Software Engineering for AI-Enabled Systems

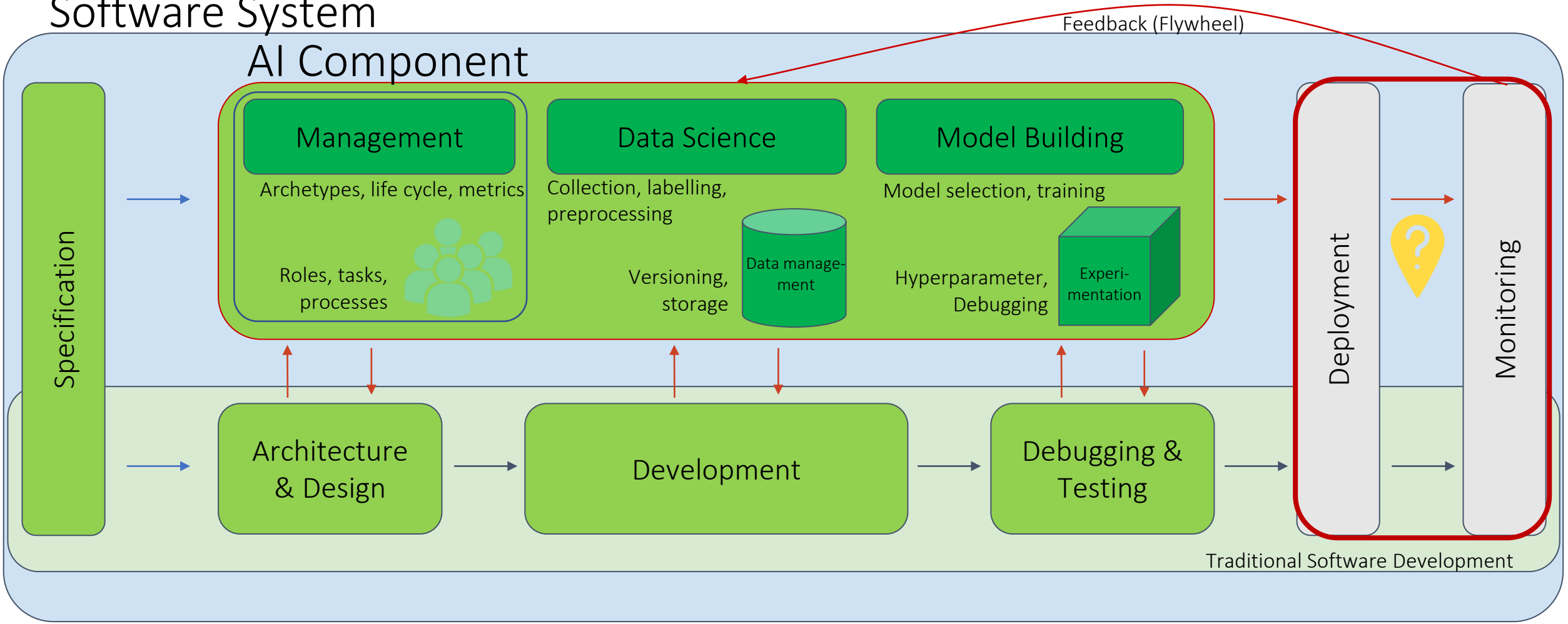
SOFTWARE SYSTEME

UNIVERSITÄT LEIPZIG

Prof. Dr.-Ing. Norbert Siegmund

Software Systems

Software System



How to deploy and monitor the AI system in production?

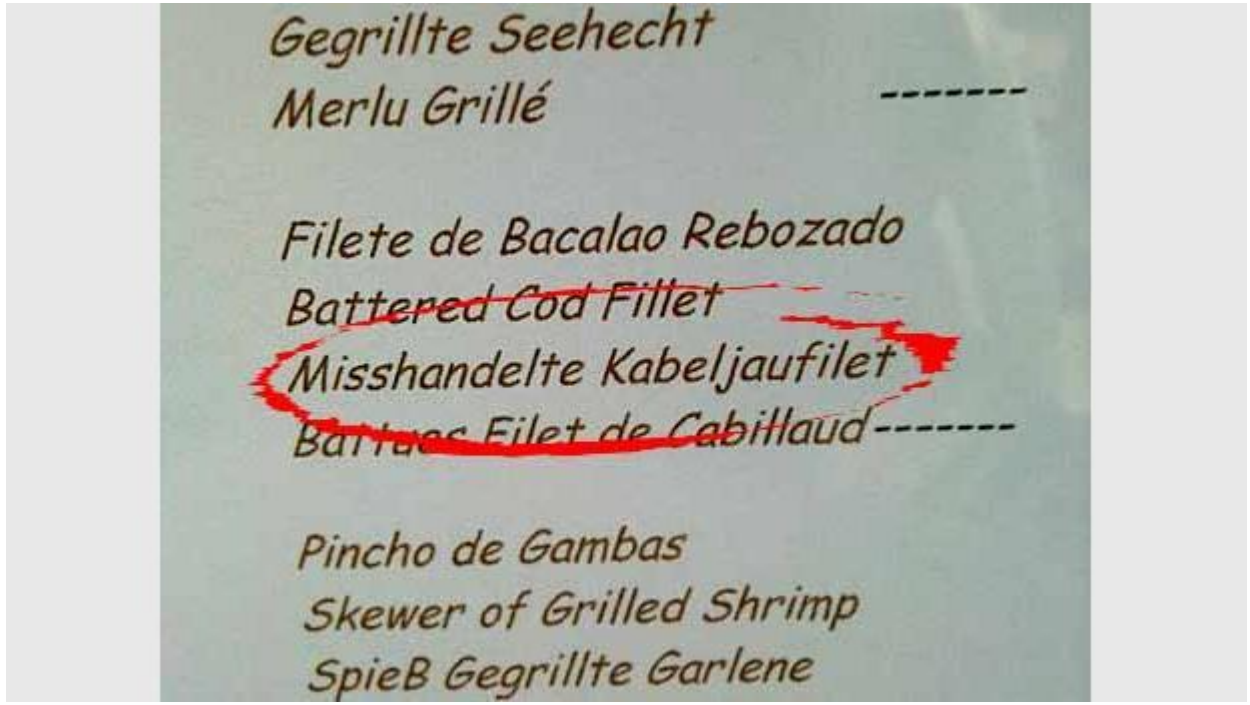
- Which tools and frameworks for to use for deployment?
- How to handle multiple versions of your model?
- How to observe data distribution to detect data and model shifts?
- How to log the AI output?
- How to prevent or inform fatal AI errors? How to implement fall back measures?
- What frameworks and tools exist to incorporate monitoring?

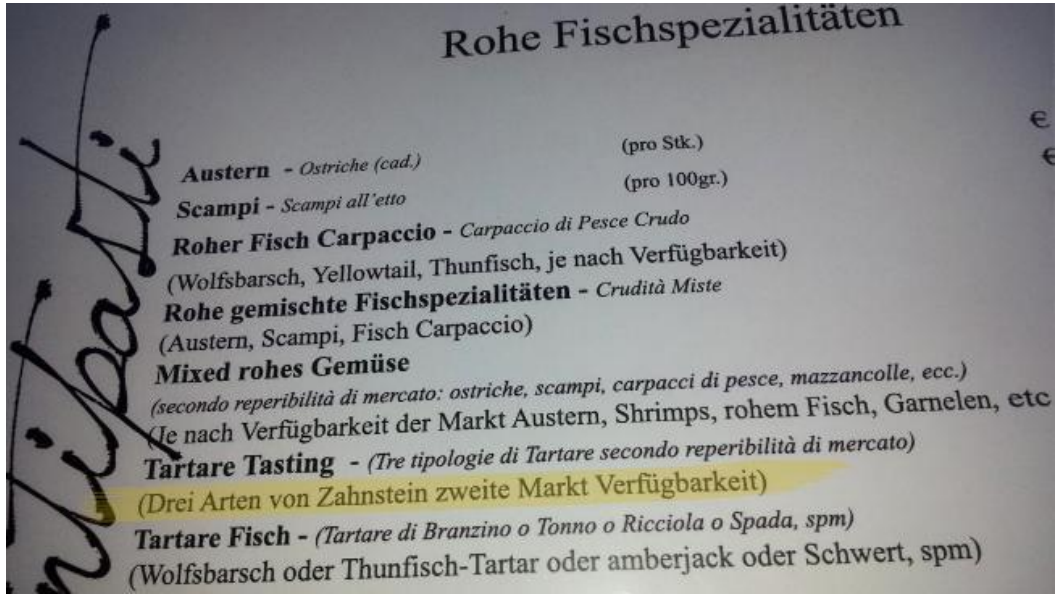
Deployment and Monitoring

TL;DR:

