




Automatic Differentiation is no Panacea for Phylogenetic Gradient Computation

Mathieu Fourment ¹, Christiaan J. Swanepoel^{2,3}, Jared G. Galloway⁴, Xiang Ji⁵, Karthik Gangavarapu⁶, Marc A. Suchard ^{6,7,8,*}, and Frederick A. Matsen IV ^{4,9,10,11,*}

¹Australian Institute for Microbiology and Infection, University of Technology Sydney, Ultimo, NSW, Australia

²Centre for Computational Evolution, The University of Auckland, Auckland, New Zealand

³School of Computer Science, The University of Auckland, Auckland, New Zealand

⁴Public Health Sciences Division, Fred Hutchinson Cancer Research Center, Seattle, Washington, USA

⁵Department of Mathematics, Tulane University, New Orleans, Louisiana, USA

⁶Department of Human Genetics, University of California, Los Angeles, California, USA

⁷Department of Computational Medicine, University of California, Los Angeles, California, USA

⁸Department of Biostatistics, University of California, Los Angeles, California, USA

⁹Department of Statistics, University of Washington, Seattle, Washington, USA

¹⁰Department of Genome Sciences, University of Washington, Seattle, Washington, USA

¹¹Howard Hughes Medical Institute, Fred Hutchinson Cancer Research Center, Seattle, Washington, USA

*Corresponding authors: E-mails: msuchard@ucla.edu; matsen@fredhutch.org.

Accepted: 25 May 2023

Abstract

Gradients of probabilistic model likelihoods with respect to their parameters are essential for modern computational statistics and machine learning. These calculations are readily available for arbitrary models via “automatic differentiation” implemented in general-purpose machine-learning libraries such as TensorFlow and PyTorch. Although these libraries are highly optimized, it is not clear if their general-purpose nature will limit their algorithmic complexity or implementation speed for the phylogenetic case compared to phylogenetics-specific code. In this paper, we compare six gradient implementations of the phylogenetic likelihood functions, in isolation and also as part of a variational inference procedure. We find that although automatic differentiation can scale approximately linearly in tree size, it is much slower than the carefully implemented gradient calculation for tree likelihood and ratio transformation operations. We conclude that a mixed approach combining phylogenetic libraries with machine learning libraries will provide the optimal combination of speed and model flexibility moving forward.

Key words: phylogenetics, Bayesian inference, variational inference, gradient.

Significance

Bayesian phylogenetic analysis plays an essential role in understanding how organisms evolve, and is widely used as a tool for genomic surveillance and epidemiology studies. The classical Markov chain Monte Carlo algorithm is the engine of most Bayesian phylogenetic software, however, it becomes impractical when dealing with large datasets. To address this issue, more efficient methods leverage gradient information, albeit at the cost of increased computational demands. Here we present a benchmark comparing the efficiency of automatic differentiation implemented in general-purpose libraries against analytical gradients implemented in specialized phylogenetic tools. Our findings indicate that implementing analytical gradients for the computationally intensive components of the phylogenetic model significantly enhances the efficiency of the inference algorithm.

© The Author(s) 2023. Published by Oxford University Press on behalf of Society for Molecular Biology and Evolution.

This is an Open Access article distributed under the terms of the Creative Commons Attribution License (<https://creativecommons.org/licenses/by/4.0/>), which permits unrestricted reuse, distribution, and reproduction in any medium, provided the original work is properly cited.

Introduction

Gradients (i.e. multidimensional derivatives) of probabilistic model likelihoods with respect to their unknown parameters are essential for modern computational statistics and machine learning. For example, gradient-based Hamiltonian Monte Carlo (HMC) (Neal 2011), implemented in the Stan statistical framework (Carpenter et al. 2017), is a cornerstone of the modern Bayesian statistical toolbox. Variational Bayesian (VB) inference algorithms (Blei et al. 2017), which use gradients to improve fit of a variational distribution to the posterior, are another key modern technique. In the more general setting of machine learning, gradients are used to train predictive models such as deep neural networks.

