

KeepDelta: A Python Library for Human-Readable Data Differencing

Aslan Noorghasemi¹ and Christopher McComb¹

¹ Department of Mechanical Engineering, Carnegie Mellon University, USA  Corresponding author

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Open Journals](#) 

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

Efficiently managing evolving data is crucial in applications like computational simulations and sensing, where dynamic data tracking and processing are essential. In simulations, the traditional method known as full-state encoding stores the entire system state, including all nested data structures and variable values, at every timestep. While simple to implement, this approach is highly storage intensive. On the other hand, recalculating states from scratch to avoid storage demands is computationally expensive. Similarly, in sensing, continuously transmitting full data snapshots is inefficient, leading to increased bandwidth consumption and latency. *KeepDelta* addresses this challenge by providing a lightweight Python library that captures and applies only the changes (deltas) between successive states of complex, nested Python data structures. Designed for clarity and ease of use, *KeepDelta* produces human-readable outputs, facilitating debugging and analysis in research applications.



Figure 1: KeepDelta logo

Statement of need

High-frequency data sampling is fundamental in both scientific simulations and real-world sensing applications, where large volumes of evolving data must be efficiently managed. Both domains, whether generating synthetic data through computational models or collecting real-time measurements from physical sensors, face a common challenge: storing, transmitting, and processing dynamic data without excessive redundancy.

First and foremost, *KeepDelta* applies to simulation. Simulation is a widely used methodology across all applied science disciplines, offering a flexible, powerful, and intuitive tool for designing processes or systems and maximizing their efficiency (Kleijnen, 2018). Specifically, computational simulations are invaluable tools for studying complex systems and their behaviors (Aumann, 2007). These studies often take the form of computer experiments, where data is generated through pseudo-random sampling from known probability distributions. This approach serves as an invaluable resource for research, particularly in evaluating new methods and comparing alternative approaches (Morris et al., 2019).

Secondly, *KeepDelta* also applies to sensing. Sensing technologies are employed across diverse scientific and engineering domains, enabling continuous monitoring and analysis of dynamic environments. It is common for these systems to utilize a set of sensors to capture real-time

34 data, which is essential for studying systems behaviors, informed decision-making, and system
35 optimization (Al-Qurabat et al., 2021; Felton et al., 2018).

36 Both simulations and sensing require mechanisms to track and store evolving states of data
37 structures over time. In simulations, the naive approach of saving full snapshots at every
38 timestep leads to excessive storage demands, while recalculating states from scratch is compu-
39 tationally expensive. Similarly, in sensing applications, continuously storing or transmitting
40 full data snapshots is impractical, particularly in bandwidth-limited and remote environments.
41 Instead of relying on these easy-to-implement but inefficient methods, KeepDelta introduces
42 an optimized middle ground by saving only the *deltas* (changes) between states, significantly
43 reducing storage and computation overhead. This Delta Encoding technique has been success-
44 fully applied in other domains where managing evolving data efficiently is critical, such as web
45 development (Hoff et al., 2002) and software version management (Percival, 2003).