- Tools and architectures for deploying ML models
- Incorporate monitoring and labelling instruments
- Closing the flywheel
- Non-functional properties to log: Telemetry
- Monitor AI inputs: Data and concept drift

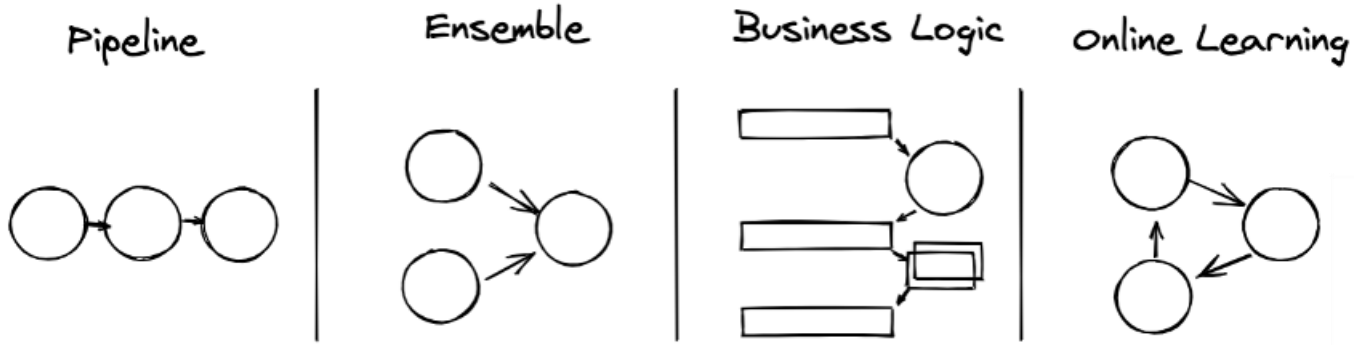
AI Failures in Production

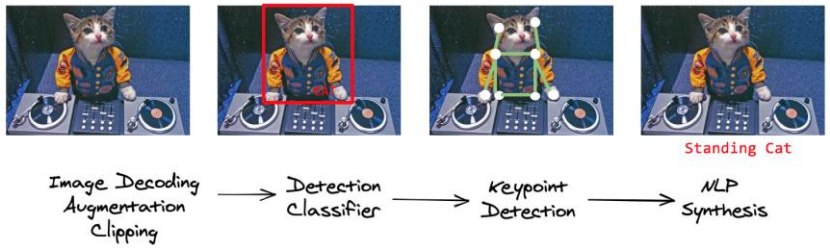


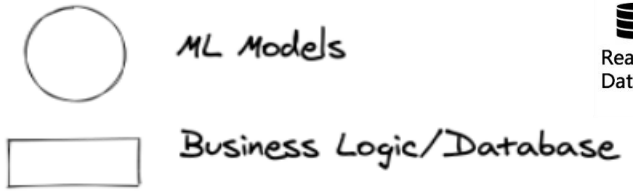


Topic I:
Deployment

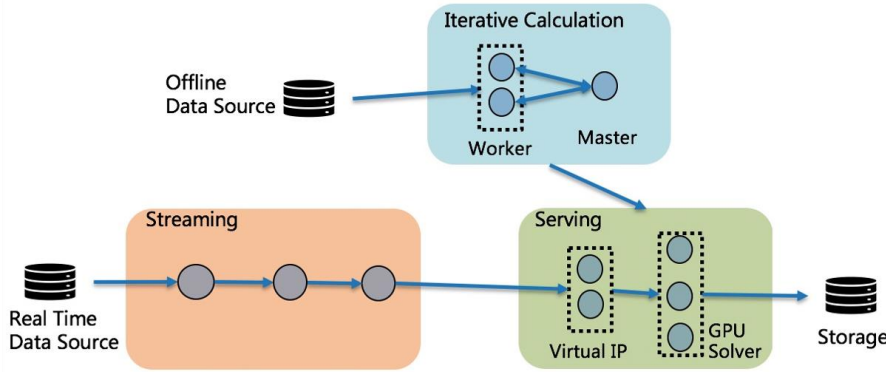
Deployment Inference Patterns



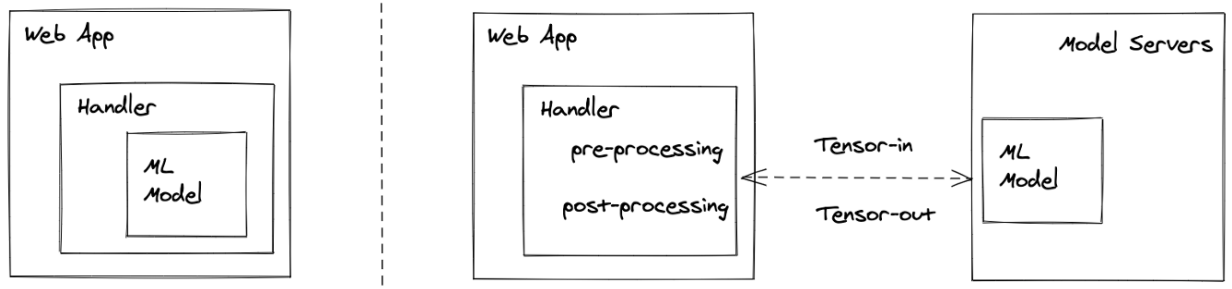




Source: Anyscale

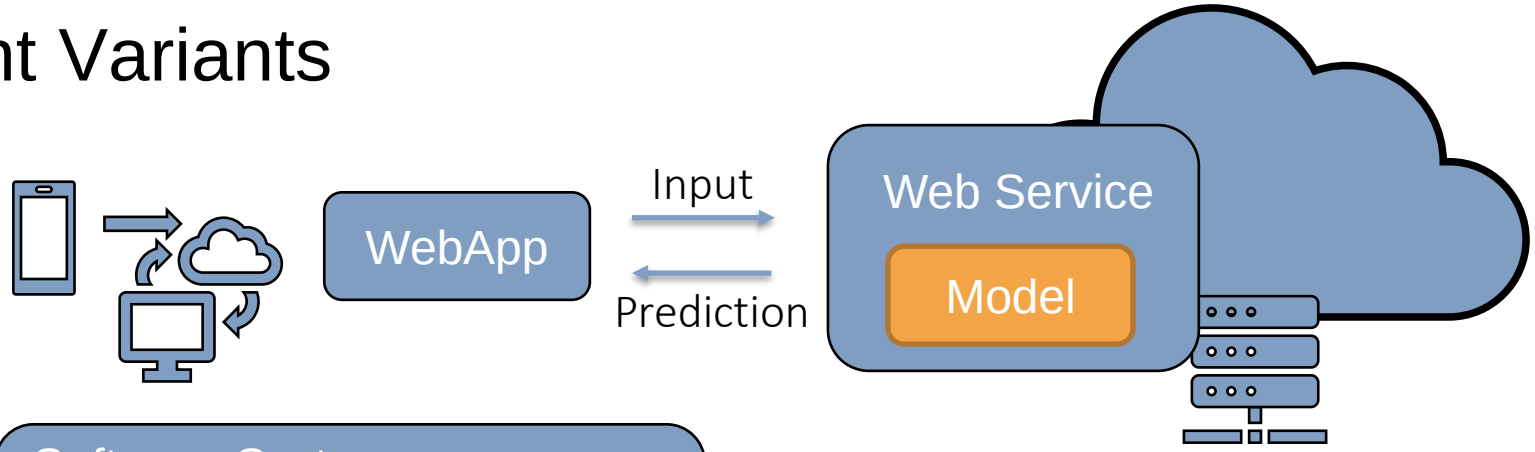


Source: Ant group

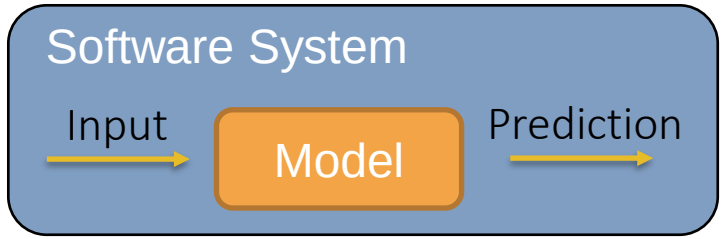


Model Deployment Variants

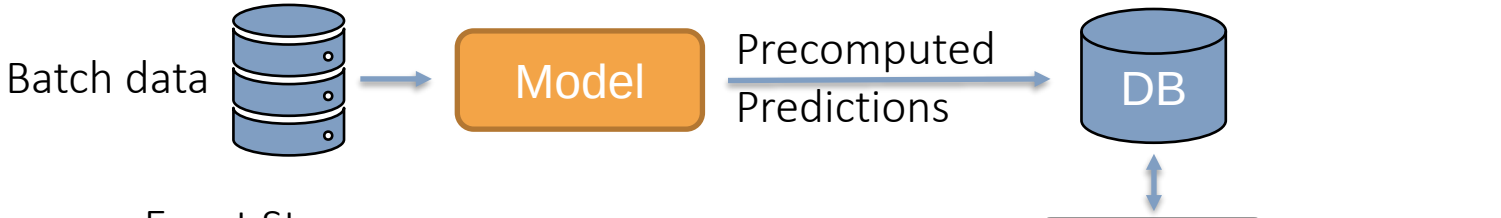
Model-as-Service:



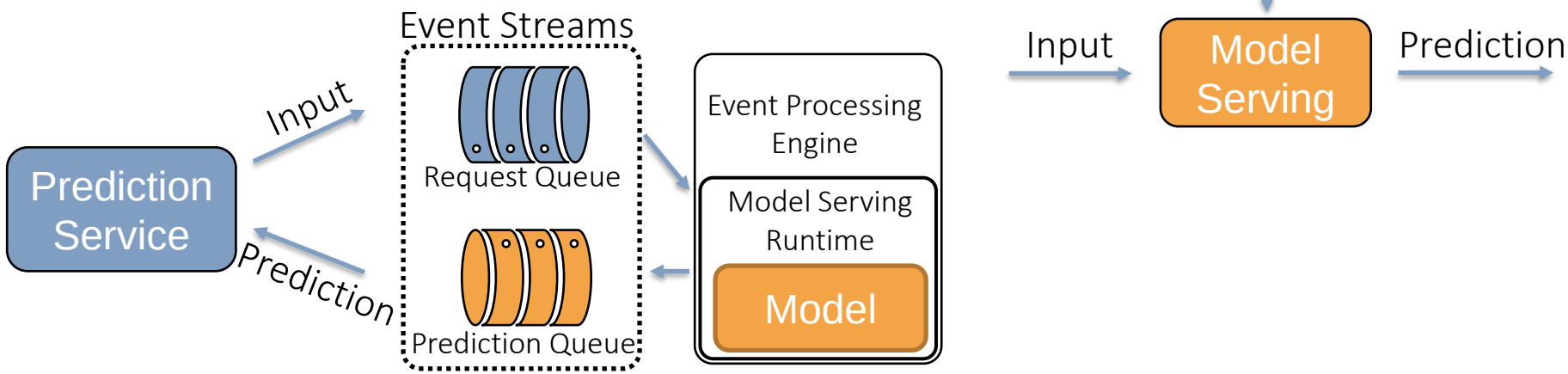
Model-as-Dependency:



Precomputed Predictions:



Model-on-Demand:



Deployment Strategies

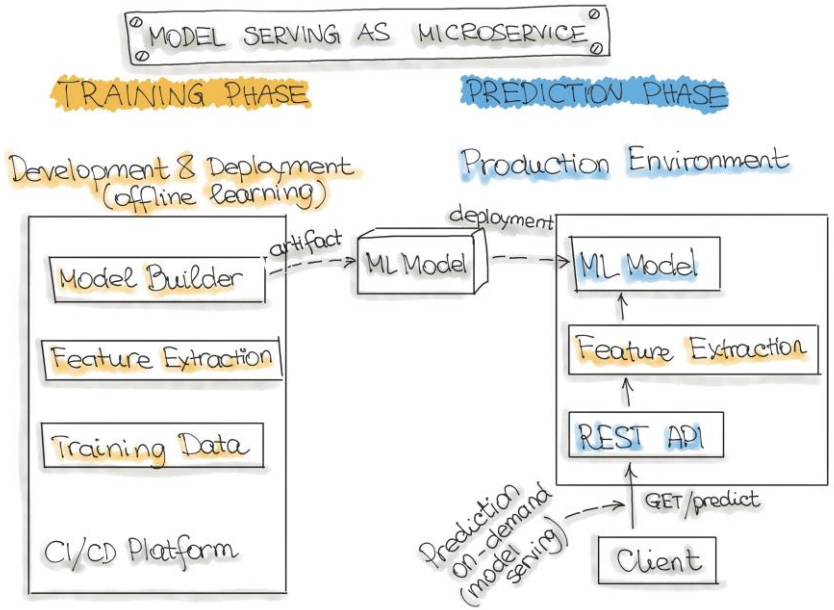
Model Service:

- Provide a REST-API to enable interaction with the model
- Requires a server to host the model, usually accompanied with a load balancer
- Sub-variant: Lambda or SaaF based provisioning: URI requests will spin up a single function that may load and trigger a model
- Central point for logging important telemetry: queries (with input data), volume, predictions (with failure or probability information), inference time, memory consumption, bandwidth, frequency of queries, timestamps, etc.

Embedded model:

- Encode model into a shippable format
- Use format-specific libraries to load the model and conduct inference

Tools and frameworks: https://mlops.toys/model-serving



Model Service: Designing a REST-API

Be futureproof:

- Foresee extension points or future usage scenarios
- Consider different data formats by choosing between strict, domain-dependent types vs. flexible general types
- Incorporate version number in API to refer to specific models (prepare A/B testing, canary deployment, etc.)
- Consider alternative models in production (for different users, regions, etc.) and how the API differentiates among them (e.g., different endpoint vs. encoded in query)

Be robust and scalability:

- Redundant server and load balancer vs. infrastructure cost
- Error handling and logging (watch out for too much telemetry data)

Flask (+ Docker)

Good basic alternative (now outdated), but not for professional use as it lacks some ML related functionality

```
app = Flask(__name__)
port = os.environ.get("PORT", 5000)
app.run(debug=False, host="0.0.0.0", port=port)
```

Endpoint (URI) that routes a request to this method

Call to a Python method that actually handles inference (model should be loaded prior to that)

Jinja2 template to embed the prediction into a rendered response Web page (could also return a JSON file)

```
@app.route("/predict", methods=["GET","POST"])
def predict():
    threshold = request.args.get("threshold", "")
    if threshold is None or threshold == '':
        threshold = 0.5
    threshold = float(threshold)
    # convert string of image data to uint8
    if request.method == 'POST':
        # check if the post request has the file part
        if 'file' not in request.files:
            flash('No file part')
            return redirect(request.url)
        file = request.files['file']
        # if user does not select file, browser also
        # submit a empty part without filename
        if file.filename == '':
            flash('No selected file')
            return redirect(request.url)
        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            full_filename = os.path.join(app.config['UPLOAD_FOLDER'], filename)
            file.save(full_filename)

            prediction, prob = pred(full_filename, threshold)
            prob = prob * 100
            if (prediction == "Cat"):
                prob = abs(prob-100)
            return render_template('prediction.html', temp_image = full_filename, pred_class = prediction, prob = prob)
    return '''
    <!doctype html>
    <title>Upload new Image</title>
    <h1>Upload new image for prediction</h1>
    <form method=post enctype=multipart/form-data>
      <input type=file name=file>
      <input type=submit value=Upload>
    </form>
```