Although gradients have been considered for a long time in phylogenetics (Schadt et al. 1998; Kenney and Gu 2012), they are now becoming of central importance to enable faster approaches to Bayesian phylogenetic analysis. Bayesian methods have gained popularity among phylogenetic practitioners due to their ability to integrate multiple data sources, including ecological factors (Lemey et al. 2020) and clinical outcomes (Bedford et al. 2014) into a single analysis. A drawback of these methods is scalability, as it is well known that Bayesian phylogenetic packages, such as BEAST (Suchard et al. 2018), struggle with datasets containing thousands of sequences with moderately complex models. Bayesian phylogenetic analysis typically uses classical Markov chain Monte Carlo (MCMC) and therefore does not need to calculate computationally intensive gradients.

In order to go beyond classical MCMC, recent research has developed HMC (Fisher et al. 2021) and Variational Bayes phylogenetic analysis (Dang and Kishino 2019; Fourment and Darling 2019; Zhang and Matsen 2019; Liu et al. 2021; Moretti et al. 2021; Ki and Terhorst 2022; Koptagel et al. 2022; Zhang and Matsen 2022). These methods require fast and efficient gradient calculation algorithms to give viable alternatives to MCMC. Correspondingly, recent work has developed fast algorithms and implementations of phylogenetic likelihood gradient calculation (Ji et al. 2020) in the BEAGLE (Ayres et al. 2019) library.

Outside of phylogenetics, gradient-based analysis has also exploded in popularity, in part driven by easy to use software libraries that provide gradients via automatic differentiation (AD). AD libraries “record” function compositions, have gradients on hand for component functions, and combine these simple gradients together via the chain rule (see Margossian 2019 for a review). This work has, remarkably, been extended to many computable operations that are not obviously differentiable such as dynamic control flow and unbounded iteration (Yu et al. 2018). These libraries, exemplified by TensorFlow (Abadi 2016) and

PyTorch (Paszke et al. 2019), are often developed by large dedicated teams of professional programmers.

The combination of these various advances raises a number of questions. Can we rely on AD exclusively in phylogenetics, and avoid calculating gradients using hand-crafted algorithms? How do AD algorithms scale when presented with interdependent calculations on a tree? Does performance depend on the package used?

In this paper, we address these questions by performing the first benchmark analysis of AD versus carefully implemented gradient algorithms in compiled languages. We find that AD algorithms vary widely in performance depending on the backend library, the dataset size and the model/function under consideration. All of these AD implementations are categorically slower than libraries designed specifically for phylogenetics; we do, however, find that they appear to scale roughly linearly in tree size. Moving forward, these results suggest an architecture in which core phylogenetic likelihood and branch-length transformation calculations are performed in specialized libraries, whereas rich models are formulated, and differentiated, in a machine learning library such as PyTorch or TensorFlow.

Results

Overview of Benchmarking Setup

To coherently describe our results, we first provide a succinct overview of the phylogenetic and machine learning packages that we will benchmark as well as the computational tasks involved.

We benchmark two packages where the core algorithm implementation is specialized to phylogenetics: BEAGLE (Ji et al. 2020), wrapped by our Python-interface C++ library `bito`, as well as `physher` (Fourment and Holmes 2014). The `bito` library also efficiently implements gradients of the ratio transformation, following (Ji et al. 2021), for unconstrained node-height optimization. We compare these to the most popular AD libraries available, namely TensorFlow (Abadi 2016), PyTorch (Paszke et al. 2019), JAX (Bradbury et al. 2018), and Stan (Carpenter et al. 2017). These are leveraged in phylogenetics via `treeflow`, `torchtree`, `phylojax`, and `phylostan` (Fourment and Darling 2019), respectively. When using AD, these programs make use of reverse-mode automatic differentiation. Every program uses double precision unless specified otherwise.

