Title

CherryML: scalable maximum likelihood estimation of phylogenetic models.

Journal

Nature Methods, 20(8)

Authors

Prillo, Sebastian

Deng, Yun

Boyeau, Pierre

et al.

Publication Date

2023-08-01

DOI

10.1038/s41592-023-01917-9

Peer reviewed

HHS Public Access

Author manuscript

Nat Methods. Author manuscript; available in PMC 2023 November 14.

Published in final edited form as:

Nat Methods. 2023 August ; 20(8): 1232–1236. doi:10.1038/s41592-023-01917-9.

CherryML: Scalable Maximum Likelihood Estimation of Phylogenetic Models

Sebastian Prillo[1], **Yun Deng**[2], **Pierre Boyeau**[1], **Xingyu Li**[1], **Po-Yen Chen**[1], **Yun S. Song**[1,3,*]

[1]Computer Science Division, University of California, Berkeley

[2]Graduate Group in Computational Biology, University of California, Berkeley

[3]Department of Statistics, University of California, Berkeley

Abstract

Phylogenetic models of molecular evolution are central to numerous biological applications spanning diverse timescales, from hundreds of millions of years involving orthologous proteins to just tens of days relating single cells within an organism. A fundamental problem in these applications is estimating model parameters, for which maximum likelihood estimation (MLE) is typically employed. Unfortunately, MLE is a computationally expensive task, in some cases prohibitively so. To address this challenge, we here introduce CherryML, a broadly applicable method that achieves several orders of magnitude speedup by using a quantized composite likelihood over cherries in the trees. The massive speedup offered by our method should enable researchers to consider more complex and biologically realistic models than previously possible. Here we demonstrate CherryML's utility by applying it to estimate a general 400 × 400 rate matrix for residue-residue coevolution at contact sites in 3D protein structures; we estimate that using current state-of-the-art methods such as the expectation-maximization algorithm for the same task would take > 100, 000 times longer.

Introduction

Phylogenetic models of molecular evolution have a plethora of applications in biology. For example, models of amino acid substitution enable the estimation of gene trees and protein alignments, among other important applications [1–9]. These models posit that molecules evolve down a phylogenetic tree according to a continuous-time Markov process

*To whom correspondence should be addressed: yss@berkeley.edu.

Author Contributions

S.P. and Y.S.S. conceived and designed the study. S.P. developed, implemented, and tested the method, with assistance from Y.D., P.B., X.L. and P.-Y.C. In particular, P.B. contributed to implementing optimization with PyTorch, while X.L. and P.-Y.C. contributed to parallelizing the computation of count matrices and the simulations. Y.D. helped with testing the method. S.P. and Y.S.S. analyzed the coevolution model. S.P. and Y.S.S. wrote the manuscript, and Y.D. made edits. Y.S.S. supervised the project.

Code Availability

Code for reproducing all results in this paper, as well as code implementing the CherryML method for the LG model and for the coevolution model, is available on GitHub at the repository: https://github.com/songlab-cal/CherryML

The CherryML package allows seamless estimation of rate matrices from MSAs. An end-to-end demonstration on the plant dataset [26] with train/test splits from the QMaker work [9] is provided in the package's README.

Competing Interests

The authors declare no competing interests.

parameterized by one or more rate matrices. These models differ in what kind of molecular data they describe (e.g., DNA, RNA, amino acid, or codon sequences), the use of site rate variation (such as invariable sites model, the Γ model, or the probability-distribution-free model [10,11]), the treatment of indels [12], and the assumption of i.i.d. (independent and identically distributed) sites [13]. A common aspect of all these models is that maximum likelihood estimation (MLE) of model parameters is computationally expensive, in some cases prohibitively so.

The main computational challenge arises from the unobserved ancestral states. A typical step during MLE involves estimating the rate matrix Q given a set of m multiple sequence alignments (MSAs) and associated phylogenetic trees, and computing the log-likelihood alone requires marginalizing out the unobserved ancestral states, which is done with Felsenstein's pruning algorithm [14]. If s denotes the state space size (e.g., $s = 20$ for amino acids), l the typical sequence length, and n the typical number of sequences per MSA, then computing the likelihood with Felsenstein's pruning algorithm has a cost of $\Omega\left(mn\left(ls^2 + s^3\right)\right)$ for the typical model. This comes from having to compute, for each of the $\Omega(mn)$ edges, (at least) one matrix exponential of cost $\Omega\left(s^3\right)$, and a recursion step of cost $\Omega\left(s^2\right)$ per alignment site.

MLE for these models is usually performed with either zeroth-order optimization [9] or with the Expectation-Maximization (EM) algorithm [15,16], both of which require Felsenstein's pruning algorithm. As a consequence, full MLE is typically slow and rarely performed in practice [9]. Instead, researchers choose to utilize generic rate matrices such as the popular LG matrix [5] – which was estimated more than a decade ago – to perform their phylogenetic analyses. For more complex models, estimation with current approaches is just infeasible. For example, learning a general 400 × 400 rate matrix describing the coevolution of contacting residues in a protein is out of reach with current approaches.

Results

The CherryML method

To overcome the above challenges, we here propose CherryML, a broadly applicable method to scale up MLE for general phylogenetic models of molecular evolution. CherryML hinges on two key ideas: *composite* likelihood and time *quantization*. We describe these ideas in turn below and illustrate them in Figure 1a.

The first key idea underlying CherryML is to use a composite likelihood over *cherries* in the trees (where a cherry corresponds to a pair of leaves separated by exactly one internal node) instead of the full likelihood. This means selecting all cherries for each tree and replacing the full log-likelihood with the sum of log-likelihoods of the cherries. Further, to maximize the number of paired sequences, we iteratively pick and prune cherries from the tree until at most one unpaired sequence remains (see Methods for full details). This reduces likelihood computation to *pairs* of sequences at a time, which is much simpler than the full likelihood. In particular, for a time-reversible model, the likelihood of a pair of sequences only depends on the distance between the nodes in the tree, and does not require marginalizing out

ancestral states. Maximum composite likelihood estimation (MCLE) enjoys many of the properties of MLE, such as consistency under weak conditions [17].

The second key idea of CherryML is to quantize time. This means approximating transition times by one of finitely many values $\tau_1 < \tau_2 < \cdots < \tau_b$. We choose the τ_k to be geometrically spaced and to cover the whole operational range of transition times; thanks to the geometric spacing, a few hundred values are enough to achieve a quantization error as low as 1% for all transition times. When time is quantized, the terms in the composite likelihood typically group together by quantized transition time τ_k, and the cost of evaluating the likelihood no longer depends on the dataset size, dramatically speeding up optimization.

CherryML applied to the LG model

We applied the CherryML method to the seminal model of Le and Gascuel (LG) [5], which assumes that each site in every protein evolves under the same rate matrix Q and with a site specific rate. As a result, the composite log-likelihood has the functional form $\sum_{k=1}^{b} \langle C_k, \log \exp(Q\tau_k) \rangle$, where the logarithm is applied entry-wise, and C_k is the frequency matrix for transitions between states occurring at a quantized time of τ_k. Hence, the cost of evaluating the likelihood no longer depends on the dataset size (see Methods for full details). This objective function can be easily optimized in modern first-order numerical optimization libraries such as PyTorch [18]. If g denotes the number of iterations of the first-order optimizer, the computational complexity of the CherryML method applied to the LG model is then $\Theta\left(mnl\log b + gbs^3\right)$. Here $\Theta\left(mnl\log b\right)$ comes from computing the count matrices C_k, and $\Theta\left(gbs^3\right)$ comes from the first-order optimizer; the $\Theta\left(mnl\log b\right)$ term will dominate for large datasets. In contrast, the cost of MLE with traditional methods such as zeroth-order optimization or EM is $\Omega\left(gmn\left(ls^2 + s^3\right)\right)$, and the $\Omega\left(gmnls^2\right)$ term will dominate. Thus, ignoring constants, a back-of-the-envelope calculation reveals that CherryML will be at least $\frac{gmnls^2}{mnl\log b} = \frac{gs^2}{\log(b)}$ times faster than traditional methods, which is typically a massive speedup. Indeed, when learning a single-site model using $g = 100$ iterates and $b = 100$ quantization points for CherryML, the speedup is at least (up to constants) $\frac{gs^2}{\log(b)} = \frac{100 \times 20^2}{\log(100)} \in \left[10^3, 10^4\right]$ fold. When learning a coevolution model, the speedup is at least (up to constants) $\frac{gs^2}{\log(b)} = \frac{100 \times 400^2}{\log(100)} \in \left[10^6, 10^7\right]$ fold. The computational bottleneck of CherryML is computing the count matrices C_k, and we have developed an efficient distributed-memory C++ implementation using MPI. (See Methods for the details.)

By using simulated data, we benchmarked the computational runtime and statistical efficiency of the CherryML optimizer for the aforementioned LG model, and compared it with the performance of EM, for which we used the implementation in the XRATE package [16]. The results are summarized in Figure 1b,c, which shows that CherryML is a thousand times faster than EM when run on 1,024 families with 128 sequences each, taking 24 seconds as compared to 6.5 hours. This comes only at the cost of approximately 50% loss of statistical efficiency; full plots of the true matrix entries versus the estimated matrix entries for CherryML and EM are shown in Extended Data Fig. 1. Using data simulated on

all 15,051 protein families studied in Yang *et al.* [19], we found that $b \approx 100$ quantization points were enough to make the error introduced by time quantization negligible, as shown in Figure 1d.