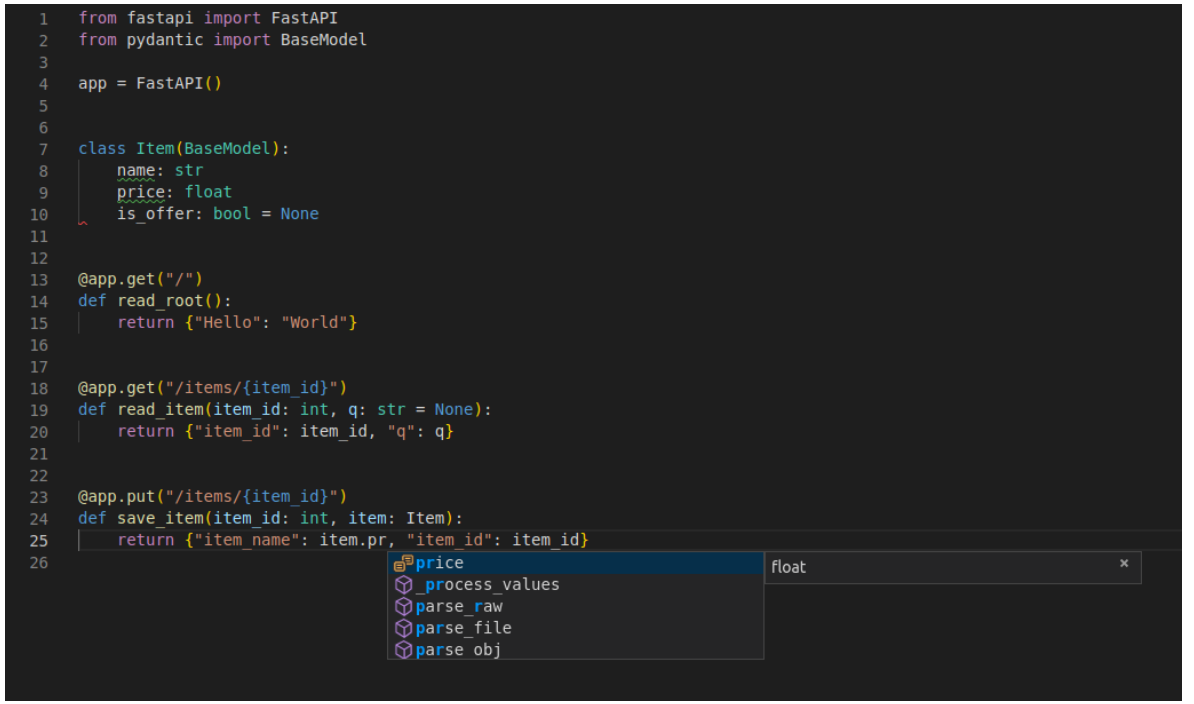
Checks for data availability, etc.

FastAPI

Web framework for building fast APIs using Uvicorn or Hypercorn as Web servers under the hood:

- Interactive and automated API documentation
- Specify types using pydantic, enabling type checks and code completion support in IDEs
- Automated data validation used the declared types
- Automated conversion of input data (JSON, headers, forms, files, etc.) to Python types and vice versa
- Customize validation checks, such as value ranges
- Authentication mechanisms, such as OAuth2
- Async APIs

Not specific to ML, but the data validation parts and focus on performance makes it an ideal first choice



Summary: Pipelining and DAG-based Ensemble Models

Wrap models in web server

```python
@app.route("/predict")
def prediction_handler(raw_input):
    models = load_models()
    output = raw_input
    for model in models:
        output = model(output)
    return output
```

Simple but not performant

Many specialized microservices

```yaml
apiVersion: "serving.kubeflow.org/v1beta1"
kind: "InferenceService"
metadata:
  name: "sklearn-irisv2"
spec:
  predictor:
    sklearn:
      protocolVersion: v2
      storageUri: "gs://seldon-models/sklearn/iris"
```

Complex and hard to operate

Source: Anyscale

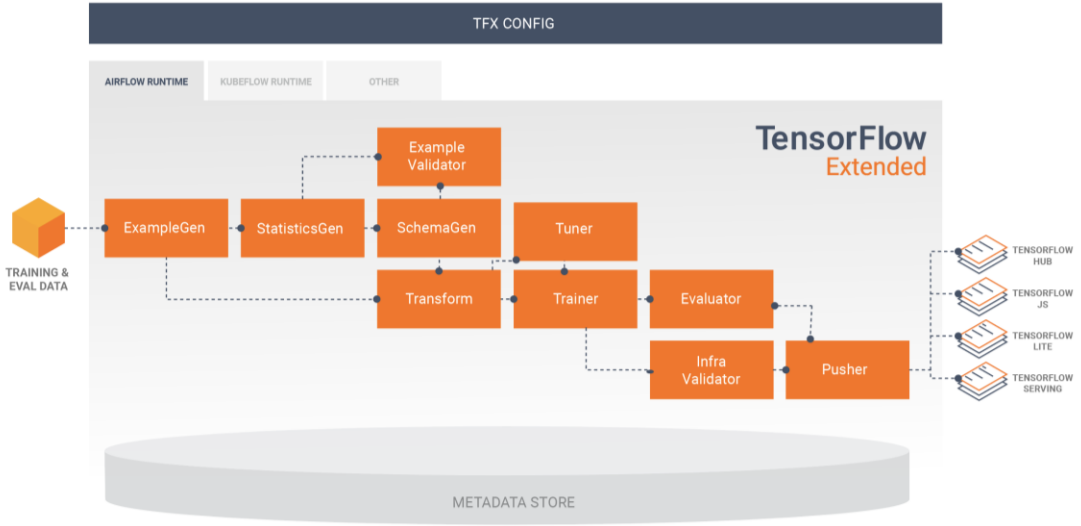
Better way: Specify pipelines

Tensorflow Serving (TFS) in TFX

TFS loads a TF model and provides either a REST interface for inference or a gRPC connection point for remote procedure calls. Seems to be faster than FastAPI when using GPUs (synchronizes requests with GPU resource), can handle distributed server computations.

TFX does more:

- Platform for orchestration of ML pipelines;
- Basic ML architecture components in the pipeline (e.g., trainer, data set validator, parameter tuner);
- Libraries providing typical ML pipeline tasks (tensorflow, ML metadata, etc.)



Tensorflow Model Server

Highly configurable to adjust the serving functionality (e.g., composite models, combinations with lookup tables, etc.):

```
docker run -t --rm -p 8501:8501 \
    -v "$(pwd)/models/:/models/" tensorflow/serving \
    --model_config_file=/models/models.config \
    --model_config_file_poll_wait_seconds=60
```

Run server as Docker container and supply a config file

```
model_config_list {
  config {
    name: 'my_first_model'
    base_path: '/tmp/my_first_model/'
    model_platform: 'tensorflow'
  }
  config {
    name: 'my_second_model'
    base_path: '/tmp/my_second_model/'
    model_platform: 'tensorflow'
  }
}
```

Provide a ModerServerConfig protocol buffer, which essentially points to a model and specifies its platform

```
max_batch_size { value: 128 }
batch_timeout_micros { value: 0 }
max_enqueued_batches { value: 1000000 }
num_batch_threads { value: 8 }
```

Enable batch processing just via configuration

```
prometheus_config {
  enable: true,
  path: "/monitoring/prometheus/metrics"
}
```

Turn on model monitoring using Prometheus

TorchServe

Tool for serving PyTorch models:

- Model management API (worker to model allocation)
- Inference API (Rest and gRPC for batch processing)
- Pipeline support for complex inference steps
- Integrates into other frameworks, such as MLflow, Kubeflow, Sagemaker, etc.
- Metrics export for Prometheus

Usage:

- Store model in the model archive as .mar file
- Start the torchserve server with the path to the archive

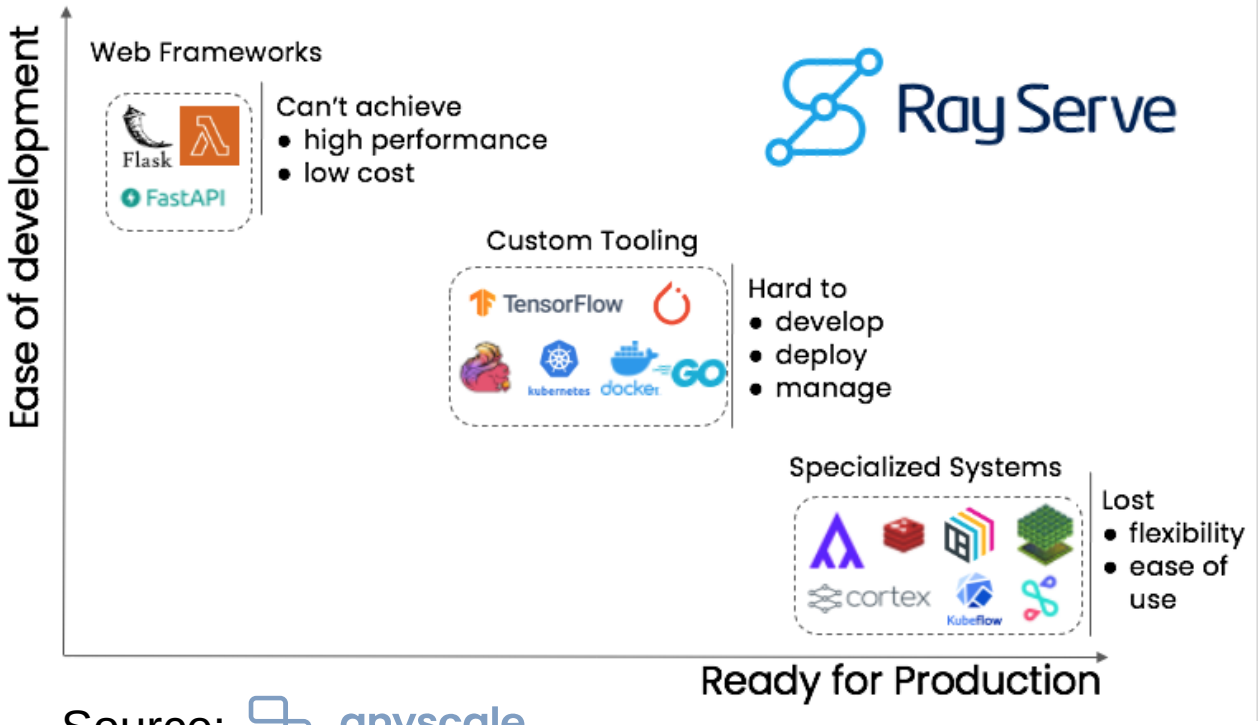
```
torchserve --model-store /models --start --models all
```