We divide the benchmarking into two flavors: a “micro-” and “macro-” benchmark. The macrobenchmark is meant to mimic running an actual inference algorithm, though stripped down to reduce the burden of implementing a complex model in each framework. Specifically, we infer parameters of a constant size coalescent process, strict clock, as well as node heights under a typical continuous-time Markov chain (CTMC) model for character substitution

along an unknown phylogeny. Every implementation uses the automatic differentiation variational inference (ADVI) framework (Kucukelbir et al. 2017) to maximize the evidence lower bound (ELBO) over 5000 iterations. *A priori* we assume the CTMC substitution rate is exponentially distributed with mean 0.001 and we use the Jeffrey's prior for the unknown population size parameter.

The microbenchmark, on the other hand, is meant to identify which parts of a phylogenetic model are the most computationally expensive in the context of gradient-based inference. This involves evaluating likelihoods and functions used in phylogenetic analysis and calculating their gradient (1) the phylogenetic likelihood, (2) the coalescent likelihood, (3) node-height transform, and (4) the determinant of the Jacobian of the node-height transform. Specifically, these tasks are:

1. **Phylogenetic likelihood:** the likelihood of observing an alignment under the Jukes–Cantor substitution model (Jukes and Cantor 1969) is efficiently calculated using the pruning algorithm (Felsenstein 1981) requiring $\mathcal{O}(N)$ operations where N is the number of taxa. In this benchmark, the derivatives are taken with respect to the branch lengths. Although a naive implementation of the gradient calculation requires $\mathcal{O}(N^2)$ calculations, efficient implementations (Fourment and Holmes 2014; Ji et al. 2020) necessitate only $\mathcal{O}(N)$ operations. We also benchmark the tree likelihood using the GTR substitution model. The gradient with respect to the GTR parameters is calculated analytically in `physher` while `bito` utilizes finite differences. Analytical gradients of the tree likelihood require $\mathcal{O}(N)$ operations for each of

the eight free parameters while numerical gradients require two evaluations of the tree likelihood per parameter.

2. **Coalescent likelihood:** the likelihood of observing a phylogeny is calculated using the constant size population coalescent model (Kingman 1982). The gradient with respect to the node heights and the population size parameter requires $\mathcal{O}(N)$ time.
3. **Node-height transform:** Node ages of time trees need to be reparameterized in order to perform unconstrained optimization (Fourment and Holmes 2014; Ji et al. 2021). Evaluating this function requires a single preorder traversal and requires $\mathcal{O}(N)$ operations.
4. **Determinant of the Jacobian of the node-height transform:** The transformation of the node ages requires an adjustment to the joint density through the inclusion of the determinant of the Jacobian of the transform (Fourment and Darling 2019). The Jacobian is triangular and the determinant is therefore straightforward to compute. Although calculating its gradient analytically is not trivial, requiring $\mathcal{O}(N^2)$ calculations, recent work (Ji et al. 2021) proposed an $\mathcal{O}(N)$ algorithm. The derivatives are taken with respect to the node heights.

AD Implementations Vary Widely in Performance, and Custom Gradients are Far Faster

We find that on the macrobenchmark, AD implementations vary widely in their speed (fig. 1). This is remarkable given that these are highly optimized libraries doing the same flavor of operations. Specifically, both just-in-time (JIT)