We next applied CherryML to the Pfam data from Le and Gascuel [5]. We implemented their end-to-end estimation procedure, replacing the EM optimizer with the CherryML optimizer. We estimated phylogenetic trees and site-specific rates using FastTree [20], and then applied CherryML to estimate the 20×20 rate matrix Q_{20}. Tree estimation and rate matrix estimation were iterated 3 times until convergence as typical, starting from the uninformative uniform rate matrix. We reproduced and extended the results from Figure 4 of Le and Gascuel [5] by adding our re-estimate of the LG rate matrix using our CherryML optimizer, as well as a re-estimate using the EM optimizer as implemented by the XRATE package [16], which Le and Gascuel employed. Figure 1e shows that the results obtained with the CherryML method are comparable to that obtained using EM. In Extended Data Fig. 2, we show that these three matrices are very close to each other, only noticeably differing in four of the smallest (harder to estimate) rates. Regarding computational cost, the *end-to-end* runtime (including tree estimation) using the EM optimizer was around 13.5 CPU hours: 1.25 CPU hours for tree estimation with FastTree and 12.25 CPU hours for the EM optimizer. In contrast, the end-to-end runtime using the CherryML optimizer was around 1.35 CPU hours: 1.25 CPU hours for tree estimation with FastTree and just 0.1 CPU hours (6 minutes) for the CherryML optimizer. This represents an order-of-magnitude speedup in the end-to-end context, without any noticeable loss of accuracy. We observed similar trends for the datasets from the QMaker paper [9], as shown in Extended Data Fig. 3 and Extended Data Fig. 4.

CherryML applied to learn a 400 × 400 coevolutionary model

Finally, we applied the CherryML method to estimate a general reversible 400×400 rate matrix Q_{400} describing the coevolution of contacting sites in a protein. The mathematics of the model are similar to the LG model except that there is no site rate variation and $s = 400$ (see Methods for full details). The large size of the state space means that the runtime of traditional optimization methods such as EM soars. To learn the model, we used all 15, 051 Pfam MSAs from Yang *et al.* [19] subsampled down to 1,024 sequences each (as typical); each MSA is associated with structure data which we used to determine contacting sites. In total, there are approximately 14 million sequences, 3.7 million sites, and 3000 million residues in this dataset. We estimated phylogenetic trees using FastTree [20] and then applied our CherryML method to estimate Q_{400}. Tree reconstruction was parallelized and took approximately 1 minute per tree. The CherryML optimizer took only 14 minutes using 8 CPU cores: 2 minutes for counting transitions and 12 minutes for the PyTorch optimizer. Extrapolating the results from Figure 1b, as well as based on our theoretical complexity results (see Methods for the details), we estimated that using EM for this task would have taken approximately 35 CPU-years – roughly a *hundred thousand* times slower!

Before analyzing the properties of our estimated rate matrix Q_{400}, we used simulations to determine whether our method had produced a reliable estimate. To this end, we simulated data using Q_{400} for all 15,051 families, and then applied our CherryML method

to re-estimate Q_{400}. Our method was able to produce accurate estimates of most entries. Transitions with just one substitution proved the easiest to recover (Figure 2a) – as expected, since they have higher rates – with a median error of 0.6%. Transitions with two substitutions were harder to estimate owing to their small rates, but were still well estimated, particularly those with a higher rate, exhibiting a median error of 15.5% (Figure 2b).

We analyzed our estimated coevolution model Q_{400} for amino acid pairs in contact and compared it with an independent model of amino acid evolution. We found that our model was able to learn the importance of disulfide bonds by assigning a low mutation rate to CC pairs, approximately 4 times smaller than what is estimated by an independent model (Figure 2c). Similarly, we found that our coevolution model was able to learn the stability of certain kinds of residue contacts based on biochemical properties such as charge: our model assigns a higher observation probability to hydrophobic pairs (Figure 2d). The mutation rates and stationary distributions of our coevolution model and the independent model are provided in Extended Data Fig. 5 and Extended Data Fig. 6. Our coevolution model has a global mutation rate of 1.42, which is much lower than the 2.0 estimated by a model of independent evolution. A retrospective analysis revealed that sites with more contacts have a lower mutation rate as estimated by FastTree, consistent with previous findings that more densely packed sites tend to evolve more slowly [21,22]. The trend is remarkably strong, as shown in Figure 2e. Finally, our model inferred interesting simultaneous substitutions at both sites in contact. For example, we observed that the KE ↔ EK transition probability is substantially higher in the coevolution model than that in the independent model. This coupled evolution seems mediated by the stability of electrostatic interaction (K is positively charged, while E is negatively charged) and we found support for it in the original MSAs: for example, in the family `4kv7_1_A`, when sites 165 and 181 contain E and K, 97% of the time they are either KE or EK, while only 3% of the time they are EE or KK.

Discussion

While our CherryML method streamlines rate matrix estimation given MSAs and trees, to obtain good trees in the first place, tree estimation given a rate matrix and MSAs is necessary. These two steps are usually interlocked, resulting in a coordinate ascent procedure, as done in the LG paper [5], as well as in Figure 1e and Extended Data Fig. 3 of this article. With the introduction of the CherryML optimization method, tree estimation will most likely dominate runtime in any end-to-end application where trees are not available *a priori*. An important line of work on speeding up the tree estimation step includes FastMG [23], which proposes cleverly splitting the input MSAs into smaller sub-MSAs to reconstruct smaller trees, as well as ModelFinder [11] for ultra-fast model selection and subsequent improved tree inference. A superior method for end-to-end rate matrix estimation may thus result from combining approaches like FastMG and ModelFinder [11] with CherryML for rate estimation, which is a promising direction of future research. In the case of a coevolutionary model, likelihood computation scales quadratically with the number of single-site states, making tree inference significantly slower than for an independent-sites model. We believe that ideas in this paper such as time quantization should help to speed up tree inference in this setting.

We envision that our CherryML framework will be broadly applicable to enable and scale up MLE under many models of molecular evolution. The ability to estimate transition rate matrices at unprecedented speed will transform the way that phylogenetic analysis is performed. Researchers will be able to estimate context-dependent rate matrices in a matter of minutes, such as by protein function, family, domain, or structure, which can then be applied – in place of the generic LG matrix – to estimate more accurate phylogenetic trees [9,24]. In particular, recent advances in protein modeling, such as AlphaFold [25], have made such contextual structural information readily available.

Finally, although the CherryML method is most effective for reversible models of evolution – where it completely solves the problem of marginalizing out ancestral states – it can also be applied to irreversible models. In this case, the composite likelihood depends on the two branch lengths t_1, t_2 of each cherry, which can be quantized separately. For a typical model like LG [5], the terms in the quantized composite likelihood group together by pairs of quantized branch lengths $(\tau_k, \tau_{k'})$. As a result, EM can be performed in its classical form, except that now only *finitely* many ancestral state inferences need to be performed, thereby dramatically speeding up EM.

Methods

Composite Likelihood over Cherries

The first idea of the CherryML method is to use a composite likelihood over all iteratively picked cherries of the trees instead of the full likelihood. Note that in a tree without multifurcations, there is a unique way of iteratively picking cherries until at most one unpaired sequence remains. Using cherries is motivated by the fact that they do not share any edges of the tree and are therefore closer to being independent samples than, say, are randomly chosen pairs of sequences. However, we should remark that the consistency of our method holds regardless of how the sequences are paired. For example, randomly pairing all sequences still yields a consistent estimator under weak assumptions [17]. In fact, it is possible to include a sequence in more than one pair; in other words, the pairs may overlap.

Let $D = (D_1, D_2, \dots, D_m)$ be the multiple sequence alignments and $T = (T_1, T_2, \dots, T_m)$ their associated phylogenetic trees. Then the log-likelihood of the data can be written as:

$$l(Q) \equiv \log\, p(D \mid T, Q) = \sum_{i=1}^{m} \log\, p(D_i \mid T_i, Q)\,. \tag{1}$$

To obtain a composite likelihood, we select all iteratively picked cherries $\{(u_j^i, v_j^i)\}_{1 \le j \le c_i}$ from each phylogenetic tree T_i, where c_i is the number of such pairs for T_i, and then substitute the likelihood from Eq. (1) with the composite likelihood over the selected cherries:

$$l_{\text{comp}}(Q) \equiv \sum_{i=1}^{m} \sum_{j=1}^{c_i} \log\, p\big(D_i[u_j^i] \,\big|\, D_i[v_j^i], T_i, Q\big), \tag{2}$$

where $D_i[w]$ is the row of D_i corresponding to leaf w. Cherries are considered in both directions, meaning that if (a, b) is a cherry then so is (b, a) (hence c_i is always even). We note that in the case of a reversible model, the likelihood term $p(D_i[u_j^i] \,|\, D_i[v_j^i], T_i, Q)$ only depends on the distance between u_j^i and v_j^i in the tree T_i, so if we denote this distance as t_j^i, we can write, abusing notation slightly, the composite log-likelihood as:

$$l_{\text{comp}}(Q) \equiv \sum_{i=1}^{m} \sum_{j=1}^{c_i} \log\, p\big(D_i[u_j^i] \,\big|\, D_i[v_j^i], t_j^i, Q\big)\,. \tag{3}$$

Quantization of Time

The second idea of the CherryML method is to quantize transition times. Concretely, we approximate each transition time t by one of the finitely many values $\tau_1 < \tau_2 < \cdots < \tau_b$. We choose them to be geometrically spaced and to cover the whole operational range of transition times. As shown in Figure 1d, around 100 quantization points are enough to obtain negligible quantization error. We use $b = 129$ quantization points with a center value of $\tau_{65} = 0.03$ and a geometric spacing of 1.1. Thus $\tau_1 \approx 6.7 \times 10^{-5}$ and $\tau_{129} \approx 13.4$. This represents a wide range of transition times. We denote the quantized value of t as $q(t)$, which is chosen to minimize the relative error, that is to say:

$$q(t) = \tau_k, \quad \text{where } k = \arg\min_{1 \le k \le b} \frac{\max(t, \tau_k) - \min(t, \tau_k)}{\min(t, \tau_k)}\,. \tag{4}$$

Quantization of a transition time t can be performed in time $\mathcal{O}(\log b)$ with binary search. Any transition showcasing a transition time falling outside of the interval $[\tau_1, \tau_{129}]$ (because it is either too small or too large) is discarded and thus dropped from the likelihood Eq. (3). Using a geometric spacing of 1.1 means that the relative error between successive points is 10%, and thus the *worst* quantization error (which happens in between two quantization points) is roughly 5%. The *average* quantization error is therefore around 2.5%.

