* $P(t)$ provide then the probability of a character changing from x to y in time t. Probabilistic models of evolution generally further assume that the rates q_{xy} are positive for $x \neq y$ so that the Markov chain is ergodic, meaning that each state can reach any other state, i.e. $P_{xy}(t) > 0$ for all $x, y \in \mathcal{A}$. This entails that the chain admits an unique stationary distribution $\pi = (\pi_x, x \in \mathcal{A})^T$ to which $X(t)$ tends as the time t goes to infinity. This can be found solving

$$\mathbf{0} = Q\pi, \qquad \sum_{x \in \mathcal{A}} \pi_x = 1 \tag{1.5}$$

or equivalently $\pi X = X$. Generally one assumes that the chain is at equilibrium so that this is the distribution of the amino acid frequencies at each node of the tree. One often refers to amino acid frequency distributions as *amino acid profiles*, and while the simplifying assumption of all the sites sharing the same profile is typically made, allowing different profiles in different sites of the sequence is one way to increase the realism of a model of evolution. We will encounter such more complex models in chapters 2 and 3, these will furthermore allow the frequency distribution of each site to change over time.

Another common simplifying assumption in markovian models of protein evolution, already mentioned when discussing tree rooting, is that of time-reversibility:

$$\pi_x q_{xy} = \pi_y q_{yx}, \tag{1.6}$$

[1] All the following applies as well if we model instead the evolution of nucleotide characters with state space $\mathcal{A} = \{A, G, C, T\}$ or the evolution of codons with 61 states (as the three stop codons are typically excluded).

meaning that when the process is stationary the frequency of mutations of amino acid x into y is equal to that of y into x. In turn this allows us to express the transition rates as a product of stationary frequencies and relative rates, known as *exchangeabilities*:

$$q_{xy} = s_{xy}\pi_y, \tag{1.7}$$

with $s_{xy} = s_{yx}$, reducing the number of parameters needed to describe the model. These parameters, for commonly employed models such as LG, PAM, JTT and WAG, are inferred from the analysis of large protein databases.

In this framework the expected number of substitutions occurring at equilibrium in a site along a branch of length t is given by

$$t \sum \pi_i \cdot (-q_{ii}) \tag{1.8}$$

therefore in order to have one expected substitution per unit length Q is typically normalised so that

$$\sum \pi_i \cdot (-q_{ii}) = 1. \tag{1.9}$$

These models necessarily suffer from several limitations given the made simplifying assumptions. The identical distribution of the random variables which implies no rate variation between sites is limiting as for instance a functionally important subregion of a coding sequence, such as that coding for a protein binding domain, is clearly subject to a particular evolutionary pressure and thus shouldn't be modeled in the same way as less functionally relevant sites [7]. We shall see in subsection 1.2.2 that this heterogeneity of the rates of change across different sites can be modeled in the presented framework. Other limitations on the other hand are harder to deal with, this is the case for instance for the independence of sites assumption which completely disregards epistatic phenomena such as the coevolution of amino acids that share a contact point in the threedimensional structure of the folded protein [8]. Whereas such phenomena can be taken into account via more complex models, phylogenetic inference under these is hard, making their applicability limited. This is notably the case in the Maximum Likelihood (ML) framework, which will be discussed in section 1.3, given the intractability of these models' likelihood functions.

One of the main goals of my PhD project has been the development of a method that would be able to overcome such limitations, nonetheless the, most commonly employed, simplified models discussed here have been the starting point of my methodological explorations.

1.2.1 Model misspecification

Several strategies had been devised and implemented in software (e.g. [9][10]) to choose the best fitting evolution model to analyse a dataset for phylogenetic reconstruction. Different criteria, based on likelihood calculations, such as AIC, AICc and BIC, have been introduced for model selection. These, defined as scores to be minimized by the chosen model, are again founded in the parsimony principle, with models having fewer parameters being preferred unless the use of additional parameters sufficiently increases their fit to the data. Although the authors in [11] showed that the models selected via these strategies do not necessarily lead to a better topological accuracy with respect to the one that one obtains using the most parameter-rich model in the presented framework, namely GTR+I+G, the general time reversible model allowing both for invariant sites and sites with Gamma-distributed rate variation (subsection 1.2.2), thus arguing that the time-consuming step of model selection may be unnecessary for the quest of the true tree topology, they do show that simpler models found through model selection can provide better estimates of branch lengths. As we shall see in chapter 2, accurate branch lenght estimation is a particular strength of the phylogenetic reconstruction method presented in this manuscript.

Similarly, *model misspecification*, the use for inference of a model different from the true evolutionary process underlying the data being analysed, has been shown to affect to a lesser extent the inference of the tree topology while often leading to wide variations of inferred numerical parameters such as branch lengths. Nevertheless under model misspecification no theoretical guarantees can be made about the statistical consistency of likelihood (Section 1.3) or distance methods (Section 1.4) when the distance estimates are attained employing the wrong model of evolution and cases in which a misspecified model leads to serious errors in the inferred tree topology have been documented (e.g. [12][13]). We also shall analyse the impact of such misspecification through several simulations in chapter 2.

1.2.2 Per-site rate variation

Whereas in the presented framework the unconstrained tree branch lengths which do not assume a molecular clock allow us to model rate variation across different lineages, so far we have assumed the rate of variation to be constant across sites, in empirical data this again may not accurately model the evolution of residues at different positions, which may be subject to disparate evolutionary pressures and thus evolve at different rates. It is customary thus to model such variation assigning an evolutionary rate $\lambda_i \in \mathbb{R}_{\geq 0}$ to each site i, so that all the sites in a sequence will be modeled as evolving along the same tree with the branches rescaled by the factors λ_i for each different site. Equivalently, this amounts to each site i evolving along the exact same tree but according to the transition matrix $\lambda_i Q$. In current practice, it is common to use for the distribution of rates a Gamma distribution [14],[2], which is in general parameterized by two positive real parameters parameters α, β and

[2]It is worth noting that there's no particular biological reason justifying this choice which is rather made for convenience given the flexibility and mathematical tractability of such a distribution.

whose density function is given by the formula

$$f(x;\alpha,\beta) = \frac{x^{\alpha-1}e^{-\beta x}\beta^{\alpha}}{\Gamma(\alpha)},\ \alpha,\beta > 0 \tag{1.10}$$

Imposing the mean of the distribution to be equal to one (so that we can still interpret the branch lengths as the expected numbers of substitutions, now averaged across all the sites) we get $\alpha = \beta$ and therefore the distribution having the density function

$$f(x,\alpha) = \frac{\alpha^{\alpha}}{\Gamma(\alpha)}x^{\alpha-1}e^{-\alpha x},\ \alpha > 0 \tag{1.11}$$

with mean 1 and variance $1/\alpha$, the function is plotted for different α values in Figure 1.7. More precisely, for computational reasons, more often than not it is not directly the Gamma distribution but rather a discretization of it, with a finite number of classes, which is used by software implementations of maximum likelihood: When using the discrete model Gamma-k [15], the k-quantiles of the distribution are identified. Then the average values of λ in each of the resulting k intervals are computed and these values are thereafter used as the rate scaling factors λ_i for k classes, each with proportion $1/k$.

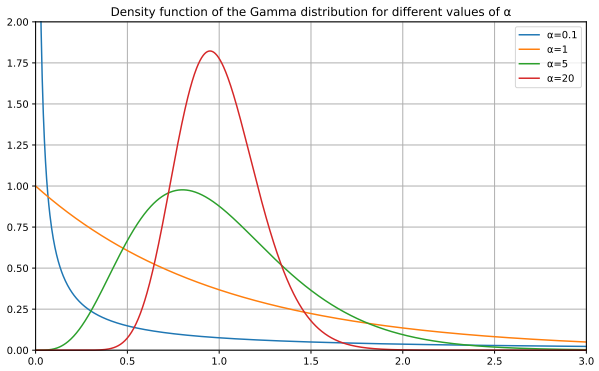


FIGURE 1.7: Probability density function of the Gamma distribution for different values of the α parameter, as the latter goes to infinity the distribution tends to a Dirac δ centered in 1 whereas as α goes to 0 more and more sites will have rates around zero while the others will be spread out across large values.

Whereas using small values of k for the discrete Gamma model one incurs in the risk for underestimating the largest rates [16], it has been shown that using more categories provides a good approximation [17]. Nevertheless, throughout this work, whenever we modeled rate variation, we rather used the full continuous Gamma distribution in order to improve the realism of the simulations on which our neural networks are trained, the same choice was then made for inference under maximum likelihood in order to test the method under the exact same model of evolution the data has been simulated with.

1.2.3 Gene trees and species trees

So far the reader may have been lead to think that the tree one reconstructs from an MSA conveys directly the evolutionary past of a set of species, it is worth then to clarify the difference between *gene trees* and *species trees*: Typically phylogenetic trees are reconstructed from a set of homologous sequences coding in different species for a specific gene, the evolutionary history of the gene however may differ from the one of the species that carry it [18].

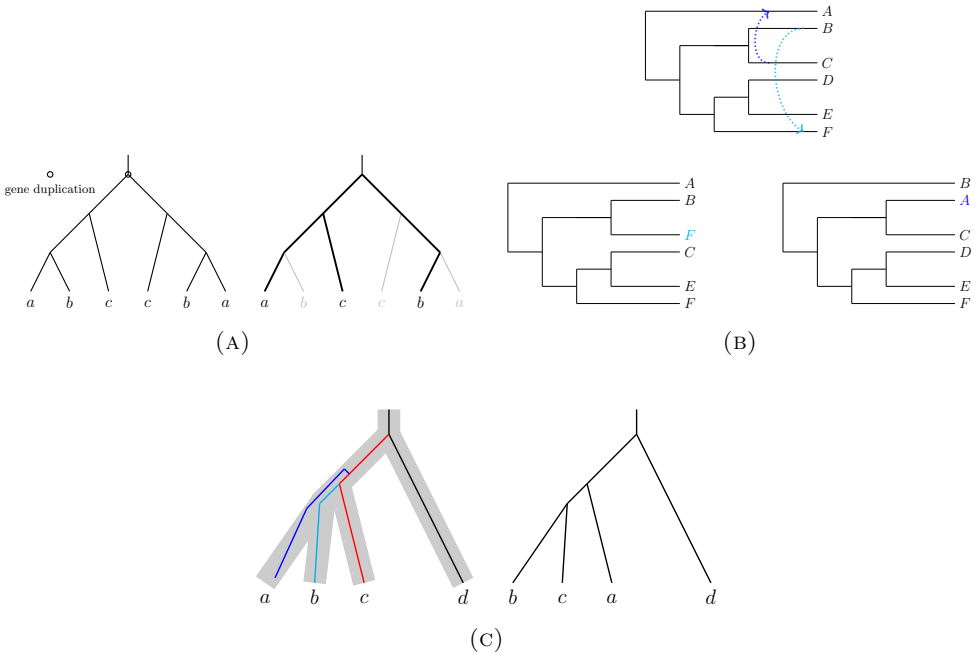


FIGURE 1.8: Gene duplication (A), horizontal gene transfer (B) and incomplete lineage sorting (C).

The main evolutionary events that can cause such a discrepancy are *gene duplications*, which may lead to the incorrect species tree via an incomplete sampling of homologous genes (Figure 1.8, A), *horizontal transfers*, transfers of genes that can take place between different species leading the trees of the involved genes to differ from the species one (Figure 1.8, B), and *incomplete lineage sorting*: In the population of a common ancestor for any given gene there are typically multiple variants (alleles) which co-exist within the population. When a speciation event occurs, the different alleles of the gene are divided among the newly diverged species. As the species continue to evolve, the gene alleles in these species may not immediately "coalesce" into a single common ancestor within each of them. This entails that lineages of these alleles may persist such that the gene tree, constructed from these, shows a different evolutionary history than the species tree (Figure 1.8, C).

These topics are beyond the scope of this work in which we focus on the reconstruction of a single tree from a set of homologous sequences, nevertheless it is worth pointing out that our phylogenetic reconstruction method presented in chapter 2, given its parallelisability, is particularly suitable for inferring multiple trees at once as it is the case when one wants to reconstruct several gene trees. Furthermore we shall see how these considerations can help us test the method on empirical data.

1.3 Likelihood methods

The state of the art approach to the problem of phylogenetic reconstruction, in terms of accuracy of the reconstructed tree topologies and branch lengths, is widely considered to be maximum likelihood, which, given the sequence data $X = \{s^1, s^2, \ldots, s^m\}$, aims at maximising the likelihood of the parameters θ (branch lengths and topology)

$$\mathcal{L}(\theta) = \Pr(X \mid \theta), \tag{1.12}$$

that is the probability of the data conditional to the parameters under a specific probabilistic model of molecular evolution. The issue with this approach is twofold, first of all a major drawback of maximum likelihood-based methods is their high computational cost which makes them unsuitable to deal with large sets of sequences, secondly, as already discussed, such probabilistic models often make several simplifying assumptions about the evolution process, assumptions which are not necessarily met in real data, leading to biased predictions.

1.3.1 Felsenstein's pruning algorithm

The assumption of evolution being i.i.d. at each site implies that the likelihood of a tree given an MSA of sequences of length L can be expressed as the product of L per-site likelihoods which at each site i can be computed independently.

$$\mathcal{L}(\tau|s^1, \ldots, s^n) = \prod_{i=1}^{L} \mathcal{L}_i(\tau|s_i^1, \ldots, s_i^n) \tag{1.13}$$

In what follows let us simply denote by $\mathcal{L}$ the per-site likelihood $\mathcal{L}_i$. We shall present Felsenstein's algorithm for the calculation of $\mathcal{L}$ along a given tree, it uses dynamic programming and employs a bottom-up approach computing recursively partial likelihoods $\mathcal{L}^p$ for each node p of the tree starting from the leaves and going up to the root: For each leaf f a partial likelihood vector L^f of size d (with $d = 20$, or $d = 4$, or $d = 61$, depending on the alphabet one is working with) is initialized with all zeros except for a one in the position corresponding to the amino-acid or nucleotide found at position i in the sequence corresponding to the leaf. This reflects the fact that at the leaf the character occupying position i is known for certainty, when there's uncertainty on the character on the other hand, due for instance to sequencing errors, this can be easily accounted for, if there's e.g. uncertainty between two possible characters we can assign the value 1 to each and 0 to the other ones.

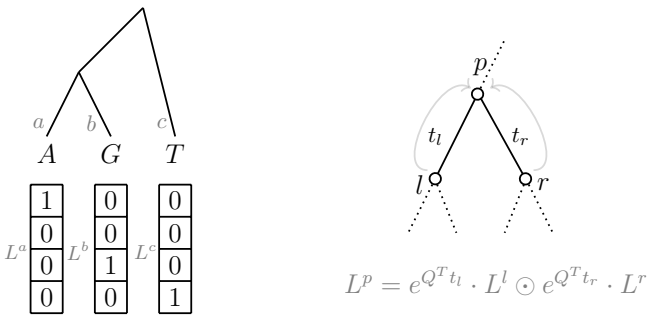


FIGURE 1.9: Left: partial likelihoods at the leaves of a tree, Right: Felsenstein's likelihood update at an internal node

Now, for each internal node p let us denote by r its right child, by l its left one and by t_r, t_l respectively the lengths of the branches which connect them to p (Figure 1.9, right), the partial likelihood vector component corresponding to the character y, $L^p(y)$ is then computed as

$$L^p(y) = \sum_{x \in \mathcal{A}} P_{yx}(t_r) L^r(x) \cdot \sum_{x \in \mathcal{A}} P_{yx}(t_l) L^l(x). \tag{1.14}$$

In matrix notation we can simply write

$$L^p = P(t_l) \cdot L^l \odot P(t_r) \cdot L^r = e^{Q^T t_l} \cdot L^l \odot e^{Q^T t_r} \cdot L^r \tag{1.15}$$

denoting with the $\odot$ operator the Hadamard product, that is the element-wise multiplication of two matrices. Finally

$$L(\tau) = \sum_{x \in \mathcal{A}} \pi_x \cdot L^{root}(x) = \pi^T \cdot L^{root}. \tag{1.16}$$

It is important to notice that the assumption of time reversibility implies that the choice of the root for the calculation of the likelihood along the tree can be made arbitrarily and does not affect the end result. Also, for numerical stability reasons, typically what's computed and maximized is not the likelihood as is but rather its logarithm, known as *log-likelihood*, which allows to replace with sums the products in the likelihood computations in order to prevent underflows. In maximum likelihood-based phylogenetic reconstruction software, once computed L in equation (1.16), the branches of the tree are optimised through several iterative steps, during which the likelihood needs to be computed again, in order to maximise its value. Then, the exploration of the tree space is continued via topological rearrangement moves in order to find a tree with a higher likelihood, furthermore, to avoid local minima, typically a pool of candidate trees is retained at each step. In practice maximum likelihood optimisation has a considerable computational footprint: Felsestein's algorithm in itself has a complexity of $\mathcal{O}(nLd^2)$, if a discrete Gamma model with k categories is employed then the per-site likelihood is given by the sum over all categories, using efficient algorithms for branch length optimisation and taking t iteration steps for the latter then results in a $\mathcal{O}(ktnLd^2)$ complexity per considered tree topology. Notably we can observe that the quadratic dependency on the dimension of the state space, the number of possible characters, is the main reason behind the i.i.d. assumption that allows the factorization (1.13). When one drops such an assumption the dimension of the state space increases exponentially in the number s of sites considered jointly, codon-based models that consider $s = 3$ nucleotides jointly are an example of this with the dimension of the state space (considering as well the typically excluded three stop codons) becoming $4^3 = 64$, it is clear then that an approach which would consider all L sites in a sequence dependent, results in a prohibitively high cost. On the other hand we shall see in chapter 2 how Phyloformer, our proposed neural network-based approach on the other hand is capable to handle a model of evolution with pairs of coevolving sites, resulting in a state space of dimension $20^2 = 400$, without an overhead computational cost for inference.

1.3.2 IQtree

Several software which reconstruct phylogenies in the maximum likelihood framework exist. In this work we mainly compared the performances of our newly developed phylogenetic reconstruction method with those of IQtree [19], which has been shown to attain the best likelihoods among diverse phylogenomic datasets [20], furthermore this choice allowed us to use its inbuilt simulator Alisim [21] so that the performances of the method could be assessed testing it under the exact same model of evolution under which the data has been simulated. IQtree first estimates 100 parsimony trees along with one reconstructed through a distance method, then optimizes branch lengths and other parameters of the model of sequence evolution, while performing local topological rearrangements (NNIs) to maximize the likelihood while retaining a candidate tree set to avoid getting stuck in local minima.

1.4 Distance methods

I shall now discuss distance-based methods for phylogenetic reconstruction which on the other hand are relatively fast compared to ML optimisation and rather work through the estimation of distances between sequences. Such distances are what will actually be predicting in the next chapter with the proposed neural network architecture, to then reconstruct phylogenies with one of the methods presented in this section.

A binary tree τ with positive branch lengths naturally defines a distance on the set of its leaves as

$$d^{\tau}(i,j) = \sum_{e \in P(i,j)} l(e), \tag{1.17}$$

$l(e)$ being the length of an edge and $P(i,j)$ the set of edges on the path from leaf i to leaf j along the tree, one oftens refers to these as *evolutionary distances*. Conversely, a map $d : X \times X \to \mathbb{R}$ satisfying $d(x,y) = d(y,x)$, $d(x,y) \geq 0$ and $d(x,x) = 0$ for all $x, y \in X$, will be called a *tree metric* if there exists a tree τ with leaves X such that equation (1.17) holds for each $i, j \in X$. Given a map which satisfies the first three conditions we can construct a symmetric distance matrix D where $D_{i,j} = d_{ij} := d(i,j)$. We say that such a matrix is *additive* if d satisfies the so-called *four point condition*:

$$d_{xy} + d_{uv} \leq \max\{d_{xu} + d_{yv}, d_{xv} + d_{yu}\}, \tag{1.18}$$

for every $(x, y, u, v) \in X^4$, or equivalently

$$d_{xy} + d_{uv} \leq d_{xu} + d_{yv} = d_{xv} + d_{yu}, \tag{1.19}$$

eventually changing the labeling of the nodes (Figure 1.10).

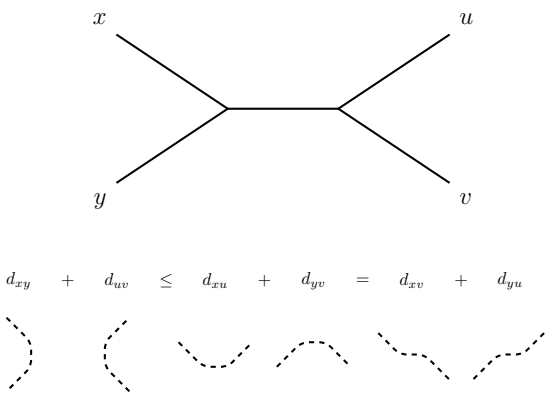


FIGURE 1.10: Four point condition, a necessary and sufficient condition for a distance to be a tree metric.

This can be shown to be equivalent to the map being a tree metric [22] (for instance it is straightforward to see that it gives the triangular inequality if one takes $u = v$), which is why the terms additive distance and tree metric are often used interchangeably in the literature. One can notice that the definition of the distance (1.17), can naturally be extended to all nodes in the tree, the term additive then comes from the fact that these distances add up along the tree: if k is a node on the path along the tree between nodes i and j then $d^\tau_{ik} + d^\tau_{kj} = d^\tau_{ij}$.

We shall see that the set of distances along a tree allows to reconstruct the underlying phylogeny, but how does one normally compute such distances? A naive approach would be to simply consider the Hamming distance between two sequences, however it is apparent that this metric is not additive. Using it one incurs in the same issues discussed in subsection 1.1.2 on parsimony, with such a distance measure failing to account for the true number of substitutions that may have occured along the branches in the tree that separate the two sequences. To account for these, and provide an unbiased estimate of the true evolutionary distances, distance estimation is again typically performed in the maximum-likelihood framework as we'll now see.

1.4.1 Model-based distances

Let s^i and s^j be two sequences of the same length L, which we suppose to have evolved according to a model of evolution $\mathcal{M}$ such as those introduced in 1.2. Relying again on the assumption of independence between sites, one can compute the probability of them being separated by time t as

$$\mathcal{L}(t|s^i, s^j) = P(t|s^i, s^j) = \prod_{k=1}^{L} P_{s^j_k s^i_k}(t). \tag{1.20}$$

The maximum likelihood estimate of the distance between the two sequences is then found by maximizing the above function:

$$d^{\mathcal{M}} = \arg\max \mathcal{L}(t|s^i, s^j)_{t\in\mathbb{R}_{\geq 0}}. \tag{1.21}$$

In general this is done numerically although we shall now see an example in which, given the simplicity of the underlying model of evolution, an analytical formula for the maximum likelihood estimate of the distance between two sequences can be derived.

ML estimate of distances under the Jukes-Cantor model

The Jukes-Cantor model [23] is the simplest continuous Markov chain model for the evolution of the 4 nucleotide characters $\mathcal{A} = \{A, C, G, T\}$, it assumes equal rates of transition between different characters so that it's stationary distribution is $\pi = \{1/4, 1/4, 1/4, 1/4\}$ and, given the normalization 1.9,

$$Q = \begin{pmatrix} -1 & 1/3 & 1/3 & 1/3 \\ 1/3 & -1 & 1/3 & 1/3 \\ 1/3 & 1/3 & -1 & 1/3 \\ 1/3 & 1/3 & 1/3 & -1 \end{pmatrix}, \tag{1.22}$$

matrix exponentiation then gives

$$P(t) = e^{Qt} = \begin{pmatrix} 1-a(t) & a(t)/3 & a(t)/3 & a(t)/3 \\ a(t)/3 & 1-a(t) & a(t)/3 & a(t)/3 \\ a(t)/3 & a(t)/3 & 1-a(t) & a(t)/3 \\ a(t)/3 & a(t)/3 & a(t)/3 & 1-a(t) \end{pmatrix}, \tag{1.23}$$

where for convenience we set $a(t) = 1 - \frac{3}{4}\left(1 - e^{-\frac{4}{3}t}\right)$ so that

$$P_{xy}(t) = \begin{cases} \frac{1}{4} + \frac{3}{4}e^{-\frac{4}{3}t} & \text{if } x = y \\ \frac{1}{4} - \frac{1}{4}e^{-\frac{4}{3}t} & \text{if } x \neq y. \end{cases} \tag{1.24}$$

Now, given two sequences s^j and s^i, let n_{xy} the number of sites at which sequence s^i presents the character x and sequence s^j the character y. We can express the likelihood of s^i and s^j being separated by time t along the tree as

$$\mathcal{L}(t|(\{n_{xy}\}_{x,y\in\mathcal{A}^2}) = \prod_{x,y\in\mathcal{A}^2} P_{xy}(t)^{n_{xy}}. \tag{1.25}$$

We can find the value $\hat{t}$ that minimizes $\mathcal{L}$ by finding the one that minimizes its logarithm $\log(\mathcal{L})$, the latter being a monotone function, we have

$$\begin{aligned} \log(\mathcal{L}(t)) &= \sum_{x,y\in\mathcal{A}^2} n_{xy} \log(P_{xy}(t)) \\ &= \log\left(\frac{a(t)}{3}\right) \cdot \sum_{\substack{x,y\in\mathcal{A}^2 \\ x\neq y}} n_{xy} + \log(1-a(t)) \cdot \sum_{x\in\mathcal{A}} n_{xx} \end{aligned} \tag{1.26}$$

so that differentiating with respect to t and equating to 0 we obtain via the chain rule

$$0 = \frac{\sum\limits_{\substack{x,y\in\mathcal{A}^2 \\ x\neq y}} n_{xy}}{a(\hat{t})} \cdot a'(\hat{t}) - \frac{\sum\limits_{x\in\mathcal{A}} n_{xx}}{1-a(\hat{t})} \cdot a'(\hat{t}), \tag{1.27}$$

we can divide by $a'(\hat{t}) = e^{-\frac{4}{3}\hat{t}}$ as the latter is always positive, and solving for $a(\hat{t})$ we then get

$$a(\hat{t}) = \frac{\sum\limits_{\substack{x,y\in\mathcal{A}^2 \\ x\neq y}} n_{xy}}{\sum\limits_{\substack{x,y\in\mathcal{A}^2 \\ x\neq y}} n_{xy} + \sum\limits_{x\in\mathcal{A}} n_{xx}} = \frac{\sum\limits_{\substack{x,y\in\mathcal{A}^2 \\ x\neq y}} n_{xy}}{\sum\limits_{x,y\in\mathcal{A}^2} n_{xy}}, \tag{1.28}$$

we can observe now that the right hand side of this equation is simply the ratio between the number of sites for which the two sequences differ and the overall number of sites, i.e. their sequence length, this is thus the normalized Hamming distance between the two sequences, also known as *p-distance* in the context of phylogenetics, denoting by d_p the latter we finally can solve for $\hat{t}$ obtaining

$$\hat{t} = -\frac{3}{4}\log\left(1 - \frac{4}{3}d_p\right). \tag{1.29}$$

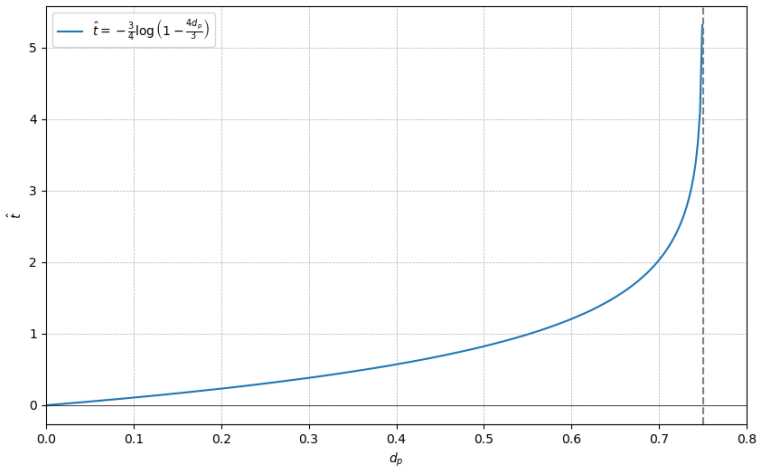


FIGURE 1.11: p-distance against the Jukes-Cantor model-based correction.

The formula allows for an estimate of the evolutionary distance between the two sequences which takes into account multiple substitutions and back-mutations. Nevertheless, as one can see in Figure 1.11, equation (1.29) breaks down for values of d_p bigger then 0.75 which limits its application in the case of highly divergent sequences, beyond this point the sequences are said to be *saturated* and it is not possible anymore to infer the expected distance from the observed one via such an equation.

1.4.2 Neighbor joining

One of the most popular algorithms to reconstruct a phylogeny from a set of distance estimates[3] $\{d_{ij}\}$ is the neighbor joining (NJ) algorithm [24][25] which, once defined

$$Q_d(i,j) = d_{ij} - \frac{1}{n-2}\left(\sum_{k\neq i} d_{ik} + \sum_{k\neq j} d_{jk}\right), \tag{1.30}$$

proceeds as follows:

Algorithm 1: NEIGHBOR JOINING

Input : $X = \{s^1, s^2, \ldots, s^n\}$, $D = \{d_{ij},\ i,j \in X\}$;
Initialisation :
$V(\tau) \leftarrow \{\}$; `#Nodes of the reconstructed tree`
$E(\tau) \leftarrow \{\}$; `#Edges of the reconstructed tree`
while $|X| > 3$:
    $a, b = \arg\min_{i,j\in X} Q_d(i,j)$;
    $D \leftarrow \{\frac{1}{2}(d_{ia} + d_{ib} - d_{ab})\}_{i\in X} \cup (D \setminus \{d_{ka}, d_{kb}\}_{k\in X})$;
    $X \leftarrow \{j_{a,b}\} \cup (X \setminus \{a,b\})$;
    $l_a \leftarrow \frac{1}{2}d_{ab} + \frac{1}{n-2}\left(\sum_{k\in X} d_{ak} - d_{bk}\right)$;
    $l_b \leftarrow d_{ab} - l_a$;
    $V(\tau) \leftarrow V(\tau) \cup \{a, b, j_{a,b}\}$;
    $E(\tau) \leftarrow E(\tau) \cup \{\{(a, j_{a,b}\}, l_a), (\{b, j_{a,b}\}, l_b)\}$;
return $(V(\tau), E(\tau))$;

Namely at each step it replaces the two elements $a, b \in X$ which minimize Q_d with a single one $j_{a,b}$ adding to the constructed tree the leaves a, b, the node $j_{a,b}$ and the edges from the leaves to said node, while replacing the distances between either a or b and another element in X with their sum minus the distance between a and b, divided by two. It is interesting to mention that Saitou and Nei [24] showed that the lengths of the branches l_a and l_b joining the neighbors a and b computed at each step of the algorithm, correspond to those one would get via an ordinary least squares estimate, an approach that will discussed in the following subsection.

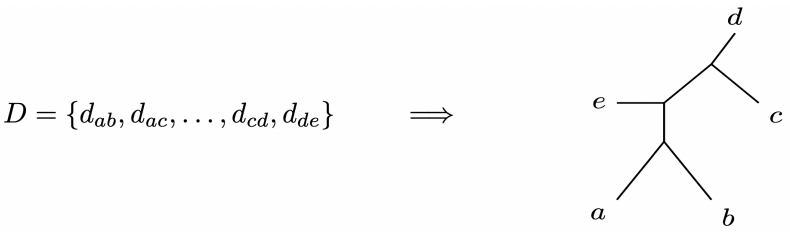


FIGURE 1.12: A clustering algorithm as NJ directly reconstructs a tree from a distance matrix without explicitly optimizing any criterion.

[3]Here and elsewhere in this work, with a slight abuse of language, I will refer to these distance estimates as distances even if they not necessarily satisfy the triangle inequality and thus are not really distances in the mathematical sense.

The definition (1.30) of Q_d, which takes into account both how close two elements are to each other and how far they are from the remaining ones, is linear in the distances (which also makes the algorithm scaling invariant if one is concerned only in retrieving the tree topology), permutation invariant and is guaranteed correctly finds the tree corresponding to a tree metric, furthermore it can be showed that it is the unique criterion that satisfies these properties [26].

While the algorithm, provided a tree metric, infallibly leads to the unique associated tree, the distances estimates we compute from real data most often are not additive failing to meet condition (1.18). For this another result comes in help, ensuring the consistency of the algorithm even when the input distance matrix is "nearly additive" [27]:

Theorem 1. Neighbor joining has l_∞ radius $\frac{1}{2}$.

This means that the correct tree topology is guaranteed to be reconstructed by the algorithm as long as the estimates of the distances given in input are at most half the shortest edge length in the tree away from their true value. This is enough to prove the *statistical consistency* of the algorithm, namely that if the input distances are unbiased estimates of the true evolutionary distances then in the limit of infinite sites the true underlying tree is guaranteed to be found via NJ. Moreover, it has been observed that often neighbor joining is successful even when such a sufficient condition is not satisfied [28]. As far as computational complexity goes, we can notice that the implementation (1) has an $O(n^3)$ time complexity, this can be even reduced to an average performance of $\Theta(n^2)$ implementing some heuristics [29] while alternatives such as FastNJ [30] have been proposed with a running time of $O(n^2)$ leading to some loss of accuracy but remaining statistically consistent. A popular variant of Neighbor Joining is BIONJ [31] which models the variances and covariances of the estimated evolutionary distances to improve the selection of the pair to join, namely the one which minimizes the variance of the updated distance matrix. It is worth mentioning that if the input distance matrix is additive then the corresponding tree can be recovered even with a more naive algorithm which has $O(n^2)$ complexity [32], the latter however lacks the statistical guarantees just discussed. One drawback of the neighbor joining algorithm is that, as the least squares approaches that will shall now encounter, it can lead to unmeaningful negative branch lengths in the reconstructed tree.

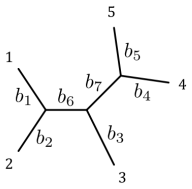
1.4.3 Least squares

Given a predicted distance matrix let $\hat{d} \in \mathbb{R}^{\binom{n}{2}}$ be the vector of predicted distances ordered lexicographically[4], and let d^τ the corresponding vector of distances between leaves along the tree τ defined as in equation (1.17), Cavalli-Sforza and Edwards ([33]) consider the squared euclidean distance between the predicted distance and the tree metric vectors

$$E = ||d^\tau - \hat{d}||^2 = \sum_{i,j \in \{1,\dots,n\}, i \neq j} (d^\tau_{s^i s^j} - \hat{d}_{s^i s^j})^2, \tag{1.31}$$

[4]Namely $\hat{d} = (\hat{d}_{s^1 s^2}, \hat{d}_{s^1 s^3} \dots, \hat{d}_{s^2 s^3}, \dots, \hat{d}_{s^{n-1} s^n})^T$

and formulate the tree reconstruction problem as the quest for the tree τ which minimizes the value of E.



$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} b_1 + b_2 \\ b_1 + b_3 + b_6 \\ b_1 + b_4 + b_6 + b_7 \\ b_1 + b_5 + b_6 + b_7 \\ b_2 + b_3 + b_6 \\ b_2 + b_4 + b_6 + b_7 \\ b_2 + b_5 + b_6 + b_7 \\ b_3 + b_4 + b_7 \\ b_3 + b_5 + b_7 \\ b_4 + b_5 \end{bmatrix} = \begin{bmatrix} d_{12}^T \\ d_{13}^T \\ d_{14}^T \\ d_{15}^T \\ d_{23}^T \\ d_{24}^T \\ d_{25}^T \\ d_{34}^T \\ d_{35}^T \\ d_{45}^T \end{bmatrix}$$

FIGURE 1.13: Factorisation of the distances vector of a phylogenetic tree as the product of the tree's path incidence matrix and the vector of the tree's branch lengths [34].

For a given topology the lengths to be assigned to its branches in order to minimize E can easily be found, indeed the problem can be simply formulated as an ordinary least square regression (OLS) one: let $e_1, \ldots, e_{2n-3}$ be an arbitrary ordering of the branches in τ and $v = (l(e_i))^T_{i\in\{1,\ldots,2n-3\}}$ be the vector of their lengths, we can then define the *path edge incidence matrix* $X \in \mathbb{R}^{\binom{n}{2}\times(2n-3)}$ so that $X_{ij} = 1$ if the branch e_j lies along the path in the tree connecting the pair of leaves corresponding to the i-th component of d^τ (Figure 1.13). This allows us to write in matrix form

$$d^\tau = X \cdot v \tag{1.32}$$

and thus to formulate the problem as the minimization of

$$||Xv - \hat{d}||^2 \tag{1.33}$$

which amounts to solving for v

$$\hat{d} = Xv, \tag{1.34}$$

multiplying on the each side by X^T we get $X^T\hat{d} = (X^TX)v$ and thus

$$\hat{v} = (X^TX)^{-1}X^T\hat{d}. \tag{1.35}$$

More generally different weights can be assigned to each summation term in the equation:

$$\sum_{i,j\in\{1,\ldots,n\},i\neq j} w_{ij}(d^\tau_{s^i s^j} - \hat{d}_{s^i s^j})^2, \tag{1.36}$$

the corresponding weighted least square (WLS) problem then leads to

$$v = (X^TWX)^{-1}X^TW\hat{d} \tag{1.37}$$

where $W \in$ is the diagonal matrix $\in \mathbb{R}^{\binom{n}{2}\times\binom{n}{2}}$ in which the weights w_{ij} along the diagonal are again ordered lexicographically as in d^τ and $\hat{d}$. Different weighting schemes have been proposed, for instance Fitch and Margoliash [35] used $w_{ij} = 1/(\hat{d}_{s^i,s^j})^2$ whereas $w_{ij} = 1/\hat{d}_{s^i,s^j}$ was proposed by Beyer et al. [36]. The cost of inverting the X^TWX matrix is in general $\mathcal{O}(n^3)$ although $\mathcal{O}(n^2)$ methods which exploit the structure of the tree to save computations had

been proposed [37][38]. Still, the problem of finding the tree which minimizes remains NP-complete [3] and an heuristic exploration of the tree space is needed in the quest of a tree which is optimal according to the criterion. It is to be noted that v as obtained through OLS regression in equation (1.35) is the best linear unbiased estimator according the Gauss-Markov theorem, assuming equal variances in the distance estimates, the OLS estimation thus implicitly makes the latter assumption whereas the WLS approach allows to model different variances as the reciprocals of the weights w_{ij}, still making the assumption that the covariances of the distance estimates are equal to zero. This is still a simplifying assumption as estimates of distances whose computation along the tree involves common branches will necessarily be correlated. In principle this could be handled with a generalised least squares regression (GLS) replacing W in equation (1.37) with the inverse of the covariance matrix V of the distance estimates, these covariances however are hard to obtain and the additional computational complexity of inverting the $V \in \mathbb{R}^{\binom{n}{2}\times\binom{n}{2}}$ matrix limits the applicability of the approach. It is nevertheless important to point out that no matter the weighting scheme the criterion is statistically consistent, using unbiased distance estimates the true tree is guaranteed to be recovered in the limit of infinite data by finding the one that minimizes E even in the case of simple OLS. A drawback that least squares approaches share with neighbor joining is that again the estimated branch lengths are not guaranteed to be positive leading at times to negative branch lengths which are void of biological interpretation.

1.4.4 Minimum evolution

A distance method related to the least squares approach is the *Minimum Evolution* (ME) method [39][40] that searches among tree topologies, computing again their branch lengths by least squares fitting, but aiming to minimize a different overall criterion, namely the sum of these branches, the tree length, instead of the squared distance between the predicted distance and the tree metric vector. The criterion is justified by the proof in [41] that when the branch lengths are estimated via OLS the expectation, under the underlying evolution model, of the tree length for the true tree is the smallest among all possible topologies, provided an unbiased estimate of the evolutionary distances. While both the OLS and ME criteria are thus statistical consistent, the authors in [40] have shown that ME is more efficient at recovering the true tree with respect to OLS in the case of large evolutionary distances and not very long sequences.

Balanced minimum evolution

Pauplin [42] proposed an alternative approach to OLS branch estimation in which each clade is given the same weight and has shown that by doing so we can express the length of a tree τ, whose branches have been fitted from a predicted distance vector $\hat{d}$ in this way, simply as

$$l(\tau, \hat{d}) = \sum_{i,j} \frac{\hat{d}_{ij}}{2^{\delta^{\tau}_{ij}}}, \tag{1.38}$$

where δ_{ij}^{τ} denotes the topological distance along the tree between leaves i and j, that is the number of edges that separate them. The *Balanced Minimum Evolution* (BME) problem can then be formulated as the search of the tree topology τ such that the function in equation (1.38) is minimized. It is interesting to remark that in order to find such a topology one does not even need to estimate its branch lengths as only the topological distances on τ and the predicted distances $\hat{d}$ are involved in the formula. Given that l as defined in equation (1.38) is a continuous function of $\hat{d}$, to prove the statistical consistency of the BME criterion it is sufficient to show that given the true tree τ,

$$l(\tau', d^{\tau}) > l(\tau, d^{\tau}) \tag{1.39}$$

whenever τ' is different from τ. This was shown by Desper and Gascuel [43] who further proved that the tree length estimate one obtains with the BME approach is the same that one gets via a WLS in which the variance of each distance estimate $\hat{d}_{ij}$ is proportional to the exponential of the corresponding topological distance $2^{\delta_{ij}^{\tau}}$. Furthermore they show that, if a tree τ is a local minimum for an NNI topology search according to the BME criterion, then all the branch length estimates of τ are positive provided that the distance estimates $\hat{d}_{ij}$ satisfy the triangular inequality while the positivity of the internal branches is guaranteed even if the inequality is violated. Finally it is important to mention that it has been proven [44] that the Neighbor-joining algorithm selects at each step the pair to join which most decreases the whole tree length as computed via Pauplin's formula thus revealing that the algorithm is a greedy heuristics for the BME criterion.

1.4.5 FastTree

Another method we shall encounter in the next chapter as a benchmark is FastTree [45]. It reconstructs a starting tree using an algorithm inspired from Neighbor-Joining [24] using profiles (the frequencies of each possible character (including gaps) at each site) of extant and ancestral sequences instead of a distance matrix, only computing the distance between two profiles if they are promising candidates to be joined, and taking advantage of heuristics such as those employed in FastNJ, this leads to a time complexity of $O\left(L\sqrt{n}\log n\right)$ for this initial tree reconstruction. The tree is subsequently refined with NNI and SPR topological rearrangements to optimize the balanced minimum evolution criterion as defined in equation (1.38). Finally the tree is further improved using an approximation of maximum likelihood with NNI topological moves. FastTree is not therefore strictly a distance-based method, and as we will discuss at the end of the next chapter, its accuracy can be mostly ascribed to these likelihood-maximing moves.

1.4.6 FastME

FastME [46] computes a distance matrix using maximum likelihood, then reconstructs a tree topology using BioNJ [31] and further refines it via NNI and SPR topological rearrangements which seek to optimize the BME score as defined in equation (1.38). In chapter 2 we shall use the same algorithm to assess the performances in terms of phylogenetic reconstruction of Phyloformer, using the distances predicted by our model instead of the ML estimates.