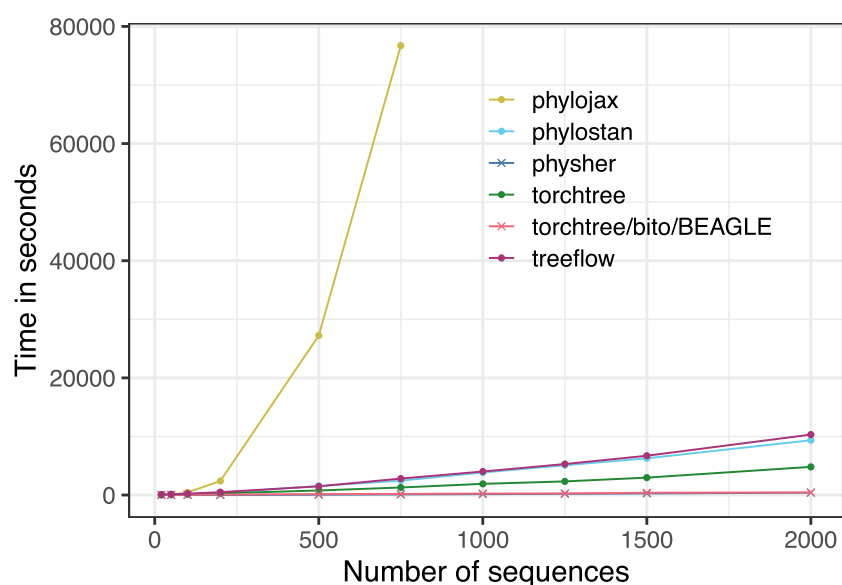


FIG. 1.—Speed of implementations for 5000 iterations of variational time-tree inference with a strict clock. See [supplementary figure S1, Supplementary Material online](#) for results without `phylojax`.

compiled JAX and compiled TensorFlow use XLA as a backend, although they have strikingly different performance. (We note that this is now a known issue with JAX <https://github.com/google/jax/issues/10197>.) Specifically, JAX was the only package that clearly scales quadratically in the number of tips. Moreover, PyTorch was several times faster than TensorFlow for our tasks of interest, which was surprising to us because of PyTorch uses a dynamic computation graph. Results for `phylojax` with datasets larger than 750 sequences are not reported as they exceeded the maximum allocated computation time.

None of these AD libraries approach the speed of hand-coded phylogenetic gradients. The BEAGLE gradients wrapped in `bito` and gradients computed in `physher` show comparable performance, which are at least eight times the speed of the fastest AD implementation ([supplementary fig. S2, Supplementary Material online](#)).

As expected, memory usage of the pure C program `physher` is the smallest, while `torchtree` is less memory heavy than `treeflow` and `phylostan`'s memory usage increases significantly more rapidly ([supplementary fig. S3, Supplementary Material online](#)). It is worth noting that `bito` noticeably decreases the memory usage of `torchtree`.

Overall using a specialized library for the tree likelihood within a Python program greatly improves the performance of a program making use of gradient-based optimization (e.g. ADVI, HMC) while incurring a small performance and memory cost compared to a fully C-based tool.

Relative Performance of AD Depends on the Task

To break down our inferential task into its components, we then performed a “microbenchmark” divided into the ingredients needed for doing gradient-based inference ([fig. 2 and supplementary fig. S4, Supplementary Material online](#)). See Methods for a precise description of the individual tasks. Across tasks, we see the following shared features. The specialized phylogenetic packages (`bito`/BEAGLE and `physher`) perform similarly to one another and are consistently faster than the AD packages, except for the Jacobian task. As expected, the tree likelihood is the computational and memory bottleneck ([fig. 2 and supplementary fig. S3, Supplementary Material online](#)) in phylogenetic models and efficient gradient calculation are warranted. TensorFlow-based `treeflow` was the slowest implementation across the board after excluding JAX.