In the case of a time-reversible model, quantizing time means replacing $p(D_i[u_j^i] \,|\, D_i[v_j^i], t_j^i, Q)$ by $p(D_i[u_j^i] \,|\, D_i[v_j^i], q(t_j^i), Q)$. Concretely, we replace Eq. (3) with:

$$l_{\text{comp, quant}}(Q) \equiv \sum_{i=1}^{m} \sum_{j=1}^{c_i} \log\, p\big(D_i[u_j^i] \,\big|\, D_i[v_j^i], q(t_j^i), Q\big)\,. \tag{5}$$

When this is done, the terms in the composite likelihood typically group together by quantized time τ_k, and the cost of evaluating the likelihood no longer depends on the dataset size. We call this the *quantized (log-)likelihood*. For a model with site rate variation such as LG, transition times are adjusted by the site rate before quantizing.

CherryML for the LG model

The LG model [5] of protein evolution posits that sites in a protein evolve independently under a scalar multiple of a rate matrix Q of size 20 × 20. The scalar for each site might be different and is called the *site rate*. If l_i denotes the length of the protein in family i, and r_k^i denotes the rate of site k ($1 \leq k \leq l_i$), then time quantization under the reversible LG model is performed by replacing $p(D_i[u_j^i]_k \mid D_i[v_j^i]_k, T_i, Q)$ by $p(D_i[u_j^i]_k \mid D_i[v_j^i]_k, q(r_k^i t_j^i), Q)$. In other words, we obtain the quantized composite log-likelihood:

$$l_{\text{comp, quant}}(Q) \equiv \sum_{i=1}^{m} \sum_{j=1}^{c_i} \sum_{k=1}^{l_i} \log\, p(D_i[u_j^i]_k \mid D_i[v_j^i]_k, q(r_k^i t_j^i), Q), \tag{6}$$

where $D_i[w]_k$ is site k at the row of D_i corresponding to leaf w. Transitions involving gaps are ignored. For the LG model, the terms of the quantized composite log-likelihood group together by quantized transition time τ_k, and so Eq. (6) reduces to the simpler form:

$$l_{\text{comp, quant}}(Q) \equiv \sum_{k=1}^{b} \langle C_k, \log \exp(Q\tau_k) \rangle, \tag{7}$$

where C_k of size 20 × 20 is the frequency matrix for transitions between states occurring at a quantized transition time of τ_k.

CherryML for the Coevolution Model

Our coevolution model posits that a given pair of contacting sites in a protein sequence evolve together via a reversible 400 × 400 rate matrix Q. Since in a given protein a site might be in contact with more than one other site, we use a maximal matching to pair up contacting sites. This way, we obtain p_i pairs of sites $(s_1, s_2), \ldots, (s_{2p_i-1}, s_{2p_i})$ for family i. The quantized composite log-likelihood is thus:

$$l_{\text{comp, quant}}(Q) \equiv \sum_{i=1}^{m} \sum_{j=1}^{c_i} \sum_{k=1}^{p_i} \log\, p(D_i[u_j^i]_{(s_{2k-1}, s_{2k})} \mid D_i[v_j^i]_{(s_{2k-1}, s_{2k})}, q(t_j^i), Q), \tag{8}$$

where $D_i[w]_{(k_1,k_2)}$ is the pair of site k_1 and k_2 at the row of D_i corresponding to leaf w. Because the order of the two sites does not matter (for example, the transition AL → GI and the transition LA → IG are equivalent), when forming the composite likelihood we also augment the site pairs with the reverse pairs $(s_2, s_1), \ldots, (s_{2p_i}, s_{2p_i-1})$. Transitions involving gaps are ignored. The log-likelihood Eq. (8) has the same functional form as LG model's Eq. (7) except that Q is a 400×400 matrix instead of 20×20. Just like for the LG model, Eq. (8) reduces to the simpler form of Eq. (7) where C_k of size 400×400 is the frequency matrix for transitions between pairs of states occurring at a quantized transition time of τ_k.

Optimization with PyTorch

Optimization under the LG model and under the coevolution model both require maximizing a log-likelihood of the form Eq. (7) with respect to a reversible rate matrix Q of size $s \times s$. We achieve this using first-order optimization in PyTorch [18]. The reversible rate matrix Q is parameterized by a vector $\theta \in \mathbb{R}^{s \times 1}$ and an upper triangular matrix $\Theta \in \mathbb{R}^{s \times s}$, as follows: letting $\pi = \mathrm{SoftMax}(\theta)$ and $S = \mathrm{SoftPlus}\left(\Theta + \Theta^{\top}\right)$, we take the off-diagonal entries of Q to be $Q_{ij} = \sqrt{\frac{\pi_j}{\pi_i}} S_{ij}$ (incidentally, note that with this parameterization, π is the stationary distribution of Q since $\pi_i Q_{ij} = \pi_j Q_{ji}$ holds). The diagonal entries of Q are then uniquely determined. With this parameterization, optimization of Q translates to unconstrained optimization of θ and Θ. To ensure that the loss is on the same scale regardless of the dataset size, we divide the log-likelihood in Eq. (7) by the total number of transitions, that is to say the sum of all the entries in all the C_k matrices. For this we use the Adam optimizer [27] with a learning rate of 0.1 and full batch iteration. We train the single-site model for $g = 2000$ epochs and the coevolution model for $g = 500$ epochs, and take the iteration with the lowest training loss as the final estimate. The matrix exponential layer is part of the PyTorch API and is based on an optimized Taylor approximation [28]. When benchmarking the performance of CherryML and EM as seen in Figure 1b,c,e and Extended Data Fig. 3 we use 1 CPU core for CherryML to make sure of a fair comparison of runtime between methods. Otherwise, we use 2 CPU cores to train the single-site model and 8 CPU cores to train the coevolution model. We found these to be the optimal number of cores in our architecture; more CPU cores lead to communication overhead and overall slowdown. Initialization of the optimizer is described in the next section.

As a final remark, note that optimization of Eq. (7) with respect to an irreversible model is just as easy, by instead parameterizing S as $S = \mathrm{SoftPlus}(\Theta_1 + \Theta_2)$ where $\Theta_1, \Theta_2 \in \mathbb{R}^{s \times s}$ are upper and lower triangular matrices respectively.

Initialization with JTT-IPW

We initialize θ and Θ using a novel variant of the JTT method [2] that takes into account branch lengths, which we call *JTT with inverse propensity weighting*, or JTT-IPW for short. We observed that using the JTT-IPW initialization sped up convergence by up to an order of magnitude with respect to random initialization, and we used it when benchmarking both CherryML and EM.

JTT-IPW is based on the Taylor expansion $\exp(tQ) = I + tQ + o(t)$ and on the decomposition $Q = \mathrm{diag}(\mu)\,\mathrm{CTP}$ where $\mu = (\mu_1, \ldots, \mu_s)$ is the vector of *mutabilities* and CTP is the matrix of *conditional transition probabilities* (except that the diagonal is filled with -1). Here $\mathrm{diag}(\mu)$ is the diagonal matrix with the entries of μ in the diagonal.

The JTT-IPW estimator of Q given the frequency matrices $(C_1, \ldots, C_b)$ and the transition times $(\tau_1, \ldots, \tau_b)$ is constructed as follows. First, to ensure that all quantities below are well defined, we add a small number of pseudocounts ϵ to the data entry-wise: $C_k \leftarrow C_k + \epsilon$ where $\epsilon = 10^{-8}$. Next, to ensure a reversible estimate of Q regardless of what the count matrices

are, we symmetrize the count matrices via $C_k \leftarrow \frac{1}{2}(C_k + C_k^\top)$. Now let $F = \sum_{k=1}^{b} C_k$ be the total transition frequency matrix. We first estimate the conditional transition probability matrix $\widehat{\mathrm{CTP}} \in \mathbb{R}^{s \times s}$ as follows:

$$\widehat{\mathrm{CTP}}_{i,j} = \frac{F_{i,j}}{\sum_{k \neq i} F_{i,k}} \text{ for } i \neq j\,. \tag{9}$$

We set $\widehat{\mathrm{CTP}}_{i,i} = -1$ such that each row of $\widehat{\mathrm{CTP}}$ adds up to zero, as in a rate matrix. Finally, we estimate the mutability $\widehat{\mu}_i$ of each state using inverse propensity weighting as follows:

$$\widehat{\mu}_i = \frac{\sum_{k=1}^{b} \frac{1}{\tau_k} \sum_{j \neq i} (C_k)_{i,j}}{\sum_{k=1}^{b} \sum_{j} (C_k)_{i,j}}\,. \tag{10}$$

Finally, the JTT-IPW estimator of Q is given by:

$$\widehat{Q}_{\mathrm{JTT-IPW}} = \mathrm{diag}(\widehat{\mu})\widehat{\mathrm{CTP}}\,. \tag{11}$$

One can verify that $\widehat{Q}_{\mathrm{JTT-IPW}}$ is a reversible rate matrix by noting that since $F_{i,j} = F_{j,i}$, then $\widehat{Q}_{\mathrm{JTT-IPW}}$ satisfies the detailed balance equation $\pi_i\left(\widehat{Q}_{\mathrm{JTT-IPW}}\right)_{i,j} = \pi_j\left(\widehat{Q}_{\mathrm{JTT-IPW}}\right)_{j,i}$ for all states i, j, where

$$\pi_i \propto \frac{\left(\sum_{k \neq i} F_{i,k}\right)\left(\sum_{k=1}^{b} \sum_{p} (C_k)_{i,p}\right)}{\left(\sum_{k=1}^{b} \frac{1}{\tau_k} \sum_{p \neq i} (C_k)_{i,p}\right)}\,. \tag{12}$$

The reversibility of $\widehat{Q}_{\mathrm{JTT-IPW}}$ and the fact that all off-diagonal entries of $\widehat{Q}_{\mathrm{JTT-IPW}}$ are positive (thanks to the pseudocounts) ensure that θ, Θ can be solved for. Concretely, we first solve for the stationary distribution π of $\widehat{Q}_{\mathrm{JTT-IPW}}$ via an SVD decomposition. Then, we compute $S = \sqrt{\pi}^\top \widehat{Q}_{\mathrm{JTT-IPW}} \sqrt{\frac{1}{\pi}}$. From π and S we can compute θ and Θ by taking $\theta_i = \log(\pi_i)$ and $\Theta_{i,j} = \log(\exp(S_{i,j}) - 1)\mathbb{1}\{i < j\}$.

