Lesson 41: Model Serialization

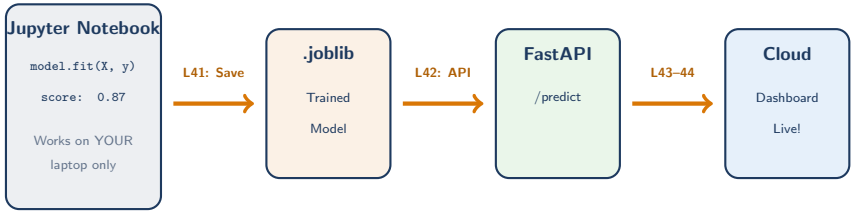
Data Science with Python – BSc Course

Data Science Program

BSc Course

45 Minutes

Module 9: From Notebook to Production



You built a model. Now: deploy it to the REAL WORLD.

Module 9 takes your trained model from a notebook to a live, deployed application.

L41: Save → L42: API → L43: Dashboard → L44: Cloud – the deployment pipeline

Learning Objectives

The Problem: You trained a model that took 2 hours to fit. You can't retrain it for every single prediction. How do you save it and load it later?

After this lesson, you will be able to:

- Save trained models with joblib and pickle
- Load saved models and make predictions instantly
- Compare serialization formats (joblib, pickle, ONNX)
- Prepare models for production deployment with metadata

Finance Application: Saving trained trading models so production systems can load them in milliseconds

Your Model Trains in 2 Hours...

The Problem is Real:

- Training a Random Forest on 10 years of stock data: **2 hours**
- Loading a saved model from disk: **0.3 seconds**
- A trading system needs predictions in **milliseconds**
- You can't retrain every time the market opens

Analogy – Saving a Video Game:

- Training = playing through a hard level (hours of effort)
- Serialization = saving your game progress to a file
- Deserialization = loading the save – right where you left off
- Without saving, you'd replay (retrain) every single time

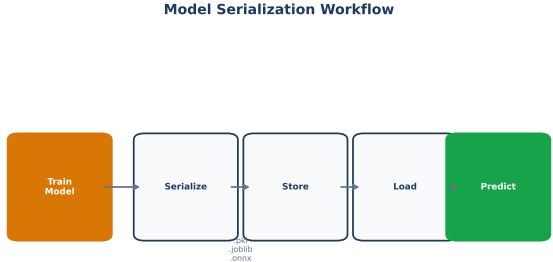
Serialization = convert a Python object to a file. Deserialization = load it back.

Train once, predict forever. That's the power of model serialization.

The Serialization Workflow

From Training to Prediction

- Step 1: Train model + preprocessing pipeline
- Step 2: Serialize (save) to disk as a file
- Step 3: Deserialize (load) on production server



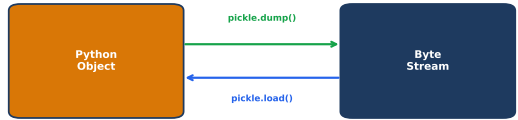
Save the ENTIRE pipeline (scaler + model), not just the model alone

pickle: Python's Built-In Serializer

The Universal Python Serializer

- Works for ANY Python object – not just ML models
- No extra installation needed (standard library)
- But slower than joblib for large numpy arrays

Pickle: Serialize Python Objects



pickle: universal but slower for ML. Never load pickle files from untrusted sources!

pickle: Save and Load

Save:

```
import pickle
with open('model.pkl', 'wb') as f:
  pickle.dump(model, f)
```

Load:

```
with open('model.pkl', 'rb') as f:
  model = pickle.load(f)
predictions = model.predict(X_new)
```

Security Warning:

- Never load pickle files from untrusted sources!
- Pickle can execute **arbitrary code** during loading
- A malicious .pkl file could delete your files or steal credentials
- Only load pickles YOU created or from trusted colleagues