- Now, it provides REST and gRPC APIs for the models in the given archive

```
curl http://localhost:8080/predictions/resnet-18/2.0 -T kitten_small.jpg
```

```
{
   "class": "n02123045 tabby, tabby cat",
   "probability": 0.42514491081237793
}
```

RayServe



Source: anyscale

Ray Serve

Pipelining models in production

```python
@serve.deployment
class Featurizer: …

@serve.deployment
class Predictor: …

@serve.deployment
class Orchestrator
   def __init__(self):
      self.featurizer = Featurizer.get_handle()
      self.predictor = Predictor.get_handle()

   async def __call__(self, inp):
      feat = await self.featurizer.remote(inp)
      predicted = await self.predictor.remote(feat)
      return predicted

if __name__ == "__main__":
    Featurizer.deploy()
    Predictor.deploy()
    Orchestrator.deploy()
```

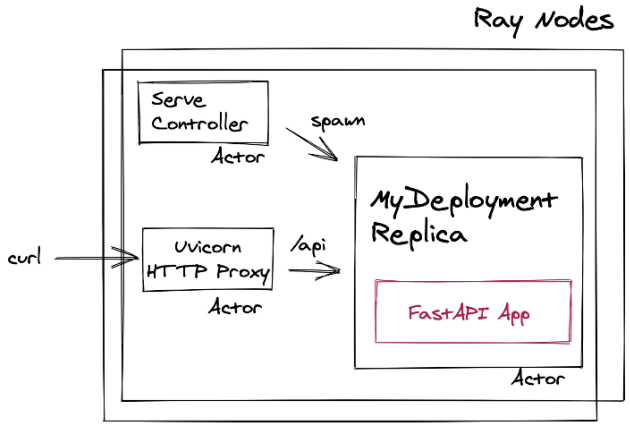
RayServe Integration with FastAPI

Get best of both worlds: input validation, authentification, code completion, etc. from FastAPI and scalability, parallelization, and ML-specific features from RayServe

Ray Serve: Ingress

```python
from fastapi import FastAPI

app = FastAPI()

@serve.deployment
@serve.ingress(app)
class MyDeployment:
    def __init__(self):
        self.model = load_model()

    @app.get("/predict/{user_id}")
    def predict(self, user_id: str):
        return self.model({"user_id": user_id})

MyDeployment.deploy()
```

FastAPI



Embedding Models

Approach: Use library specific functions to transform the model into an optimized one that trades accuracy for size and inference time

Advantages:

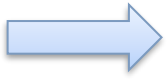
- Latency (no round trip to server)
- Privacy (data stays at the device)
- Connectivity (no internet connection required to process a query)
- Minimal resources (binary and memory model size optimized for resource constrained systems)
- Energy consumption (efficiency over accuracy; no communication required)

SKompiler & PMML & PFA (Portable Format for Analytics)

SKompiler transforms trained SKLearn models into other forms, such as SQL queries, Excel formulas, or SymPy expressions.

SymPy expressions can be translated to other programming languages.

```
from skompiler import skompile
expr = skompile(m.predict)
sql = expr.to('sqlalchemy/sqlite')
```



```
WITH _tmp1 AS
(SELECT .... FROM data)
_tmp2 AS
( ... )
SELECT ... from _tmp2 ...
```

PMML is an XML-based format for describing a model and a pipeline.

```
<MiningSchema>
<MiningField name="Income" usageType="active"/>
<MiningField name="Gender" usageType="active"/>
<MiningField name="Deductions" usageType="active"/>
<MiningField name="Hours" usageType="active"/>
</MiningSchema>
<RegressionTable targetCategory="1" intercept="-6.47942380092536">
<CategoricalPredictor name="Occupation" value="Professional" coefficient="1.46855586721193"/>
<CategoricalPredictor name="Occupation" value="Protective" coefficient="2.10303370845987"/>
<CategoricalPredictor name="Occupation" value="Repair" coefficient="0.795620651090366"/>
<CategoricalPredictor name="Occupation" value="Sales" coefficient="0.874412173270209"/>
<CategoricalPredictor name="Occupation" value="Service" coefficient="-0.496502223605296"/>
<CategoricalPredictor name="Occupation" value="Support" coefficient="1.05942018386932"/>
<CategoricalPredictor name="Occupation" value="Transport" coefficient="0.178577748117911"/>
<CategoricalPredictor name="Gender" value="Female" coefficient="0"/>
<CategoricalPredictor name="Gender" value="Male" coefficient="0.375114585003982"/>
</RegressionTable>
```

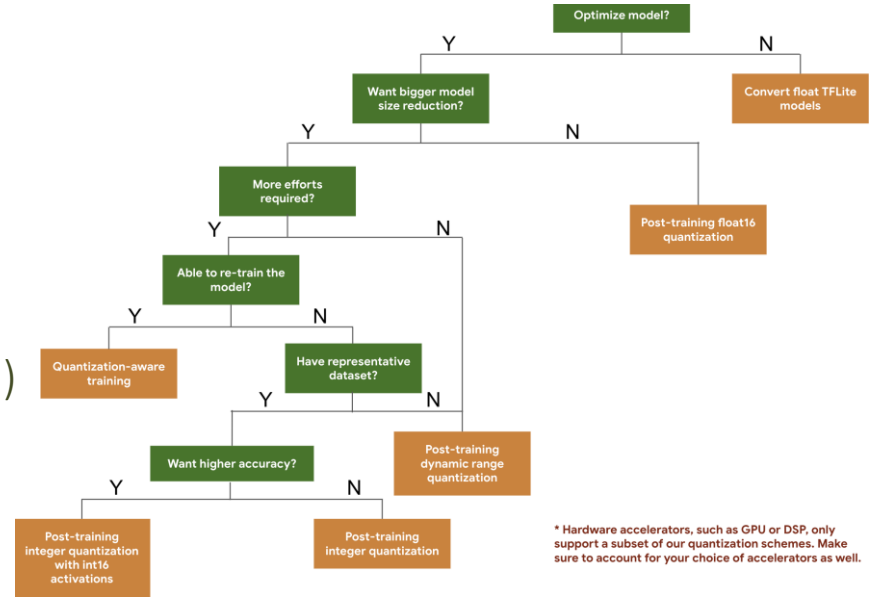
PFA replaces PMML with JSON + control structures & callbacks

```
<xs:element name="PMML">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Header"/>
      <xs:element ref="MiningBuildTask" minOccurs="0"/>
      <xs:element ref="DataDictionary"/>
      <xs:element ref="TransformationDictionary" minOccurs="0"/>
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:group ref="MODEL-ELEMENT"/>
      </xs:sequence>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="version" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

<xs:group name="MODEL-ELEMENT">
  <xs:choice>
    <xs:element ref="AnomalyDetectionModel"/>
    <xs:element ref="AssociationModel"/>
    <xs:element ref="BayesianNetworkModel"/>
    <xs:element ref="BaselineModel"/>
    <xs:element ref="ClusteringModel"/>
    <xs:element ref="GaussianProcessModel"/>
    <xs:element ref="GeneralRegressionModel"/>
    <xs:element ref="MiningModel"/>
    <xs:element ref="NaiveBayesModel"/>
    <xs:element ref="NearestNeighborModel"/>
    <xs:element ref="NeuralNetwork"/>
    <xs:element ref="RegressionModel"/>
    <xs:element ref="RuleSetModel"/>
    <xs:element ref="SequenceModel"/>
    <xs:element ref="Scorecard"/>
    <xs:element ref="SupportVectorMachineModel"/>
    <xs:element ref="TextModel"/>
    <xs:element ref="TimeSeriesModel"/>
    <xs:element ref="TreeModel"/>
  </xs:choice>
</xs:group>
```

ONNX (Open Neural Network eXchange

Most ML libraries provide import and export functionality to this library-agnostic format (open standard).
ONNX-enabled runtime environments provide library-independent functionality to load and inference a model.

