AUTOMATIC DIFFERENTIATION IS NO PANACEA FOR PHYLOGENETIC GRADIENT COMPUTATION

MATHIEU FOURMENT,[1] CHRISTIAAN J SWANEPOEL,[2,3] JARED G GALLOWAY,[4] XIANG JI,[5] KARTHIK GANGAVARAPU,[6] MARC A SUCHARD,[*,6,7,8] FREDERICK A MATSEN IV[*,4,9,10,11]

[1] *Australian Institute for Microbiology and Infection, University of Technology Sydney, Ultimo NSW, Australia;*

[2] *Centre for Computational Evolution, The University of Auckland, Auckland, New Zealand;*

[3] *School of Computer Science, The University of Auckland, Auckland, New Zealand, 1010;*

[4] *Public Health Sciences Division, Fred Hutchinson Cancer Research Center, Seattle, Washington, USA;*

[5] *Department of Mathematics, Tulane University, New Orleans, USA;*

[6] *Department of Human Genetics, University of California, Los Angeles, USA;*

[7] *Department of Computational Medicine, University of California, Los Angeles, USA;*

[8] *Department of Biostatistics, University of California, Los Angeles, USA;*

[9] *Department of Statistics, University of Washington, Seattle, USA;*

[10] *Department of Genome Sciences, University of Washington, Seattle, USA;*

[11] *Howard Hughes Medical Institute, Fred Hutchinson Cancer Research Center, Seattle, Washington, USA;*

Corresponding authors: Marc A. Suchard, Departments of Biomathematics, Biostatistics and Human Genetics, 6558 Gonda Building, 695 Charles E. Young Drive, South Los Angeles, CA 90095-1766, USA, E-mail: msuchard@ucla.edu; Frederick A. Matsen IV, Fred Hutchinson Cancer Research Center, 1100 Fairview Ave. N, Mail stop S2-140, Seattle, WA, 98109, USA, E-mail: matsen@fredhutch.org

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.

1

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 [Neal, 2011], implemented in the Stan statistical framework [Carpenter et al., 2017], is a cornerstone of the modern Bayesian statistical toolbox. Variational Bayesian 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. Indeed the "market" for fast gradients continues to expand, with phylogenetic applications of Hamiltonian Monte Carlo Fisher et al. [2021] and Variational Bayes phylogenetic analysis [Fourment and Darling, 2019a, Zhang and Matsen, 2019, Dang and Kishino, 2019, Liu et al., 2021, Moretti et al., 2021, Ki and Terhorst, 2022, Koptagel et al., 2022, Zhang and Matsen, 2022] gaining traction. These algorithms require efficient gradient calculations. 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 et al., 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 et al., 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, 2019a] 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 et al., 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 8 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, 2019a]. 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 (Figure 1). This is remarkable given that these are highly optimized libraries doing the same flavor of operations. Specifically, both just-in-time compiled JAX and compiled TensorFlow use XLA as a backend, although they have strikingly different performance.[1] 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 8 times the speed of the fastest AD implementation (Figure S2).

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 (Figure S3). 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 an ADVI program while incurring a small performance and memory cost compare 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 (Figure 2 and S4). 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 (Figure 2 and S3) in phylogenetic models

