The Journal of Open Source Software

sklearn-migrator: Cross-version migration of scikit-learn models for reproducible MLOps

Alberto Andres Valdes Gonzalez [1]

1 Independent Researcher (Chile)

DOI: 10.21105/joss.10374

Software

- Review
- Repository
- Archive

Editor: Mehmet Hakan Satman

Reviewers:

- @solegalli
- @smk508

Submitted: 13 December 2025
Published: 19 May 2026

Summary

`sklearn-migrator` is a Python library for **safely migrating scikit-learn models across versions** while preserving inference behavior and remaining robust to internal attribute changes. scikit-learn is among the most widely used machine learning libraries in both research and industry, and its estimators are commonly deployed in tabular-data domains such as finance, risk, operations, and marketing (Kaggle, 2021; Pedregosa et al., 2011). In these settings, model upgrades often coincide with dependency upgrades, container base-image updates, or security patching cycles—making version-to-version portability a practical requirement for production MLOps.

The core problem is that standard persistence mechanisms (pickle/joblib) are **version-fragile**: models saved under one scikit-learn release may fail to load—or may load with altered behavior—under another. This limitation is explicitly cautioned in the official documentation and has been observed in empirical and experiential work (Fitzpatrick & Manning, 2024; Parida et al., 2025; scikit-learn developers, 2025). `sklearn-migrator` addresses this gap by exporting supported estimators into **portable, JSON-compatible Python dictionaries** and reconstructing them in a different environment running a target scikit-learn version. The resulting payloads are readable, inspectable, and transportable across environments and teams, enabling long-term reproducibility and governance.

The migration proceeds in two stages. **Stage 1 (parity):** the library captures a minimal set of prediction-critical attributes that guarantee parity of outputs across versions, validated under a strict tolerance (e.g., `max |y_in - y_out| < 1e-2`). **Stage 2 (compatibility):** remaining attributes are serialized with a version-aware policy that gracefully handles additions, removals, and renames so deserialization does not break across releases. This strategy is designed around the practical reality that estimator internals, defaults, and attribute names shift over time—even when the public API remains stable (Parida et al., 2025; scikit-learn developers, 2025).

This submission supports **21 models** across supervised and unsupervised learning:

Category	#	Estimators
Classification	7	`DecisionTreeClassifier`, `RandomForestClassifier`, `GradientBoostingClassifier`, `LogisticRegression`, `KNeighborsClassifier`, `SVC`, `MLPClassifier`

Category	#	Estimators
Regression	10	`DecisionTreeRegressor`, `RandomForestRegressor`, `GradientBoostingRegressor`, `LinearRegression`, `Ridge`, `Lasso`, `KNeighborsRegressor`, `SVR`, `AdaBoostRegressor`, `MLPRegressor`
Clustering	3	`AgglomerativeClustering`, `KMeans`, `MiniBatchKMeans`
Dimensionality Reduction	1	`PCA`
Total	**21**	

`sklearn-migrator` has been validated across **32 scikit-learn versions** (`0.21.3` → `1.7.2`), covering **1,024 migration pairs**, with automated, environment-isolated testing and strict parity checks. Continuous integration runs on every push: unit tests across four Python versions (3.9–3.12), followed by the automated build of 64 isolated Docker images and 21,504 (21 models × 32 × 32 version pairs) environment-isolated integration executions, each validated against a strict parity threshold.

Statement of need

Persisted scikit-learn models frequently break across library upgrades because internal attributes, defaults, and serialization details change over time. Standard persistence mechanisms (e.g., pickle/joblib) are **version-fragile**: a model saved under one release may fail to load—or load with altered behavior—under another (Fitzpatrick & Manning, 2024; Parida et al., 2025; scikit-learn developers, 2025). The resulting brittleness complicates production upgrades, environment migrations, cross-team sharing, and long-term reproducibility—especially in regulated or audit-heavy contexts where models must remain verifiable over time.

In practice, this fragility creates costly failure modes. A dependency upgrade may break deserialization of a mission-critical model artifact. Conversely, pinning old versions indefinitely increases security risk and operational burden. Teams often face an uncomfortable trade-off: **upgrade safely** (but risk breaking legacy models) or **freeze environments** (but accumulate technical debt). This is particularly acute in organizations that operate many model-serving services, notebooks, and batch pipelines with slightly different dependency constraints.

`sklearn-migrator` addresses this need with a two-stage, version-aware (de)serialization strategy. First, it captures the minimal, prediction-critical attributes to guarantee output parity between versions. Second, it serializes remaining attributes with explicit, version-conditioned defaults so that parameters added, removed, or renamed across releases do not break deserialization. Unlike pickle/joblib, the library uses portable, JSON-compatible Python dictionaries, enabling safe transport, inspection, and storage independent of the original runtime (Fitzpatrick & Manning, 2024; Parida et al., 2025).

