Mastering ML-Systems in Production

Herausforderungen beim Betrieb von ML-Systemen

regiocom Energie

Der Meter Reading Recognition Service (MRRS)

- Hochspezialisierter Dienst zum **Erkennen von Zählerständen** in Fotos von Ferraris-Zählern und modernen Messeinrichtungen
- **Typische Mängel**, wie Unschärfe, Schatten/Dunkelheit etc. werden berücksichtigt
- Unterstützt die Sparten **Strom**, **Gas** und **Wasser**
- Zuverlässige Erkennung **drehender Ziffern**
- **Validierung der Werte** durch die Endkunden dank nahtloser Integration in Selfservice-Portale möglich

Anwendungsbeispiel „Aufforderung per Postkarte“

1. Ablesekarte mit QR-Code/Link zur Webseite
2. Fotoupload
3. Bestätigung
4. Fertig ☺

regiocom Energie

In 3 Schritten vom Zählerfoto zum Zählerstand

Am Anfang steht das Bild…

1

Erkennen der sog. ***Region-of-Interest (RoI)***, also dem Zählwerk bzw. dem Display bei modernen Messeinrichtungen

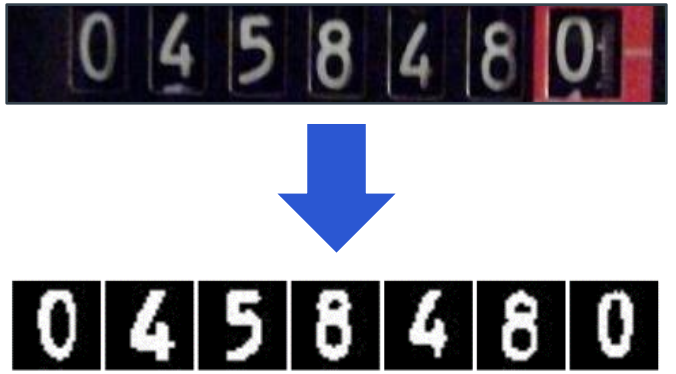
- Segmentierungsaufgabe zum Eleminieren unnötiger Bildpunkte
- Bei analogen Mehrtarifzählern werden mehrere RoI erkannt
- Identifikation der Nachkommastellen, um diese entweder weiterzugeben oder abzuschneiden
- Ermitteln der durchschnittlichen Helligkeit, um diese ggf. auf einen mittleren Normwert zu korrigieren (Gamma-Korrektur)
- 4 sparten- und typspezifische YOLOv5-Modelle mit ca. 99% Genauigkeit
  - Vortrainiert mit dem COCO-Datensatz (ca. 330.000 Bilder)
  - Training mit ca. 1.250 Datensätzen (*Fotos*) im Verhältnis 4:1 (Trainingsmenge zu Validierungsmenge)

In 3 Schritten vom Zählerfoto zum Zählerstand

…dann kommen die einzelnen Ziffern und zum Schluss der Zählerstand!

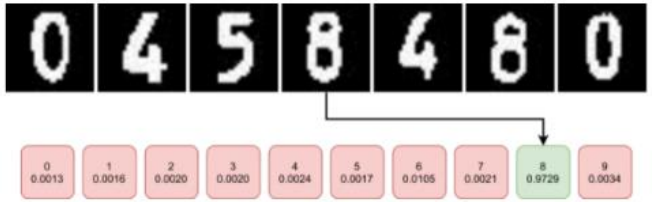
2 Erkennen der sog. ***Area-of-Digits (AoD)**, also der Ziffernbereiche*

- Segmentierungsaufgabe als Grundlage für die Ziffernklassifikation
- Entfernen von AoDs mit zu geringem Abstand (meist Blitzlichtreflexionen)
- Zusammenführen überlappender AoDs bei drehenden Ziffern
- 4 sparten- und typspezifische YOLOv5-Modelle mit ca. 96% Genauigkeit
  - Training mit ca. 520 Datensätzen (*Fotos*) im Verhältnis 4:1



3 Erkennen der ***einzelnen Ziffern**, indem jeder ausgeschnittenen Ziffer (AoD) eine Zahl zugeordnet wird*

- Klassifizierungsaufgabe der Ziffern von 0 bis 9
- Unterscheidung zwischen vollständig-sichtbaren und drehenden Ziffern
- 4 sparten- und typspez. ResNet-Modelle für sichtbare Ziffern (ca. 99,6% G.)
- 1 ResNet-Modell für drehende Ziffern mit ca. 99,3% Genauigkeit



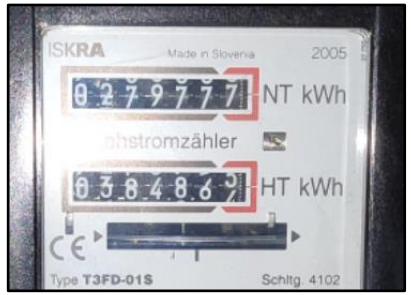
regiocom Energie

2 kleine Hilfsdienste für einen reibungslosen Ablauf

…denn wir müssen noch mehr erkennen!

