KeepDelta: A Python Library for Human-Readable Data Differencing

Aslan Noorghasemi [1¶] **and Christopher McComb** [1]

1 Carnegie Mellon University, Pittsburgh, PA, USA ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

Efficiently managing evolving data states is crucial in cases like computational simulations and IoT remote sensing, where dynamic data tracking and processing are essential. In simulations, traditional methods of storing complete snapshots (full-state encoding) at each step can be highly storage-intensive, while recalculating states from scratch is computationally expensive. Similarly, in IoT remote 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.

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.

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). 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). Computational simulations are invaluable tools for studying complex systems and their behaviors. (Aumann, 2007)

Sensing technologies are also employed across diverse scientific and engineering domains, enabling continuous monitoring and analysis of dynamic environments. It is common for these systems utilize an array of sensors to capture real-time data, which is essential for studying systems behaviors, informed decision-making, and system optimization (Felton et al., 2018) (Al-Qurabat et al., 2021).

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

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

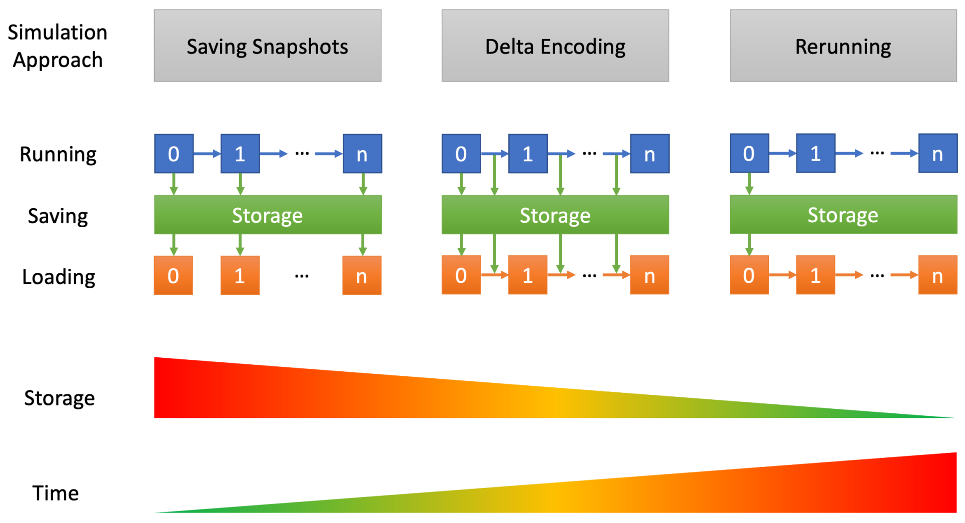

Figure 1: caption

In practice

To install the current version of the KeepDelta package, run the command:

```
pip install keepdelta
```

Then, import it into your project:

~~~
import keepdelta as kd
``F

`create` and `apply` are the two core functionalities of the library.

To *create* the delta between `old` and `new` variables:
```python
delta = kd.create(old, new)
~~~

To *apply* the delta to `old` variable to reconstruct `new` variable:

```
new = kd.apply(old, delta)
```

KeepDelta supports nested and mixed native Python data structures. Native Python varibales types are: - Primitives: integer (`int`), float (`float`), string (`str`), boolean (`bool`), complex

(`complex`), none (`None`) - Collections: list (`list`), tuple (`tuple`), dictionary (`dict`), set(`set`)

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, n.d.) and its predecessor, xdelta (MacDonald, n.d.), 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. detools (Moqvist, n.d.) is a more recent addition, offering binary delta encoding capabilities within Python applications. It efficiently handles large binary files and is designed for ease of use. Nevertheless, like xdelta3 and xdelta, detools operates at the binary level, producing non-human-readable outputs and not being specifically optimized for Python's native data structures. deepdiff (Dehpour, n.d.) 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. However, this integration can lead to outputs that are less human-readable compared to KeepDelta. Additionally, deepdiff introduces dependencies that may not be necessary for all projects. In contrast, KeepDelta is a lightweight Python library optimized for simulations, focusing on efficient delta management for native Python data structures. It operates directly on Python's native data types, producing human-readable outputs that facilitate debugging and research applications. Implemented in pure Python, KeepDelta eliminates external dependencies, making it an ideal choice for Python-centric workflows, particularly in simulations and data analysis tasks.

References

Al-Qurabat, A. K. M., Mohammed, Z. A., & Hussein, Z. J. (2021). Data traffic management based on compression and MDL techniques for smart agriculture in IoT. *Wirel. Pers. Commun.*, *120*(3), 2227–2258.

Aumann, C. A. (2007). A methodology for developing simulation models of complex systems. *Ecological Modelling*, *202*(3), 385–396. https://doi.org/https://doi.org/10.1016/j.ecolmodel.2006.11.005

Colvin, S. (n.d.). *xdelta3* (Version 0.0.5). https://github.com/samuelcolvin/xdelta3-python

Dehpour, S. (n.d.). *DeepDiff* (Version 8.1.1). https://github.com/seperman/deepdiff

Felton, C. L., Gilbert, B. K., & Haider, C. R. (2018). Data compression via low complexity delta transition lossless encoding for remote physiological and environmental monitoring. *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 4379–4384. https://doi.org/10.1109/EMBC.2018.8513277

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hoff, A. van, Douglis, F., Krishnamurthy, B., Goland, Y. Y., Hellerstein, D. M., Feldmann, A., & Mogul, J. (2002). *Delta encoding in HTTP* (No. 3229). RFC 3229; RFC Editor. https://doi.org/10.17487/RFC3229

Kleijnen, J. P. C. (2018). Design and analysis of simulation experiments. In J. Pilz, D. Rasch, V.

B. Melas, & K. Moder (Eds.), *Statistics and simulation* (pp. 3–22). Springer International Publishing. ISBN: 978-3-319-76035-3

MacDonald, J. (n.d.). *xdelta* (Version 3.1.0). https://github.com/jmacd/xdelta

Moqvist, E. (n.d.). *detools* (Version 0.53.0). https://github.com/eerimoq/detools

Morris, T. P., White, I. R., & Crowther, M. J. (2019). Using simulation studies to evaluate statistical methods. *Statistics in Medicine*, *38*(11), 2074–2102. https://doi.org/https://doi.org/10.1002/sim.8086

Percival, C. (2003). Naive differences of executable code. *Draft Paper, Http://Www.daemonology.net/Bsdiff*.