Train in Python, deploy anywhere and anyhow

Christian Bourjau

July 26, 2023

QuantCo
PyData Zurich

quantco

Outline

What makes deploying models difficult?

What is ONNX?

Expressing models in Spox

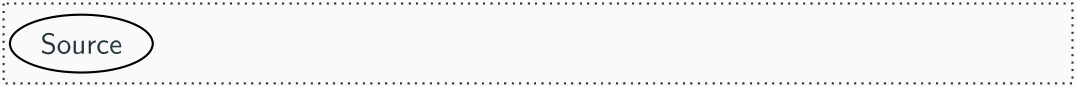
Summary

What makes deploying models difficult?

Rapid model development and (?) stable deployments

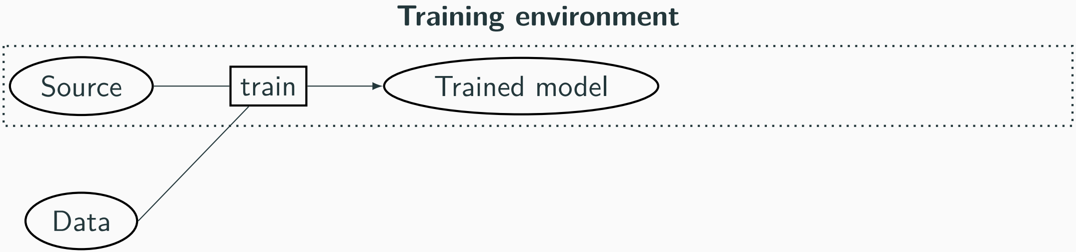
- Define pipeline with feature engineering: Python rocks!

Training environment



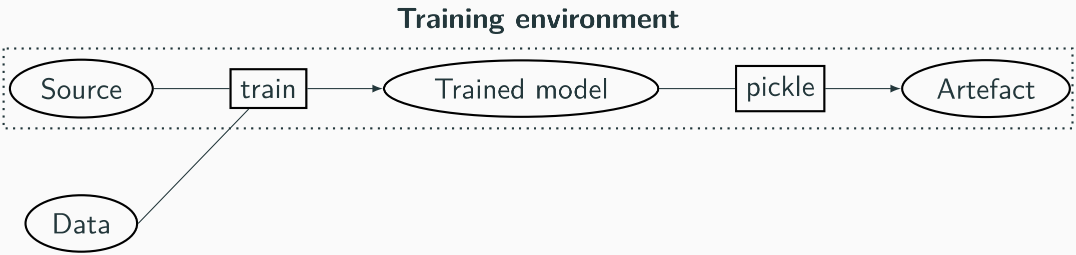
Rapid model development and (?) stable deployments

- Define pipeline with feature engineering: Python rocks!
- Train: Python rocks!



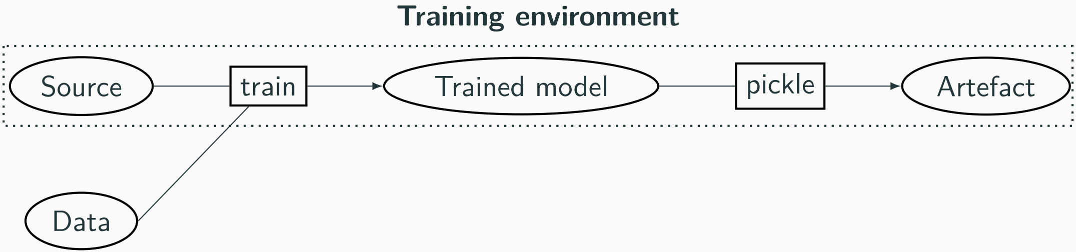
Rapid model development and (?) stable deployments

- Define pipeline with feature engineering: Python rocks!
- Train: Python rocks!
- Pickle the trained model: Python rocks?



Rapid model development and (?) stable deployments

- Define pipeline with feature engineering: Python rocks!
- Train: Python rocks!
- Pickle the trained model: Python rocks?
- Deploy ...



