“Deployment for free”: removing the need to write model deployment code at Stitch Fix

Stanford CS329S February 2022

Stefan Krawczyk

@stefkrawczyk
linkedin.com/in/skrawczyk

Try out Stitch Fix → goo.gl/Q3tCQ3

#CS329S #MLOps #machinelearning

STITCH FIX

> Stitch Fix

"Deployment for free"

Model Envelope & envelope mechanics

Impact of being on-call

Summary & Future Work

Stitch Fix is a personal styling service

Key points:

1. Very algorithmically driven company
2. Single DS Department: Algorithms (135+)
3. "Full Stack Data Science"
    a. No reimplementation handoff
    b. End to end ownership
    c. Built on top of data platform tools & abstractions.

For more information: https://algorithms-tour.stitchfix.com/ & https://cultivating-algos.stitchfix.com/

Where do I fit in?


Pre-covid look

Stefan Krawczyk

Mgr. Data Platform - Model Lifecycle

MSCS’10

STITCH FIX

Checkout out our open source dataflow library that helps manage feature/workflow code for you:

- https://github.com/stitchfix/hamilton/

Stitch Fix

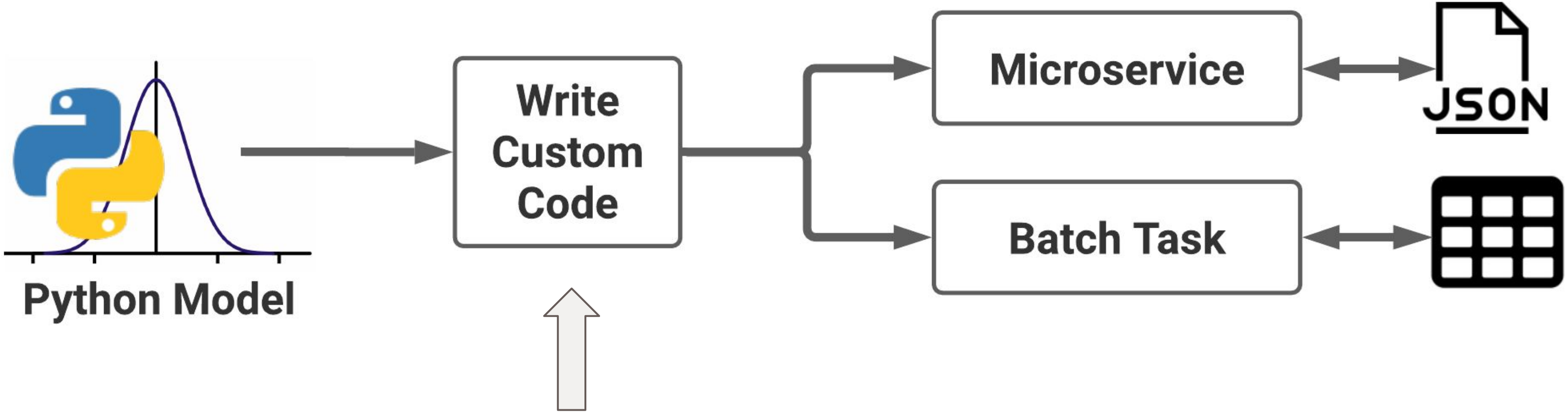
> "Deployment for free"

Model Envelope & envelope mechanics

Impact of being on-call

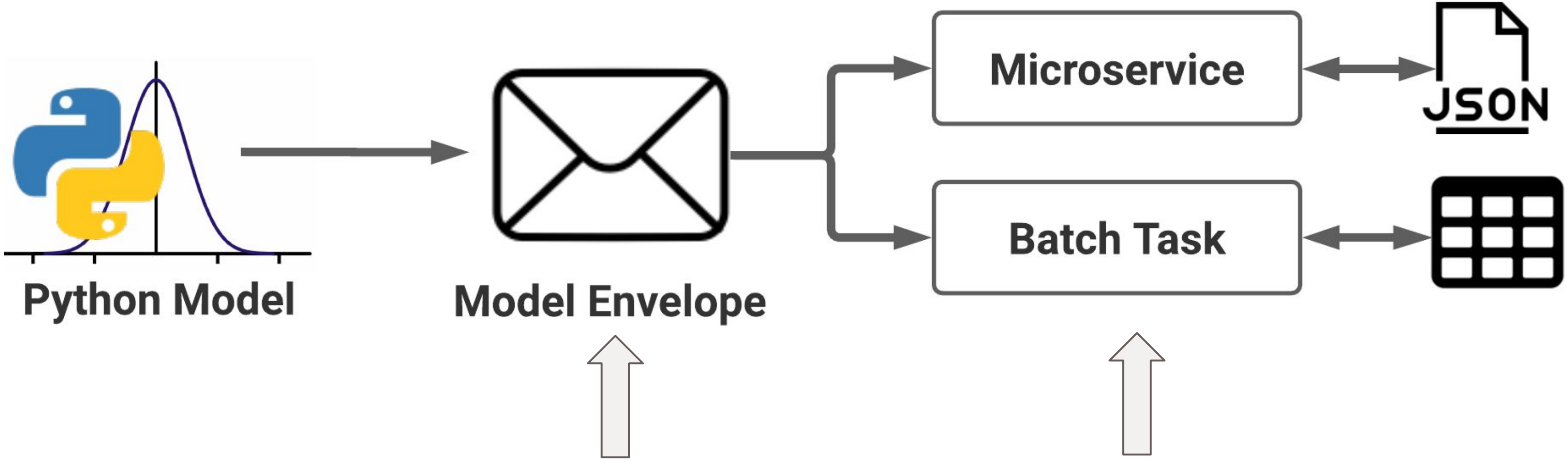
Summary & Future Work

Typical Model Deployment Process



- Many ways to approach.
- Heavily impacts MLOps.

