The Journal of Open Source Software

aimz: Scalable probabilistic impact modeling

Eunseop Kim [1]

1 Eli Lilly and Company, United States

DOI: 10.21105/joss.09738

Software

- Review
- Repository
- Archive

Editor: Richard Liu

Reviewers:

- @DanWaxman
- @ankurankan

Submitted: 05 October 2025
Published: 23 April 2026

Summary

`aimz` is a Python library that provides scalable infrastructure for Bayesian modeling and probabilistic simulation. It offers an intuitive, estimator-like interface for fitting Bayesian models, drawing posterior samples, generating large-scale posterior predictive simulations, and comparing outcomes across intervention scenarios with minimal boilerplate. Researchers and analysts can explore "what-if" scenarios to understand potential effects under different decisions, generating predictions that reflect the full range of possible outcomes. `aimz` supports advanced inference methods with efficient, reproducible workflows, making it easy to run large-scale simulations and serialize structured outputs. These design choices reduce bespoke glue code, enable high-throughput analyses, and support rapid iteration and experimentation.

Statement of need

Standard Bayesian workflows often encounter significant engineering friction when transitioning from model specification to large-scale predictive simulation and impact evaluation. While core probabilistic programming frameworks offer mature inference algorithms, they lack native infrastructure for the repetitive engineering tasks essential to production-grade analysis: streaming predictive draws without exceeding system memory, managing device-aware sharding for simulation, and coordinating complex interventional scenarios. General machine learning libraries provide scalable prediction pipelines but lack the calibrated uncertainty quantification provided by Bayesian inference. `aimz` addresses these gaps by consolidating model tracing, sharded predictive sampling, and experiment lineage into a single estimator-like object. This reduces the bespoke engineering effort required to connect flexible statistical research with high-throughput data pipelines, supporting reliable and reproducible probabilistic analyses at scale.

State of the field

The landscape of Bayesian software is largely divided between low-level flexibility and high-level rigidity. Core probabilistic programming languages like NumPyro (Phan et al., 2019), PyMC (Oriol et al., 2023), and Stan (Carpenter et al., 2017) provide the flexible primitives necessary for custom research but require users to manually handle scaling, parallelization, and other performance considerations. Conversely, domain-specific frameworks like Meridian (Team, 2025), Robyn (Zhou et al., 2024), PyMC-Marketing (PyMC Labs, 2025), or Prophet (Taylor & Letham, 2018) can offer robust end-to-end pipelines but are specialized for particular domains such as marketing mix modeling or time series forecasting. These tools frequently enforce fixed model architectures (e.g., specific adstock transformations or trend decompositions) and opaque internal logic that are difficult to adapt for research problems requiring custom hierarchical specifications or structural modifications beyond what the framework provides.

`aimz` is designed to target a middle ground by providing Bayesian infrastructure in an estimator-like form. It does not prescribe a specific model form; instead, it provides a scalable execution

layer for user-defined models, focusing on high-throughput posterior predictive simulation, structured artifact management, and parallelized execution of user-defined interventions. This makes `aimz` suited for production-grade Bayesian workflows where reproducibility, experiment lineage, and scalable simulation are critical requirements.

Software design

`aimz` is designed to support scalable Bayesian analyses in applied settings, allowing users to iterate quickly across model specifications, inference settings, and intervention scenarios, as well as to handle large datasets and produce reproducible artifacts. It combines the usability of general machine learning APIs with the flexibility of probabilistic programming through a single high-level object (`ImpactModel`). An `ImpactModel` wraps a user-defined NumPyro model function together with an inference strategy, and exposes methods for fitting, prediction, and scenario comparison that manage batching, sharding, and artifact serialization internally. Built atop JAX (Bradbury et al., 2018) and NumPyro, `aimz` leverages modeling flexibility, accelerator-native execution, and composable program transformations such as vectorization and sharded execution. It supports (minibatch) stochastic variational inference (SVI) and Markov chain Monte Carlo sampling, just-in-time–compiled parallel predictive streaming to chunked Zarr (Miles et al., 2020) stores exposed through Xarray (Hoyer & Hamman, 2017), and first-class intervention handling for effect estimation. Integrated MLflow (Zaharia et al., 2018) support enables experiment tracking and model lineage.