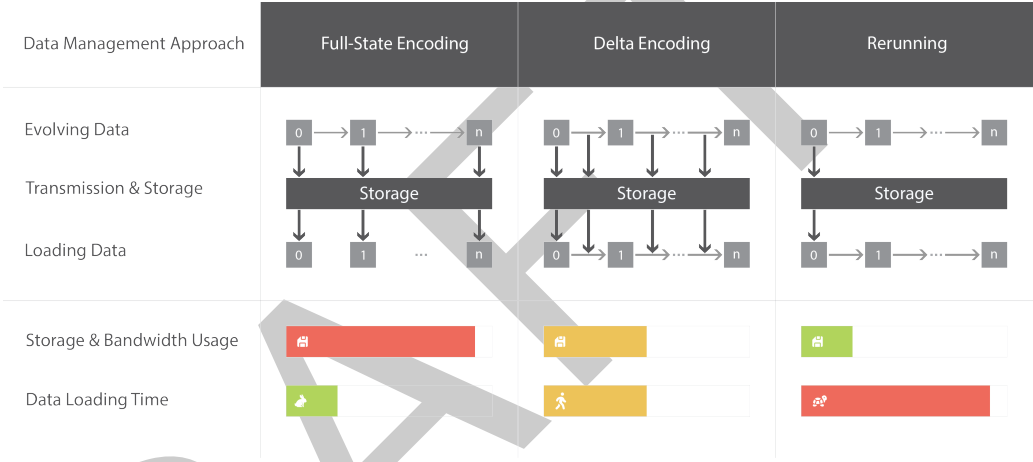


Figure 2: Comparison of data management approaches in evolving systems. Full-state encoding incurs high storage and bandwidth costs, while delta encoding saves only changes for efficiency. Rerunning reduces storage but increases computation and is often impractical for sensing real-world data. The bottom gradients illustrate trade-offs: storage/bandwidth decrease left to right, while data loading time increases.

46 Human-readability is crucial for debugging and development in both simulation and sensing
47 projects, and KeepDelta is tailored specifically for this purpose. It is lightweight and has no
48 dependencies, supports native Python data structures, and generates results that are easy to
49 interpret. This makes it an ideal tool for Python developers and researchers seeking a simple
50 yet powerful solution for change management. By providing clear, human-readable output,
51 KeepDelta enhances both the development process and debugging efficiency, making it easier
52 to track and manage changes in complex projects involving both computational models and
53 real-world sensing systems.



Figure 3: An example of human-readable change tracking with KeepDelta. (a) Previous smart-home state. (b) State after updates to temperature, lights, door lock, schedule, and alerts. (c) Delta produced by KeepDelta, which records only the changes—numeric adjustments, boolean toggles, a tuple insertion, and a new set element, offering a concise, easily interpretable summary.

Comparison to Existing Tools

In the landscape of Python libraries designed for delta encoding, several notable tools have emerged, each with distinct features and applications. `xdelta3` (Colvin, 2017) and its predecessor, `xdelta` (MacDonald, 2016), are tools that perform delta encoding at the binary level. These utilities are particularly effective for binary file differencing and are widely used in version control systems and data synchronization tasks. However, both are considered outdated and are no longer actively maintained. Their operation at the binary level results in outputs that are not human-readable, and they are not tailored for Python data structures, limiting their applicability in Python-centric workflows. `detoools` (Moqvist, 2023) is a Python package that focuses on applying and generating binary patches using a custom delta algorithm optimized for embedded systems. While it is efficient and actively maintained, `detoools` operates strictly at the binary level, producing outputs that are not human-readable and lacking support for native Python data structures. As such, it is more suitable for firmware and embedded development than for Python-centric simulation or data analysis workflows. `difflib` is part of the Python Standard Library and requires no external dependencies. It provides unified- and context-style text diffs for strings and lists, making it readily available but limited to sequence comparison rather than structured data differencing. It does not expose a programmatic delta object, so you cannot save its diff output as a delta for later state reconstruction. `datadiff` (Brondsema & R. Coombs, 2023) offers human-readable, unified-diff style comparisons of Python objects, including nested dictionaries, sequences, sets, and multi-line strings, and integrates with testing frameworks via `nose-assert`. Similar to `difflib`, it produces text-based diffs, but unlike `difflib`, it handles arbitrary Python objects rather than only strings or lists. However, like `difflib`, `datadiff` only generates one-off diffs and does not expose a programmatic delta object for incremental change tracking or later state reconstruction. `deeptdiff` (Dehpour, 2025) is a contemporary library that facilitates the identification of differences between complex Python data structures, including dictionaries, lists, and sets. It extends support to external libraries like NumPy (Harris et al., 2020), enhancing its versatility. DeepDiff also offers a range of configuration options, such as ignoring iterable order, controlling significant-digit precision in numeric comparisons, custom comparator functions, and an optional `deeptdiff[optimize]`

83 installation for performance tuning in certain processes such as json serialization which makes it
 84 highly flexible but adds complexity and dependencies. Also, this extensibility can lead to outputs
 85 that are less human-readable compared to KeepDelta, and the added dependencies may not be
 86 necessary for all projects. In contrast to these alternatives, KeepDelta is a lightweight Python
 87 library tailored for simulations and sensing, focusing on efficient delta management for built-in
 88 Python data structures. It produces human-readable outputs that facilitate debugging and
 89 research workflows. Written entirely in Python and free of external dependencies, KeepDelta
 90 integrates seamlessly into Python-centric workflows, including simulation, sensing, and data
 91 analysis pipelines.