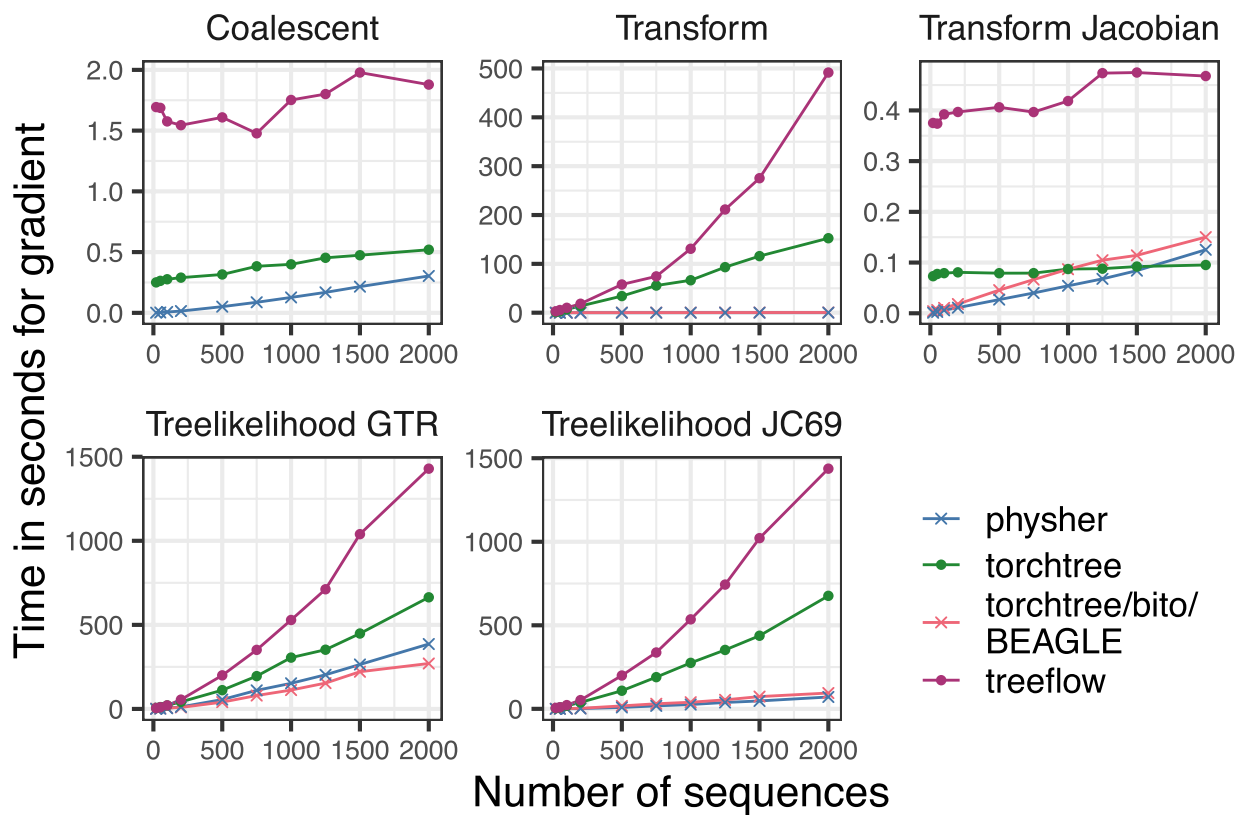


FIG. 2.—Speed of implementations for the gradient of various tasks needed for inference. See text for description of the tasks. JAX is excluded from this plot due to slow performance stretching the y-axis; see [supplementary figure S5, Supplementary Material online](#) for JAX. See [supplementary figure S6, Supplementary Material online](#) for function evaluations.

Table 1

Code Availability and Version Number of Each Phylogenetic Program. Version Identifiers Correspond to Git Tags.

Program	Availability	Version
bito	https://github.com/phylovi/bit0	autodiff-benchmark
phylojax	https://github.com/4ment/phylojax	v1.0.1
phylostan	https://github.com/4ment/phylostan	v1.0.5
physher	https://github.com/4ment/physher	v2.0.0
torchtree	https://github.com/4ment/torchtree	gradient-benchmark
torchtree-bit0	https://github.com/4ment/torchtree-bit0	gradient-benchmark
treeflow	https://github.com/christiaanjs/treeflow	autodiff-benchmark

The AD programs also performed significantly worse in the node-height transform and tree likelihood tasks. Function calls in python are notoriously more expensive than in C and C++, potentially explaining the decrease in performance for algorithms involving a tree traversal. In addition, the tree likelihood implementations in `BEAGLE` and `physher` are highly optimized with SSE vectorization (Ayres et al. 2019) and manual loop unrolling.

The calculations of the coalescent function and its gradient were slightly faster in `physher` than in `torchtree`, although the difference was slight. The ratio transform has nontrivial computational expense—comparable to the phylogenetic likelihood gradient—in AD packages; however, specialized algorithms for calculating these gradients scale much better. Interestingly, for large datasets, `torchtree` outperforms the specialized phylogenetic packages for the Jacobian ratio transform gradient calculation. Since this is the fastest task, the overall execution time is not, however, significantly impacted.