1.4.7 Advantages and limitations of distance-based methods

Distance methods such as neighbor joining can provide a fast alternative to resource-intensive maximum likelihood optimization to reconstruct phylogenetic trees while remaining statistically consistent. This makes them suitable for the analysis of large datasets[5] for which a maximum likelihood approach would be unfeasible, or in cases where a more complex model of evolution is considered for which the likelihood function is intractable while unbiased estimates of evolutionary distances can still be obtained. A significant advantage of distance-based methods is their ability to work whenever a dissimilarity measure between different taxa can be computed, this allows them to be applicable as well when working with unaligned sequences [47][48][49], sequence alignment being a costly and error-prone step, and with different inputs other than molecular sequence data (such as immunological data [50], gene frequencies [33] or sequence data enriched with protein structure information [51]), while the combined analysis of multiple distance matrices can replace traditional supertree approaches (subsection 1.5.3) [52][53]. Furthermore several techniques have been developed for effectively imputing missing data [54],[55],[56] allowing the methods to work even with incomplete distance matrices.

On the other hand numerous studies have shown that distance methods are commonly less accurate than maximum likelihood or Bayesian ones, this can be attributed to the fact that, as distance estimation is typically performed considering each pair of sequences independently, such methods fail to exploit the full information contained in the alignment data. For instance when evolutionary rates vary between sites (subsection 1.2.2) likelihood methods are able to propagate information from changes in one part of the tree to inform the correction in others whereas a classical distance-based method is inherently incapable of doing so [1], similarly the latter is not able to benefit from the information about ancestral sequences as FastTree, parsimony and likelihood methods do.

These considerations motivate the main contribution of this work, namely the development of a distance estimation method which through a joint prediction of all evolutionary distances exploits the information provided by the entirety of the input MSA which, as we shall see, indeed leads to significantly better distance estimates.

1.5 Assessing the performances of phylogenetic reconstruction methods

1.5.1 Comparing trees

Since phylogenetic trees cannot be directly observed but only inferred, primarily via molecular data, the reliability of such inferred trees depends heavily on the methods and models used for the reconstruction. As we have seen such methods typically rely on several assumptions about evolution which may fail to reflect the true complexity of the underlying biological process. Given such a lack of ground truth, the accuracy of all tree reconstruction methods has been primarily tested via simulations as these allow to directly compare the reconstructed trees with the one along which the data has been simulated.

[5] Although the quadratic memory requirement to store the distance matrix can become a bottleneck when datasets with several thousands of sequences need to be analysed.

One of the most commonly employed metric to compare two trees is the Robinson-Foulds (RF) distance [57]: We can see each branch on a phylogenetic tree as defining a bipartition (or split) of the set of its leaves, consisting in the two subsets of leaves found in the two components obtained removing the branch from the tree, paired with a weight (*i.e.*, the branch length). Conversely these $2n-3$ bipartitions fully characterize the tree. Let τ and τ' be two trees with n leaves, A and B be the sets of the corresponding bipartitions induced on the set of leaves $\{1,\ldots,n\}$, and $l_\tau(e)$ the weight of a bipartition e in the tree τ, namely the length of the branch inducing the bipartition if the latter is present in the tree τ and 0 otherwise. Then, the Robinson-Foulds distance between τ and τ' is defined as the number of bipartitions not shared by the two trees:

$$RF(\tau,\tau') = \left(|A\cup B| - |A\cap B|\right), \tag{1.40}$$

and its values range between 0 (when the two trees share the same topology) and $4n-6$ when they have no leaf bipartition in common. We can observe that the two trees are NNI neighbors if and only if $RF(\tau,\tau')=2$. In order to compare the values of the metric when testing phylogenetic tree reconstruction methods on trees of different sizes throughout this work we will rather use the normalized version of the metric

$$RF_{norm}(\tau,\tau') = \frac{1}{4n-6}\left(|A\cup B| - |A\cap B|\right), \tag{1.41}$$

and with a slight abuse of notation denote simply as RF the latter.

While the RF distance only considers the tree topologies we can generalise its formula to take the branch lengths into account, weighting each bipartition by the square of its corresponding weight, this gives us the Kuhner-Felsenstein distance [58]

$$KF(\tau,\tau') = \sum_{e\in A\cap B}\left(l_\tau(e)-l_{\tau'}(e)\right)^2 + \sum_{e\in A\setminus B} l_\tau(e)^2 + \sum_{e\in B\setminus A} l_{\tau'}(e)^2, \tag{1.42}$$

which amounts to the square of the Euclidean distance between the weight vectors l_τ and $l_{\tau'}$ and of which the unnormalized RF distance is a special case when all branches on the two trees have length 1. The RF distance has been criticized, among other reasons, for a lack of robustness as simply moving the placement of one leaf on the tree (an SPR move on a terminal edge) can change its value drastically (Figure 1.14), nevertheless, although several improvements on it had been proposed, it remains the most widely used measure of topological dissimilarity in phylogenetics given its simplicity and ease of calculation. For this reason it is the one we adopted as well throughout this work whenever a topological comparison of trees was warranted. Nevertheless, given the aforementioned issues, as a sanity check we also kept an eye on another topological metric, the quartet distance [59], defined as the fraction of 4-leaves subtrees in τ and τ' which do not share the same topology, as this metric is not affected as much by moving a single leaf from one part of the tree to another.

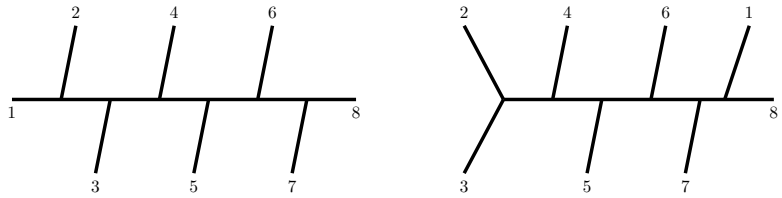


FIGURE 1.14: Two trees which do not share any bipartition and thus have maximal Robinson-Foulds distance although they differ only by the placement of species 1.

Finally we can compare two phylogenetic trees by computing their likelihood under a given model via Felsestein's algorithm (subsection 1.3.1) which allows us to assess the performances of different reconstruction methods when the model used for inference coincides with the one underlying the data, in the case of model misspecification (subsection 1.2.1) such an approach is less justifiable. Moreover, even when the model is correct, working only with a finite number of sites will often lead to several trees having a higher likelihoods with respect to that of the true one. Nevertheless, this allows to compare the performances of the methods on empirical data (as it is done e.g. in [20]) for which a reliable ground truth tree for the comparison is unavailable, and such an approach can still provide a good assessment of different methods provided that the employed model of evolution provides a sufficiently good fit of the data being analysed.

1.5.2 Bootstrap

To assess the uncertainty in the phylogeny reconstructed by any given method, a general and most commonly used technique is that of the bootstrap [60], firstly applied in phylogenetics by Felsenstein [61]. Once reconstructed a tree from an MSA of length L it consists of resampling, *with replacement*, L sites among the original ones in order to produce an artificial alignment, this will almost always miss some of the original sites while containing others multiple times. The process is repeated k times (typically a few hundreds) and a tree is reconstructed from each artificial alignment using the same method as for the original one. The process then generates a set of trees that can be summarized for instance by annotating the branches of the originally reconstructed tree with *support values*, namely the proportion of trees among the bootstrap samples in which the given branch (again seen as the corresponding bipartition induced on the leaves) is found. Naturally the whole bootstrap procedure is computationally expensive and exacerbates the computational footprint of the underlying tree reconstruction method. Although approximate techniques to obtain support values on phylogenies had been devised (e.g. rapid bootstrap implemented in the ML software RAxML [62] or ultrafast bootstrap implemented in IQ-Tree [63]), the computational cost of bootstrap generally limits its applicability for large datasets. In this work we did not resort to bootstrapping for uncertainty assessment but as we shall see in the next chapter our newly introduced method for phylogenetic reconstruction is particularly suitable for the inference of multiple trees at once, even more so when the input MSAs share the same length as it is the case when one wishes to compute bootstrap values. Possible extensions of our method to compute uncertainty values associated to its prediction will also be discussed.

1.5.3 Supertree methods and quartet puzzling

Sometimes one may wish to combine several, potentially overlapping, phylogenetic trees inferred for different subsets of taxa into a single comprehensive one, describing the evolutionary history of all considered taxonomic units. This may be the case if the complete tree cannot be obtained directly due to limitations of the tree reconstruction method, computational bottlenecks, or different and incompatible sources of data from which the partial trees are inferred. To address these challenges, *supertree* methods have been developed, with the most popular being Matrix Representation with Parsimony (MRP) [64][65] which transforms the input trees into a matrix representing the splits of all the considered taxa induced by the trees, with a supertree being then inferred using parsimony. Whereas originally the method ignores branch length information in the input trees, it has been extended to deal with metric trees in [66] while variants which use heuristics for maximum likelihood optimisation instead of maximum parsimony to construct the supertree from the matrix also have been introduced [67]. Another class of approaches is that of *quartet methods* which exploit the fact that the topologies induced on all subsets of four leaves in a phylogenetic tree are sufficient to fully recover the latter. Among these a popular one is quartet puzzling [68] which computes the maximum likelihood solution[6] for all quartets of taxa (which entails considering only the three possible tree topologies for four leaves) with the full tree for all taxa being then constructed via a majority-rule consensus. Although still slow for many taxa, the method is much faster than computing directly the exact maximum likelihood tree. Despite the popularity in the late 1990s and early 2000s of these methods for combining quartet trees, their accuracy was found to be lower than that of ML or distance methods [69] slowing down research in this direction.

1.6 Machine and deep learning

Over the last few years machine learning and most notably the subfield of deep learning have seen unprecedented success in a wide range of scientific fields. This success can be ascribed to different factors such as better algorithms, the constant increase of labelled data available to train large deep neural networks, open source software and increasing computing power with the exploitation of graphical processing units (GPUs) as well as the development of dedicated hardware for the training of neural networks such as Tensor Processing Units (TPUs). Such advances have enabled breakthroughs in disparate areas like natural language processing, computer vision, physics and biology, with neural network-based approaches often attaining state of the art performances across numerous tasks. While a comprehensive review of the domain is far beyond the focus of the present work, I will provide an introduction to the essential concepts of machine learning and building blocks of neural networks which shall be brief, yet hopefully complete enough for the reader to be able to follow the ensuing chapters.

[6] This is however applicable as well to any tree reconstruction method for four leaves.

1.6.1 Supervised learning

Most machine learning algorithms can be classified in two broad categories, that of *supervised* and that of *unsupervised* learning. Whereas the aim of supervised learning is to train a model (a parameterized function) on labeled data, learning to map input features to corresponding outputs for eventually making predictions on new, previously unseen datapoints, unsupervised learning aims at recovering patterns and relationships between the features of data when no labelled outputs are available for training. In this work we shall solely focus on the former paradigm. In this context we are typically working with a collection of samples, data points x_i associated to corresponding labels y_i, which shall be referred to as the *training dataset*, and our goal is to learn the parameters θ of a function f whose output $f(x_i, \theta)$ aims at predicting the corresponding label y_i (or customarily in classification problems, the distribution $p(y_i \mid x_i)$ of the latter given x_i). Learning the predictor f, or training the model, is done via the empirical minimization principle, by minimizing the training error or empirical risk

$$R_n(f) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)), \tag{1.43}$$

where L is a real-valued loss function measuring how far the predictions of the model are from the desired output. Ultimately the goal of such a learning task is for the predictor to be able to generalize to unseen data, that is, we want the generalization error

$$R(f) = E_{(x,y)\sim P} L(y, f(x)), \tag{1.44}$$

to be minimized, where P is the distribution underlying the data. The training error then is an approximation of the generalisation one obtained using the training samples. This paradigm distinguishes machine learning from classical optimization such as that of the likelihood in phylogenetics via hill climbing algorithms, the optimization is not performed on each new data point but upfront, on samples supposedly issued from the same distribution as the ones on which subsequently we wish to run the model for inference. In machine learning one customarily works with functions which are differentiable with respect to the parameters θ, as well as differentiable losses, so that one can compute the gradient of 1.43:

$$\nabla L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla L(y_i, f(x_i, \theta)), \tag{1.45}$$

and update the model's parameters via gradient descent:

$$\theta_{t+1} \leftarrow \theta_t - \lambda \nabla L(\theta), \tag{1.46}$$

where λ is known as the *learning rate* and the update as a training step. In practice, as the complexity of computing the full gradient (1.45) grows linearly with the typically very high number of samples n, to avoid such a computational bottleneck one rather computes stochastic gradients

$$\frac{1}{|B|} \sum_{i \in B} \nabla L(y_i, f(x_i; \theta)) \tag{1.47}$$

where $B \subset \{1, \dots, n\}$ is sampled uniformly at random at each step, the samples $\{(x_i, y_i)\}_{i \in B}$ are then referred to as a *batch* and $|B|$ as the *batch size*. The stochastic gradient 1.47 provides an unbiased approximation of the full gradient and the optimization of f using these gradients in the update step (1.46) is known as *Stochastic Gradient Descent* (SGD), in practice one chooses in advance a batch size $b \ll n$ and then partitions the training dataset in disjoint batches of size b, a full pass of training steps over these is then referred to as an *epoch*. Several *optimizers*, algorithms to update the model parameters during training, are commonly employed, notably variations of SGD that employ per-parameter adaptive learning rates and use momentum to accelerate convergence by adding a fraction of the previous update to the current one, these include the popular Adam [70], RMSProp [71] and AdaGrad [72]. Despite the techniques implemented by these methods, it is often still beneficial to decrease the global learning rate λ over time, numerous *learning rate schedules* have been proposed such as reducing λ linearly or exponentially in the number of epochs or training steps, with the training of some models, such as transformers, often also including a warm-up phase during which the learning rate is increased up to a given maximum value before being decreased.

1.6.2 Validation, overfitting, hyperparameters and regularisation

Once trained a function f we can get an estimate of its generalization error by computing the error on a *test dataset*, a previously held out dataset of samples which we assume following the same distribution as those used during training. Under this hypothesis the expected test error will be greater than or equal to the expected training error. Ideally we want to reduce the gap between the two as much possible, when such a gap is large we say that the model is *overfitting*, having learned patterns which are only specific to the training data and which fail to generalise to previously unseen data points. To avoid this and reduce the generalisation error, several *regularisation* techniques can be employed. The form of regularisation to employ, the type of function with its set of parameters to be optimized, the learning rate, its schedule and the optimizer to be used are all *hyperparameters*, settings that we can modify to control the behavior of a learning algorithm but are not optimized by the algorithm itself. The choice of which hyperparameters to use is typically based on the performances of the learned predictor on a separate held out *validation dataset*, with the validation error, which we can keep track of during training, being a proxy of the test error: One can choose the best hyperparameters as the ones that minimize the former to then assess the performances of the predictor on the test dataset only when the learning process has completed. This assures that the test error is an unbiased estimate of the generalisation one as the test data plays no role in the optimization of the parameters of the learned function nor in the choice the hyperparameters of the learning algorithm. Some regularisation techniques are explicit as adding a penalisation term to the loss, this is the case for instance for L^2 regularisation which adds an $\alpha \|\vartheta\|_2^2$ term to the loss, where α is a positive real number and ϑ is a subset of the function's parameters θ. The rationale is that such a modification of the loss shifts the optimum of the training error towards simpler functions reducing the risk of overfitting the data. Other forms of regularization are rather implicit, this is for instance the case of *early stopping*, which consists

in stopping the optimization whenever the validation error does not decrease for a certain number of epochs and returning the parameters for which the learned function has the lowest validation error. Looking at the number of steps or epochs the predictor is trained for as an additional hyperparameter, early stopping can then be seen as a efficient selection criterion for the latter.

1.6.3 Fully connected neural networks

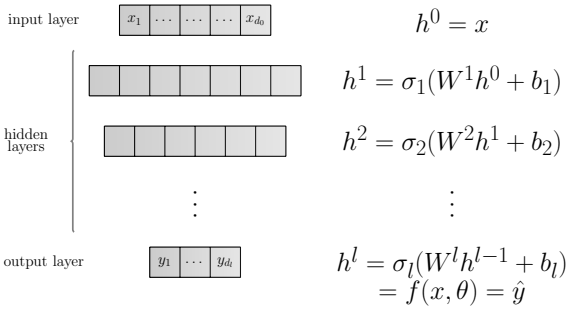


FIGURE 1.15: Fully connected neural network: The input is processed via successive affine functions intertwined with non-linear activations and every i-th component of layer k can affect the j-th component in the successive layer through the weight W^l_{ij}.

In deep learning, a subfield of machine learning, the predictor f one wishes to learn is implemented through a neural network, the composition of several functions, referred to as the *layers* of the network[7], with their number being referred to as its *depth* and the intermediate layers often being called *hidden layers*. One of the simplest neural network *architectures* is that of *fully connected neural networks* (FCNN) which process the input via a series of linear (or rather affine) functions intertwined with non-linear *activation functions* (figure 1.16): Whereas the composition of affine functions remains an affine function, the introduction of such non-linearities allow modeling increasingly complex and expressive functions. In such architectures the input $x = h^0$ is transformed through successive layers $h^1, \dots, h^l = \hat{y}$ with the function defining each layer $h^i = \sigma_i(W^i h^{i-1} + b_i)$ consisting in the multiplication by a *weight* matrix $\in \mathbb{R}^{d_i \times d_{i-1}}$, the addition of a *bias* term $\in \mathbb{R}^{d_i}$ and the element-wise application of an activation function $\sigma_i : \mathbb{R} \to \mathbb{R}$. In practice it is customary to use the same activation function σ across different layers, with the potential exception of the last one σ_l depending on the task at hand:

- For regression tasks typically no activation function is applied to the output layer given that for regression problems the network needs to predict a continuous range of values and applying an activation function could constrain the output, in this case the network's output is then an affine transformation of the features of the next to last layer. Exceptions are made if the range of the output values is meant to be bounded in which case an eventually scaled Sigmoid function can be used, or if the predicted outputs are meant to be positive, in which case one can use e.g. a Softplus function (figure 1.16).
- For binary classification tasks the Sigmoid activation is generally used to ensure that the output is a between 0 and 1, so that it can be interpreted as the probability that the input belongs to the positive class. On the other hand for multi-class classification tasks (with m mutually exclusive

[7]Note that, depending on the context, the term layer is used to denote either one of these intermediate functions or the representation of the input that the function gives rise to.

classes) the softmax activation

$$\text{softmax}(z) = \left[\frac{e^{z_1}}{\sum_{k=1}^{m} e^{z_k}}, \cdots, \frac{e^{z_K}}{\sum_{k=1}^{m} e^{z_k}}\right] \tag{1.48}$$

is typically applied to the output layer, converting the outputs into a probability distribution over the different classes.

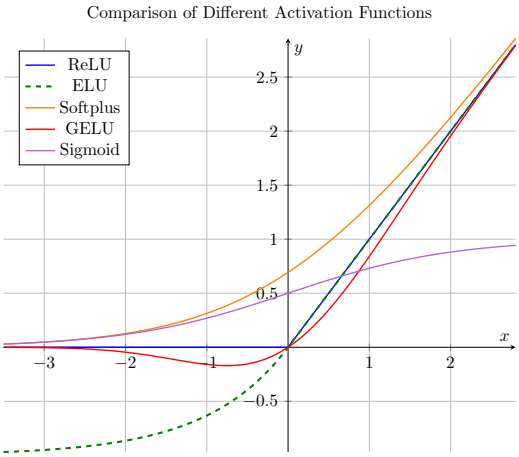


FIGURE 1.16: Some activation functions typically employed in deep neural networks:

- $\text{ReLU}(x) = \max(0, x)$ (Rectified Linear Unit)
- $ELU(x) = \max(0, \alpha(e^x - 1)), \quad \alpha > 0$ (Exponential Linear unit, plotted with $\alpha = 1$)
- $\text{Softplus}(x) = \log(1 + e^x)$
- $\text{GELU}(x) = x\Phi(x)$ (Gaussian Exponential Linear Unit, typically employed in transformer architectures), where $\Phi(x)$ is the standard Gaussian cumulative distribution function.
- Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$

The parameters θ of a network such as that described above are the collection of all weights and biases. Whereas during optimization the output of the last layer is driven to approximate the desired label, no explicit constraints are imposed on the intermediate ones. As a result we can see a neural network as a feature extractor, with the model learning the most suitable representation of the data for the task at hand, sometimes referred to as an *embedding*, in such intermediate layers.

1.6.4 Convolutional neural networks

Whereas fully connected layers, as those found in the neural networks just described, operate on vector representations (or matrices if the input is processed in batches), other components of neural networks may operate on data having more than two dimensions: In general neural networks operate on *tensors*, multi-dimensional arrays $X \in \mathbb{R}^{d_1 \times \cdots \times d_m}$ that can be seen as a generalization of vectors and matrices to higher dimensions. In the following I shall use a python-inspired notation to denote a sub-tensor of X, indicating with $X_{[a_1:b_1, a_2:b_2, \ldots, a_m:b_m]}$, with $1 \leq a_i \leq b_i \leq d_i$ for each i, the tensor obtained from X by considering only the components going from a_i to b_i along dimension i,

with the convention that when a_i and b_i are not written all the components along dimension i shall be considered. I shall now define the *convolution operation* in neural networks, although the concept is intuitive once grasped, the equations describing it can become very cumbersome, to mitigate this to some extent I introduce here another notation: Given two tensors A and B having the same shape, $A \diamond B$ shall denote the operation of taking the sum of all entries of their component-wise product (this coincides with the scalar product of A and B if the latter are vectors, and with the Frobenius inner product if they are matrices). This, combined with visual illustrations will (hopefully) ease the understanding of the operation. Essentially a convolution is linear transformation applied locally everywhere preserving the input's structure.

1D convolutions

We shall start by defining the one-dimensional convolution operation between two vectors: Let $v \in \mathbb{R}^l$ and $k \in \mathbb{R}^m$, with $l \geq m$ then we can define the convolution of the two vectors $v * k \in \mathbb{R}^{l-m+1}$ by

$$(v * k)_i = v_{[i:i+m]} \cdot k, \tag{1.49}$$

so that the output of the operation can be visualised as sliding k, often referred to as *convolutional filter* or kernel, along the vector v and taking dot products along the way (figure 1.17). Here we implicitly assumed that at each step the filter is shifted by one unit, more generally we can set this shift, the *stride* of the convolution, to allow bigger jumps, the result of a convolution with stride $s \in \mathbb{N}_+$ is then defined by

$$(v *_s k)_i = v_{[is:is+m]} \cdot k, \tag{1.50}$$

for each i such that $is + m \leq l$. Notice that the application of the convolution operation reduces the size of the input vector whenever $m > 1$, this may be limiting in a deep neural network architecture in which we wish to process the input through several such operations, and can be accommodated by *padding* the input vector i.e. extending it on both sides with 0s in order for the output to have the desired dimension (figure 1.17, right).

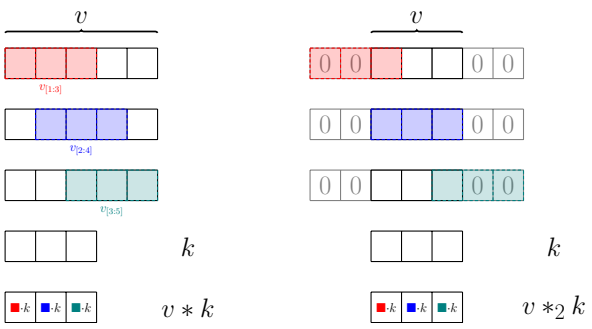


FIGURE 1.17: One-dimensional convolution of two vectors, with stride 1 (left), and with stride 2 and *valid* zero-padding to retain the input dimension (right).

2D convolutions

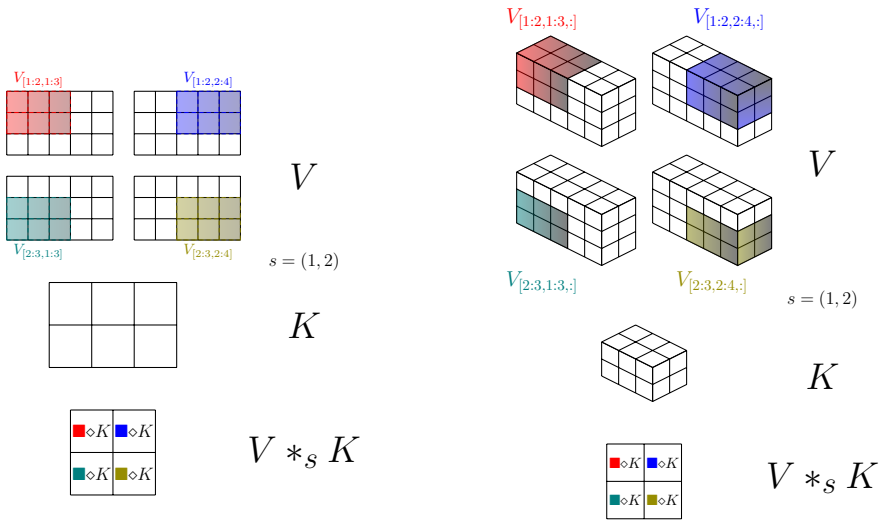


FIGURE 1.18: Two-dimensional convolution over a two-dimensional or a three-dimensional input.

The operation generalises naturally when the input and the convolutional filters are two-dimensional: Suppose now $W \in \mathbb{R}^{l_1 \times l_2}$ and $K \in \mathbb{R}^{m_1 \times m_2}$, with $l_1 \geq m_1$ and $l_2 \geq m_2$ and let $s = (s_1, s_2) \in \mathbb{N}_+^2$. We define then the convolution of the two matrices with stride s by

$$(W *_s K)_{i,j} = W_{[is_i:is_1+m_1, js_2:js_2+m_2]} \diamond K \tag{1.51}$$

for all i such that $is_1 + m_1 \leq l_1$ and $is_2 + m_1 \leq l_2$ (figure 1.18, left). Again padding can be used to accommodate the desired output dimensions. Similarly 2D convolutions can be applied to three-dimensional inputs, as it is typically done when processing an image with the third dimension representing the pixel intensity in the channels R,G and B, in this case $W \in \mathbb{R}^{l_1 \times l_2 \times l_3}$ and $K \in \mathbb{R}^{m_1 \times m_2 \times l_3}$, and the convolution is defined by [8]

$$(W *_s K)_{i,j} = W_{[is_i:is_1+m_1, js_2:js_2+m_2, :]} \diamond K \tag{1.52}$$

Finally, we can have any number n of filters stacked together, with n typically being referred to as the number of output features of the convolution operation. In this case $K = [K_1, \ldots, K_n]$, with each $K_i \in \mathbb{R}^{m_1 \times m_2 \times l_3}$ if we're convolving over a three-dimensional input, and the result of the operation is defined by

$$(W *_s K)_{i,j,k} = W_{[is_i:is_1+m_1, js_2:js_2+m_2, :]} \diamond K_k. \tag{1.53}$$

A *convolutional layer* in a neural network is then defined by one of the above operations combined with the addition of a bias term, followed by the application of an activation function such as those described in the previous subsection. Another operation commonly employed in convolutional neural networks is that of *pooling*, a down-sampling technique that reduces the dimensions of the input, we can see this again as a window sliding across the input, with a given size and stride, whose output at each step is a

[8] Notice that this is a 2D convolution as the convolutional filter slides along the first two dimensions, the output is still a two-dimensional matrix (figure 1.18, right).

summary statistics of the values of the considered sub-tensor of the input, such as the average or the maximum. Finally, typical convolutional neural networks (CNNs) also include fully connected layers which operate on the input tensor after a *flattening* operation which preserves the data in the tensor and simply reshapes it into a vector. It is worth pointing out that in principle the convolution operations defined above can be expressed as simple matrix multiplications, convolutional layers can then be seen as a special case of fully connected ones in which the weight matrices are very sparse and contain many repeated values, this *parameter sharing* allows them to reduce the computational burden of processing high-dimensional data and acts again as a form of regularisation.

1.6.5 Attention

The self-attention mechanism has been recently popularized in the context of natural language processing [73] with the introduction of the transformer neural network architecture. Since then it has been proven to be among the most successful methods for learning from sequential data [74][75] both in the context of natural language [76][77] and biological sequences [78][79], allowing notably to model long range dependencies, a problem with which previously employed architectures such as recurrent neural networks (RNNs) struggled [80]. In a nutshell, self-attention defines a mechanism to update one object z_i using all elements in a set $\{z_j\}_{j=1,\dots,m}$ through a (*query, key, value*) triplet of functions acting on individual objects. The update replaces z_i by a weighted average of all $value(z_j)$, where the weights are computed from $query(z_i)$ and the corresponding $key(z_j)$. The *value* function therefore defines what information of an object z_j should be provided to others, and the interaction between *query* and *key* determines how much is shared from a given z_j to another z_i. For example, the popular dot-product attention (figure 1.19), used in the original transformer architecture [73], relies on learned linear embeddings of vectors $z \in \mathbb{R}^d$ to define a query q, a key k and a value v:

$$q_i = z_i W^Q \in \mathbb{R}^d, \quad k_i = z_i W^K \in \mathbb{R}^d, \quad v_i = z_i W^V \in \mathbb{R}^d. \tag{1.54}$$

Denoting $K \in \mathbb{R}^{d \times m}$ the matrix whose columns are the $(k_j)_{j=1,\dots,m}$, it updates every element z_i by $z'_i = \sum_{j=0}^{m} s_{i,j} \cdot v_j$, where $(s_{i,1}, \dots, s_{i,m}) := \text{softmax}\left(\frac{q_i^\top K}{\sqrt{d}}\right)$ simply contains dot products between q_i and every k_j, scaled by $\sqrt{d}$ and forced to sum to one by the softmax function. The weights $s_{i,j}$ are known as *attention scores* and indicate how much the value v_j contributes to the updated representation z'_i of z_i.

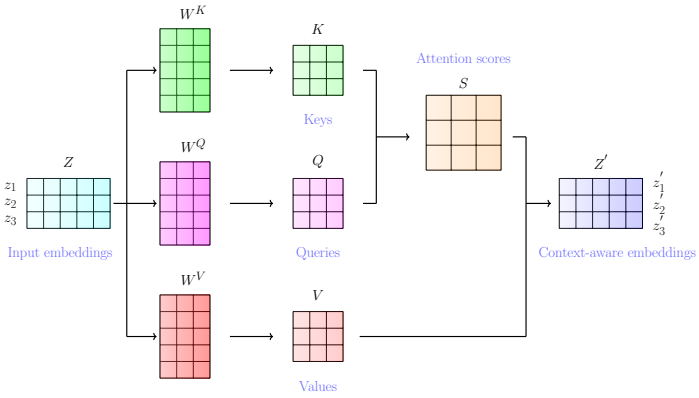


FIGURE 1.19: Visualization of the scaled dot product attention mechanism, the keys, queries and values are computed as $K = ZW^K$, $Q = ZW^Q$, $V = ZW^V$, the updated embeddings z' are then given by the product SV via the attention scores matrix $S = \text{softmax}(QK^T)/\sqrt{d}$.

Several such updates can be recursively applied to the m elements, progressively sharing information across them and thus allowing for *context-aware* embeddings, for instance in the context of machine translation, in which attention has been originally introduced, this allows taking into account the context, i.e. the sentence, a word finds itself in, as the latter may affect its meaning. Finally, often what is used in practice is multi-head self attention, which conceptually allows focusing on several different features of the inputs. It is realized using h so-called *attention heads*, that is h different triplets (W_i^Q, W_i^K, W_i^V), $i \in \{1 \dots h\}$, each computing a set of outputs with the aforementioned attention mechanism. The outputs corresponding to each input and the different attention heads are then concatenated and multiplied by a common matrix W^O. Because the *query*, *key* and *value* functions act on single objects, they define a flexible framework to share information across elements of a set, regardless of their number or order. As we shall see this natural *equivariance* (see subsection 1.6.6) with respect to the order of the input elements may be beneficial but isn't always a desired property, for instance in the aforementioned context of machine translation one typically wishes to take into account the position of a word inside a sentence and not represent the latter simply as a "bag of words". To obviate the issue the authors in [73] resort to *positional encoding*, explicitly adding this positional information to the representations given in input to the network, in particular for each position p they define the vector $\text{PE}(p)$ via

$$\text{PE}(p)_{2i} = \sin\left(\frac{p}{10000^{\frac{2i}{d}}}\right), \qquad \text{PE}(p)_{2i+1} = \cos\left(\frac{p}{10000^{\frac{2i}{d}}}\right), \tag{1.55}$$

for each $i \leq d/2$, where d (assumed to be even) is the dimension of the vector that the encoding is added to. This choice of using sine and cosine functions for positional encoding in transformers, rather than directly encoding the position as a single scalar (e.g. just appending it to the input representation), is motivated by the fact that the periodicity of the functions helps models to generalize to sequences longer than those encountered during their training, and by the fact that this can allow the network to easily exploit relative positions as for any k $\text{PE}(p+k)$ can be represented as a linear function of $\text{PE}(p)$.

1.6.6 Invariance and equivariance

In recent years it has become increasingly clear that deep neural networks can greatly benefit from the exploitation of the symmetries underlying the input data. Specifically, depending on the task at hand, one may wish for the learned function to be *equivariant* for certain transformations of the input, which can be formalized as a group $\mathcal{G}$ acting on the input space, so that whenever an element of the group acts on the input the output is transformed accordingly. We have seen for instance how the self-attention mechanism is equivariant for permutations on the input so that $\text{Attention}(\sigma(z_1, \cdots, z_m)) = \sigma(\text{Attention}((z_1, \cdots, z_m)))$ for any permutation in the symmetric group S_m. On the other hand for different problems one may wish for the learned function to be *invariant* for certain transformations: $\hat{f}(g(x)) = \hat{f}(x)$ for any $g \in \mathcal{G}$.

When discussing convolutional neural networks we already have encountered both cases, indeed the convolution operation is equivariant [9] for the group of translations $s_1\mathbb{Z} + s_2\mathbb{Z}$ defined by the stride vector $s = (s_1, s_2)$, whereas subsequently applying a maximum pooling step makes the whole operation translation-invariant. These are often desired properties in image processing tasks, for instance in the context of object classification we wish the network to be still able to classify the input image even if the concerned object in it is shifted by some amount of pixels. In CNNs such properties are attained through the parameter sharing which allows reducing the complexity of the model: In general enforcing the network to be invariant or equivariant effectively reduces the input space X to its quotient $X/\mathcal{G}$, this typically leads a model to generalize better because the latter doesn't overfit to specific configurations of the input (such as the position of an object in an image) which are irrelevant to the learning task. Furthermore, accounting for such transformations generally leads to more efficient learning as not taking them into account in the structure of the network warrants for *data augmentation* if one doesn't want such potential transformations of the input to affect the model's performances: standard CNNs for instance are not invariant for reflections or rotations of input images so that one may need to enrich the training dataset with many possible variations of the same image in order for the network to handle these transformations correctly. Several structurally invariant or equivariant neural networks have been proposed to deal with different groups of transformations, for instance the previously mentioned limitations of CNNs can be overcome with G-CNNs [81] which through a higher degree of parameters sharing can implement layers equivariant to rotations and reflections. To deal with rotations and translations in 3D space architectures such as the SE(3)-Transformer [82] have been devised. In the context of processing nucleotide sequences, networks equivariant to reverse-complementation have been introduced [83].

One group of transformation of particular relevance for the work presented in this manuscript is that of permutations of sequences inside an MSA. Several expedients have been devised to deal with these. Within the context of population genetics, [84] sorted aligned sequences by similarity before feeding them to a non-invariant function, while [85], [86] directly devised permutation-equivariant prediction functions. More specifically, [85] used a

[9] Or rather locally equivariant given that this ceases to hold when we reach the border of the tensor we're convolving along.

so-called exchangeable network that first applied the same function, a one-dimensional convolutional neural network, to each sequence and then applied a permutation invariant function, such as the average across sequences, at each site. [86] chose a more progressive approach where they also applied the same convolutional network to each sequence, but iteratively concatenated to the obtained representation of each sequence the average over the current representation of all sequences in the alignment. This process makes all sequences converge to a common representation across iterations.

1.6.7 Interpretability

Deep learning models have been often referred to as "black boxes" given the difficulty in deciphering their inner workings. Indeed whereas the coefficient of e.g. a linear regression are easy to interpret as the weights given to each input variable to form a prediction, this becomes increasingly hard for non-linear functions having hundreds of thousands or millions of parameters. Nevertheless, several techniques such as model interpretability tools (e.g. LIME [87]) and visualization methods (e.g. saliency maps [88]), have been developed to provide insights into how deep learning models make predictions [89][90][91]. Feature maps, the outputs of convolutional layers in CNNs, provide a way to understand how the network processes its input, showing for instance how for image-processing related tasks these networks typically capture basic visual elements like edges and textures in their initial layers to then detect more complex patterns and shapes as the image is further processed in the subsequent layers [92]. The idea of saliency maps on the other hand is to compute the gradient of the model's output with respect to the input. This then tells us how much infinitesimal changes in each feature of the input would affect the network's output highlighting the most relevant ones for the model's prediction. Notably in the field of bioinformatics convolutional filters have been shown to be interpretable as sequence motifs, recurring patterns of nucleotides that play functional roles [93] [94]. In attention-based architectures, as the ones that will be presented in the following chapters of this manuscript, one can gain some understanding by inspecting the attention maps S (figure 1.19), with the attention scores indicating for each embedding the ones that contribute the most to its update. We shall visualise several such attention maps in the following chapters and see how they can help us explain the considered networks' capabilities.

1.6.8 Fine-tuning and transfer learning

An advantage of deep neural networks is that one can often exploit their efficiency in feature extraction to adapt a pretrained model to a task which is novel, but related to the one that the model has been previously trained on. Such a technique is known as *transfer learning* and generally consists in stacking additional layers upon those of the pretrained network (eventually removing some of the last layers which may extract features that are too task-specific) and retraining the resulting model on the new task while keeping the pretrained layers "frozen", that is updating only the parameters of the newly added layers during the optimization procedure. This is particularly useful when few labelled data points for the new task are available, notably large language models (LLMs) based on the transformer architecture such

as BERT [76] and GPT [77] are commonly pretrained on a mask language modeling task which involves predicting missing words in a text, allowing them to learn effective representations of words in the context of natural language. Once pretrained, these models can then be employed via transfer learning for a different downstream task such as sentiment analysis or text classification. The same paradigm has been followed for biological sequence processing with transformer models such as ProtBERT [79], ESM [95] and DNABERT [96] being pretrained to predict missing amino acids or nucleotides inside a sequence, to be then used for tasks like secondary structure prediction, protein classification or identification of transcription factor binding sites. On the other hand when the novel task is very similar to the one the network has been already trained on, or even is the exact same task but applied to data points issuing from a different distribution, case known as *domain adaptation*, one can rather resort to *fine-tuning*, simply taking the parameters of the trained model as a starting point for the optimization on the new training dataset, without stacking more layers or necessarily freezing any of them. In particular we shall see in the following chapters how networks trained on data issuing from a specific model of molecular evolution can be easily adapted via fine-tuning if the model or its parameters are changed.

1.6.9 Deep learning in phylogenetics

While the application of deep learning methods in phylogenetics is still relatively underdeveloped with respect to other fields, the recent years have seen the development of several approaches which aim to exploit the promises of deep learning for different tasks related to phylogenetic reconstruction. A nice review of such developments is given by [97] (Figure 1.20).

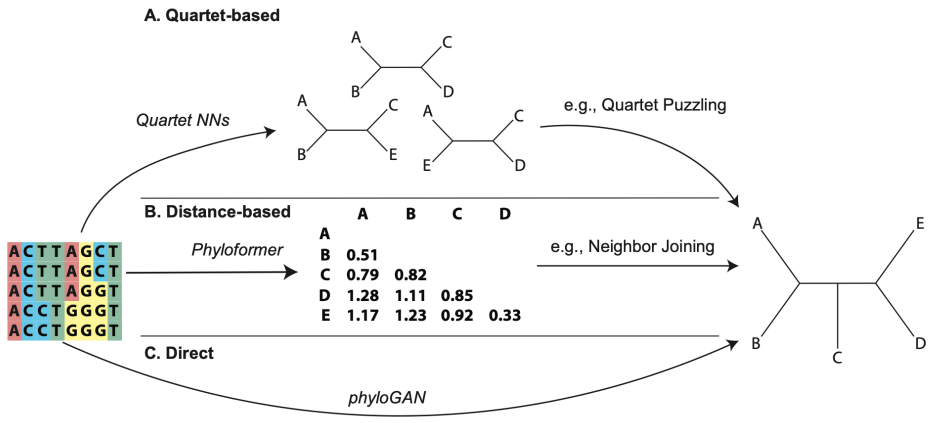


FIGURE 1.20: Neural network-based approaches to phylogenetic reconstruction, including our method, Phyloformer, which will be presented in the next chapter [97].

[98] and [99] proposed likelihood-free methods (section 1.7) for phylogenetic inference framing the problem as a classification across possible topologies. Given the super-exponential growth of the number of possible unrooted tree topologies in the number of sequences, they restricted themselves to quartet trees, that could then be combined to obtain larger trees via the quartet puzzling algorithm (subsection 1.5.3). Both methods relied on convolutional neural networks sensitive to the order of the sequences in the alignment. More

recently, while still considering mostly quartet trees, [100] proposed a network that, exploiting the symmetries of the trees is independent of the sequence order, and reported accuracies similar to [99] using fewer training samples, nevertheless the generalisability of the approach to deal with larger trees is not straightforward and the authors report the method already underperforming, with respect to traditional phylogenetic reconstruction approaches, when dealing with alignments with five sequences. [101] showed that the accuracy of the network introduced in [99] was lower than that of ML or distance methods when evaluated on more difficult problems involving long branches and shorter sequences (200 sites), for both quartet trees and trees with 20 leaves. [102] proposed a generative adversarial network for phylogenetic inference. While also likelihood-free, this approach required a new training for each inference, and did not scale beyond fifteen species. [103] introduced a distance-based learning method for the related problem of adding new tips into an existing tree. Beyond the issues of scalability which derive from framing the problem as a classification one, the aforementioned approaches focus solely on the topology of the inferred tree ignoring its branch lengths, on the other hand [104] proposed a deep learning approach to estimate the branch lengths on given topologies with four leaves. Another class of approaches has rather focused on optimising the tree space exploration, with [105] using firstly a traditional machine learning model, namely a random forest regressor, to predict optimal, likelihood-maximising, SPR moves, the authors then resorted to reinforcement learning and a fully connected neural network architecture in a follow-up paper [106], and showed that, allowing for suboptimal moves during the tree space exploration, the method could outperform state of the art techniques. Finally, beyond direct phylogenetic reconstruction, deep learning-based techniques have shown their potential in dealing with related tasks such as model selection [107], [108] and imputing missing data in incomplete distance matrices [56].

1.7 Simulation-based inference

Beyond phylogenetics, the development of increasingly complex models and simulators to represent reliably the processes underlying the empirical data we can observe, has been carried on in numerous domains of science. While such simulations can model said processes with increasing realism, this comes at the expenses of the computational challenge represented by the inverse problem of inferring the parameters of these models which best describe the data at hand, limiting their applicability for large scale analyses. As already discussed, this computational bottleneck represents the primary reason behind the several simplifying assumptions made by the commonly employed models in phylogenetics in order for the corresponding likelihood function to be tractable. The problem is further exacerbated in the context of Bayesian inference where the goal is typically to calculate, instead of a point estimate of the parameters maximising the likelihood, the whole posterior distribution of the parameters given the observed data and a prior $P(\theta)$ on the former:

$$P(\theta \mid x) = \frac{P(x \mid \theta)P(\theta)}{\int P(x \mid \vartheta)P(\vartheta)d\vartheta}. \tag{1.56}$$

The integral over all the parameters θ in the denominator in equation (1.56) then represents another source of intractability beyond that of the likelihood function in problems with a high-dimensional parameter space, and is typically dealt with via Markov Chain Monte Carlo (MCMC) [109][110] or variational inference (VI) methods [111][112][113].