---

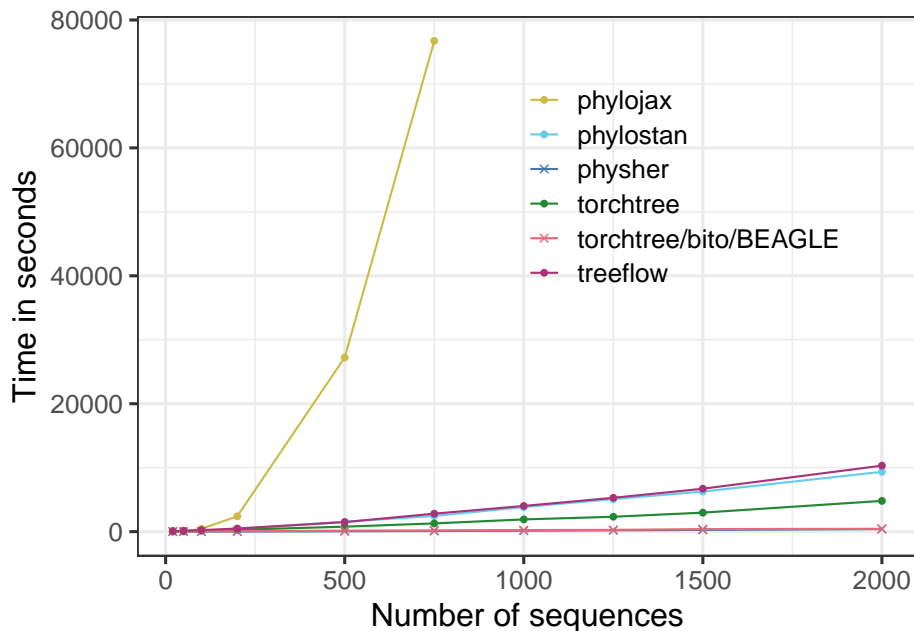[1]We note that this is now a known issue with JAX https://github.com/google/jax/issues/10197.

FIGURE 1. Speed of implementations for 5000 iterations of variational time-tree inference with a strict clock. See Figure S1 for results without `phylojax`.

and efficient gradient calculation are warranted. TensorFlow-based `treeflow` was the slowest implementation across the board after excluding JAX.

The AD programs also performed significantly worse in the node height transform and tree likelihood tasks. This is likely due to the tree traversal not being a vectorized computation in these AD libraries.

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 (Figure S7), 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 (Figure S8). 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 (Figure S8). In contrast, enabling JIT in `torchtree` showed no improvement and was not included in the results.
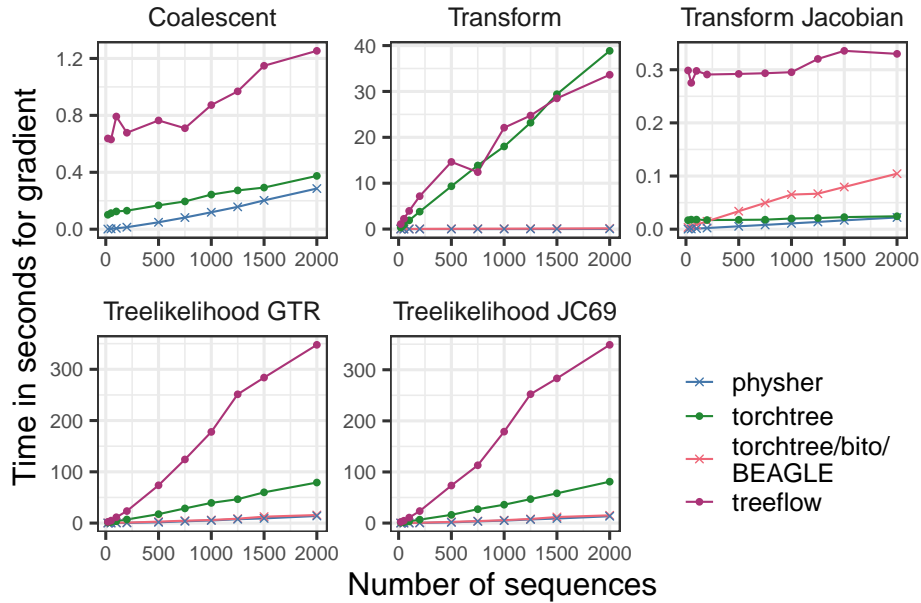
FIGURE 2. Speed of implementations for the gradient of various tasks needed for inference. See text for description of the tasks. JAX is excluded from this plot due to slow performance stretching the y-axis; see Figure S5 for JAX. See Figure S6 for function evaluations.

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 consists 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 available from `https://github.com/4ment/autodiff-experiments`.

Software benchmarked. `torchtree` is a Python-based tool that leverages the Pytorch library to calculate gradients using reverse mode AD. It is available from `https://github.com/4ment/torchtree`.

`torchtree-bito` is a `torchtree` plugin (`https://github.com/4ment/torchtree-bito`) 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.

`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]. It is available from `https://github.com/4ment/physher`.

`phylostan` is a Python-based program [Fourment and Darling, 2019b] that generates phylogenetic models that are compatible with the Stan package and it is available from `https://github.com/4ment/phylostan`.

