

Consist: Provenance-Aware Caching and Queryable Provenance for Scientific Workflows

Zachary A. Needell ¹ ¶

¹ Lawrence Berkeley National Laboratory  ¶ Corresponding author

DOI: [10.21105/joss.11131](https://doi.org/10.21105/joss.11131)

Software

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

Editor: [Adithi R Upadhy](#) 

Reviewers:

- [@DeIn0r](#)
- [@sk8ordie84](#)

Submitted: 17 June 2026

Published: 27 October 2026

License

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

Summary

Computational scientific workflows produce outputs that depend on three things: the code used to produce them, the configuration settings that parameterize their behavior, and the set of input objects they consume. Each of those inputs may itself be the output of an earlier computational step, creating dependency chains that can extend across model runs. In practice, these chains are rarely explicit: researchers frequently cannot trace which exact code, configuration, and upstream inputs produced a given result, or which downstream outputs are affected by an upstream change.

Consist is a Python library that makes this lineage explicit, persistent, and queryable. It wraps ordinary Python computations in a provenance layer that records which code, configuration, and input artifacts produced each output, computing a deterministic identity signature. Each step's identity incorporates the identities of its inputs, including the signatures of the runs that produced them. As a result, any output can be traced back through its full chain of upstream dependencies to the original inputs and assumptions at the system boundary. A consequence of this identity-based lineage is correct, automatic caching: upstream changes invalidate only the downstream steps that genuinely depend on them, and unchanged steps are safely reused without re-execution.

Provenance is persisted in a local DuckDB database ([Raasveldt & Mühleisen, 2019](#)) that also serves as an analytical engine: users can query across run outputs—whether ingested or remaining distributed on disk—so that cross-scenario comparison, lineage tracing, and result recovery are available through standard SQL. The entire stack is self-contained with no external services, reducing barriers to provenance tracking in research settings where simplicity of deployment matters.

Statement of Need

Many scientific domains rely on workflows where multiple simulation models interact iteratively: climate projections couple atmosphere and ocean models, energy system analyses couple capacity expansion with dispatch, and epidemiological studies couple population dynamics with interventions. In transportation planning—the domain that motivated Consist—land-use models produce population and employment forecasts; vehicle ownership models determine household fleets; activity-based demand models generate travel behavior; and network assignment models evaluate congestion. These models run iteratively across multi-year horizons under multiple policy scenarios, with congestion feeding back into demand and demand feeding back into land use.

These workflows create four recurring problems. First, **provenance ambiguity**: when a result changes between runs, it is often unclear which upstream input, configuration parameter, or code change caused the difference. Second, **redundant recomputation**: extending a converged 10-

iteration equilibrium run to 15 iterations should not require re-executing the first 10, yet without explicit state management it typically does. Third, **scattered outputs**: results accumulate as directories of files with ad hoc naming, making systematic comparison across scenarios difficult. Fourth, **implicit wiring**: dependencies between steps are encoded through file paths, global variables, or framework conventions rather than in inspectable, queryable structure. Consist addresses these problems by treating each workflow step as a provenance-aware computation with explicit identity and lineage.

State of the Field

Existing tools address fragments of this problem. Snakemake (Mölder et al., 2021) provides directed acyclic graph (DAG)-based execution with file-content-based recomputation, but it requires expressing workflows in its own rule-based domain-specific language (DSL) rather than wrapping existing Python code, and its focus is correct re-execution rather than a persistent, queryable provenance store. Workflow engines such as Nextflow (Di Tommaso et al., 2017) similarly emphasize orchestration without providing queryable provenance. Earlier Python-native provenance tools such as Sumatra (Davison, 2012) and Recipy (Jackson et al., 2018) track runs with minimal code changes but lack identity-based caching, multi-step lineage propagation, and queryable output stores. DVC (Kuprieiev et al., 2026) versions data files and pipelines but targets data-asset management rather than run-level identity and experiment history, while MLflow (Chen et al., 2020) offers rich experiment tracking oriented toward machine-learning training loops rather than multi-step simulations with heterogeneous configuration ecosystems. OpenLineage (OpenLineage contributors, 2026) standardizes lineage event emission but does not itself persist or use provenance or lineage metadata; Consist can optionally emit OpenLineage events while natively handling provenance storage, querying, and identity-based cache resolution.