Erkennen der ***Sparte*** und des ***Zählertyps*** (analog oder digital)

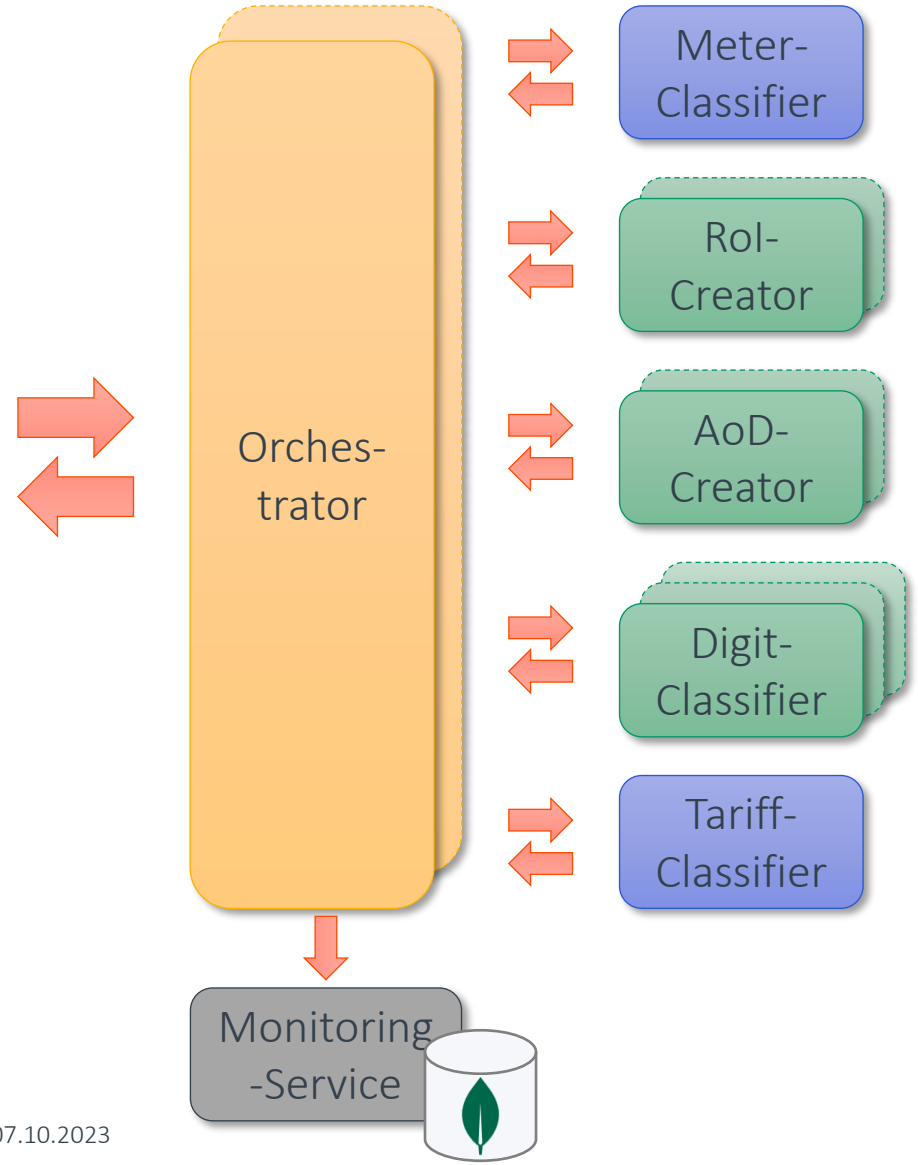
- Klassifizierungsaufgabe als Grundlage für die weitere Verarbeitung
- Aufrechtdrehen schräger Wasserbilder, um die Erkennungsrate zu verbessern
- 1 übergreifendes ResNet-Modell mit ca. 99,8% Genauigkeit
- 1 ResNet-Modell zum Drehen von Wasserzählern mit ca. 98% Genauigkeit



Erkennen der ***Tarife*** bei Stromzählern

- Segmentierungsaufgabe (11 trainierte Label wie „HT“, „1.8.0“ oder „Lieferung“)
- 2 YOLOv5-Modelle (eines für jeden Zählertyp) mit ca. 89% Genauigkeit

Die Architektur hinter dem MRRS



Hochskalierbare, container-basierte Microservice-Architektur

Die Dienste

- kapseln die Modelle für eine spezifische Aufgabe
- arbeiten autark
- skalieren unabhängig voneinander und bedarfs-gerecht (*lastabhängig*)

Der Orchestrator

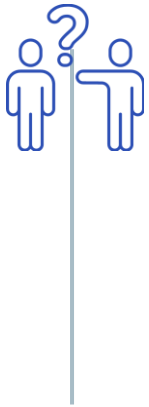
- hält und bearbeitet alle eingehenden Requests
- koordiniert das Zusammenspiel der Dienste
- skaliert eher progressiv

Der MonitoringService

- übernimmt das fachliche Logging der Ergebnisse (inkl. der eingehenden Bilder)

Hinaus in die Betriebsumgebung … Welcher Cluster?

„I works on my machine!“ … „well, then we ship your machine to the customer!“



Grundlegende Entscheidung (wenn cloud native entwickelt wurde) ist somit immer:

- On Prem (aber dann bitte im Cluster!)
- Hyperscaler(s)
- Hosting („cloud is just other peoples computer“)

- Datenschutz (sind beispielsweise personenbezogene Nebeninformationen im Bild erkennbar?)
- Auftragsdatenverarbeitung klären, Auftraggeber muss es erlauben (wir sind Dienstleister!)
- Weitergabe an Dritte müsste in AGB mit dem Kunden abgestimmt sein
- Letztendlich sind aber die Betriebsstrukturen bei regiocom vorhanden und durch Verträge abgedeckt

wir machen es bei uns

Unsere Container-Betriebslandschaft

Irgendwann hat jeder mal mit einem k8s Cluster angefangen

State	Name	Provider	CPU	Memory	Pods
Active	dt-ci	Custom RKE	40 cores	202 GiB	14/1430
Active	dt-prod	Custom RKE	50 cores	249 GiB	8/1760
Active	dt-staging	Custom RKE	40 cores	187 GiB	0/1320
Active	dt-test	Custom RKE	46 cores	202 GiB	8/1430
Active	gpu-cluster	Custom RKE	16 cores	236 GiB	10/220