To tackle this problem of statistical inference under intractable likelihoods several approaches have been devised, falling under the umbrella term of *simulation-based inference*, with the term *likelihood-free inference* being used somewhat interchangeably, the latter can nevertheless be a bit misleading given that several of these approaches do work by trying to estimate the intractable likelihood.

1.7.1 Approximate Bayesian Computation

Among these methods the most popular is indubitably Approximate Bayesian Computation (ABC) [114][115] (figure 1.21) (see [116] for a review of the applications in evolution and ecology).

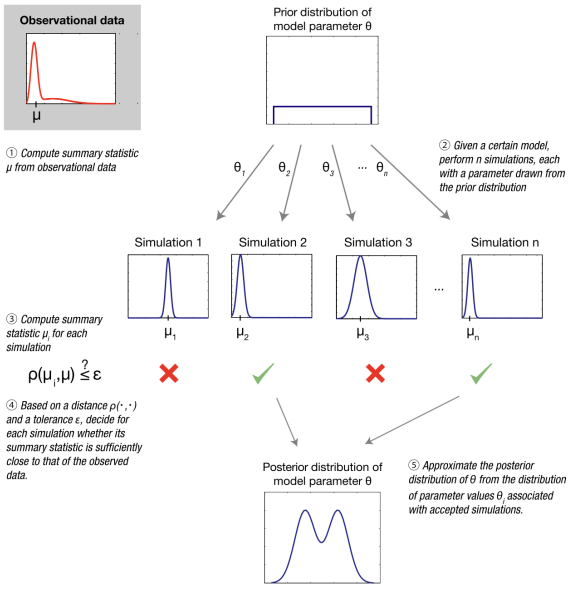


FIGURE 1.21: Diagram of the Approximate Bayesian Computation method for posterior parameter distribution estimation [115].

The idea underpinning the method is to overcome the intractability of the likelihood function using simulated data in order to approximate the posterior distribution via *rejection sampling*. The method operates through the choice of a set of *summary statistics* $S(x)$ for the data x aimed at reducing the dimensionality of the latter, the choice of a distance measure ρ, and of a non-negative *tolerance treshold* ε. It then proceeds as follows: Once computed the summary statistics of the observed data $S(x)$, a given (typically very high) number of parameters $\theta_1, \ldots, \theta_n$ is sampled from the prior distribution and each is fed into the simulator to produce a simulated data point x_{θ_i}, the rejection step then consists in discarding the simulated data points for which $\rho(S(x), S(x_{\theta_i})) > \varepsilon$. The distribution of the parameter values for which the corresponding simulated datapoint has been retained is subsequently used as a proxy of the entire posterior distribution, the idea being that the likelihood of a datapoint x_{θ_i} is approximated by the probability of the condition $\rho(S(x), S(x_{\theta_i})) \leq \varepsilon$ being satisfied. If the summary statistics are *sufficient*,

that is if they capture all the relevant information regarding the parameter θ (more formally, if the conditional distribution of the data x given the statistics $S(x)$ is independent of θ), then inference with ABC becomes exact in the limit of the number of samples $n \to \infty$ and $\varepsilon \to 0$. Despite the widespread application of ABC, the method suffers from several drawbacks. Firstly, it is not *amortized* as inference for a new observation x' requires repeating most of its steps, secondly, the quality of inference depends on the summary statistics which need to be carefully chosen and rarely are sufficient with an almost inevitable loss of information occurring in the dimensionality reduction step. The quality of inference then depends also on the tolerance treshold ε, with large values leading to poor approximations, whereas the acceptance probability goes to 0 along with ε while also typically exponentially decreasing as the dimensionality of the parameter space increases [117], both these factors can then lead to a prohibitively expensive number of simulations necessary to get good approximations.

1.7.2 Neural network-based methods

In recent years numerous approaches to the problem, based on neural networks have seen their development, this is logical given that deep learning models offer the opportunity of solving the aforementioned shortcomings of ABC. Indeed such models are known for their capability in handling high-dimensional data and for *automatic feature extraction* which allows them to progressively learn from the data a set of informative features that are most effective to handle the problem at hand, without requiring human supervision and handcrafted summary statistics. Neural network methods are then typically amortized, often offering very fast inference after the single upfront computational cost of training the model. Thirdly, by using the entirety of the simulated data to train a model without a sampling rejection step, the neural network-based approaches provide a better sample efficiency which generally goes along with better performances of such methods with respect to ABC [118]. Whilst reviewing the vast and rapidly growing literature on such methods is outside the scope of this work, it is worth mentioning the three classes in which most of these can be categorized. The first is known as *Neural Posterior Estimation* (NPE) [119][120] and encompasses methods which directly aim at approximating the posterior distribution (1.56) by training a conditional neural density estimator [121]. The main advantage of these methods is that they allow fully amortized Bayesian inference as, once trained on simulated data, obtaining an estimate of the posterior for a new observed datapoint involves simply a single forward pass in the neural network. *Neural Likelihood Estimation* (NLE) [122][123] on the other hand, aims at approximating the likelihood $P(x \mid \theta)$ instead of the posterior $P(\theta \mid x)$, by training a density estimator to map simulation parameters to data. This approach leads then to a density which can be evaluated and sampled from, when it comes to posterior inference such methods are therefore only partially amortized as their use in conjuction with techniques as MCMC or VI is warranted for each new datapoint. Such is the case as well for the third category of methods, *Neural Ratio Estimation* (NRE) [124][125], which instead of performing conditional density estimation, train neural networks to ultimately predict an approximation of the likelihood ratio function $P(x \mid \theta_0)/P(x \mid \theta_1)$ by training a classifier to discern datapoints

generated under the different parameters θ_0 and θ_1.

While [126] provides a review of the developments in the field of simulation-based inference, the paper only focuses on methods, as those discussed above, whose aim is ultimately to approximate the full posterior distribution $p(\theta \mid x)$, whereas more straightforward approaches to provide point estimates of the true parameters θ as a function of the data x by training a model on simulations, framing the problem as a regression one in a supervised learning context, are not discussed, arguing that their probabilistic interpretation is less straightforward. Still I can try to provide here such an interpretation in the Bayesian framework:

Suppose we want to estimate a parameter θ, having a prior distribution π, given an observation x, with an estimator $f(x)$. Let us recall the definition of a *Bayes estimator* as the one that minimizes the *posterior expected loss*, that is posterior expected value of a loss function given an observation x

$$\mathbb{E}_{\theta|x}[L(\theta, f(x))], \tag{1.57}$$

across all x. Let us now consider a setting in which the observations follow a probability distribution $P(x \mid \theta)$ induced by a probabilistic model $\mathcal{M}(\theta)$ and that we can sample data from the latter via a simulator. We can then adopt a supervised learning approach and use the prior distribution $\pi(\theta)$ simulate training data points $(\theta, \mathcal{M}(\theta))$ in order to train, with a loss function L, a model, such as a deep neural network, to learn an estimator function $f(x)$. In this setting the generalisation error 1.44 is the expected loss over the entire data distribution accounting for the variability in θ described by the prior and in the observed data x given θ, governed by the likelihood $P(x \mid \theta)$:

$$\begin{aligned} R(f) &= \mathbb{E}_{\theta\sim\pi(\theta)}[\mathbb{E}_{x\sim P(x|\theta)}[L(\theta, f(x))]] \\ &= \int\int L(\theta, f(x))P(x \mid \theta)\pi(\theta)dxd\theta, \end{aligned} \tag{1.58}$$

or equivalently, using Bayes' theorem $P(x \mid \theta)\pi(\theta) = \pi(\theta \mid x)P(x)$,

$$R(f) = \int\int L(\theta, f(x))\pi(\theta \mid x)P(x)dxd\theta = \mathbb{E}_{x\sim P(x)}[\mathbb{E}_{\theta|x}[L(\theta, f(x)]], \tag{1.59}$$

where $P(x) = \int P(x \mid \theta)\pi(\theta)d\theta$ is the marginal distribution of the data x. This shows that in this setting the generalisation error is the expected value of the expected posterior loss across all data points. Minimising the generalisation error here, via the training error proxy, is then equivalent to minimising the expected posterior loss across all datapoints which therefore ideally leads the learned function to coincide with the Bayes estimator. The nature of this estimator then depends on the choice of the loss, for instance if one choses a quadratic L^2 loss $L(\theta, a) = (\theta - a)^2$, then the expected posterior loss becomes

$$h(a) = \int (\theta - a)^2\pi(\theta \mid x)d\theta, \tag{1.60}$$

differentiating with respect to a and equating to 0 one then finds

$$a\int \pi(\theta \mid x)d\theta = \int \pi(\theta \mid x)\theta d\theta, \tag{1.61}$$

from which follows, given that the integral on the left hand side equals to one,

that the Bayes estimator $\hat{\theta}$ is given by the posterior mean $\mathbb{E}[\theta \mid x]$. Similarly one can show that an L^1 absolute error loss $L(\theta, a) = |\theta - a|$ leads $\hat{\theta}$ to be the posterior median. On the other hand if one wanted the Bayes estimator to coincide with the mode of the posterior distribution, that is the *maximum a posteriori* (MAP) estimate $\theta_{MAP} = \arg\max_a \pi(a \mid x)$, then the 0-1 loss

$$L(\theta, a) = \begin{cases} 0 & \text{if } a = \theta \\ 1 & \text{if } a \neq \theta \end{cases} \tag{1.62}$$

would have to be employed, but the latter not being differentiable in θ and having a zero gradient almost everywhere, makes it not suitable for this learning setting whereas using a smooth approximation thereof, such as the logistic function, leads again the Bayes estimator to coincide with the posterior mean. Finally it is worth noting that in the case in which, instead of a single scalar value, multiple parameters are estimated at once so that θ is a vector in $\mathbb{R}^d$, all the above reasonings are straightforward to generalise, in this case for instance the L^1 loss would lead the Bayes estimator to be the vector of the marginal medians $\hat{\theta} = (\text{median}(\pi(\theta_1|x)), \text{median}(\pi(\theta_2|x)), \ldots, \text{median}(\pi(\theta_d|x)))$.

Although to our knowledge a comprehensive review of methods employing the paradigm just described is missing in the literature, this approach is becoming increasingly popular and has proven its success in the estimation of model parameters in disparate fields such as spatial point process theory [127], physical modeling [128], phylogenetics [104], epidemiology [129] and notably population genetics (a review of such a supervised learning paradigm in the field, based either on real or simulated data, is provided by [130]), with the estimation of effective population size [86], the identification and characterization of selective sweeps [84] or recombination hotspots [131]. For the latter task the authors in [85] adopt a different approach, following [132] and [133], reframing the regression problem to predict at once the mean and variance of an approximate posterior distribution instead of a simple point estimate of the parameters, this allows them to get uncertainty estimates of the inferred parameters and fit the approach in the previously described framework of predicting posterior distributions. The authors of [86] on the other hand, while still predicting point parameter estimates, subsequently use the latter as summary statistics for an ABC approach, this again allows them to approximate the posterior distribution and explore the advantages of such a combination of the two frameworks. On the other hand such expedients may not be necessary when a model is trained on a classification task instead of a regression one, as the former approach already provides the probabilities for the parameter of interest being in each class and thus a posterior distribution, this is the case e.g. for the previously mentioned topology classifiers for phylogenetic inference, and will also hold true for the neural networks I shall present in chapters 3 and 4.

1.8 Supervised learning to estimate evolutionary distances

Given the context and the premises provided in this chapter we are ready to encounter in the following one our novel approach to phylogenetic reconstruction. The approach, falling into the framework described in the previous section, will consist in generating labelled training data, with each data point consisting in an MSA $(s^1, \ldots, s^n)$ as well as the pairwise distances $(d_{s^1 s^2}, d_{s^1 s^3} \ldots, d_{s^2 s^3}, \ldots, d_{s^{n-1} s^n})$ on the tree that its sequences evolved along. This will be done through a simulator, sampling from a probabilistic model of evolution such as those described in section 1.2, to subsequently train a deep neural network for the regression task of predicting evolutionary distances on such a dataset. Such a prediction will be made jointly for all the pairs of sequences in the input MSA, rather then considering them independently. This will allow the network to exploit the information contained in the whole alignment for each distance estimate and thus to overcome this limitation of traditional distance estimation approaches. The predicted distances can then be used to reconstruct a phylogeny via any of the methods described in section 1.4, with the statistical consistence of distance-based methods justifying the approach with the guarantee of recovering the true phylogenetic tree as long as we can compute sufficiently accurate estimates of the evolutionary distances. In order to feed the input data into the network, we need a suitable representation, we shall resort to one-hot encoding to represent different characters in an MSA as a set of orthogonal vectors, each containing all zeros but a one in the index corresponding to the character, this is the prevalent representation for biological sequence data [134] and allows to avoid the inductive bias of implicitly assigning an unmeaningful ordering of the characters. Such encodings can then be stacked together, preserving the spatial relationship of the MSA, to obtain a representation of the latter as a tensor (figure 1.22).

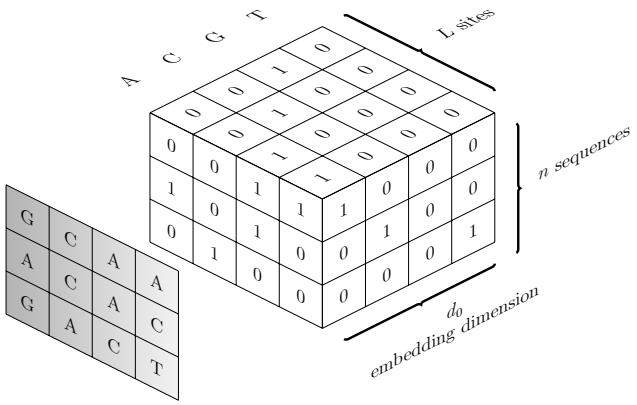


FIGURE 1.22: One-hot encoding of a multiple sequence alignment, in practice, except for the model presented in chapter 4, we will actually be working with amino acid sequences so that each character will be encoded with a vector of size 22 (to account for each amino acid and the two symbols X and $-$).

Notice that by feeding the entire MSA into the network we are avoiding any potential loss of information entailed by the use of summary statistics, rather relying on the network to learn automatically the features that are most relevant for the prediction task. Given the importance of the prior distribution for the learning task, the parameters of the simulations will be chosen with

a particular care, trying to match them with those inferred from empirical alignments. Finally, given the discussion in subsection 1.6.6, we devised our neural network architecture to exploit all the symmetries of the problem. One such symmetry is the invariance of the tree topology and branch lengths to permutations of sequences in the MSA: $p(\tau|s^1,\ldots,s^n) = p(\tau|\sigma(s^1),\ldots,\sigma(s^n))$ for any $\sigma \in S_n$, such an invariance will be attained through a network which is equivariant to sequence permutations, any permutation σ of the input sequences will result in the corresponding permutation $\sigma' \in S_{\binom{n}{2}}$ induced by σ on the predicted distances. The other symmetry stems from the i.i.d assumption made by the evolution models presented in section 1.2 which entails that $p(\theta|X_1,\ldots,X_L) = p(\theta|\sigma(X_1),\ldots,\sigma(X_L)) \quad \forall\sigma \in S_L$, where X_i denotes the i-th column of the alignment, again this will be accounted for with the proposed network being invariant for permutations of the sites. Both invariances will be attained through a high degree of parameter sharing.

Chapter 2

Phyloformer

2.1 Preface

In this chapter I will present the main work of this thesis, the development of Phyloformer, a neural network that predicts evolutionary distances in order to reconstruct phylogenetic trees, relying on the potentials offered by deep learning and simulation-based inference, in an attempt to overcome some of the limitations of traditional phylogenetic reconstruction methods presented in the previous chapter. The main findings of this work have been put together in an article, "Phyloformer: Fast, accurate and versatile phylogenetic reconstruction with deep neural networks", a preprint of which can be found at `https://www.biorxiv.org/content/10.1101/2024.06.17.599404v1`, and which is currently submitted to Molecular Biology and Evolution. I shall incorporate the text here in its entirety, with minor adjustments to make it fit the format of the present manuscript. The development of the method has been a long and winding journey with numerous challenges along the way. The text presented here represents thus only the façade of the great amount of work that has been put into the project. The discussion of all attempted approaches, along with the uncountable failed or inconclusive experiments, would have made the present work lenghty and difficult to read. The choice to simply present the polished manuscript, the tip of the iceberg, has therefore been made, nevertheless I shall provide some background on the project here and an additional discussion will be carried out at the end of this chapter.

An earlier preprint (`https://www.biorxiv.org/content/10.1101/2022.06.24.496975v1`), dating back to 2022 and for which we already received positive feedback and interest both from the machine learning and the phylogenetics communities, predates the one presented in this chapter. In it we showed the promises of our approach, demonstrating how it could surpass a previously published neural network-based method [99] and attain equal or superior performances with respect to a maximum likelihood approach (figure 2.1), whilst running significantly faster, when trained and tested on a complex model of evolution, showing as well its robustness to deviations from the parameters the network has been trained on.

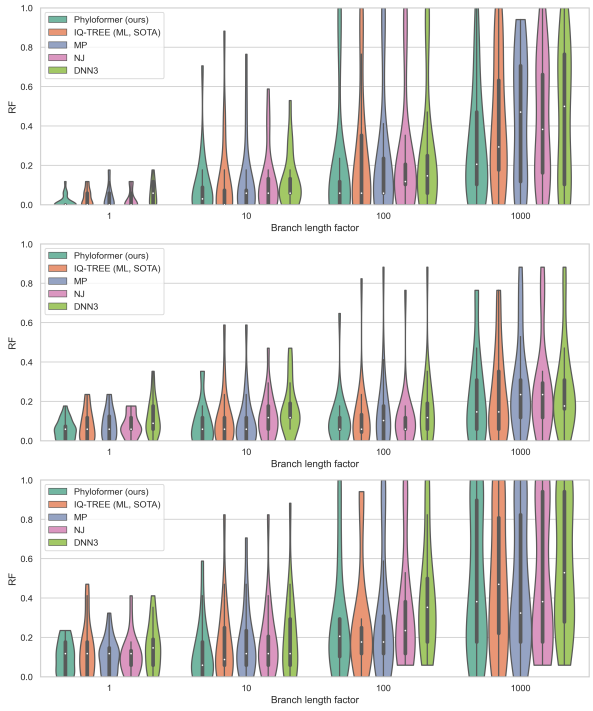


FIGURE 2.1: Violin plots of normalized Robinson-Foulds distances between the true trees and the trees estimated by 5 different methods, on datasets generated with different sets of parameters which reflect different difficulties in phylogenetic reconstruction. Results are shown for DNN3 (the neural network presented in [99]), IQ-TREE with ModelFinder, Maximum Parsimony , NJ (with Hamming distances), and Phyloformer. Although our network was trained only on simulations generated with the easiest set of parameters (first dataset, upper left corner) it manages to always concentrate more mass towards low RF distance values than all other methods. As a result it outperforms them all in 9 out of 12 test datasets in terms of mean RF distance.

The complex model of evolution, developed in [99], combined 9 different amino acid substitution matrices, modeling rate heterogeneity across sites and site-specific amino acid equilibrium frequencies, as well as heterogeneity across branches in both the rate of sequence evolution and equilibrium frequencies. The parameters of such a model, notably the branch lengths, however were hard to interpret, motivating us to move on, resorting to a different alignment simulation pipeline. Still we wanted to showcase the capabilities of our method to deal with complex models under which likelihood calculations are hard, this lead us to consider in the final manuscript Cherry, which accounts for pairwise site dependencies and SelReg, which models heterogeneous selective pressures. Moreover, while in the first preprint we showed that our approach could surpass traditional distance-based methods even under a more common model of evolution, the gap between Phyloformer and IQ-TREE widened quickly as the number of leaves of the trees we tested the methods on grew (figure 2.2). These rapidly decaying performances of our network on larger trees had to be dealt with in order to provide to the end users a versatile and reliable phylogenetic reconstruction tool beyond a simple proof of concept.

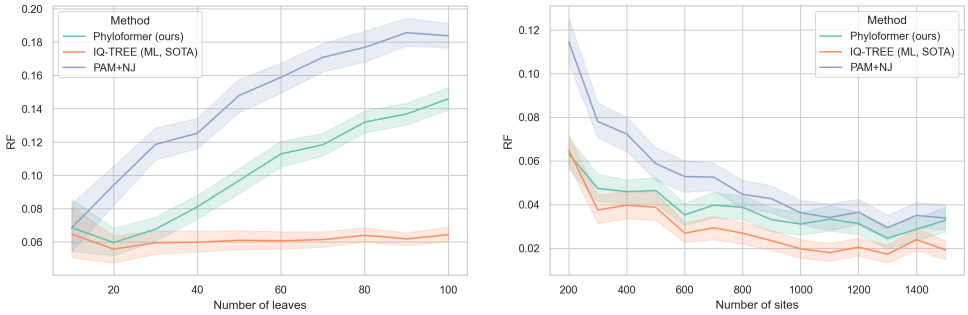


FIGURE 2.2: Mean performances of IQ-TREE (ML, State Of The Art, orange), PAM+NJ (distance method, blue), and Phyloformer (ours, green) on datasets varying in the number of leaves (left) or the number of sites (right). Shaded areas correspond to 95% confidence intervals of the mean. 150 trees were reconstructed for each number of leaves or each number of sites.

The main hypothesis to explain the degraded performance of Phyloformer on larger trees was the observation that, simulating all branch lengths with the same distribution as we did, larger trees have a longer average path length between leaves, which results in longer distances (given by the cumulative branch lengths along the path). These long distances can fall outside the range of distances seen by the network during training. Indeed we realized that, while perhaps reasonable for trees of fixed size, the simulation scheme we adopted, directly following the one presented in [99], isn't suited for trees with varying number of leaves: In realistic datasets subsampling larger trees by pruning them should ideally lead to the same distribution on the pairwise distances. A great deal of effort has then been put into the development of a simulation scheme for trees of varying sizes with realistic branch lengths. Despite several discussions with researchers who have devoted their career to phylogenetics, it has been hard to find a consensus on what would be the ideal way to simulate such trees. Indeed the work is unprecedented as, even if simulations have been used to compare the performances of phylogenetic tree reconstruction methods for decades, the dependency of the reconstruction method itself on the prior used to train the model in the presented context of simulation based inference warrants for an increased realism of the simulations, beyond the one that would be sufficient to simply compare the performances of different methods. The simulation procedure we eventually adopted still has its limitations but I do believe it represents a step forward in this direction given the reasonable performances on empirical data of Phyloformer, the model presented in this chapter as well as those of Deepelican, presented in the following one.

Phyloformer: Fast, accurate and versatile phylogenetic reconstruction with deep neural networks

Luca Nesterenko[1†], Luc Blassel[1†], Philippe Veber[1], Bastien Boussau[1‡], Laurent Jacob[2‡]

[1]Laboratoire de Biométrie et Biologie Évolutive, Villeurbanne, France.
[2]Laboratory of Computational and Quantitative Biology, Paris, France.

Contributing authors: luca.nesterenko@univ-lyon1.fr; luc.blassel@univ-lyon1.fr; philippe.veber@univ-lyon1.fr; bastien.boussau@univ-lyon1.fr; laurent.jacob@cnrs.fr;
[†]Equal contribution
[‡]Equal contribution

Abstract

Phylogenetic inference aims at reconstructing the tree describing the evolution of a set of sequences descending from a common ancestor. The high computational cost of state-of-the-art Maximum likelihood and Bayesian inference methods limits their usability under realistic evolutionary models. Harnessing recent advances in likelihood-free inference and geometric deep learning, we introduce Phyloformer, a fast and accurate method for evolutionary distance estimation and phylogenetic reconstruction. Sampling many trees and sequences under an evolutionary model, we train the network to learn a function that enables predicting the former from the latter. Under a commonly used model of protein sequence evolution and exploiting GPU acceleration, it outpaces fast distance methods while matching maximum likelihood accuracy on simulated and empirical data. Under more complex models, some of which include dependencies between sites, it outperforms other methods. Our results pave the way for the adoption of sophisticated realistic models for phylogenetic inference.

Keywords: phylogenetic reconstruction, neural network, attention, machine learning, regression

2.2 Introduction

Phylogenetics, the reconstruction of evolutionary relationships between biological entities, is used in many research domains to provide essential insights into evolutionary processes. It is employed in epidemiology to track viral spread [135], in virology to identify events of recombination [136], in biochemistry to evaluate functional constraints operating on sequences [137], in ecology to characterize biodiversity [138]. Central to these works, the phylogeny is a binary tree whose internal nodes correspond to ancestral entities, branches represent the amount of evolutionary divergence, and leaves correspond to extant entities. Most of the time, molecular phylogenies are estimated from aligned nucleotide or amino acid sequences using probabilistic model-based approaches in the Maximum Likelihood (ML) or Bayesian frameworks. The models typically describe the probability of substitution events along a branch of the phylogenetic tree, whereby an amino acid (or nucleotide) is replaced by another. Parameters of these models include rates of substitution, the topology of the phylogeny, and its branch lengths—representing the expected number of substitutions per site occuring along that branch. In the ML framework, parameter inference is achieved by heuristics that attempt to maximize the likelihood. In the Bayesian framework, it is often achieved by Markov Chain Monte Carlo algorithms that sample the posterior distribution. Both approaches are computationally expensive for two reasons. First, they need to explore the space of tree topologies, which grows super-exponentially in the number of leaves [1]. Second, this exploration involves numerous computations of the likelihood, each obtained with a costly sum-product algorithm (Felsenstein's pruning algorithm [139]). This computational cost has kept researchers from using more realistic models of sequence evolution, which would for instance take into account interactions between sites of a protein (as in e.g., [8]). Such simplifications are well-known to be problematic, as several reconstruction artifacts directly associated to model violations were discovered early in the history of model-based phylogenetic reconstruction [140]–[142]. Much faster methods exist, but they are generally less accurate [143]. In particular, distance methods (e.g., Neighbor Joining (NJ) [24], BioNJ [144], FastME [46]) build a hierarchical clustering of sequences based on some estimate of their evolutionary pairwise distances, *i.e.*, the sum of the branch lengths along the path between pairs of sequences on the true unobserved phylogenetic tree. NJ is guaranteed to reconstruct the true tree topology if applied to the true distances [27], making the problem of estimating the tree and the set of distances equivalent. In practice, distances are typically estimated under the same probabilistic models as ML and Bayesian methods but considering each pair separately—whereas the latter consider all sequences at once—which greatly simplifies computations but discards part of the global information contained in the full set of homologous sequences.

Here we present Phyloformer, a phylogenetic inference method exploiting all sequences at once with the speed of distance methods. Importantly, Phyloformer can handle complex models of sequence evolution for which likelihood computations would not be feasible. We build on recent advances in deep learning for multiple sequence alignments [MSAs, 145] and in the likelihood-free inference paradigm (Fig. 2.3). Sometimes referred to as simulation-based inference [118], this paradigm exploits the fact that simulating data under probabilistic models of sequence evolution is computationally

affordable, even in cases where computing likelihoods under these models is expensive. Through simulation we sample a large number of phylogenetic trees and MSAs evolved along these trees, given a probabilistic model under which we want to perform phylogenetic inference. We then learn a function that takes an MSA as input and outputs the evolutionary distances between all pairs of sequences on the tree. This function provides a point inference of the *full set of pairwise distances* under the chosen probabilistic model, conditional to the observed MSA. Learning the function is computationally intensive, but once done, Phyloformer can be used in combination with a distance method to reconstruct a tree from an MSA very rapidly, regardless of the complexity of the model of sequence evolution. We show that under the common LG+GC model [146], Phyloformer leads to phylogenies as accurate as state of the art ML methods but runs two orders of magnitude faster. Under more realistic models, *e.g.* accounting for pairwise dependencies between sites, Phyloformer provides more accurate estimates than all other inference methods.

Related work [97] offer a recent review on deep learning for phylogenetics. [98], [99] proposed likelihood-free methods for phylogenetic inference, by casting the problem as a classification across possible topologies. Given the super-exponential growth of the number of possible unrooted tree topologies in the number of sequences, they restricted themselves to trees with four leaves (quartet trees), that could then be combined to obtain larger trees [68]. Both methods relied on convolutional neural networks and were therefore sensitive to the order of the sequences in the alignment and restricted to a fixed sequence length—smaller sequences being accommodated with padding. More recently, while still only considering quartet trees, [100] proposed a network that was independent of sequence order, and reported accuracies similar to [99] using fewer training samples. [101] showed that the accuracy of the network introduced in [99] was lower than that of ML or distance methods when evaluated on difficult problems involving long branches and shorter sequences (200 sites), for both quartet trees and trees with 20 leaves. [102] proposed a generative adversarial network for phylogenetic inference. While also likelihood-free, this approach required a new training for each inference, and did not scale beyond fifteen species. [103] introduced a distance-based learning method for the related problem of adding new tips into an existing tree. Our work is also related to the recent corpus of methods predicting contact between pairs of residues from MSAs, a crucial step in protein structure prediction [145], [147]. These methods infer distances between sites (columns in the MSA) whereas we infer distances between sequences (rows in the MSA). Our network is trained end-to-end to predict distances, whereas the [145] network is pre-trained on a masked language modeling task to learn a data representation that is then used as input for residue contact prediction learning.

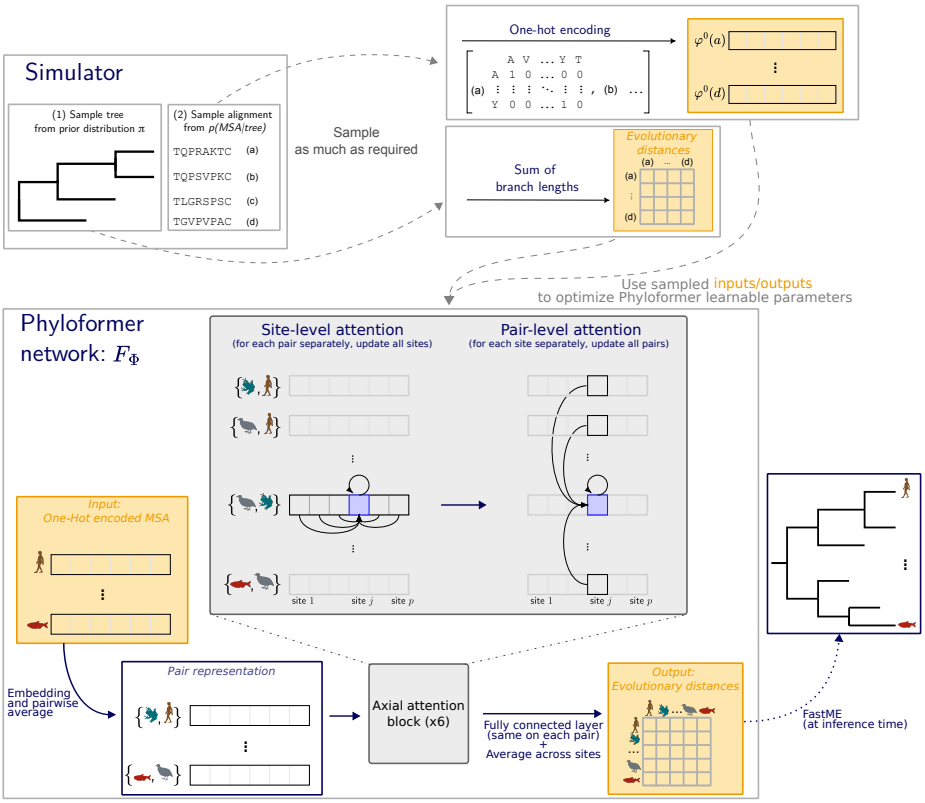


FIGURE 2.3: Learning a function that reconstructs a phylogenetic tree from an MSA. We simulate phylogenetic trees and evolve MSAs along these trees under a given probabilistic model (Simulator panel). Once encoded, we use the examples of MSAs and corresponding trees to optimize the prediction function, described in the Phyloformer network panel. Each square denotes a vector of dimension d representing one site in one sequence or pair in the MSA, where the value of d can be different at each step. Phyloformer starts (bottom left) from a one-hot encoded MSA, and builds a representation for the pairs. These pairs then go through several axial attention blocks which iteratively build a new representation for each pair that accounts for the entire MSA, by successively sharing information across sites within each pair and across pairs within each site (See The Phyloformer neural network). The sharing mechanism relies on self-attention (central panel). We finally use a fully connected network on each site of the resulting representation and average across sites to predict the evolutionary distance between each pair (bottom right). At training time, we compare these distances against real one to optimize the network parameters Φ. At inference time, we feed them to FastME to reconstruct a phylogeny.

2.3 Results

2.3.1 Likelihood-free phylogenetic inference with Phyloformer

Phyloformer is a learnable function for reconstructing a phylogenetic tree from an MSA representing a set of homologous sequences (Fig. 2.3). It produces an estimate, under a chosen probabilistic model, of the distances between all pairs of sequences, which is then fed to a fast clustering method to infer a phylogenetic tree. The key feature of Phyloformer is its ability to produce pairwise distance estimates that account for all sequences in the alignment—providing more accuracy than the fast approaches that consider each pair of sequences independently—without computing likelihoods—leading to much faster inference than full ML or Bayesian approaches.

For a given model of sequence evolution $p(\text{MSA}|\tau,\theta)$ describing how an observed MSA evolves conditionally to a phylogeny τ and evolutionary parameters θ—substitution rates, equilibrium frequencies—and priors $\pi(\theta)$ and $\pi(\tau)$, we generate a large number of samples $\{(\text{MSA},\tau,\theta)\}$ under the unnormalized posterior $p(\text{MSA},\tau,\theta) = p(\text{MSA}|\tau,\theta)\pi(\theta)\pi(\tau)$ (Fig. 2.3, Simulator panel). We then use these samples to build a function estimating the tree τ, by optimizing a parameterized function $F_\Phi(\text{MSA})$ that takes the MSA as input and outputs an estimate of τ. More precisely we output point estimates of the distances between pairs of aligned sequences in τ, and minimize the average absolute error between these point estimates and the real distances, which amounts to estimating the median of the posterior distribution $p(\text{MSA}|\tau,\theta)$, see Supplementary Methods 2.6.4. Assuming that the family of functions described by F_Φ is expressive enough and that enough samples are used, this approach offers posterior inference under the model (π,p), effectively replacing likelihood evaluations by samplings of $p(\text{MSA}|\tau,\theta)$.

Our F_Φ relies on self-attention—a mechanism popularized by the Transformer architecture [73]—to build a vector representation for each pair of sequences that contains all the information from the MSA required to determine the corresponding distance. During each self-attention block, the representation of each pair is updated using information extracted from all others. The learnable weights of the block determine how much each pair weighs in the update of any particular pair, as well as what information it contributes. More precisely, we maintain a separate representation for each position within each pair, and alternate between a separate update for each site—whereby information is shared among pairs as we just described—and a similar separate update for each pair whereby information flows among the sites [145], [148]. Following the attention blocks, we use a fully connected neural network on the enriched representation of each pair of sequences to predict the corresponding distance on the phylogenetic tree. The initial representation of each pair is an average of the one-hot encodings of its sequences, that is blind to the rest of the MSA. Because the learnable weights are chosen to make the predicted distances as close as possible to the real ones, we expect them to adaptively extract an MSA-aware representation for each pair, that captures the relevant information from all sequences.

2.3.2 Under a standard model of evolution, Phyloformer is as accurate and much faster than ML

We first assessed the performances of Phyloformer on data generated under the LG+GC model of sequence evolution which combines the LG matrix of amino-acid substitution [146] with rate heterogeneity across sites [149]. The LG model is widely used, implemented in many phylogenetic tools [150]–[153] and amenable to likelihood computation, making it a good model to compare against state of the art ML inference methods. Following [154], we sampled trees under a birth-death process, subsequently rescaling the branches to simulate variations of the rate of sequence evolution. We chose simulation parameters to match empirical data in the HOGENOM [155] and RaxMLGrove [156] databases (see Online methods). We then evolved MSAs of 50 sequences and 500 sites under LG+GC along these trees, and used the resulting data to train Phyloformer. We compared Phyloformer (PF) followed by FastME to reconstruct the tree from estimated distances against two ML methods,

IQTree and FastTree, and one distance method, FastME using LG pairwise distances. Fig. 2.4a shows the average Kuhner-Felsenstein (KF) distance [58] between the true and reconstructed phylogeny for each of these methods over 500 samples from the same model for increasing numbers of leaves. The KF distance is widely used to compare phylogenies and captures both topological and branch length reconstruction errors. Phyloformer achieved a performance similar to ML methods. It is noteworthy that this performance was stable across numbers of leaves, even though our network was trained on 50-leave phylogenies only. The performance was also stable when doing inference over a range of sequence lengths, even though Phyloformer was trained only on alignments with 500 positions (Supplementary Figs. 2.21 and 2.22). The distance method was much less accurate. Interestingly, the high accuracy of Phyloformer was achieved with the lowest runtime among all benchmarked methods (Fig. 2.5). In particular, it was up to 135 times faster than the ML method IQTree, for a similar accuracy. FastTree—a faster and supposedly less accurate heuristic for ML—also had similar accuracy on this dataset, but remained one order of magnitude slower than Phyloformer. Phyloformer was even twice as fast as FastME combined with LG distances. As Phyloformer itself runs FastME to reconstruct a tree from its distance estimates, this difference indicates that inferring distances that exploit the full MSA with a trained Phyloformer on a GPU is actually faster than computing the ML distances independently for each pair. Conversely, Phyloformer was the most memory intensive method, using up to 7.4GB of GPU RAM (Supplementary Fig. 2.23), although this can be halved in some cases by using automatic mixed precision.

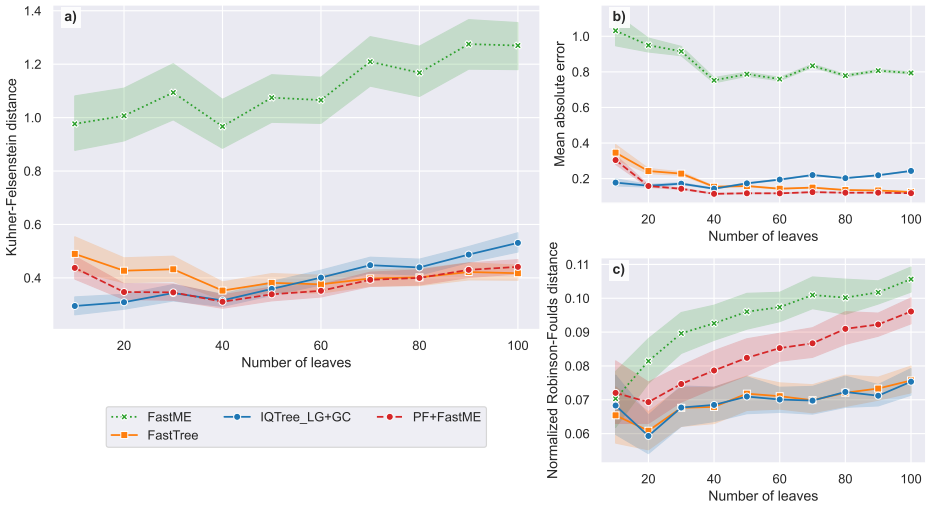


FIGURE 2.4: Performance measures for different tree reconstruction method. **a)** Kuhner-Felsenstein (KF) distance, which takes into account both topology and branch lengths of the compared trees; **b)** mean absolute error (MAE) on pairwise distances, which ignores topology; **c)** normalized Robinson-Foulds (RF) distance, which only takes into account tree topology. The alignments for which trees are inferred, were simulated under the LG+GC sequence model and are all 500 amino acids long. For each measure, we show 95% confidence intervals estimated with 1000 bootstrap samples.

Fig. 2.4b&c stratify the reconstruction error in terms of their topology (panel c, using the normalized Robinson-Foulds (RF) metric [57]) and pairwise distances (panel b, using the Mean Absolute Error (MAE) between true and estimated distances). Regardless of the criterion, Phyloformer dominated

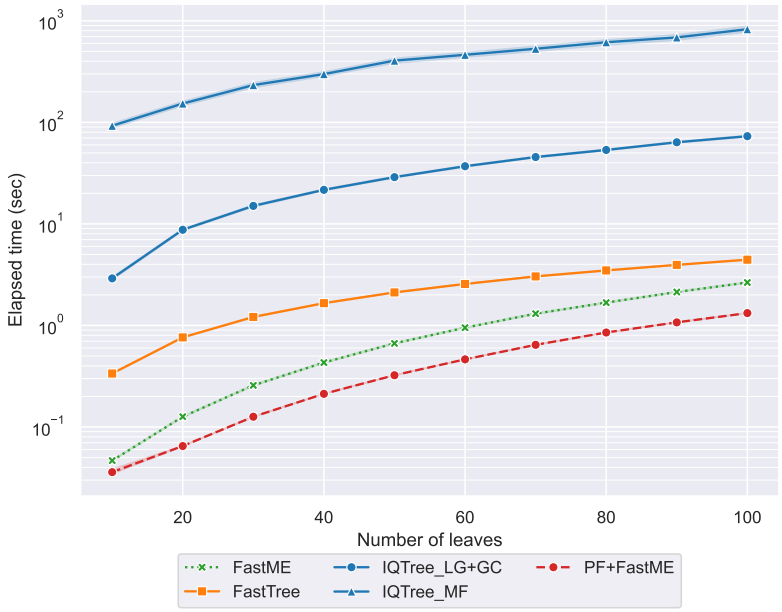


FIGURE 2.5: Execution time for different tree reconstruction methods on the LG+GC test set with alignments of length 500. For IQTree ModelFinder (MF) times were measured on the Cherry testing set (see Section 2.3.3). For all methods except Phyloformer, total wall time was measured. For Phyloformer, the elapsed time is the sum of the time it takes to infer the distances and the time FastME takes to infer the tree from these distances. It is important to note that the distance prediction time does not include the time it takes to load the Phyloformer weights to the GPU as we did that once before inferring distances for all the testing alignments.

FastME by being both faster and more accurate. On the other hand, Phyloformer reconstructed topologies that were less accurate than ML methods, and increasingly so for larger numbers of leaves, but estimated distances as or more accurately. A possible explanation for this discrepancy is that since we control the tree diameter in our simulation, larger trees have shorter branches on average. As branch lengths decrease, the number of mis-predicted branches increases leading to larger topological errors (see Supplementary Results 2.7.3 for an in-depth explanation).