`phylojax` is a Python-based tool that leverages the JAX library to calculate gradients using reverse mode AD. It is available from `https://github.com/4ment/phylojax`.

`treeflow` is a Python-based tool that leverages the TensorFlow library to calculate gradients using reverse mode AD. It is available from `https://github.com/christiaanjs/treeflow`. `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 this 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.

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.

Acknowledgements

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: a system for large-scale machine learning. In *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*, OSDI'16, pages 265–283, USA, November 2016. USENIX Association. ISBN 9781931971331. URL `https://dl.acm.org/doi/10.5555/3026877.3026899`.

Daniel L Ayres, Michael P Cummings, Guy Baele, Aaron E Darling, Paul O Lewis, David L Swofford, John P Huelsenbeck, Philippe Lemey, Andrew Rambaut, and Marc A Suchard. BEAGLE 3: Improved performance, scaling, and usability for a high-performance computing library for statistical phylogenetics. *Syst. Biol.*, 68(6):1052–1061, November 2019.

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *J. Am. Stat. Assoc.*, 112(518):859–877, April 2017. ISSN 0162-1459. doi: 10.1080/01621459.2017.1285773. URL `https://doi.org/10.1080/01621459.2017.1285773`.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs. `http://github.com/google/jax`, 2018.

Bob Carpenter, Andrew Gelman, Matthew Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of Statistical Software, Articles*, 76(1):1–32, 2017. ISSN 1548-7660. doi: 10.18637/jss.v076.i01. URL `https://www.jstatsoft.org/v076/i01`.

Tung Dang and Hirohisa Kishino. Stochastic variational inference for Bayesian phylogenetics: A case of CAT model. *Mol. Biol. Evol.*, 36(4):825–833, April 2019. ISSN 0737-4038, 1537-1719. doi: 10.1093/molbev/msz020. URL `http://dx.doi.org/10.1093/molbev/msz020`.

Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. *Nat. Biotechnol.*, 35(4):316–319, April 2017. ISSN 1087-0156, 1546-1696. doi: 10.1038/nbt.3820. URL `http://dx.doi.org/10.1038/nbt.3820`.

J Felsenstein. Evolutionary trees from DNA sequences: a maximum likelihood approach. *J. Mol. Evol.*, 17(6):368–376, 1981. ISSN 0022-2844. URL `http://www.ncbi.nlm.nih.gov/pubmed/7288891`.

Alexander A Fisher, Xiang Ji, Zhenyu Zhang, Philippe Lemey, and Marc A Suchard. Relaxed random walks at scale. *Syst. Biol.*, 70(2):258–267, February 2021. ISSN 1063-5157, 1076-836X. doi: 10.1093/sysbio/syaa056. URL `http://dx.doi.org/10.1093/sysbio/syaa056`.

Mathieu Fourment and Aaron E Darling. Evaluating probabilistic programming and fast variational Bayesian inference in phylogenetics. *PeerJ*, 7:e8272, December 2019a. ISSN 2167-8359. doi: 10.7717/peerj.8272. URL `http://dx.doi.org/10.7717/peerj.8272`.

Mathieu Fourment and Aaron E Darling. Evaluating probabilistic programming and fast variational Bayesian inference in phylogenetics. *PeerJ*, 7:e8272, December 2019b.

Mathieu Fourment and Edward C Holmes. Novel non-parametric models to estimate evolutionary rates and divergence times from heterochronous sequence data. *BMC Evol. Biol.*, 14:163, July 2014. ISSN 1471-2148. doi: 10.1186/s12862-014-0163-6. URL `http://dx.doi.org/10.1186/s12862-014-0163-6`.

Mathieu Fourment, Andrew F Magee, Chris Whidden, Arman Bilge, Frederick A Matsen, and Vladimir N Minin. 19 dubious ways to compute the marginal likelihood of a phylogenetic tree topology. *Syst. Biol.*, 69(2):209–220, March 2020.