The library targets practitioners and MLOps teams who must migrate or reproduce models across heterogeneous environments. It supports forward and backward migration across 32 scikit-learn releases (`0.21.3` → `1.7.2`), covering **1,024** version pairs with unit tests and environment-isolated validation.

State of the field

Model persistence and portability are longstanding challenges in applied machine learning (scikit-learn contributors, 2018). In the scikit-learn ecosystem, the officially recommended mechanisms—`pickle` and `joblib`—are explicitly documented as *not* guaranteeing forward or backward compatibility across library versions (scikit-learn developers, 2025). As a result, serialized models are tightly coupled to the exact scikit-learn release and Python environment in which they were created.

Several alternative approaches partially address related concerns, but each has significant limitations:

- **pickle/joblib**: the default scikit-learn persistence mechanism and the most widely used in practice, but explicitly documented as *not* guaranteeing cross-version compatibility (scikit-learn developers, 2025). A model serialized under one release may raise `AttributeError`, `ModuleNotFoundError`, or silently produce different predictions under another—with no warning to the user.
- **skops**: designed to replace pickle for *secure* model sharing by avoiding arbitrary code execution on deserialization. However, it does not address cross-version compatibility: a `.skops` file is still tied to the scikit-learn version used at serialization time, and loading it under a different release is subject to the same attribute-level breakage as pickle.
- **sklearn-onnx**: converts fitted estimators to the ONNX intermediate representation, enabling deployment across runtimes (e.g., `onnxruntime`, C++, Java). However, it covers only a subset of scikit-learn estimators, requires a non-trivial conversion step, introduces a dependency on the ONNX runtime, and does not guarantee numerical parity with the original scikit-learn predictions due to differences in floating-point arithmetic across backends (Parida et al., 2025).
- **PMML**: a mature XML-based interchange standard with broad tooling support across platforms and languages. However, estimator coverage is limited, the format is verbose and difficult to inspect programmatically, and numerical parity with the original model is not guaranteed—particularly for ensemble methods and neural networks.
- **Re-training**: the most common production workaround—simply refit the model on the target environment. This is often infeasible due to missing or proprietary training data, regulatory constraints that prohibit model changes, computational cost at scale, or the need to preserve a specific model artifact for audit or reproducibility purposes.

None of these alternatives directly addresses the version-fragility problem for native scikit-learn estimators in production. Pinning old scikit-learn versions accumulates Common Vulnerabilities and Exposures (CVEs) and security debt over time, exposing production systems to known vulnerabilities. This makes a migration tool particularly compelling in organizations where dependency upgrades are driven by security patching cycles.

`sklearn-migrator` addresses this gap by enabling native, Python-centric cross-version migration without re-training, runtime conversion, or version pinning—while maintaining full compatibility with the scikit-learn API.

Research Impact Statement

This software addresses a critical reproducibility challenge in applied machine learning: the inherent fragility of serialized `scikit-learn` models across library versions. Standard persistence mechanisms (e.g., `pickle`) tie model artifacts to specific environments, creating a "dependency lock-in" that hinders long-term research reproducibility and complicates production MLOps workflows.

By enabling deterministic, cross-version migration, `sklearn-migrator` mitigates the operational risk and technical debt associated with mandatory security patching and dependency upgrades.

The library is particularly impactful for high-stakes domains—such as finance, healthcare, and risk modeling—where models must remain verifiable over long lifecycles and where retraining may be computationally expensive or ethically restricted.

More broadly, this work promotes **transparent and inspectable model artifacts**, filling a significant gap in the machine learning ecosystem by providing a native, Python-centric path to model portability that complements existing interoperability standards.

Design and validation

Software Design

The design of `sklearn-migrator` follows a modular, estimator-centric architecture that separates model inspection, version-aware serialization, and controlled reconstruction in the target environment. Each supported estimator family is implemented as an **isolated migration unit**, allowing fine-grained handling of structural changes across versions without affecting unrelated models.

The migration pipeline consists of three functional components:

1. **Introspection Layer**: Extracts constructor parameters and prediction-critical attributes from a fitted estimator in the source environment.
2. **Version-aware Serialization Layer**: Encodes attributes into a portable, JSON-compatible dictionary, explicitly recording defaults for attributes that may not exist in all versions.
3. **Deserialization Layer**: Reconstructs an equivalent estimator in the target environment, assigning only attributes valid for the destination version and safely skipping deprecated fields.

Serialization format

Each supported estimator is serialized into a Python dictionary containing:

1. **Metadata**: source version, estimator type, and migration-relevant flags.
2. **Parity-critical reconstruction parameters**: the minimal set of fields required to produce matching predictions under strict tolerance.
3. **Compatibility attributes**: additional learned attributes stored with version-aware rules, including explicit defaults for fields that exist only in some versions.