Finally, we investigated the ability of Phyloformer to handle gaps contained in empirical MSAs because of insertion-deletion (indel) events that have occurred during sequence evolution. Standard models of sequence evolution consider gaps as wildcard 'X' characters, and thus cannot benefit from the information they provide. Models that account for insertion-deletion processes are more complicated to implement and more costly to run [157], but can easily be included using our paradigm. We fine-tuned the Phyloformer network previously trained on ungapped LG+GC data on a smaller dataset that includes indels, inserted through a model of insertion/deletion events in Alisim [21], choosing parameters as in [158]. Fig. 2.6 shows that the accuracy of all methods dropped on alignments that include gaps compared to alignments that do not (Fig. 2.4), probably because gaps remove information from the alignments. However the difference between Phyloformer and ML methods shrinked, with Phyloformer outperforming ML methods according to the RF metric for 10 to 30-leaf trees. This is likely due to Phyloformer's ability to extract information from gaps, which are encoded as a separate character and not as a wildcard character.

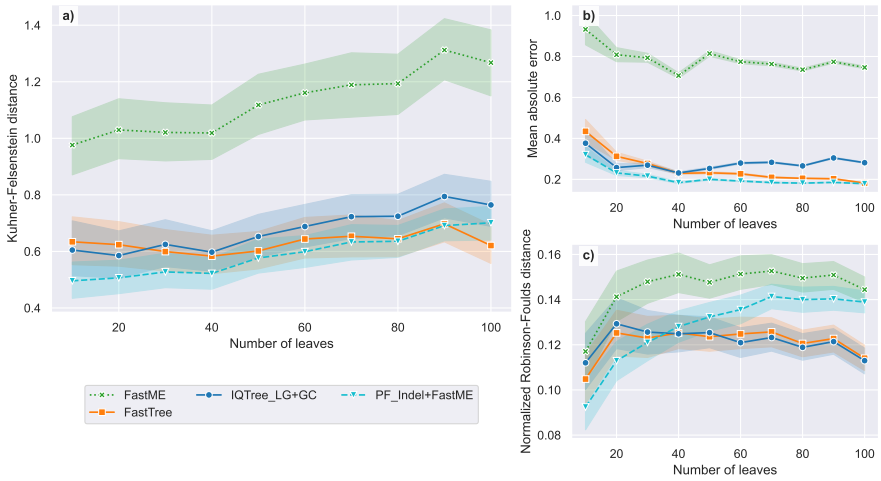


FIGURE 2.6: Tree comparison metrics for different tree reconstruction methods on the LG+GC+indels test set (alignment length=500). Legend as in Fig. 2.4, with Phyloformer fine-tuned on alignments with gaps named PF_{Indel}+FastME and in cyan.

2.3.3 Under more realistic models, Phyloformer outperforms all other inference methods

Because ML and Bayesian inference approaches must compute the likelihood, in practice they can only be used under simple models such as LG+GC for which these likelihood calculations are affordable. Phyloformer on the other hand can reconstruct phylogenies under arbitrarily complex models of sequence evolution, as long as we can efficiently sample training data from these models. We now illustrate this feature by considering inference tasks under two substitution models that relax common simplifying assumptions: independence between sites, and the homogeneity of selective constraints across sites. The first model we used (Cherry, Supplementary Methods 2.6.2) is derived from a model of sequence evolution that includes pairwise amino-acid interactions [159]. ML inference under such a model would be very costly for two reasons: the substitution matrix has size 400×400, and would need to be applied to pairs of interacting sites, which would need to be identified with additional computations. The second model (SelReg, Supplementary Methods 2.6.2) draws different selective regimes for each site of the alignment: a site can evolve under neutral evolution, negative selection, or persistent positive selection. ML inference under such a model is achievable with a mixture model [*e.g.*, 160], but costly, because the SelReg mixture includes 263 distinct amino acid profiles, plus a profile for neutral evolution, and a different matrix for positively selected sites. We fine-tuned the Phyloformer network previously trained under the LG+GC model on alignments sampled under the Cherry or the SelReg model. We compared its performances against the same methods as before, but allowing IQTree to search for the best evolution model available (with the Model Finder option). Fig. 2.7 shows that under both the Cherry and SelReg models all methods performed worse than under LG+GC, presumably because both models decrease the information provided by a given number of sites, by including pairwise correlations (Cherry), or positively selected sites that are likely to saturate (SelReg). However, Phyloformer outperformed all other methods by a substantial margin, with KF distances around 1 whereas others range between 2 and up to 10 for IQTree under

SelReg. Of note, the Model Finder option was costly, further increasing the computational edge of Phyloformer (Fig. 2.5). Not using this option markedly decreased the accuracy of IQTree on the Cherry alignments (Supplementary Figs. 2.19 and 2.20). As we observed under LG+GC, Phyloformer was better at estimating distances than topologies (Fig. 2.7), with the latter becoming more challenging for larger numbers of leaves. Nonetheless, the RF distances of Phyloformer remained lower than for other methods, except for SelReg on trees with more than 40 leaves where it was only outperformed by IQTree—which in turn had the worst distance estimates among all benchmarked methods.

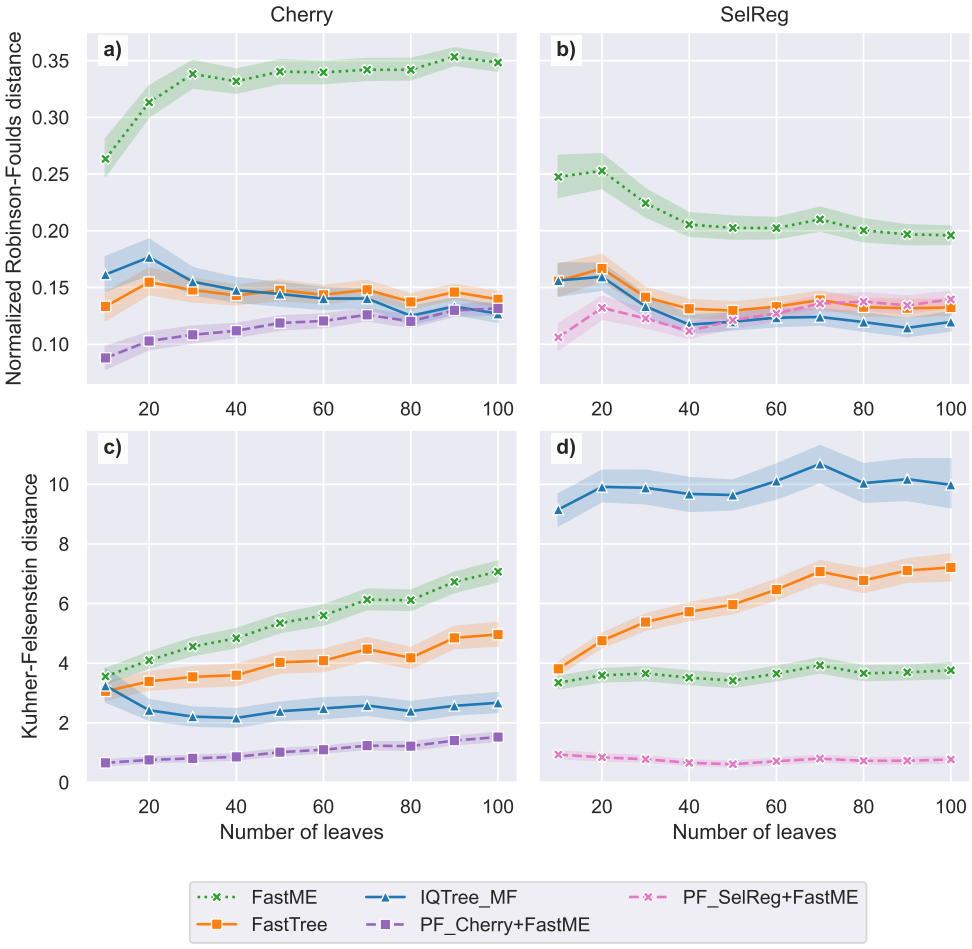


FIGURE 2.7: Normalized Robinson-Foulds distance (above) and Kuhner-Felsenstein distance (below) for different tree reconstruction methods on the Cherry (left) and SelReg (right) test sets (alignment length=500).

2.3.4 Phyloformer performs on par with ML methods on empirical data

We compared the performance of Phyloformer and other methods on 346 orthologous gene alignments from 36 Cyanobacteria [161], reasoning that good reconstruction methods should more often infer trees that match the tree obtained on the concatenated gene alignments. We compared the LG+GC-with-indel version of Phyloformer to the same three methods assessed in section 2.3.2. Fig. 2.8a shows that Phyloformer performed as well as the other standard methods on empirical data, and did so faster.

We conducted a similar analysis on gene-trees over many different species and orders obtained from [20]. In this study the authors collected a large number of sequence datasets and inferred gene-trees using IQTree and FastTree under the evolutionary model found by IQTree's ModelFinder for each alignment. For IQTree they inferred 10 trees and only kept the one with the best likelihood. The authors also reconstructed species trees from concatenated alignments for each dataset. We reconstructed trees on the gene families where at least 80% of alignments were classified as LG by IQTree using the LG+GC-with-indel version of Phyloformer with FastME. We then compared our gene trees as well as the ones from [20] to the concatenate trees. Here again, Fig. 2.8b shows that in most cases Phyloformer performed as well as the best of 10 trees estimated with ML methods. Here the computational speed of Phyloformer shines as we were able to infer about $12,000$ trees in under two hours with one GPU. In [20], the authors measured execution times of only 10% of tree inference tasks, for which the total runtimes of IQTree and FastTree were approximately 10.5 days and 4 hours respectively. On the same subset of trees, we measured the total runtime of Phyloformer+FastME and standalone FastME at approximately 11.5 and 15 minutes respectively. Furthermore, Phyloformer consistently produced trees with a higher likelihood than FastME trees though still lower than pure ML methods (Supplementary Fig. 2.15).

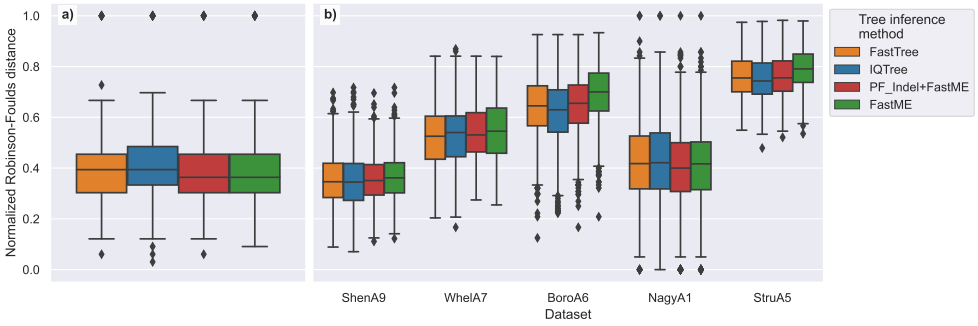


FIGURE 2.8: Comparison of topology reconstruction accuracy between Phyloformer and other methods on empirical data. In both panels, we show the normalized Robinson-Foulds distance between reconstructed gene trees and the corresponding concatenate tree. In **a)** inferred gene trees on alignments from [161] using the same pipeline as in Section 2.3.2 and with the gap-aware version of Phyloformer shown in Fig. 2.6. In **b)** gene-alignments, species trees and some gene trees were obtained from [20]. We inferred gene-trees using the gap-aware version of Phyloformer and FastME as in panel **a)**. The IQTree predictions were made in [20] under the evolutionary model found by IQTree-ModelFinder, then 10 predictions were done and only the one with the best likelihood was kept. The datasets shown here, have $\geq 80\%$ of alignments detected as LG by IQTree.

2.4 Discussion

Drawing on recent breakthroughs in likelihood-free inference and geometric deep learning, we have demonstrated that Phyloformer achieves rapid and precise phylogenetic inference. The likelihood-free paradigm only requires samples from the probabilistic model of sequence evolution, which allows inference under much more complex models than ML or Bayesian inference. Furthermore we exploited an amortized form of this paradigm, requiring a

single training of a neural network that takes an MSA as input and outputs evolutionary distances between pairs of sequences—as opposed to approaches like ABC [117] that require a new sampling step at each inference. We based our neural network on axial self-attention, an expressive mechanism that accounts for the symmetries of the MSA and seamlessly handles arbitrary numbers of sequences of any length.

Phyloformer was faster and as accurate as ML inference methods on data sampled under the standard LG+GC model. Computing likelihoods under LG+GC is expensive but possible, making ML inference the gold standard: reaching the same accuracy faster was the best outcome one could hope for. On the other hand, computing likelihoods under more complex models accounting for local dependencies (Cherry) or heterogeneous selective pressures (SelReg) is too costly, forcing ML methods to work under misspecified models whereas Phyloformer can still perform inference under the correct model, without any effect on its speed. As a result, Phyloformer yields the most accurate inference by a substantial margin while retaining its computational edge.

More generally, we stress that likelihood-free inference using neural networks has a model-based nature identical to ML or Bayesian methods. It formally estimates the posterior distribution defined by the prior and probabilistic model used to simulate training data, accessing this model through sampling instead of likelihood evaluations. As such, it is not immune to model misspecification: we observed for example that Phyloformer trained on LG+GC underperformed on data simulated under Cherry or SelReg and vice-versa (Supplementary Figs. 2.19, 2.20 and 2.28). Rather than replacing model choice, we believe that the crucial contribution of a likelihood-free method like Phyloformer is to offer a way to work under more realistic models of sequence evolution that were so far not amenable to inference.

It is noteworthy that the inference speed that we report for Phyloformer was recorded on a GPU, a less widespread hardware than the CPU used for other methods, which may limit its interest for analysing a single gene alignment under models amenable to ML. However, we expect Phyloformer to have a significant impact in experiments where many reconstructions are necessary, *e.g.* for bootstrapping, reconstructing several gene trees from whole genomes or transcriptomes, or where more complex models are warranted. Another current limitation of Phyloformer is its scalability. The current bottleneck is on its memory usage, mostly driven by applying self-attention to pairs of sequences. A better scaling version could be obtained by working at the sequence level— attempts to do so have so far led to lower accuracies.

An important extension of Phyloformer will be to train with a topological loss function, *e.g.* directly minimizing the RF metric rather than a distance metric. Such a version would address the gap that we observed between accuracies in distance and topological reconstruction, and could also lead to a more scalable method by working around the need for all pairwise distances—of quadratic size in the number of sequences whereas the tree itself has linear numbers of nodes and edges. We also believe that extending Phyloformer to unaligned sequences will be of interest, both because multiple alignments are computationally intensive, and because they are error-prone. This could be addressed by including the alignment step in the network [162], [163]. Alternatively, one could forego alignment altogether, *e.g.* by producing a length-independent representation early in the neural network.

We expect that Phyloformer will have its largest impact on phylogenetic inference after versions are trained on a collection of more realistic models of sequence evolution which could include nucleotides, variations along the sequence or between branches and position-specific dependencies among sites [8], [164], [165]. Our self-attention network could exploit these latter dependencies via the addition of positional encodings—a standard approach in the transformers literature. Beyond phylogenetic reconstruction, our network can be trained to infer other parameters of the simulation model. This would provide an efficient and flexible way to study phylodynamics, phylogeography, and selective pressures operating on the sequences, for instance.

2.5 Online methods

The Phyloformer neural network

Phyloformer is a parameterized function F_Φ that takes as input an MSA of n sequences of length L and outputs an estimate of the $N = \binom{n}{2}$ distances between all pairs of sequences. Φ denotes the set of learnable parameters of F_Φ. We then input these distances to FastME [46] to obtain a phylogenetic tree (Fig. 2.3).

The Phyloformer network starts with a one-hot encoding of the aligned sequences: every sequence x is represented as a matrix $\varphi^{(0)}(x) \in \{0,1\}^{22\times L}$ in which column j contains a single non-zero element $\varphi^{(0)}_{ij}(x) = 1$, whose coordinate $i \in \{1,\dots,22\}$ denotes the amino acid or gap present in sequence x at position j. It then represents each pair (x, x') of sequences in the MSA by the average of their individual representations *i.e.*, with a slight abuse of notation, $\varphi^{(0)}(x,x') = \frac{1}{2}\left(\varphi^{(0)}(x) + \varphi^{(0)}(x')\right)$. Of note, $\varphi^{(0)}(x,x')$ does not depend on the order of sequences x and x'. At this stage, the network represents each site within each pair independently of all others, encoding information such as "at site 4, sequences x and x' contain a Leucine and an Isoleucine". The whole purpose of F_ϕ is to account for relevant information about the evolutionary distance between x and x' contained in other sequences from the alignment. To extract this information, F_ϕ uses $r = 6$ self-attention layers [73] that iteratively build updated $\varphi^{(l)}(x,x') \in d \times L$ representations of each pair using all others in the MSA. More precisely, we use axial attention [145, Fig. 2.3, central panel] and successively update each pair (resp. site) separately by sharing information across sites (resp. pairs). Along each axis, we rely on a modified linear attention [166, see Scalable self-attention], with $h = 4$ attention heads and embeddings of dimension 64 for the value matrix and only 1 for the query and key matrices. The r axial attention blocks of Phyloformer output for every pair of sequences a tensor $\varphi^{(r)}(x,x') \in \mathbb{R}^{d\times L}$ informed by all other pairs in the same MSA. We convert this representation into a single estimate of the evolutionary distance between x and x' by applying an $\mathbb{R}^d \to \mathbb{R}$ fully connected layer to each site of each pair, followed by an average over the sites. We provide more details on the F_Φ architecture in Supplementary Section 2.6.3.

Accounting for symmetries

It is now well understood that accounting for known symmetries is key to the success of deep learning, as formalized in geometric deep learning [167].

Following this principle, we parameterize the function F_Φ by a neural network that exploits two symmetries of the estimation task: the estimated evolutionary distances should not depend on the order of the n sequences or L sites in the MSA. More precisely, we want F_Φ to be equivariant by permutations of the sequences: if it returns values d_{ab}, d_{ac}, d_{bc} when presented with sequences (a, b, c), it should return d_{ac}, d_{bc}, d_{ab} when given (c, a, b) as input. On the other hand, F_Φ should be invariant to permutations of the sites—any such permutation should lead to the same F_Φ distances. This last point may seem counterintuitive as the order of residues in a protein matters for its function, and it is known that close residues do not evolve independently. Nonetheless in all our experiments we train—or pre-train—F_Φ on data generated under the LG+GC model, which is site-independent. The self-attention updates act on the $\mathbb{R}^d$ representations of a site within a pair of sequences regardless of their order, yielding the desired equivariances. Enforcing these equivariances would be more difficult if the updates were general functions acting on entire MSAs represented by $\mathbb{R}^{d\times N\times L}$ tensors. The final average across sites within each pair makes F_Φ invariant rather than equivariant by permutation of these sites. In addition because none of the operations in F_Φ depend on the number of sites or pairs, we can use the same F_Φ seamlessly on MSAs with an arbitrary number of sequences of arbitrary length.

Scalable self-attention

Naive implementations of self-attention over M elements scale quadratically in M—in our case, both the number of sites and pairs of sequences. Indeed, softmax attention as introduced by [73] is parameterized by three matrices $Q, K, V \in \mathbb{R}^{M\times d}$ for some embedding dimension d, respectively called Queries, Keys and Values, and every update for an element i computes attention weights $(s_{i,1}, \dots, s_{i,M}) = \text{softmax}\left(\frac{q_i^\top K}{\sqrt{d}}\right)$. We resorted to the linear attention of [166], who exploited the fact that $s_{ij} = \frac{\langle\phi(q_i),\phi(k_j)\rangle}{\sum_{h=1}^{M}\langle\phi(q_i),\phi(k_h)\rangle}$ for some non-linear infinite-dimensional mapping $\phi : \mathbb{R}^d \to \mathcal{H}$ to a Hilbert space $\mathcal{H}$ [168] and proposed to replace ϕ by some other non-linear, finite-dimensional mappings $\tilde{\phi} : \mathbb{R}^d \to \mathbb{R}^t$. We can then re-write the self-attention updates $z'_i = \sum_{j=1}^{M} s_{i,j} v_j$ as

$$z'_i = \frac{\sum_{j=1}^{M}\tilde{\phi}(q_i)^\top\tilde{\phi}(k_j)v_j}{\sum_{h=1}^{M}\tilde{\phi}(q_i)^\top\tilde{\phi}(k_h)} = \frac{\tilde{\phi}(q_i)^\top\sum_{j=1}^{M}\tilde{\phi}(k_j)v_j}{\tilde{\phi}(q_i)^\top\sum_{h=1}^{M}\tilde{\phi}(k_h)}. \tag{2.1}$$

Because we can pre-compute each of the two sums and re-use it for every query, this simple factorization reduces both the number of operations and memory usage from $\mathcal{O}(M^2 \cdot L \cdot d)$ to $\mathcal{O}(M \cdot L \cdot d \cdot t)$. Following [166] we used an ELU-based mapping [169]

$$\tilde{\phi}(x) = \begin{cases} x + 1, & \text{if } x > 0 \\ \exp\{(x)\} & \text{if } x \leq 0, \end{cases}$$

where the operation is applied entrywise, yielding $\tilde{\phi}(x) \in \mathbb{R}^d$ vectors for $x \in \mathbb{R}^d$. In our experiments, we used $d = 64$ for the Values matrix, but noticed that using $d = 1$ for Queries and Keys led to slightly lower training-loss values (Supplementary Fig. 2.27a), while substantially reducing the memory footprint of the self-attention layers (Supplementary Fig. 2.27b). This observation is

consistent with recent research showing that Transformers and other neural networks learn through gradual rank increase [170], [171]. However, applying (2.1) with queries and keys of dimension 1 leads to identical updates z'_i for all elements. To work around this issue, we normalized each update by the average of queries and the sum of keys instead of the usual sum of attention weights, leading to

$$z'_i = \frac{\tilde{\phi}(q_i)}{M^{-1}\sum_{g=1}^{M}\tilde{\phi}(q_g)} \cdot \frac{\sum_{j=1}^{M}\tilde{\phi}(k_j)v_j}{\sum_{h=1}^{M}\tilde{\phi}(k_h)}. \tag{2.2}$$

Training Phyloformer

We trained F_Φ using 6 NVIDIA A100 80GB GPUs on simulated examples through a loss function (see Metrics) comparing the estimated and true evolutionary distance (Fig. 2.3). We used the Adam optimizer [172], batches of size 4 and a maximum learning rate of 10^{-3} with 3000 linear warmup steps followed by a linear decrease of 213,270 steps, corresponding to 30 epochs. We also implemented an early stopping criterion that stopped training when the validation loss did not decrease over 5 successive 3000 step intervals.

We first trained an F_Φ^{pre} function that served as a starting point for all the functions used in our experiments, by optimizing Φ with respect to the MAE loss for 20 epochs ($\approx$ 79 hours) over the 170,616 examples (see Section 2.5) simulated under LG+GC, saving a model every 3000 steps, and eventually retaining the one with lowest Robinson-Foulds error (see Metrics) over the validation dataset (17016 examples). For the results in Fig. 2.4, we further optimized the parameters of F_Φ^{pre} for 4 epochs (20 hours) with respect to the MRE loss leading to a slightly improved error over small distances (Supplementary Fig. 2.24) and on the overall Robinson-Foulds metric (Supplementary Fig. 2.18). For the results in Figs. 2.6 and 2.7 we further optimized the parameters of F_Φ^{pre} for the MAE loss on gapped MSAs and MSAs generated under the Cherry or SelReg substitution models respectively (see Datasets).

Baselines

IQTree LG+GC [19, p. v2.2.0] reconstructs phylogenies in the Maximum Likelihood framework. It first estimates several parsimony trees along with one reconstructed through a distance method, then optimizes branch lengths and other parameters of the model of sequence evolution, while performing local topological rearrangements (Nearest Neighbor Interchanges, NNIs) to maximize the likelihood. We ran it with the LG model of amino acid substitution [146] combined with a continuous gamma distribution to model rate heterogeneity across site [14]. In our experiments we did 5 rounds of NNIs since we observed that optimizing for more rounds rarely improved the topology of the final tree while substantially adding to the running time. The software was run with `iqtree2 -T 1 -m LG+GC -n 5`.

IQTree MF uses the Model finder (MF) mode of IQTree [173], in which likelihoods of an initial tree are computed for a large set of substitution models and models of rate-heterogeneity accross sites. The best fitting model is selected using BIC. The rest of the tree search is done as above but using the selected model for likelihood estimations. The software was run with `iqtree2 -T 1 -n 5`.

FastTree [45, v2.1.11 SSE3] reconstructs a starting tree using an algorithm inspired from Neighbor-Joining [24] which is subsequently refined with topological rearrangements to optimize the minimum evolution criterion. The tree is then improved using maximum likelihood with NNIs. It was run under the LG+G4 model of sequence evolution. The software was run with `fasttree -lg -gamma`.

FastME [46, p. v2.1.6.4] computes a distance matrix using Maximum Likelihood, then reconstructs a tree topology using BioNJ [31] and further refines it via topological rearrangements which seek to optimize the Balanced Minimum Evolution score. In virtually all performed experiments we observed that the FastME tree search algorithm led to slightly better performances than the neighbor joining algorithm [24]. We didn't resort to the `--gamma` option as in our experiments we observed that this lead to worse performances. Using FastME as our baseline distance method makes the comparison with Phyloformer insightful, as the only difference between the two methods is the distance matrix used as input. The software was run with `fastme --nni --spr --protein=LG` to reconstruct trees using the inbuilt evolutionary distance estimation and simply with `fastme --nni --spr` when Phyloformer's predicted distance matrix was provided.

All methods were run on a single CPU thread (Intel Xeon E5-2660 2.20GHz) except for Phyloformer distance prediction which was run on a single GPU (NVIDIA V100 32GB).

Datasets

We generated ultrametric phylogenies under a birth-death process. We used 50-leaf trees for training, and 10-leaf to 100-leaf trees for testing. We rescaled branch lengths as in [154] to yield non-ultrametric trees. Finally, we rescaled each tree to resemble trees found in public empirical databases. We used each rescaled phylogeny to simulate one MSA with AliSim [21] for the LG+GC model, or in-house code for Cherry, or Pastek [174] for SelReg. For LG+GC, we sampled the parameter of the gamma distribution to match values estimated on empirical data. We provide more details in Supplementary Methods 2.6.

Metrics

We now describe the metrics used throughout this article to compare phylogenies or optimize our network.

Let d_i be the i^{th} of N true evolutionary distances in a phylogeny, and $\hat{d}_i$ the corresponding estimate output by a given tree inference method. Then the mean absolute error (MAE) and mean relative error (MRE) are defined as

$$\ell_{MAE} = \frac{1}{N}\sum_{i=1}^{N} |d_i - \hat{d}_i| \qquad \text{and} \qquad \ell_{MRE} = \frac{1}{N}\sum_{i=1}^{N} \frac{|d_i - \hat{d}_i|}{d_i}.$$

When used to compute the loss during Phyloformer training, $\hat{d}_i$ values correspond to distance estimates directly output by F_Φ. When used as a metric (e.g. in Fig. 2.4) we use $\hat{d}_i$ values extracted from the reconstructed tree, by summing all branch lengths on the paths between each pair of leaves—even for Phyloformer— in order to fairly compare different methods.

In phylogenetic trees, each branch describes a bipartition of the set of leaves, paired with a weight (*i.e.*, the branch length). Let A and B be the sets of leaf-bipartitions describing trees T_A and T_B, and $w_{e,T}$ the weight of a bipartition e in tree T. Then, the Normalized Robinson-Foulds distances and the Kuhner-Felsenstein distance between T_A and T_B can be written

$$RF_{norm}(T_A, T_B) = \left(|A| + |B|\right)^{-1} \left(|A \cup B| - |A \cap B|\right)$$
$$\text{and } KF(T_A, T_B)^2 = \sum_{e \in A \cap B} (w_{e,T_A} - w_{e,T_B})^2 + \sum_{e \in A \setminus B} w_{e,T_A}^2 + \sum_{e \in B \setminus A} w_{e,T_B}^2.$$

Code and data availability

The code for Phyloformer, the pretrained models, and all the datasets analyzed in this work can be found at
`https://github.com/lucanest/Phyloformer`.

Acknowledgements

The authors thank Dexiong Chen, Flora Jay, Martin Ruffel, Johanna Trost for insightful discussions.

This work was funded by the Agence Nationale de la Recherche (ANR-20-CE45-0017). It was granted access to the HPC/AI resources of IDRIS under the allocation AD011011137R1 made by GENCI. We estimate that our

computations on GPUs to experiment different architectures, and to train and test the networks have generated about 520kgs eCO2. This includes 15kgs eCO2 for training the final PF network on LG-GC. Part of this work was performed using the computing facilities of the CC LBBE/PRABI. The taxon silhouettes in Fig. 2.3 are modified from public domain images in the PhyloPic database.

2.6 Supplementary Methods

Here we provide a detailed description of our protocol to sample trees and multiple sequence alignments along these trees.

2.6.1 Simulating phylogenies

We chose our simulation parameters to generate trees similar to empirical tree distributions.

Empirical tree distributions

We collected 63,245 phylogenies from the HOGENOM and 4,893 phylogenies from the RaxMLGrove databases. In order to have the same order of magnitude across datasets we multiplied by 10 the frequency of the latter in the distribution. We furthermore kept all the trees with diameters in the range $(0.02, 15)$, discarding 9,055 trees out of 112,175. The trees sampled from the HOGENOM database correspond to the rooted "nocore" trees from the phyla Delta-Epsilon (7,687 trees), Alpha-proteobacteria (14,125 trees), Beta-proteobacteria (11,961 trees), Tenericutes (1,634 trees), Archaea (7,658 trees), Spirochaetes (3,600 trees), Cyanobacteria (4,898 trees) and Bacteroidetes-Chlorobi (11,682 trees). The trees sampled from RaxMLGrove correspond to all the trees reconstructed from MSAs of amino acid sequences.

Simulating trees

We trained all neural networks presented in our work on trees/alignments with 50 leaves/sequences, and considered smaller and larger numbers for testing. We followed the same procedure for all trees. Following [154] we simulated each tree using a birth-death process which returns an ultrametric tree using dendropy's `treesim.birth_death_tree` method [175, p. v4.6.1], then obtained a non-ultrametric tree by rescaling the branches of the tree. To do so we simulated changes of the rate of evolution by running a process that generates small and big rate changes from the root of the tree to the leaves. Then we rescaled all the trees, irrespective of the number of leaves, to match the distribution of diameters (maximum pairwise distance in the tree) observed in empirical data (subsection 2.6.1). For each tree we sampled a diameter from this distribution, and added gaussian noise to provide variation ($\mu = d, \sigma = d/10$, with d the sampled diameter).

Given that all trees share the same diameter distribution irrespective of their number of leaves, trees with more leaves will have a larger number of short branches. This leads to a distribution shift between training and testing data (see Supplementary Fig. 2.17a). Moreover very short distances also lead to duplicate sequences during the alignment simulation step. To mitigate this and avoid too many alignments with duplicate sequences, we further rescaled terminal branches, resampling their length l from a normal distribution, centered around the minimum allowed valued $\mu = 0.001$, and having $\sigma = 0.005$, until we reached $l \geq \mu$. Supp. Fig. 2.9 illustrates the whole tree simulation pipeline.

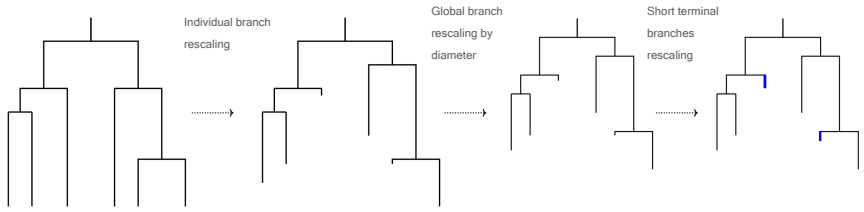


FIGURE 2.9: Tree simulation pipeline: we simulate an ultrametric tree with a birth-death process, rescaling the branches individually to simulate changes in the rates of evolution, then rescaling all the branches to match a diameter sampled from empirical data, and finally rescaling terminal branches which are too short to make them longer than the minimum allowed value.

Comparison of simulated and empirical trees

To make sure the simulations were realistic enough we computed several statistics of the training trees: mean distance between leaves, standard deviation of distances, maximum distance, mean branch length, standard deviation of branch lengths, maximum branch length, mean root to leaf distances, standard deviation of root to leaf distances. We compared these statistics to those of the trees extracted from the empirical databases. The distributions of the statistics were markedly different between the two databases (Supplementary Fig. 2.10) and we chose simulation parameters to overlap with both.

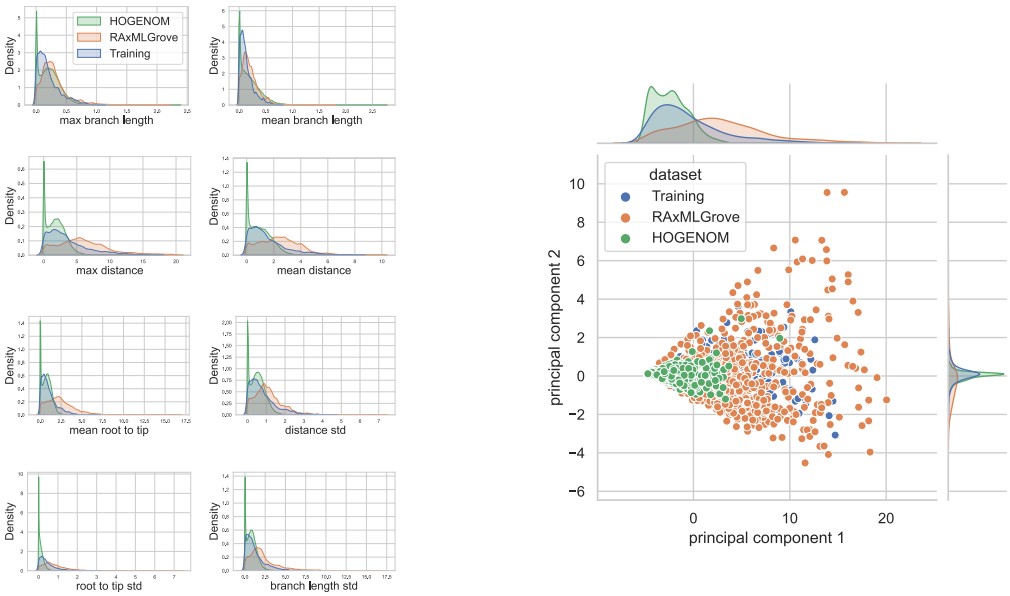


FIGURE 2.10: The distribution of trees used to train and test our networks is similar to the empirical tree distributions. Left: Distributions of the considered tree statistics for the empirical tree datasets and the training simulations. The blue curves correspond to simulated trees, the orange curves to RAxMLGrove trees, and the green curves to HOGENOM trees. Right: Two component PCA based on all 8 statistics for the three datasets (1000 points per dataset shown).

2.6.2 Simulating multiple sequence alignments

In the following we describe how we simulated alignments along the generated phylogenies. Given that the whole tree+alignment simulation pipeline still leads to some alignments having duplicate sequences, we simulate a larger number of data points than we need and discard alignments with duplicated sequences and the corresponding trees.

LG+GC We generated alignments under the LG+GC model of evolution along the trees with the Alisim software implemented in IQTree2. We sampled the α parameters of the continuous gamma distribution from those inferred by IQTree on $12,408$ alignments from the HOGENOM core database, and added gaussian noise to ensure variation ($\mu = \alpha, \sigma = \alpha/10$, with α the sampled value). A threshold of 0.05 was set for the minimum allowed value.

LG+GC+Indels We simulated alignments as above but with an additional step for the simulation of indels. We used the simulation procedure of [158], who retrieved indel-specific parameters for the rich indel model (RIM) [176] from empirical MSAs in the TreeBASE database [177]. We limited the sequence length for the indel simulation to be in $(500, 4000)$ and cropped the alignments to have a sequence length $L = 500$.

SelReg The SelReg model of evolution accounts for different selective regimes for each site of the alignment. It is a codon based Mutation-Selection model [178] in which each codon site is associated to one of 263 empirically assessed profiles [174] which store the fitness of each possible amino acid occurring at the site. The rate of substitution of a codon into another is stored in a 61×61 matrix and depends upon the fitness profiles, and a 4×4 matrix of mutation rates between A, C, G, T [174]. Under this model, a site can evolve under neutral evolution, when all amino acids have the same fitness, under negative selection, in which case one or several amino acids have higher fitnesses and are therefore selected at the given position, or under persistent positive selection which leads the fitness profile of a site to continuously change along the phylogeny to adapt to an ever-changing selective pressure [179]. Both for training and testing we used the Pastek simulator [174] to evolve alignments along the simulated phylogenies, sampling each site's selective regime with probabilities of 25%, 50% and 25% respectively.

Cherry Cherry is a model of sequence evolution that represents pairwise amino-acid interactions by using a 400×400 transition matrix inferred from 15,051 Pfam MSAs with associated structure data which was used to determine contacting sites [159]. We used the Cherry model to simulate evolution along the training and testing phylogenies. The resulting MSAs, of length $L = 500$ contain 250 pairs of adjacent coevolving sites. Although in these simulations the coevolving sites are side by side it is important to note that this does not affect in any way the presented results as none of the considered methods exploit positional information for the reconstruction of the phylogeny: the distance and ML methods assume that sites evolve independently of each other, and our network, which does not use positional encoding, is invariant to permutations of the columns in the input MSA.

2.6.3 Detailed architecture of the Phyloformer network, training and fine-tuning

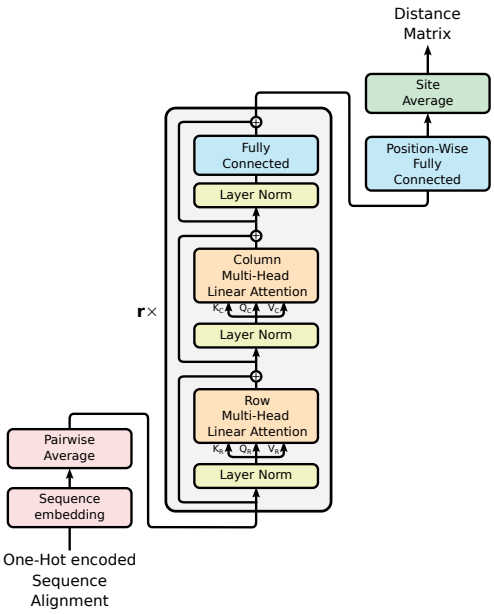


FIGURE 2.11: Network architecture of Phyloformer. We embed the one-hot encoded amino acids (a vector in $\mathbb{R}^{22}$ for each position-sequence pair in the input alignment) in $\mathbb{R}^d$ via a position-wise fully connected layer. We then take pairwise averages over sequences to obtain a representation in $\mathbb{R}^{d\times N\times L}$ of all the pairs of sequences in the alignment. Up to here each encoded amino acid is processed independently. The subsequent $r = 6$ axial attention blocks then allow them to interact building a context aware representation. After these multiple blocks we add a last position-wise fully connected layer with a softplus activation and a single output feature. Finally averaging over sites gives us the final network prediction.

The neural network architecture of the Phyloformer model is essentially that of an encoder-only transformer with Pre-Layer normalization [73], [180], with 6 attention blocks, an embedding dimension of $d = 64$, and 4 attention heads for a total number of $308,449$ trainable parameters. The network takes as input the one-hot encoded representation of an MSA as a tensor in $\mathbb{R}^{22\times n\times L}$ and transforms it through a position-wise fully connected layer which acts identically on each amino acid representation to embed it into $\mathbb{R}^d$. We then take pairwise averages over the rows (corresponding to the sequences in the MSA) of the resulting tensor in $\mathbb{R}^{d\times n\times L}$ to obtain one in $\mathbb{R}^{d\times N\times L}$, with $N = \binom{n}{2}$, having as rows the representations of each pair of sequences from which the network will eventually predict evolutionary distances. Subsequently the tensor is processed through 6 attention blocks. We resort here to the axial attention paradigm [148], which has been employed to deal with MSA data [145], and which allows passing information across positions in two steps. First through an attention mechanism applied to each row, allowing to share information across all sites in a given pair of sequences. Second through an attention mechanism applied to each column, permitting the same information flow across all pairs of sequences at a given site. We include one LayerNorm normalization layer before and one skip connection layer after each of the row-wise, column-wise attention and fully connected layers (Fig. 2.11). As standard practice in the transformer literature, we use, for the final position-wise fully connected neural network inside each block, a hidden dimension $d_h = 4 \times d$, using a GELU activation function [181] following [145]. We deviate from the implementation in [145] by allowing for independent attention maps for each row, and by using the novel variant of the linear attention mechanism [166] described in section 2.5 of the main text. At the end of the attention

Network Name	Starting Network	Batch Size	Dataset Size	Model of evolution	Effective number of Steps/Epochs	GPUs used	Target learning rate	Target schedule steps	Selected checkpoint step	Loss Function
PF_{Base}	Initialized network	4	170k	LG+GC	145.18k/20.5	6×A100	10^{-3}	213.2k	144k	MAE
PF	PF_{Base}	4	200k	LG+GC	40.3k/4.32	6×A100	10^{-4}	66k	40,3k	MRE
PF_{Indel}	PF_{Base}	1	55k	LG+GC+indels	240k/17.45	4×V100	10^{-3}	240k	136.5k	MAE
PF_{Cherry}	PF_{Base}	4	1M	Cherry	30k/0.72	6×A100	10^{-3}	66k	18k	MAE
PF_{SelReg}	PF_{Base}	4	1M	SelReg	66k/1.58	6×A100	10^{-3}	66k	66k	MAE

TABLE 2.1: Details of the training runs for all considered networks. We trained these networks with the Adam optimizer [172] and selected the best checkpoint based on the loss on a separate validation dataset. All models were trained using a linear learning rate schedule with 3000 steps of warmup followed by linear decay for a total of steps shown in the target steps column. The effective number of steps the models trained for after early stopping is also shown. We either used NVIDIA A100 GPUs with 80GB of VRAM or NVIDIA V100 GPUs with 32GB of VRAM.

blocks a last position-wise feed forward layer with a softplus activation (which ensures the positivity of the outputs) and a single output feature transforms the representation into a $\mathbb{R}^{N\times L}$ tensor, before an average over the sites gives the final network's prediction of the evolutionary distances as an $\mathbb{R}^N$ vector.

We recapitulate the training procedure for each version of Phyloformer used in our manuscript in Table 2.1, and the relationship between these different versions in Fig. 2.12.

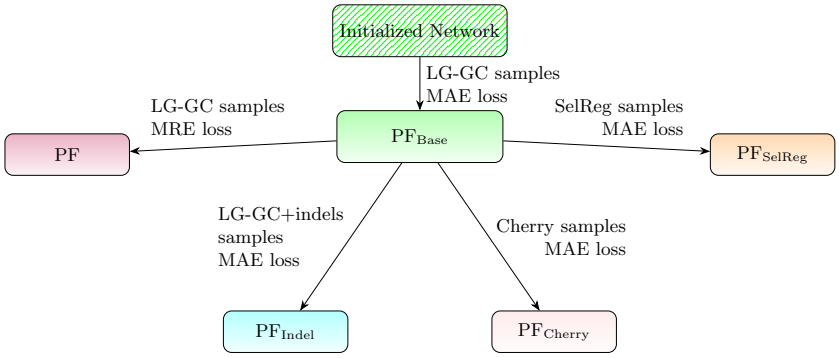


FIGURE 2.12: Trained versions of Phyloformer considered in the manuscript.