As for the intuition of why JTT-IPW works well as an initialization: as dataset size increases (in a suitable manner), one can show that $\widehat{\mathrm{CTP}}$ converges to the true conditional probability transition matrix. The mutabilities $\widehat{\mu}$ exhibit bias but it tends to 0 as branch lengths tend to 0.

Computing the Count Matrices C_k

The computational bottleneck of the CherryML optimizer is typically computing the count matrices C_k. However, this is a simple and embarrassingly parallel task, for which we wrote a fast distributed-memory C++ implementation using MPI. To compute the transition frequency matrices C_k from the m multiple sequence alignments D_i and trees T_i using p MPI ranks, we give each worker i approximately m/p families for which it computes

the count matrices $C_k^{(i)}$. Once all workers are done, we compute C_k as $C_k = \sum_{i=1}^{p} C_k^{(i)}$. Time quantization is performed with binary search, introducing the $\log b$ factor in the $\mathcal{O}(mnl \log b)$ time complexity.

When benchmarking the performance of CherryML and EM as seen in Figure 1b,c,e and Extended Data Fig. 3 we used 1 CPU core to count transitions for CherryML. Otherwise, we used 8 CPU cores.

FastTree

We used FastTree [20] to estimate trees. FastTree was compiled to use double-precision arithmetic with:

```
gcc -DNO_SSE -DUSE_DOUBLE -03 -finline-functions -funroll-loops
```

The number of rate categories was set using the `-cat` argument.

Running XRATE

We used the EM implementation provided by the XRATE package [16], which was used in the LG paper [5]. We ran XRATE with the following arguments:

```
-f 3 -mi 0.000001
```

We set `-mininc 0.000001` to ensure that EM converges to a good solution; a more lenient threshold of 10^{-5} produces inferior results on our simulated data benchmark of Figure 1c. We further used `-f 3` to require 3 iterations without meeting the convergence threshold to terminate. XRATE was run with ground truth trees specified in the Stockholm file format with the `#=GF NH` identifier. Site rate variation was handled by first splitting each MSA into one MSA per site rate, and scaling the tree accordingly by the site rate for each sub-MSA (this is the reduction described in the LG paper [5]).

MSA Preprocessing

We preprocessed the MSAs in the Pfam dataset [19] by subsetting only sites that were aligned against residues in the reference sequence (which is the sequence for which we have structure data). In other words, we removed all sites aligned against gaps in the reference sequence. The length of the resulting MSA is therefore equal to the length of the reference sequence (which has no gaps).

Protocol for Figure 1b,c

After preprocessing the MSAs from Yang *et al.* [19] as described above, among all 15,051 families, we selected those with proteins having length between 190 and 230 sites inclusive, and at least 128 sequences. We then subsampled each MSA down to 128 sequences uniformly at random, making sure to sample the reference sequence. We then ran FastTree [20] with the LG matrix and to estimate trees for each family. We did not use site rate

variation. Having estimated these realistic-looking trees, we then proceeded to simulate MSAs for each family. To do this, we simulated each site independently under the LG matrix. The root state was sampled from the stationary distribution of the LG matrix. Having simulated MSAs for all families, we then selected a random increasing sequence of family sets $\mathcal{F}_4 \subset \mathcal{F}_8 \subset \cdots \subset \mathcal{F}_{1024}$ which were used to train CherryML and EM. CherryML and EM were run with access to the ground truth trees. For CherryML, we formed the composite log-likelihood as described in CherryML for the Coevolution Model and optimized for Q as described in Optimization with PyTorch. For EM, we used the XRATE package as described in Running XRATE. Our simulation scheme ensures that all MSAs are roughly of the same size, such that doubling the number of families is approximately equal to doubling the amount of Fisher information, which is important to obtain a reliable estimate of the relative efficiency of CherryML as compared to EM.

Protocol for Figure 1d

After preprocessing the MSAs from Yang *et al.* [19] as described in MSA Preprocessing, for each of the 15,051 families, we subsampled them down to 1,024 sequences uniformly at random, making sure to sample the reference sequence. We then ran FastTree [20] with the LG matrix and 4 rate categories to estimate trees for each family as well as the site-specific rates. Having estimated these realistic-looking trees and site-specific rates, we then proceeded to simulate MSAs for each family. To do this, we simulated each site independently under the LG matrix and with the site rates estimated by FastTree. The root state was sampled from the stationary distribution of the LG matrix. CherryML was run with access to the ground truth trees and site rates. We formed the composite log-likelihood as described in CherryML for the Coevolution Model and optimized for Q as described in Optimization with PyTorch. We explored varying the number of quantization points b while keeping the center value $\tau_{\lfloor b/2 \rfloor} = 0.03$ and the range $[\tau_1, \tau_b]$ approximately the same for all quantization points. For this, we chose the geometric increments 445.79, 21.11, 4.59, 2.14, 1.46, 1.21, 1.1, 1.048, 1.024 for quantization points 3, 5, 9, 17, 33, 65, 129, 257, 513 respectively.

Protocol for Figure 1e

We followed the testing protocol described in the LG paper [5] to evaluate rate matrices. This means running PhyML [6] with the following arguments:

```
--nclasses 4 --datatype aa --pinv e --r_seed 0 --bootstrap 0 -f m \
--alpha e --print_site_lnl
```

The average improvement in per-site log-likelihood over JTT is shown. Thus, we evaluated the following four rate matrices: JTT [2], WAG [3], LG [5], and LG re-estimated using our CherryML method. To re-estimate LG using the CherryML method, we first used FastTree [20] with 4 rate categories as described in FastTree to estimate trees for each family as well as site-specific rates. The uninformative uniform rate matrix was used in FastTree. We then formed the composite loglikelihood as described in CherryML for the Coevolution Model and optimized for Q as described in Optimization with PyTorch. The process was

repeated, this time using the estimated Q to estimate trees with FastTree. This can be seen as coordinate ascent in tree and rate matrix space. The process was repeated 3 times until convergence, as typical.

Determining Contacting Sites

To train our coevolution model we used the Pfam dataset with structure data from Yang *et al.* [19]. A pair of sites was considered a non-trivial contact if (i) the distance between the beta carbons was less than 8 angstrom, and (ii) the distance in primary sequence was at least 7.

Protocol for Figure 2c,d

After preprocessing the MSAs from Yang *et al.* [19] as described in MSA Preprocessing, for each of the 15,051 families, we subsampled them down to 1,024 sequences uniformly at random, making sure to sample the reference sequence. We then ran FastTree [20] with the LG matrix and 1 rate category to estimate trees for each family. We determined for each family which pairs of sites were in non-trivial contact as described in Determining Contacting Sites. We then used a maximal matching to pair up sites that were in contact. Finally, we formed the composite log-likelihood as described in CherryML for the Coevolution Model and optimized for Q as described in Optimization with PyTorch. The independent model was obtained by training a single-site model on sites with at least one non-trivial contact, and then taking the product of the chain with itself.

Protocol for Figure 2a,b

After preprocessing the MSAs from Yang *et al.* [19] as described in MSA Preprocessing, for each of the 15,051 families, we subsampled them down to 1,024 sequences uniformly at random, making sure to sample the reference sequence. We then ran FastTree [20] with the LG matrix and 1 rate category to estimate trees for each family. Having estimated these realistic-looking trees, we then proceeded to simulate MSAs for each family. To do this, we first determined for each family which pairs of sites were in non-trivial contact as described in Determining Contacting Sites. We then used a maximal matching to pair up sites that are in contact. Sites that are in contact were simulated under the coevolution rate matrix Q estimated from real data as described in Protocol for Figure 2c,d. The root state was drawn from the stationary distribution of Q. CherryML was run with access to the ground truth trees.

Protocol for Figure 2e

After preprocessing the MSAs from Yang *et al.* [19] as described in MSA Preprocessing, for each of the 15,051 families, we subsampled them down to 1,024 sequences uniformly at random, making sure to sample the reference sequence. We then ran FastTree [20] with the LG matrix and 20 rate categories to estimate trees for each family as well as the site-specific rates. We determined for each family which pairs of sites were in non-trivial contact as described in Determining Contacting Sites.

Extrapolating the Runtime of Traditional Methods for Learning a Coevolution Model

To estimate the time that it would take for traditional methods such as zeroth-order optimization or EM to estimate a 400 × 400 coevolution rate matrix, we performed the following extrapolation. From Figure 1b we observe that it takes EM as implemented by XRATE [16] around 6.5 CPU-hours to learn a single-site model on 1,024 families with 128 sequences each. Since the runtime of traditional methods scales linearly in the dataset size, this implies that learning a single-site model on all 15,051 families with approximately 1,024 sequences each would take on the order of $6.5 \times \frac{15,051}{1,024} \times \frac{1,024}{128} \approx 750$ CPU-hours. However, runtime scales quadratically in the state space size, which means that increasing the state space size from $s = 20$ to $s = 20$ increases runtime by a factor of $\frac{400^2}{20^2} = 400$. As a result, we estimate that learning a coevolution model on all 15,051 families with approximately 1,024 sequences each would take on the order of $6.5 \times \frac{15,051}{1,024} \times \frac{1,024}{128} \times \frac{400^2}{20^2} \approx 300000$ CPU-hours with traditional methods. The latter is approximately 35 CPU-years.

