




Rose: Composable Autodiff for the Interactive Web (Artifact)

Sam Estep   

Software and Societal Systems Department, Carnegie Mellon University, Pittsburgh, PA, USA

Wode Ni   

Software and Societal Systems Department, Carnegie Mellon University, Pittsburgh, PA, USA

Raven Rothkopf   

Barnard College, Columbia University, New York, NY, USA

Joshua Sunshine   

Software and Societal Systems Department, Carnegie Mellon University, Pittsburgh, PA, USA

— Abstract —

Reverse-mode automatic differentiation (autodiff) has been popularized by deep learning, but its ability to compute gradients is also valuable for interactive use cases such as bidirectional computer-aided design, embedded physics simulations, visualizing causal inference, and more. Unfortunately, the web is ill-served by existing autodiff frameworks, which use autodiff strategies that perform poorly on dynamic scalar programs, and pull in heavy dependencies that would result in unacceptable webpage sizes. This document accompanies the research paper introducing Rose, a lightweight autodiff frame-

work for the web using a new hybrid approach to reverse-mode autodiff, blending conventional tracing and transformation techniques in a way that uses the host language for metaprogramming while also allowing the programmer to explicitly define reusable functions that comprise a larger differentiable computation. We demonstrate the value of the Rose design by porting two differentiable physics simulations, and evaluate its performance on an optimization-based diagramming application, showing Rose outperforming the state-of-the-art in web-based autodiff by multiple orders of magnitude.

2012 ACM Subject Classification Software and its engineering → Compilers; Information systems → Web applications; Software and its engineering → Domain specific languages; Computing methodologies → Symbolic and algebraic manipulation; Software and its engineering → Formal language definitions; General and reference → Performance; Computing methodologies → Neural networks

Keywords and phrases Automatic differentiation, differentiable programming, compilers, web

Digital Object Identifier 10.4230/DARTS.10.2.7

Funding This material is based upon work supported by the Aqueduct Foundation and by National Science Foundation under Grant Numbers 1910264, 2119007, and 2150217.

Related Article Sam Estep, Wode Ni, Raven Rothkopf, and Joshua Sunshine, “Rose: Composable Autodiff for the Interactive Web”, in 38th European Conference on Object-Oriented Programming (ECOOP 2024), LIPIcs, Vol. 313, pp. 15:1–15:27, 2024.

<https://doi.org/10.4230/LIPIcs.ECOOP.2024.15>

Related Conference 38th European Conference on Object-Oriented Programming (ECOOP 2024), September 16–20, 2024, Vienna, Austria

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2024 Call for Artifacts and the ACM Artifact Review and Badging Policy.

1 Scope

This section describes the claims in the paper supported by this artifact, along with the steps one must follow in order to reproduce each such claim. To start, follow the instructions in Section 3 to download the Docker image archive as `image.tar`, then load it into Docker:



© Sam Estep, Wode Ni, Raven Rothkopf, and Joshua Sunshine; licensed under Creative Commons License CC-BY 4.0

Dagstuhl Artifacts Series, Vol. 10, Issue 2, Artifact No. 7, pp. 7:1–7:4



DAGSTUHL
ARTIFACTS SERIES

Dagstuhl Artifacts Series

Schloss Dagstuhl – Leibniz-Zentrum für Informatik,
Dagstuhl Publishing, Germany



7:2 Rose: Composable Autodiff for the Interactive Web (Artifact)

```
docker load --input image.tar
```

Then run that Docker image:

```
docker run -it ghcr.io/rose-lang/ecoop24:latest
```

The initial working directory should be `/root/ecoop24`; unless noted otherwise, all commands in this section should be run within the Docker container from that directory.

- The output from Listing 1 can be obtained by running `node listing1.js`.
- The output from Listing 2 can be obtained by running `node listing2.js`.
- The bundle sizes reported in Section 5.2 can be obtained by running these commands:

```
npm run tfjs-size
npm run rose-size
```

We note that this produces some minor discrepancies in the gzip sizes compared to the numbers reported in the paper, which we suspect are just due to running different versions of gzip. Specifically, we found these commands to report the gzip size of TensorFlow.js to be 86.21 kB, and the gzip size of the Rose Wasm binary and JS wrapper to be 63.78 kB and 8.78 kB respectively.

- Figure 5 can be generated by running `./plots.py`. To retrieve the generated PDF, stop the running Docker image and run the following command from outside of it (alternatively, you can substitute the actual container ID instead of writing `$(docker ps -lq)`):

```
docker cp $(docker ps -lq):/root/ecoop24/plots.pdf .
```

The same `plots.py` script also reproduces the quartiles and 10 omitted diagrams reported in Section 5.3. The script itself does not generate the data for the plots, as that is already present in `tfjs.json` and `rose.json`; to generate those, run `./penrose-tfjs.sh` and `./penrose-rose.sh` respectively. The latter should complete in a couple minutes, while the former may take multiple days. To instead run just the 88 quickest examples listed in `quickest.json`, run `./quicken.sh` which will copy that to replace the contents of `/root/penrose-tfjs/packages/core/src/registry.json`; then run `./penrose-tfjs.sh`, which should now only take about ten minutes.