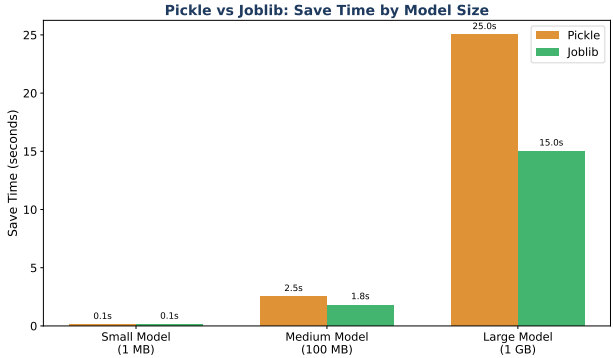
---

pickle.dump() to save, pickle.load() to load. Watch out for security!

joblib vs pickle: Which One?

joblib: The sklearn Recommended Choice

- Optimized for large numpy arrays (10–100x faster than pickle)
- Built-in compression support (`compress=3`)
- The standard for sklearn pipelines



Rule of thumb: sklearn models → joblib. General Python objects → pickle.

joblib: Save and Load

Save (simple):

- `import joblib`
- `joblib.dump(model, 'model.joblib')`

Save (with compression):

- `joblib.dump(model, 'model.joblib', compress=3)`
- Compress level 1–9: higher = smaller file, slower save

Load:

- `model = joblib.load('model.joblib')`
- `predictions = model.predict(X_new)` – ready immediately

Best Practice: Save the Pipeline

- `from sklearn.pipeline import Pipeline`
- `pipe = Pipeline([('scaler', StandardScaler()), ('rf', RandomForestClassifier())])`
- `pipe.fit(X_train, y_train)`
- `joblib.dump(pipe, 'pipeline.joblib')` – scaler + model together

Checkpoint: pickle vs joblib

Quick Check – Which would you use?

1. Saving a trained sklearn RandomForest to disk?
2. Saving a custom Python dictionary of configuration?
3. Deploying a model that includes StandardScaler + PCA + LogisticRegression?
4. Sharing a model file with a colleague via email?

Answers:

1. **joblib** – optimized for sklearn + numpy arrays
2. **pickle** – general Python objects, no numpy advantage
3. **joblib** – save the entire Pipeline object in one file
4. **Either** – but warn about pickle security if untrusted source

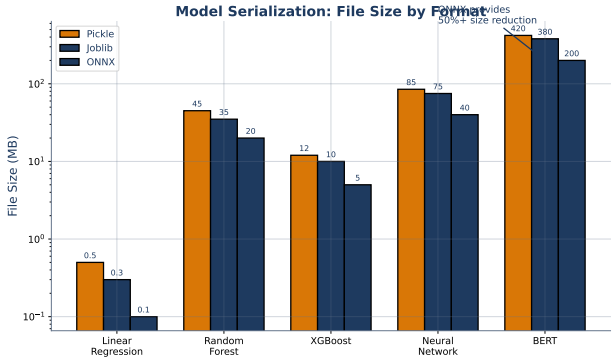
---

joblib for ML models, pickle for general Python. Both save Python objects to files.

File Size Comparison

How Format Affects File Size

- ONNX offers best compression for deployment
- joblib with compress=3 significantly reduces file size
- Format choice depends on: speed vs size vs compatibility

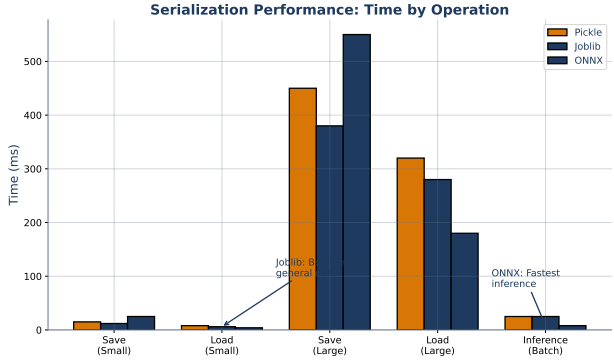


Large models benefit most from format optimization

Load Time Performance

How Fast Can We Load Each Format?