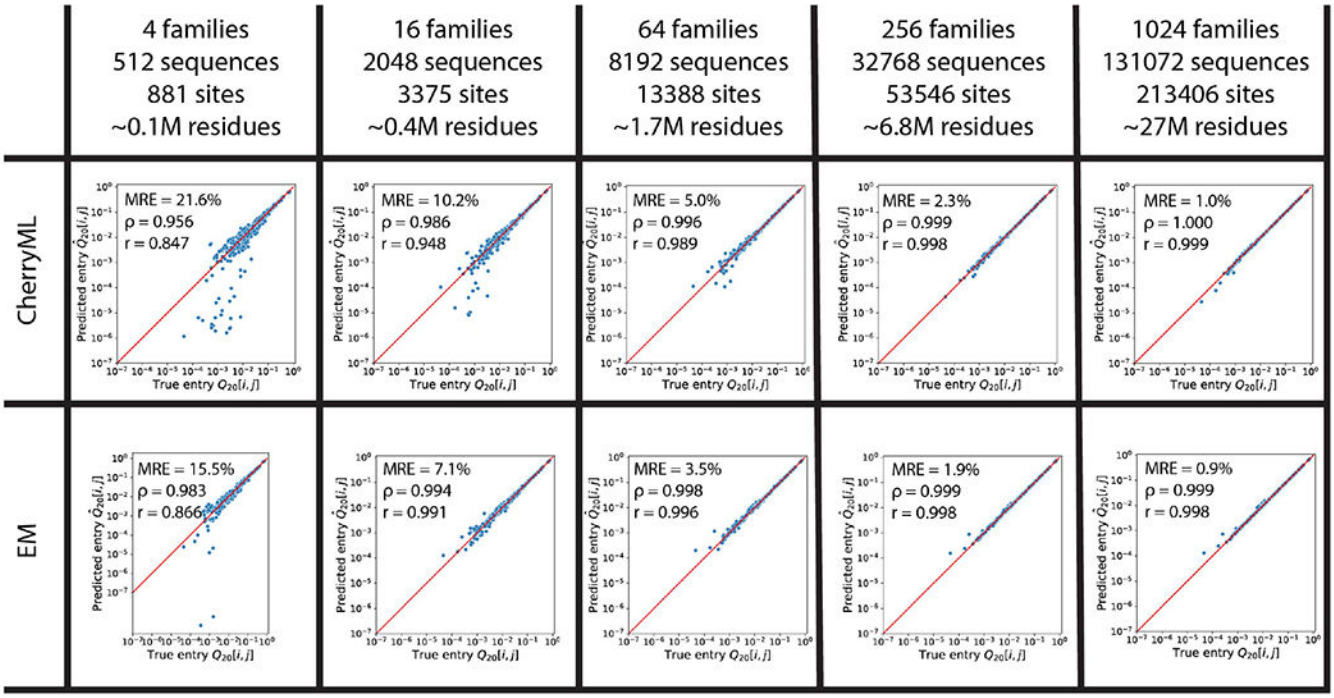
Protocol for Extended Data Fig. 3

We used the same training procedure as for Figure 1e, except that we started from the more informative LG rate matrix and thus performed only one additional round of coordinate ascent in tree and rate matrix space. Held-out log-likelihood evaluation was performed with FastTree using the discrete Gamma model with 4 rate categories.

Hardware Configuration

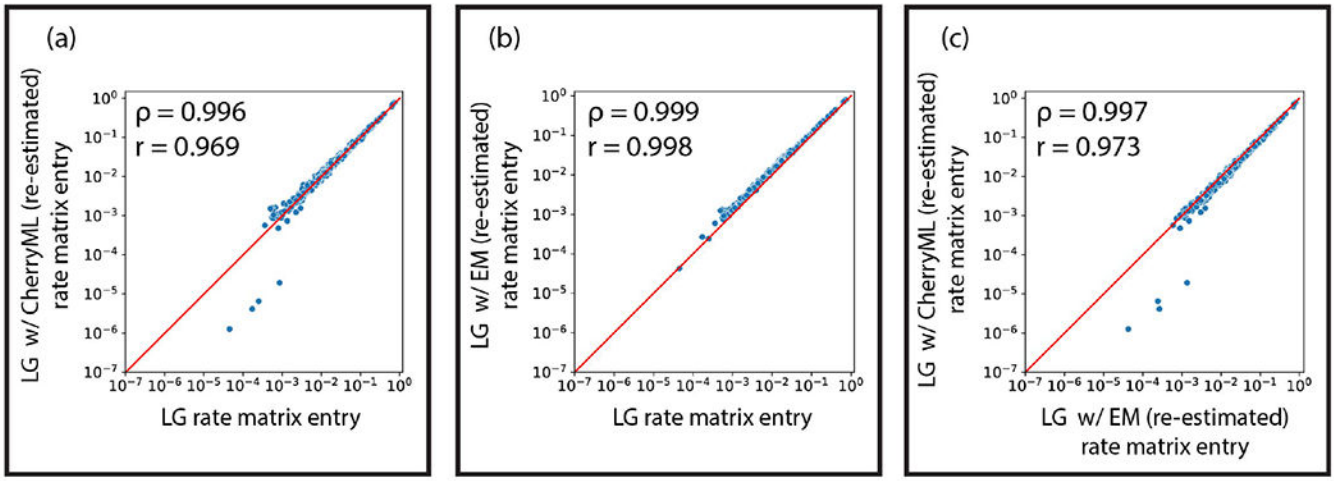
We used a node in Berkeley's Savio cluster with 40 Intel Xeon Skylake 6230 @ 2.1 GHz cores and 384 GB RAM (which far exceeds our needs).

Extended Data



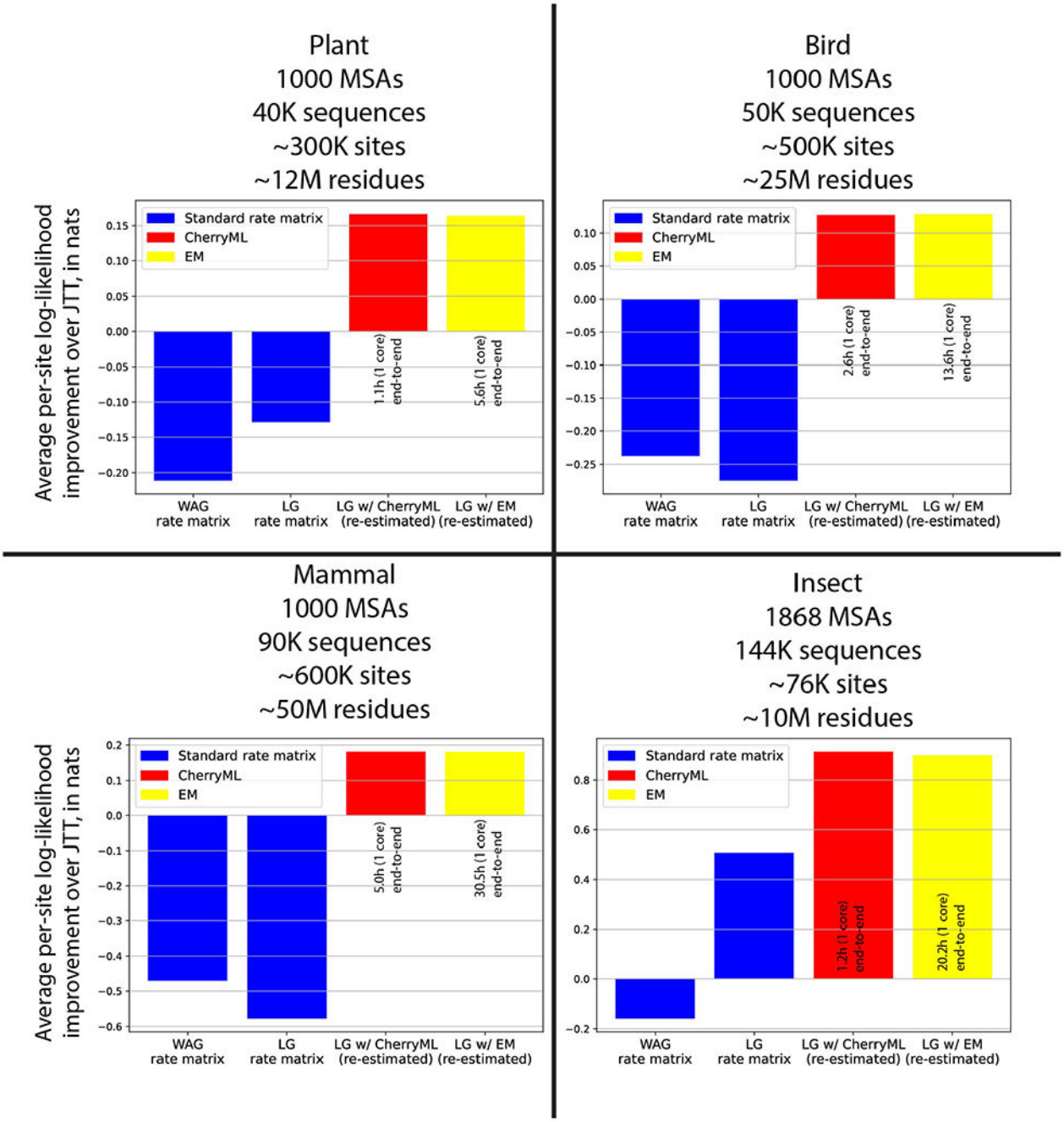
Extended Data Fig. 1: Plot of true versus estimated rate matrix entries for Figure 1b,c.

For a select number of families (the multiples of 4), we plot the true versus estimated rate matrix entries. MRE stands for median relative error; ρ is Spearman's rank correlation; r is Pearson correlation. For reference, we also indicate the total number of sequences, sites and residues in each dataset. As more data become available, estimation accuracy increases for both methods. Importantly, the loss of statistical efficiency of CherryML with respect to EM is relatively small (an estimated ≈ 50% as seen in Figure 1c). Interestingly, with small dataset sizes, the smallest transition rates tend to be underestimated by both methods, possibly because no (or relatively too few) transitions between these states are observed.



Extended Data Fig. 2: Plot of rate matrix estimates from Figure 1e.