- The paper reports that we found `@tensorflow/tfjs-node` and `@tensorflow/tfjs-node-gpu` to be slower than `@tensorflow/tfjs` for Penrose. We were unable to package either of those in our Docker image, due to specific constraints on native dependencies. The code which we used to run them outside of Docker was nearly identical to the code we already provide here for TensorFlow.js; we obtained it by applying `tfjs-node.diff` and `tfjs-node-gpu.diff` (respectively) to `/root/penrose-tfjs`.
- To reproduce the performance statistics in Section 5.3 comparing the WebAssembly backend to an alternative JavaScript backend, run `./js.py`. Similar to the `plots.py` script above, this does not actually run the benchmarks, and instead reads the data from `rose.json` and `js.json`. To run the benchmarks and generate new timing data for these, run `./penrose-rose.sh` or `./penrose-js.sh` respectively.
- The DiffTaichi timings can be obtained by running the scripts `./taichi-py-billiards.sh` and `./taichi-py-mass-spring.sh`, and the Rose port timings can be obtained by running `./taichi-rose-billiards.sh` and `./taichi-rose-mass-spring.sh`. All these scripts print timing data to the console.
- The Wasm compilation failure noted in Section 5.4.1 can be reproduced by running the script `./taichi-error.sh`, which applies `less-fn.diff` to `/root/taichi-rose` to get rid of one key usage of `fn` in each of the two examples, then tries to run both, demonstrating the V8 engine limits on both function size and local count.

2 Content

The Docker image is built from the `python:3.10` image¹; everything on top of that lies underneath the `/root` directory. These are all the changes made on top of the base image:

- The following Python packages have been installed:
 - `imageio`
 - `matplotlib`
 - `numpy`
 - `opencv-python`
 - `scipy`
 - `seaborn`
 - `taichi>=1.1.0`
 - `torch`
 - `torchvision`
- Node.js v20.12.2 and Yarn v1.22.22 have been installed.
- Two versions of the Penrose codebase are present:
 - `/root/penrose-tfjs` holds a version of the Penrose codebase using TensorFlow.js, with `node_modules` already populated.
 - `/root/penrose-rose` holds a version of the Penrose codebase using Rose, with `node_modules` already populated.
 - In each case, the Penrose repository structure relevant to this artifact is as follows:
 - * `packages/core` holds the core Penrose source code.
 - `packages/core/src/lib` holds all the differentiable functions Penrose uses for diagram layout and optimization.
 - `packages/core/src/engine/Autodiff.ts` holds most of the interface between the Penrose compiler and the underlying autodiff backend.
 - * `packages/solids` holds examples using the Penrose API rather than the three Penrose domain-specific languages.
 - * `packages/examples` holds the remaining Penrose examples, along with a script to run all these examples and collect timing data.
 - `packages/core/src/registry.json` lists all the diagrams in the benchmark suite; in particular, when running the TensorFlow.js part of the evaluation (see below), one could edit out large parts of this file to produce a subset that completes in a reasonable amount of time.
 - `packages/examples/diagrams/data.json` is generated by running `yarn registry` from the Penrose repository root, and includes all the timing data produced by the benchmark suite.
- `/root/taichi-py` holds the original DiffTaichi codebase.
 - `experiment.md` includes instructions on how to run the two examples that we ported; the dependency installation step has already been completed.
 - `packages/billiards.py` holds the original source code for the billiards example.
 - `packages/mass_spring.py` holds the original source code for the robot example.
- `/root/taichi-rose` holds the DiffTaichi examples ported to Rose.
 - `billiards` holds the ported billiards example.
 - `mass-spring` holds the ported robot example.
- `/root/ecoop24` holds the remaining files needed for this artifact, described in Section 1.

¹ <https://hub.docker.com/layers/library/python/3.10/images/sha256-cf5cac6010f4caa5446516c18f48369215df2e912a12ff314ceb9a1d95a5fd60>

7:4 Rose: Composable Autodiff for the Interactive Web (Artifact)

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). In addition, the artifact is also available at: <https://zenodo.org/records/11055122>.

4 Tested platforms

The experiments referenced in the paper itself were all run natively on a 2020 MacBook Pro with M1 chip and 16 GiB RAM. This artifact, however, is packaged to run inside Docker using Linux, and was checked in WSL2 on an AMD Ryzen 5 3600X with 24 GiB RAM.

Any fairly modern desktop or laptop computer hardware should suffice. The whole evaluation is fairly quick to complete, except for the TensorFlow.js part, which takes several days to run. Because of this, we have separated out the TensorFlow.js part into a separate script, and have provided a copy of the data we originally collected when we ran the experiment natively, so one can still run our provided scripts to generate our performance plots.

We also attempted to run the Docker image on the same MacBook that was originally used to run the experiments natively, and while it does run, the Penrose part of the evaluation for some reason seems to exit before running to completion. We are not sure whether this is an issue with the CPU architecture being ARM versus x86, or a bug in the Vitest library we use, or something else.

5 License

The artifact is available under the Creative Commons Attribution 4.0 International license.

6 MD5 sum of the artifact

609d5d4bc6f1182191b303b77dca2d7b

7 Size of the artifact

16.7 GB