Issues in deployment

- Pickle makes it hard to:
  - Distribute models

Issues in deployment

- Pickle makes it hard to:
  - Distribute models
  - Archive models
  - Compose models from different sources
  - Keep training IP out of the distributed model

Issues in deployment

- Pickle makes it hard to:
  - Distribute models
  - Archive models
  - Compose models from different sources
  - Keep training IP out of the distributed model
- Insufficient performance:
  - Slow and single-threaded

Issues in deployment

- Pickle makes it hard to:
  - Distribute models
  - Archive models
  - Compose models from different sources
  - Keep training IP out of the distributed model
- Insufficient performance:
  - Slow and single-threaded
- No Python on deployment platform:
  - Deploy on the edge (e.g. mobile)

Issues in deployment

- Pickle makes it hard to:
  - Distribute models
  - Archive models
  - Compose models from different sources
  - Keep training IP out of the distributed model
- Insufficient performance:
  - Slow and single-threaded
- No Python on deployment platform:
  - Deploy on the edge (e.g. mobile)

Why is this so hard?

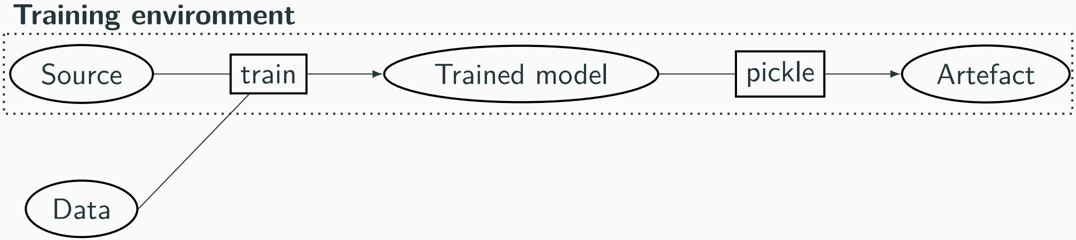
Can we do better?

Non-ML software

Can we do better?

Non-ML software

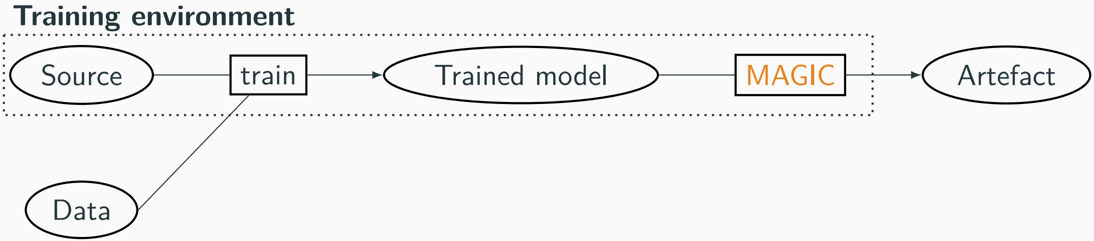
What we have



Can we do better?