Tensorflow Lite



Workflow:

- Transform to a TensorFlow Lite model (small footprint + direct access of data)
- Optionally include meta-data for pre- and post-processing pipelines
- Copy model to device and load it
- TensorFlow Lite interpreter API enables inference support
- Inference API are built via TensorFlow Lite Task Library

Optimizations:

- Quantization: change from 32Bit float to 8Bit integer representation; often comes with some loss of accuracy; reduces size substantially and improves inference time
- Hardware-specific acceleration (e.g., for Edge TPU)
- Pruning: Drop parameters that have only a minor effect on accuracy
- Clustering: Weights in a layer are clustered and the centroid values of the clusters are stored; substantially compresses the model

PyTorch Mobile

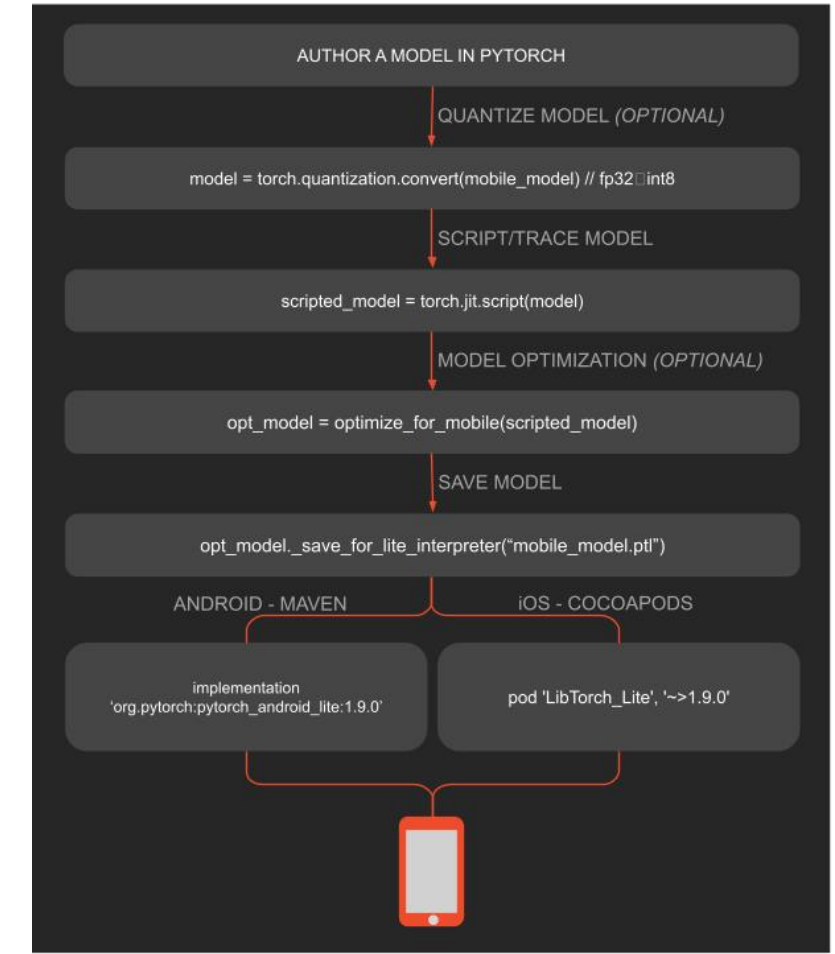
Enables to load and execute a PyTorch model locally in Android, iOS, and Linux

Support for model optimizations for mobile devices, CPU types (e.g., ARM), device-specific interpretation, etc.

Compresses model to lower bandwidth and memory requirements; speed up computations at the cost of accuracy

Encodes the model to torch script to be interpreted and runtime-optimized by the mobile library

Store model and load on the device the model



Topic II: Monitoring

Bindu Reddy 🔥❤️
@bindureddy

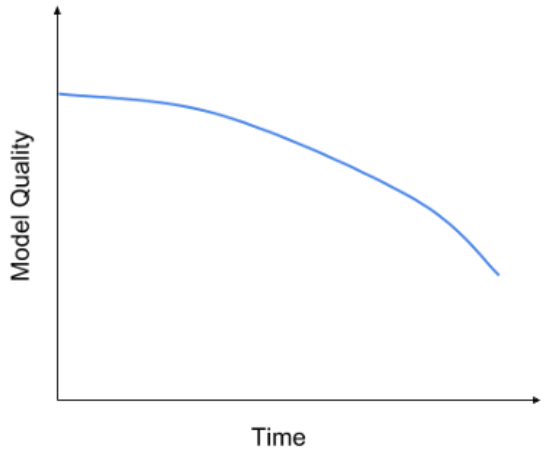
If your model isn't continuously learning, it's experiencing drift and rotting

Data science teams who are doing one-off data cleaning and transformation in python are doing it wrong

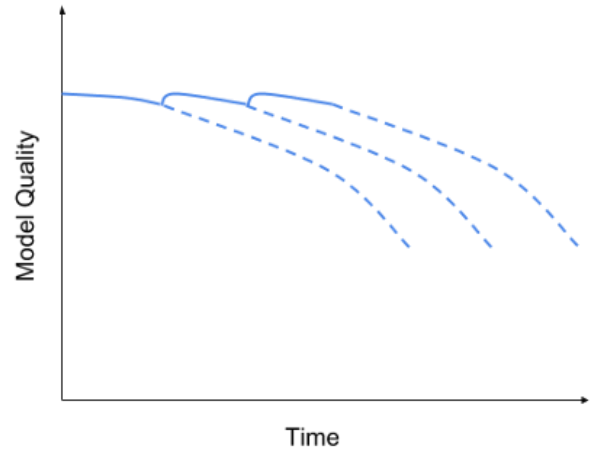
You need data transformation & cleaning pipelines that run and re-train models regularly