Apache Hamilton (Krawczyk & ben lzzy, 2022) is the closest existing tool: it builds function-level DAGs with caching and Merkle-style identity propagation. However, Hamilton requires its steps to be pure Python functions, a major barrier for scientific workflows that rely on containers and subprocesses. It also lacks support for the complex configuration structures common in simulation pipelines and provides no analytical engine to make provenance metadata and artifact contents queryable.

Consist fills this gap by combining non-invasive provenance tracking, deterministic identity-based caching with lineage propagation, and a queryable analytical surface for post-hoc comparison—without requiring a workflow DSL or external services.

Software Design

The Identity Model

Consist's core mechanism is a deterministic three-part signature computed for each computational step: `signature = SHA256(code_hash || config_hash || input_hash)`. The code hash captures relevant implementation state (configurable from Git state, callable-scoped source code, or container manifest). The config hash captures a canonicalized dictionary of model settings. The input hash captures the identity of declared input artifacts, and when those artifacts were themselves produced by Consist, their identity incorporates the producing run's signature, inducing a Merkle-style DAG. A change to any upstream step propagates invalidation only through genuinely dependent downstream computations (Figure 1).

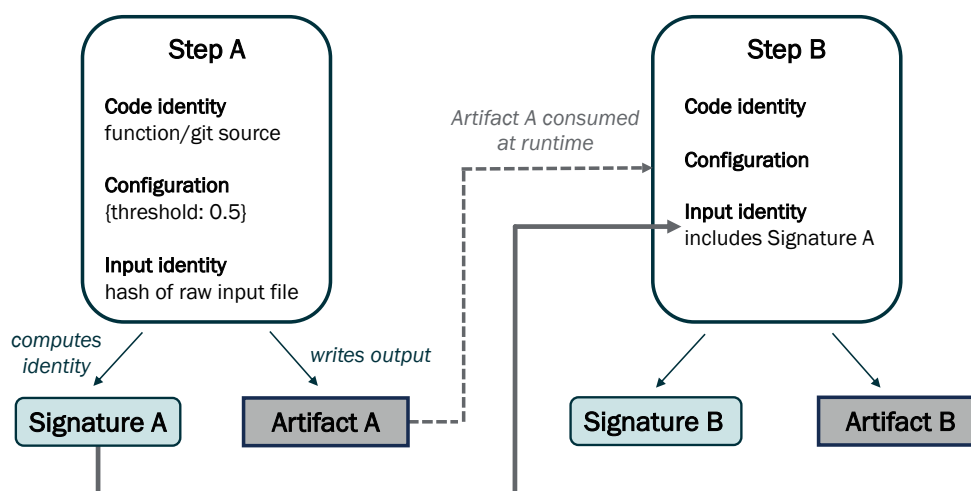


Figure 1: Identity propagation across workflow steps. Each step's signature is computed from its code identity, configuration, and input identity. When Step B consumes an artifact produced by Step A, Signature A is incorporated into Step B's input identity (solid arrow), propagating cache invalidation through the workflow. The artifact's bytes are consumed separately at runtime (dashed arrow) and need not be materialized when a cache hit makes re-execution unnecessary.

Why This Is Hard in Practice

Real workflows complicate this clean conceptual model.

Structured configurations: In complex simulation ecosystems, “configuration” simultaneously plays several roles: experimental parameters, descriptive metadata, and external file trees (YAML, CSV, XML). Consist separates these explicitly into identity-bearing configuration (structured parameters that are both hashed for cache identity and indexed for analytical queries), facets (indexed but not hashed), and identity inputs (files or directories hashed but not indexed).

Functions with side effects: Domain models can write files, call subprocesses, and interact with external tools. Consist tracks such processes through their declared inputs, configuration, and outputs without requiring changes to domain code. Artifacts live on disk rather than in memory, and Consist loads or attaches them to steps only as required, better accommodating outputs too large to hold in memory. Separating “hydration” (recovering metadata and provenance) and “materialization” (ensuring bytes exist on disk) allows a cache hit to resolve cheaply without reading or duplicating files at all. Correctness, however, depends on inputs and code changes being faithfully declared—undeclared dependencies (e.g., untracked files, environment state, or non-deterministic computation) can produce stale cache hits.