The library restricts values to JSON-encodable primitives (numbers, strings, booleans, lists, dicts), enabling storage in plain JSON files, object storage, or artifact registries, and supporting inspection without executing arbitrary code.

Validation methodology

`sklearn-migrator` validates migrations through:

- **Environment isolation**: 64 dedicated Docker images (32 source + 32 target versions) are built and pushed to GHCR on every push, ensuring `version_in` and `version_out` represent fully independent, real scikit-learn installations with no dependency leakage.
- **Automated threshold enforcement**: each of the 21,504 migration executions (21 models $\times$ 32 $\times$ 32 version pairs) is validated automatically against `max |y_in - y_out|` $\leq$ `1e-2`; any failure blocks the pipeline.
- **Multi-Python compatibility**: unit tests run in parallel across Python 3.9, 3.10, 3.11, and 3.12 on every push.
- **Fixed synthetic datasets** for deterministic evaluation.

Each of the 21 supported models is tested with a specific parameter configuration, ranging from default instantiations (e.g., `LinearRegression()`, `RandomForestClassifier()`)

to configurations with explicit parameters (e.g., `DecisionTreeClassifier(max_depth=4, random_state=42)`, `PCA(n_components=2, whiten=True, svd_solver='full')`). Exhaustive combinatorial testing is outside the current scope.

The library has been tested across a full 32×32 version compatibility matrix, totaling **1,024 migration pairs**.

Example

The example below trains a `RandomForestRegressor`, serializes it to a portable dictionary, deserializes it, and checks prediction parity within a strict tolerance.

```
import json
import numpy as np
import sklearn
from sklearn.datasets import make_regression
from sklearn.ensemble import RandomForestRegressor

from sklearn_migrator.regression.random_forest_reg import (
    serialize_random_forest_reg,
    deserialize_random_forest_reg,
)

X, y = make_regression(n_samples=200, n_features=10, random_state=0)
src_version = sklearn.__version__

src_model = RandomForestRegressor(
    n_estimators=50, random_state=0
).fit(X, y)

payload = serialize_random_forest_reg(src_model, version_in=src_version)

with open("model.json", "w") as f:
    json.dump(payload, f)

# --- In a different environment, load and deserialize ---
# In practice, version_out may differ.
tgt_version = sklearn.__version__
with open("model.json") as f:
    payload_loaded = json.load(f)

tgt_model = deserialize_random_forest_reg(payload_loaded, version_out=tgt_version)

y_src = src_model.predict(X)
y_tgt = tgt_model.predict(X)
assert np.max(np.abs(y_src - y_tgt)) < 1e-2
print("Prediction parity verified.")
```

Analogous functions exist for all covered estimators. In a true two-environment workflow, serialization runs in the source environment (e.g., 0.24.2) and deserialization in the target (e.g., 1.7.2).

Limitations

- **Two environments required.** End-to-end validation relies on two isolated environments. While this can be simulated on one machine, truly isolated setups (e.g., Docker images) are recommended to avoid dependency leakage.
- **Partial scikit-learn coverage.** Additional models (e.g., pipelines, transformers) are not yet supported but are planned for future releases. Community contributions are welcome.
- **Parity tolerance depends on model family.** Some families may be sensitive to floating-point or solver differences across versions; tolerances can be adjusted depending on operational requirements.
- **Parameter configuration coverage.** Each model is validated under a representative configuration. Exhaustive coverage of all parameter combinations and future breaking changes is outside the current scope.

AI Usage Disclosure

Large language models were used to assist with minor grammar checking and phrasing improvements during manuscript preparation. All software design decisions, implementation, experiments, validation, and technical content were authored and verified by the submitting author.

Acknowledgements

We thank the scikit-learn core developers and contributors for their open-source work and documentation, as well as the broader NumPy/SciPy/joblib ecosystems on which this project depends. We are grateful to colleagues and early adopters for testing and feedback, and to the JOSS editors and reviewers for guidance during the review process.

References

Fitzpatrick, P. C., & Manning, J. R. (2024). Davos: A Python package "smuggler" for constructing lightweight reproducible notebooks. *SoftwareX*, *25*, 101614. https://doi.org/10.1016/j.softx.2023.101614

Kaggle. (2021). *Kaggle's state of machine learning and data science 2021*. https://storage.googleapis.com/kaggle-media/surveys/Kaggle%27s%20State%20of%20Machine%20Learning%20and%20Data%20Science%202021.pdf.

Parida, S. K., Gerostathopoulos, I., & Bogner, J. (2025). *How do model export formats impact the development of ML-enabled systems? A case study on model integration*. https://doi.org/10.48550/arXiv.2502.00429

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

scikit-learn contributors. (2018). *Model persistence across versions — scikit-learn issue #10319*. https://github.com/scikit-learn/scikit-learn/issues/10319.

scikit-learn developers. (2025). *Model persistence*. https://scikit-learn.org/stable/model_persistence.html.