Xiang Ji, Zhenyu Zhang, Andrew Holbrook, Akihiko Nishimura, Guy Baele, Andrew Rambaut, Philippe Lemey, and Marc A Suchard. Gradients do grow on trees: a linear-time $\mathcal{O}(N)$-dimensional gradient for statistical phylogenetics. *Mol. Biol. Evol.*, May 2020. ISSN 0737-4038, 1537-1719. doi: 10.1093/molbev/msaa130. URL `http://dx.doi.org/10.1093/molbev/msaa130`.

Xiang Ji, Alexander A Fisher, Shuo Su, Jeffrey L Thorne, Barney Potter, Philippe Lemey, Guy Baele, and Marc A Suchard. Scalable Bayesian divergence time estimation with ratio transformations, October 2021. URL `http://arxiv.org/abs/2110.13298`.

Thomas H Jukes, Charles R Cantor, and Others. Evolution of protein molecules. *Mammalian protein metabolism*, 3:21–132, 1969.

Toby Kenney and Hong Gu. Hessian calculation for phylogenetic likelihood based on the pruning algorithm and its applications. *Stat. Appl. Genet. Mol. Biol.*, 11 (4):Article 14, September 2012. ISSN 1544-6115. doi: 10.1515/1544-6115.1779. URL `http://dx.doi.org/10.1515/1544-6115.1779`.

Caleb Ki and Jonathan Terhorst. Variational Phylodynamic Inference Using Pandemic-scale Data. *Molecular Biology and Evolution*, 39(8), 07 2022. ISSN 1537-1719. doi: 10.1093/molbev/msac154. URL `https://doi.org/10.1093/molbev/msac154`. msac154.

J F C Kingman. The coalescent. *Stochastic Process. Appl.*, 13(3):235–248, September 1982.

Hazal Koptagel, Oskar Kviman, Harald Melin, Negar Safinianaini, and Jens Lagergren. VaiPhy: a variational inference based algorithm for phylogeny, March 2022. URL `http://arxiv.org/abs/2203.01121`.

Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic differentiation variational inference. *J. Mach. Learn. Res.*, 18 (1):430–474, January 2017.

Xinhao Liu, Huw A Ogilvie, and Luay Nakhleh. Variational inference using approximate likelihood under the coalescent with recombination. *Genome Res.*, 31(11):2107–2119, November 2021. ISSN 1088-9051, 1549-5469. doi: 10.1101/gr.273631.120. URL `http://dx.doi.org/10.1101/gr.273631.120`.

Charles C Margossian. A review of automatic differentiation and its efficient implementation. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, 9(4):e1305, July 2019.

Antonio Khalil Moretti, Liyi Zhang, Christian A Naesseth, Hadiah Venner, David
    Blei, and Itsik Pe'er. Variational combinatorial sequential Monte Carlo methods
    for Bayesian phylogenetic inference. In *Uncertainty in Artificial Intelligence*,
    pages 971–981. PMLR, 2021.

Radford Neal. MCMC using Hamiltonian dynamics. In S Brooks, A Gelman,
    G Jones, and XL Meng, editors, *Handbook of Markov Chain Monte Carlo*, Chap-
    man & Hall/CRC Handbooks of Modern Statistical Methods. Taylor & Fran-
    cis, 2011. ISBN 9781420079425. URL `http://books.google.com/books?id=`
    `qfRsAIKZ4rIC`.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gre-
    gory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga,
    Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison,
    Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and
    Soumith Chintala. PyTorch: An imperative style, high-performance deep learn-
    ing library, December 2019. URL `https://papers.nips.cc/paper/2019/file/`
    `bdbca288fee7f92f2bfa9f7012727740-Paper.pdf`.

Pavel Sagulenko, Vadim Puller, and Richard A Neher. TreeTime: Maximum-
    likelihood phylodynamic analysis. *Virus Evol*, 4(1):vex042, January 2018.

E E Schadt, J S Sinsheimer, and K Lange. Computational advances in maximum
    likelihood methods for molecular phylogeny. *Genome Res.*, 8(3):222–233, March
    1998. ISSN 1088-9051. URL `http://www.ncbi.nlm.nih.gov/pubmed/9521926`.

Yuan Yu, Martín Abadi, Paul Barham, Eugene Brevdo, Mike Burrows, Andy Davis,
    Jeff Dean, Sanjay Ghemawat, Tim Harley, Peter Hawkins, et al. Dynamic control
    flow in large-scale machine learning. In *Proceedings of the Thirteenth EuroSys
    Conference*, pages 1–15, 2018.

