Deployment

Prototyping with Deep Learning

Learning outcomes

- Identify best practices in model deployment
- Save, load, and use your DL models

What can we do with Jupyter notebooks?

Add interactive widgets

Create slideshow presentations

Write technical blogs

… But software **deployment** is something else

Goals

Use trained models in **production** environments

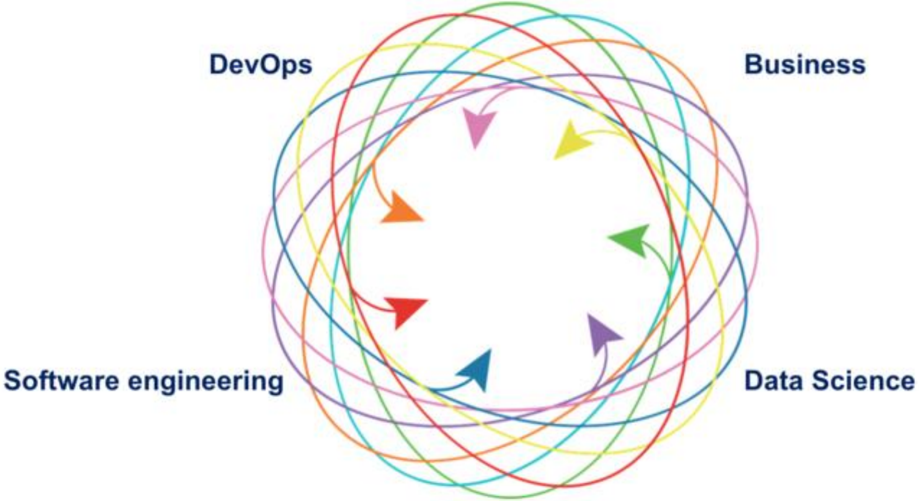
Also monitor models usage over time

Also improve models with new data

Why are ML systems hard?

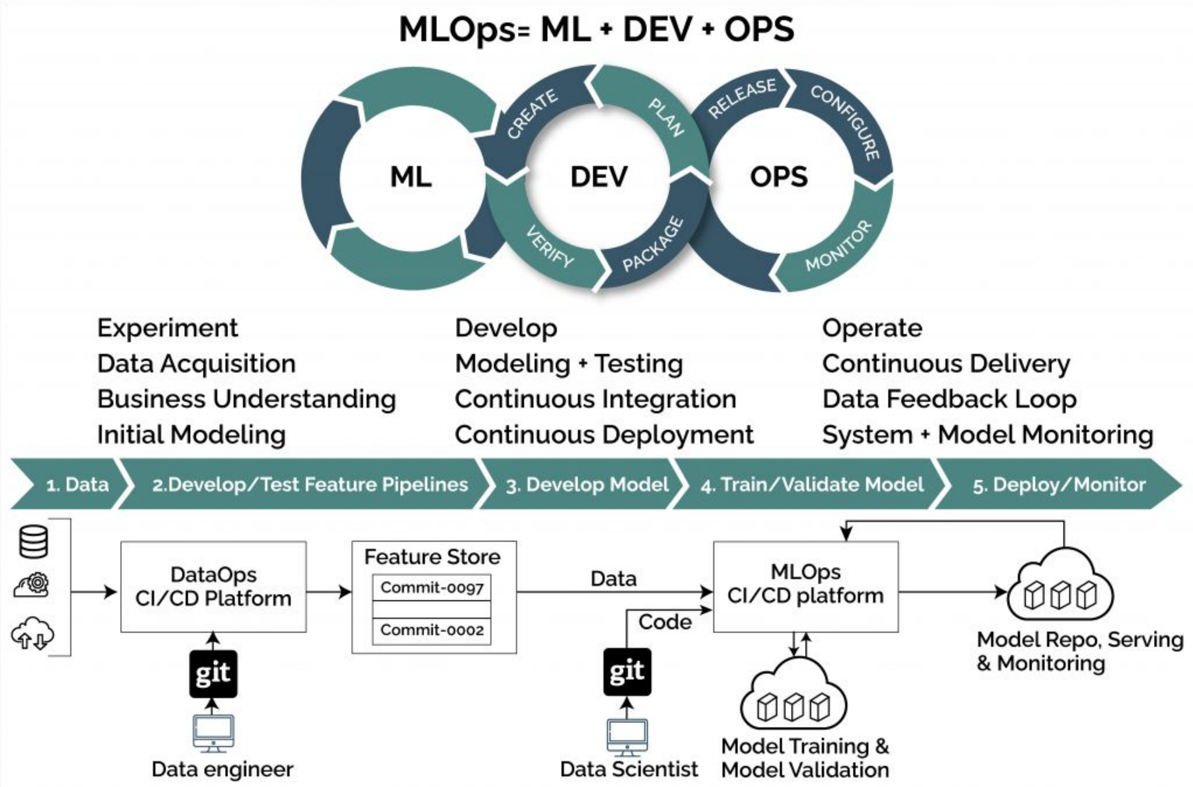
Some *key* issues:

- Entanglement
- Data dependencies
- Configuration
- Data preparation
- Tracking model errors
- Separation of expertise



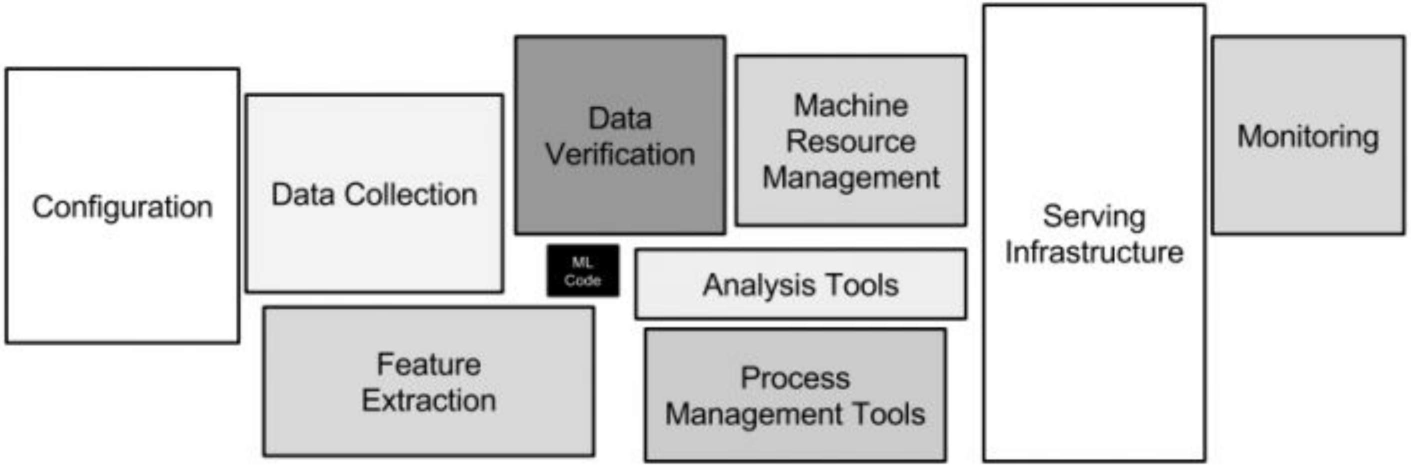
https://christophergs.github.io/machine%20learning/2019/03/17/how-to-deploy-machine-learning-models/

MLOps pipeline



https://radiant.digital/the-fundamentals-of-mlops-the-enabler-of-quality-outcomes-in-production-environments/