- joblib is fastest for loading sklearn models
- ONNX slower to save but faster at inference time
- For production APIs: consider ONNX for inference speed



ONNX: slower save, faster inference – best for high-throughput production APIs

ONNX: The Universal Model Format

Open Neural Network Exchange

- Framework-agnostic: train in Python, deploy **anywhere**
- Faster inference than native Python serialization
- Supported by sklearn, PyTorch, TensorFlow, and more

When to Use ONNX:

- Production APIs where inference speed matters
- Cross-platform: Python $\rightarrow$ C++, Java, JavaScript, mobile
- Edge deployment (embedded devices, IoT)

When NOT to Use:

- Quick prototyping (joblib is simpler)
- Models with custom transformers ONNX can't convert
- Frequent retraining (the export step adds overhead)

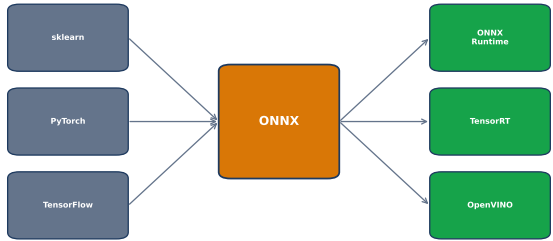
ONNX: train in Python, deploy anywhere. Best for production inference speed.

ONNX Conversion Workflow

From sklearn to ONNX in 3 Steps

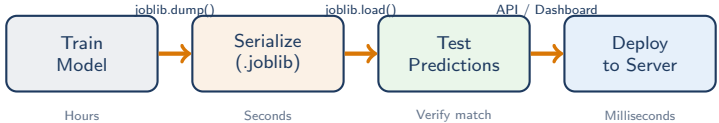
- Convert trained model to ONNX format
- Validate the conversion produces identical outputs
- Deploy the .onnx file with ONNX Runtime

ONNX: Universal Model Format



ONNX Runtime provides 2–10x speedup over native Python inference

The Deployment Pipeline



Critical Step: Always verify that the loaded model produces **identical** predictions to the original trained model before deploying.

Train (hours) → Save (seconds) → Test (verify) → Deploy (milliseconds)

Saving Metadata with Models

What to Save Alongside Your Model

1. **Feature names:** Which columns does the model expect?
2. **Training date:** When was this model trained?
3. **Performance:** CV score, test accuracy, F1
4. **sklearn version:** Models may not load across versions
5. **Training data hash:** For reproducibility audits

Save as Dictionary:

```
artifact = {'model': pipeline, 'features': feature_list,
  'cv_score': 0.85, 'trained_at': '2024-01-15',
  'sklearn_version': sklearn.__version__}
joblib.dump(artifact, 'model_v1.joblib')
```

Loading:

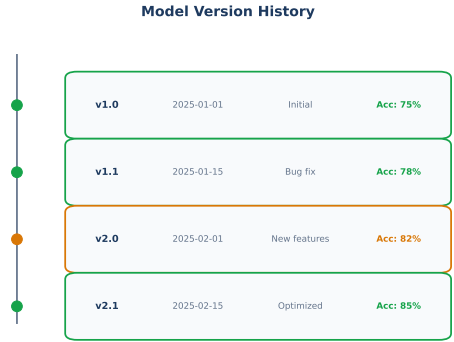
```
artifact = joblib.load('model_v1.joblib')
model = artifact['model']; features = artifact['features']
```

Always save metadata with your model – you'll thank yourself later

Model Versioning Best Practices

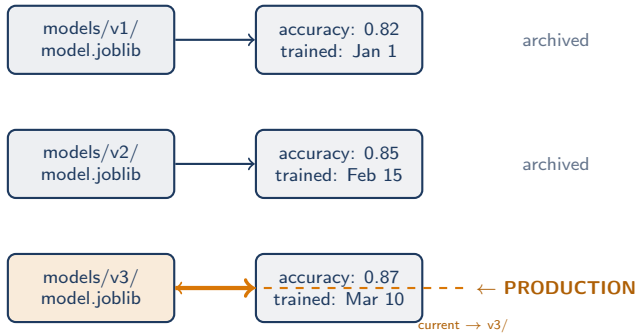
Managing Multiple Model Versions

- Include version + date in filename: `model_v2_2024-01-15.joblib`
- Keep a model registry (simple CSV or database)
- Never overwrite production model without testing



Version your models like you version your code — always be able to rollback

A Simple Model Registry



Finance: Regulatory compliance requires reproducing past predictions – version everything.