Cheng Zhang and Frederick A Matsen, IV. Variational Bayesian phylogenetic in-
    ference. In *International Conference on Learning Representations (ICLR)*, 2019.
    URL `https://openreview.net/pdf?id=SJVmjjR9FX`.

Cheng Zhang and Frederick A Matsen, IV. A variational approach to Bayesian
    phylogenetic inference, April 2022. URL `http://arxiv.org/abs/2204.07747`.
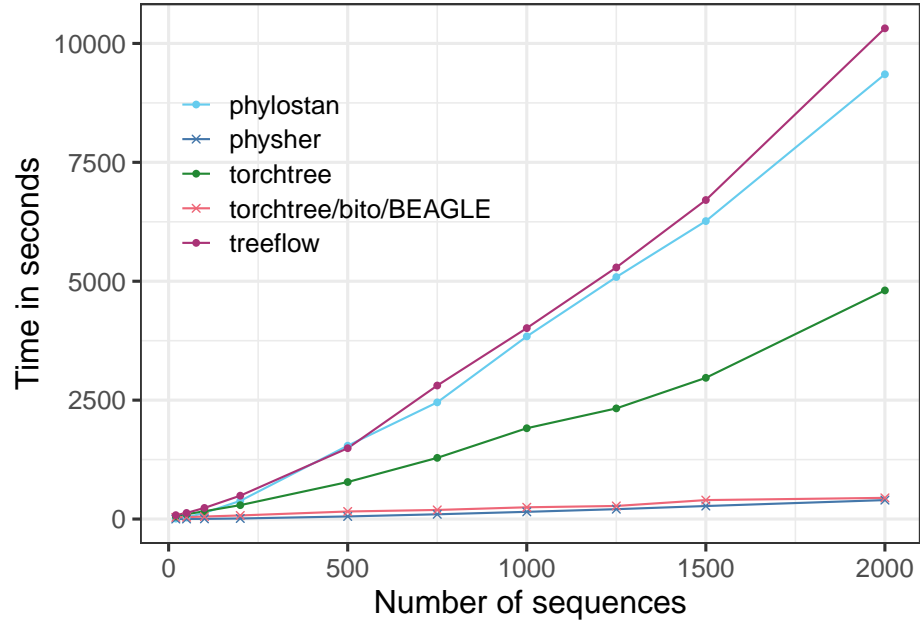
Supplementary Materials