`aimz` prioritizes performance as a key consideration. Predictive sampling and effect estimation are organized around JIT-accelerated execution, data-parallel sharding, and streaming results to disk-backed, chunked storage, enabling sustained throughput for large simulation workloads without requiring all draws to be retained in memory. This architectural choice comes with trade-offs: only models that are compatible with sharded execution can fully leverage these optimizations. Additional orchestration around execution and I/O is required, which `aimz` manages through a small set of high-level methods, enabling users to focus on analysis while maintaining a consistent interface. At the user interface level, `aimz` adopts an estimator-like API with familiar methods such as `fit` and `predict`, while still supporting a wide range of model specifications.

`aimz` is also designed for integration: results are materialized as structured artifacts (e.g., Xarray objects backed by Zarr stores) and can optionally be logged via MLflow for experiment tracking and lineage. The combination of a stable, method-based interface and standardized outputs makes `aimz` well suited for AI-enabled workflows, where agentic tools can invoke a small set of operations and reliably consume the resulting artifacts.

Research impact statement

`aimz` enables a class of Bayesian modeling and probabilistic analyses that are otherwise costly to implement and difficult to reproduce at scale: fitting flexible probabilistic models, generating large posterior predictive simulations, and comparing outcomes across intervention scenarios. Its research impact is therefore primarily infrastructural, lowering the engineering barrier to scalable, uncertainty-aware probabilistic analysis while remaining general enough to support diverse model specifications and inference strategies.

To illustrate these properties in practice, we benchmark `aimz`'s `predict` method, which performs posterior predictive sampling with streaming Zarr writes, on a single AWS EC2 g5.24xlarge instance. The benchmark model is a Bayesian neural network with 100 input features, two hidden layers of 128 units each, and a binary outcome, fitted via SVI prior to the benchmark, replicating a scenario in which the model is already trained with no cold-start overhead. For each dataset size $n \in \{10\text{K}, 100\text{K}, 1\text{M}, 10\text{M}, 100\text{M}\}$, reflecting the range from static exploratory analysis to real-time production-scale inference, `aimz` draws 1,000 posterior predictive samples

and writes them to a chunked Zarr store backed by Amazon EFS. Each configuration is repeated across five independent runs.

The table below reports the mean wall-clock time and mean peak incremental resident set size (RSS), a measure of the additional physical memory consumed by the process during the call. For comparison, NumPyro's `Predictive` class, which materializes the full $(1{,}000 \times n)$ output array in device memory, is included where it can run to completion. Both benchmarks use only the public API of each library without custom modifications. Note that the comparison is not like-for-like: `aimz` batches input data, shards computation across all available devices, draws posterior predictive samples using a JIT-compiled forward pass, and streams results to a network-attached Zarr store, whereas `Predictive` runs on a single device with the full dataset in memory and returns results directly.

n	aimz time (s)	aimz RSS (MiB)	NumPyro time (s)	NumPyro RSS (MiB)
10K	0.4	607.1	0.9	44.3
100K	2.6	789.3	2.0	483.6
1M	21.3	966.5	9.3	4806.0
10M	200.1	1102.4	N/A	N/A
100M	2032.9	2225.3	N/A	N/A

Because `aimz` writes output incrementally to the Zarr store, its peak RSS grows slowly with n (from roughly 600 MiB at $n = 10$K to about 2.2 GiB at $n = 100$M), while NumPyro's RSS scales linearly and exceeds available memory before reaching $n = 10$M. With 1,000 posterior samples, the dominant cost at large n is persisting approximately $4n$ KB of predictive draws rather than the forward-pass computation itself. The `predict` method's streaming architecture is specifically designed for this regime, where the volume of output exceeds what can be held in memory. Closing this gap manually with `Predictive` (e.g., splitting data into batches, sharding across devices, JIT-compiling the sampling loop, incrementally collecting partial outputs) would effectively replicate the infrastructure that `aimz` already provides. For cases where the full output fits comfortably in memory, `aimz` also provides the `predict_on_batch` method, which returns results directly without disk I/O. The scripts and configuration details used to produce these results are available in the repository's `paper/benchmark/` directory.

