Impulse: Momentously Fast, General, and Portable Probabilistic Programming via Compiler Augmentation

Siyuan Brant Qian
University of Illinois Urbana-Champaign · siyuanq4@illinois.edu

Funded by the U.S. National Science Foundation and the Simons Foundation

Code you want to *write* ≠ Code you want to *run*

Natural Code

The model **as written** — scientific math transcribed directly into code.

```
function
sky_counts(x, counts, ρ)
    α ~ HalfNormal(2)
    z ~ Normal(0, 1, length(x))

    K = α^2 * covariance(x, ρ)
    L = cholesky(K)

    intensity = exp.(L * z)
    counts ~ Poisson.(intensity)
end
```

Expensive op (Cholesky) reruns **once per iteration.**

Fast Inference Program

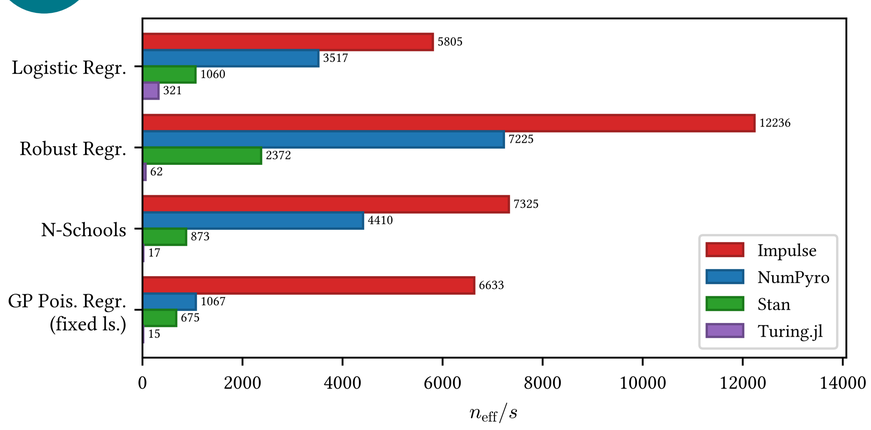
Efficient code one wants to run. Today this requires a **manual rewrite.**

```
R  = covariance(x, ρ)
LR = cholesky(R)

function
sky_counts_fast(x, counts, ρ, LR)
    α ~ HalfNormal(2)
    z ~ Normal(0, 1, length(x))

    intensity = exp.(α * (LR * z))
    counts ~ Poisson.(intensity)
end
```

After rewrite, the expensive op runs **only once outside the loop.**

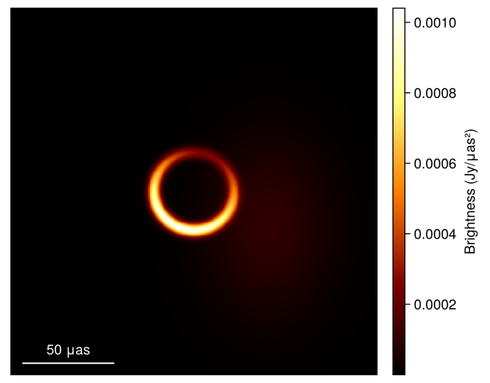
Impulse keeps probabilistic programming constructs as a *first-class compiler primitive*, so the compiler removes the redundant work *automatically.*

Benchmarks



Competitive performance: **1.7×** faster than NumPyro, **6.4×** than Stan, **53×** than Turing.jl.

Comrade Imager



Credit: ptiede.github.io/Comrade.jl — GeometricModeling tutorial

Real case study. SICM folds a non-uniform FFT (NUFFT) buried deeply within the model library into one precomputed operator.

The run. 4,428-parameter NUTS posterior on GPU; a NUFFT maps a 64×64 image to visibilities on a 128×128 oversampled grid — now lifted out of the inference loop.

End-to-end 3.7× speedup – the FFT operator is precomputed once instead of re-evaluated at every NUTS step.

Why Existing Frameworks Can't Fix It

- Sampled parameter entangle with deterministic computation so the entire term **looks loop-variant**; standard **loop-invariant code motion** cannot hoist it.
- Existing PPLs resolve probabilistic semantics **before the compiler runs**: NumPyro/Pyro/PyMC effect handlers fire at the Python level; Gen/Stan erase sample sites before any compiler IR exists.
- What reaches the compiler is numerical code with **no notion of sample-dependence;** the structure needed to optimize this redundancy is already gone.

Existing – Early Lowering

Resolve probabilistic semantics → General-Purpose Optimization → CPU/GPU

⚠ *Probabilistic structure lost before the compiler sees it.*

Impulse — Deferred Lowering

Probabilistic Compiler IR → Probabilistic Optimization → Inference Materialization → General-Purpose Optimization → CPU / GPU / TPU

✓ *Probabilistic structure preserved → Standard passes + Probabilistic passes fire on high-level IR*

Impulse's rewrite fired on the example

`cholesky(α²·R)` → `|α|·cholesky(R)`

α² is **sample-dependent** · R is **sample-invariant** ⇒ cholesky(R) **hoisted out of the loop**, computed once.

Fully extensible: new probabilistic rewrites can be added via the MLIR transform dialect.

How Impulse Works

1 Deferred lowering
Keep sampling & inference as first-class MLIR primitives (e.g., sample sites survive into the IR).

↓

2 Sample-dependence analysis
Forward dataflow analysis from sample sites marks every value sample-dependent or sample-invariant.

↓

3 Sample-Invariant Code Motion
Decompose a partly sample-invariant op, hoist the sample-invariant factor out of the inference loop.

↓

4 Materialize & lower
Materialize inference and lower to CPU/GPU/TPU via Reactant.jl.

Run on CPU/GPU/TPU with Reactant.jl

References

[1] **Siyuan Brant Qian, Vimarsh Sathia, Jesse Michel, William S. Moses. Impulse: Momentously Fast, General, and Portable Probabilistic Programming via Compiler Augmentation. Under review. 2026.**
[2] Paul Tiede. Comrade: Composable Modeling of Radio Emission. JOSS 7(76):4457, 2022.
[3] William S. Moses et al. Reactant.jl: A Julia Front End for MLIR and XLA. 2024.

Acknowledgements

This is work done in collaboration with Vimarsh Sathia (UIUC), Jesse Michel (MIT), and advisor William S. Moses (UIUC & Google). Built on Enzyme, Reactant.jl, and the MLIR / XLA ecosystem. We gratefully acknowledge the support of the NSF-Simons AI-Institute for the Sky (SkAI) via grants NSF AST-2421845 and Simons Foundation MPS-AI-00010513

Tutorial & Documentation

enzymead.github.io/Reactant.jl

Step-by-step Impulse tutorial with runnable code.

Reactant.jl on GitHub

github.com/EnzymeAD/Reactant.jl

Community-driven and actively developed. New features shipping regularly, contributors welcome.