- 4 Stages (plus Sonder-Cluster wie GPU, Sandbox etc)
- Eingebettet in Vsphere Bereitstellung (damit „ohne Schrauben“ anpassbar … alles außer ~~Tiernahrung~~ GPU)
- Gitlab basierte CI/CD Strukturen (Terraform + helm, alternativ auch Pulumi)

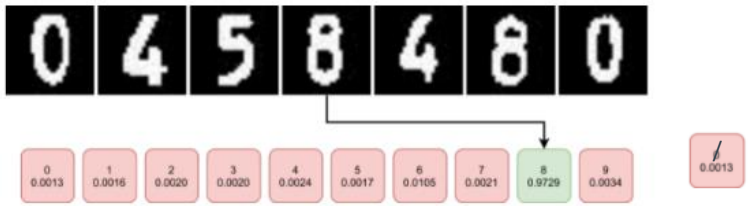
SRE Group als zentrale Organisationseinheit dafür, Ansprechpartner und irgendwie immer zuständig… wo nötig, sind die dann natürlich mit Firewall-Admins etc. vernetzt

- Large Filesystem im GitLab, Interne Container Registry
- Damit können wir auch „dicke Images“ bauen, was den Start der Container ohne Abhängigkeiten ermöglicht

Herausforderungen beim Deployment von ML-Modellen

Cluster ist doch im 21. Jahrhundert Standard, warum soll es mit ML plötzlich anders sein?

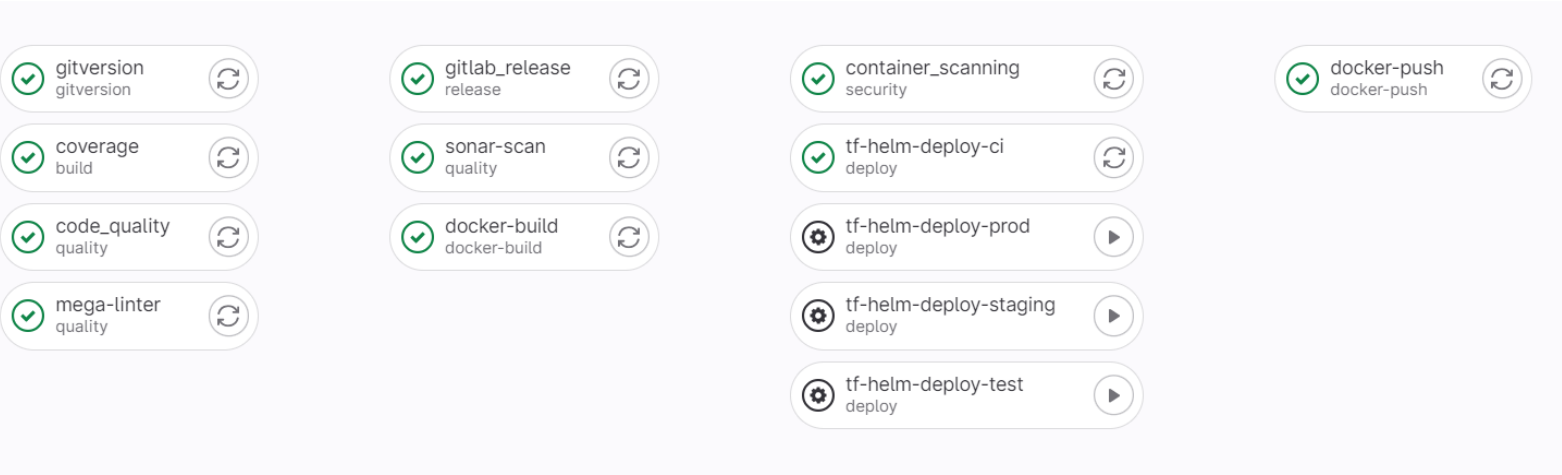
- Modelle sind inhärenter Teil der Programmlogik
- Classifier-Outputs transportieren Semantik, die dann gegebenenfalls auch wieder die Logik beeinflusst
- Modelle sind nicht `diff`-bar
- Modelle entwickeln sich weiter und haben Versionsnummern (das muss man ggf. selber machen, siehe auch MLOps)



- Primäre Optionen:
  - Deployment der Modelle als Blob im Source Tree
  - Deployment als extra Ressource, extra Service
  - Download von getrenntem Hub, Caching etc.