The Hidden Technical Debt in ML Systems



Configuration

Data Collection

Data Verification

Machine Resource Management

Serving Infrastructure

Monitoring

ML Code

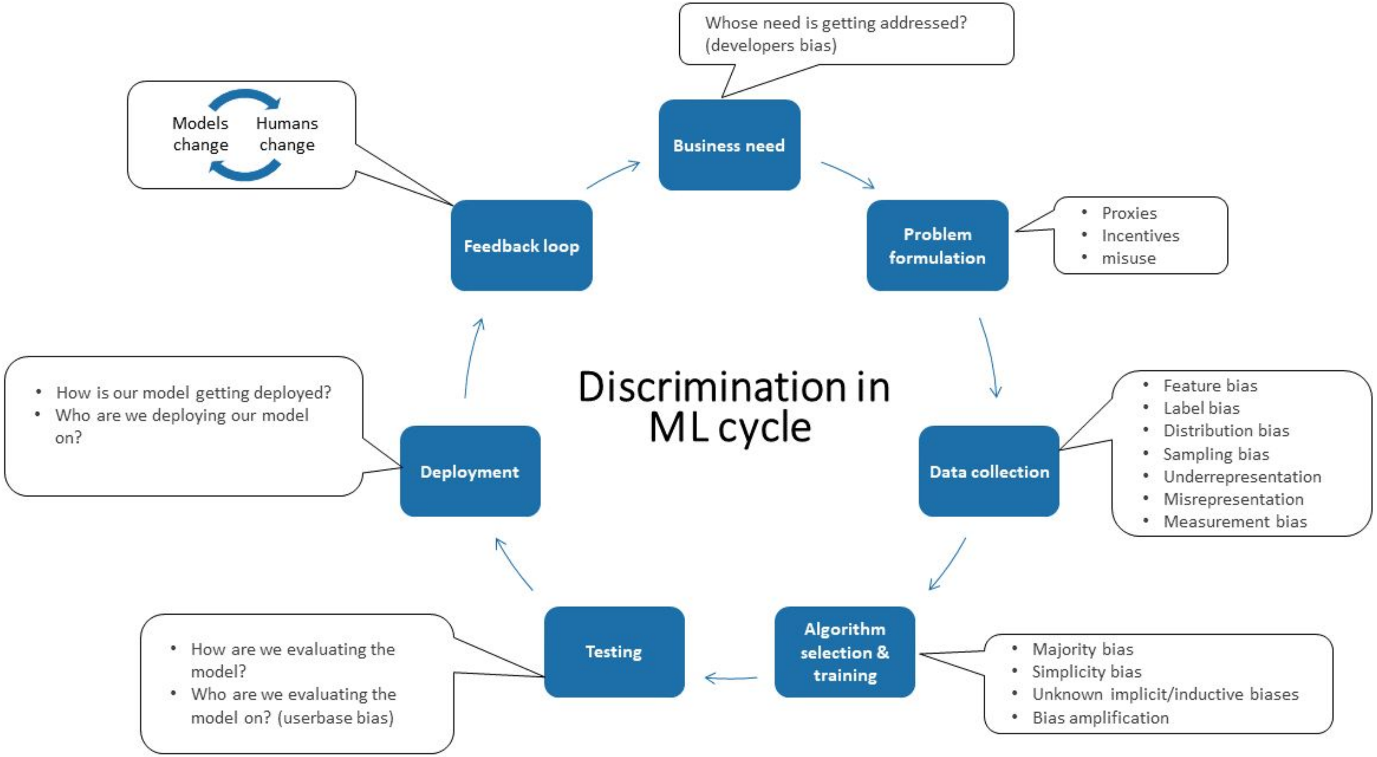
Analysis Tools

Feature Extraction

Process Management Tools

https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf

Biases everywhere



Whose need is getting addressed? (developers bias)

- Proxies
- Incentives
- misuse

- Feature bias
- Label bias
- Distribution bias
- Sampling bias
- Underrepresentation
- Misrepresentation
- Measurement bias

- Majority bias
- Simplicity bias
- Unknown implicit/inductive biases
- Bias amplification

- How are we evaluating the model?
- Who are we evaluating the model on? (userbase bias)

- How is our model getting deployed?
- Who are we deploying our model on?