Figure S1. Speed of implementations for 5000 iterations of variational time-tree inference with a strict clock.
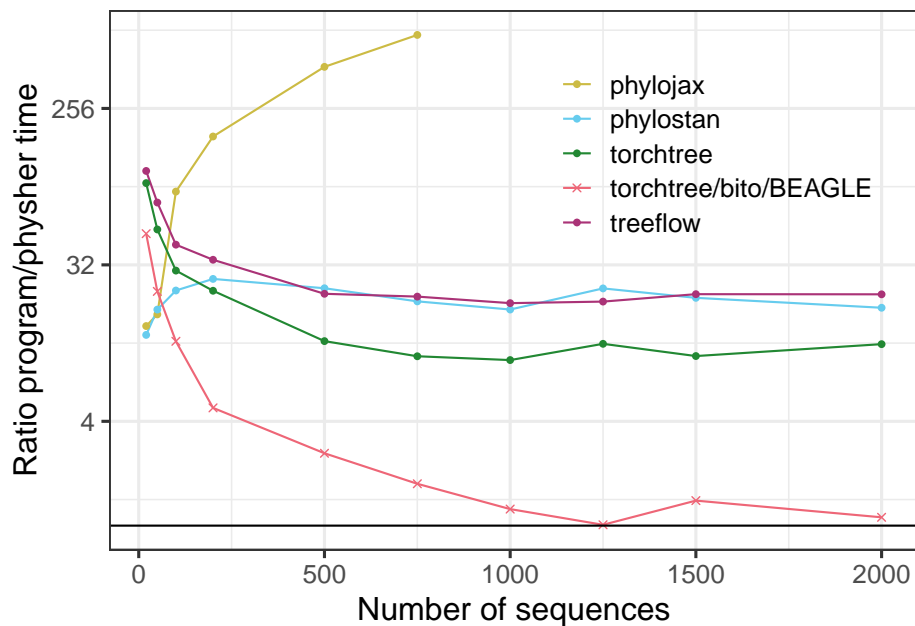
FIGURE S2. Relative performance of each implementation against `physher`. The black horizontal line intersects the y-axis at 1.
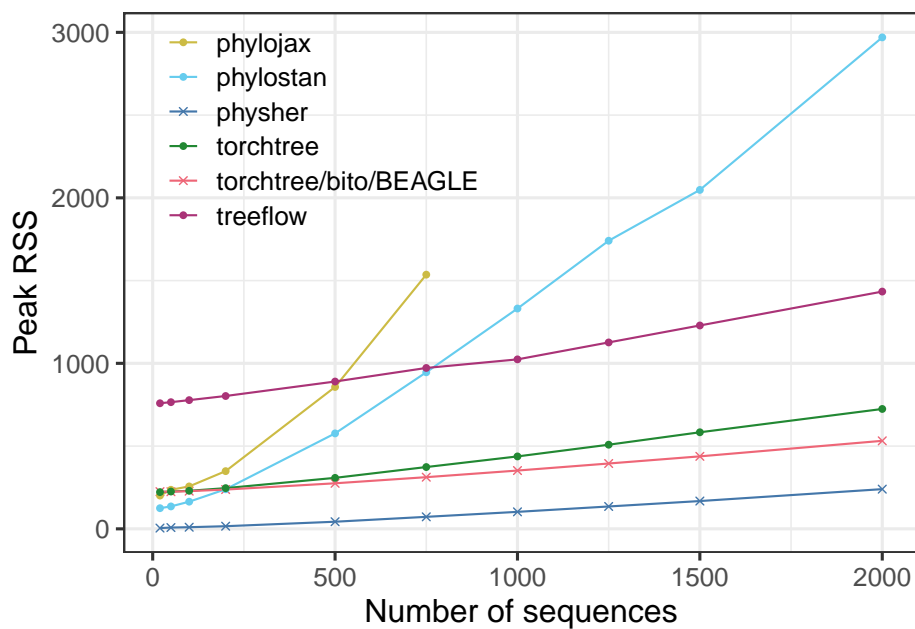