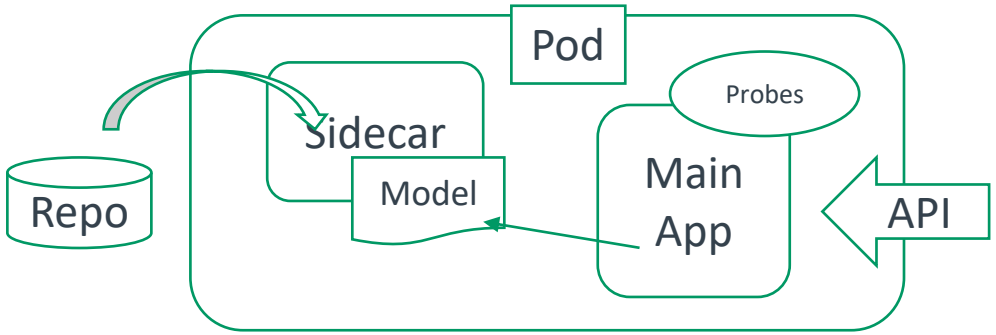
Consider Rollbacks, not Deployments

„Wenn es sich schnell ändern kann, will ich es auch schnell zurücknehmen können!“



- Wir installieren natürlich Quality Gateways
- Wir prüfen natürlich, bevor wir deployen
- Wir machen dennoch Fehler (selbst ohne Statistik im Spiel)
- Lasst uns keine permanenten Fehler machen und lasst uns die Chance haben, diese zurückzunehmen

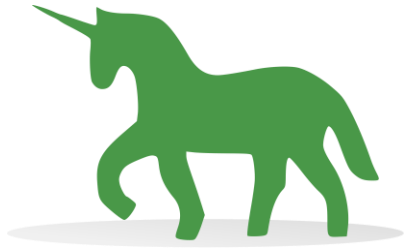
- Versioniert Eure ML-Modelle, auch hier ist SemVer sinnvoll
- Sorgt dafür, dass alte Versionen namentlich ansprechbar sind („ich glaube, vorgestern ging‘s noch“)
- Wenn Ihr den Sourcecode vom Versionsverlauf der Modelle entkoppeln wollt, nutzt *Sidecar Containers*



regiocom Energie

Absichern der Laufzeitumgebung

Denn nichts ist so, wie es im Labor mal war



- Ressource Limits mit Throttling
  - Auch weil Kostenkontrolle
  - Angriffsszenarien mit Ressource Exhaustion
  - Nicht vergessen: Inferenz meist in CPU!
- Aufgaben delegieren (kümmert Ihr Euch um ML!)
  - Access Control in die Middleware (oder über BFF)
  - Pay per Use in die Middleware (wenn vorgesehen)
  - Transportkontrolle in ein Service Mesh (wenn vorhanden)
- Angriffserkennung
- Containerscanning, CVE Datenbank
- Perimetersicherung

- Ein Python Standard-Webservice Flask ist keine Produktivumgebung – die Empfehlung ist ein vollständiger WSGI Stack
- Man kann auch in der Anwendung ein Multi-Worker-Modell fahren (gunicorn workers)
- Wenn es Eure Anwendungslogik erlaubt, dann nutzt –preload (shared memory) und reduziert Containergrößen etc. … *Ausprobieren!*

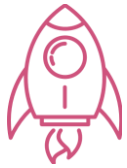
- Üblich ist

  #Workers ~ #Container-Instanzen

regiocom Energie

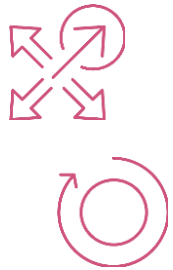
Überraschungen während der Laufzeit

„Vereinbarungen sind dazu da, sie auf die Probe zu stellen“



Was kann Dir in der Praxis passieren

- Das Frontend „vergisst/schafft nicht“, die Bilder zu skalieren
- Der Hersteller der Mobiltelefone denkt sich einen neuen Codec aus
- Ggf. interpretiert Eure Grafik-Bibliothek plötzlich die Meta-Daten und dreht die Bilder (ein weiteres Mal)
- Nachtrainieren verändert ggf. das Laufzeitverhalten

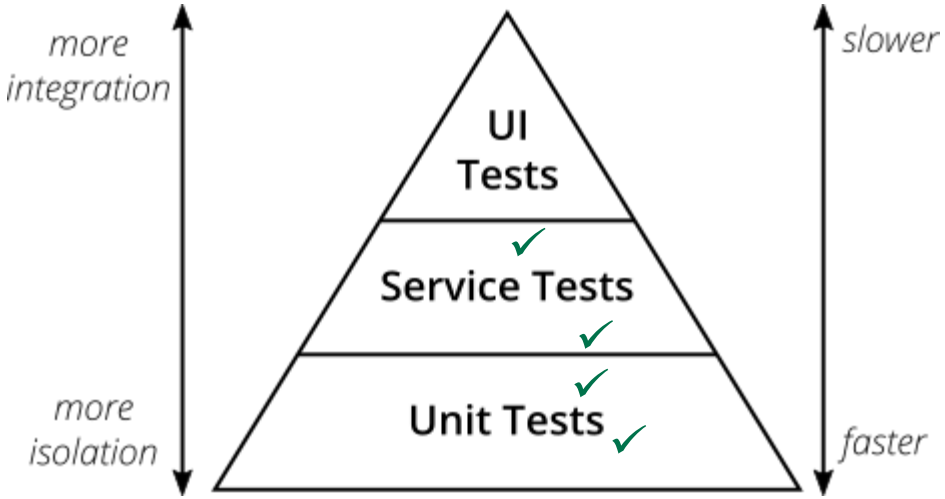


Monitoring *mit wiederholbarer* Protokollierung

- NoSQL DB mit *originalem* Anfrageobjekt (MongoDB, Azure Cosmos, S3 …)
- TTL setzen, keine sekundäre Speicherung im Filesystem oder so (oder der DSB kommt Euch holen)
- Gerne auch in *getrennter* Collection länger speichern, was wir geantwortet haben
- Aber: Monitoring ist *kein Mittel zur Abrechnung*!