2.6.4 Estimating posterior distributions

Here we intend to provide some intuition on why our neural network trained on simulated data provides a likelihood-free estimator of the posterior distribution $p(\tau|\text{MSA})$. More precisely for our network minimizing the MAE, it provides an estimate of the median of this distribution. We consider a two-sequence alignment for which the network outputs a single scalar predicting their evolutionary distance—and the tree τ is reduced to a single branch. Using enough sampling, we could generate several $\{(\tau_i, \text{MSA})\}_{i=1}^t$ where we sampled different values τ_i of the distance from the prior $\pi(\tau)$ which all led to sampling the *same* MSA from $p(\text{MSA}|\tau)$. The set $\{\tau_i\}_{i=1}^t$ is a sample of the unnormalized

posterior distribution of $\tau|\text{MSA}$. For any particular choice of the parameters Φ, our network will produce the same output $F_\Phi(\text{MSA}) = \delta$ for all these sampled alignments. Optimizing Φ over this restricted set of samples, *i.e.*, to minimize $t^{-1}\sum_{i=1}^{t}|\delta - \tau_i|$, would therefore lead F_Φ to output a median of the $\{\tau_i\}_{i=1}^{t}$, which is identical to the median of the posterior $p(\tau|\text{MSA})$.

In this artificial example, using a parameterized network would be a clear overkill, and a single scalar would suffice. In practice however, it is unlikely that we ever sample the same MSA twice or that the MSA for which we want to do inference was even once among the samples. This justifies using a parameterized F_Φ that interpolates between the n available observations by minimizing $n^{-1}\sum_{i=1}^{n}|F_\Phi(MSA_i) - \tau_i|$ over Φ. By doing so, it produces an amortized estimator for the median of the posterior distribution of $\tau|MSA$ that borrows information among close samples.

2.7 Supplementary Results

2.7.1 Phyloformer's attention maps reveal coevolution patterns

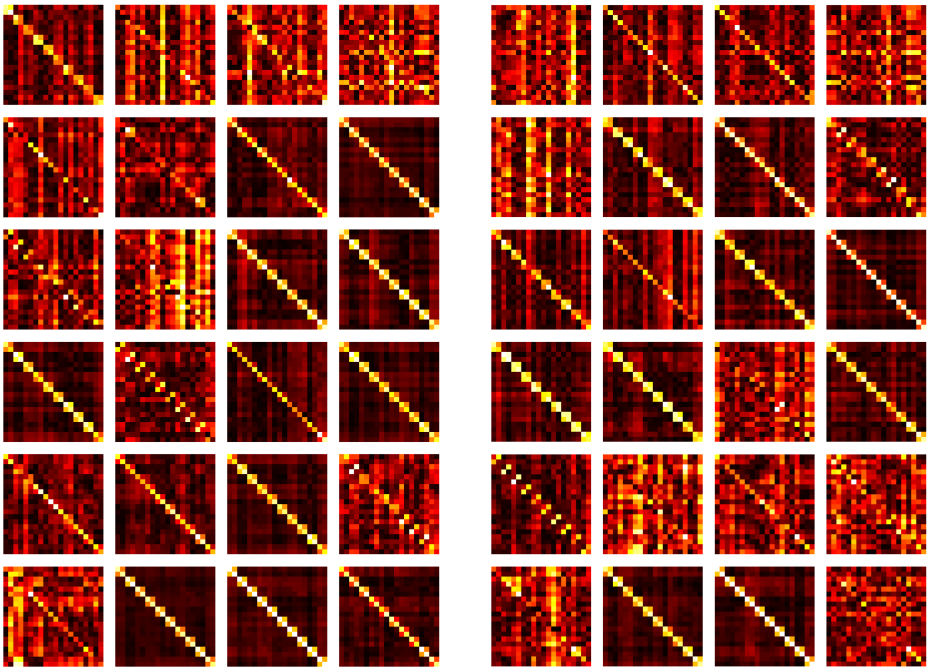


FIGURE 2.13: Row attention maps (rescaled and averaged over all alignments and sequence pairs) obtained during inference on Cherry test data with the cherry fine-tuned model (left) and the base one (right). Only the first 20 sites are shown here for ease of visualisation. Attention heads from 1 to 4 along the columns and attention blocks from 1 to 6 along the rows. The 2×2 blocks along the diagonal show that the model identifies sites that are coevolving and takes it into account for distance predictions. Interestingly the coevolution pattern can also be observed in the maps of the non fine-tuned model which has only been trained on alignments simulated without any coevolution. When the models run on the LG+GC test data without coevolution only 1×1 blocks along the diagonal can be seen (Table 2.2).

The linear attention approach that we use in this work does not compute attention maps explicitly. Nonetheless, we can obtain these maps by computing $A = \phi(Q) \cdot \phi(K)^T$. Focusing on attention across sites, which axial attention

Network on **test dataset**:	PF_{Cherry} on **Cherry**	PF_{Base} on **Cherry**	PF_{Cherry} on **LG**	PF_{Base} on **LG**
a, mean of $\bar{A}_{i,j}$ values, $\lvert i-j\rvert = 1$	0.256	0.255	0.120	0.135
b, mean of $\bar{A}_{i,j}$ values, $\lvert i-j\rvert > 1$	0.098	0.115	0.121	0.136
Ratio a/b	4.424	3.408	0.999	0.995
mean of $\bar{A}_{i,j}$ values, $i = j$	0.579	0.535	0.542	0.523

TABLE 2.2: Quantitative measure of the behaviour of the base network and the fine-tuned network on the different test datasets of 20-leaves trees. We denote here by $\bar{A}$ the rescaled attention maps averaged across all 24 attention heads (4 in each of the 6 blocks) and all alignments. Whereas the values $\bar{A}_{i,j},\ i = j$, along the diagonal, are naturally higher for all combinations of network-test dataset, only when the networks are run on data containing coevolution the $\bar{A}_{i,j},\ |i-j| > 1$, values stand out.

performs separately for each pair, we obtained $\binom{n}{2}$ maps of dimension $\mathbb{R}^{L\times L}$. Inspecting these maps can provide insight into the network's inner workings. In particular to investigate the performances of our approach in the simulations which take site coevolution into account, we computed the row attention maps (24 in total corresponding to each block and attention head) during inference on a subset (corresponding to the alignments with 20 sequences) of the test data generated both under the Cherry and LG+GC evolution model, using both the base model and the model fine-tuned on Cherry data. We averaged all $L \times L$ maps of each head and layer over all sequence pairs and further averaged over all alignments which gave us 2-dimensional matrices that can be visualized as heatmaps. To take into account both the cases where a pair of position receives particularly low or high attention score, we centered the attention maps values around 0, took their absolute value and finally normalized them to be between 0 and 1. Fig. 2.13 (left panel) shows that averaged attention maps obtained on data simulated with pairs of coevolving sites contain high values in 2×2 blocks along the diagonal. Remarkably we observed the same pattern on attention maps obtained on the same Cherry alignments with the network that we trained on LG+GC alignments, where each site evolved independently (Fig. 2.13, right panel). Our understanding is that both versions of the neural network have learned to share information between correlated sites, regardless of their position (again, no positional information is available to our network). The block pattern that we observe is just a consequence of coevolving sites being near each others in the Cherry alignments over which we compute these attention maps.

This can explain why PF_{Base}'s performances appear to be less affected compared to FastME and FastTree although all methods are subject to the same extent of model misspecification. Similarly this could be the reason why the base model still achieves state of the art performance across all metrics for smaller trees (Fig. 2.19).

On the other hand, as expected, we do not observe such pattern when running either version of the network on the LG+GC test data (Table 2.2) except for naturally occurring high values along the main diagonal.

2.7.2 Phyloformer reconstructs likely trees

While working under a model amenable to likelihood calculations we can also compare the performances of the different methods in terms of the likelihood of the reconstructed tree. In Fig. 2.14 we can see the ratio of the log-likelihood values (computed by IQTree on the LG+GC test dataset) of the true and the predicted tree for the different methods. Higher likelihood trees lead to lower ratios. Despite the fact that our network does not explicitly optimize the tree likelihood, we observe that our method typically reconstructs trees with higher likelihoods than the true ones, as observed with the full scale ML approaches. This highlights the difference with the standard distance-based method which shows the opposite trend. Similarly when testing the different approaches on the empirical data one observe that the trees reconstructed by Phyloformer consistently have higher likelihoods than those inferred by FastME (Fig. 2.15, where for each tree we plot the likelihood of the one inferred by the different methods divided by that of the one inferred by IQ-TREE10 for comparison).

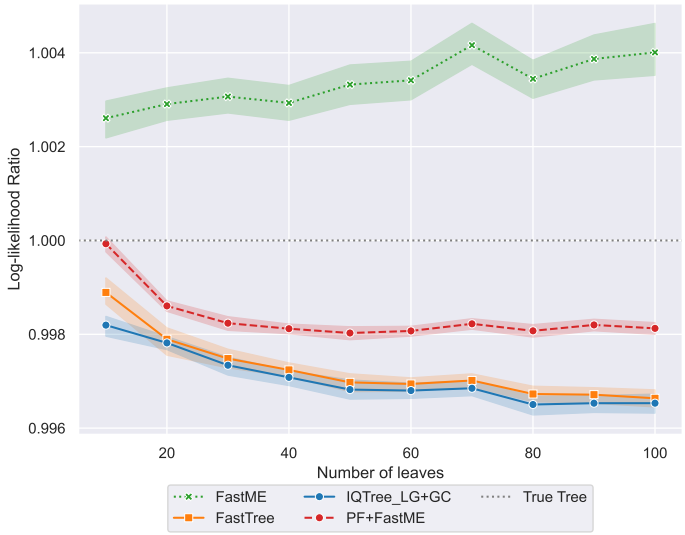


FIGURE 2.14: Log-likelihood ratios ($\mathcal{L}(T_{predicted})/\mathcal{L}(T_{real})$) for trees inferred from MSAs simulated under the LG+GC model. We use the true known simulated tree compute $\mathcal{L}(T_{real})$.

2.7.3 Short branches, not short distances, explain Phyloformer's deteriorating topological performance as the number of leaves grows

Phyloformer yields larger relative errors on small pairwise distances (Figs. 2.24 and 2.26). It is natural to think that the increase in topological error along with the number of leaves might be due to a corresponding increase in smaller pairwise distances within such trees. However, Fig. 2.16 partly disproves this. Indeed the distribution of pairwise distances stays relatively stable across trees with different numbers of leaves with only a slight shift towards smaller distances for bigger trees. Somewhat paradoxically, the smallest pairwise distances are found in the trees with the lowest number of leaves, in particular 10-leaf trees. This is due to our simulation procedure where simulated (tree,alignments) pairs with duplicated sequences are discarded. In trees with fewer leaves, small distances do not necessarily imply small branches

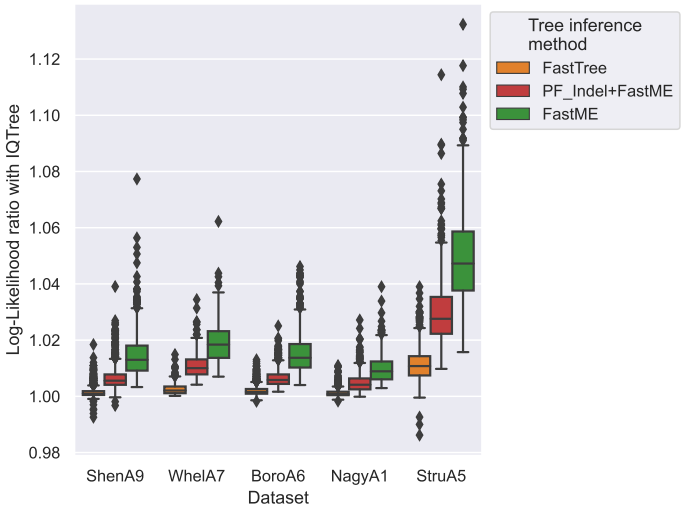


FIGURE 2.15: Log-likelihoods ratios between inferred tree and best one found by IQTree ($\mathcal{L}(T_{predicted})/\mathcal{L}(T_{IQTree})$) for the trees reconstructed on empirical alignments from [20].

therefore small distances are less likely to lead to duplicated sequences than in larger trees and therefore are more likely to be part of the testing data. This important distribution shift between 10-leaf trees and trees with more leaves most likely explains Phyloformer's relatively poor performance on these small trees.

Given that the distribution of empirical tree-diameters we sample from to rescale simulated trees is the same regardless of the simulated number of leaves (see Section 2.6), the distribution of branch lengths within the trees changes with the number of leaves. Indeed Fig. 2.17a shows that the average branch length decreases as the number of leaves increases inducing a shift in the distribution of branch lengths in our simulated testing data compared to our training data. In parallel, Fig. 2.17b shows that Phyloformer mis-predicts shorter branches, and branches of the simulated trees that are recovered in Phyloformer trees are on average ≈ 10 times longer than those that are not found in Phyloformer trees. Therefore, since (1) Phyloformer is more likely to make mistakes for shorter branches and (2) due to our rescaling of trees with empirical diameters, trees with more leaves are more likely to have shorter branches, it stands to reason that as the number of leaves grows so does the number of mis-predicted branches and consequently so does Phyloformer's topological error.

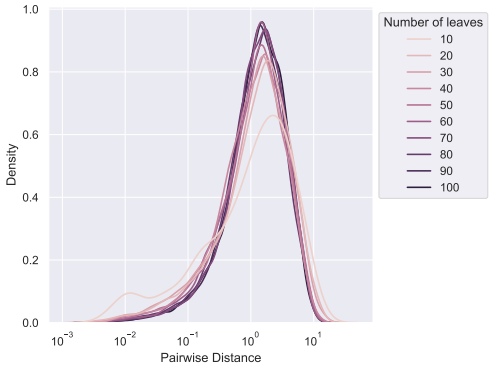


FIGURE 2.16: Distribution of pairwise distances, stratified by number of leaves in simulated testing data.

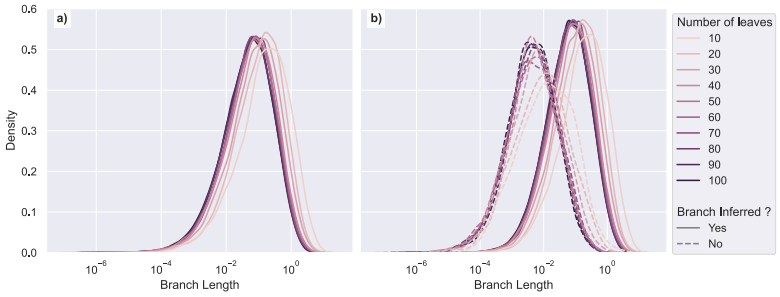


FIGURE 2.17: Distribution of branch lengths and its effect on Phyloformer performance.
a) Distribution of branch lengths in simulated test trees, stratified by number of leaves.
b) Distribution of branch lengths for branches recovered or not in Phyloformer trees, stratified by number of leaves.

2.8 Supplementary Figures

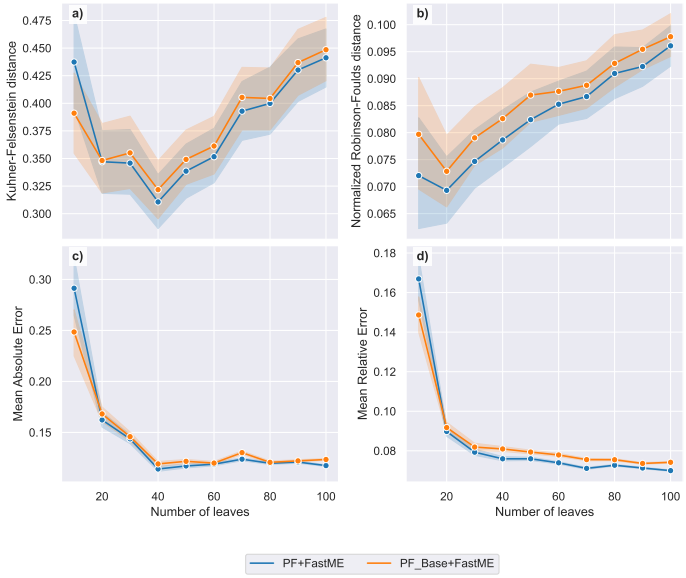


FIGURE 2.18: Performances of the base model (PF_{Base}+FastME) against the model fine-tuned with a mean relative error loss (PF+FastME). For each measure, we show 95% confidence intervals estimated with 1000 bootstrap samples.Performance is measured with: **a)** Kuhner-Felsenstein distance; **b)** normalized Robinson-Foulds distance; **c)** mean absolute error (MAE); **d)** mean relative error.

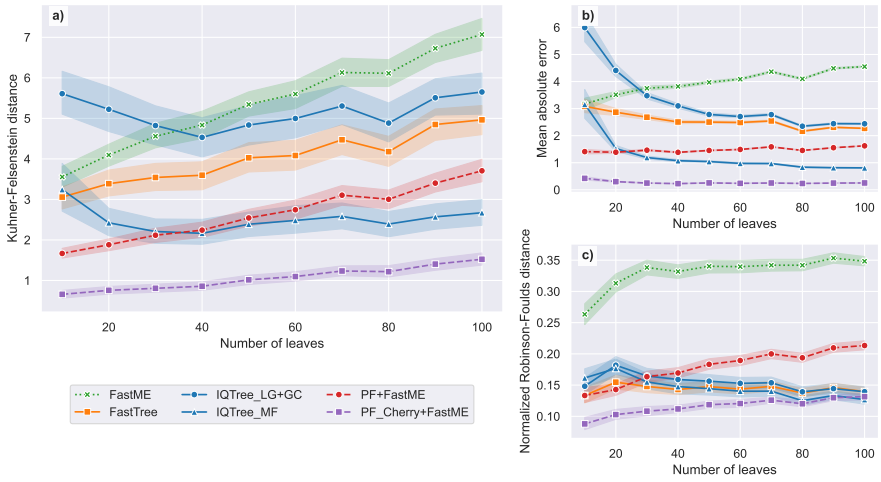


FIGURE 2.19: Performance measures for different tree reconstruction method on data simulated under the Cherry model.
a) Kuhner-Felsenstein distance **b)** mean absolute error (MAE) on pairwise distances **c)** normalized Robinson-Foulds (RF) distance. The alignments for which trees are inferred were simulated under the Cherry sequence model and are all 500 amino acids long. For each measure, we show 95% confidence intervals estimated with 1000 bootstrap samples. Trees were inferred with (1) maximum likelihood methods: IQTree with LG+GC model, IQTree with model finder option and FastTree, (2) the commonly used FastME distance method and (3) Phyloformer models either trained on LG+GC or trained on LG+GC and fine-tuned on Cherry data.

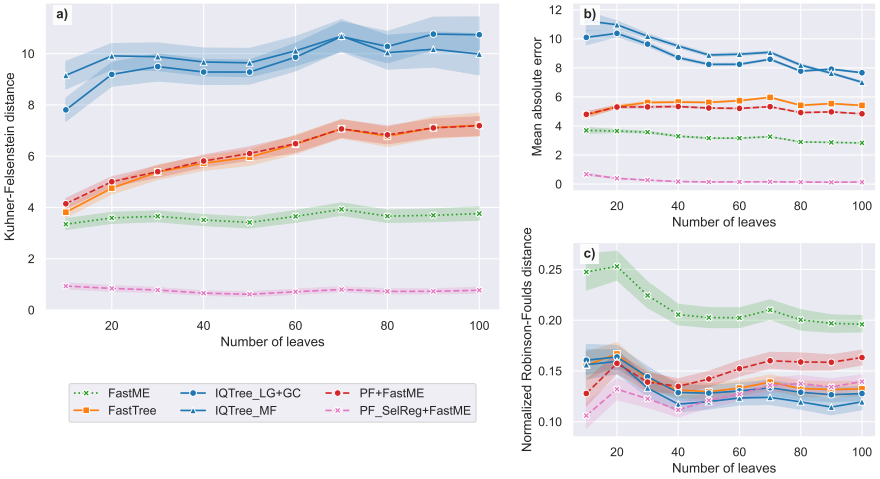


FIGURE 2.20: Performance measures for different tree reconstruction method on data simulated under the SelReg model.
a) Kuhner-Felsenstein (KF) distance; **b)** mean absolute error (MAE) on pairwise distances; **c)** normalized Robinson-Foulds (RF) distance. The alignments for which trees are inferred were simulated under the SelReg sequence model and are all 500 amino acids long. For each measure, we show 95% confidence intervals estimated with 1000 bootstrap samples. Trees were inferred with (1) maximum likelihood methods: IQTree with LG+GC model, IQTree with model finder option and FastTree, (2) the commonly used FastME distance method and (3) Phyloformer models either trained on LG+GC or trained on LG+GC and fine-tuned on SelReg data.

FIGURE 2.21: Effect of alignment length on the Kuhner-Felsenstein distance. Performance is measured on alignments simulated under the LG+GC model. 95% confidence intervals are estimated with 1000 boostrap samples.

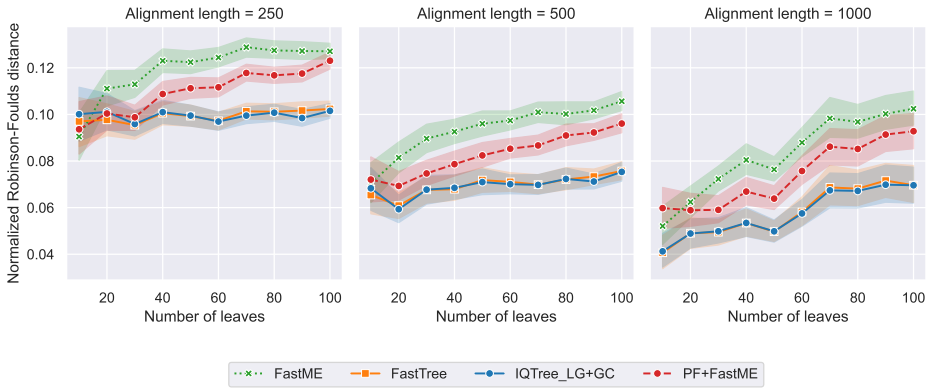


FIGURE 2.22: Effect of alignment length on the normalized Robinson-Foulds distance. Performance is measured on alignments simulated under the LG+GC model. 95% confidence intervals are estimated with 1000 boostrap samples.

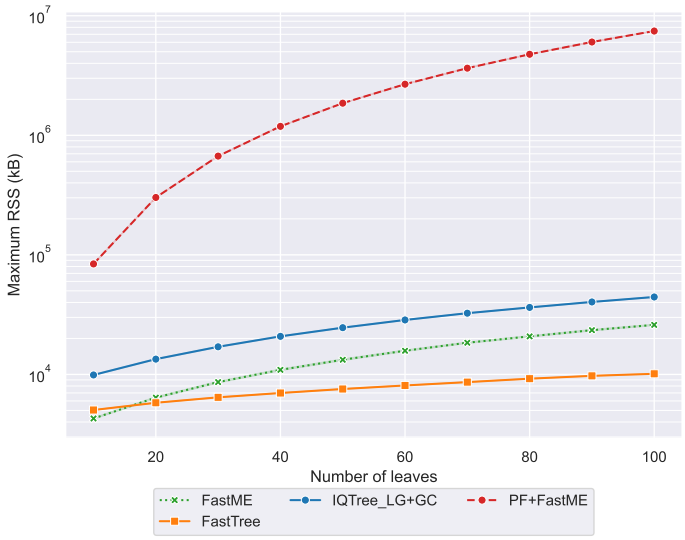


FIGURE 2.23: Memory usage during tree inference for selected tree inference methods. For methods executed on the CPU (FastME, IQTree and FastTree) the maximum resident set size (RSS), measured with the `/usr/bin/time` executable is shown. For Phyloformer, the maximum number of GPU allocated bytes added to FastME's memory usage is shown.

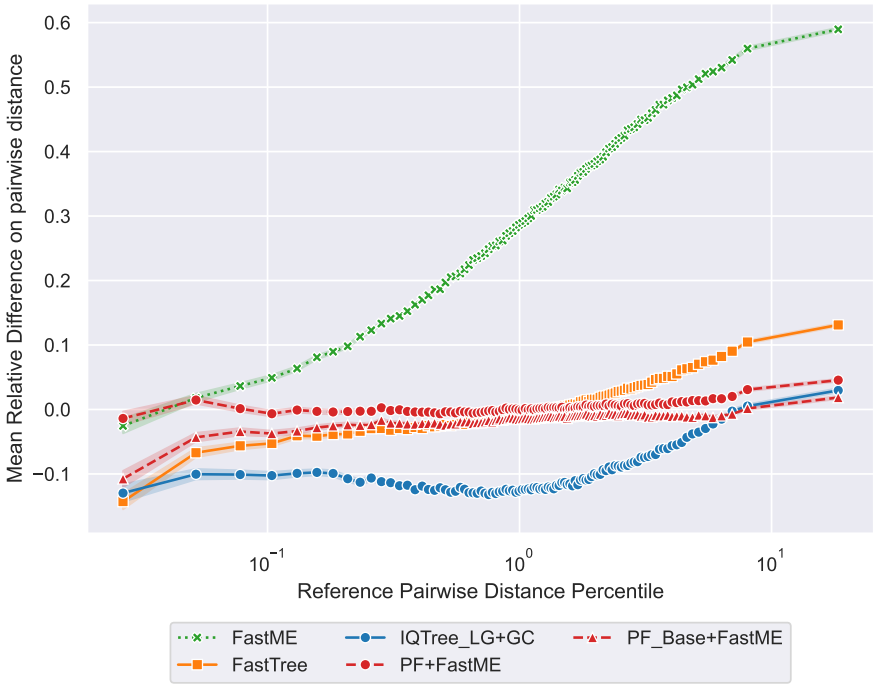


FIGURE 2.24: Mean Relative Difference (MRD) per percentile of true pairwise distances. The MRD is defined as $\ell_{MRD} = N^{-1} \sum_{i=1}^{N} d_i^{-1}(d_i - \hat{d}_i)$ with N the number of pairwise distances and, d_i and $\hat{d}_i$ the true and estimated pairwise distances respectively. In this figure the PF_{Base}+FastME model's performance is shown in addition to that of the MRE-fine-tuned PF+FastME model. The fine-tuning with MRE improved the performance on smaller pairwise distances especially with the MRD for the smallest percentile improving from -0.1 (PF_{Base}+FastME) to ≈ 0 (PF+FastME). Overall, the curve of the MRE-fine-tuned PF+FastME is flatter and closer to 0 than all other tree inference methods.

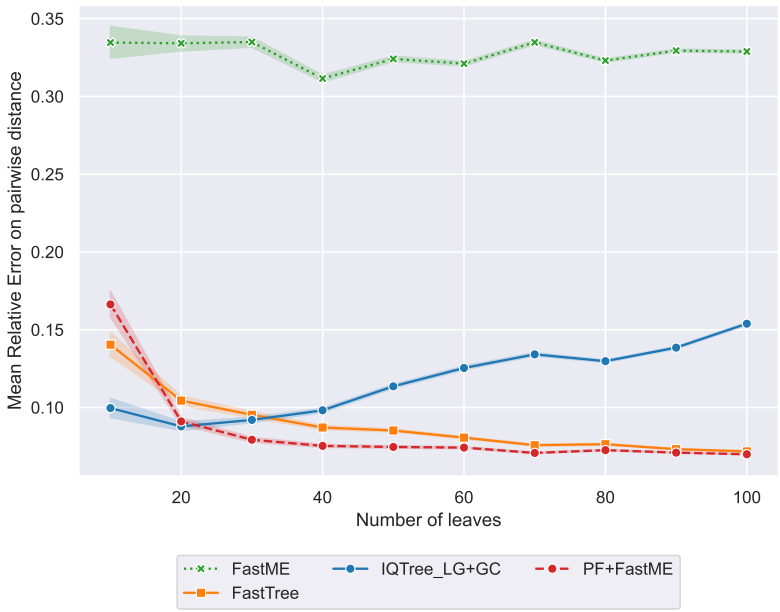
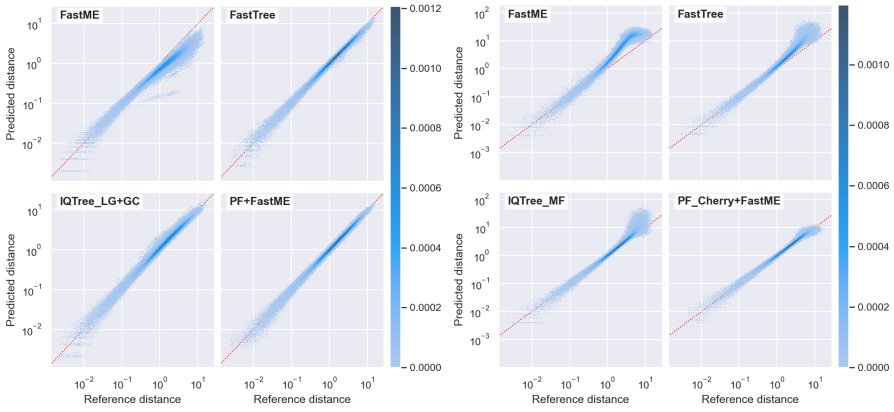
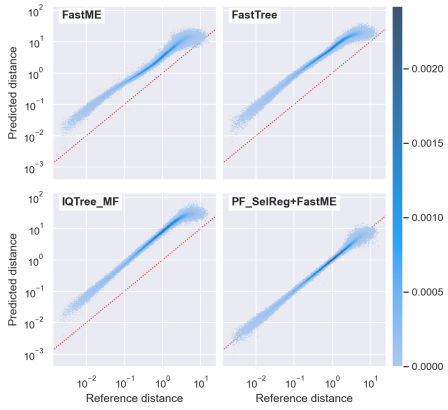


FIGURE 2.25: Mean relative error (MRE) per number of leaves for tree inference methods. Performance is measured on 500-amino acid long MSAs simulated under the LG+GC model. 95% Confidence intervals are estimated with 1000 bootstrap samples.



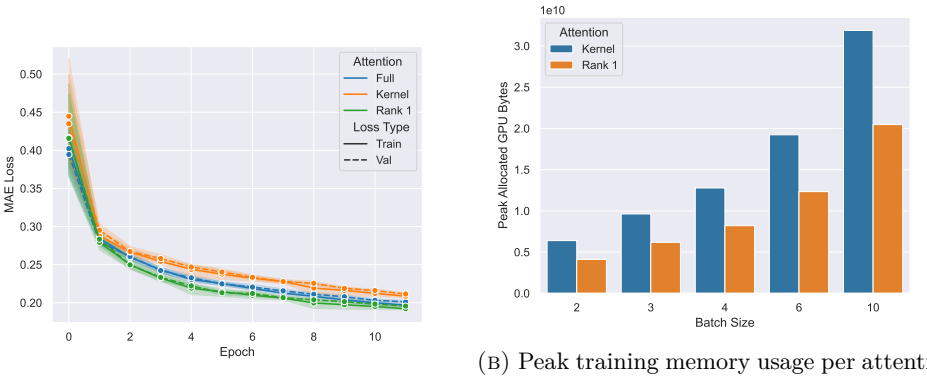
(A) LG+GC

(B) Cherry



(C) SelReg

FIGURE 2.26: 2-dimensional histogram of predicted vs real distances on **(a)** the LG+GC, **(b)** Cherry and **(c)** SelReg test datasets. Under the LG+GC model (a), FastME tends to underestimate longer distances. This is expected as long distances give rise to multiple substitutions per site which are hard to detect when looking at pairs of sequences separately. PF on the other hand, exploiting the information contained in the whole alignment, recovers long distances more accurately. IQTree tends to overestimate all distances. On data simulated with Cherry (b), standard methods overestimate distances, long ones in particular. This is expected as the Cherry model considers pairs of interacting sites as states, and not the indvidual sites themselves. A substitution at the level of the pair can affect both sites, which is interpreted by site-independent models as two independent substitutions. After training on Cherry alignments, Phyloformer predicts accurate distances. On data simulated with SelReg (c), standard methods overestimate distances. Two reasons can explain this. First, the data has been simulated with site-wise vectors of amino acid equilbrium frequencies, whereas the inference model assumes that all sites share a single vector of amino acid frequencies. Second, the model of rate heterogeneity across sites used in inference (the gamma model) assumes a continuous distribution of rates, and probably does not fit well on data in which sites evolved under negative selection, neutral evolution, or positive selection. After training on SelReg data, Phyloformer predicts distances accurately.



(A) MAE loss per attention function.

(B) Peak training memory usage per attention function.

FIGURE 2.27: **a)** Training and validation MAE loss per epoch for the same model with different attention functions: Full scaled dot product attention, Linear Kernel attention and our version of Rank 1 Linear Kernel attention. **b)** Peak allocated bytes on the GPU for the same model with either Linear Kernel attention or our Rank 1 version of Linear Kernel attention. All measures were done on 50 leave simulated trees.

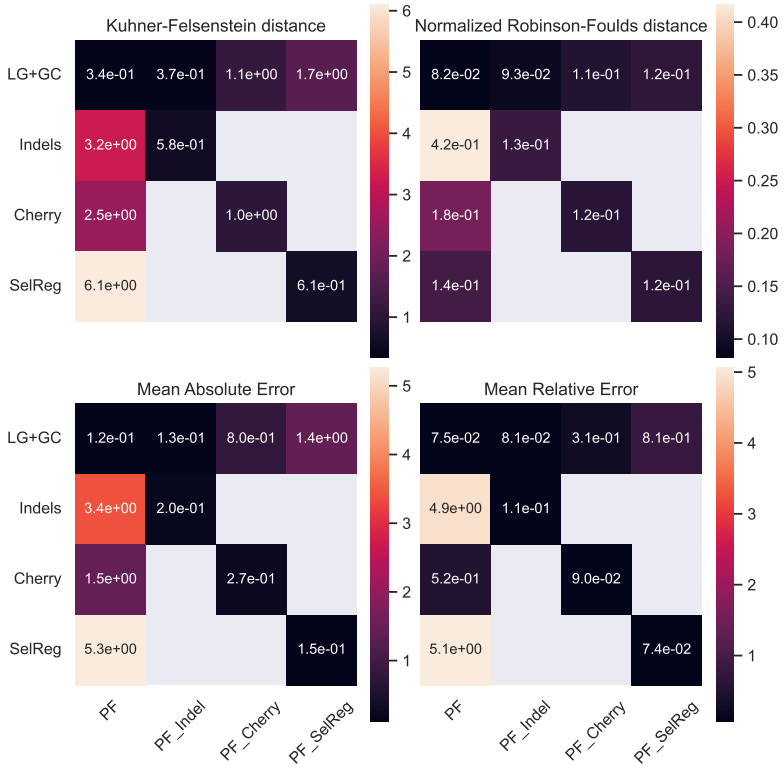


FIGURE 2.28: Effects of model specification for Phyloformer. We measured the performance of Phyloformer models (columns from left to right) fine-tuned with MRE on LG+GC, or fine tuned with MAE on LG+GC data with indels, on Cherry data or on SelReg data. For each model, the performance was measured on 50-leaf trees and corresponding alignments simulated with (rows from top to bottom) LG+GC, LG+GC with indels, Cherry and SelReg. Performance was measured with Kuhner-Felsenstein distance; normalized Robinson-Foulds distance; mean absolute error (MAE) ; mean relative error

2.9 Additional discussion

The previous page was the end of Phyloformer manuscript, here I shall provide some additional discussion and some more insight into the work behind it.

Insertions and deletions

The performances of the PF_{Indel} model 2.6 are remarkable with our method outcompeting the others across all considered metrics, on trees with up to 30 leaves. These results are attained despite the model having been fine-tuned on a dataset three times smaller than the one employed to train the original one. Additional experiments, testing different methods on artificial alignments in which all the information pertaining the substitution process had been removed, allowed us to confirm that Phyloformer is able to exploit to a certain extent the phylogenetic information provided solely by the gaps, a characteristic shared with FastTree, whereas IQTree, treating gaps as missing characters doesn't. It is worth pointing out here that in the considered simulations, although the process of substitution is independent and identically distributed at each position, the indel process introduces correlations in adjacent columns of the resulting MSA whenever an insertion or a deletion with length > 1 occurs. This entails that the architecture presented here, invariant to permutations of sites in the MSA, is not ideal for the problem, being incapable to fully exploit such correlations: Let us consider for instance an insertion or deletion event of length 2, while the network can identify that a gap is present in the same sequence in two different columns, it lacks the information that these are adjacent, information that would suggest that the presence of the gaps is due to a common indel event rather than two independent ones. Again, this limitation could be overcome simply adding a positional encoding to the network's input MSA representation which could improve its performances on these simulations and on empirical data.

FastTree and the advantages of ML optimization

One striking trend one can observe in the presented results is that, while FastTree leads to trees with lower likelihood values with respect to IQTree (figures 2.14 and 2.15), in terms of topological accuracy the two methods generally exhibit the same performances. This is remarkable given the computational advantage that FastTree has over the full ML method. Such a trend was already reported in [182] in which the authors observe no substantial differences between RAxML-NG, IQ-TREE, and FastTree in terms of RF and quartet distances when the methods are tested on simulated data. Such an observation leads them to discuss the issues that come with testing phylogenetic reconstruction methods on empirical data when one takes as a reference the tree with the highest likelihood among those found by different methods. In the context of our work, a supplementary analysis on the LG+GC test dataset was conducted, looking at the starting trees of IQTree and FastTree as well as the trees obtained by the latter after the BME optimization step, before the NNI moves which maximise their likelihood. The results showed that before the likelihood maximisation both methods provide trees which, in terms of topological accuracy, are no better than the ones reconstructed with Phyloformer, this permitted us to conclude that in this case, where the

likelihood of the model is tractable, ML optimization moves are highly effective in minimising the RF distance, we shall further discuss in chapter 5 possible strategies to improve the performances of a method such as ours in this case.

Attention mechanism

Different implementations of attention mechanisms have been explored during this thesis work. Looking at the column attention maps of a Phyloformer model, instead of the row ones, indicates that during this step each pair representation pays more attention to the pairs it shares one of the sequences with. This observation suggested that a sparse attention approach, allowing only such interactions could be tried, nevertheless while its performances appeared to be on par with those of a full attention mechanism the resulting complexity in the number of sequences was $\mathcal{O}(n^3)$ and a way to combine this with a linear attention mechanism (which in turn has $\mathcal{O}(n^2)$ complexity working on the pairs of sequences) couldn't be found. As for the linear attention, multiple versions have been tried before settling on the one presented in [166] which with respect to others has the advantage of maintaining the permutational equivariance. Finally, the rank-1 version presented in this paper is an example of how sometimes in research one makes progress in unexpected ways: The version on which the Phyloformer model presented in the first preprint relied was the result of an inaccurate implementation of the mechanism proposed in [166], roughly equivalent to the rank-1 version presented here. Rather than seeing this just as a bug we realized it could be a feature as ultimately for the problem at hand it retains the same performances as the full rank implementation while being more efficient.

Coevolution patterns

The ability of Phyloformers models to identify coevolution patterns exhibited in figure 2.13 is quite interesting, even more so as it appears to emerge even in networks that have been trained solely on alignments without coevolving sites. A natural question that arises is whether these models could showcase the same ability when run on empirical alignments, to test this I investigated if we could fit a function of the attention maps to predict empirical contact maps, used as a proxy of coevolution patterns in this case, as the authors of [145] do, unfortunately the results were inconclusive but I do believe that further research in this direction could be fruitful.

Is length generalization an issue?

During the development of Phyloformer, in several tests of our model we have observed that it seemed to perform on par or even better than maximum likelihood in the reconstruction of trees with 20 leaves, the same number on which the model initially had been trained on. We already discussed how this could be explained by the shift in the distribution of distances between training and testing data given by the initially adopted tree simulation scheme, nevertheless the hypothesis that the number of sequences itself could play a role, with the model suffering from *length generalisation*, a phenomenon known in the transformers literature [183], could not be excluded. To investigate this

phenomenon several experiments had been conducted, testing the model on simulations with different schemes to rescale the branch lengths in order to match the distribution of evolutionary distances in the training data to that of the test trees. These include, apart from the finally adopted strategy to rescale the branch lengths in order to make all trees have the same diameter distribution as in the training data (matching the empirical one), trying to scale the branch lengths in order to make the mean of the distribution match and different quantile normalizations strategies (for the branch lengths or the evolutionary distances). The results of several of these experiments lead us to believe that the model was indeed performing best when given the same number of sequences it has been trained on, for instance we observed better performances when bigger alignments were cut into several subalignments with 20 sequences each and the predicted distances were combined to reconstruct the entire tree. Eventually, moving from training the model on alignments with 20 sequences to alignments with 50 sequences allowed us to conclude that the main factor affecting the model's performances was the total number of distances it has seen during training rather then the size of the training MSAs, phenomena as the one previously mentioned can then be partly explained by the advantages given by model averaging. Indeed, the phenomenon of the model at times performing better for trees with 20 leaves can still be observed to a certain extent in the plots shown in this chapter, even if the networks we presented here have been trained exclusively on 50-leaf trees, for instance figures 2.19 and 2.20 show that, in terms of RF distance, even the non-fine-tuned network attains state of the art performances on the Cherry and SelReg test datasets for trees up to 20 leaves, while in figure 2.18, panel b, we can see that both PF and PF_{Base} attain the minimum of the Robinson-Foulds distance for trees with 20 leaves. This is probably an artifact of the simulation procedure. As we discussed in subsection 2.7.3 two phenomena are at play, on the one hand, given that we always use the same distribution of diameters, bigger trees contain more small distances, leading the peak of their distribution to slowly shift to the left as the number of leaves increases (figure 2.16), on the other the resampling procedure to avoid duplicate sequences filters out trees with branches which are to short to provide variation in the corresponding alignment, this phenomenon is less prominent in smaller trees, these therefore retain more small distances as, given the shorter paths along the trees, these lead to short branches to a lesser extent. The trees with 20 leaves, among the ones considered for testing, then likely represent a sweet-spot between the two phenomena leading to easier inference for our model.

Optimization

The optimization of the models presented in this chapter has been a considerable challenge in itself, significant work has gone into making the training more efficient and robust through the exploration of different optimizers and warmup schedules. A source of instability has been identified to be the automatic mixed precision (AMP) as implemented in PyTorch we had been employing to reduce the computational footprint of training the models, notably getting rid of the AMP made retraining models easier and allowed for efficient fine-tuning. As for the learning rate schedule, the warm-up schedule adopted here helped stability during training, in the next two chapters however we shall see that

these types of models can also be trained with a simple constant learning rate, although very small values for the latter are sometimes warranted to avoid optimization issues.

MRE fine-tuning

In particular training the Phyloformer model under the MRE loss has been challenging due to optimization difficulties (as one can notice in table 2.1 PF is the only network which has been trained with a learning rate of 10^{-4} instead of 10^{-3}). This was expected as small values (small distances) in the denominator lead to very big loss values and the use of such a loss in regression problems is often not recommended given that it leads the model to underestimate larger values prioritizing accurate predictions for the smaller ones. Nevertheless, empirical risk minimization based on MRE is still consistent [184], and in our case the approach was justified by several experiments which highlighted the accuracy of the network on smaller distances as one of the main factors behind the performances of the method in terms of topological accuracy. Given that the fine-tuning of the model under this loss improved its performances the reader may wonder why the same strategy wasn't adopted for the networks fine-tuned on the more complex models of evolutions $PF_{\mathrm{I}ndel}$, $PF_{\mathrm{S}elReg}$ and $PF_{\mathrm{C}herry}$, or at least why $PF_{\mathrm{B}ase}$ instead of the better performing PF was chosen as a starting point for these fine-tunings. This is simply due to the chronological order in which the experiments had been carried on, we had already trained $PF_{\mathrm{I}ndel}$, $PF_{\mathrm{S}elReg}$ and $PF_{\mathrm{C}herry}$ before the optimization under the MRE loss which lead to the PF model. Although we believe that the same strategy could improve their performances as well, the showcased results of Phyloformer method on the more complex models of evolution we considered are already remarkable so we didn't feel the need to retrain the networks for the purpose of this paper.