---

models/current symlink points to active version. Each version has metadata.json.

Security: The Pickle Danger

Why Pickle is Dangerous:

- Pickle executes **arbitrary Python code** during `load()`
- A malicious .pkl file can run ANY command on your machine
- Real attack: model file on GitHub contains hidden malware

Safe Practices:

1. **Never** load .pkl/.joblib from untrusted sources
2. Verify file hashes before loading shared models
3. Use ONNX for cross-organization model sharing
4. Consider `safetensors` format (no code execution)
5. Run model loading in sandboxed environments

Finance Context: A compromised model file in a trading system could execute unauthorized trades or exfiltrate sensitive data.

Pickle = powerful but dangerous. Only load files you trust completely.

Finance: Saving Trained Trading Models

Real-World Serialization in Finance:

- Train credit scoring model on historical data (overnight batch)
- Save model + scaler + feature list as single artifact
- Production system loads model at startup (0.3s)
- Serves predictions to loan application system (5ms per request)

Regulatory Requirements:

- **Audit trail:** Log which model version made each prediction
- **Reproducibility:** Must recreate any past decision
- **Model governance:** Approval required before production deployment
- **Version retention:** Keep all model versions for 7+ years

Key Insight: In finance, model serialization isn't just convenient – it's **legally required**.

Financial regulators require model version tracking and prediction reproducibility

Serialization Best Practices

Do:

- Save entire pipeline (preprocessing + model) as one object
- Include metadata: features, version, scores, training date
- Use joblib for sklearn models, ONNX for production inference
- Verify loaded model produces identical predictions
- Version every model with date and performance metrics

Don't:

- Save model and scaler separately (easy to mismatch)
- Overwrite production models without testing
- Load pickle files from untrusted sources
- Forget to pin your sklearn version in requirements
- Ship models without metadata – you'll lose track

One file, one pipeline, one metadata dict. Keep it simple and versioned.

Hands-On Exercise (25 min)

Task: Save and Load a Complete ML Pipeline

1. Train a pipeline: StandardScaler $\rightarrow$ PCA $\rightarrow$ RandomForestClassifier
2. Save with joblib (with and without compression)
3. Compare file sizes: joblib vs pickle vs compressed
4. Load the model and verify predictions match exactly
5. Save metadata dictionary alongside the model

Verification:

- `assert np.array_equal(original_preds, loaded_preds)`
- Compare file sizes in a table: format, size, load time

Deliverable: File size comparison table + verification that loaded model produces identical predictions.

Extension: Convert to ONNX format and compare inference speed

The Journey So Far

Before L41

"My model works!"

"Let me retrain it. . ."

2 hours later. . .

"OK now I can predict"

(repeat daily)

After L41

"My model works!"

joblib.dump() ✓

joblib.load() – 0.3s

"Predictions ready!"

(every time, instantly)

Your model is saved. Next step: make it accessible to the world via an API.

Serialization turns a one-time experiment into a reusable production asset

Lesson Summary

Problem Solved: We can now save trained models to disk and load them for deployment without retraining.

Key Takeaways:

- **joblib:** best for sklearn models – fast, compression support
- **pickle:** universal Python serializer – but slower for ML
- **ONNX:** framework-agnostic, best for production inference
- Always save metadata (features, version, scores) with your model
- Security: never load pickle files from untrusted sources

Next Lesson: FastAPI (L42) – your saved model goes online as a web API

L41: Save the model. L42: Serve it. L43: Show it. L44: Deploy it.