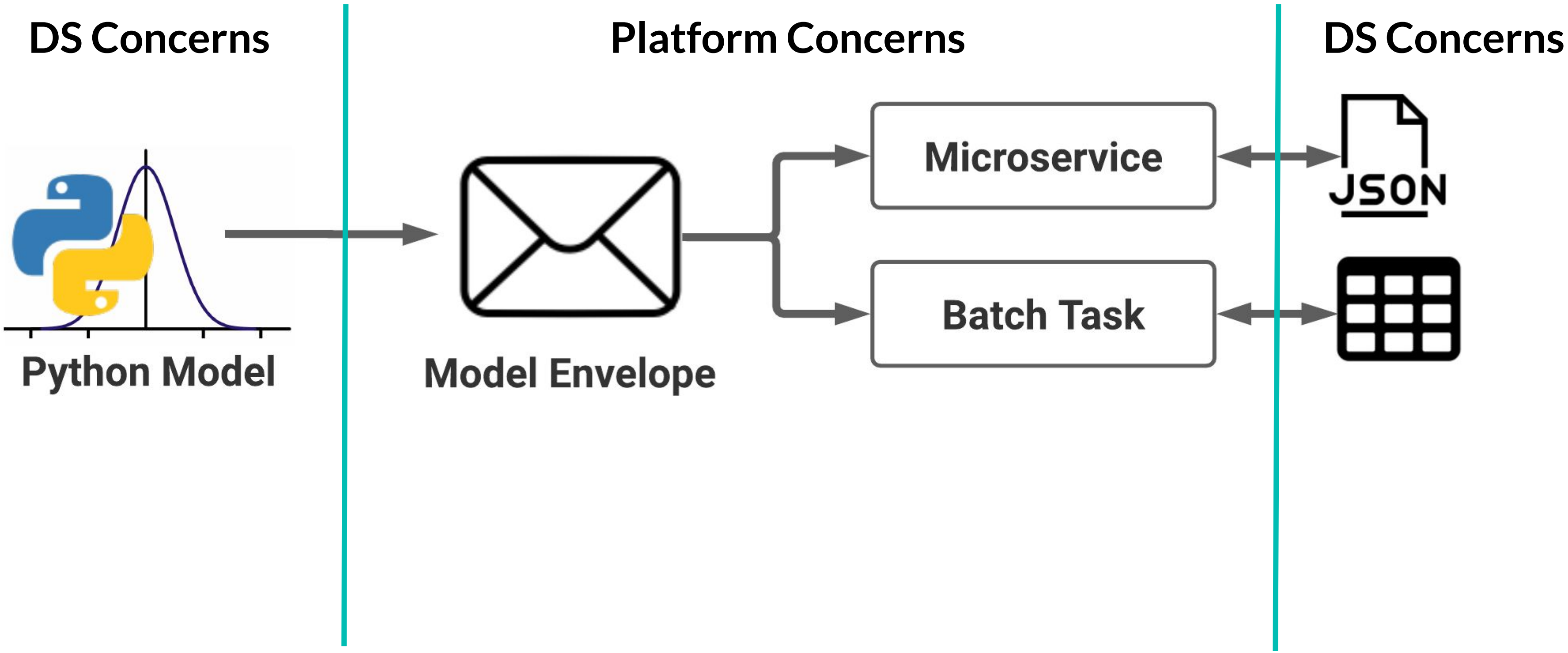
Model Deployment at Stitch Fix



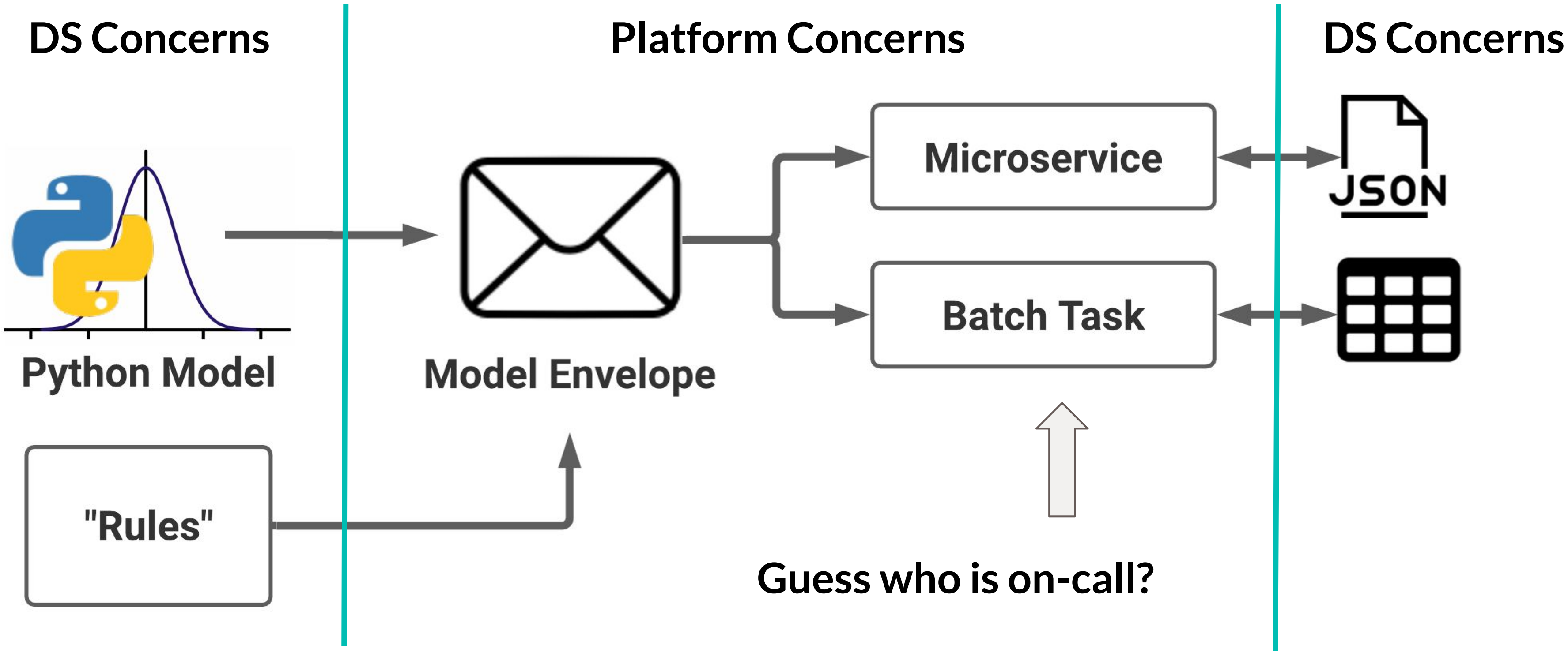
Once a model is in an envelope...

This comes for free!

Who owns what?

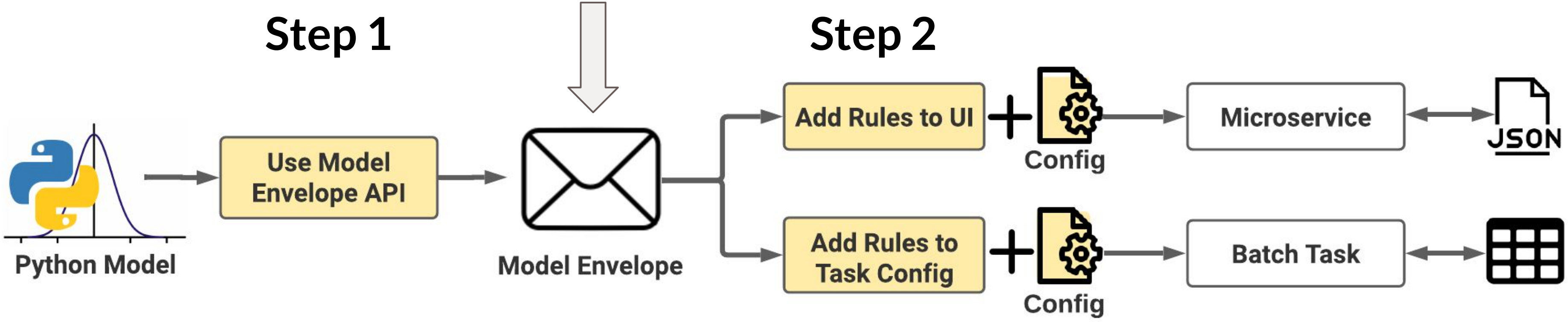


Deployments are “triggered”



Reality: two steps to get a model to production

Can be a terminal point.



Self-service: takes <1 hour
No code is written!

Step 1. save a model via Model Envelope API

etl.py

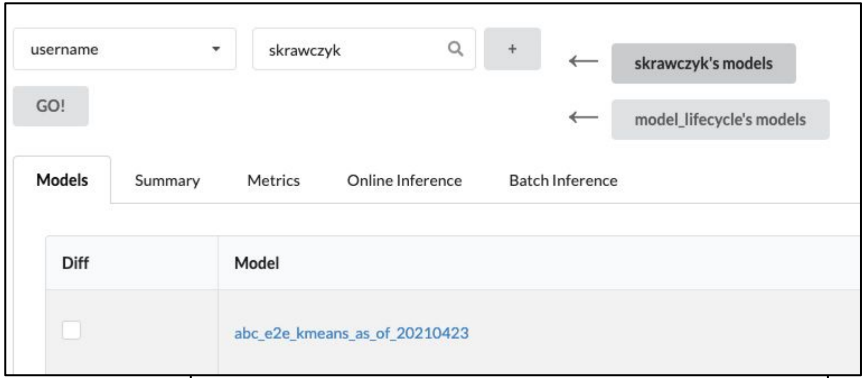
```
import model_envelope as me
from sklearn import linear_model

df_X, df_y = load_data_somehow()
model = linear_model.LogisticRegression(multi_class='auto')
model.fit(df_X, df_y)

my_envelope = me.save_model(instance_name='my_model_instance_name',
                            instance_description='my_model_instance_description',
                            model=model,
                            query_function='predict',
                            api_input=df_X, api_output=df_y,
                            tags={'canonical_name':'foo-bar'})
```

Note: no deployment trigger in ETL code.