Working with sequence representations

Currently the bottleneck limiting the direct application of the Phyloformer model to alignments with more than a few hundred sequences is given by its memory consumption. While naturally bigger trees can be dealt with either with supertree methods (subsection 1.5.3) or combining the distances predicted on smaller subalignments, it would be desirable to have a more scalable method. While no matter that the complexity of a method based on evolutionary distances will at best remain quadratic in the number of sequences, as it already is in our case, we still can hope to be able to reduce the computational footprint of the method for practical applications, given the high coefficient of the quadratic term that stems from the linear attention on all the pairs of sequences. A great deal of work has then been put in trying to develop a version of the Phyloformer network that would work using a representation of the sequences instead of the pairs thereof. Such a network would have the same architecture but would be missing the pairwise average over the sequences step and the final average over the sites, the representation of the MSA the network would be working with, a tensor in $\mathbb{R}^{n\times L\times d}$, would then represent in the final step a context-aware embedding of

each sequence in $\mathbb{R}^{L\times d}$ (with the context being provided by all other sequences in the alignment). The final output of the network, the distance for each pair of sequences, could then be given by the Euclidean distances between these embeddings (or of a transformation thereof). Numerous approaches have been explored, including learning a custom symmetric bilinear form as in [162] to extract the pairwise distances from the embeddings instead of simply taking their Euclidean distance. Additionally, given that the elementwise square root of phylogenetic distance matrices has been proven to be Euclidean [185] [186] we tried using such a transformation (replacing the labels to predict with their square roots), whereas the growing literature on representations of trees in hyperbolic spaces [187] [188], including applications to phylogenetics [189][190][191] lead us to try using hyperbolic distances to extract the network's predictions from the embeddings[1]. Unfortunately hyperbolic approaches lead to numerous optimisation issues whereas the others resulted in networks with fairly good performances on alignments with the same number of sequences they have been trained on but which failed to generalise to smaller or bigger trees. In conclusion, further work will be needed in this direction to develop a more scalable version of our model.
We shall however see in the following two chapters how, for different prediction tasks, the architecture developed for Phyloformer can effectively work with a sequence representation as the one described here, offering great computational efficiency.

[1]Of note the approach adopted [192] to embed sequences in an hyperbolic space is conceptually close to the one discussed here, nevertheless the authors do not explore applications to phylogenetic reconstruction and the resulting embeddings are not context-aware, with each sequence being processed independently.

Chapter 3

Deepelican

The work presented in this chapter stems from a project on which Estelle Bergiron, an M1 student, has worked on for her internship, cotutored by me, Bastien Boussau and Julien Barnier. The work builds upon the one that has been carried by Louis Duchemin during his PhD thesis for the development of the Pelican method. Although the neural network-based approach presented in this chapter is fundamentally different, the two methods tackle the same problem, the detection of sites subject to a selection shift associated with a given phenotype, and the name Deepelican was chosen for the model presented here as an hommage to its non-neural counterpart.

3.1 Detecting shifts in selective pressure associated with a phenotype

We have already encountered in the previous chapter a model of evolution which takes into account different selective pressures that may act on a site, the three scenarios we considered in those simulations were:

- The absence of selection, otherwise known as neutral evolution, which entails all amino acids having the same fitness, so that the evolution of a site subject to no selective pressure can be modeled with a simple continuous time Markov chain such as those introduced in section 1.2.
- Negative, or purifying, selection, in which case one or few amino acids have higher fitnesses with respect to the others and are therefore selected for (respectively the others are selected against) at the given position.
- Persistent positive selection, which leads the fitness profile of a site to continuously change along the phylogeny to adapt to an ever-changing selective pressure as the one that may rise from the co-evolution of competing species that find themselves in an evolutionary arms race, such as the one that often can be observed in host–parasite dynamics.

One evolutionary scenario that has not been considered in these simulations though, is that of a shift in selection, namely the change of the fitness profile of a site which happens episodically along the phylogeny. Such a shift may arise when the evolutionary pressure acting on one or multiple sites changes following for instance a change in the environmental conditions or the arisal of

a new phenotype (potentially caused by such an environmental change). This scenario in particular, the change of the fitness profile of one or several sites correlated with a phenotypic change, is the one I will be focusing on in this chapter. I shall present a novel method, Deepelican, for detecting the sites which have undergone a shift in selective pressure (figure 3.1, right) given a multiple sequence alignment and the phenotypic traits which can be observed at the leaves of the phylogeny the sequences and the phenotypic trait evolved along (figure 3.1, left). While possible extensions of the method to multiple or continuous traits will be discussed, I shall focus here on the case where the trait is discrete and binary and which will therefore be denoted either by 0 or 1 or, alternatively, as background and foreground, sites associated with the phenotype shall further be referred to as positive while the others will be denoted as negative.

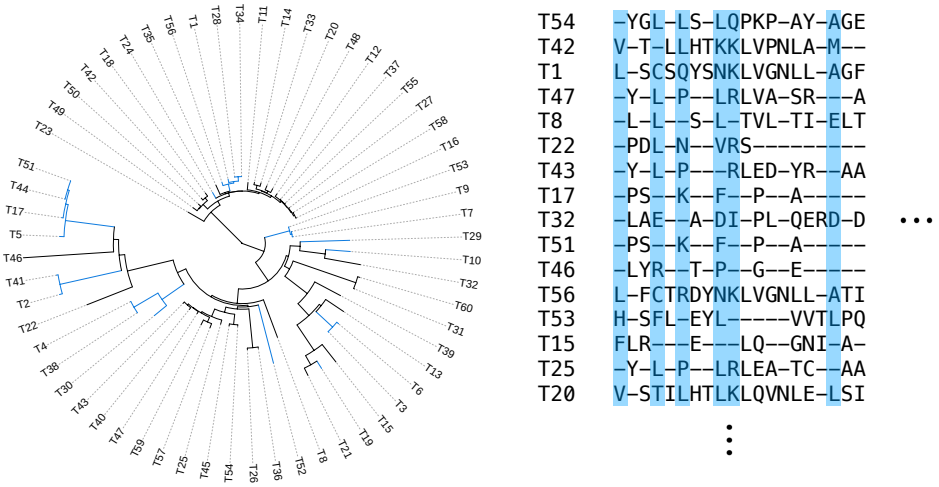


FIGURE 3.1: On the left evolution of a binary phenotypic trait along a phylogeny (represented by the two different colorings of the branches), on the right the corresponding multiple sequence alignment with the sites associated with the phenotypic change highlighted.

3.1.1 Existing methods

Existing methods for the task can be mainly classified in two categories:

d_N/d_S methods

The first category consists of methods working at the codon level and relying on the ω metric. ω is defined as the ratio of non-synonymous to synonymous substitutions d_N/d_S a codon has undergone during evolution along one or several branches of a phylogenetic tree, the synonymous substitutions being those that although changing the codon do not change the amino acid it codes for due to the redundancy of the genetic code, and the non-synonymous being those that do lead to a change of the coded amino acid. Indeed d_N/d_S is a useful metric to detect the intensity of selective pressure as in the absence of selection on the scale of amino acids one would expect the number of synonymous and non-synonymous substitutions to be roughly equal and therefore a value of $\omega \approx 1$, in the case of purifying selection amino acid changes, that is non-synonymous substitutions, are selected against, this scenario can then be detected by an ω value < 1, finally persistent positive selection on the

contrary pushes the amino acid with higher fitness to change continuously and is therefore characterized by $\omega > 1$. Finally the evolutionary scenario we focus on here is characterized by an initial ω value which is smaller than 1 up to the selection shift, a transient phase during which the value becomes greater than 1 as adaptation to the shifted selective pressure takes place, followed by another phase in which the ω value becomes again smaller than 1 provided that no other selection shift arises (figure 3.2).

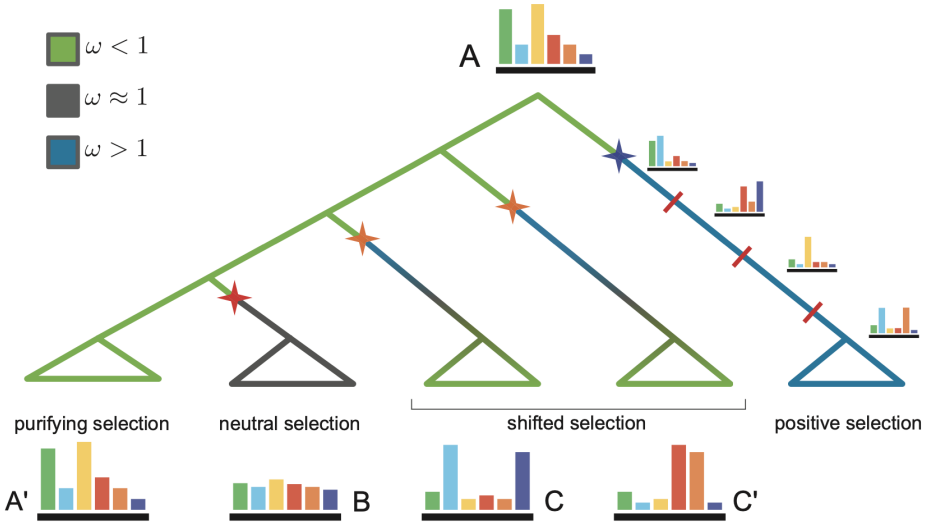


FIGURE 3.2: Different cases of selection regimes for a site, with the corresponding values for the ω metric visualised along with the associated changes in the fitness profile of the site [193]. The shifted selection regime will be the focus of this chapter.

The transient nature of the value in the case of shifted selection in particular might cause d_N/d_S to be less effective in detecting this evolutionary scenario [194]. One such method is the popular "branch-site model A" implemented in the codeml program inside PAML [195] which compares the likelihood of having different ω values for different branches in the phylogeny against that of model having one global set of ω values through a likelihood ratio test (LRT).

Profile methods

The other category of methods comprises of those that do not rely on inferring d_N/d_S values across the phylogeny but rather on amino-acid frequency profiles, which can be used as a proxy of their fitness [1]. Again inference is typically performed in the maximum likelihood framework and a likelihood test can be performed for each site to compare the likelihood of a model allowing a different profile per phenotypic trait against that of a model having a unique profile per site which does not change along the phylogeny. Two such models are considered in this chapter: *Multinomial*, reportedly the fastest profile method, which does not take into account the phylogeny and does not assume an underlying model of evolution, and *Pelican*, an efficient implementation of maximum likelihood inference under the TDG09 model [196] and which has been shown to have comparable performances with the best performing

[1] For this reason, with a slight abuse of notation, we shall use the terms frequency and fitness profile interchangeably throughout this chapter.

methods [174]. More specifically, the Multinomial method models the observed amino acid frequencies at a given site i with multinomial distributions and compares the two hypotheses:

- H_0: A single profile describes the frequencies of amino acids observed in all sequences at a given site.
- H_1: Two distinct profiles describe the observed frequencies at a site depending on the sequence's phenotypic trait.

Denoting by θ_0 the vector of observed amino acid frequencies at site i looking at all sequences, by θ_1^0 the vector of observed frequencies at site i corresponding to sequences with phenotype 0 and by θ_1^1 the one corresponding to sequences with phenotype 1, we can compute the likelihoods under the two models as:

- $L(H_0) = \prod_{j=1}^{n} P\left(s_i^j \mid \theta_0\right)$
- $L(H_1) = \prod_{j=1}^{n} P\left(s_i^j \mid \theta_1^{\varphi(s^j)}\right)$

Where P is the probability given by the multinomial distribution, s_i^j the amino acid found at position i in the sequence s^j, and $\varphi(s) \in \{0, 1\}$ the phenotype associated to the sequence s.

Pelican on the other hand takes into account the annotated phylogeny τ and models the evolution along its branches with a continuous time Markov chain model described by a transition matrix Q (The WAG transition matrix is used by default) and by either a single stationary distribution π (hypothesis H_0) or by two distributions π_0 and π_1 (hypothesis H_1) depending on the phenotypic trait annotation of the branch along which the sequence is evolving. Felsestein's pruning algorithm (subsection 1.3.1) is used to compute likelihoods along the phylogeny and the distributions π, π_0, π_1 which maximize them are found so that in this case

- $L(H_0) = \max_{\pi} P\left(s_i \mid Q, \pi, \tau\right)$
- $L(H_1) = \max_{\pi_0, \pi_1} P\left(s_i \mid Q, \pi_0, \pi_1, \tau\right)$

Where this time P is the probability given by the model of evolution. For both methods one can then compute the LRT statistic

$$D = 2\left(\log\left(L\left(H_1\right)\right) - \log\left(L\left(H_0\right)\right)\right) \tag{3.1}$$

and its significance can be tested with a chi-square distribution, having a number of degrees of freedom equal to the difference in the number of parameters estimated under hypotheses H_1 and H_0, which provides a p-value for the site being identified as positive.

3.1.2 Model of evolution

The model of evolution employed for all the alignment simulations, both for training and testing the introduced model, again implemented in the Pastek software, is a codon-based Mutation-Selection model, the same as in [174] and [197] , which allows only transitions between codons differing by at most one nucleotide and which weights the probability of such a transition by a factor proportional to the fitness difference between the corresponding amino

acids: At the root of the tree, for each site i, an amino acid frequency profile β^i is sampled, by default among the 263 representative profiles selected by [197] among those experimentally assessed in [198]. Given such a profile, the evolution along the branches of the tree of the codon associated to the site is then governed by the transition matrix $Q^i \in \mathbb{R}^{61\times 61}$ such that

$$q^i_{xy} = \begin{cases} \mu_{xy} \cdot \frac{S^i_{xy}}{1-e^{-S^i_{xy}}} & \text{if} \quad x \neq y \\ -\sum_{y\neq x} q^i_{xy} & \text{otherwise} \end{cases} \tag{3.2}$$

where

$$\mu_{xy} = \begin{cases} J_{uv} & \text{if codon } y \text{ is obtained from } x \text{ changing one nucleotide } u \text{ to } v \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

where J is the transition matrix for the Jukes-Cantor model we already encountered in subsection 1.4.1 , and

$$S^i_{xy} = \rho \log \frac{\beta^i_{AA(y)}}{\beta^i_{AA(x)}} \tag{3.4}$$

where ρ is a positive scaling factor which governs the intensity of selection and $\beta^i_{AA(x)}$ is the frequency in the profile β^i corresponding to the amino acid encoded by the codon x. Finally, to simulate the effect of episodic shifts in selection, two distinct profiles β^i_0 and β^i_1 (corresponding to the two different phenotypes) are actually sampled at the root for each positive site, the profile β^i_0 is initially assigned to the site and then the site's profile, which governs its evolution through equations 3.2, 3.3 and 3.4, switches from β^i_0 to β^i_1 or viceversa any time a phenotypic change along the tree is encountered. Whereas in [174] simulations were run on empirical phylogenies already annotated with the phenotypic traits along them, for the simulations presented in this chapter the evolution of a phenotypic trait along a phylogeny was simulated as well by simply modeling it as a two-state continuous Markov chain running along the branches of the tree, with the root being assigned to state 0 and with the rate of substitution being set so that the expected number of substitutions along the longest path from the root to a leaf is equal to 2.

3.2 Deepelican

As described in subsection 3.1.1 the Pelican method requires an annotated phylogeny whereas most generally in empirical data one only has access to the annotation of the leaves, corresponding to the observable phenotypes. If the trait annotation of the phylogeny is not given to the method, the phenotypes of the internal nodes are inferred by maximum parsimony via Fitch's algorithm using the input phylogenetic tree and leaf annotation. The idea underlying the development of Deepelican instead was to skip the tree reconstruction and annotation steps altogether in order to have an accurate method capable of predicting positive sites directly from the MSA and the leaves annotation, the model was therefore trained end to end to predict for each site the probability of it being positive given only the aforementioned data in input.

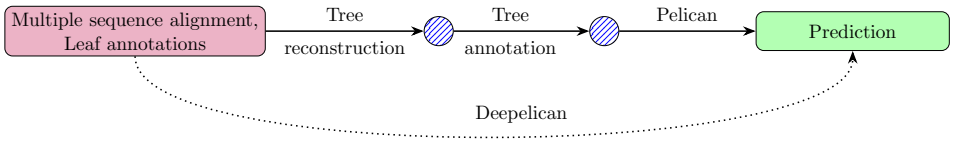


FIGURE 3.3: Inference paradigm using the Deepelican model compared to Pelican.

This paradigm is illustrated in figure 3.3 and was motivated by the demonstrated capabilities of a model such as Phyloformer to deal with MSA data and capture phylogenetic information. Of note this is analogous to what is done in the context phylodynamics by the authors of [199] who use a similar architecture to the one employed throughout this work and which allows them to skip the usually necessary tree reconstruction step for the inference of epidemiological parameters.

3.2.1 Adapting the neural network architecture

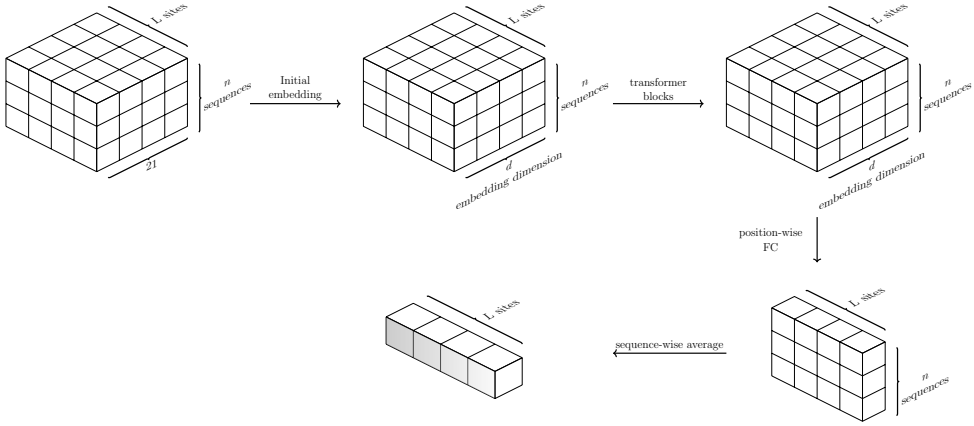


FIGURE 3.4: Overview of how Deepelican processes the one-hot encoding representation of an input MSA, the only differences with the Phyloformer architecture are the missing sequence-to-pair step and the final average over the sequences instead of the sites.

Given the results presented in the previous chapter the neural network architecture of Deepelican was chosen to closely mirror that of Phyloformer, with minor changes to make it suitable to the task, namely that of a per-site binary classification. A schema of how an alignment is processed by deepelican is depicted in 3.4: After an embedding through a position-wise fully connected layer the MSA representation goes through several axial transformer blocks that once more rely on the low rank variant of the linear attention mechanism described in chapter 2, a final position-wise feed forward layer without any activation function and a single output feature then gives us a $\mathbb{R}^{n \times L}$ tensor that, once averaged over the sequences and passed through a sigmoid function gives us the final network's prediction of the per-site probabilities as a $\mathbb{R}^L$ vector. During inference, when a binary classification needs to be made, a site will be classified as positive if the corresponding output probability is greater than 0.5. The trained model presented in this chapter has 8 attention blocks, an embedding dimension of $d = 64$, and 4 attention heads totalling 410,819 trainable parameters. It is worth noticing that despite having a larger number of parameters with respect to the Phyloformer model, Deepelican misses the

pairwise average over the sequence representations step, working therefore with a representation in which the rows directly correspond to the sequences in the input MSA and not to the pairs thereof. Such a difference makes its computational footprint, both in terms of memory and execution time, linear at once in the number of sequences and in the number of sites. This provides, as we shall see in subsection 3.4.3, a significantly more scalable method capable therefore of handling seamlessly alignments with several hundreds of sequences.

3.2.2 Encoding global information

Another main difference with respect to the problem of evolutionary distance prediction is the representation of the input which is fed to the model, indeed whereas Phyloformer took simply an encoding of the MSA as input, for the task at hand we need to provide to Deepelican the annotation of the phenotypic traits at the leaves as well. This raises the question of how to encode a global value shared across all the sites in a sequence, namely the phenotype associated to the latter. To do so two different approaches have been tested, the first consisted in simply encoding the phenotype along a supplementary dimension extending each one-hot encoded vector either with a 0 or a 1 depending on the phenotype associated to the sequence the amino acid finds itself in (which results in an input tensor with dimensions $(N, L, 22)$), the second relied on adding a sinusoidal positional encoding vector to the one-hot encoded one, encoding thus the two different phenotypes with a vector corresponding to position 0 and one corresponding to position 1 respectively (without changing the dimensions $(N, L, 21)$ of the input tensor). Denoting by $e(c)$ the encoding of a character (amino acid or gap) and by $s(c)$ the sequence in which the character being encoded finds itself in we thus have:

$$PE_{(x,2i)} = \sin\left(\frac{x}{1000^{2i/21}}\right), \quad PE_{(x,2i+1)} = \cos\left(\frac{x}{1000^{2i/21}}\right) \tag{3.5}$$

and

$$e(c) = \text{one-hot}(c) + PE_{\varphi(s(c))}. \tag{3.6}$$

Although such a nonstandard usage of positional encodings is rare in the literature it appeared as effective as the other encoding approach in preliminary experiments, eventually leading to its adoption for the model presented in this chapter, of note it is the same approach adopted in [199] to encode the age of the sequences in the input MSA.

3.3 Simulations and training

Simulations

To train and test the Deepelican model alongside the other methods, phylogenetic trees were again simulated following the exact same procedure as in subsection 2.6.1. Alignments were simulated along those with the Pastek simulator using the previously described model of evolution, a notable difference in the whole simulation pipeline with respect to that adopted for Phyloformer is that in this framework the duplicate sequences seldom appearing in alignments simulated along trees with short branches were kept thus avoiding the re-simulation step described in subsection 2.6.1. This choice was made for simplicity as in this framework, given that predictions are made at the site

level, the presence of duplicate sequences wouldn't give rise to the potential issues discussed in chapter 2, their presence indeed would at worst provide redundant information.

Whereas in [174] different methods were compared on simulations with a fixed proportion of positive sites, namely 10%, the simulations we relied on in this chapter were made slightly more diverse, drawing uniformly from $\{0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$ the proportion of positive sites in each simulated alignment.

The simulations were further enriched with the presence of gaps, to introduce the latter in the simulations a simple model was employed in this case. We relied on a custom implementation of the TKF91 model of insertion-deletion [200] which typically models the latter as a birth-death process running along the branches of the tree. Allowing insertion events though wouldn't be straightforward in the considered framework as naturally the question of which class to assign to the newborn site would arise, to avoid this the simplifying choice of allowing only deletion events was made, with gap deletion rates drawn uniformly from $\{0.0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$ for each alignment. A value of $\rho = 0.5$ for the intensity of selection parameter was used to train the base model whereas $\rho = 4$ has been used to obtain the fine-tuned version presented in subsection 3.4.4

Training

The models presented in this chapter were all trained with a fixed learning rate of 10^{-3} using the Adam optimizer and a batch size of 20 on a single NVIDIA RTX A4000 GPU with 16GB of VRAM. Since in this case we're dealing with a per-site binary classification task the loss under which the models had been optimized was chosen to be the commonly employed binary cross entropy (BCE) $\mathcal{L}(\hat{y}, y) = -\left(y \log(\hat{y}) + (1-y)\log(1-\hat{y})\right)$. As for constant sites in the MSA no meaningful prediction can be made regarding the association with the phenotype, during the training of the model a binary mask is applied to exclude the latter from the loss calculation, similarly, the predictions for the sites which are constant in the MSA are not considered in the computation of different performance metrics for all the methods considered in this chapter. Furthermore, to deal with class imbalance during training, the positive sites in the input MSA are selected and an equal number of negative sites is sampled in order to compute the loss on a balanced proportion of sites belonging to each class.[2] The base model was trained for 19 epochs on 400k alignments with 50 sequences and 250 sites using a validation dataset of 20k alignments, the fine-tuned model was trained on 20k alignments for 5 epochs with the same hyperparameters as the base one.

[2]In hindsight this was a poorly chosen strategy as by doing so one effectively makes the proportion of positive sites in the input MSA irrelevant, indeed overall this is equivalent to having simply trained the model on alignments with 50% of positive sites. This penalises equally type I and type II errors, whereas to apply the network to empirical alignments, with potentially thousands of sites, or even millions at the genomic scale, one is rather interested in reducing the number of false positives. This could have been done either by computing the loss for all sites, instead of a subsample, during training, or by simply simulating alignments with a 50% positive sites proportion and assigning different weights in the loss for the two classes.

3.4 Results

3.4.1 Testing with different tree sizes

Firstly the base Deepelican model was tested on alignments simulated under the same conditions as the training data, only varying the number of leaves of the simulated trees. Figure 3.5 shows the comparison of the performances of the model with those of Multinomial and Pelican in terms of area under the Precision-Recall curve (PR-AUC) and in terms of simple balanced accuracy (BAcc), namely the average of the per-class accuracies. Whereas PR-AUC is the same metric already used to compare different methods in [174] and [197], it is important to note that the BAcc metric, looking at which one would be lead to think that Multinomial is doing better than Pelican here, is less meaningful for these two methods given that Pelican is reportedly not well-calibrated [193], furthermore the threshold chosen here to classify a site as positive is when the corresponding p-value is < 0.05, i.e. the same that was used to assess the performances of the method on empirical data [193]. As we shall see this choice indeed leads Pelican to have a better BAcc with respect to that of Multinomial when the methods are tested on simulations using a more realistic value of $\rho = 4$, whereas in this case with $\rho = 0.5$ a better threshold choice could have been made as the PR-AUC metric does indicate the advantage of Pelican over the simpler Multinomial method. Given these premises, the choice to report this metric was still made as it is the one we kept track of alongside the loss during the training of Deepelican and it is a simple and intuitive indicator of its performances across different simulation settings.

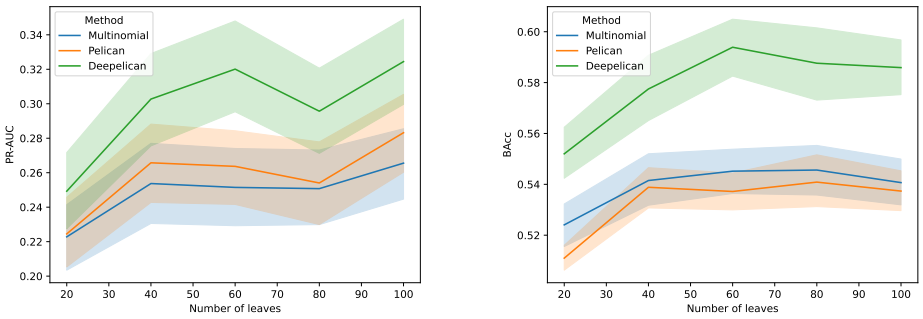


FIGURE 3.5: Performances of different methods tested on simulated trees with increasing number of leaves (100 per tree size) and alignments of length 250 simulated along them. 95% confidence intervals estimated with 1000 bootstrap samples are shown.

As expected the performances of all methods improve with the tree size as the additional information provided by the growing number of sequences allows one to better identify positively selected sites. While the capability of all methods to identify positive sites is limited, reflecting the difficulty of the problem, which as we shall see is to be attributed to the low value of $\rho = 0.5$ governing the intensity of selection chosen for these simulations, one can see that the newly developed Deepelican method outperforms the others no matter the tree size. These performances are attained by the model despite not being given in input the original tree the sequences evolved along as it is the case for Pelican and with a significantly lower computational footprint with respect to the latter as we shall see in subsection 3.4.3

3.4.2 Testing with different profiles

Given the showcased remarkable performances of Deepelican naturally the question of what may explain them is raised, in particular one may hypothesize that they are due to the fact that the training phase of the model allowed it to learn the 263 fitness profiles employed in the simulations and would fail to generalise in evolutionary scenarios in which a different set of profiles is involved. To test such an hypothesis we once again tested Multinomial, Pelican and Deepelican on alignments which were simulated using different sets of profiles, for this we employed the ones from [201]. These consist of 14,509 sets of profiles (one per Orthomam gene alignment), for a total of $8,895,129$. These profiles were estimated with a mutation-selection model [201] while it is worth reminding that on the other hand the 263 frequency profiles from [198] used in the training simulations for the model were instead obtained through an experimental deep mutational scanning. As one can see in table 3.1 the results of the experiment show that actually the positive site detection task for alignments simulated using these profiles appears to be slightly easier for all methods, Deepelican included.

Method	**263 profiles**		**8,895,129 profiles**	
	BAcc	**PR-AUC**	**BAcc**	**PR-AUC**
Multinomial	0.542	0.239	0.562	0.278
Pelican	0.526	0.240	0.542	0.281
Deepelican	0.585	0.289	0.599	0.305

TABLE 3.1: Performance Metrics for Different Methods on simulations under different sets of profiles, 300 alignments with 50 sequences and 250 amino acid were simulated for both tests, using for each alignment the same set from [198] or sampling uniformly one of the 14,509 sets from [201] respectively.

Further investigation is required to pinpoint the reasons of such a difference, for instance looking at the Jensen–Shannon divergence between the profiles as had been done in [197], this however goes beyond the scope of the experiment, which was simply to rule out the overfitting of the Deepelican model with respect to the profiles used in the simulations it has been trained on. Given that no drop in the performances has been observed one can safely assume that the method is generalisable to evolutionary scenarios in which different amino acid profiles are involved and overfitting with respect to the latter is not an issue.

3.4.3 Speed and memory performances

As previously mentioned the different neural network architecture of Deepelican with respect to Phyloformer makes its computational complexity linear both in the numbers of sites and sequences. As we can see in figure 3.6, where we compare its execution time with that of Pelican, this gives the method a great computational edge even when the model is run on a CPU.

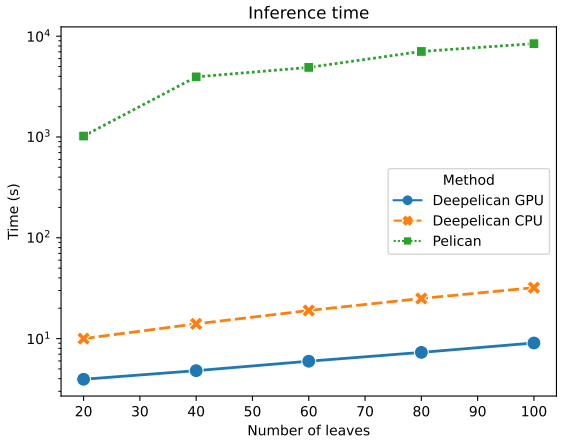


FIGURE 3.6: Inference time (in seconds) for different methods on 100 test alignments of length 250 for increasing numbers of leaves. It is worth to point out that the inference time for Pelican here includes that of Multinomial as well, as the latter is run by default in the Pelican software prior to the Pelican analysis, nevertheless this time overhead is negligible given that the former's execution time is several orders of magnitude smaller [174].

For this comparison both Deepelican CPU and Pelican were run on a single CPU thread (Intel Core i7-10700 CPU 2.90GHz) while Deepelican GPU was run on an NVIDIA RTX A4000 GPU. Both on CPU and GPU Deepelican was run with a batch size of one. One can see here that the neural network approach is 2 to 3 orders of magnitude faster. Similarly the memory consumption of the method is no longer a bottleneck with a maximum of 1.4 GB required for inference on the alignments with 476 sequences simulated along the HIV phylogeny (subsection 3.4.4). Again the parallelisation of inference is straightforward by simply running the model with larger batch sizes which in this case can be easily handled given the relatively low memory footprint of running the model.

3.4.4 Testing on empirical trees

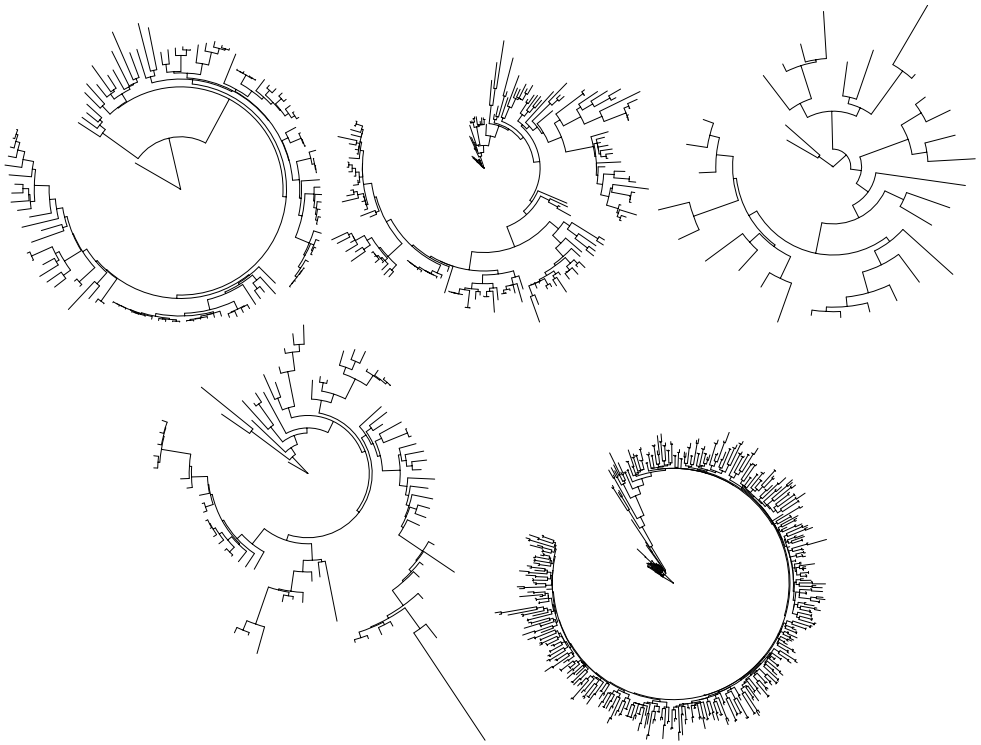


FIGURE 3.7: Empirical trees along which alignments were simulated: Orthomam, Amaranthaceae and Rodents (top), Cyperaceae and HIV (bottom).

The performances of the three considered methods were further tested simulating alignments under the same model of evolution but along five empirical phylogenies already considered in [174]. The experiments shown here further differ from the ones there reported in that, notwithstanding the fact the phylogenies are already annotated with different phenotypic traits, here the latter are re-simulated along the branches as described in section 3.3. Multiple alignments evolved along the same tree will then have different corresponding trait annotations for the sequences. The phylogenies in question are depicted in figure 3.7 and are:

- Orthomam: A 116 leaves phylogeny of mammalian species. [202]
- Amaranthaceae: A 179 leaves phylogeny of a family of flowering plants. [203]
- Rodents: A 32 leaves phylogeny of rodent species.[197]
- Cyperaceae: A 79 leaves phylogeny of another family of flowering plants. [204]
- HIV: A 476 leaves phylogeny of HIV strains. [205]

100 alignments, again with 50 sequences and 250 sites, were simulated along each phylogeny and the results of testing the three methods on them are reported in table 3.2, Precision-Recall curves are further shown in figure 3.8.

Method	Amaranthaceae		Cyperaceae		HIV		Orthomam		Rodents	
	BAcc	PR-AUC	BAcc	PR-AUC	BAcc	PR-AUC	BAcc	PR-AUC	BAcc	PR-AUC
Multinomial	0.515	0.236	0.519	0.213	0.545	0.235	0.514	0.209	0.506	0.209
Pelican	0.507	0.251	0.513	0.217	0.527	0.261	0.503	0.212	0.505	0.216
Deepelican	0.520	0.251	0.551	0.240	0.549	0.255	0.527	0.229	0.537	0.227

TABLE 3.2: Performance metrics for different methods on alignments simulated along 5 different empirical phylogenies.

Whereas the PR-AUC values reported in the table are averaged over all alignments simulated along an empirical phylogeny, the curves in the figure are obtained computing the precision and recall of different methods for all the per-site predictions ($250 \cdot 100$ minus the number of constant sites) for a given phylogeny.

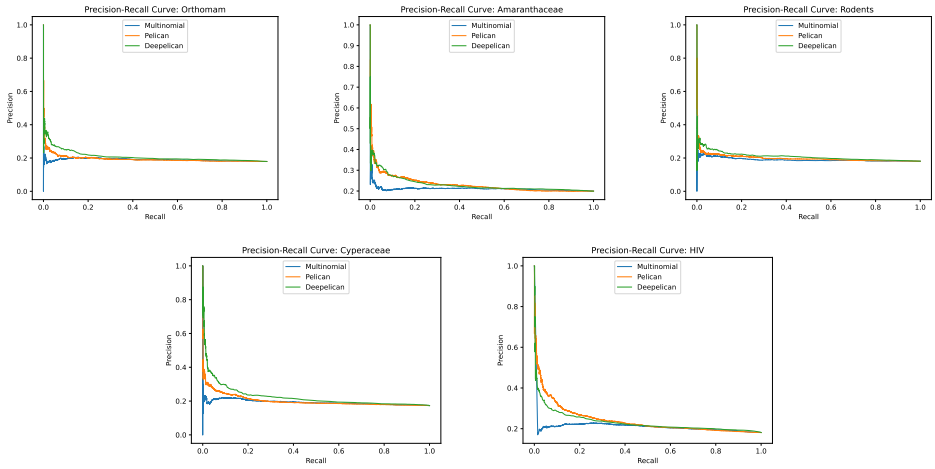


FIGURE 3.8: Precision-Recall curves obtained considering all per-site predictions of the different methods for the alignments simulated along 5 empirical phylogenies.

We can observe then a relative performance drop for Deepelican when tested on these simulations: It remains the best performing one on 4 out of 5 dataset but it is on par with Pelican on the Amaranthaceae dataset and is surpassed by the latter on the HIV one. The HIV tree is the largest one, having a number of leaves one order of magnitude bigger than those used for training the Deepelican model, therefore some issue of length generalisation might play a role here. To further interpret these results we can resort to the same statistics considered in subsection 2.6.1[3] to visualise the empirical phylogenies alongside the simulated one from Deepelican's training dataset, a two component PCA plot based on these statistics is shown in figure 3.9. Zooming in we can see that indeed the Amaranthaceae tree falls slightly outside the distribution of those used for training the model and that the others, with the exception of Cyperaceae, fall on the edge of it, notably Orthomam. This can explain Deepelican's relative performance drop and shows a limitation of our tree-simulation protocol.

[3]That is: Mean distance between leaves, standard deviation of distances, maximum distance, mean branch length, standard deviation of branch lengths, maximum branch length, mean root to leaf distances and standard deviation of root to leaf distances.

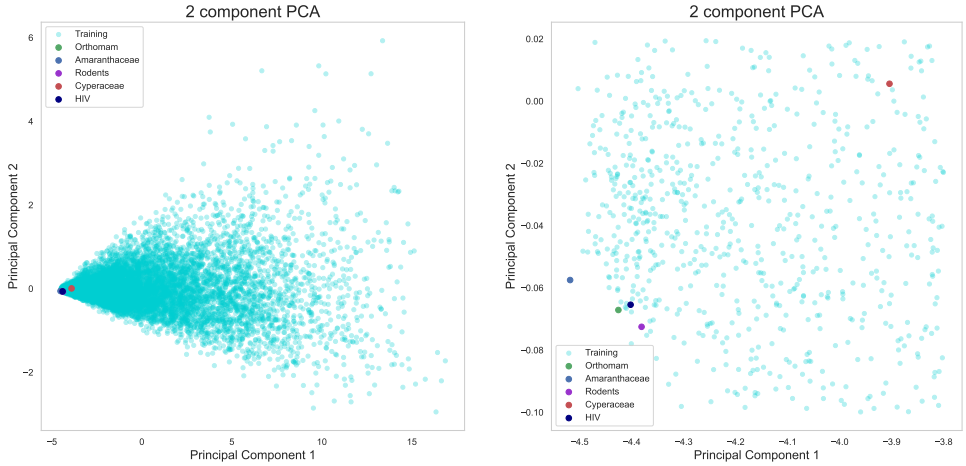


FIGURE 3.9: Two components PCA plot of the simulated and empirical phylogenies, the PCA is based on the same tree statistics that were considered in subsection 2.6.1, full plot on the left (10k trees sampled from the training dataset) and zoomed-in view on the right.

More realistic simulations: $\rho = 4$

All the alignment simulations presented so far have been obtained using the value $\rho = 0.5$ for the intensity of selection in the simulator. This choice was somewhat arbitrary and, as previously pointed out, makes the inference task particularly hard, leading to PR-AUC values which greatly differ from those reported in [174]. We therefore now show how the use of a more realistic value of $\rho = 4$, chosen in [197] to fit empirical data, affects them. Again 100 alignments were simulated along each empirical phylogeny and the results are presented in a table (3.3) and visualized with full Precision-Recall curves (figure 3.10). Alongside the methods already considered we included the performances of a fine-tuned version of the Deepelican model, here denoted as Deepelican4. The latter was obtained further training Deepelican, on a significantly smaller dataset, consisting of 20k alignments, simulated in the exact same fashion as the ones already used to train the Deepelican model, only setting the ρ value in the simulator to 4 instead of 0.5.

Method	**Amaranthaceae**		**Cyperaceae**		**HIV**		**Orthomam**		**Rodents**	
	BAcc	**PR-AUC**	**BAcc**	**PR-AUC**	**BAcc**	**PR-AUC**	**BAcc**	**PR-AUC**	**BAcc**	**PR-AUC**
Multinomial	0.667	0.686	0.741	0.758	0.779	0.769	0.655	0.661	0.594	0.707
Pelican	0.696	0.809	0.779	0.841	0.811	0.854	0.676	0.728	0.625	0.715
Deepelican	0.741	0.804	0.769	0.815	0.771	0.808	0.699	0.731	0.715	0.753
Deepelican4	0.777	0.841	0.834	0.881	0.823	0.861	0.768	0.792	0.777	0.839

TABLE 3.3: Performance metrics for different methods with on alignments simulated with $\rho = 4$ along 5 different empirical phylogenies.

As expected the results show greatly improved performances for all methods, with those of Multinomial and Pelican now resembling the ones reported in [174]. The performance of Deepelican, which was trained only on a much harder task, generalise quite well staying clearly above those of Multinomial, the method is nevertheless surpassed by Pelican in this setting. On the other hand the fine-tuning of the model proves to be successful with the new Deepelican4 version attaining the best performances across all the datasets. This shows at once that, although the use parameters that differ between training and testing data does affect the performances of the model, the latter shows some robustness to such a misspecification, and that the model can be easily adapted to deal with different evolutionary scenarios.