Post-run analysis: Consist persists provenance in both JSON snapshots and a DuckDB database for analytical querying. Selected tabular outputs can be ingested alongside provenance metadata, while large or infrequently queried outputs remain file-backed. Users can query across both through a unified analytical surface with provenance columns (run ID, scenario, year) attached for grouping and filtering. Consequently, agent-level trip records and population tables—millions of rows stored as distributed Parquet files—can be queried through a single SQL surface, enabling analyses such as grouping vehicle miles traveled by home geography across policy scenarios without loading full datasets into memory.

Graduated Formalization

At a minimum, Consist requires explicit inputs, outputs, and identity-bearing configuration for each step—typically a few additional lines per function call. Users can optionally add facets for analytical filtering, configuration adapters for complex domain settings (current adapters exist for ActivitySim (Zephyr Foundation, 2026) and BEAM (Laarabi et al., 2024)), and typed artifact schemas. Each layer delivers proportionally richer capabilities, from basic cache reuse through queryable provenance to memory-efficient typed analytical views over large simulation outputs.

Research Impact Statement

Consist was developed to support PILATES, a workflow runtime used in the BEAM CORE modeling ecosystem at Lawrence Berkeley National Lab (Spurlock et al., 2024). PILATES coordinates UrbanSim (Waddell, 2002) (land use), ATLAS (Jin et al., 2024) (vehicle ownership), ActivitySim (Zephyr Foundation, 2026) (travel demand), and BEAM (Laarabi et al., 2024) (network assignment) across multi-year, multi-iteration scenario lifecycles. PILATES is currently used in DOE-funded scenario analyses of vehicle technology change, curb utilization, and supply-chain optimization.

During PILATES integration, adding Consist required making model-step input and output bindings explicit. That process surfaced a latent cross-model dependency error: ActivitySim's population-source input was bound to a valid but wrong-year UrbanSim artifact, using the current-year datastore rather than the forecast-year population snapshot produced after ATLAS updates. The issue was difficult to detect from individual files alone because each artifact was internally valid; the lineage graph made the semantic mismatch between upstream artifact roles visible.

Consist's caching also delivers practical savings: scenarios that vary ActivitySim or BEAM configurations can reuse cached upstream UrbanSim and ATLAS steps. Six example notebooks demonstrate these mechanics in self-contained workflows, covering cache behavior, scenario comparison, parameter sweeps, iterative extension, and cross-run analysis.

Availability

Consist is available on PyPI (`pip install consist`) and developed openly at <https://github.com/LBNL-UCB-STI/consist> under a BSD 3-Clause license. Documentation including a quickstart, usage guide, API documentation, and the example notebook gallery is hosted at <https://lbnl-ucb-sti.github.io/consist/latest/>. The repository includes automated tests run via continuous integration and contribution guidelines (CONTRIBUTING.md) that include a code of conduct. Consist requires Python 3.11+ and depends on DuckDB, SQLAlchemy, pandas, and PyArrow, with optional dependencies for spatial formats, xarray-based matrix views, and data ingestion via dlt.

AI Usage Disclosure

Anthropic Claude Opus 4.5–4.7 and OpenAI ChatGPT 5.2–5.5 were used across software development, documentation, and paper editing. The author reviewed, edited, and validated all AI-assisted text and code; AI-assisted software components were validated through automated tests and real-world research use in PILATES. All architectural decisions, technical claims, and domain-specific content reflect the author's direct experience developing and deploying Consist.

Acknowledgements

This paper and the work described were sponsored by the U.S. Department of Energy (DOE) Vehicle Technologies Office (VTO) under the Systems and Modeling for Accelerated Research in Transportation (SMART) Mobility Laboratory Consortium, an initiative of the Energy Efficient Mobility Systems (EEMS) Program. Lawrence Berkeley National Laboratory operates under DOE Contract No. DE-AC02-05CH11231. The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes.