https://twitter.com/fereshte_khani/status/1496281636924960770

Key design principles

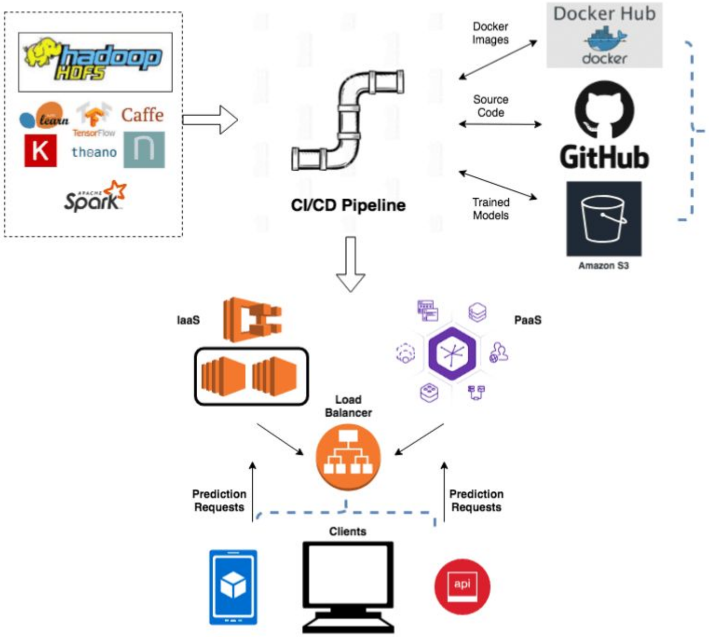
Plan for **reproducibility**

ML as part of your **build pipeline**

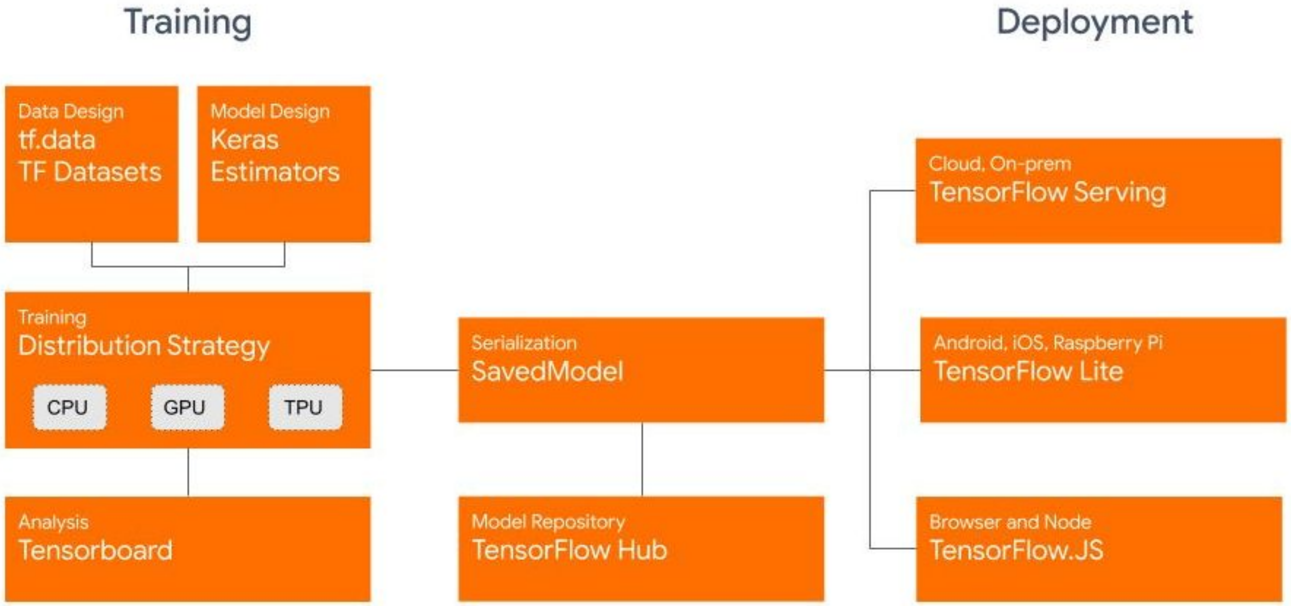
Plan for **extensibility**

Plan for **modularity**

Testing, testing, testing



Some TF tools



Training

Data Design
tf.data
TF Datasets

Model Design
Keras
Estimators

Training
Distribution Strategy
CPU GPU TPU

Analysis
Tensorboard

Serialization
SavedModel

Model Repository
TensorFlow Hub

Deployment

Cloud, On-prem
TensorFlow Serving

Android, iOS, Raspberry Pi
TensorFlow Lite

Browser and Node
TensorFlow.JS

https://blog.tensorflow.org/2019/09/tensorflow-20-is-now-available.html

Saving trained models

In Keras

`model.save(…)`

Only architecture: `model.to_json()` or `model.to_yaml()`

Only weights: `model.save_weights()`

https://keras.io/getting-started/faq/#how-can-i-save-a-keras-model

In Pytorch

`torch.save(model.state_dict(), …)`

https://pytorch.org/tutorials/beginner/saving_loading_models.html

What can (and should) be saved?