Step 2a. deploy model as a microservice

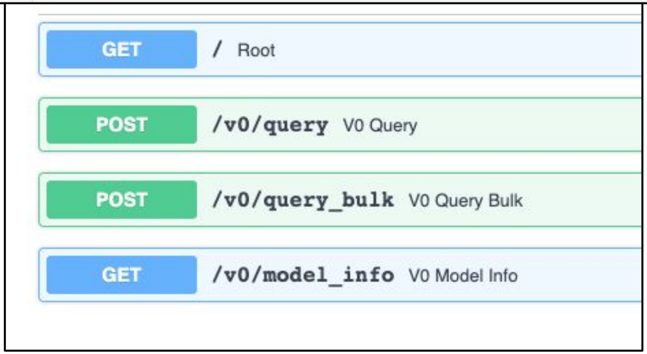
Go to Model Envelope Registry UI:

1) Create deployment configuration.
2) Create **Rule** for auto deployment.
   a) Else query for model & hit deploy.
3) Done.



Result:

- Web service with API endpoints
  - Comes with a Swagger UI & schema →
- Model in production < 1 hour.



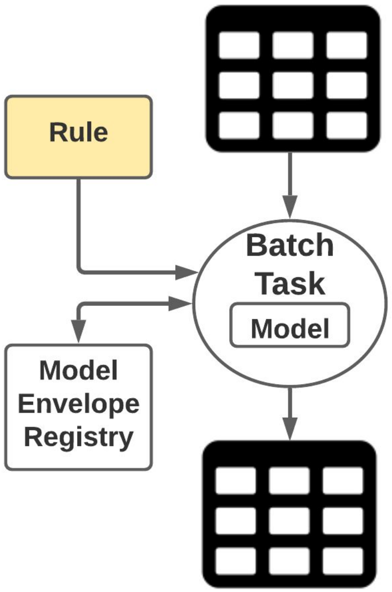
Step 2b. deploy model as a batch task

Create workflow configuration:

1) Create batch inference task in workflow.
    a) Specify **Rule** & inputs + outputs.
2) Deploy workflow.
3) Done.

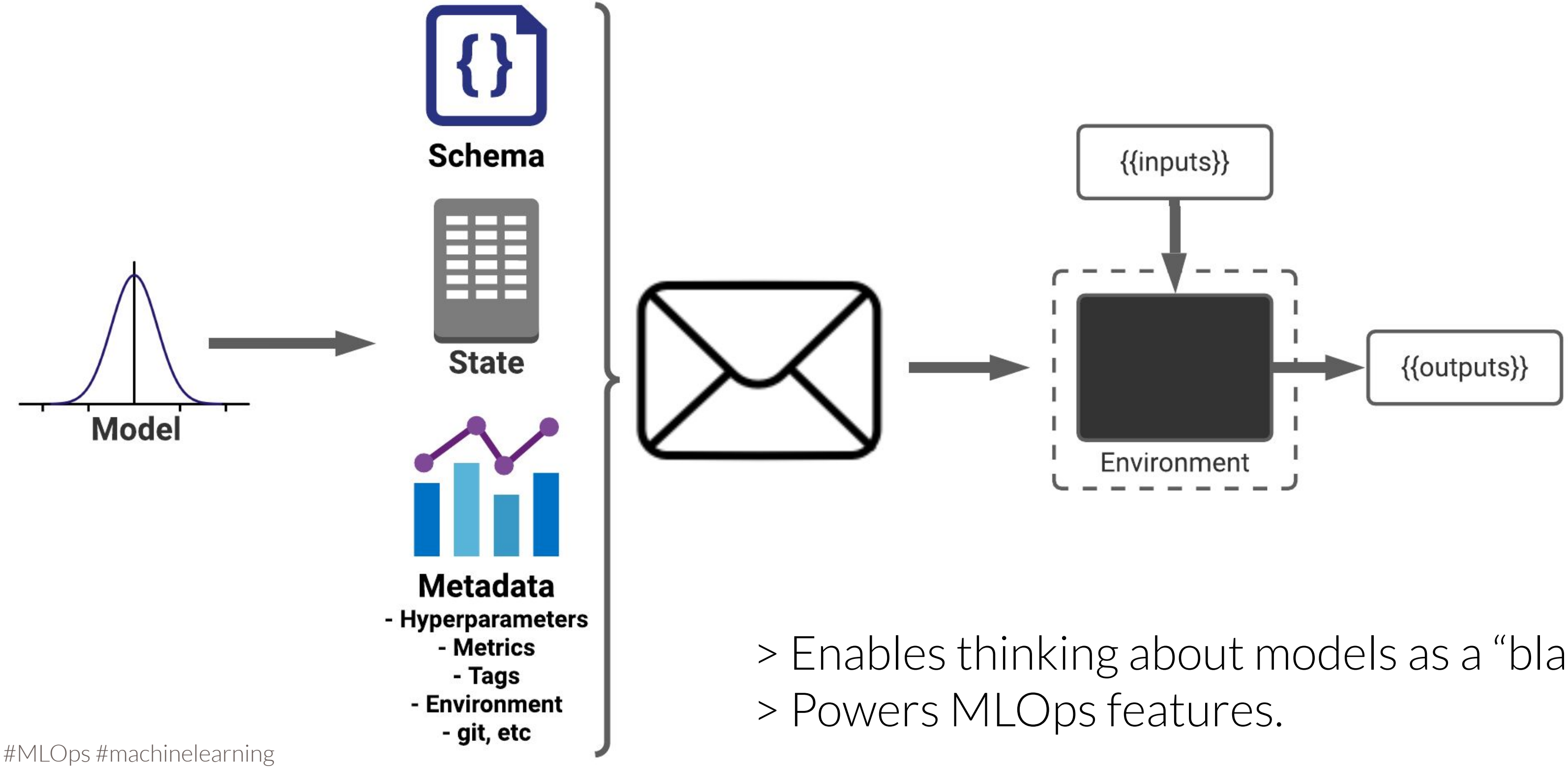
Result:

- Spark or Python task that creates a table.
- We keep an inference log.
- Model in production < 1 hour.



Stitch Fix
"Deployment for free"
> Model Envelope & envelope mechanics
Impact of being on-call
Summary & Future Work

Q: What is the Model Envelope? A: It's a container.



> Enables thinking about models as a "black box".
> Powers MLOps features.

🤔 Wait this feels familiar?

You: “MLFlow/Verta much?”
Me: Yes & No.

This is all internal code -- nothing from open source.

In terms of functionality we’re closer to a mix of:

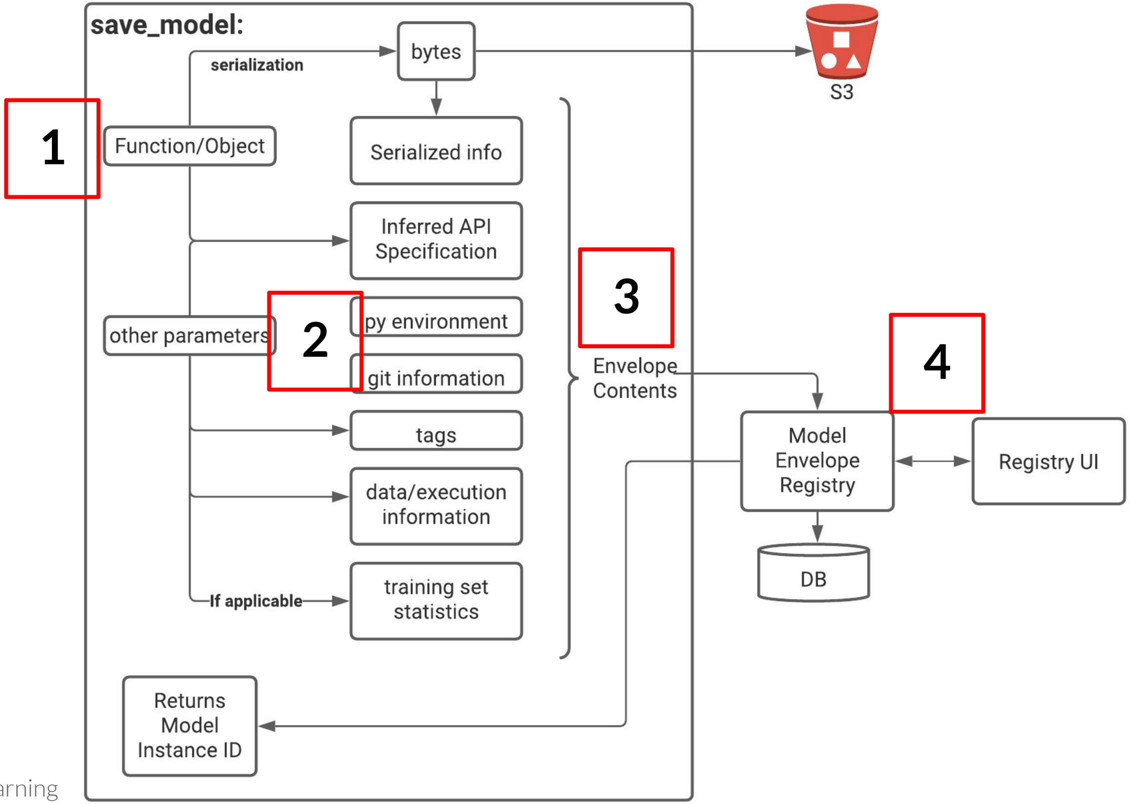
- MLFlow | Verta.ai
- ModelDB
- TFX

But this talk is too short to cover everything...