Non-ML software

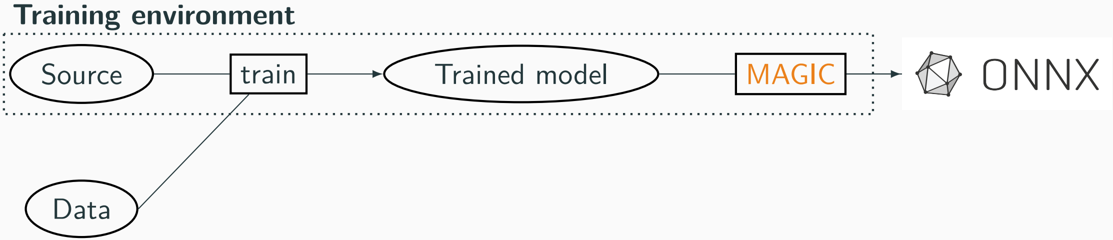
What we **want**



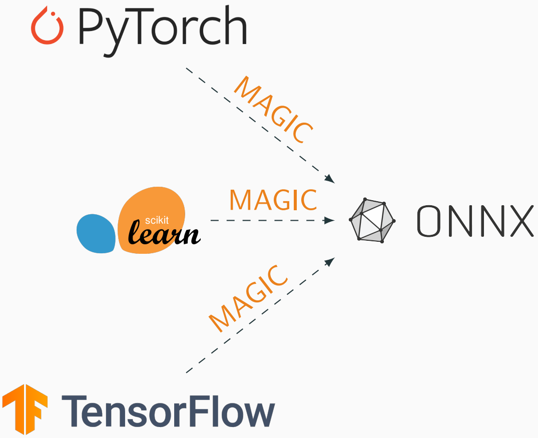
Can we do better?