2:41 AM · Apr 8, 2021 · Twitter Web App

Model Staleness over time



Refreshing models over time



Monitoring Goals

Enable the flywheel of continuous self-improvement of the AI system

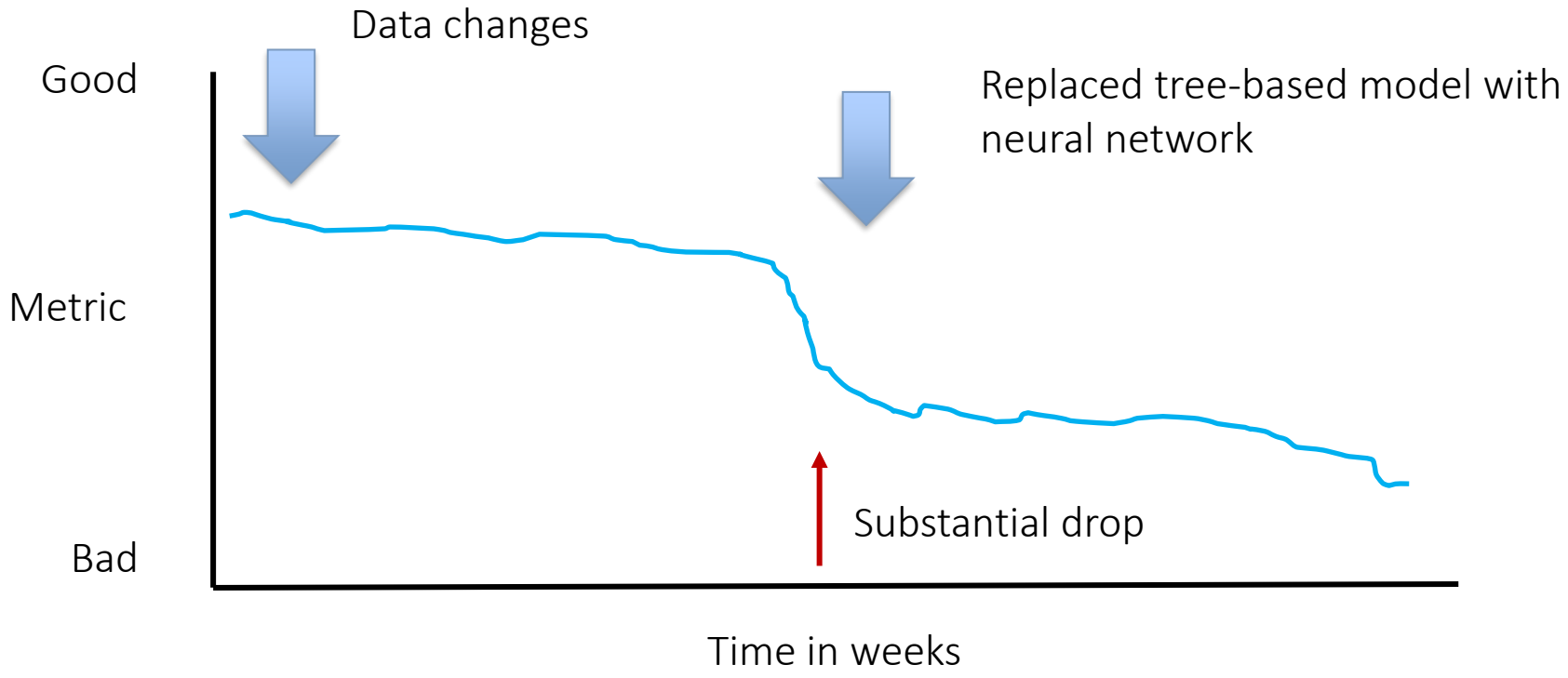
Avoid or recognize system and AI failures

Avoid model deterioration by detecting data and model shifts

Comply to SLAs by monitoring non-functional properties of the system (telemetry)

Obtain an overall health status of the system

Silent Failures



Neural network is more sensitive towards outliers and changes in data than a tree-based algorithm
Actual cause: offline data distribution != online data distribution

Example from Grace Huang, data science manager, Pinterest

Avoid System Failures due to AI Failures

What to do when a feature is hazardous or predictions are entirely wrong?

- Have multiple models in parallel to avoid single model failure
- Have white lists and thresholds to overwrite certain feature problems (e.g., falsely blocking IPs)
- Have simple rules, guidelines, and heuristics to ignore ML model

Policy layer:

- Filter ML output for undesired behavior (e.g., bad word filter)

Out-of-bounce protection:

- Define sensible data ranges and check for those ranges

What to Monitor?

Model metrics

- Prediction distributions
- Feature distributions
- Evaluation metrics (when ground truth is available)

System metrics

- Request throughput
- Error rate
- Request latencies
- Request body size
- Response body size

Resource metrics

- CPU utilization
- Memory utilization
- Network data transfer
- Disk I/O

Monitor ML Accuracy

All relevant metrics aligned with accuracy:

- Click through rate
- Visiting time on a web page
- Duration of a watched video that has been recommended before
- Task completion rate
- Happiness or involvement factors

Monitor ML Predictions

Monitor direct model output to determine drifts and to assess differences in different model variants and versions

Monitor uncommon events, such as all predictions have the same value for a certain time frame

Monitor ML Features

Monitor distribution of all features and compare it with feature distribution of training data

Monitor data schema (i.e., constraints, value ranges, dependencies) and whether it conforms to training data schema

Typical things:

- Val(f1) > Val(f2)
- Dom(f3) is in set of categorical values
- Min, max, avg, median distribution of f4 similar to training statistics
- Word frequencies
- Regex conformance
- Etc.

Tools:

- Great Expectations (see prior slide)
- Pydantic Validators: https://pydantic-docs.helpmanual.io/usage/validators/

```python
from pydantic import BaseModel, ValidationError, validator


class UserModel(BaseModel):
    name: str
    username: str
    password1: str
    password2: str

    @validator('name')
    def name_must_contain_space(cls, v):
        if ' ' not in v:
            raise ValueError('must contain a space')
        return v.title()

    @validator('password2')
    def passwords_match(cls, v, values, **kwargs):
        if 'password1' in values and v != values['password1']:
            raise ValueError('passwords do not match')
        return v

    @validator('username')
    def username_alphanumeric(cls, v):
        assert v.isalnum(), 'must be alphanumeric'
        return v
```

```python
user = UserModel(
    name='samuel colvin',
    username='scolvin',
    password1='zxcvbn',
    password2='zxcvbn',
)
print(user)
#> name='Samuel Colvin' username='scolvin' password1='zxcvbn' password2='zxcvbn'

try:
    UserModel(
        name='samuel',
        username='scolvin',
        password1='zxcvbn',
        password2='zxcvbn2',
    )
except ValidationError as e:
    print(e)
    """
    2 validation errors for UserModel
    name
      must contain a space (type=value_error)
    password2
      passwords do not match (type=value_error)
    """
```

Input Monitoring: Detect Data and Concept Drift

Taken from Chip Huyen:

Type	Meaning	Decomposition
Covariate shift	• P(X) changes • P(Y|X) remains the same	P(X, Y) = P(Y|X)P(X)
Label shift	• P(Y) changes • P(X|Y) remains the same	P(X, Y) = P(X|Y)P(Y)
Concept drift	• P(X) remains the same • P(Y|X) changes	P(X, Y) = P(Y|X)P(X)

Data Drift / Covariate Drift

Data used for training differs compared to data in production: x != x’
Change can be gradual or suddenly.

Causes of data drift:

- Systematic: different environments, different data sources, different preprocessing (e.g., synthesized speech samples vs. real)
- Time: Properties, assumptions, preferences, etc. changed since training
- Selection: Skewed and biased training data did not resemble real data distribution (e.g., bought data from another country)
- Random: Random effects not accounted for during training; Noise in real-world data vs. clean lab environment

Does a test detect this drift?
No! Since the input distribution changes in a data drift, but the test remains the same, we won’t detect it!

Label Drift

Target distribution changes over time, but the input distribution is constant:
For a given output, the input distribution stays the same.

Covariate drift can cause also a label drift, but not always.

Example:

- A new treatment results in more healthy outcomes, but the number of patiens who require treatment and their feature vector stays the same

Concept Drift

Mapping from input to label changed x->y != x->y’

For the same input, we obtain a different target/label, e.g., due to changes in behavior, preferences, laws, environments

Examples: inflation, sudden unemployment rate, catastrophic events

How to detect?

- Monitor deterioration of accuracy
- Update training sets AND labels and retrain

Detecting Distribution Drifts

1. Simple method: compute statistics, such as median, mean, standard deviation, percentiles, etc.
   Problem: Similar statistics can come from different metrics; compte statistics on recent data as a countermeasure

2. Compare distributions: Metrics and statistical tests
   Kolmogoro-Smirnov test; Anderson-Darling test; Pearson‘s correlation

3. Consider time intervalls for shifts
   Seasonal effects are harder to detect (manually plotting distributions might work)

Closing the Flywheel: Challenges of User Labels

Synchronization between query and feedback:

- Time delay between prediction and feedback
- Requires caches, identifiers, etc.

Long feedback cycles:

- Examples: fraud detection, real-world actions (agriculture)

Implicit feedback:

- What happens when no recommendation was clicked or too late?