Tests, ein Weg, um Überraschungen zu minimieren

Jetzt könnte es etwas trocken werden …



(Quelle: martinfowler.com)

- Frontend-Tests: der jeweilige Integrationspartner
- REST-Schnittstelle: OWASP Zap
- Inhaltliche Abdeckung: Accuracy-Test, Retraining KPI
- Contract Testing: bislang nur Load/Performance Testing
- Functional/Servicetests: Pytest gegen Flask-Testclient (in IDE + CI/CD)
- Unit-Tests: PyTest (in IDE und CI/CD)

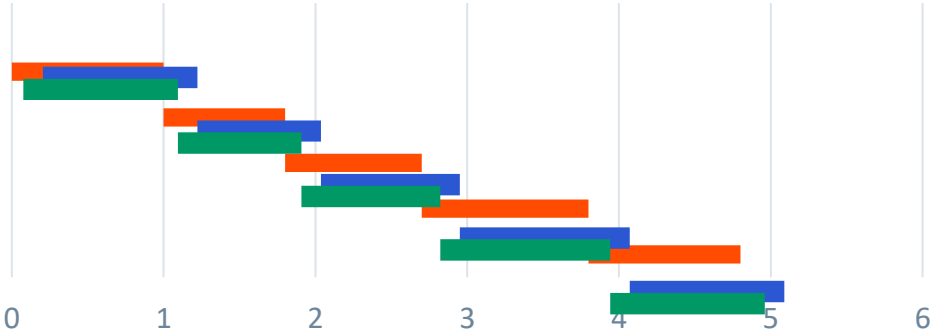
- Wir messen die statistischen Kernparameter des Gesamtproduktes in situ
- Wir prüfen Resilienz und Robustheit der gesamten Dienstkonstellation in einer produktionsgleichen Testumgebung

- Wir prüfen die Abdeckung des Dienste-Vertrages jedes Microservices
- Wir testen die Einzelfunktionen, insbesondere auch den nicht-ML-Code mit ~100% Coverage

regiocom Energie

Last- und Performancetests bei langläufigen ML-Modellen

Wie schnell schaffen wir es und wie lange können wir durchhalten?



- Aus dem Cluster in den Cluster, wir nutzen dabei mehrere Testszenarien
  - repräsentativ sortenrein (z.B. nur analoge Stromzähler),
  - repräsentatives Leistungsspektrum (alle Sparten, analog und digital),
  - wohldefiniertes Einzelbild (Eliminierung inhaltsbezogener Einflüsse)

- Sequentielle Abfrage
  - Warten mit dem Senden des nächsten Bildes, bis das vorhergehende Bild verarbeitet wurde
  - Prüfziel: 100% Erfolg, Messung der Average, Median, Min und Max Ausführungszeiten

- Parallele Abfrage
  - Simulation mehrerer rücksichtsloser virtueller User mit parallelen Abfragen (wie separate Sequenzen)
  - Prüfziel: Bestimmung des Beginns der Skalierung, Bestimmung des Beginns von Timeouts oder Drops

- Anforderung des Tests durch Trigger in der Pipeline, somit jederzeit manuell zu starten, als organisatorisches Quality Gate vor dem Produktivdeployment und/oder zeitgesteuert

Grenzen eines Deployments prüfen

Nur gucken, nicht anfassen!

```
export const options = {
  stages: [
    { duration: '1m', target: 1 },
    { duration: '1m', target: 3 },
    { duration: '1m', target: 5 },
    { duration: '1m', target: 7 },
    { duration: '1m', target: 9 },
    { duration: '1m',
    { duration: '1m',
    { duration: '1m',
    { duration: '1m',
  ],
  thresholds: {
    http_req_failed: [
    http_req_duration:
    http_req_duration:
  },
  noConnectionReuse: t
  userAgent: 'MyK6User
};

export default functio
  const fd = new FormD
  fd.append('image', h

  const res = http.pos
    headers: { 'Conten
  timeout: "10s"
  });
  check(res, {
    'is status 200': (
  });
}
```

Kein einziger Pod musste bei diesem Test neu gestartet werden – alles lief durch

State	Name			Ready	Up To Date	Available	Restarts	Age
Active				1/1	1	1	0	237 days
Active			42	1/1	1	1	0	327 days
Active			das/meterclassifierservice:1.1.45	1/1	1	1	0	327 days
Active	mlvisionpresentation	meter-reading-recognition	das/mlvisionpresentation:0.1.112	1/1	1	1	0	238 days
Active	monitoringservice	meter-reading-recognition	das/monitoringservice:1.0.27	1/1	1	1	0	26 days
Active	orchestrator	meter-reading-recognition	das/orchestrator:1.0.104	1/1	1	1	0	237 days
Active	roicreatorservice	meter-reading-recognition	das/roicreatorservice:1.1.28	1/1	1	1	0	263 days
Active	tariffclassifierservice	meter-reading-recognition	das/tariffclassifierservice:0.1.65	1/1	1	1	0	264 days

```
checks.....
data_received....
data_sent.....
http_req_blocked
http_req_connec
http_req_duration
  { expected_resp
http_req_failed..
http_req_receivin
http_req_sending.
http_req_tls_hand
http_req_waiting.............: avg=3.42s    min=0s       med=3.13s    max=9.74s    p(90)=4.94s    p(95)=5.64s
http_reqs.....................: 1057   1.924011/s
iteration_duration............: avg=4s       min=381.92ms med=3.73s    max=10.62s   p(90)=5.49s    p(95)=6.15s
iterations....................: 1057   1.924011/s
vus...........................: 1      min=1      max=17
vus_max.......................: 17     min=17     max=17
```


regiocom Energie

Täglicher Accuracy-Test

Because all machine learning models degrade

- Baseline zur Erkennung von Veränderungen, Prüfung gegen „das Leistungsversprechen“ [TM]