In practice, `aimz` has supported internal analytics workflows within Eli Lilly and Company that require scalable posterior and posterior predictive sampling, consistent output structures, and experiment traceability. In this setting, the library has reduced duplicated glue code for streaming predictive draws, coordinating intervention scenarios, and tracking model lineage, thereby improving iteration speed and reproducibility across analyses. Its stable estimator-like interface and structured, reproducible outputs also make it well suited for integration with AI-enabled workflows, where agentic tools can leverage standardized artifacts for downstream tasks.

Beyond immediate use, `aimz` is designed as reusable research infrastructure: it exposes arbitrary NumPyro model functions through a stable estimator-like interface and standardizes artifacts (samples, predictions, and effect estimates) in formats that integrate cleanly with the broader scientific Python ecosystem (e.g., Xarray/Zarr) and with experiment tracking via MLflow. This makes it easier for researchers and applied scientists to share models and results, compare alternative specifications, and operationalize workflows on larger datasets than would be practical with bespoke scripts. Early external adoption is reflected in package index downloads: since its initial public release in June 2025, `aimz` has been downloaded over 10,000 times via PyPI and conda-forge as of January 2026.

AI usage disclosure

Microsoft Copilot was used solely to assist with drafting code docstrings and adding type hints. All suggestions generated by Copilot were carefully reviewed and edited to ensure accuracy and consistency with the implemented functionality. No other generative AI tools were used in the development of the software, the writing of the manuscript, or the preparation of supporting materials.

Acknowledgements

The author acknowledges support from Eli Lilly and Company.

References

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). *JAX: Composable transformations of Python+NumPy programs* (Version 0.3.13). http://github.com/jax-ml/jax

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., & Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of Statistical Software*, *76*(1), 1–32. https://doi.org/10.18637/jss.v076.i01

Hoyer, S., & Hamman, J. (2017). Xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*, *5*(1). https://doi.org/10.5334/jors.148

Miles, A., Kirkham, J., Durant, M., Bourbeau, J., Onalan, T., Hamman, J., Patel, Z., shikharsg, Rocklin, M., dussin, raphael, Schut, V., Andrade, E. S. de, Abernathey, R., Noyes, C., sbalmer, bot, pyup.io, Tran, T., Saalfeld, S., Swaney, J., … Banihirwe, A. (2020). *Zarr-developers/zarr-python: v2.4.0* (Version v2.4.0). Zenodo. https://doi.org/10.5281/zenodo.3773450

Oriol, A.-P., Virgile, A., Colin, C., Larry, D., J., F. C., Maxim, K., Ravin, K., Jupeng, L., C., L. C., A., M. O., Michael, O., Ricardo, V., Thomas, W., & Robert, Z. (2023). PyMC: A modern and comprehensive probabilistic programming framework in Python. *PeerJ Computer Science*, *9*, e1516. https://doi.org/10.7717/peerj-cs.1516

Phan, D., Pradhan, N., & Jankowiak, M. (2019). Composable effects for flexible and accelerated probabilistic programming in NumPyro. *arXiv Preprint arXiv:1912.11554*.

PyMC Labs. (2025). *Marketing statistical models in PyMC* (Version 0.16.0). https://github.com/pymc-labs/pymc-marketing

Taylor, S. J., & Letham, B. (2018). Forecasting at scale. *The American Statistician*, *72*(1), 37–45. https://doi.org/10.1080/00031305.2017.1380080

Team, G. M. M. M. M. (2025). *Meridian: Marketing mix modeling* (Version 1.2.1). https://github.com/google/meridian

Zaharia, M. A., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M., Xie, F., & Zumar, C. (2018). Accelerating the machine learning lifecycle with MLflow. *IEEE Data Eng. Bull.*, *41*, 39–45. https://dblp.org/rec/journals/debu/ZahariaCD0HKMNO18

Zhou, G., Lares, B., Skokan, I., & Sentana, L. (2024). *Robyn: Semi-automated marketing mix modeling (MMM) from Meta marketing science* (Version 3.12.0). https://doi.org/10.32614/cran.package.robyn