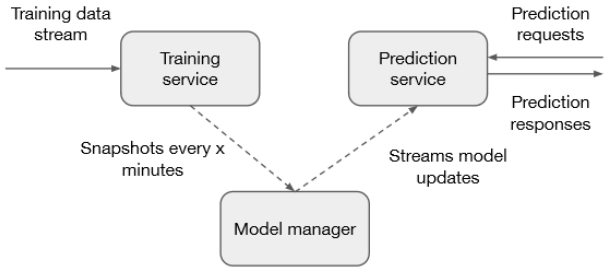
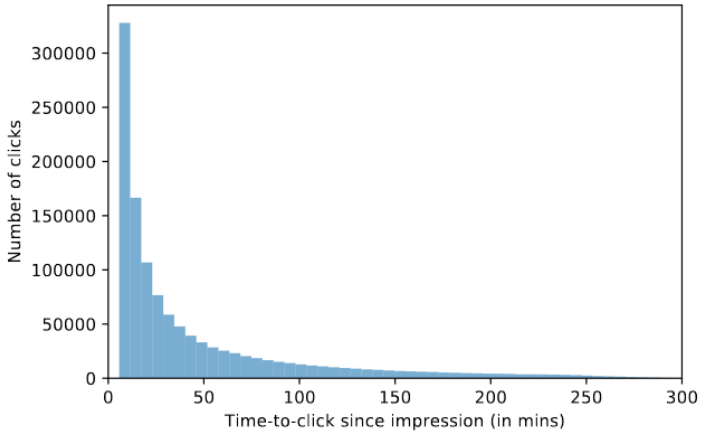
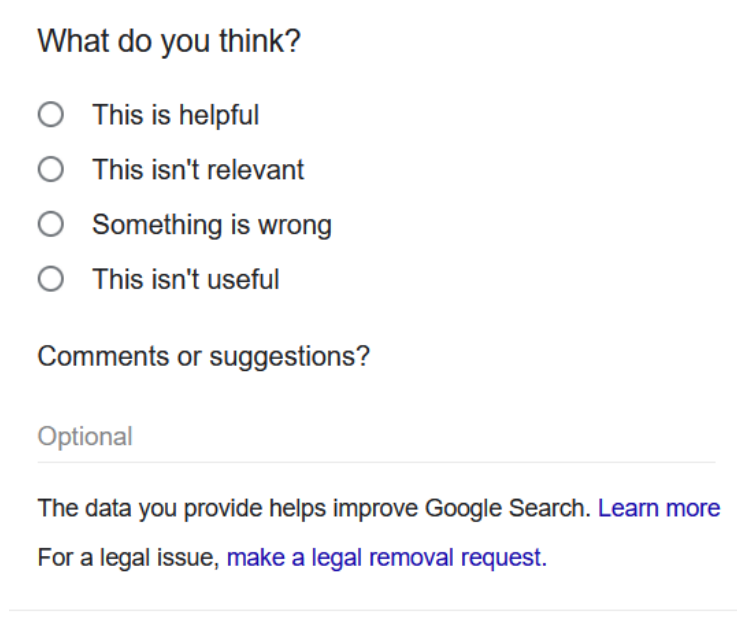
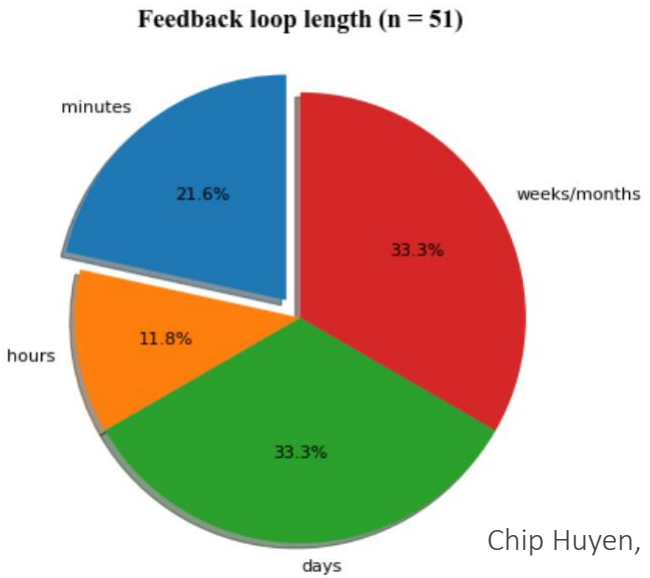


Figure 3: Continuous training framework.







Chip Huyen, 2022

Degeneration of Flywheel Performance

Problem: User feedback may enforce certain directions of the model so that small differences in probabilities of predictions become large differences; problem exists only in production and can not be spotted in training

Effect: Reduced variation of predictions, homogeneous results, limited software functionality/features

Examples: Pre-selection of applications based on certain features (e.g., university); app shortcut recommendations; music recommendations;

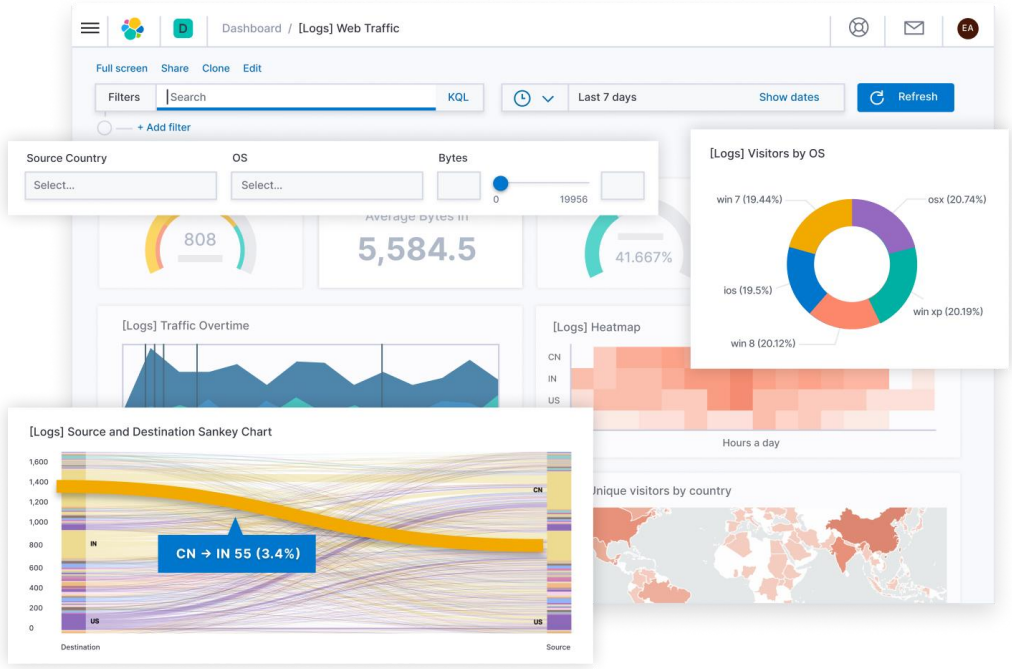
Counter measures:

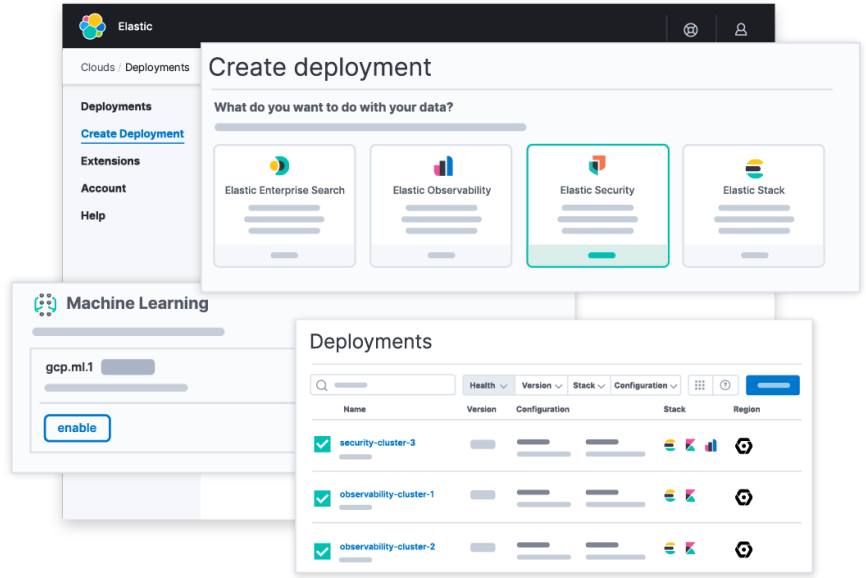
- Feature importance monitoring (watch for the emergence of too predictive features)
- Random elements (recommend random or heuristic-based items; retrain with random picks)
- Encode positional / GUI features: create a feature that encode the prominence of recommendation (e.g., ranking number or size) and use it for retraining, but set it to false for inference; idea is to attribute the impact of the position not to the original features, but to the position feature, but ignore this feature for prediction

Monitoring Tools

Similar to monitoring tools for microservice architectures, for instance, Elastic Stack (ELK)

- ElasticSearch (search- and analysis engine)
- Logstash (data processing pipeline)
- Kibana (visualization of data)
- (optional) Beats (sharing of data)





Resources

- https://christophergs.com/machine%20learning/2020/03/14/how-to-monitor-machine-learning-models/
- https://www.shreya-shankar.com/rethinking-ml-monitoring-1/
- https://huyenchip.com/2022/02/07/data-distribution-shifts-and-monitoring.html