Wir ändern uns

- Unser Code verändert die Verarbeitungsschritte
- Neue Library-Versionen ändern das Verhalten
  - Grafik-Libs
  - ML-Libs
- Unsere Modelle berechnen etwas anders als vorher

Die Umgebung ändert sich

- Installation anderer Zähler-baureihen
- Vorverarbeitende Portale und Apps stellen für uns auch „Umwelt“ dar
- Hier sind wir exponiert und wollen genau die Probleme erkennen, an die wir noch nicht proaktiv gedacht haben

Ggf. Anpassung der Baseline und/oder Nachtrainieren

Es wird eine kuratierte Menge von Datensätzen genutzt, um die Auswirkungen unserer Änderungen (bewusste oder unbewusste) erkennbar zu machen. Wir kennen die Testergebnisse und bestimmen jede Nacht den Anteil der erfolgreich und korrekt ausgelesenen Zählerbilder.

Täglicher Accuracytest

A test per day keeps the bugs away!

Testbedingungen (Stand 8.5.23):

- Nebenwirkungsfreier, sequentieller Test
- 2114 Bilder Bestandteil des Prüfdatensatzes
- 1910 werden erfolgreich ausgelesen, be… …n die Ablehnungskriterien individuell evaluier…
- Das ergibt eine Accuracy von 90,35% -> *Marketingaussage* und nicht als solche…
- Es gibt keine Nutzerinteraktion, der Tes…
- Laufzeitmessung ist ein zweiter KPI, da… Leistungsversprechens

Todos:

- Anwendung auf Feature Branches erweitern
- Bildung einer Custom Metric für Prometheus

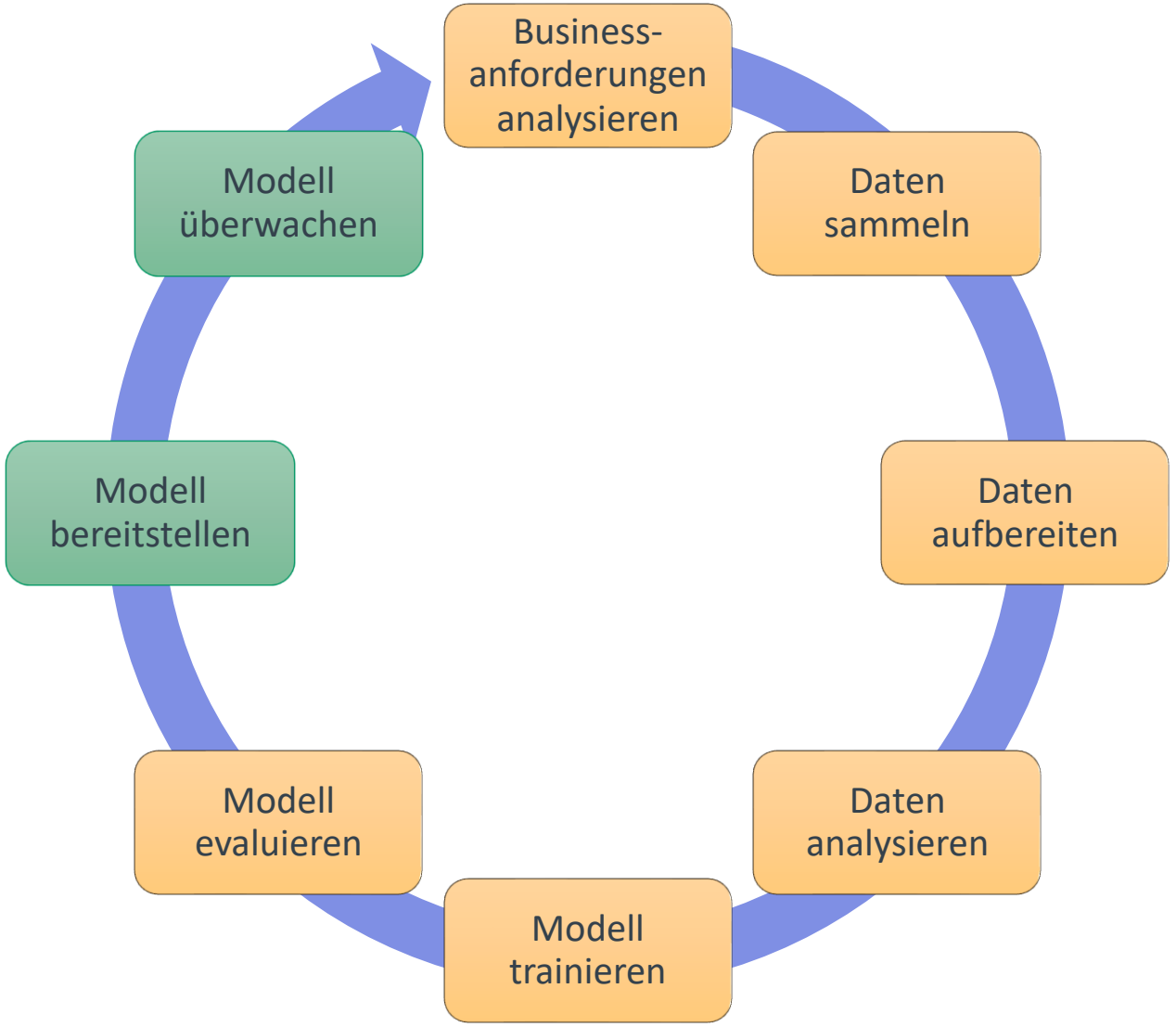
Konsequenzen

- Keine Prüfung auf komplexe Wechselwirkungen (bewusst so gewollt)
- Änderung der Baseline ist ein echter …hange-Prozess und ist Aufgabe des …duct Owners
- …h eine Verbesserung der …ennungsquote muss beachtet werden