The phylogenetic gradient is approximately linear for packages other than JAX (supplementary fig. S7, Supplementary Material online), although the specialized phylogenetic packages are about 10 times faster. For the GTR calculation, we actually compare two flavors of evaluation: finite differences for `bito` and analytic gradients for `physher`. As expected, `bito` is increasingly faster than `physher` as the datasets increase in size.

With the exception of the tree likelihood, JAX's JIT capabilities greatly improved the performance of the algorithms in the microbenchmark (supplementary fig. S8, Supplementary Material online). Analytically calculating the gradient of the tree likelihood considerably improved the running time of `phylojax` pointing at implementation issues in the gradient function in JAX for this type of algorithm (supplementary fig. S8, Supplementary Material

online). In contrast, enabling JIT in `torchtree` showed no improvement and was not included in the results. The calculation of the tree likelihood and its gradient were significantly slower using single precision for datasets larger than 500 sequences. This is because `torchtree`, like most phylogenetic programs, rescales partial likelihood vectors in order to avoid underflow; using single precision requires more rescaling operations.

Discussion

We have found that, although AD packages provide unrivaled flexibility for model development and flexible likelihood formulation, they cannot compete with carefully implemented gradients in compiled languages. Furthermore, they do differ between each other significantly in computation time and memory usage for phylogenetic tasks.

Our results motivate the design of `bito`: leverage specialized algorithms for phylogenetic gradients and ratio transforms, but wrap them in a way that invites model flexibility. In this paper, we have focused on two functionalities of `bito`: first as a wrapper for the high-performance `BEAGLE` library, and second, as a fast means of computing the ratio transforms. This is our first publication using this library, which will be the computational core of our future work on Bayesian phylogenetic inference via optimization. We will defer a more comprehensive description of `bito` to future work.

Our results also motivate us to focus our future model developments using the PyTorch library, which shows the best performance as well as ease of use.

Our study has the following limitations. First, these libraries are developing quickly and they may gain substantially in efficiency in future versions. Second, these results concern CPU computation only. Future work, including development of phylogenetic gradients using graphics processing units (GPUs), will evaluate the promise of GPUs for gradient-based inference. However, we note that initial results using GPUs for AD packages did not lead to a significant speedup.

Methods

Data

To evaluate the performance of each implementation, we reused parts of the validation workflow introduced by Sagulenko et al. (2018). The data in this workflow consist of a collection of influenza A datasets ranging from 20 to 2000 sequences sampled from 2011 to 2013. Our benchmark is built on top of this pipeline and makes use of a reproducible Nextflow (Di Tommaso et al. 2017) pipeline.

Software Benchmarked

`torchtree` is a Python-based tool that leverages the Pytorch library to calculate gradients using reverse mode AD.

`torchtree-bito` is a `torchtree` plugin that offers an interface to the `bito` library (<https://github.com/phylovi/bito>). Within `bito`, analytical derivatives with respect to the branch lengths are calculated through the `BEAGLE` library (Ayres et al. 2019; Ji et al. 2020) while the gradient with respect to the GTR substitution model parameters are calculated numerically using finite differences. `bito` and `BEAGLE` do not provide analytical derivatives of the coalescent function, hence no results are shown in figure 2 and supplementary figures S4–S7, Supplementary Material online.

`physher` is a C program that allows one to approximate distributions using ADVI (Fourment et al. 2020), while every derivative is calculated analytically. The derivatives with respect to the branch lengths are efficiently calculated using a linear-time algorithm developed independently of Ji et al. (2020). The gradient of the Jacobian transform is efficiently calculated using the method proposed by Ji et al. (2021).

`phylostan` is a Python-based program (Fourment and Darling 2019) that generates phylogenetic models that are compatible with the Stan package.

`phylojax` is a Python-based tool that leverages the JAX library to calculate gradients using reverse mode AD.