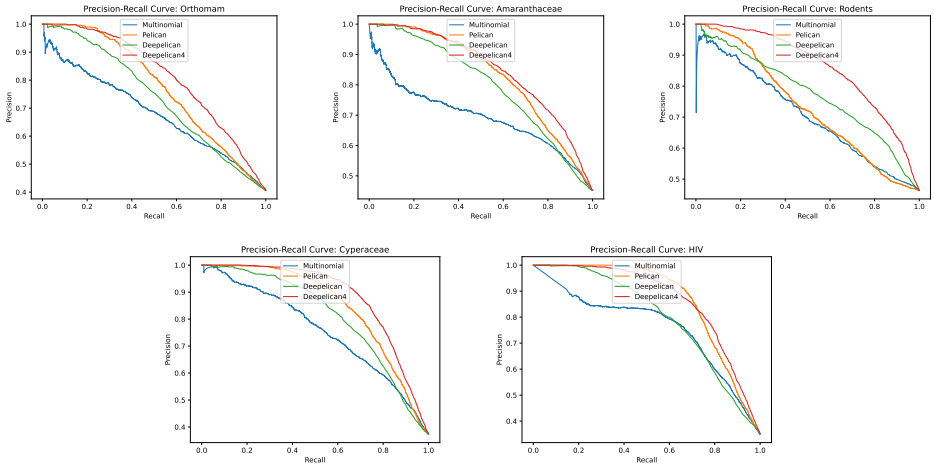


FIGURE 3.10: Precision-Recall curves obtained considering all per-site predictions of the different methods for the alignments simulated with $\rho = 4$ along 5 empirical phylogenies.

3.4.5 Testing on empirical alignments: The Prestin gene

To assess the performances of the novel method on empirical data the fine-tuned version Deepelican4, which from now on we shall just denote as Deepelican for simplicity, will be used. An empirical alignment we chose to test our model on is the one of the Prestin gene, which codes for a protein which plays a crucial role in hearing in mammals, and is studied here with respect to its association to the echolocation phenotype, a trait that has independently evolved in bats and whales through convergent evolution. This allowed us to compare in figure 3.11 the predictions of Deepelican with those of Pelican and to the sites found to be associated with the phenotype in [206].

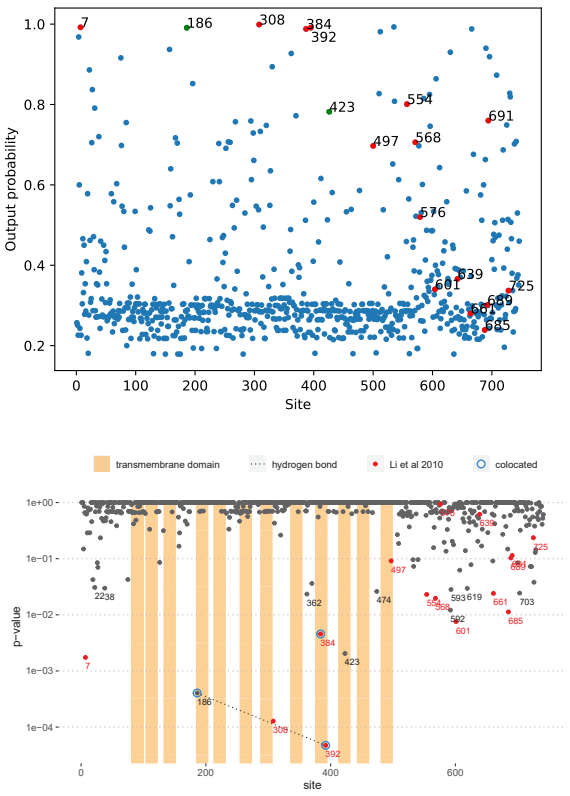


FIGURE 3.11: Deepelican's (above) and Pelican's (below) predictions [193] for the Prestin gene, in red the sites reported by Li et al. [206], in green the sites found by Pelican with a p-value $< 10^{-2}$ but not previously reported in the literature. The reported site indexes along the sequence here are the ones relative to the human sequence of Prestin which contains three gaps at the positions 374-376 in the Orthomam multiple sequence alignment of the gene.

As Pelican Deepelican finds site 186 which although not directly reported in the publication is bound to the reported site 392 by an hydrogen bond which probably causes the sites to be coevolving. Overall, although with an higher proportion of potentially false positives, most probably to be attributed to the incorrect way class imbalance was dealt with during training, the ability of the neural network to recover relevant sites appears to be comparable to that of Pelican.

3.4.6 Testing on empirical data: Gene enrichment analysis

As discussed in [193] the speed efficiency of a method such as Pelican allows its use for genome-wide association analyses, and, given the clear-cut computational edge that Deepelican has over the former, this is even more true for the method presented here. To showcase the potential of the method in this context Deepelican was run on all 14509 gene MSAs from Orthomam, using again the phenotypic annotation of echolocating species at the leaves. This allowed us to compare the performances of Pelican and Deepelican in identifying genes reported in the literature as being involved in sound processing and thus potentially associated to the echolocation phenotype. In order to do so one needs to aggregate the per-site predictions into a single per-gene value, the Pelican method used in [193] an adapted version of Fisher's method [207], which allows to aggregate p-values from multiple independent tests having the same null hypothesis into a single statistic, namely just taking the sum of the $k = 5$ best (lowest) per-site p-values for each gene. Here we simply adapted this method for aggregating the probabilities given by Deepelican into a single per-gene score using the sum of the k best (highest) per-site probabilities. In figure 3.12 we can see the results of such an experiment, the 14509 genes were sorted according to the computed score and their rank n is shown along the x axis, on the y axis the cumulative number of genes reportedly associated with sound processing progressively identified among the first n ranked genes is plotted. We show the performances of Pelican (using $k = 5$ as in [193]), of a random baseline and of Deepelican using the values 1, 5 and 10 for k.

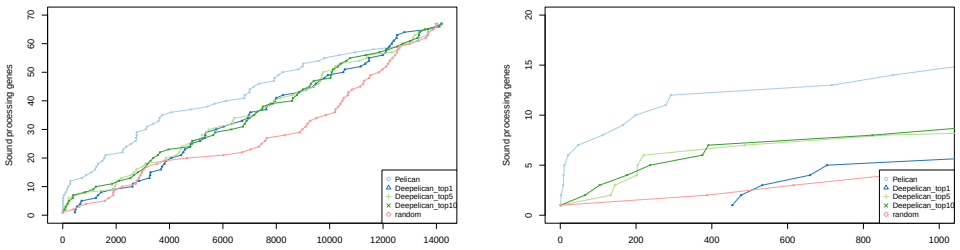


FIGURE 3.12: Gene enrichment analysis with respect to sound processing genes, with all the 14509 Orthomam genes on the left and zooming on the first 1000 genes on the right.

We can see that although the signal provided by Deepelican is weaker than that of Pelican, it is still noticeably above the random baseline, showing the ability of the method to make meaningful predictions at the gene-level as well.

3.5 Discussion and future perspectives

The results in this chapter showcase the versatility of the developed neural network architecture with a novel method for detecting sites having undergone a selection shift associated to a phenotypic trait, which can outperform previous methods in all the considered simulations while running up to three orders of magnitude faster. The success of the model can be partly attributed to the capability of such architectures do deal with MSA data and encode phylogenetic relationships as already demonstrated by Phyloformer, this can be visualised in figure 3.13 where an (averaged) row attention map, (computed as in subsection 2.7.1) is shown alongside the tree along which the corresponding alignment was simulated. Although it is not straightforward to ascertain to which extent the phenomenon is simply due to the similarity of the input sequences, the different clades in the tree are reflected in corresponding blocks of the attention map suggesting indeed the capability of the model to capture phylogenetic information.

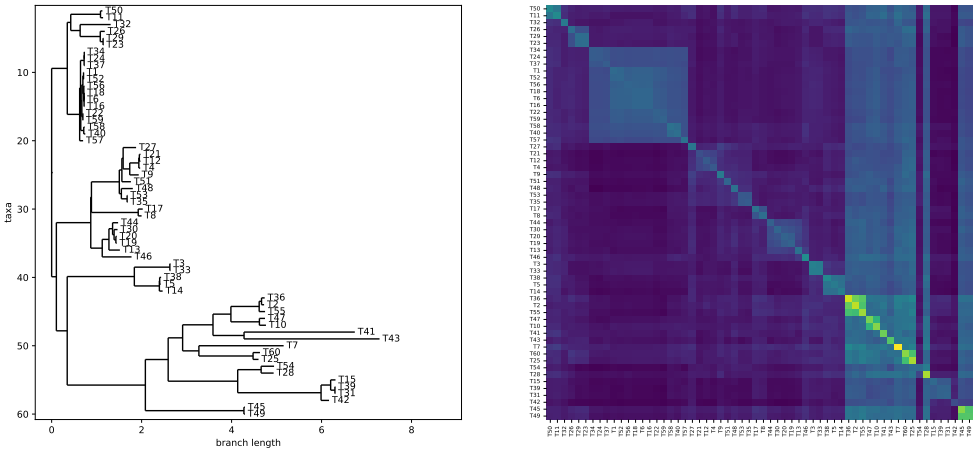


FIGURE 3.13: 60-leaves phylogenetic tree on the left and the first row attention map (averaged over the 4 heads and the 250 sites) of Deepelican during inference on the alignment simulated along corresponding tree on the right.

Despite the undeniable advantages of the presented approach it is important to discuss its limitations, figure 3.11 suggests that the model, when run on empirical data, could be prone to output a high proportion of false positives, this behaviour likely stems from the poorly chosen way in which we dealt with class imbalance during training, this however is straightforward to adjust and hopefully will lead the model to exhibit better performances. Whereas in [174] the authors show that Pelican's performances improve with the tree length, an exhaustive assessment of the parameters which affect our model's performances the most hasn't been carried out yet and could better pinpoint the most advantageous use-cases for the method.

Furthermore, regardless the promising performances on empirical data, the experiments have shown that the model is somewhat sensitive to differences between training and testing data, the analyses in subsection 3.4.4 in particular show how the performances of Deepelican can be affected by out of distribution trees and thus underline a limitation in the tree simulation protocol suggesting that an improvement of the latter could lead to a corresponding improvement of the performances of Deepelican and possibly Phyloformer. Concerning the

simulation of the alignments a lot can still be improved, although reasonable values were chosen, no big effort was made to make the proportion of positive sites and the deletion rates as realistic as possible. Similarly the simulations could be improved by sampling different values for the ρ parameter which ideally would again be chosen from a distribution that fits well empirical data. Finally, the choice of setting the expected number of trait changes from the root to the farthest leaf to 2 is somewhat arbitrary and could be improved in a similar fashion.

All the aforementioned limitations can partly explain the worse performances of Deepelican with respect to Pelican in the gene enrichment experiment in subsection 3.4.6, another perspective to improve the capabilities of the method in detecting relevant genes would be to explore the relevant literature to come up with a more suitable probability aggregation method than the naive adaptation of Pelican's gene-wise truncated Fisher score. It will also be interesting to perform the same analysis with respect to the other phenotypes that have been considered in [193].

Also, given the great performances of the presented method, it would be enticing to compare it as well to the other best performing methods considered in [174], namely codeml and Diffsel, as the results presented here suggest that our method could attain the state of the art across a wide range on evolutionary scenarios.

Finally, a possible extension of the method to continuous traits and to multiple discrete traits is definitely worth to be explored. Whereas dealing with the first case could be straightforward as a continuous value can be used in the encoding 3.5, as it is done in [199], dealing with a non-binary discrete phenotype may require a different encoding strategy in the case in which imposing an ordering of the encoded phenotypes is not desirable.

Chapter 4

DaNaiDeS

4.1 Introduction

The aim of the present chapter is to present some preliminary results of an ongoing collaboration with Charlotte West who is currently pursuing her PhD with Nick Goldman at the European Bioinformatics Institute in Cambridge. The problem that she set to tackle during her thesis is the identification of genes containing sites subject to positive selection, which, as we already discussed in the previous chapter can be characterized by a ratio of non-synonymous to synonymous substitutions $\omega = d_N/d_S > 1$. Given that likelihood-based, state of the art methods for the task are computationally very demanding, this problem has been again framed as a prediction task in a supervised learning simulation-based inference, with the development of OmegaAI [208], a convolutional neural network which takes in input a nucleotide MSA and outputs a single binary prediction, to classify whether the alignment contains sites (codons) which have been subject to positive selection or not. Here we shall present DaNaiDeS, a deep neural network essentially based on the same architecture as Phyloformer and Deepelican, adapted to deal with the same problem as OmegaAI. The head-to-head comparison between the two models will allow us to further discuss the advantages of the architecture presented in this work.

4.1.1 OmegaAI and DaNaiDeS

OmegaAI is a convolutional neural network with an architecture based on that presented in [98], it comprises of six convolutional layers which gradually reduce the size of the representation of the input MSA through convolutions and average pooling operation, a final global pooling reduces this representation to a single vector which is fed into a fully connected layer with a single output feature, representing the network's prediction. The input MSA is represented via a one-hot encoding of the characters in the alphabet $\{A, C, G, T, -\}$. It is worth noting that, whereas the final global pooling allows the network to process alignments with different numbers of sites, the first convolutional filter in the network with size $(m, 3)$ (figure 4.1, right), constrains the number of sequences in input to be m, the same number the model has been trained with, making the applicability of the model to smaller or bigger alignments less straightforward, the authors in [208] therefore fixed the value of m to 8 and did not explore the capabilities of the model to deal with more sequences.

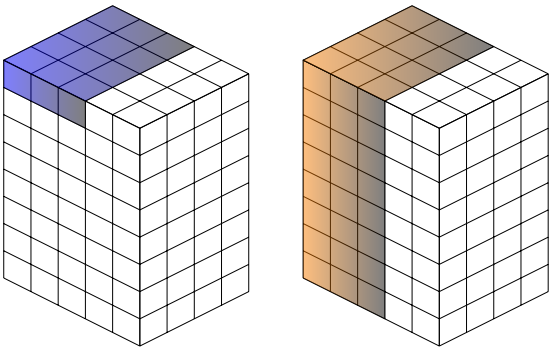


FIGURE 4.1: Convolutions employed by the OmegaAI (right) and DaNaiDes (left) models to embed the initial one-hot encoded MSA representation. Whereas a a convolutional filter of size $(m, 3)$ is used in the OmegaAI model, DaNaiDeS's $(1, 3)$ filter allows for generalisability to alignments with different numbers of sequences and guarantees the invariance of the network's predictions to permutations of sequences in the input MSA.

We have seen on the other hand that an architecture as that of Phyloformer or Deepelican allows handling alignments of varying size (along both dimensions) seamlessly, we then chose the architecture of DaNaiDeS to closely mirror that of Deepelican. A minor adaptation was warranted as Deepelican provides a per-site prediction whereas here we want the network to output a single per-alignment predicted value, this was accomodated simply adding an additional average over the sites step[1] before the final average over the sequence representations. The only other difference between the two networks is then the initial embedding of the one-hot representation, whereas Deepelican and Phyloformer processed the vector encodings of amino acids independently via a convolution with stride $(1, 1)$, for DaNaiDeS we chose a stride of $(1, 3)$ in order for the model to embed codons (figure 4.1, left), effectively dividing by 3 the size of the input representation along the sites dimension. Finally, the DaNaiDeS models presented in this chapter are a bit smaller than the ones we've seen in the previous one, having 6 attention blocks (each with 4 attention heads, using the same rank-1 variant of the linear attention mechanism employed in the previous chapters) and working with an embedding dimension $d = 32$, the total number of trainable parameters in these models then amounts to 80,321 (whereas for comparison OmegaAI has 2,643,841 of them).

It is worth noting that, as it was the case for Deepelican, both OmegaAI and DaNaiDeS, being trained end-to-end to detect the presence of positive selection solely from an MSA, grant direct inference without having to rely on a phylogenetic tree, the latter is on the other hand warranted by the state of the art, maximum likelihood codeml program, implemented in the PAML software [195], to which the authors of [208] compare the performances of the OmegaAI method.

4.1.2 Simulations and training

In the present work we solely relied on the same datasets used to train and test the OmegaAI models in order to provide a fair comparison of the two neural network architectures. All alignments were simulated along a symmetric ultrametric tree with 8 leaves where all the branches share the same length,

[1] As ultimately it's the maximum value of ω in the input alignment which matters for the prediction, a different aggregation strategy, taking the maximum across the sites instead of the average, has been explored, this however lead to sligthly worse performances.

further referred to as divergence, δ (figure 4.2). The same divergence value was used inside each different testing and training dataset with the sole exception of the mixed training dataset we shall consider in subsection 4.2.2.

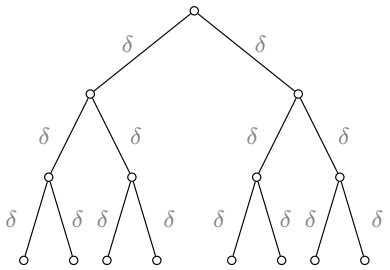


FIGURE 4.2: Tree along which all the training and testing MSAs were simulated. A single divergence value $\delta \in \{0.1, 0.2, \ldots, 0.9, 1\}$ was used in each different training and testing dataset with the exception of the mixed training dataset considered in subsection 4.2.2.

The model of evolution employed to simulate MSAs, of varying lengths, along such trees is a codon-based model of evolution [209], implemented in INDELIBLE [210]. The model allows among site variation of selective pressure, modeling each codon site as belonging to one of three classes, each with a corresponding value of ω sampled from a given distribution. The label of the alignment to predict will be then given by the distribution chosen for the sites belonging to the third class: These are simulated using an $\omega > 1$ value if the MSA represents an example of positive selection and with an $\omega \leq 1$ value otherwise. An equal proportion of MSAs belonging to each class was simulated for each training and testing dataset. Insertion and deletions events were simulated as well, setting a fixed value of 1 for the ratio of insertion to deletions and a fixed value of 0.1 for the ratio of indel to substitution events. The specifics of the simulation procedure can be found in [208] to which we refer the reader for further details. To assess the capabilities of the OmegaAI models to handle misalignment errors that often are found in empirical data, the authors removed the gaps from the simulated sequences and realigned them with a commonly employed aligner. The Clustal Omega [211] aligner was used to realign the sequences in the training datasets whereas MAFFT, PRANKaa and PRANKc [212] were additionally employed for the testing datasets. 1 million samples were generated for each training dataset and 2 thousands for each testing one.

The two DaNaiDeS models which we shall present in this chapter were both trained with a fixed learning rate of 10^{-4} using the Adam optimizer and a batch size of 40 on a single NVIDIA RTX A4000 GPU with 16GB of VRAM. As OmegaAI and Deepelican the models were optimized under a BCE loss. Both models presented in subsections 4.2.1 and 4.2.2 were trained on 1M alignments with varying sequence lengths, ranging from 306 to 3945 nucleotides, with a mean of about of about 1000, using a validation dataset of 10k alignments, for 18 and 23 epochs respectively. Working with different lengths, as equal dimensions are warranted for batched inputs to a neural network, the alignments were grouped by the result q of their integer division by 102 and each of their one-hot encoding representation was then zero-padded to have length $102q + 102$ along the sites dimension.

4.2 Results

4.2.1 Baseline divergence

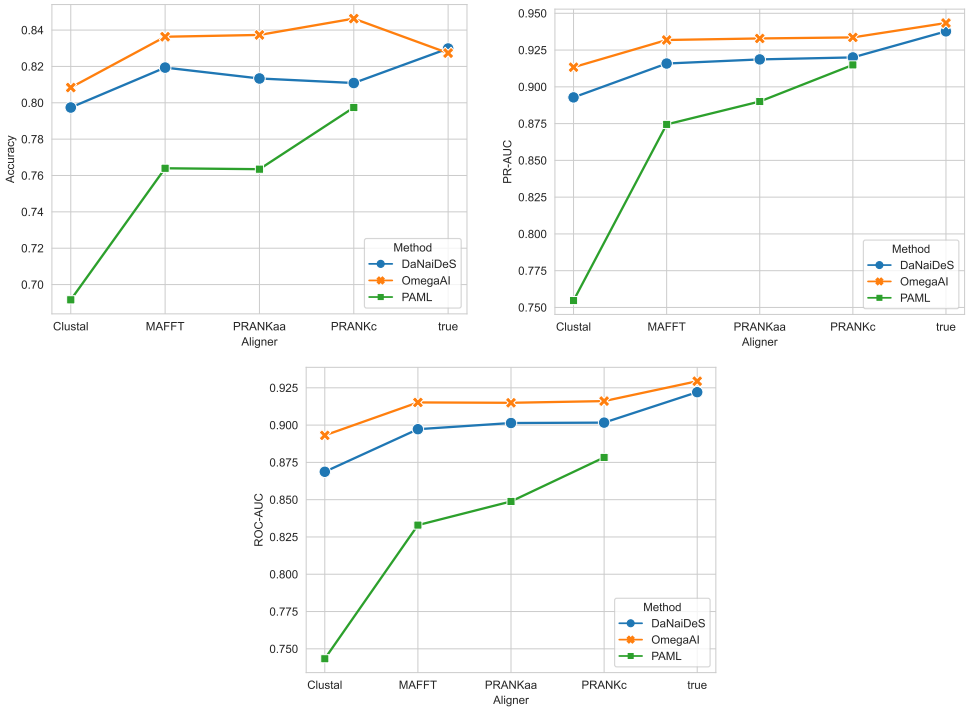


FIGURE 4.3: Performances of DaNaiDeS, OmegaAI and PAML on the 2k test dataset simulated with divergence $\delta = 0.2$ (the same used for the training data of the two NN models). The performances are assessed on MSAs obtained realigning the sequences with different aligners (sorted by their accuracy along the x axis), the results on the true (not realigned) alignments directly issued from the simulator are shown as well for the two neural networks.

While the authors in [208] trained and tested a different OmegaAI model for each value of divergence $\delta \in \{0.1, 0.2, \ldots, 0.9, 1\}$, in this subsection we shall only consider the performances of DaNaiDeS trained and tested on alignments simulated with the value that they chose as a baseline, namely $\delta = 0.2$. The generalisation to different values on the other hand will be discussed in the next subsection. In figure 4.3 we compare, on a test dataset generated with this baseline value for the parameter, the performances of three methods: OmegaAI and DaNaiDeS, both trained on alignments with the same divergence, and PAML. The performances are assessed here in terms of three metrics, accuracy, PR-AUC, and the area under the receiver operating characteristic curve ROC-AUC. While both DaNaiDeS and OmegaAI where trained exclusively on sequences realigned with Clustal, all methods were tested on the same dataset obtained using different aligners, with the results on the true (not realigned) alignments being shown as well for the two neural networks. It is worth noting that here PAML is given the true underlying simulation tree in input. Looking at the figure we can observe that the performances of all methods improve with the quality of the input alignment (having sorted the different aligners along the x axis according to their tendency to produce alignment errors [208]). While DaNaiDeS slightly underperforms with respect to OmegaAI here its performances remain distinctly superior to those of PAML, whose predictive

capability appears to be markedly more sensitive to the quality of the aligned MSA with respect to that of the neural networks. In [208] the authors argue that, beyond the practical advantage of using the faster and less resource intensive Clustal aligner for training, training the model with higher amounts of misalignment may improve the model's robustness without affecting its capability to leverage the quality of the input alignment, for instance they show that using instead the more accurate PRANKc for the training dataset leads to a model which is less accurate when faced with Clustal alignments while its performance gain on PRANKc alignments is marginal.

4.2.2 Mixed divergences

As in [208] we set to test the generalisibility of the network to different divergence values. To do so we trained DaNaiDeS on a mixed dataset, consisting again of 1M alignments with corresponding labels, this time simulated with different values $\delta \in \{0.1, 0.2, \ldots, 0.9, 1\}$, sampled in equal proportions. We did not resort to fine-tuning but retrained the model from scratch in order to provide a fair comparison between the DaNaiDeS model and the OmegaAI one trained on the same dataset. Figure 4.4 shows the performances of this model, DaNaiDeS mixed, across different divergence values and using different aligners for the test MSAs, compared against those of OmegaAI mixed (trained on the same dataset), OmegaAI (a different model, trained specifically on 1M alignments simulated with δ, shown for each divergence value δ), and PAML. In the figure we can observe that here the two neural networks trained on the mixed divergence dataset show a strikingly different behaviour, on one hand the performances of DaNaiDes mixed decrease with the increasing amounts of divergence in the test sets, as naturally higher divergence values make the problem harder, with PAML and OmegaAI showing the same trend, on the other hand OmegaAI mixed appears to loose most of the predictive capabilities of OmegaAI, in terms of accuracy, outside the range of divergences $\{0.6, 0.7, 0.8\}$, with a surprising peak in performances corresponding to $\delta = 0.7$. Looking at the PR-AUC and ROC-AUC values one can see that the drop in performances of the model is not that drastic for lower divergence values in terms of these metrics indicating that a different threshold value, instead of the usual 0.5, to classify an alignment as positive would be warranted here. For DaNaiDeS mixed the phenomenon can be observed to a certain extent as well, whereas in the previous subsection the accuracy of the model increased along with that of the aligner, here the DaNaiDes mixed model, in terms of accuracy, appears to be performing worse as the quality of the alignment increases, with its accuracy even dropping to 0.5 for the dataset with divergence $\delta = 0.7$ realigned using PRANKc, whereas the PR-AUC and ROC-AUC values do show a performance increase which generally goes along with the quality of the input alignments. Further investigation is required to pinpoint the reasons behind this phenomena.

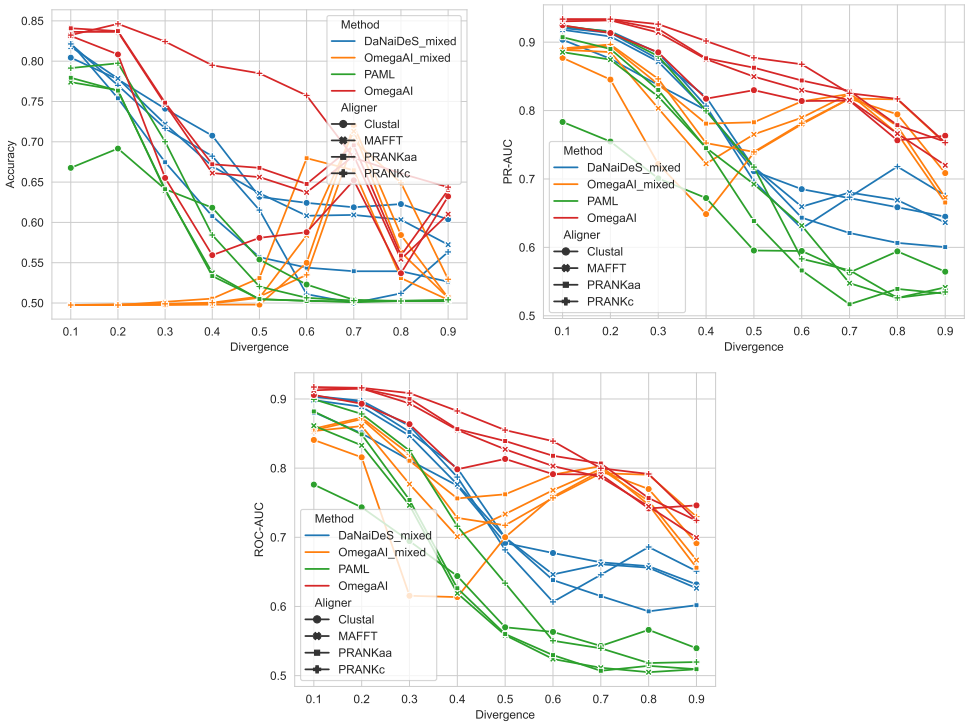


FIGURE 4.4: Performances in terms of accuracy, PR-AUC and ROC-AUC of the mixed OmegaAI and DaNaiDeS models on alignments with different divergences (2k per divergence) and obtained using different aligners. The results for PAML and OmegaAI (a different model, trained specifically on 1M alignments for each divergence) are shown as well for reference.

Given the very different behaviour of the compared neural network-based approaches, whether DaNaiDeS has a better capability to handle different divergence values is debatable. Nevertheless we can observe that, unlike those of OmegaAI mixed, the performances of the DaNaiDeS mixed model remain above those of PAML across all divergences and according to all considered metrics, naturally decreasing as higher divergences make the problem harder.

4.2.3 Advantages of permutation invariance

We already discussed the advantages of exploiting the symmetries of the underlying problem via invariant or equivariant transformations of the input (subsection 1.6.6). The direct comparison between DaNaiDeS and OmegaAI allows us here to showcase this with a concrete example, indeed, whereas the DaNaiDeS neural network architecture is structurally invariant to permutations of the input MSA, that is not the case for OmegaAI which breaks the invariance through the application of a convolutional filter of size (8×3) to the initial one-hot encoded input (figure 4.1). In figure 4.5, we show the performances, in terms of accuracy of DaNaiDeS and OmegaAI when the two models are tested on a dataset generated under the exact same parameter settings under which the models have been trained on, but in which the rows of the input MSAs have been randomly shuffled before feeding them into the models.

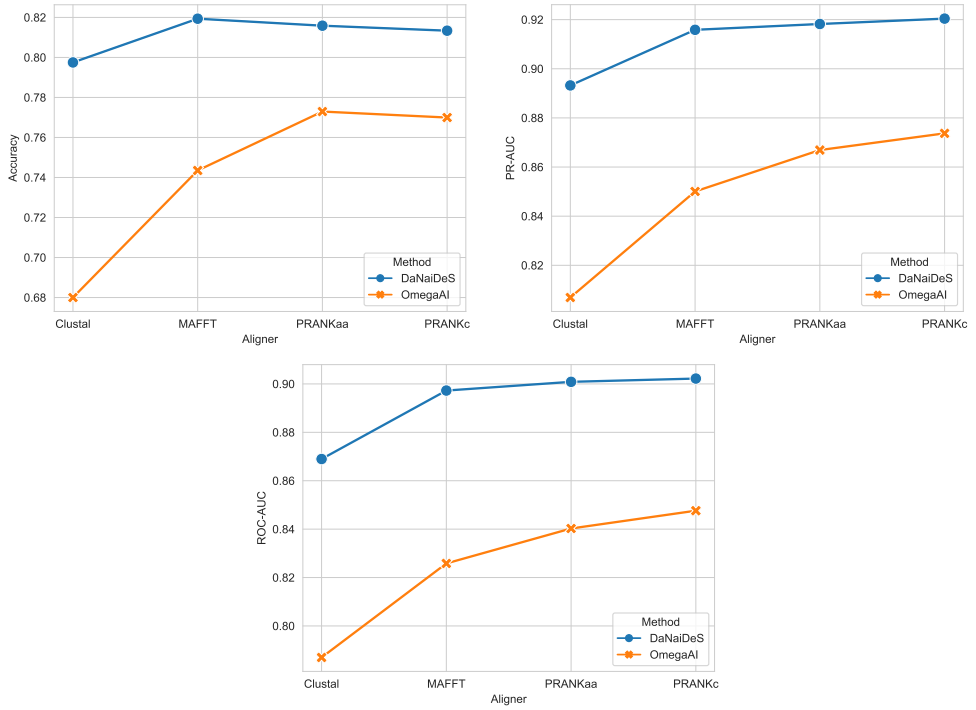


FIGURE 4.5: Performances for different aligners of the DaNaiDeS and OmegaAI models when tested on a dataset (2k alignments, $\delta = 0.2$) in which the rows of the alignments have been randomly shuffled before being processed by the networks.

As expected, no noticeable difference can be observed between the performances of DaNaiDes here with respect to those showcased (figure 4.3) on aligments simulated in the same way but without shuffling the sequences inside each MSA. OmegaAI on the other hand suffers from the sequence reordering with its accuracy now falling below that of DaNaiDes. This suggests that the OmegaAI network has learned to rely on the order of the input sequences, partially reflecting the structure of the underlying tree, during training, failing then to generalise when this structure is not provided. The authors in [197] furthermore show that a similar drop in performances on the testing dataset can be observed when the OmegaAI model has been trained solely on shuffled MSAs which do not directly provide information regarding the underlying tree topology along which the sequences have evolved. This suggests that, to attain the same performances as those of the model trained and tested without shuffling the sequences, such a training dataset would have to be enlarged either by simulating more MSAs or via data augmentation, enriching the dataset with several different permutations of the sequences in the alignments. The structural invariance of DaNaiDeS with respect to these permutations then shows an undeniable advantage of its architecture, allowing for a better training sample efficiency and permitting its direct applicability, without any predictive capability loss, when no information regarding the tree topology is explicitly provided.

4.2.4 Interpretability

Whereas inspecting the row attention maps of the DaNaiDeS models presented in this chapter one can again identify the structure of the underlying tree, as it was the case for Deepelican, for the problem tackled here it is more interesting to look at the model's column attention maps.

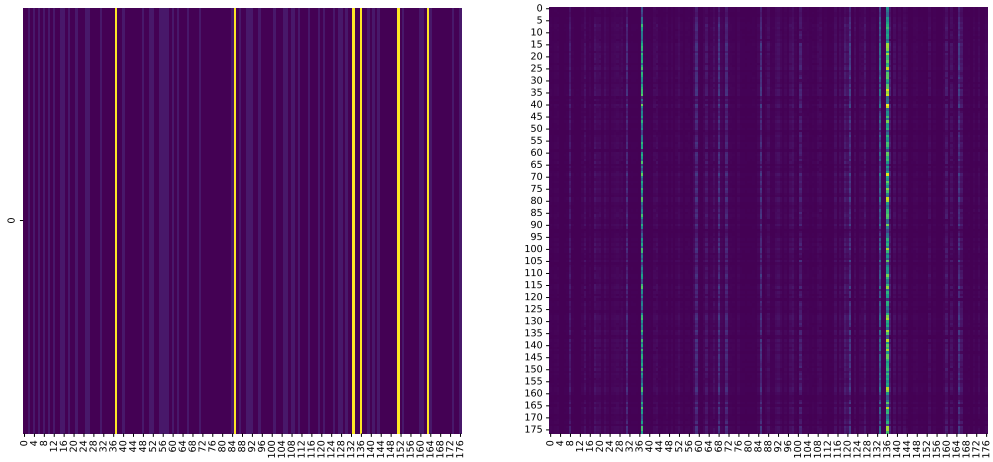


FIGURE 4.6: Example of interpretability of the column attention maps of DaNaiDeS, on the left we show the heatmap of the per-codon ω values used to simulate an alignment, on the right a column attention map of DaNaiDeS (corresponding to the first head of the second attention block, averaged over all 8 sequences) obtained feeding the alignment to the model. We can see that the model tends to pay more attention to the columns of its internal representation (corresponding to triplets of columns in the input nucleotide MSA), for which the corresponding true ω values are the highest: In this particular case the phenomenon is apparent for codons 37, 132 and 136 and can be observed to a lesser extent for codon 85.

Indeed, as one can see in figure 4.6, these attention maps can reveal the codons in the input alignments which have undergone positive selection, which the model learns to identify to ultimately predict whether the alignment contains any or not. Such attention values could then be used as a proxy to provide per-site predictions without explicitly adapting the network's architecture and retraining the model for the task. This visualisation is similar to the one presented in [208] where the authors show that codons with high ω values correlate with columns with high values in the OmegaAI's saliency map. Although the results are similar, the way these visualisations are obtained is fundamentally different: Whereas saliency maps are derived computing the gradient of the model's output with respect to the input, the attention maps presented here can be computed directly during the forward pass. This makes them rather analogous to the feature maps of the convolutional layers in the OmegaAI model, the key difference here is that the DaNaiDeS model preserves the spatial structure and dimensions of the input throughout all its layers, making such maps more easily interpretable. Furthermore, the saliency maps in [208] reveal that the sites towards the right end of the input MSAs tend to influence the OmegaAI model's prediction to a significantly lesser extent, it is argued then that this could be attributed to the way the model is trained, zero-padding the sequences to the right end to accomodate variable lengths when batching inputs: This could induce the model to learn to rely less on the positions towards the right end of the MSA, as these positions will typically

contain many zeros and will thus be less informative for the network's prediction. We can point out here that, although an analogous strategy is employed to train the DaNaiDeS models, the network being invariant to permutations of the codons (triplets of adjacent columns in the MSA), it exploits no positional information and thus cannot learn to rely less or more on different regions in the input alignment for its predictions.

4.3 Discussion and future perspectives

The preliminary results presented in this chapter demonstrate once again the flexibility provided by the developed neural network architecture to deal with MSA data across different tasks. In particular we have seen here how the DaNaiDeS model can effectively be employed for the problem of per-alignment positive selection detection, discussing its advantages with respect to a previously developed NN architecture, namely its generalisibility to MSAs of arbitrary size along both dimensions and its invariance to permutations of sequences inside the MSA. While the results in subsection 4.2.1 show the model slightly underperforming with respect to the convolutional network we compared it to, its performances remain competitive showing a clearcut advantage with respect to the state of the art, maximum-likelihood-based approach. Furthermore we believe that a more extensive hyperparameter search, beyond the limited one that had been carried out at this stage of the project, could provide an improvement of said performances. The results of a hyperparameter search carried out on a subset of the training data for instance hint to the fact that in this case the full linear attention mechanism may be better performing with respect to the rank-1 variant we employed, providing more expressivity which may be warranted by the problem at hand.

Another research direction worthy of further exploration is whether positional encoding can help improve the performances of the model. Indeed, while the employed model of substitution entails that the evolution of each codon is independent and identically distributed, again the introduction of indel events breaks the symmetry of the simulation process. This entails that a model such as DaNaiDeS, invariant to permutations of the input codons cannot exploit the potential information provided by adjacent gaps in the alignment and could benefit from some positional information. Some initial experiments suggested that this was indeed the case: A model trained with an initial convolution with stride $(1, 1)$ instead of $(1, 3)$, embedding thus all adjacent triplets of nucleotides in the input alignment instead of just the codons in the correct reading frame, seemed to provide slightly better performances, this however came with the computational overhead of working with a three times longer representation of the data. To avoid such an overhead a model working with a stride of $(1, 3)$ but with a sinusoidal positional encoding added to the codon embeddings was trained, again such a model showed marginally better performances with respect to those presented in this chapter, nevertheless, when the model was tested shuffling the codon positions no drop in performances was observed indicating thus that the model has not learned to rely on such positional information for its predictions. Further investigations are therefore required to pinpoint whether a model which actually exploits such information could be advantageous.

As of the time of writing an approach to predict per-site ω values, using an architecture such as that of DaNaiDeS, framing thus the problem as a per-site regression one, is being explored. It will be interesting to compare its prediction with respect to those that one could obtain fitting a linear regressor on the values of the column attention maps shown in the previous subsection (as it has been done e.g. in [145] to predict distances between sites in the 3D structure of a protein from the attention maps of the MSA transformer model, or to predict Hamming distances [213] from the same attention maps).

Finally, the generalisability to different numbers of sequences provided by the DaNaiDeS model allows training and testing it on a wider range of tree simulations. More realistic simulations, beyond the simple 8 leaf tree used so far, as those we have employed in the previous chapters, can be used to make the model more robust and applicable to empirical data.

Chapter 5

Conclusion

In this work I explored the promises offered by deep learning and simulation based inference in the context of phylogenetic reconstruction and for related tasks in the field of molecular evolution. In chapter 2 we showed how the proposed method provides more accurate predictions with respect to existing distance-based approaches in all considered experiments. Moreover, the method shows the potential to outperform resource-intensive maximum-likelihood methods under more complex evolution models with a fraction of the computational cost, paving the way to the adoption of more realistic models of molecular evolution for inference. The developed neural network architecture allows dealing with MSAs of varying sizes, exploiting the symmetries of the problem at hand, and in chapters 3 and 4 I've shown that the architecture is effectively adaptable to other inference tasks concerning sequences which share an evolutionary history, again outperforming costly maximum-likelihood methods.

I will conclude this manuscript briefly discussing some potential future work directions.

Self supervised pretraining vs end-to-end learning

Most commonly transformers models such as ProtBERT [79] and ESM [95] are pretrained on a large corpus of unlabeled data via self-supervised learning with a task as masked token prediction before being fine-tuned for the target task. This is typically done as for the target task not enough labeled data may be available which would allow to train the models from scratch directly for it. In the scenarios we considered throughout this work however this is not the case, as in the context of SBI, simulations make the amount of available labeled data virtually infinite. Studies show that the benefits of pretraining diminish with the increase of task-specific data [214] and that when the latter is abundant direct end-to-end training of a model may be preferable as it avoids the resource-intensive pretraining step, while such a step could even be counterproductive as the prior learned during pretraining may later hurt the performances of the model on the downstream task [215]. However, the goal ultimately is being able to use networks as those presented in this work for accurate inference on empirical data, not simulations. Despite the efforts to make the simulations realistic a gap between these and real data will necessarily remain. It is not clear then if including a pretraining phase for

such models, using real alignments, could lead them to learn representations which are more effective in capturing the complexity of empirical data, such a phase then may potentially have a regularization effect preventing the models from overfitting features which are simulation-specific. Besides, given the showcased adaptability of the proposed neural network to different tasks related to molecular evolution, it is not to exclude that a pretrained model could be useful to transfer knowledge across different domains. In general to bridge the simulations-to-real gap it would be worth exploring different domain adaptation techniques, with some of them having recently found success in the context of population genetics [216][217].

On the fly simulations

A strategy that hasn't been fully explored in this work but which may be adopted for future endeavors is that of on the fly simulation, namely training networks such as those presented in this work on freshly simulated data points at each step instead of using a fixed training dataset for several epochs, this has better statistical guarantees [85] and removes the risk of overfitting the training data altogether. Of note, despite the use of a fixed training dataset, the fine-tuning of the PF_{Cherry} model actually fits into this framework, with the model having been trained for 0.72 epochs, effectively never seing the same training data point twice, while that of PF_{SelReg} model comes close to this having been trained for only 1.58 epochs (table 2.1).

Measuring uncertainty in the predictions

The Phyloformer method presented in chapter 2, provides only point estimates of the evolutionary distances without any measure of uncertainty. A way to obviate this limitation would be to follow the approach discussed in [132] and [133], modifying the regression task to predict at once the mean and variance of a posterior distribution instead of simple point estimates. Beyond the interest in itself of having such a measure of uncertainty, which could also aid the interpretation of the performances of the model, having an estimate of the variance in the predictions could potentially improve its performances via phylogenetic reconstruction methods such as weighted least squares (subsection 1.4.3) or BIONJ, which exploit the variance in the distance estimates to provide more accurate predictions. I shall mention another, more direct, tempting approach that could have been taken and may be considered in the future. The final vector of predicted distances provided by the Phyloformer network is given by the average over the sites dimension of the final representation of the input data, it is tempting to think then that each column of such a representation represents a per-site estimate of the distance vector to be predicted, however unfortunately if one looks at all these per site "predictions" of a given distance, their distribution does not seem to reflect any site-specific information, with many values around 0 and several others way above the true distance, compensating the former in order for the average to be a good estimate. A strategy that could have been adopted when training the model would have been taking as its predictions, instead of the average over all sites, an average over a subset of the latter, randomly sampled at each time, which would have then forced the model to provide meaningful estimates at each position. The variance of the distribution of these values across the sites could

have been then interpreted as a measure of uncertainty for the corresponding predicted distance.