Non-ML software

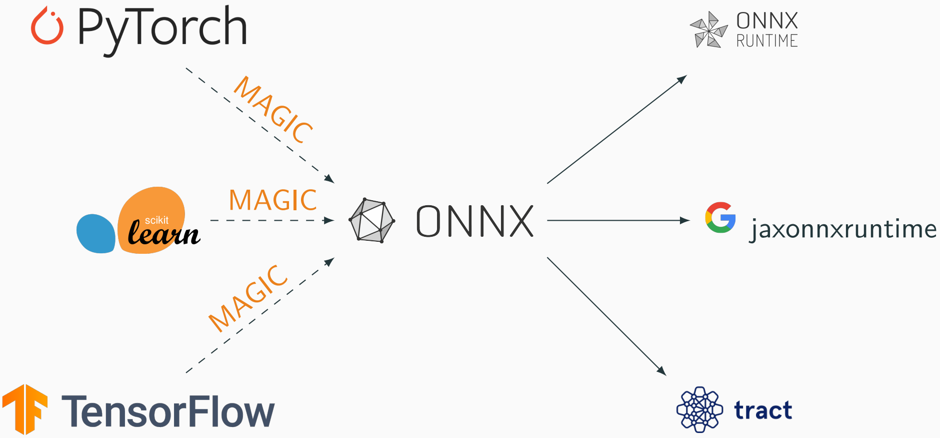
What we want



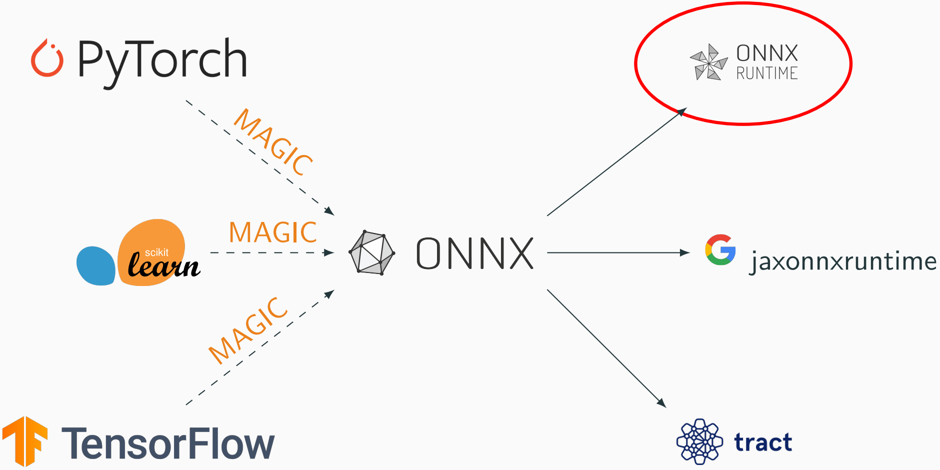
Ecosystem



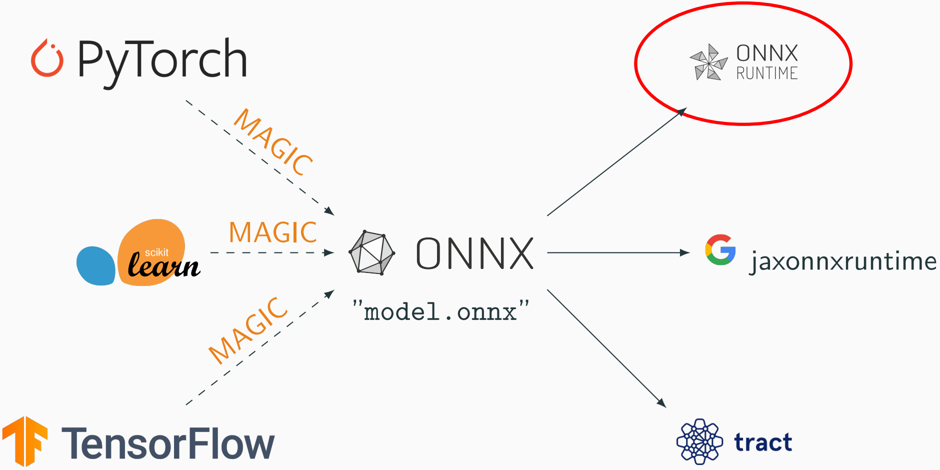
Ecosystem


PyTorch
MAGIC
scikit learn
MAGIC
MAGIC
TensorFlow
ONNX
ONNX RUNTIME
jaxonnxruntime
tract

Ecosystem


PyTorch
MAGIC
scikit learn
MAGIC
TensorFlow
MAGIC
ONNX
ONNX RUNTIME
jaxonnxruntime
tract

Ecosystem


PyTorch
MAGIC
scikit learn
MAGIC
MAGIC
TensorFlow
ONNX
"model.onnx"
ONNX RUNTIME
jaxonnxruntime
tract

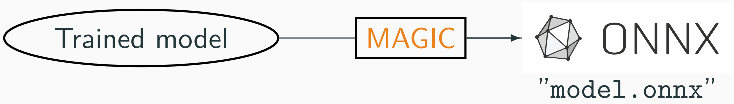
Deployment with ONNX

```python
import numpy as np
import onnxruntime

# Load the model into the runtime
session = onnxruntime.InferenceSession("model.onnx")

# Execute the model
def predict(**inputs: nd.ndarray) -> list[nd.ndarray]:
    return session.run(None, inputs)
```

About that magical step...



Serialise inference logic and weights

What is ONNX?

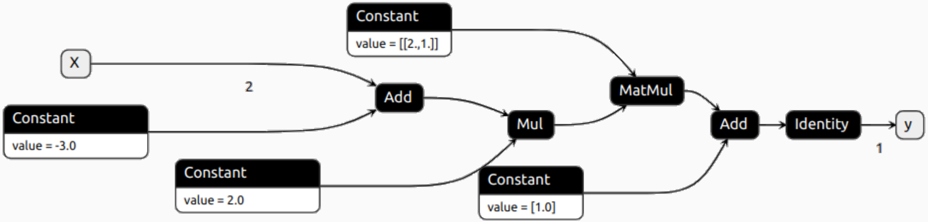
What is ONNX?

ONNX is a standard

What is ONNX?

ONNX is a standard

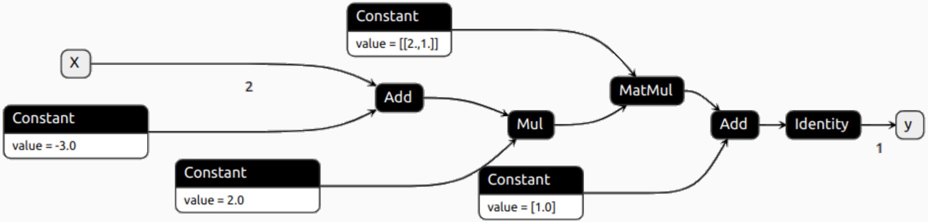
- Strongly typed computational DAG
  - Edges are Tensors
  - Nodes are operators
  - Nodes store state as attributes



What is ONNX?