92 References

- 93 Al-Qurabat, A. K. M., Mohammed, Z. A., & Hussein, Z. J. (2021). Data traffic management
 94 based on compression and MDL techniques for smart agriculture in IoT. *Wirel. Pers.*
 95 *Commun.*, 120(3), 2227–2258. <https://doi.org/10.1007/s11277-021-08563-4>
- 96 Aumann, C. A. (2007). A methodology for developing simulation models of complex systems.
 97 *Ecological Modelling*, 202(3), 385–396. <https://doi.org/10.1016/j.ecolmodel.2006.11.005>
- 98 Brondsema, D., & R. Coombs, J. (2023). *DataDiff* (Version 2.2.0). [https://pypi.org/project/](https://pypi.org/project/datadiff/)
 99 [datadiff/](https://pypi.org/project/datadiff/)
- 100 Colvin, S. (2017). *xdelta3*. In *GitHub repository* (Version 0.0.5). GitHub. [https://github.](https://github.com/samuelcolvin/xdelta3-python)
 101 [com/samuelcolvin/xdelta3-python](https://github.com/samuelcolvin/xdelta3-python)
- 102 Dehpour, S. (2025). *DeepDiff*. In *GitHub repository* (Version 8.4.2). GitHub. [https:](https://github.com/seperman/deepdiff)
 103 [//github.com/seperman/deepdiff](https://github.com/seperman/deepdiff)
- 104 Felton, C. L., Gilbert, B. K., & Haider, C. R. (2018). Data compression via low complexity delta
 105 transition lossless encoding for remote physiological and environmental monitoring. *2018*
 106 *40th Annual International Conference of the IEEE Engineering in Medicine and Biology*
 107 *Society (EMBC)*, 4379–4384. <https://doi.org/10.1109/EMBC.2018.8513277>
- 108 Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D.,
 109 Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk,
 110 M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., ... Oliphant,
 111 T. E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362. [https:](https://doi.org/10.1038/s41586-020-2649-2)
 112 [//doi.org/10.1038/s41586-020-2649-2](https://doi.org/10.1038/s41586-020-2649-2)
- 113 Hoff, A. van, Douglass, F., Krishnamurthy, B., Golland, Y. Y., Hellerstein, D. M., Feldmann,
 114 A., & Mogul, J. (2002). *Delta encoding in HTTP* (No. 3229). RFC 3229; RFC Editor.
 115 <https://doi.org/10.17487/RFC3229>
- 116 Kleijnen, J. P. C. (2018). Design and analysis of simulation experiments. In J. Pilz, D. Rasch, V.
 117 B. Melas, & K. Moder (Eds.), *Statistics and simulation* (pp. 3–22). Springer International
 118 Publishing. ISBN: 978-3-319-76035-3
- 119 MacDonald, J. (2016). *xdelta*. In *GitHub repository* (Version 3.1.0). GitHub. [https:](https://github.com/jmacd/xdelta)
 120 [//github.com/jmacd/xdelta](https://github.com/jmacd/xdelta)
- 121 Moqvist, E. (2023). *detools*. In *GitHub repository* (Version 0.53.0). GitHub. [https://github.](https://github.com/erimoq/detools)
 122 [com/erimoq/detools](https://github.com/erimoq/detools)
- 123 Morris, T. P., White, I. R., & Crowther, M. J. (2019). Using simulation studies to evaluate
 124 statistical methods. *Statistics in Medicine*, 38(11), 2074–2102. [https://doi.org/10.1002/](https://doi.org/10.1002/sim.8086)
 125 [sim.8086](https://doi.org/10.1002/sim.8086)
- 126 Percival, C. (2003). Naive differences of executable code. *Draft Paper*, [Http://Www.dae-](http://www.dae-monology.net/Bsdiff)
 127 [monology.net/Bsdiff](http://www.dae-monology.net/Bsdiff).