Model architecture

Model weights

Training configuration: loss function, epochs

Optimizer: learning rate, state (to resume training)

Model file formats

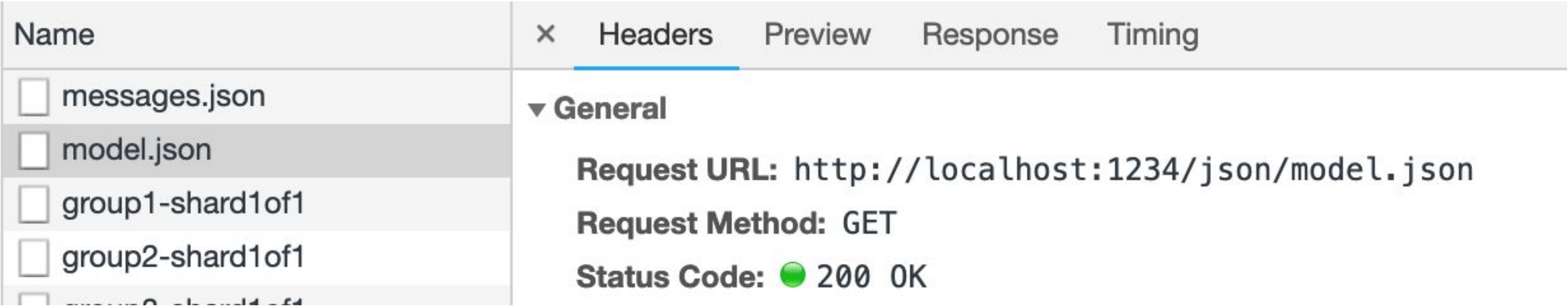
`.pt` or `.pth`	Pytorch
`.pb`	Protobuf (TF native)
`.tflite`	Tensorflow, on-device
`.h5`	HDF, cross-platform
`.json`	TF.js

Don’t use Python’s `pickle`!

Model format conversion

https://github.com/tensorflow/tfjs/tree/master/tfjs-converter

```
~$ tflite_convert --graph_def_file=model.pb --output_file=model.tflite

~$ tflite_convert --keras_model_file=model.h5 --output_file=model.tflite

~$ tensorflowjs_converter --input_format keras model.h5 /path/to/dir
```