`treeflow` is a Python-based tool that leverages the TensorFlow library to calculate gradients using reverse mode AD. `treeflow`'s implementation of the phylogenetic likelihood uses TensorFlow's `TensorArray` construct (Yu et al. 2018), a data structure which represents a collection of arrays. Each array can only be written once in a computation, and read many times. Using these data structure to implement the dynamic programming steps of the pruning algorithm potentially allows for more scalable gradient computations.

Computational Infrastructure

The automated workflow was run using the Fred Hutchinson `gizmo` scientific computing infrastructure. A single node with 36 (2 sockets by 18 cores) Intel® Xeon Gold 6254 CPU @ 3.10GHz cores was used for all individual processes in the pipeline. A total of 48G RAM was allocated. The node was running on Ubuntu 18.04.5 LTS (Bionic Beaver) with Nextflow (version 22.04.3.5703) and Singularity (version 3.5.3) modules installed.

Supplementary Material

Supplementary data are available at *Genome Biology and Evolution* online.

Acknowledgments

We are grateful to Jonathan Terhorst for discussions concerning phylogenetic gradients in JAX. This work was supported through US National Institutes of Health grants AI162611 and AI153044. Scientific Computing Infrastructure at Fred Hutch was funded by ORIP grant

S100D028685. Computational facilities were provided by the UTS eResearch High-Performance Compute Facilities. Dr. Matsen is an Investigator of the Howard Hughes Medical Institute.

Data Availability

The Nextflow pipeline is available from <https://github.com/4ment/gradient-benchmark>. The versions of the programs used in this study are provided in table 1.

Literature Cited

- Abadi M, et al. 2016. TensorFlow: a system for large-scale machine learning. In: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, OSDI'16; 2016 Nov; USA. USENIX Association. p. 265–283. Available from: <https://dl.acm.org/doi/10.5555/3026877.3026899>.
- Ayres DL, et al. 2019. BEAGLE 3: improved performance, scaling, and usability for a high-performance computing library for statistical phylogenetics. *Syst Biol.* 68(6):1052–1061.
- Bedford T, et al. 2014. Integrating influenza antigenic dynamics with molecular evolution. *elife.* 3:e01914.
- Blei DM, Kucukelbir A, McAuliffe JD. 2017. Variational inference: a review for statisticians. *J Am Stat Assoc.* 112(518):859–877. doi:10.1080/01621459.2017.1285773
- Bradbury J, et al. 2018. JAX: composable transformations of Python +NumPy programs. Available from: <http://github.com/google/jax>.
- Carpenter B, et al. 2017. Stan: a probabilistic programming language. *J Stat Softw.* 76(1):1–32. doi:10.18637/jss.v076.i01
- Dang T, Kishino H. 2019. Stochastic variational inference for Bayesian phylogenetics: a case of CAT model. *Mol Biol Evol.* 36(4):825–833. doi:10.1093/molbev/msz020
- Di Tommaso P, et al. 2017. Nextflow enables reproducible computational workflows. *Nat Biotechnol.* 35(4):316–319. doi:10.1038/nbt.3820
- Felsenstein J. 1981. Evolutionary trees from DNA sequences: a maximum likelihood approach. *J Mol Evol.* 17(6):368–376.
- Fisher AA, Ji X, Zhang Z, Lemey P, Suchard MA. 2021. Relaxed random walks at scale. *Syst Biol.* 70(2):258–267. doi:10.1093/sysbio/syaa056
- Fourment M, Darling AE. 2019. Evaluating probabilistic programming and fast variational Bayesian inference in phylogenetics. *PeerJ.* 7: e8272. doi:10.7717/peerj.8272
- Fourment M, Holmes EC. 2014. Novel non-parametric models to estimate evolutionary rates and divergence times from heterochronous sequence data. *BMC Evol Biol.* 14:163. doi:10.1186/s12862-014-0163-6
- Fourment M, et al. 2020. 19 dubious ways to compute the marginal likelihood of a phylogenetic tree topology. *Syst Biol.* 69(2):209–220.
- Ji X, et al. 2020. Gradients do grow on trees: a linear-time $\mathcal{O}(N)$ -dimensional gradient for statistical phylogenetics. *Mol Biol Evol.* 37(10):3047–3060. doi:10.1093/molbev/msaa130
- Ji X, et al. 2021. Scalable Bayesian divergence time estimation with ratio transformations; October. Available from: <http://arxiv.org/abs/2110.13298>.
- Jukes TH, Cantor CR. 1969. Evolution of protein molecules. In: *Mammalian protein metabolism*. Vol. 3. New York: Academic Press. p. 21–132.
- Kenney T, Gu H. 2012. Hessian calculation for phylogenetic likelihood based on the pruning algorithm and its applications. *Stat Appl Genet Mol Biol.* 11(4):Article 14. doi:10.1515/1544-6115.1779