…r Test liefert Handlungssicherheit

Und wenn wir etwas an unseren Systemen ändern müssen, dann sind wir bei der Erweiterung gegenüber der klassischen Programmierung und müssen Nachtrainieren

Exkurs: Der ML-Lebenszyklus



- Zyklus kombiniert DataScience mit dem Betrieb
- Die Datenarbeit (Sammeln, Aufbereiten und Analysieren) beansprucht i.d.R. die meiste Zeit
  - Bestimmen der Quellen
  - Kombinieren von Daten aus versch. Quellen
  - Eliminieren von Duplikaten
  - Entfernen von verrauschten Datensätzen
  - Annotieren der Datensätze
  - Etc.
- „Saubere Daten“ sind essentiell für alle nachfolgenden Schritte → *„Shit in, shit out“*

Rücksprünge bspw. bei unzureichenden Modellen sind möglich

Warum muss überhaupt nachtrainiert werden?

„Nichts ist so beständig wie der Wandel.“ [Heraklit von Ephesus, 535-475 v. Chr.]

Allgemeine Gründe für das Nachtrainieren von Modellen

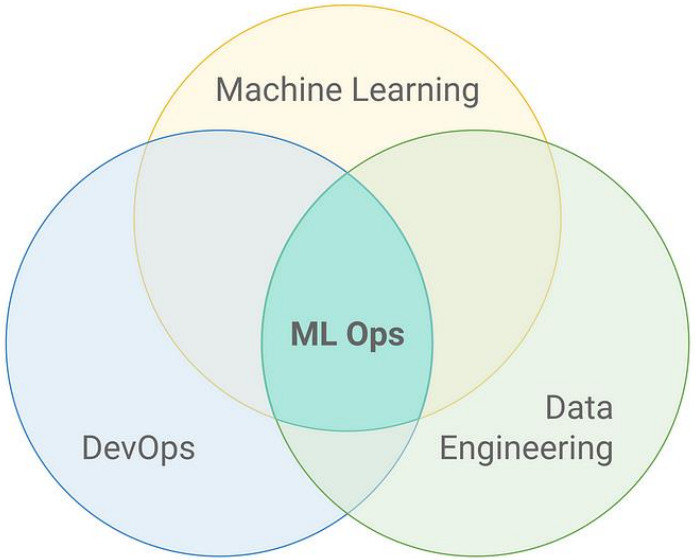
- Im Training wurden nicht alle Eventualitäten berücksichtigt (z.B. Modell benötigt gut ausgeleuchtete Fotos, der Kunde erstellt diese aber im Mondschein bei Nacht…)
- „Concept drifts/data drifts“, wenn sich die Bedingungen für das Modellziel ändern (bspw. wird ein Prognosemodelle zum erwarteten Strom- und Gasverbrauch vor dem Ukrainekrieg heute vermutlich schlechtere Ergebnisse abliefern)
- Die Daten sind veraltet (In der Welt von ChatGPT ist Angela Merkel die Bundeskanzlerin von Deutschland)

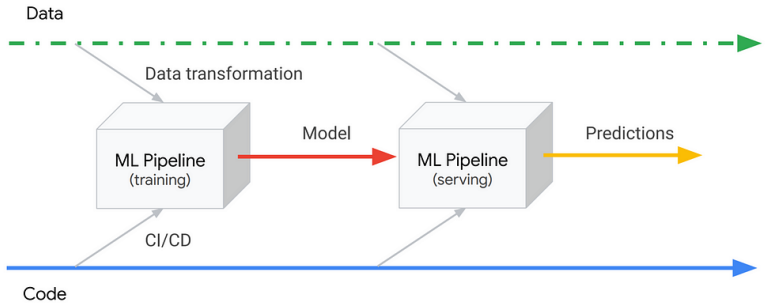
Gründe für den MRRS

- Unbekannte Zählertypen durch neue Mandanten/Netzgebiete, Zählerwechsel etc.
- Unbekannte Fotomotive, wie Ablesekarten oder Hausmeisteraushänge oder…
- Geänderte Kameratechnik (Formate, Belichtung, Effekte, Auflösung etc.)
  - **Aktuelle Lösung:** Inputvalidierung mit entsprechenden Gegenmaßnahmen

MLOps zur Bereitstellung und Wartung von ML-Modellen

Machine Learning = Quellcode und Daten!