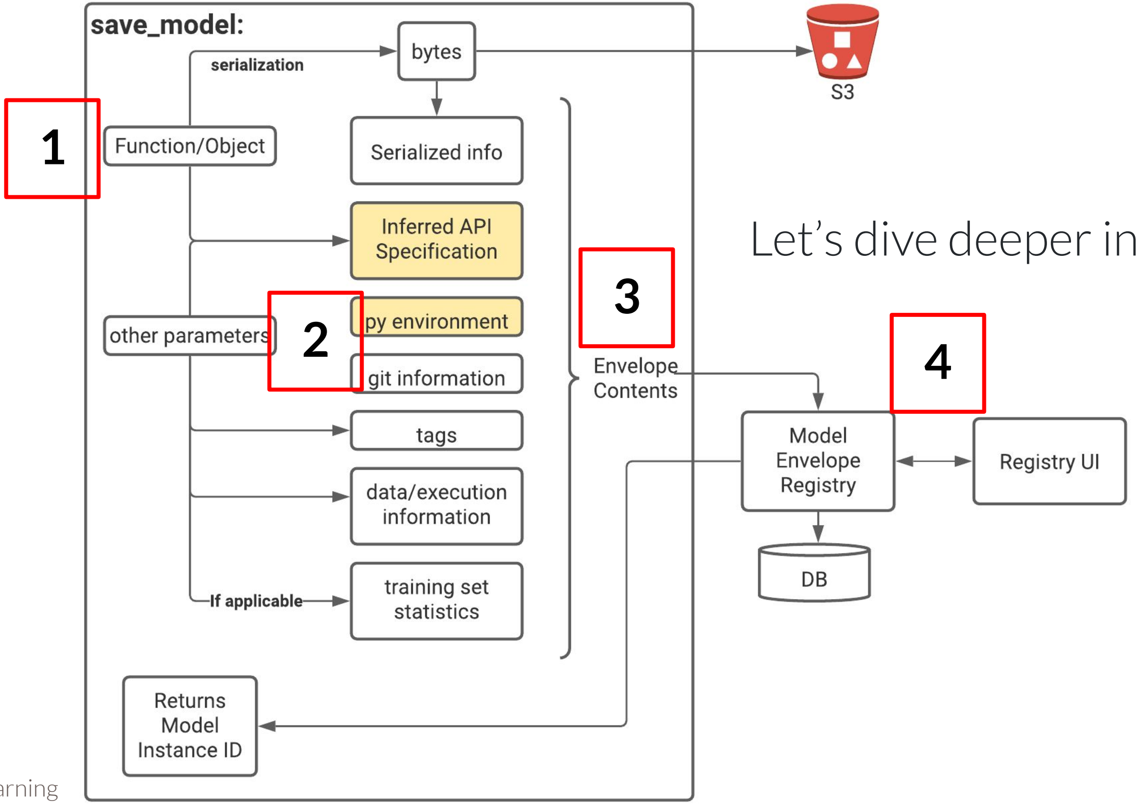
Typical Model Envelope use

1. call **save_model()** right after model creation in an ETL.
2. also have APIs to save metrics & hyperparameters, and retrieve envelopes.
3. once in an ✉️ **information is immutable** *except*:
    a. tags -- for curative purposes.
    b. metrics -- can add/adjust metrics.

What does save_model() do?



What does save_model() do?



Let's dive deeper into these.

How do we infer a Model API Schema?

Goal: infer from code rather than explicit specification.

Require either fully annotated functions with only python/typing standard types:

```
def good_predict_function(self, x: float, y: List[int]) -> List[float]:
def predict_needs_examples_function(self, x: pd.Dataframe, y):
```

Or, example inputs that are inspected to get a schema from:

```
my_envelope = me.save_model(instance_name='my_model_instance_name',
                            instance_description='my_model_instance_description',
                            model=model,
                            query_function='predict',
                            api_input=df_X, api_output=df_y,
                            tags={'canonical_name':'foo-bar'})
```

required for DF inputs →

How do we infer a Model API Schema?

Goal: infer from code rather than explicit specification.

Require either fully annotated functions with only python/typing standard types:

```
def good_pr
def predict
```

Or, examp

```
my_envelope
                  ion',
          model=model,
          query_function='predict',
          api_input=df_X, api_output=df_y,
          tags={'canonical_name':'foo-bar'})
```

required for DF inputs →

Why get a schema?

> Required for any form of validation:

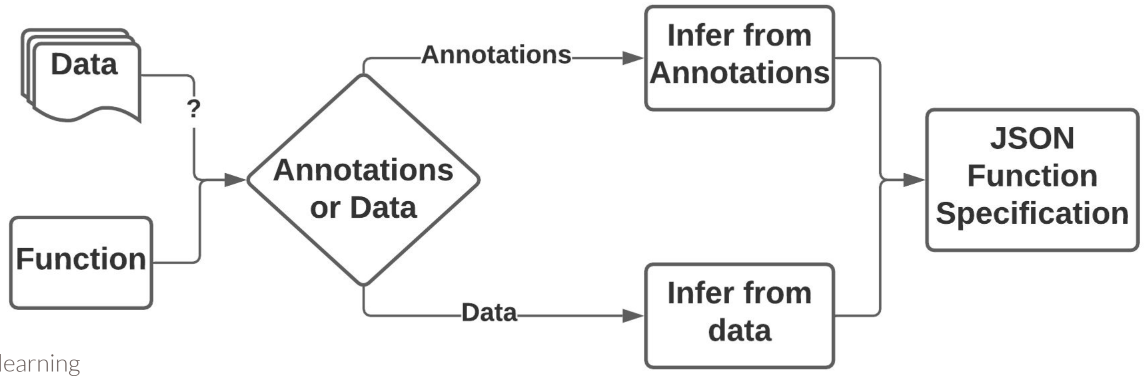
E.g. did the model get passed the right inputs?