FIGURE S3. Peak RSS (resident set size) memory usage of implementations for 5000 iterations of variational time-tree inference with a strict clock.
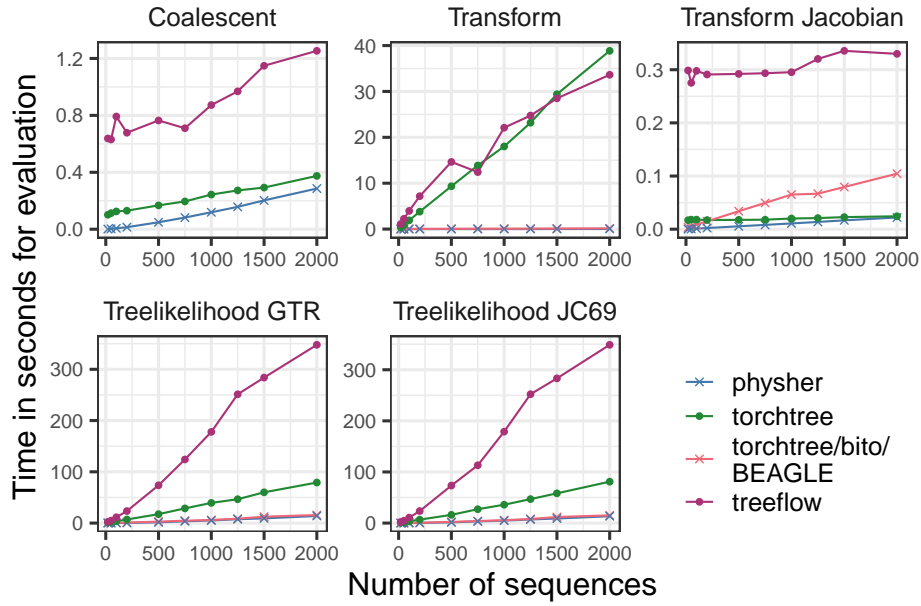
FIGURE S4. Speed of implementations for the evaluation component various inferential tasks. See text for description of the tasks. `phylojax` results are excluded from this plot; see Figure S6 for `phylojax`. See Figure S6 for function evaluations.
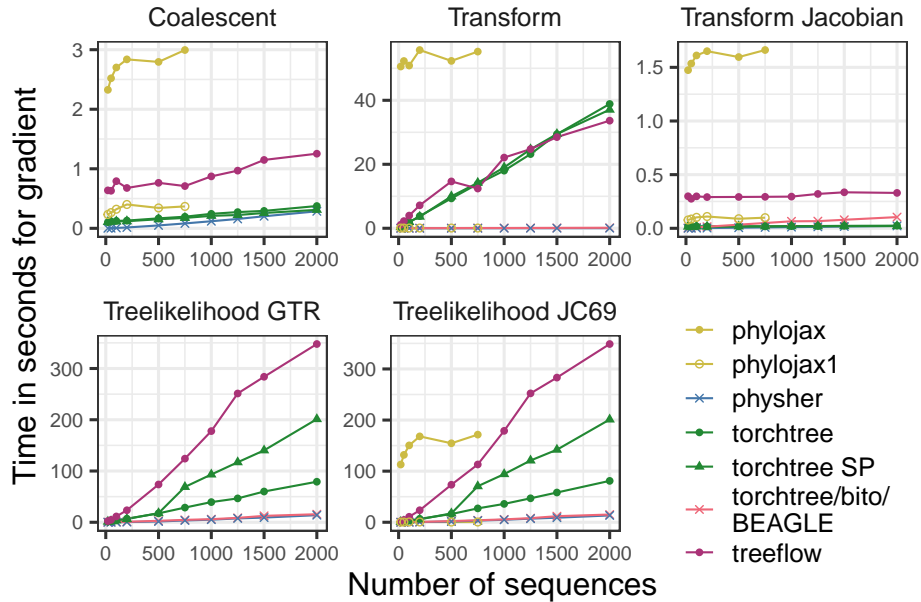
FIGURE S5. Speed of implementations for the gradient component for various inferential tasks. `phylojax1` shows the evaluation time of the first call of the gradient function. The `torchtree SP` label denotes `torchtree` running with single precision.
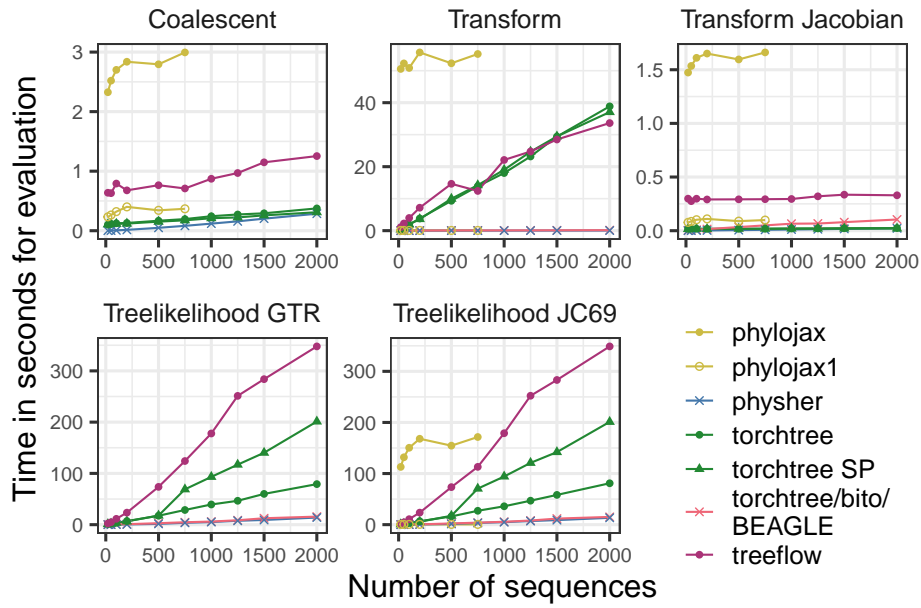
FIGURE S6. Speed of implementations for the evaluation component for various inferential tasks. phylojax1 represents the evaluation time of the first call of the function. The torchtree SP label denotes torchtree running with single precision.
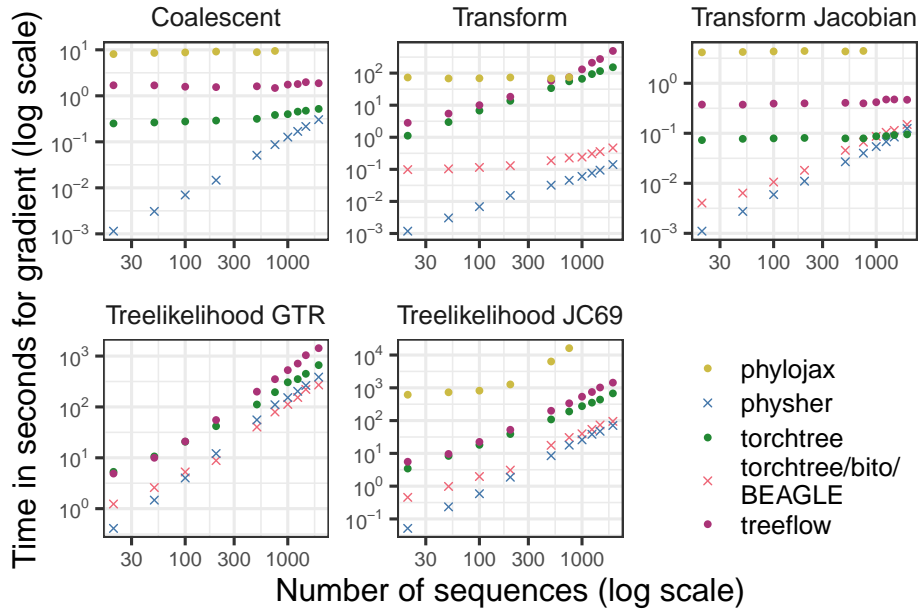