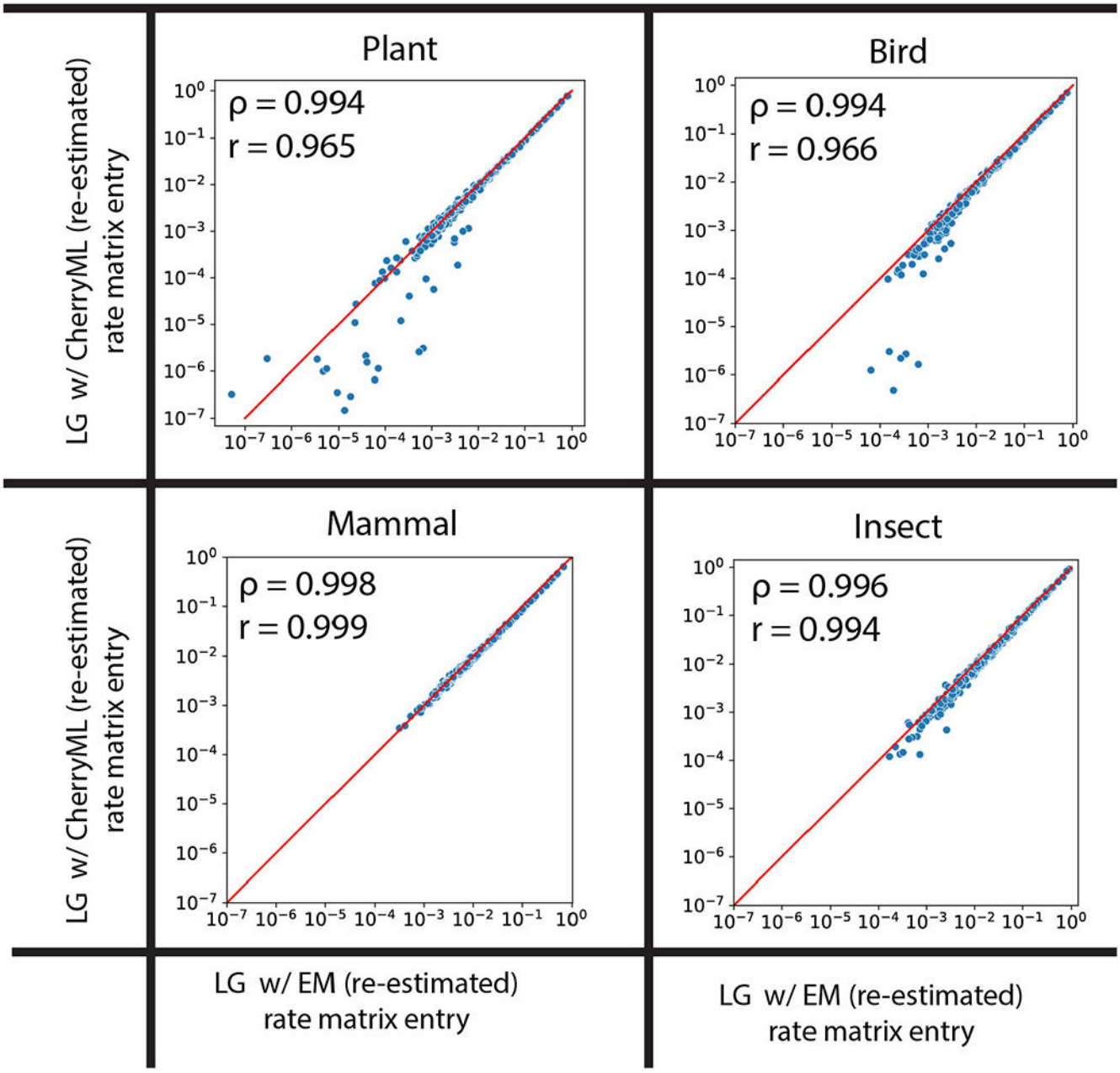
We see that the entries of the LG rate matrix [5] and re-estimates with CherryML and EM are quite similar. The only noticeable differences correspond to four of the rarest (and thus harder to estimate) rates, which are between C and E, and C and K (in both directions). It is possible that CherryML is underestimating these rates, but since there is no ground truth and the model may be misspecified (as these are real data estimates), we cannot say with certainty; bootstrap estimates of variance could partially answer this question. For reference, the dataset size in terms of number of families, sequences, sites and residues is: 3, 412, ≈ 50, 000, ≈ 600, 000 and ≈ 6.5M respectively. In principle, this is roughly comparable in size to 256 families in the (well-specified) simulations from Figure 1b,c, where both EM and CherryML are accurate even for small rates, as seen in Extended Data Fig. 1. However, these direct comparisons of dataset size might overestimate the information content of real datasets. Indeed, it is possible that the effective amount of information for these small rates is more comparable to 16 families in the simulations from Figure 1b,c, where CherryML produces more underestimates than does EM.



Extended Data Fig. 3: CherryML matches EM accuracy on diverse datasets.

On diverse datasets from the QMaker paper [9], CherryML matches the accuracy of the EM method. The end-to-end runtime of each approach (including tree estimation) is shown. The runtime of the CherryML optimizer was in all cases negligible (less than 5 minutes), therefore end-to-end runtime was dominated by phylogeny reconstruction with FastTree, which took a few CPU hours depending on the dataset. In contrast, for the EM approach, the EM optimizer dominated runtime, leading to an overall slowdown of 5-20 fold in end-to-end runtime compared to the CherryML approach. Since tree estimation is embarrassingly

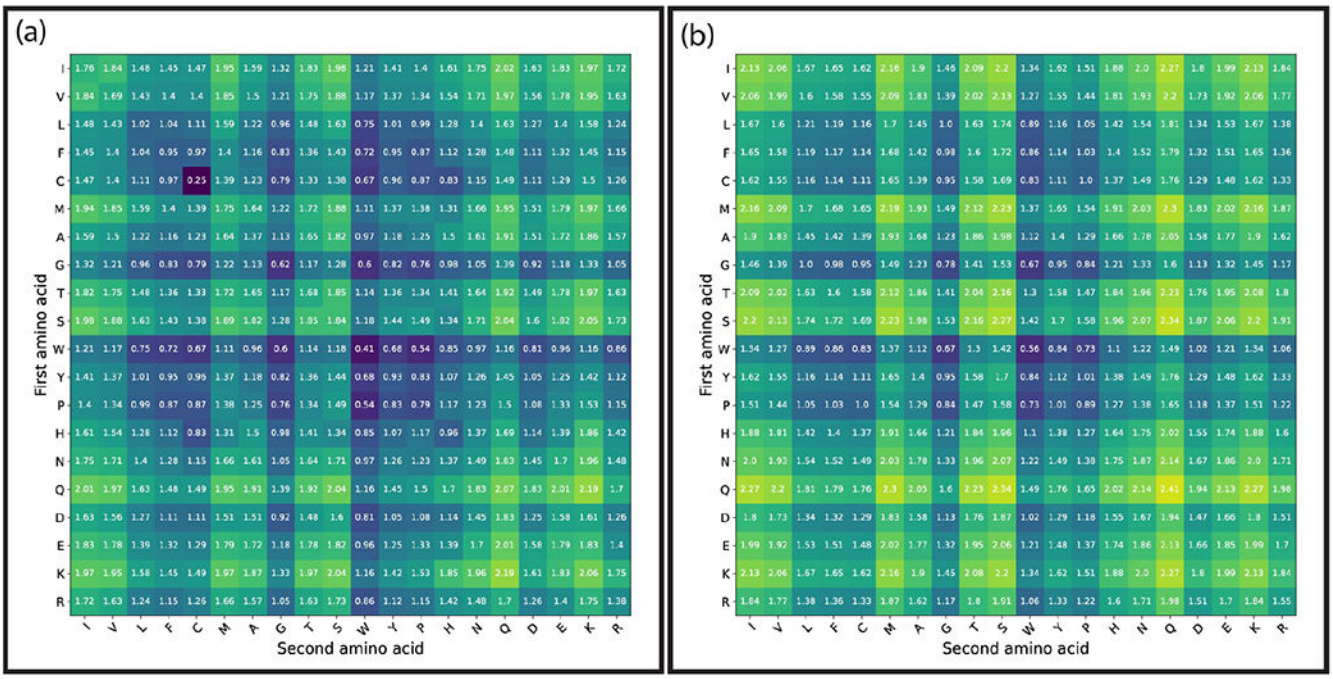
parallel, end-to-end estimation with the CherryML method using 32 CPU cores takes only a few minutes on all of these datasets. The diversity of the datasets means that LG is no longer the best fit rate matrix compared to JTT and WAG. In fact, JTT is preferred in three of these datasets. This highlights the need to estimate new rate matrices for improved phylogenetic inference in specific applications [9]. Training dataset sizes are included for reference.



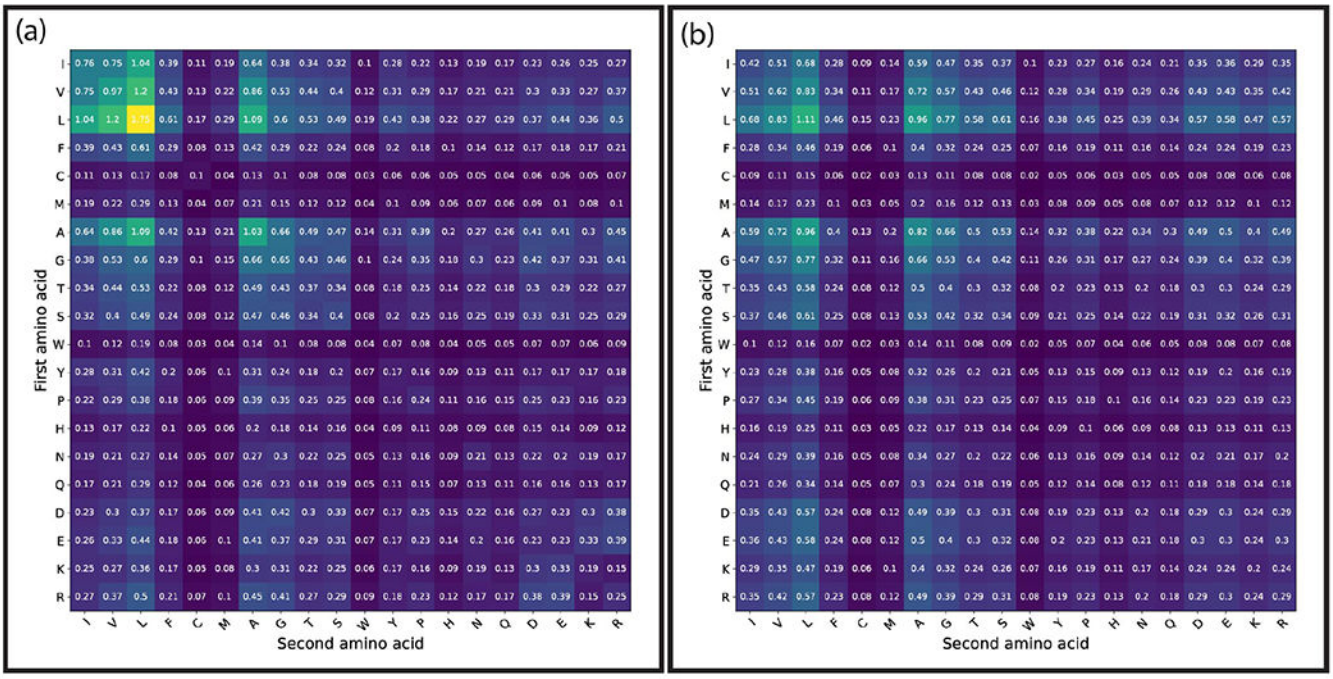
Extended Data Fig. 4: Plot of rate matrix estimates from Extended Data Fig. 3.