- Ki C, Terhorst J. 2022. Variational phylodynamic inference using pandemic-scale data. *Mol Biol Evol.* 39(8):msac154. doi:10.1093/molbev/msac154
- Kingman JFC. 1982. The coalescent. *Stoch Process Appl.* 13(3):235–248.
- Koptagel H, Kviman O, Melin H, Safinianaini N, Lagergren J. 2022. VaiPhy: a variational inference based algorithm for phylogeny, March. Available from: <http://arxiv.org/abs/2203.01121>.
- Kucukelbir A, Tran D, Ranganath R, Gelman A, Blei DM. 2017. Automatic differentiation variational inference. *J Mach Learn Res.* 18(1):430–474.
- Lemey P, et al. 2020. Accommodating individual travel history, global mobility, and unsampled diversity in phylogeography: a SARS-CoV-2 case study. *bioRxiv*.
- Liu X, Ogilvie HA, Nakhleh L. 2021. Variational inference using approximate likelihood under the coalescent with recombination. *Genome Res.* 31(11):2107–2119. doi:10.1101/gr.273631.120
- Margossian CC. 2019. A review of automatic differentiation and its efficient implementation. *Wiley Interdiscip Rev Data Min Knowl Discov.* 9(4):e1305.
- Moretti AK, et al. 2021. Variational combinatorial sequential Monte Carlo methods for Bayesian phylogenetic inference. In: *Uncertainty in artificial intelligence*. PMLR. p. 971–981.
- Neal R. 2011. MCMC using Hamiltonian dynamics. In: Brooks S, Gelman A, Jones G, Meng XL, editors. *Handbook of Markov chain Monte Carlo*. Chapman & Hall/CRC Handbooks of Modern Statistical Methods. Taylor & Francis. Available from: <http://books.google.com/books?id=qfRsAlKZ4rIC>.
- Paszke A, et al. 2019. PyTorch: an imperative style, high-performance deep learning library; December. Available from: <https://papers.nips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf>.
- Sagulenko P, Puller V, Neher RA. 2018. TreeTime: maximum-likelihood phylodynamic analysis. *Virus Evol.* 4(1):vex042.
- Schadt EE, Sinsheimer JS, Lange K. 1998. Computational advances in maximum likelihood methods for molecular phylogeny. *Genome Res.* 8(3):222–233. doi:10.1101/gr.8.3.222
- Suchard MA, et al. 2018. Bayesian phylogenetic and phylodynamic data integration using beast 1.10. *Virus Evol.* 4(1):vey016.
- Yu Y, et al. 2018. Dynamic control flow in large-scale machine learning. In: *Proceedings of the Thirteenth EuroSys Conference*. New York: Association for Computing Machinery (ACM). p. 1–15.
- Zhang C, Matsen FA IV. 2019. Variational Bayesian phylogenetic inference. In: *International Conference on Learning Representations (ICLR)*. New Orleans: OpenReview.net. Available from: <https://openreview.net/pdf?id=SJVmjjR9FX>.
- Zhang C, Matsen FA IV. 2022. A variational approach to Bayesian phylogenetic inference, April. Available from: <http://arxiv.org/abs/2204.07747>.

Associate editor: Tom Williams