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.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 13 December 2025
Published: unpublished

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, n.d.; 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, n.d.). `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 (constructor parameters and other parity-relevant settings) that guarantee parity of outputs across versions; these are used to reconstruct an equivalent estimator in the target version and 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, n.d.).

Concretely, consider version `0.21.3` exposing `param_1, param_2, param_3, param_4` and version `1.7.0` exposing `param_1, param_2, param_3, param_5`. We use `param_1` and `param_2` to enforce identical predictions across versions. Since `param_3` exists in both versions, it is stored in the payload and reassigned on load. For version-specific attributes, the serializer records values for both `param_4` and `param_5`: in `0.21.3`, the payload stores the real value of `param_4` and the **default** value that `param_5` would take in newer versions; in `1.7.0`, it stores the real value of `param_5` and the default value that `param_4` would take in older versions. During deserialization, attributes are assigned using a simple `try/except` (or `hasattr`) so only valid attributes for the target version are set, and missing ones are safely skipped. This design makes migrations resilient to evolutionary changes such as renamed fields (e.g., `affinity` → `metric`), reorganized tree/boosting internals, and added default parameters across releases.

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

- **Classification (7):** `DecisionTreeClassifier`, `RandomForestClassifier`, `GradientBoostingClass` `LogisticRegression`, `KNeighborsClassifier`, `SVC`, `MLPClassifier`
- **Regression (10):** `DecisionTreeRegressor`, `RandomForestRegressor`, `GradientBoostingRegresso` `LinearRegression`, `Ridge`, `Lasso`, `KNeighborsRegressor`, `SVR`, `AdaBoostRegressor`, `MLPRegressor`
- **Clustering (3):** `AgglomerativeClustering`, `KMeans`, `MiniBatchKMeans`
- **Dimensionality reduction (1):** `PCA`

Collectively, these estimators represent a large fraction of classical ML model families used in practice and commonly reported in practitioner surveys and applied workflows (Kaggle, n.d.). `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, unit tests, and an MIT license support reproducibility and adoption in production MLOps workflows.

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. This is explicitly cautioned in the official documentation and has been observed in empirical and experiential studies (Fitzpatrick & Manning, 2024; Parida et al., 2025; scikit-learn developers, n.d.). 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 failure modes that are costly and hard to debug. 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—each with slightly different dependency constraints.

Given scikit-learn's broad adoption in research and industry, there is strong practical value in keeping legacy models usable across releases (Kaggle, n.d.; Pedregosa et al., 2011). The estimator families supported in this submission—tree ensembles, linear/logistic models, nearest neighbors, support vector machines, and MLPs—are among the most commonly taught, prototyped, and deployed methods in applied machine learning. Unsupervised components such as k-means clustering and PCA are similarly ubiquitous in feature engineering and exploratory analysis pipelines.

`sklearn-migrator` addresses this need with a two-stage, version-aware (de)serialization strategy. First, it captures the minimal, prediction-critical attributes to guarantee parity of outputs 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. This design aligns with modern reproducibility needs and with ecosystem efforts focused on lightweight, inspectable artifacts and reproducible computational environments (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 and has been exercised across 32 scikit-learn releases (`0.21.3 → 1.7.2`), covering **1,024** version pairs with unit tests and environment-isolated validation. This foundation reduces upgrade risk today while remaining extensible to additional estimators and components in future releases.

Design and validation

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 reconstruct an estimator that produces matching predictions under strict tolerance.
3. **Compatibility attributes**: additional learned attributes and internal fields stored with version-aware rules, including explicit defaults for fields that exist only in some versions.

To keep payloads portable, the library restricts values to JSON-encodable primitives (numbers, strings, booleans, lists, dicts) and encodes arrays using standard Python lists where necessary. This enables storage in plain JSON files, object storage, databases, or artifact registries, and supports inspection and debugging without executing arbitrary code (a common concern with pickle).

Validation methodology

`sklearn-migrator` validates migrations through:

- **Environment isolation** (e.g., containers) to ensure `version_in` and `version_out` represent real installations.
- **Fixed synthetic datasets** for deterministic evaluation.
- **Strict parity checks** comparing source and migrated predictions under a tolerance (e.g., `1e-4`).

The library has been tested across a full 32×32 version compatibility matrix, totaling **1,024 migration pairs**, and across all supported estimators. This automated validation provides confidence that the two-stage strategy behaves consistently across a large portion of the modern scikit-learn release history.

Example

The example below trains a `RandomForestRegressor`, serializes it to a portable (JSON-compatible) dictionary, deserializes it, and checks prediction parity within a strict tolerance. In practice, the `version_out` may correspond to a different scikit-learn installation (e.g., upgrading from 1.5.x to 1.7.x).

```python
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,
)

# Train in the "source" environment
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
```

```python
).fit(X, y)

# Serialize to a portable payload (JSON-compatible dict)
payload = serialize_random_forest_reg(src_model, version_in=src_version)

# (Optional) store as JSON
with open("model.json", "w") as f:
    json.dump(payload, f)

# --- In a different environment, load and deserialize ---
# (For illustration we reuse the same environment; 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)

# Prediction parity check
y_src = src_model.predict(X)
y_tgt = tgt_model.predict(X)
assert np.max(np.abs(y_src - y_tgt)) < 1e-4
print("Prediction parity verified.")
```

Analogous functions exist for all covered estimators (classification, regression, clustering, and dimensionality reduction). In a true two-environment workflow, the serialization code runs in the source environment (e.g., 0.24.2) and the deserialization code runs in the target environment (e.g., 1.7.2); the assertion above remains the same.

Limitations

- **Two environments required.** End-to-end validation relies on two isolated environments: a source environment (`version_in`) to serialize and a target environment (`version_out`) to deserialize and verify prediction parity. 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.** scikit-learn contains many estimators and pipeline components beyond the 21 currently supported. Additional models (e.g., pipelines, transformers, and further estimators) are not yet supported but are planned for future releases. We actively welcome community contributions to expand coverage. **Parity tolerance depends on model family.** Some model families may be sensitive to floating-point or solver differences across versions; the library uses strict tolerances by default but these can be adjusted depending on operational requirements.

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 that shaped the design, and to the JOSS editors and reviewers for guidance during the review process.

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