Similarly to Extended Data Fig. 2, CherryML and EM agree on most rates, except for some of the smallest (harder to estimate) rates, where CherryML usually reports smaller rates. It is possible that these are underestimates from CherryML, for instance if the information

content for these small rates is similar to 16 families in Extended Data Fig. 1, where CherryML produces more underestimates compared to EM.



Extended Data Fig. 5: Comparison of mutation rates.
(a) Our 400 × 400 co-evolution model. (b) Independent-sites model.



Extended Data Fig. 6: Comparison of stationary distributions.

(a) Our 400 × 400 coevolution model. (b) Independent-sites model.

Acknowledgments

Sebastian Prillo would like to acknowledge Alfredo Umfurer for many helpful discussions on software design. We would like to acknowledge the reviewers for their helpful feedback. We thank Olivier Gascuel for suggesting that we pair up more sequences beyond the original cherries in the trees, which empirically boosted the statistical efficiency of the method by around 10%–30% (estimated). We would also like to thank Ian Holmes, John Huelsenbeck, Neil Thomas and William DeWitt for helpful discussions. This research is supported in part by an NIH grant R35-GM134922 (Y.S.S.). The funders had no role in study design, data collection and analysis, decision to publish or preparation of the manuscript.

Data Availability

The LG paper [5] training and testing Pfam datasets consisting of 3,912 and 500 families respectively are available at: http://www.atgc-montpellier.fr/models/index.php?model=lg

The Pfam dataset with structure data from Yang *et al.* [19] consisting of 15,051 families is located at: https://files.ipd.uw.edu/pub/trRosetta/training_set.tar.gz

The QMaker [9] datasets are available at: https://figshare.com/articles/dataset/QMaker-datasets_zip/9768101

Our simulated datasets used for Figure 1b,c, Figure 1d, and Figure 2a,b are available on Zenodo at: https://zenodo.org/record/7830072#.ZDnPBuzMKTc

Instructions for how to reproduce all results in this paper using the above datasets can be found at:

References

[1]. Dayhoff MO and Schwartz RM. Chapter 22: A model of evolutionary change in proteins. In in Atlas of Protein Sequence and Structure, 1978.

[2]. Jones David T., Taylor William R., and Thornton Janet M.. The rapid generation of mutation data matrices from protein sequences. Comput. Appl. Biosci, 8(3):275–282, 1992. [PubMed: 1633570]

[3]. Whelan Simon and Goldman Nick. A General Empirical Model of Protein Evolution Derived from Multiple Protein Families Using a Maximum-Likelihood Approach. Molecular Biology and Evolution, 18(5):691–699, 05 2001. [PubMed: 11319253]

[4]. Yang Ziheng. PAML 4: phylogenetic analysis by maximum likelihood. Molecular Biology and Evolution, 24(8):1586–1591, 2007. [PubMed: 17483113]

[5]. Le Si Quang and Gascuel Olivier. An Improved General Amino Acid Replacement Matrix. Molecular Biology and Evolution, 25(7):1307–1320, 03 2008. [PubMed: 18367465]

[6]. Guindon Stéphane, Dufayard Jean-François, Lefort Vincent, Anisimova Maria, Hordijk Wim, and Gascuel Olivier. New Algorithms and Methods to Estimate Maximum-Likelihood Phylogenies: Assessing the Performance of PhyML 3.0. Systematic Biology, 59(3):307–321, 05 2010. [PubMed: 20525638]

[7]. Bouckaert Remco, Vaughan Timothy G, Barido-Sottani Joëlle, Duchêne Sebastián, Fourment Mathieu, Gavryushkina Alexandra, Heled Joseph, Jones Graham, Kühnert Denise, De Maio Nicola, et al. BEAST 2.5: An advanced software platform for Bayesian evolutionary analysis. PLoS Computational Biology, 15(4):e1006650, 2019. [PubMed: 30958812]

[8]. Minh Bui Quang, Schmidt Heiko A, Chernomor Olga, Schrempf Dominik, Woodhams Michael D, von Haeseler Arndt, and Lanfear Robert. IQ-TREE 2: New Models and Efficient Methods for

Phylogenetic Inference in the Genomic Era. Molecular Biology and Evolution, 37(5):1530–1534, 02 2020. [PubMed: 32011700]

[9]. Minh Bui Quang, Dang Cuong Cao, Vinh Le Sy, and Lanfear Robert. QMaker: Fast and Accurate Method to Estimate Empirical Models of Protein Evolution. Systematic Biology, 70(5):1046–1060, 02 2021. [PubMed: 33616668]

[10]. Yang Ziheng. Maximum likelihood phylogenetic estimation from dna sequences with variable rates over sites: approximate methods. Journal of Molecular Evolution, 39(3):306–314, 1994. [PubMed: 7932792]

[11]. Kalyaanamoorthy Subha, Minh Bui Quang, Wong Thomas K. F., von Haeseler Arndt, and Jermiin Lars S.. Modelfinder: fast model selection for accurate phylogenetic estimates. Nature Methods, 14(6):587–589, Jun 2017. [PubMed: 28481363]

[12]. Holmes Ian. A Model of Indel Evolution by Finite-State, Continuous-Time Machines. Genetics, 216(4):1187–1204, 12 2020. [PubMed: 33020189]

[13]. Yeang Chen-Hsiang and Haussler David. Detecting coevolution in and among protein domains. PLOS Computational Biology, 3(11):1–13, 11 2007.

[14]. Felsenstein Joseph. Maximum Likelihood and Minimum-Steps Methods for Estimating Evolutionary Trees from Data on Discrete Characters. Systematic Biology, 22(3):240–249, 09 1973.

[15]. Siepel Adam and Haussler David. Phylogenetic Estimation of Context-Dependent Substitution Rates by Maximum Likelihood. Molecular Biology and Evolution, 21(3):468–488, 03 2004. [PubMed: 14660683]

[16]. Klosterman Peter S., Uzilov Andrew V., Bendaña Yuri R., Bradley Robert K., Chao Sharon, Kosiol Carolin, Goldman Nick, and Holmes Ian. Xrate: a fast prototyping, training and annotation tool for phylo-grammars. BMC Bioinformatics, 7(1):428, 2006. [PubMed: 17018148]

[17]. Varin Cristiano, Reid Nancy, and Firth David. An overview of composite likelihood methods. Statistica Sinica, 21(1):5–42, 2011.

[18]. Paszke Adam, Gross Sam, Chintala Soumith, Chanan Gregory, Yang Edward, Devito Zachary, Lin Zeming, Desmaison Alban, Antiga Luca, and Lerer Adam. Automatic differentiation in pytorch. In Advances in Neural Information Processing Systems 30, 2017.

[19]. Yang Jianyi, Anishchenko Ivan, Park Hahnbeom, Peng Zhenling, Ovchinnikov Sergey, and Baker David. Improved protein structure prediction using predicted interresidue orientations. Proceedings of the National Academy of Sciences, 117(3):1496–1503, 2020.

[20]. Price Morgan N., Dehal Paramvir S., and Arkin Adam P.. Fasttree 2 – approximately maximum-likelihood trees for large alignments. PLoS ONE, 5(3):e9490, Mar 2010. [PubMed: 20224823]

[21]. Franzosa Eric A and Xia Yu. Structural determinants of protein evolution are context-sensitive at the residue level. Molecular Biology and Evolution, 26(10):2387–2395, 2009. [PubMed: 19597162]

[22]. Echave Julian, Spielman Stephanie J, and Wilke Claus O. Causes of evolutionary rate variation among protein sites. Nature Reviews Genetics, 17(2):109–121, 2016.

[23]. Dang Cuong, Vinh Le, Gascuel Olivier, Hazes Bart, and Le Quang. Fastmg: a simple, fast, and accurate maximum likelihood procedure to estimate amino acid replacement rate matrices from large data sets. BMC bioinformatics, 15:341, 10 2014. [PubMed: 25344302]

[24]. Canh Nguyen Duc, Dang Cuong Cao, Vinh Le Sy, Minh Bui Quang, and Hoang Diep Thi. pqmaker: empirically estimating amino acid substitution models in a parallel environment. In 2020 12th International Conference on Knowledge and Systems Engineering (KSE), pages 324–329, 2020.