Why this way?

> To avoid breakage when something is updated.

Model API Schema - Under the hood

- One of the most complex parts of the code base (90%+ test coverage!)
- We make heavy use of the *typing_inspect* module & *isinstance*().
  - We create a schema similar to TFX.
- Key component to enable exercising models in different contexts.
  - Enables code creation and input/output validation.
- Current limitations: **no default values in functions**.



How do we capture python dependencies?

```python
import model_envelope as me
from sklearn import linear_model

df_X, df_y = load_data_somehow()
model = linear_model.LogisticRegression(multi_class='auto')
model.fit(df_X, df_y)

my_envelope = me.save_model(instance_name='my_model_instance_name',
                            instance_description='my_model_instance_description',
                            model=model,
                            query_function='predict',
                            api_input=df_X, api_output=df_y,
                            tags={'canonical_name':'foo-bar'})
```

Point: no explicit passing of scikit-learn to save_model().

How do we capture python dependencies?

```
import model_envelope as me
from sklearn import linear_model

df_X, df_y
model = li
model.fit(

my_envelop
                                                                  tion',
                              model=model,
                              query_function='predict',
                              api_input=df_X, api_output=df_y,
                              tags={'canonical_name':'foo-bar'})
```

Why auto capture dependencies?

> Want to be able to reproduce & reuse models.

> Easy for the user to get wrong.

Point: no explicit passing of scikit-learn to save_model().

How do we capture python dependencies?

Assumption:
We all run on the same* base linux environment in training & production.

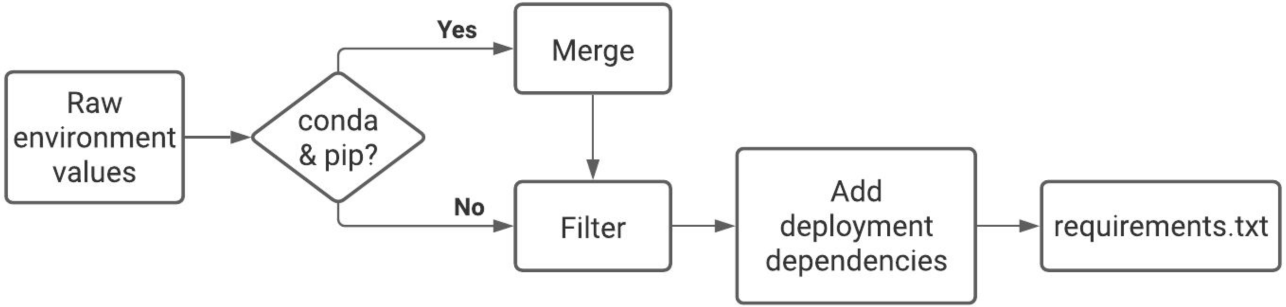
Store the following in the Model Envelope:

- Result of `import sys; sys.version_info`
- Results of > `pip freeze`
- Results of > `conda list --export`

Local python modules (not installable)**:**

- Add modules as part of save_model() call.
- We store them with the model bytes.

How do we build the python deployment env.?