References

- Chen, A., Chow, A., Davidson, A., DCunha, A., Ghodsi, A., Hong, S. A., Konwinski, A., Mewald, C., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M., Singh, A., Xie, F., Zaharia, M., Zang, R., Zheng, J., & Zumar, C. (2020). Developments in MLflow: A system to accelerate the machine learning lifecycle. *Proceedings of the Fourth Workshop on Data Management for End-to-End Machine Learning*. <https://doi.org/10.1145/3399579.3399867>
- Davison, A. (2012). Automated capture of experiment context for easier reproducibility in computational research. *Computing in Science & Engineering*, 14(4), 48–56. <https://doi.org/10.1109/MCSE.2012.41>
- Di Tommaso, P., Chatzou, M., Floden, E. W., Barja, P. P., Palumbo, E., & Notredame, C. (2017). Nextflow enables reproducible computational workflows. *Nature Biotechnology*, 35(4), 316–319. <https://doi.org/10.1038/nbt.3820>
- Jackson, M., Wilson, R., Zwaan, J. van der, Steinbrook, D. W., Rathgeber, F., Alegre, R., Edwards, T., Costello, C., Danny, S., Chwiejczak, J., Cottaar, M., Martinez-Ortiz, C., Vieglais, D., Kucharavy, A., Ruben, G., & Upadhyay, U. (2018). *recipy* (Version v0.3.1). <https://github.com/recipy/recipy>
- Jin, L., Jackson, C. P., Wang, Y., Chen, Q., Ho, T., Spurlock, C. A., Brooker, A., Holden, J., Gonder, J., Bouzaghrane, M. A., Sun, B., Sharda, S., Garikapati, V., Wenzel, T., & Caicedo, J. (2024). Technology progress and clean vehicle policies on fleet turnover and equity: Insights from household vehicle fleet micro-simulations with ATLAS. *Transportation Planning and Technology*, 47(8), 1399–1422. <https://doi.org/10.1080/03081060.2024.2353784>
- Krawczyk, S., & ben Izzy, E. (2022). Hamilton: A modular open source declarative paradigm for high level modeling of dataflows. In S. R. Valluri & M. Zait (Eds.), *1st International Workshop on Composable Data Management Systems, CDMS@VLDB 2022, Sydney, Australia, September 9, 2022*. https://cdmsworkshop.github.io/2022/Proceedings/ShortPapers/Paper6_StefanKrawczyk.pdf
- Kuprieiev, R., skshetry, Rowlands, P., Petrov, D., Redzyński, P., Costa-Luis, C. da, Iglesia Castro, D. de la, Schepanovski, A., Shcheklein, I., Gao, Y., Taskaya, B., Orpinel, J., Santos, F., Daniele, Lamy, R., Sharma, A., Kaimuldenov, Z., Hodovic, D., Kodenko, N., ... Mangal, S. (2026). *DVC: Data Version Control - Git for Data & Models* (Version 3.67.1). Zenodo. <https://doi.org/10.5281/zenodo.19344026>
- Laarabi, H., Needell, Z., Waraich, R., Poliziani, C., & Wenzel, T. P. (2024). *A modeling framework for behavior, energy, autonomy and mobility (BEAM)*. Lawrence Berkeley National Laboratory. <https://doi.org/10.48550/arXiv.2308.02073>
- Mölder, F., Jablonski, K. P., Letcher, B., Hall, M. B., Tomkins-Tinch, C. H., Sochat, V., Forster, J., Lee, S., Twardziok, S. O., Kanitz, A., Wilm, A., Holtgrewe, M., Rahmann, S., Nahnsen, S., & Köster, J. (2021). Sustainable data analysis with Snakemake. *F1000Research*, 10,

33. <https://doi.org/10.12688/f1000research.29032.2>
- OpenLineage contributors. (2026). *OpenLineage: An open framework for data lineage collection and analysis*. Software documentation. <https://openlineage.io/>
- Raasveldt, M., & Mühleisen, H. (2019). DuckDB: An embeddable analytical database. *Proceedings of the 2019 International Conference on Management of Data*, 1981–1984. <https://doi.org/10.1145/3299869.3320212>
- Spurlock, C. A., Bouzaghrane, M. A., Brooker, A., Caicedo, J., Gonder, J., Holden, J., Jeong, K., Jin, L., Laarabi, H., Needell, Z., Poliziani, C., Sharda, S., Sun, B., Waddell, P., Wang, Y., Waraich, R., Wenzel, T. P., & Xu, X. (2024). *Behavior, energy, autonomy & mobility comprehensive regional evaluator: Overview, calibration and validation summary of an agent-based integrated regional transportation modeling workflow*. Lawrence Berkeley National Laboratory. <https://transportation.lbl.gov/publications/behavior-energy-autonomy-mobility>
- Waddell, P. (2002). UrbanSim: Modeling urban development for land use, transportation, and environmental planning. *Journal of the American Planning Association*, 68(3), 297–314. <https://doi.org/10.1080/01944360208976274>
- Zephyr Foundation. (2026). *ActivitySim*. <https://github.com/ActivitySim/activitysim>