ONNX is a standard

- Strongly typed computational DAG
  - Edges are Tensors
  - Nodes are operators
  - Nodes store state as attributes
- Set of standardized operators (∼180)



Creating ONNX graphs – with a beautiful abstraction

- ONNX is a tensor library API
- Spox[1] exposes that API as Python library

[1] github.com/Quantco/spox

Expressing models in Spox

Linear regression in NumPy

```python
from numpy import ndarray
import numpy as np

def lin_reg(X: ndarray, coef, intercept) -> np.ndarray:
    return coef @ X + intercept
```

Linear regression in NumPy

```python
from numpy import ndarray
import numpy as np

def lin_reg(X: ndarray, coef, intercept) -> np.ndarray:
    return coef @ X + intercept


def lin_reg_verbose(X: ndarray, coef, intercept) -> np.ndarray:
    return np.add(np.matmul(coef, X), intercept)
```

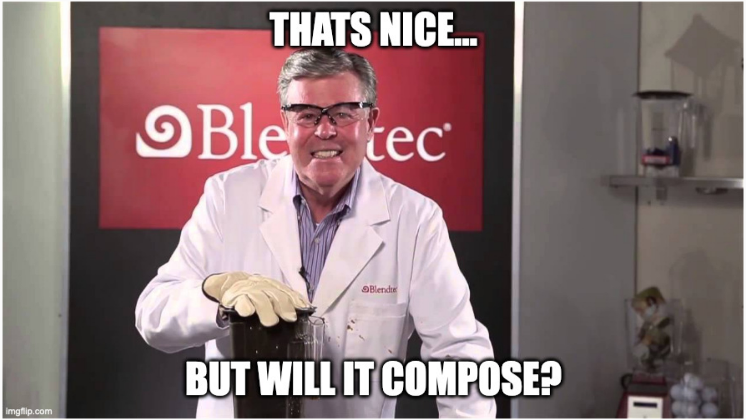
Linear regression in Spox/ONNX

```python
from spox import Var
import spox.opset.ai.onnx.v18 as op
```

Linear regression in Spox/ONNX

```python
from spox import Var
import spox.opset.ai.onnx.v18 as op


def lin_reg(X: Var, coef, intercept) -> Var:
    # Move state into constants
    coef = op.const(coef)
    intercept = op.const(intercept)
    return op.add(op.matmul(coef, X), intercept)
```


THATS NICE...
Blendtec
Blendtec
BUT WILL IT COMPOSE?
imgflip.com

Expressing sklearn-like pipelines in Spox

```python
class SpoxLinearRegression:
    def __init__(self, state: LinearRegression):
        self.state = state

    def predict(self, X: Var) -> Var:
        return ...
```

Expressing sklearn-like pipelines in Spox

```python
class SpoxLinearRegression:
    def __init__(self, state: LinearRegression):
        self.state = state

    def predict(self, X: Var) -> Var:
        coef = op.const(self.state.coef)
        intercept = op.const(self.state.intercept)
        return op.add(op.matmul(coef, X), intercept)
```

Composing converters in a Pipeline

```python
def converter(model):
    if isinstance(model, LinearRegression):
        return SpoxLinearRegression(model)
    ...
```

Composing converters in a Pipeline

```python
def converter(model):
    if isinstance(model, LinearRegression):
        return SpoxLinearRegression(model)
    ...

class SpoxPipeline:
    ...
    def predict(self, X: Var) -> Var:
        for step in self.model.steps[:-1]:
            X = converter(step).transform(X)
        last = self.model.steps[-1]
        return converter(last).predict(X)
```

Building the model

```
pipe = Pipeline([("scaler", MinMaxScaler()),
                 ("linear", LinearRegression(...))])
pipe.fit(X_train, y_train)
...
```