Name

messages.json

model.json

group1-shard1of1

group2-shard1of1

× Headers Preview Response Timing

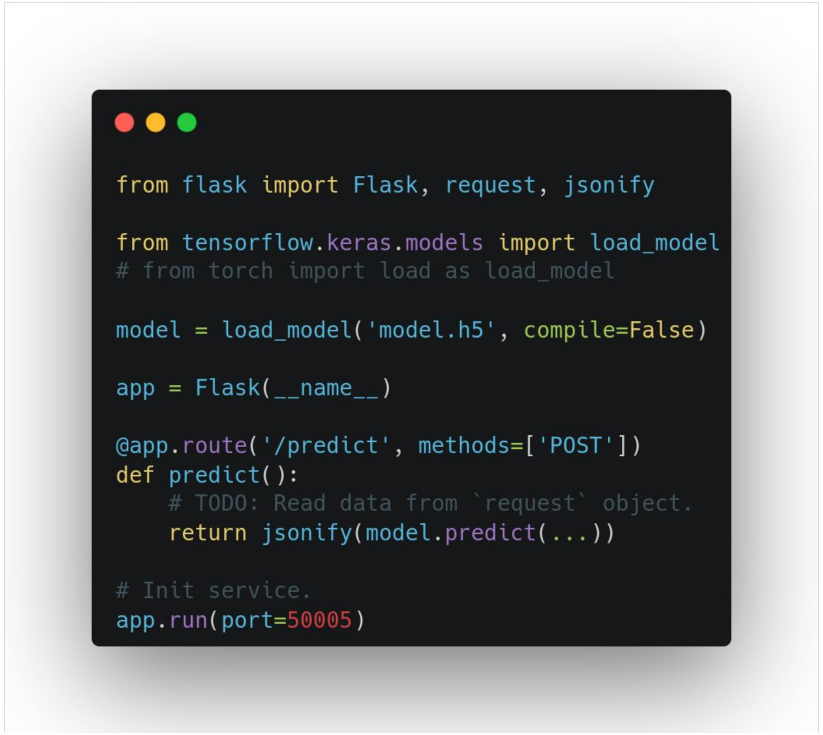
▾ General

Request URL: http://localhost:1234/json/model.json

Request Method: GET

Status Code: ● 200 OK

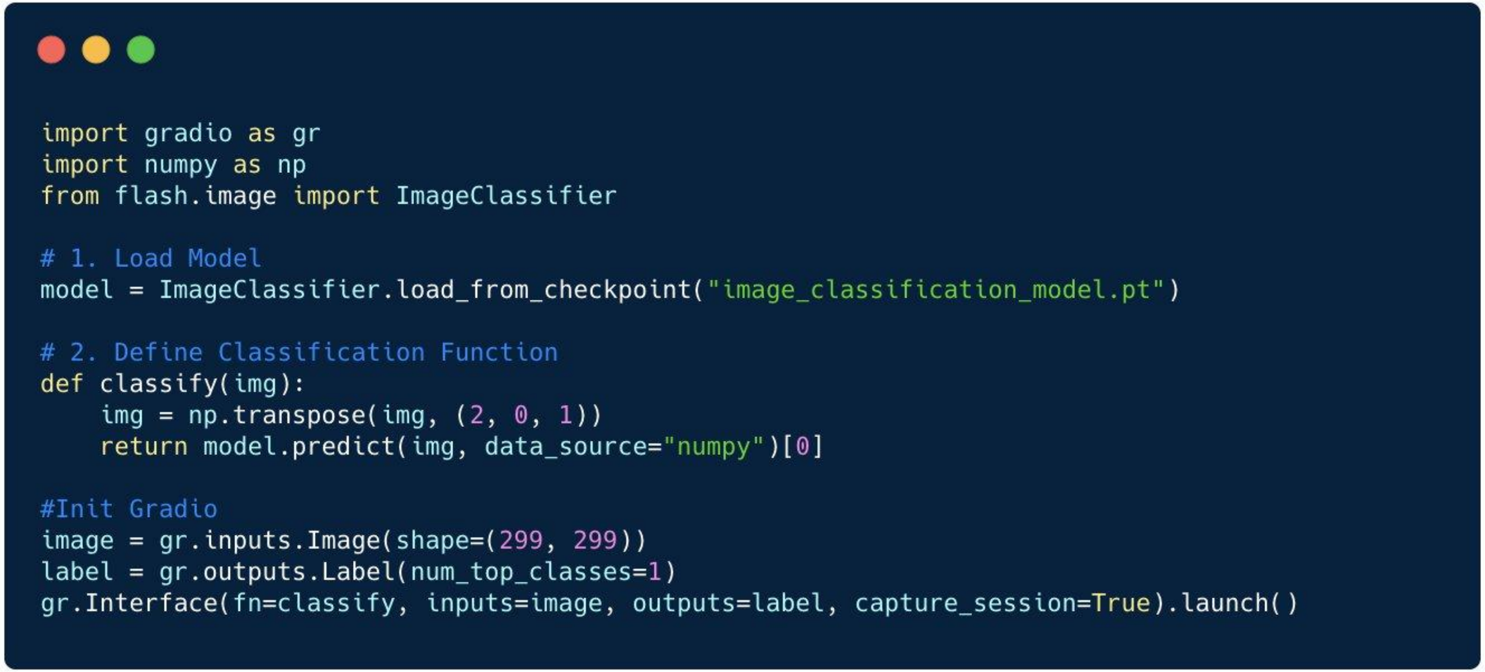
Model serving with Flask (web APIs)



```python
from flask import Flask, request, jsonify

from tensorflow.keras.models import load_model
# from torch import load as load_model

model = load_model('model.h5', compile=False)

app = Flask(__name__)

@app.route('/predict', methods=['POST'])
def predict():
    # TODO: Read data from `request` object.
    return jsonify(model.predict(...))

# Init service.
app.run(port=50005)
```

https://flask.palletsprojects.com/en/2.1.x/

Model serving with Gradio (graphical UIs)



```
import gradio as gr
import numpy as np
from flash.image import ImageClassifier

# 1. Load Model
model = ImageClassifier.load_from_checkpoint("image_classification_model.pt")

# 2. Define Classification Function
def classify(img):
    img = np.transpose(img, (2, 0, 1))
    return model.predict(img, data_source="numpy")[0]

#Init Gradio
image = gr.inputs.Image(shape=(299, 299))
label = gr.outputs.Label(num_top_classes=1)
gr.Interface(fn=classify, inputs=image, outputs=label, capture_session=True).launch()
```

https://gradio.app/getting_started/

Monitoring

https://wandb.ai/site

Some resources

Best practices: https://opendatascience.com/best-practices-for-deploying-machine-learning-in-the-enterprise/

Challenges: https://towardsdatascience.com/443af67493cd

CI/CD in ML: https://martinfowler.com/articles/cd4ml.html

DL in production (book): https://github.com/The-AI-Summer/Deep-Learning-In-Production