[25]. Jumper John M., Evans Richard, Pritzel Alexander, Green Tim, Figurnov Michael, Ronneberger Olaf, Tunyasuvunakool Kathryn, Bates Russ, Zídek Augustin, Potapenko Anna, Bridgland Alex, Meyer Clemens, Kohl Simon A A, Ballard Andy, Cowie Andrew, Romera-Paredes Bernardino, Nikolov Stanislav, Jain Rishub, Adler Jonas, Back Trevor, Petersen Stig, Reiman David A., Clancy Ellen, Zielinski Michal, Steinegger Martin, Pacholska Michalina, Berghammer Tamas, Bodenstein Sebastian, Silver David, Vinyals Oriol, Senior Andrew W., Kavukcuoglu Koray, Kohli Pushmeet, and Hassabis Demis. Highly accurate protein structure prediction with AlphaFold. Nature, 596:583–589, 2021. [PubMed: 34265844]

[26]. Ran Jinhua, Shen Ting-Ting, Wang Ming-Ming, and Wang Xiao-Quan. Phylogenomics resolves the deep phylogeny of seed plants and indicates partial convergent or homoplastic evolution between gnetales and angiosperms. Proceedings of the Royal Society B: Biological Sciences, 285:20181012, 06 2018.

[27]. Kingma Diederik P and Ba Jimmy. Adam: A method for stochastic optimization. In Proceedings of the 3rd International Conference on Learning Representations (ICLR), 2015.

[28]. Bader Philipp, Blanes Sergio, and Casas Fernando. Computing the matrix exponential with an optimized taylor polynomial approximation. Mathematics, 7(12), 2019.

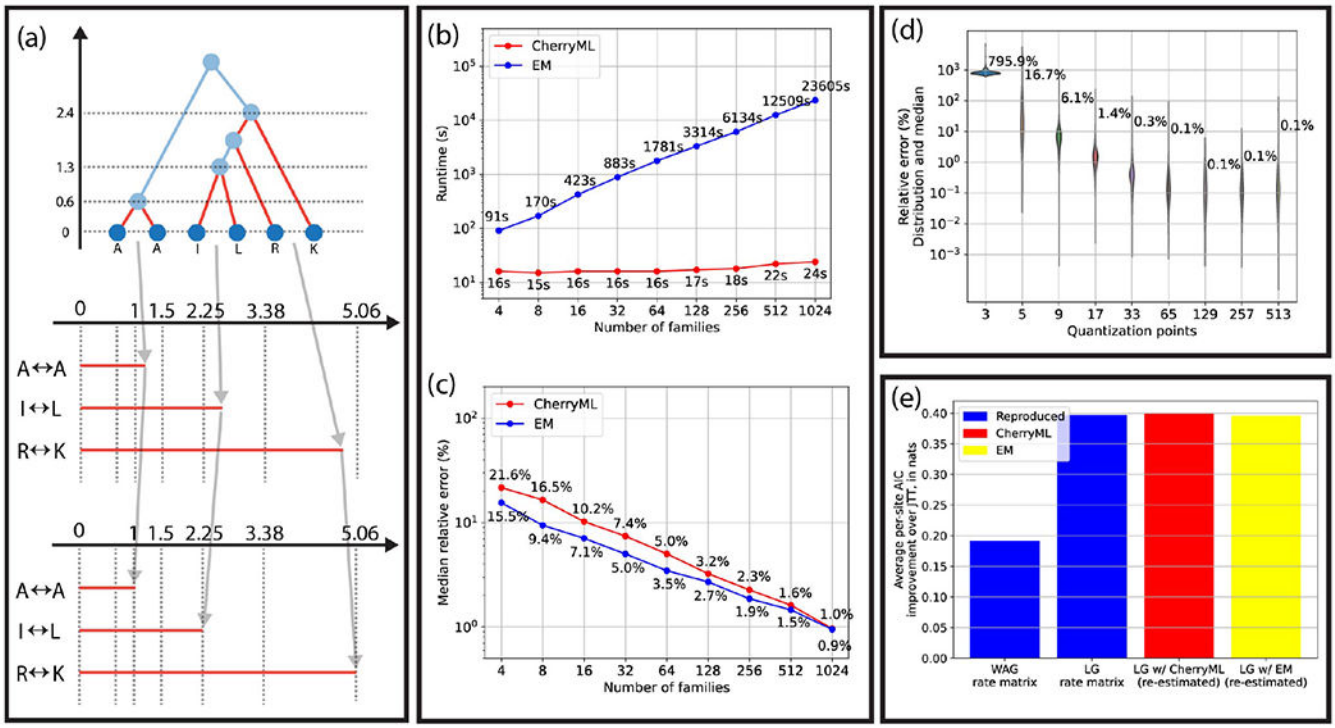


Figure 1: CherryML method applied to the LG model.

(a) Sketch of the CherryML method as applied to a reversible model: transitions between all iteratively picked cherries are treated as independent observations and quantized; here the quantization grid is {1, 1.5, 2.25, 3.38, 5.06}. (b) Runtime and (c) median estimation error as a function of sample size for our CherryML optimizer and the classical EM optimizer. The empirical loss of statistical efficiency for CherryML is relatively small (≈ 50%) while being a thousand times faster when applied to 1,024 families. Each family has 128 sequences. (d) On a large simulated dataset, time quantization error becomes negligible with as few as ≈ 100 quantization points. Distribution of relative error and median shown. (e) Using the evaluation protocol of the LG paper [5], we verified that using the CherryML optimizer produces comparable likelihoods to EM on held-out families. Here “LG rate matrix” stands for the rate matrix originally published in the LG paper [5], whereas “LG w/ CherryML (re-estimated)” and “LG w/ EM (re-estimated)” correspond to our re-implementation of the inference pipeline in the LG paper [5] while using either CherryML or EM for optimization.

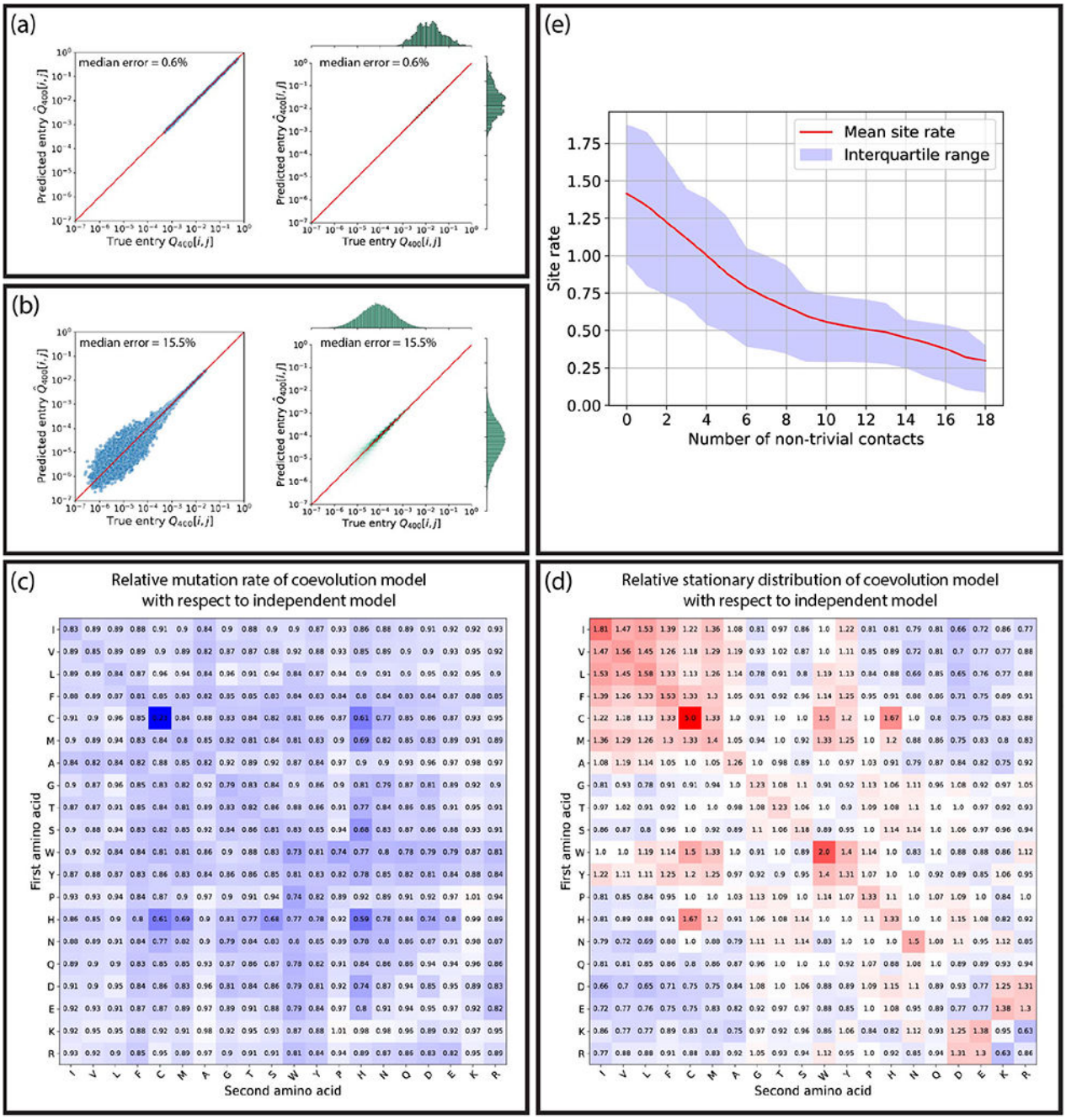


Figure 2: CherryML method applied to learn a 400 ×400 coevolution model.

Using simulated coevolution data from 15,051 Pfam families each subsampled down to 1024 sequences, we verified that our method is able to accurately estimate co-transition rates for (a) single-site transitions (such as IL ↔ IA), and (b) the more challenging joint transitions (such as KE ↔ EK). The left plot is a scatter plot which reveals outliers, while the right plot is a density plot which shows that there are few outliers. (c) Mutation rates of the coevolution model differ from that of the independent-sites model, and recapitulate known biology such as the importance of disulfide bonds by assigning a significantly lower

mutation rate to CC pairs. Residues are ordered by the hydropathy index. (d) Stationary distribution of the coevolution model differs from that of the independent-sites model, and recapitulates favorable residue pairings, such as hydrophobic amino acid pairs and electrostatically interacting pairs. (e) The more contacts a site has, the lower its mutation rate as estimated by FastTree [20].