Building the model

```
pipe = Pipeline([("scaler", MinMaxScaler()),
                 ("linear", LinearRegression(...))])
pipe.fit(X_train, y_train)
...

import numpy as np
from spox import argument, build, Tensor
from your_converter_library import converter

X = argument(Tensor(np.float64, ("N",)))
y = converter(pipe).predict(X)
model = build({"X": X}, {"y": y})
onnx.save(model, "model.onnx")
```

Next steps

Spox/ONNX is just another **tensor library**

- **Spox**

[1] data-apis.org/

Next steps

Spox/ONNX is just another **tensor library**

- **Spox**
- NumPy

[1] data-apis.org/

Next steps

Spox/ONNX is just another **tensor library**

- **Spox**
- NumPy
- `jax.numpy`

[1] data-apis.org/

Next steps

Spox/ONNX is just another **tensor library**

- **Spox**
- NumPy
- `jax.numpy`
- `tf.experimental.numpy`

[1] data-apis.org/

Next steps

Spox/ONNX is just another **tensor library**

- **Spox**
- NumPy
- `jax.numpy`
- `tf.experimental.numpy`
- Dask

[1] data-apis.org/

Next steps

Spox/ONNX is just another **tensor library**

- **Spox**
- NumPy
- `jax.numpy`
- `tf.experimental.numpy`
- Dask
- PyTorch

[1] data-apis.org/

Next steps

Spox/ONNX is just another **tensor library**

- **Spox**
- NumPy
- `jax.numpy`
- `tf.experimental.numpy`
- Dask
- PyTorch
- CuPy

[1] data-apis.org/

Next steps

Spox/ONNX is just another **tensor library**

- **Spox**
- NumPy
- `jax.numpy`
- `tf.experimental.numpy`
- Dask
- PyTorch
- CuPy
- ...

[1] data-apis.org/

Next steps

Spox/ONNX is just another **tensor library**

- **Spox**
- NumPy
- `jax.numpy`
- `tf.experimental.numpy`
- Dask
- PyTorch
- CuPy
- ...

DataAPIs

[1] data-apis.org/

Next steps

Spox/ONNX is just another **tensor library**

- **Spox**
- NumPy
- `jax.numpy`
- `tf.experimental.numpy`
- Dask
- PyTorch
- CuPy
- ...

DataAPIs ❤️ scikit-learn

[1] data-apis.org/

Summary

Summary

Spox ❤️ ONNX:

- Compile your trained model to a self-contained artefact

Summary

Spox ❤️ ONNX:

- Compile your trained model to a self-contained artefact
- Stable, very fast, and maintainable deployment

Summary

Spox ❤️ ONNX:

- Compile your trained model to a self-contained artefact
- Stable, very fast, and maintainable deployment
- Ready for production use!

Summary

Spox ❤️ ONNX:

- Compile your trained model to a self-contained artefact
- Stable, very fast, and maintainable deployment
- Ready for production use!
- ONNX has corporate backing by Microsoft and others

Summary

Spox ❤️ ONNX:

- Compile your trained model to a self-contained artefact
- Stable, very fast, and maintainable deployment
- Ready for production use!
- ONNX has corporate backing by Microsoft and others
- It is a key technology at **QuantCo**

Summary

Spox ❤️ ONNX:

- Compile your trained model to a self-contained artefact
- Stable, very fast, and maintainable deployment
- Ready for production use!
- ONNX has corporate backing by Microsoft and others
- It is a key technology at **QuantCo**

Tutorials and Docs: `github.com/Quantco/spox`

Appendix

Integrating existing converters

Spox also allows integration with existing converter libraries by way of `inline`.

```
class SpoxComplicatedRegressor:
    ...

    def predict(X):
        onnx_model = skl2onnx.to_onnx(self.model, ...)
        (y,) = inline(onnx_model)(X)
        return y
```