Alignment-free phylogenetic reconstruction

The alignment of homologous sequences, an NP-hard problem in itself [218], is a costly and error-prone step therefore extending the method presented here to be able to deal with unaligned sequence would provide a great advantage. The approach holds promises as in principle the computation of evolutionary distances does not necessarily require the sequences to be aligned, with several alignment free distance-based phylogenetic reconstruction methods, often based on k-mer frequencies, having been proposed in recent years [219][220][48]. Moreover we have seen how an approach as ours can allow direct inference for different tasks bypassing steps, such as the reconstruction of a phylogenetic tree, which are usually required by other methods, it is tempting then to assess whether this could be the case for the alignment step as well. The most straightforward way to implement this would simply be providing to a model such as Phyloformer the unaligned sequences in input instead of an MSA, padding the shorter ones with zeros so that each sequence representation has the same length, and adding positional information via positional encoding. Whether this would be effective is to be assessed and some modifications to the architecture may be warranted but I feel this is definitely an approach worth to be explored.

Bibliography

[1] J. Felsenstein and J. Felenstein, *Inferring phylogenies*. Sinauer associates Sunderland, MA, 2004, vol. 2.

[2] W. M. Fitch, "Toward defining the course of evolution: Minimum change for a specific tree topology," *Systematic Biology*, vol. 20, no. 4, pp. 406–416, 1971.

[3] W. H. Day and D. Sankoff, "Computational complexity of inferring phylogenies by compatibility," *Systematic Biology*, vol. 35, no. 2, pp. 224–229, 1986.

[4] L. R. Foulds and R. L. Graham, "The steiner problem in phylogeny is np-complete," *Advances in Applied mathematics*, vol. 3, no. 1, pp. 43–49, 1982.

[5] J. Felsenstein, "Cases in which parsimony or compatibility methods will be positively misleading," *Systematic zoology*, vol. 27, no. 4, pp. 401–410, 1978.

[6] R. Nielsen, *Statistical methods in molecular evolution*. Springer, 2006.

[7] J. Felsenstein and G. A. Churchill, "A hidden markov model approach to variation among sites in rate of evolution.," *Molecular biology and evolution*, vol. 13, no. 1, pp. 93–104, 1996.

[8] C. L. Kleinman, N. Rodrigue, N. Lartillot, and H. Philippe, "Statistical Potentials for Improved Structurally Constrained Evolutionary Models," *Molecular Biology and Evolution*, vol. 27, no. 7, pp. 1546–1560, Jul. 2010.

[9] D. Posada and K. A. Crandall, "Modeltest: Testing the model of dna substitution.," *Bioinformatics (Oxford, England)*, vol. 14, no. 9, pp. 817–818, 1998.

[10] D. Darriba, G. L. Taboada, R. Doallo, and D. Posada, "Prottest-hpc: Fast selection of best-fit models of protein evolution," in *Euro-Par 2010 Parallel Processing Workshops: HeteroPar, HPCC, HiBB, CoreGrid, UCHPC, HPCF, PROPER, CCPI, VHPC, Ischia, Italy, August 31–September 3, 2010, Revised Selected Papers 16*, Springer, 2011, pp. 177–184.

[11] S. Abadi, D. Azouri, T. Pupko, and I. Mayrose, "Model selection may not be a mandatory step for phylogeny reconstruction," *Nature communications*, vol. 10, no. 1, p. 934, 2019.

[12] L. Shavit Grievink, D. Penny, M. D. Hendy, and B. R. Holland, "Phylogenetic tree reconstruction accuracy and model fit when proportions of variable sites change across the tree," *Systematic biology*, vol. 59, no. 3, pp. 288–297, 2010.

[13] T. R. Buckley, "Model misspecification and probabilistic tests of topology: Evidence from empirical data sets," *Systematic Biology*, vol. 51, no. 3, pp. 509–523, 2002.

[14] Z. Yang, "Maximum-likelihood estimation of phylogeny from DNA sequences when substitution rates differ over sites.," *Molecular biology and evolution*, vol. 10, no. 6, pp. 1396–1401, 1993.

[15] Z. Yang, "Maximum likelihood phylogenetic estimation from dna sequences with variable rates over sites: Approximate methods," *Journal of Molecular evolution*, vol. 39, pp. 306–314, 1994.

[16] E. Susko, C. Field, C. Blouin, and A. J. Roger, "Estimation of rates-across-sites distributions in phylogenetic substitution models," *Systematic biology*, vol. 52, no. 5, pp. 594–603, 2003.

[17] F. Jia, N. Lo, and S. Y. Ho, "The impact of modelling rate heterogeneity among sites on phylogenetic estimates of intraspecific evolutionary rates and timescales," *PLoS One*, vol. 9, no. 5, e95722, 2014.

[18] W. P. Maddison, "Gene trees in species trees," *Systematic biology*, vol. 46, no. 3, pp. 523–536, 1997.

[19] B. Q. Minh, H. A. Schmidt, O. Chernomor, *et al.*, "IQ-TREE 2: New Models and Efficient Methods for Phylogenetic Inference in the Genomic Era," eng, *Molecular Biology and Evolution*, vol. 37, no. 5, pp. 1530–1534, May 2020.

[20] X. Zhou, X.-X. Shen, C. T. Hittinger, and A. Rokas, "Evaluating fast maximum likelihood-based phylogenetic programs using empirical phylogenomic data sets," *Molecular Biology and Evolution*, vol. 35, no. 2, pp. 486–503, Feb. 1, 2018.

[21] N. Ly-Trong, S. Naser-Khdour, R. Lanfear, and B. Q. Minh, "AliSim: A Fast and Versatile Phylogenetic Sequence Simulator for the Genomic Era," *Molecular Biology and Evolution*, vol. 39, no. 5, msac092, May 2022.

[22] P. Buneman, "A note on the metric properties of trees," *Journal of combinatorial theory, series B*, vol. 17, no. 1, pp. 48–50, 1974.

[23] T. H. Jukes, C. R. Cantor, *et al.*, "Evolution of protein molecules," *Mammalian protein metabolism*, vol. 3, no. 24, pp. 21–132, 1969.

[24] N. Saitou and M. Nei, "The neighbor-joining method: A new method for reconstructing phylogenetic trees.," *Molecular biology and evolution*, vol. 4, no. 4, pp. 406–425, 1987.

[25] J. A. Studier and K. J. Keppler, "A note on the neighbor-joining algorithm of saitou and nei.," *Molecular biology and evolution*, vol. 5, no. 6, pp. 729–731, 1988.

[26] D. Bryant, "On the uniqueness of the selection criterion in neighbor-joining," *Journal of Classification*, vol. 22, no. 1, pp. 3–15, 2005.

[27] K. Atteson, "The performance of neighbor-joining methods of phylogenetic reconstruction," *Algorithmica*, vol. 25, no. 2, pp. 251–278, 1999.

[28] R. Mihaescu, D. Levy, and L. Pachter, "Why neighbor-joining works," *Algorithmica*, vol. 54, no. 1, pp. 1–24, 2009.

[29] T. Mailund, G. S. Brodal, R. Fagerberg, C. N. Pedersen, and D. Phillips, "Recrafting the neighbor-joining method," *BMC bioinformatics*, vol. 7, no. 1, pp. 1–8, 2006.

[30] I. Elias and J. Lagergren, "Fast neighbor joining," *Theoretical computer science*, vol. 410, no. 21-23, pp. 1993–2000, 2009.

[31] O. Gascuel, "BIONJ: An improved version of the NJ algorithm based on a simple model of sequence data," eng, *Molecular Biology and Evolution*, vol. 14, no. 7, pp. 685–695, Jul. 1997.

[32] M. S. Waterman, T. F. Smith, M. Singh, and W. A. Beyer, "Additive evolutionary trees," *Journal of theoretical Biology*, vol. 64, no. 2, pp. 199–213, 1977.

[33] L. L. Cavalli-Sforza and A. W. Edwards, "Phylogenetic analysis. models and estimation procedures," *American journal of human genetics*, vol. 19, no. 3 Pt 1, p. 233, 1967.

[34] F. Pardi and O. Gascuel, "Distance-based methods in phylogenetics," 2016.

[35] W. M. Fitch and E. Margoliash, "Construction of phylogenetic trees: A method based on mutation distances as estimated from cytochrome c sequences is of general applicability.," *Science*, vol. 155, no. 3760, pp. 279–284, 1967.

[36] W. A. Beyer, M. L. Stein, T. F. Smith, and S. M. Ulam, "A molecular sequence metric and evolutionary trees," *Mathematical Biosciences*, vol. 19, no. 1-2, pp. 9–25, 1974.

[37] D. J. Bryant and P. J. Waddell, "Rapid evaluation of least squares and minimum evolution criteria on phylogenetic trees," 1997.

[38] O. Gascuel, "Bionj: An improved version of the nj algorithm based on a simple model of sequence data.," *Molecular biology and evolution*, vol. 14, no. 7, pp. 685–695, 1997.

[39] K. K. Kidd and L. A. Sgaramella-Zonta, "Phylogenetic analysis: Concepts and methods.," *American journal of human genetics*, vol. 23, no. 3, p. 235, 1971.

[40] A. Rzhetsky and M. Nei, "Statistical properties of the ordinary least-squares, generalized least-squares, and minimum-evolution methods of phylogenetic inference," *Journal of molecular evolution*, vol. 35, pp. 367–375, 1992.

[41] A. Rzhetsky and M. Nei, "Theoretical foundation of the minimum-evolution method of phylogenetic inference.," *Molecular biology and evolution*, vol. 10, no. 5, pp. 1073–1095, 1993.

[42] Y. Pauplin, "Direct calculation of a tree length using a distance matrix," *Journal of Molecular Evolution*, vol. 51, pp. 41–47, 2000.

[43] R. Desper and O. Gascuel, "Theoretical foundation of the balanced minimum evolution method of phylogenetic inference and its relationship to weighted least-squares tree fitting," *Molecular Biology and Evolution*, vol. 21, no. 3, pp. 587–598, 2004.

[44] O. Gascuel and M. Steel, "Neighbor-joining revealed," *Molecular biology and evolution*, vol. 23, no. 11, pp. 1997–2000, 2006.

[45] M. N. Price, P. S. Dehal, and A. P. Arkin, "Fasttree 2–approximately maximum-likelihood trees for large alignments," *PloS one*, vol. 5, no. 3, e9490, 2010.

[46] V. Lefort, R. Desper, and O. Gascuel, "Fastme 2.0: A comprehensive, accurate, and fast distance-based phylogeny inference program," *Molecular biology and evolution*, vol. 32, no. 10, pp. 2798–2800, 2015.

[47] A. Zielezinski, H. Z. Girgis, G. Bernard, *et al.*, "Benchmarking of alignment-free sequence comparison methods," *Genome biology*, vol. 20, pp. 1–18, 2019.

[48] M. Balaban, N. A. Bristy, A. Faisal, M. S. Bayzid, and S. Mirarab, "Genome-wide alignment-free phylogenetic distance estimation under a no strand-bias model," *Bioinformatics Advances*, vol. 2, no. 1, vbac055, 2022.

[49] A. Criscuolo, "A fast alignment-free bioinformatics procedure to infer accurate distance-based phylogenetic trees from genome assemblies," *Research Ideas and Outcomes*, vol. 5, e36178, 2019.

[50] V. M. Sarich and A. C. Wilson, "Immunological time scale for hominid evolution," *Science*, vol. 158, no. 3805, pp. 1200–1203, 1967.

[51] W. Cao, L.-Y. Wu, X.-Y. Xia, X. Chen, Z.-X. Wang, and X.-M. Pan, "A sequence-based evolutionary distance method for phylogenetic analysis of highly divergent proteins," *Scientific Reports*, vol. 13, no. 1, p. 20 304, 2023.

[52] M. Binet, O. Gascuel, C. Scornavacca, E. J. P. Douzery, and F. Pardi, "Fast and accurate branch lengths estimation for phylogenomic trees," *BMC bioinformatics*, vol. 17, pp. 1–18, 2016.

[53] A. Criscuolo, V. Berry, E. J. Douzery, and O. Gascuel, "Sdm: A fast distance-based approach for (super) tree building in phylogenomics," *Systematic biology*, vol. 55, no. 5, pp. 740–755, 2006.

[54] X. Xia, "Imputing missing distances in molecular phylogenetics," *PeerJ*, vol. 6, e5321, 2018.

[55] A. Criscuolo and O. Gascuel, "Fast nj-like algorithms to deal with incomplete distance matrices," *BMC bioinformatics*, vol. 9, pp. 1–16, 2008.

[56] A. Bhattacharjee and M. S. Bayzid, "Machine learning based imputation techniques for estimating phylogenetic trees from incomplete distance matrices," *BMC genomics*, vol. 21, no. 1, p. 497, 2020.

[57] D. F. Robinson and L. R. Foulds, "Comparison of phylogenetic trees," *Mathematical biosciences*, vol. 53, no. 1-2, pp. 131–147, 1981.

[58] M. K. Kuhner and J. Felsenstein, "A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates.," *Molecular biology and evolution*, vol. 11, no. 3, pp. 459–468, 1994.

[59] G. F. Estabrook, F. McMorris, and C. A. Meacham, "Comparison of undirected phylogenetic trees based on subtrees of four evolutionary units," *Systematic Zoology*, vol. 34, no. 2, pp. 193–200, 1985.

[60] B. Efron, "Bootstrap methods: Another look at the jackknife," in *Breakthroughs in statistics: Methodology and distribution*, Springer, 1992, pp. 569–593.

[61] J. Felsenstein, "Confidence limits on phylogenies: An approach using the bootstrap," *evolution*, vol. 39, no. 4, pp. 783–791, 1985.

[62] A. Stamatakis, P. Hoover, and J. Rougemont, "A rapid bootstrap algorithm for the raxml web servers," *Systematic biology*, vol. 57, no. 5, pp. 758–771, 2008.

[63] B. Q. Minh, M. A. T. Nguyen, and A. Von Haeseler, "Ultrafast approximation for phylogenetic bootstrap," *Molecular biology and evolution*, vol. 30, no. 5, pp. 1188–1195, 2013.

[64] B. R. Baum, "Combining trees as a way of combining data sets for phylogenetic inference, and the desirability of combining gene trees," *Taxon*, vol. 41, no. 1, pp. 3–10, 1992.

[65] M. A. Ragan, "Phylogenetic inference based on matrix representation of trees," *Molecular phylogenetics and evolution*, vol. 1, no. 1, pp. 53–58, 1992.

[66] F.-J. Lapointe and G. Cucumel, "The average consensus procedure: Combination of weighted trees containing identical or overlapping sets of taxa," *Systematic Biology*, vol. 46, no. 2, pp. 306–312, 1997.

[67] N. Nguyen, S. Mirarab, and T. Warnow, "Mrl and superfine+ mrl: New supertree methods," *Algorithms for Molecular Biology*, vol. 7, pp. 1–13, 2012.

[68] K. Strimmer and A. Von Haeseler, "Quartet puzzling: A quartet maximum-likelihood method for reconstructing tree topologies," *Molecular biology and evolution*, vol. 13, no. 7, pp. 964–969, 1996.

[69] V. Ranwez and O. Gascuel, "Quartet-based phylogenetic inference: Improvements and limits," *Molecular biology and evolution*, vol. 18, no. 6, pp. 1103–1116, 2001.

[70] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, Jan. 29, 2017.

[71] G. Hinton, *Neural networks for machine learning - lecture 6a: Overview of mini-batch gradient descent*, Coursera Lecture, `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`, 2012.

[72] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, 2011.

[73] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[74] M. Zaheer, G. Guruganesh, K. A. Dubey, *et al.*, "Big bird: Transformers for longer sequences," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[75] Ž. Avsec, V. Agarwal, D. Visentin, *et al.*, "Effective gene expression prediction from sequence by integrating long-range interactions," *Nature Methods*, vol. 18, no. 10, pp. 1196–1203, 2021.

[76] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[77] T. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[78] A. Rives, J. Meier, T. Sercu, *et al.*, "Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences," *Proceedings of the National Academy of Sciences*, vol. 118, no. 15, 2021.

[79] A. Elnaggar, M. Heinzinger, C. Dallago, *et al.*, "Prottrans: Towards cracking the language of life's code through self-supervised deep learning and high performance computing," *arXiv preprint arXiv:2007.06225*, 2020.

[80] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[81] T. Cohen and M. Welling, "Group equivariant convolutional networks," in *International conference on machine learning*, PMLR, 2016, pp. 2990–2999.

[82] F. Fuchs, D. Worrall, V. Fischer, and M. Welling, "Se (3)-transformers: 3d roto-translation equivariant attention networks," *Advances in neural information processing systems*, vol. 33, pp. 1970–1981, 2020.

[83] V. Mallet and J.-P. Vert, "Reverse-complement equivariant networks for dna sequences," *Advances in neural information processing systems*, vol. 34, pp. 13 511–13 523, 2021.

[84] L. Flagel, Y. Brandvain, and D. R. Schrider, "The Unreasonable Effectiveness of Convolutional Neural Networks in Population Genetic Inference," *Molecular Biology and Evolution*, vol. 36, no. 2, pp. 220–238, Dec. 2018.

[85] J. Chan, V. Perrone, J. Spence, P. Jenkins, S. Mathieson, and Y. Song, "A likelihood-free inference framework for population genetic data using exchangeable neural networks," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31, Curran Associates, Inc., 2018.

[86] T. Sanchez, J. Cury, G. Charpiat, and F. Jay, "Deep learning for population size history inference: Design, comparison and combination with approximate bayesian computation," *Molecular Ecology Resources*, vol. 21, no. 8, pp. 2645–2660, 2021.

[87] M. T. Ribeiro, S. Singh, and C. Guestrin, "" why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[88] K. Simonyan, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[89] P. Linardatos, V. Papastefanopoulos, and S. Kotsiantis, "Explainable ai: A review of machine learning interpretability methods," *Entropy*, vol. 23, no. 1, p. 18, 2020.

[90] Q.-s. Zhang and S.-C. Zhu, "Visual interpretability for deep learning: A survey," *Frontiers of Information Technology & Electronic Engineering*, vol. 19, no. 1, pp. 27–39, 2018.

[91] W Samek, "Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models," *arXiv preprint arXiv:1708.08296*, 2017.

[92] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, Springer, 2014, pp. 818–833.

[93] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey, "Predicting the sequence specificities of dna-and rna-binding proteins by deep learning," *Nature biotechnology*, vol. 33, no. 8, pp. 831–838, 2015.

[94] P. K. Koo and S. R. Eddy, "Representation learning of genomic sequence motifs with convolutional neural networks," *PLoS computational biology*, vol. 15, no. 12, e1007560, 2019.

[95] Z. Lin, H. Akin, R. Rao, *et al.*, "Evolutionary-scale prediction of atomic-level protein structure with a language model," *Science*, vol. 379, no. 6637, pp. 1123–1130, 2023.

[96] Y. Ji, Z. Zhou, H. Liu, and R. V. Davuluri, "DNABERT: pre-trained Bidirectional Encoder Representations from Transformers model for DNA-language in genome," *Bioinformatics*, vol. 37, no. 15, pp. 2112–2120, Feb. 2021.

[97] Y. K. Mo, M. W. Hahn, and M. L. Smith, "Applications of Machine Learning in Phylogenetics," *Molecular Phylogenetics and Evolution*, p. 108 066, Mar. 2024.

[98] A. Suvorov, J. Hochuli, and D. R. Schrider, "Accurate Inference of Tree Topologies from Multiple Sequence Alignments Using Deep Learning," *Systematic Biology*, vol. 69, no. 2, pp. 221–233, Sep. 2019.

[99] Z. Zou, H. Zhang, Y. Guan, and J. Zhang, "Deep residual neural networks resolve quartet molecular phylogenies," *Molecular biology and evolution*, vol. 37, no. 5, pp. 1495–1507, 2020.

[100] X. Tang, L. Zepeda-Nuñez, S. Yang, Z. Zhao, and C. Solís-Lemus, "Novel symmetry-preserving neural network model for phylogenetic inference," *Bioinformatics Advances*, vol. 4, no. 1, vbae022, Feb. 2024.

[101] P. Zaharias, M. Grosshauser, and T. Warnow, "Re-evaluating deep neural networks for phylogeny estimation: The issue of taxon sampling," *Journal of Computational Biology*, 2022.

[102] M. L. Smith and M. W. Hahn, "Phylogenetic inference using generative adversarial networks," *Bioinformatics*, vol. 39, no. 9, btad543, Sep. 2023.

[103] Y. Jiang, M. Balaban, Q. Zhu, and S. Mirarab, "DEPP: Deep Learning Enables Extending Species Trees using Single Genes," *Systematic Biology*, syac031, Apr. 2022.

[104] A. Suvorov and D. R. Schrider, "Reliable estimation of tree branch lengths using deep neural networks," *PLOS Computational Biology*, vol. 20, no. 8, e1012337, 2024.

[105] D. Azouri, S. Abadi, Y. Mansour, I. Mayrose, and T. Pupko, "Harnessing machine learning to guide phylogenetic-tree search algorithms," *Nature communications*, vol. 12, no. 1, p. 1983, 2021.

[106] D. Azouri, O. Granit, M. Alburquerque, Y. Mansour, T. Pupko, and I. Mayrose, "The tree reconstruction game: Phylogenetic reconstruction using reinforcement learning," *Molecular Biology and Evolution*, vol. 41, no. 6, 2024.

[107] S. Abadi, O. Avram, S. Rosset, T. Pupko, and I. Mayrose, "Modelteller: Model selection for optimal phylogenetic reconstruction using machine learning," *Molecular biology and evolution*, vol. 37, no. 11, pp. 3338–3352, 2020.

[108] S. Burgstaller-Muehlbacher, S. M. Crotty, H. A. Schmidt, F. Reden, T. Drucks, and A. von Haeseler, "Modelrevelator: Fast phylogenetic model estimation via deep learning," *Molecular Phylogenetics and Evolution*, vol. 188, p. 107 905, 2023.

[109] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The journal of chemical physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[110] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications," 1970.

[111] L. Saul and M. Jordan, "A mean field learning algorithm for unsupervised neural networks," in *Learning in graphical models*, Springer, 1998, pp. 541–554.

[112] M. J. Wainwright, M. I. Jordan, *et al.*, "Graphical models, exponential families, and variational inference," *Foundations and Trends® in Machine Learning*, vol. 1, no. 1–2, pp. 1–305, 2008.

[113] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.

[114] D. B. Rubin, "Bayesianly justifiable and relevant frequency calculations for the applied statistician," *The Annals of Statistics*, pp. 1151–1172, 1984.

[115] M. Sunnåker, A. G. Busetto, E. Numminen, J. Corander, M. Foll, and C. Dessimoz, "Approximate bayesian computation," *PLoS computational biology*, vol. 9, no. 1, e1002803, 2013.

[116] M. A. Beaumont, "Approximate bayesian computation in evolution and ecology," *Annual review of ecology, evolution, and systematics*, vol. 41, no. 1, pp. 379–406, 2010.

[117] K. Csilléry, M. G. Blum, O. E. Gaggiotti, and O. François, "Approximate bayesian computation (abc) in practice," *Trends in Ecology & Evolution*, vol. 25, no. 7, pp. 410–418, 2010.

[118] J.-M. Lueckmann, J. Boelts, D. Greenberg, P. Goncalves, and J. Macke, "Benchmarking simulation-based inference," in *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, A. Banerjee and K. Fukumizu, Eds., ser. Proceedings of Machine Learning Research, vol. 130, PMLR, 2021, pp. 343–351.

[119] J.-M. Lueckmann, P. J. Goncalves, G. Bassetto, K. Öcal, M. Nonnenmacher, and J. H. Macke, "Flexible statistical inference for mechanistic models of neural dynamics," *Advances in neural information processing systems*, vol. 30, 2017.

[120] D. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *International conference on machine learning*, PMLR, 2015, pp. 1530–1538.

[121] M. Magdon-Ismail and A. Atiya, "Neural networks for density estimation," *Advances in Neural Information Processing Systems*, vol. 11, 1998.

[122] G. Papamakarios, D. Sterratt, and I. Murray, "Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows," in *The 22nd international conference on artificial intelligence and statistics*, PMLR, 2019, pp. 837–848.

[123] J.-M. Lueckmann, G. Bassetto, T. Karaletsos, and J. H. Macke, "Likelihood-free inference with emulator networks," in *Symposium on Advances in Approximate Bayesian Inference*, PMLR, 2019, pp. 32–53.

[124] M. U. Gutmann, R. Dutta, S. Kaski, and J. Corander, "Likelihood-free inference via classification," *Statistics and Computing*, vol. 28, pp. 411–425, 2018.

[125] O. Thomas, R. Dutta, J. Corander, S. Kaski, and M. U. Gutmann, "Likelihood-free inference by ratio estimation," *Bayesian Analysis*, vol. 17, no. 1, pp. 1–31, 2022.

[126] K. Cranmer, J. Brehmer, and G. Louppe, "The frontier of simulation-based inference," *Proceedings of the National Academy of Sciences*, vol. 117, no. 48, pp. 30 055–30 062, 2020.

[127] N. Vihrs, "Using neural networks to estimate parameters in spatial point process models," *Spatial Statistics*, vol. 51, p. 100 668, 2022.

[128] L. Gabrielli, S. Tomassetti, S. Squartini, C. Zinato, *et al.*, "Introducing deep machine learning for parameter estimation in physical modelling," in *Proceedings of the 20th international conference on digital audio effects*, 2017.

[129] J. Voznica, A. Zhukova, V. Boskova, *et al.*, "Deep learning from phylogenies to uncover the epidemiological dynamics of outbreaks," *Nature Communications*, vol. 13, no. 1, p. 3896, 2022.

[130] D. R. Schrider and A. D. Kern, "Supervised machine learning for population genetics: A new paradigm," *Trends in Genetics*, vol. 34, no. 4, pp. 301–312, 2018.

[131] J. R. Adrion, J. G. Galloway, and A. D. Kern, "Predicting the Landscape of Recombination Using Deep Learning," *Molecular Biology and Evolution*, vol. 37, no. 6, pp. 1790–1808, Feb. 2020.

[132] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017.

[133] D. A. Nix and A. S. Weigend, "Estimating the mean and variance of the target probability distribution," in *Proceedings of 1994 ieee international conference on neural networks (ICNN'94)*, IEEE, vol. 1, 1994, pp. 55–60.

[134] S. Tsimenidis, E. Vrochidou, and G. A. Papakostas, "Omics data and data representations for deep learning-based predictive modeling," *International Journal of Molecular Sciences*, vol. 23, no. 20, p. 12 272, 2022.

[135] J. Hadfield, C. Megill, S. M. Bell, *et al.*, "Nextstrain: Real-time tracking of pathogen evolution," *Bioinformatics*, vol. 34, no. 23, pp. 4121–4123, Dec. 2018.

[136] M. I. Nelson, C. Viboud, L. Simonsen, *et al.*, "Multiple Reassortment Events in the Evolutionary History of H1N1 Influenza A Virus Since 1918," *PLoS Pathogens*, vol. 4, no. 2, e1000012, Feb. 2008.

[137] M. J. Harms and J. W. Thornton, "Evolutionary biochemistry: Revealing the historical and physical causes of protein properties," *Nature reviews. Genetics*, vol. 14, no. 8, pp. 559–571, Aug. 2013.

[138] B. Perez-Lamarque, M. Öpik, O. Maliet, *et al.*, "Analysing diversification dynamics using barcoding data: The case of an obligate mycorrhizal symbiont," eng, *Molecular Ecology*, vol. 31, no. 12, pp. 3496–3512, Jun. 2022.

[139] J. Felsenstein, "Evolutionary trees from dna sequences: A maximum likelihood approach," *Journal of molecular evolution*, vol. 17, no. 6, pp. 368–376, 1981.

[140] W. G. Weisburg, S. J. Giovannoni, and C. R. Woese, "The Deinococcus-Thermus phylum and the effect of rRNA composition on phylogenetic tree construction," eng, *Syst Appl Microbiol*, vol. 11, pp. 128–34, Jan. 1989.

[141] Z. Yang, "Among-site rate variation and its impact on phylogenetic analyses," *Trends in Ecology & Evolution*, vol. 11, no. 9, pp. 367–372, Sep. 1996.

[142] M. J. Telford, M. J. Wise, and V. Gowri-Shankar, "Consideration of RNA Secondary Structure Significantly Improves Likelihood-Based Estimates of Phylogeny: Examples from the Bilateria," *Molecular Biology and Evolution*, vol. 22, no. 4, pp. 1129–1136, Apr. 2005.

[143] S. Guindon and O. Gascuel, "A Simple, Fast, and Accurate Algorithm to Estimate Large Phylogenies by Maximum Likelihood," *Systematic Biology*, vol. 52, no. 5, pp. 696–704, Oct. 2003.

[144] O. Gascuel, "Bionj: An improved version of the nj algorithm based on a simple model of sequence data.," *Molecular biology and evolution*, vol. 14, no. 7, pp. 685–695, 1997.

[145] R. M. Rao, J. Liu, R. Verkuil, *et al.*, "Msa transformer," in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 8844–8856.

[146] S. Q. Le and O. Gascuel, "An improved general amino acid replacement matrix," *Molecular biology and evolution*, vol. 25, no. 7, pp. 1307–1320, 2008.

[147] J. Jumper, R. Evans, A. Pritzel, *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[148] J. Ho, N. Kalchbrenner, D. Weissenborn, and T. Salimans, "Axial attention in multidimensional transformers," *CoRR*, vol. abs/1912.12180, 2019.

[149] Z. Yang, "Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: Approximate methods," eng, *Journal of Molecular Evolution*, vol. 39, no. 3, pp. 306–14, Sep. 1994.

[150] A. Rambaut, *Seq-Gen*, `http://tree.bio.ed.ac.uk/software/seqgen/`, 2017.

[151] J. P. Huelsenbeck and F. Ronquist, "Mrbayes: Bayesian inference of phylogenetic trees," *Bioinformatics*, vol. 17, no. 8, pp. 754–755, 2001.

[152] Z. Yang, "Paml 4: Phylogenetic analysis by maximum likelihood," *Molecular biology and evolution*, vol. 24, no. 8, pp. 1586–1591, 2007.

[153] S. Höhna, M. J. Landis, T. A. Heath, *et al.*, "RevBayes: Bayesian Phylogenetic Inference Using Graphical Models and an Interactive Model-Specification Language," en, *Systematic Biology*, vol. 65, no. 4, pp. 726–736, Jul. 2016, Publisher: Oxford Academic.

[154] G. J. Szöllõsi, S. Höhna, T. A. Williams, D. Schrempf, V. Daubin, and B. Boussau, "Relative time constraints improve molecular dating," *Regular Manuscript*, vol. 71, no. 4, pp. 797–809, 2022.

[155] *HOGENOM*, `http://hogenom.univ-lyon1.fr/`, 2019.

[156] D. Höhler, W. Pfeiffer, V. Ioannidis, H. Stockinger, and A. Stamatakis, "Raxml grove: An empirical phylogenetic tree database," *Bioinformatics*, vol. 38, no. 6, pp. 1741–1742, 2022.

[157] B. D. Redelings and M. A. Suchard, "Incorporating indel information into phylogeny estimation for rapidly emerging pathogens," *BMC Evolutionary Biology*, vol. 7, no. 1, p. 40, 2007.

[158] J. Trost, J. Haag, D. Höhler, L. Jacob, A. Stamatakis, and B. Boussau, "Simulations of sequence evolution: How (un) realistic they are and why," *Molecular Biology and Evolution*, vol. 41, no. 1, msad277, 2024.

[159] S. Prillo, Y. Deng, P. Boyeau, X. Li, P.-Y. Chen, and Y. S. Song, "Cherryml: Scalable maximum likelihood estimation of phylogenetic models," *Nature methods*, vol. 20, no. 8, pp. 1232–1236, 2023.

[160] L. Si Quang, O. Gascuel, and N. Lartillot, "Empirical profile mixture models for phylogenetic reconstruction," *Bioinformatics*, vol. 24, no. 20, pp. 2317–2323, Oct. 2008.

[161] G. J. Szöllősi, W. Rosikiewicz, B. Boussau, E. Tannier, and V. Daubin, *Data from: Efficient exploration of the space of reconciled gene trees*, version 1, Artwork Size: 45688314 bytes Pages: 45688314 bytes, Aug. 7, 2013.

[162] F. Llinares-López, Q. Berthet, M. Blondel, O. Teboul, and J.-P. Vert, "Deep embedding and alignment of protein sequences," *Nature Methods*, vol. 20, no. 1, pp. 104–111, 2023.

[163] S. Petti, N. Bhattacharya, R. Rao, *et al.*, "End-to-end learning of multiple sequence alignments with differentiable Smith–Waterman," *Bioinformatics*, vol. 39, no. 1, btac724, Nov. 2022.

[164] B. Boussau and M. Gouy, "Efficient Likelihood Computations with Nonreversible Models of Evolution," *Systematic Biology*, vol. 55, no. 5, pp. 756–768, Oct. 2006.

[165] S. Blanquart and N. Lartillot, "A Site- and Time-Heterogeneous Model of Amino Acid Replacement," *Molecular Biology and Evolution*, vol. 25, no. 5, pp. 842–858, May 2008.

[166] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, *Transformers are rnns: Fast autoregressive transformers with linear attention*, PMLR, 2020.

[167] M. M. Bronstein, J. Bruna, T. Cohen, and P. Velickovic, "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges," *CoRR*, vol. abs/2104.13478, 2021.

[168] B. Schölkopf and A. J. Smola, *Learning with kernels : support vector machines, regularization, optimization, and beyond* (Adaptive computation and machine learning). MIT Press, 2002.

[169] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016.

[170] E. Abbe, S. Bengio, E. Boix-Adserà, E. Littwin, and J. M. Susskind, *Transformers learn through gradual rank increase*, 2023.

[171] J. Zhao, Y. Zhang, B. Chen, F. T. Schaefer, and A. Anandkumar, *Incremental low-rank learning*, 2023.

[172] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2015.

[173] S. Kalyaanamoorthy, B. Q. Minh, T. K. Wong, A. Von Haeseler, and L. S. Jermiin, "Modelfinder: Fast model selection for accurate phylogenetic estimates," *Nature methods*, vol. 14, no. 6, pp. 587–589, 2017.

[174] L. Duchemin, V. Lanore, P. Veber, and B. Boussau, "Evaluation of methods to detect shifts in directional selection at the genome scale," *Molecular Biology and Evolution*, vol. 40, no. 2, msac247, 2023.

[175] J. Sukumaran and M. T. Holder, "Dendropy: A python library for phylogenetic computing," *Bioinformatics*, vol. 26, no. 12, pp. 1569–1571, 2010.

[176] G. Loewenthal, D. Rapoport, O. Avram, *et al.*, "A probabilistic model for indel evolution: Differentiating insertions from deletions," *Molecular biology and evolution*, vol. 38, no. 12, pp. 5769–5781, 2021.

[177] W. H. Piel, M. Donoghue, M. Sanderson, and L Netherlands, "Treebase: A database of phylogenetic information," in *Proceedings of the 2nd International Workshop of Species*, vol. 2000, 2000.

[178] A. L. Halpern and W. J. Bruno, "Evolutionary distances for protein-coding sequences: Modeling site-specific residue frequencies.," *Molecular biology and evolution*, vol. 15, no. 7, pp. 910–917, 1998.

[179] A. U. Tamuri and M. d. Reis, "A mutation-selection model of protein evolution under persistent positive selection," en, *bioRxiv*, p. 2021.05.18.444611, May 2021, Publisher: Cold Spring Harbor Laboratory Section: New Results.

[180] R. Xiong, Y. Yang, D. He, *et al.*, "On layer normalization in the transformer architecture," in *International Conference on Machine Learning*, PMLR, 2020, pp. 10 524–10 533.

[181] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[182] D. Höhler, J. Haag, A. M. Kozlov, and A. Stamatakis, "A representative performance assessment of maximum likelihood based phylogenetic inference tools," *bioRxiv*, pp. 2022–10, 2022.

[183] C. Anil, Y. Wu, A. Andreassen, *et al.*, "Exploring length generalization in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 38 546–38 556, 2022.

[184] A. De Myttenaere, B. Golden, B. Le Grand, and F. Rossi, "Mean absolute percentage error for regression models," *Neurocomputing*, vol. 192, pp. 38–48, 2016.

[185] D. M. de Vienne, G. Aguileta, and S. Ollier, "Euclidean nature of phylogenetic distance matrices," *Systematic biology*, vol. 60, no. 6, pp. 826–832, 2011.

[186] M. Layer and J. A. Rhodes, "Phylogenetic trees and euclidean embeddings," *Journal of mathematical biology*, vol. 74, pp. 99–111, 2017.

[187] M. Nickel and D. Kiela, "Poincaré embeddings for learning hierarchical representations," *Advances in neural information processing systems*, vol. 30, 2017.

[188] N. Monath, M. Zaheer, D. Silva, A. McCallum, and A. Ahmed, "Gradient-based hierarchical clustering using continuous representations of trees in hyperbolic space," in *Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining*, 2019, pp. 714–722.

[189] H. Matsumoto, T. Mimori, and T. Fukunaga, "Novel metric for hyperbolic phylogenetic tree embeddings," *Biology Methods and Protocols*, vol. 6, no. 1, bpab006, 2021.

[190] Y. Jiang, P. Tabaghi, and S. Mirarab, "Learning hyperbolic embedding for phylogenetic tree placement and updates," *Biology*, vol. 11, no. 9, p. 1256, 2022.

[191] M. Macaulay, A. Darling, and M. Fourment, "Fidelity of hyperbolic space for bayesian phylogenetic inference," *PLoS computational biology*, vol. 19, no. 4, e1011084, 2023.

[192] G. Corso, Z. Ying, M. Pándy, P. Veličković, J. Leskovec, and P. Liò, "Neural distance embeddings for biological sequences," *Advances in Neural Information Processing Systems*, vol. 34, pp. 18 539–18 551, 2021.

[193] L. Duchemin, "Phylogenetic detection of protein sites associated to a phenotype, at the genome scale," Ph.D. dissertation, Université Claude Bernard-Lyon I, 2023.

[194] S. Parto and N. Lartillot, "Molecular adaptation in rubisco: Discriminating between convergent evolution and positive selection using mechanistic and classical codon models," *PloS one*, vol. 13, no. 2, e0192697, 2018.

[195] Z. Yang, "Paml 4: Phylogenetic analysis by maximum likelihood," *Molecular biology and evolution*, vol. 24, no. 8, pp. 1586–1591, 2007.

[196] A. U. Tamuri, M. Dos Reis, A. J. Hay, and R. A. Goldstein, "Identifying changes in selective constraints: Host shifts in influenza," *PLoS computational biology*, vol. 5, no. 11, e1000564, 2009.

[197] C. Rey, V. Lanore, P. Veber, *et al.*, "Detecting adaptive convergent amino acid evolution," *Philosophical Transactions of the Royal Society B*, vol. 374, no. 1777, p. 20 180 234, 2019.

[198] J. D. Bloom, "Identification of positive selection in genes is greatly improved by using experimentally informed site-specific models," *Biology direct*, vol. 12, pp. 1–24, 2017.

[199] V. Garot, L. Blassel, L. Nesterenko, A. Zhukova, S. Alizon, and L. Jacob, "Phylodynamics without phylogenies: Deep-learning-based inference of epidemiological parameters from sequence alignments," *In preparation*, 2024.

[200] J. L. Thorne, H. Kishino, and J. Felsenstein, "An evolutionary model for maximum likelihood alignment of dna sequences," *Journal of Molecular Evolution*, vol. 33, pp. 114–124, 1991.

[201] T. Latrille, J. Joseph, D. Hartasánchez, and N. Salamin, "Mammalian protein-coding genes exhibit widespread beneficial mutations that are not adaptive," 2023.

[202] C. Scornavacca, K. Belkhir, J. Lopez, *et al.*, "Orthomam v10: Scaling-up orthologous coding sequence and exon alignments with more than one hundred mammalian genomes," *Molecular Biology and Evolution*, vol. 36, no. 4, pp. 861–862, 2019.

[203] M. V. Kapralov, J. A. C. Smith, and D. A. Filatov, "Rubisco evolution in c4 eudicots: An analysis of amaranthaceae sensu lato," *PloS one*, vol. 7, no. 12, e52974, 2012.

[204] G. Besnard, A. M. Muasya, F. Russier, E. H. Roalson, N. Salamin, and P.-A. Christin, "Phylogenomics of c4 photosynthesis in sedges (cyperaceae): Multiple appearances and genetic convergence," *Molecular Biology and Evolution*, vol. 26, no. 8, pp. 1909–1919, 2009.

[205] B. Murrell, J. O. Wertheim, S. Moola, T. Weighill, K. Scheffler, and S. L. Kosakovsky Pond, "Detecting individual sites subject to episodic diversifying selection," *PLoS genetics*, vol. 8, no. 7, e1002764, 2012.

[206] Y. Li, Z. Liu, P. Shi, and J. Zhang, "The hearing gene prestin unites echolocating bats and whales," *Current Biology*, vol. 20, no. 2, R55–R56, 2010.

[207] R. A. Fisher, "Statistical methods for research workers," in *Breakthroughs in statistics: Methodology and distribution*, Springer, 1970, pp. 66–70.

[208] C. R. Walker, C. West, S. Arasti, *et al.*, "Detecting interspecific positive selection using convolutional neural networks," *bioRxiv*, 2024.

[209] R. Nielsen and Z. Yang, "Likelihood models for detecting positively selected amino acid sites and applications to the hiv-1 envelope gene," *Genetics*, vol. 148, no. 3, pp. 929–936, 1998.

[210] W. Fletcher and Z. Yang, "Indelible: A flexible simulator of biological sequence evolution," *Molecular biology and evolution*, vol. 26, no. 8, pp. 1879–1888, 2009.

[211] F. Sievers, A. Wilm, D. Dineen, *et al.*, "Fast, scalable generation of high-quality protein multiple sequence alignments using clustal omega," *Molecular systems biology*, vol. 7, no. 1, p. 539, 2011.

[212] A. Loytynoja and N. Goldman, "Phylogeny-aware gap placement prevents errors in sequence alignment and evolutionary analysis," *science*, vol. 320, no. 5883, pp. 1632–1635, 2008.

[213] U. Lupo, D. Sgarbossa, and A.-F. Bitbol, "Protein language models trained on multiple sequence alignments learn phylogenetic relationships," *Nature Communications*, vol. 13, no. 1, p. 6298, 2022.

[214] K. He, R. Girshick, and P. Dollár, "Rethinking imagenet pre-training," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 4918–4927.

[215] D. Hernandez, J. Kaplan, T. Henighan, and S. McCandlish, "Scaling laws for transfer," *arXiv preprint arXiv:2102.01293*, 2021.

[216] Z. Mo and A. Siepel, "Domain-adaptive neural networks improve supervised machine learning based on simulated population genetic data," *PLoS Genetics*, vol. 19, no. 11, e1011032, 2023.

[217] Z. Wang, J. Wang, M. Kourakos, *et al.*, "Automatic inference of demographic parameters using generative adversarial networks," *Molecular ecology resources*, vol. 21, no. 8, pp. 2689–2705, 2021.

[218] I. Elias, "Settling the intractability of multiple alignment," *Journal of Computational Biology*, vol. 13, no. 7, pp. 1323–1339, 2006.

[219] J. C. Aledo, "Phylogenies from unaligned proteomes using sequence environments of amino acid residues," *Scientific reports*, vol. 12, no. 1, p. 7497, 2022.

[220] R. Tang, Z. Yu, and J. Li, "Kinn: An alignment-free accurate phylogeny reconstruction method based on inner distance distributions of k-mer pairs in biological sequences," *Molecular Phylogenetics and Evolution*, vol. 179, p. 107 662, 2023.