FIGURE S7. Log-log plot of gradient calculation time against dataset size for various inferential tasks. phylojax1 represents the evaluation time of the first call of the gradient function.
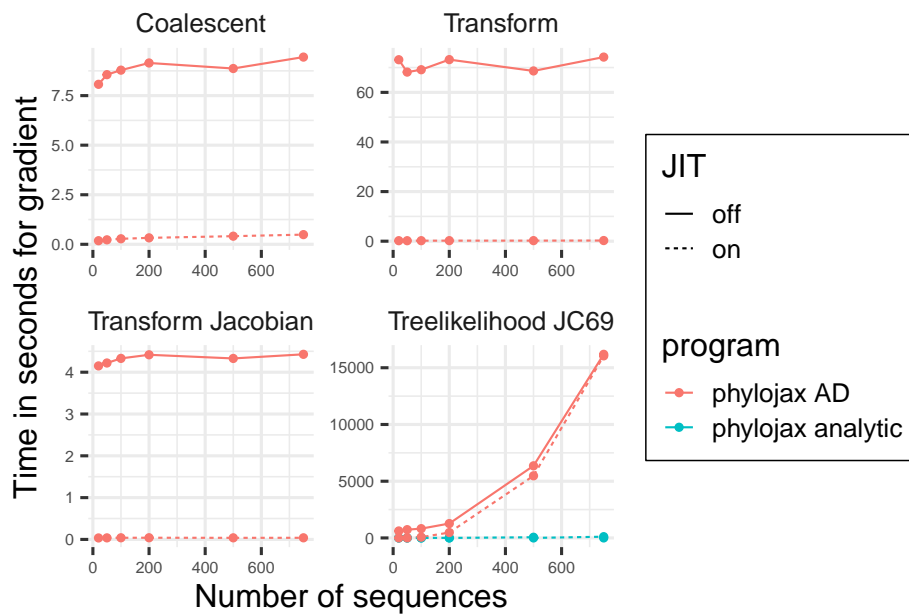
FIGURE S8. Gradient calculation time against dataset size for various inferential tasks with `phylojax`. Just-in-time (JIT) compilation is either turned on or off.