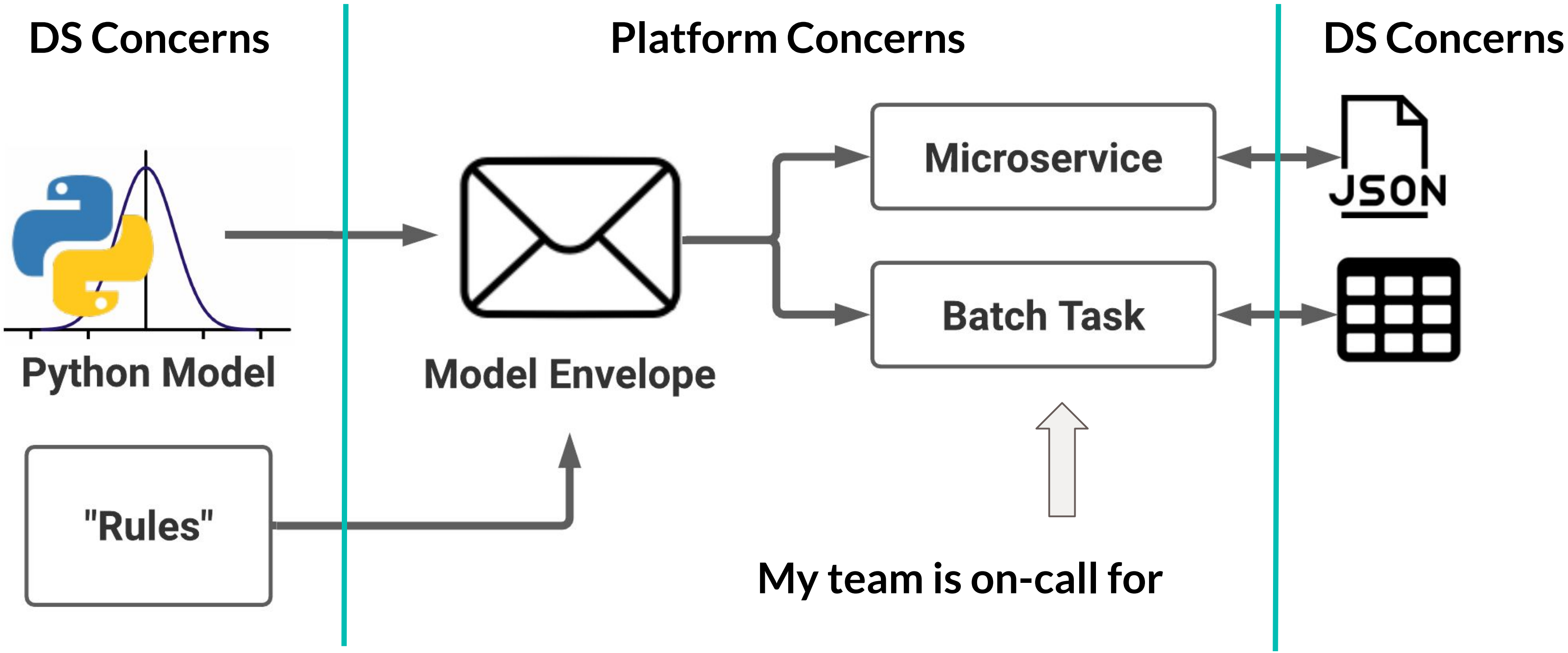


Filter:

- hard coded list of dependencies to filter. E.g. jupyterhub.
- upkeep cheap; add/update every few months.

Stitch Fix
“Deployment for free”
Model Envelope & envelope mechanics
> Impact of being on-call
Summary & Future Work

Remember this split:



Impact of being on-call

a pager

Two truths:

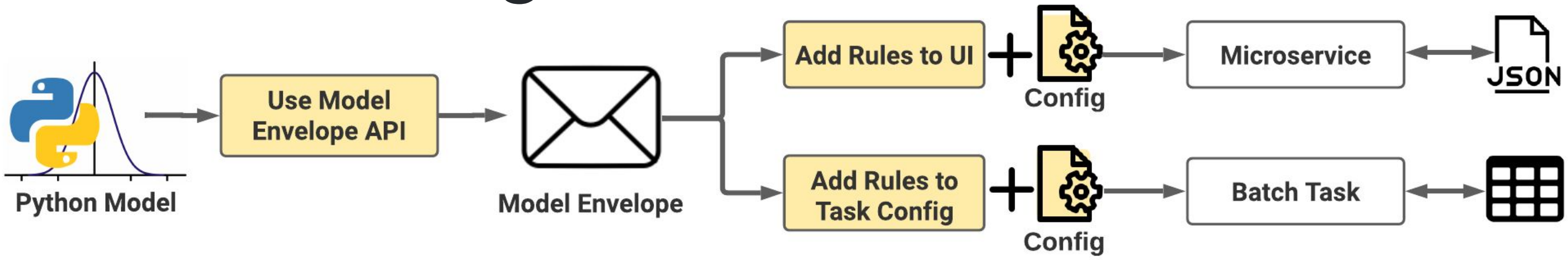
- No one wants to be paged.
- No one wants to be paged for a model they didn't write!

But, this incentivizes Platform to build out MLOps capabilities:

- Capture bad models before they're deployed!
- Enable observability, monitoring, and alerting to speed up debugging.

Luckily we have autonomy and freedom to do so!

What can we change?



API

Automatic capture == license to change:

- Model API schema
- Dependency capture
- Environment info: git, job, etc.

Incentives for DS to additionally provide:

- Datasets for analysis
- Metrics
- Tags

Deployment

MLOps approaches to:

- Model validation
- Model deployment & rollback
- Model deployment vehicle:
  - From logging, monitoring, alerting
  - To architecture: microservice, or Ray, or?
- Dashboarding/UIs

Overarching benefit

1. Data Scientists get to focus more on modeling.
    a. more business wins.
2. Platform focuses on MLOps:
    a. can be a rising tide that raises all boats!

Stitch Fix
“Deployment for free”
Model Envelope & envelope mechanics
Impact of being on-call
> Summary & Future Work

Summary - “Deployment for free”

We enable deployment for free by:

- Capturing a comprehensive model artifact we call the Model Envelope.
- The Model Envelope facilitates code & environment generation for model deployment.
- Platform owns the Model Envelope and is on-call for generated services & tasks.

Business wins:

- Data Scientists get to focus more on modeling.
- Platform is incentivized to improve and iterate on MLOps practices.

Future Work

- **Better MLOps features:**
  - Observability, scalable data capture (e.g. whylogs), & alerting.
  - Model Validation & CD patterns.
- **“Models on Rails”:**
  - Target specific SLA requirements.
- **Configuration driven model creation:**
  - Abstract away glue code required to train & save models.

Thank you! We’re hiring! Questions?

@stefkrawczyk
linkedin.com/in/skrawczyk

Try out Stitch Fix → goo.gl/Q3tCQ3

#CS329S #MLOps #machinelearning

STITCH FIX