"ML Ops is a set of practices that combines Machine Learning, DevOps and Data Engineering, which aims to deploy and maintain ML systems in production reliably and efficiently."
[Christiano Breul, 2020]

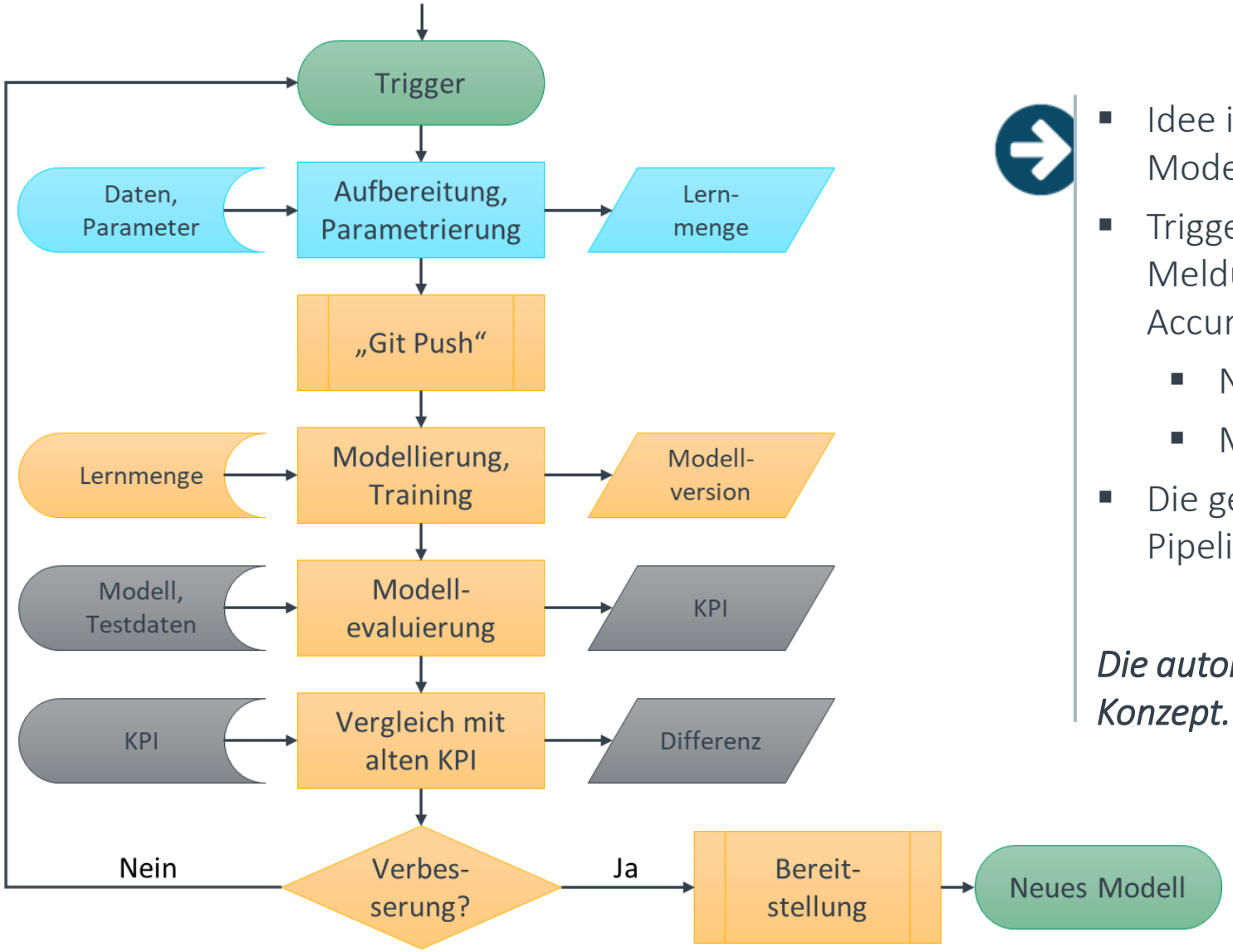
MLOps-Methoden

- Hybride Teams mit Fachexperten aus jedem Bereich
- ML-Pipelines, in der Regel zwei:
  - Datenaufbereitung und Modelltraining
  - Datenvorverarbeitung und Modellanwendung
- Modell- und Datenversionierung (kombiniert, mit Metadaten)
- Modellvalidierung (analog zur Testautomatisierung bei DevOps)
- Datenvalidierung (entspricht den klassischen Unittests)
- Monitoring

regiocom Energie

Quelle: https://towardsdatascience.com/ml-ops-machine-learning-as-an-engineering-discipline-b86ca4874a3f

ML-Pipeline zum Nachtrainieren der MRRS-Modelle



- Idee ist eine automatisierte Bereitstellung neuer Modellversionen als Teil von MLOps
- Trigger für die manuelle Modellerstellung ist eine Meldung aus dem Betrieb oder dem täglichen Accuracy-Test
  - Notifikation via MS Teams oder Mail
  - Metrik muss noch festgelegt werden
- Die gelben Schritte sollen als Jobs in einer CI/CD-Pipeline in unserem GitLab laufen

Die automatisierte Pipeline existiert bislang leider nur als Konzept. Es läuft eine Abschlussarbeit zu dem Thema.

Unsere Lessons Learned

- Machine Learning (ML)-Modelle können viel Ressourcen benötigen → Beim Betriebskonzept berücksichtigen!
- ML wird häufig bei Unsicherheiten eingesetzt, daher ist eine kontinuierliche Überwachung der ML-Applikationen unerlässlich für einen erfolgreichen Einsatz (MLOps)
- Auch ML-Modelle bedürfen einer Pflege und Wartung → Die Datenarbeit hört nach der Produktivsetzung nicht auf!

Prämissen für den Erfolg:

- ML-gerechte Laufzeitumgebungen berücksichtigen die Ressourcenbedarfe (bspw. bei Skalierungsanforderungen)
- Ein fachliches Monitoring ist wichtig für den Betrieb von ML-Anwendungen
- Praxisnahe Tests (Last, Performance, Ergebnisqualität etc.) erfordern Fachwissen und kuratierte Daten
- *Gute Daten sind das Gold der Branche – trotz aller Versuche die Trainingsmengen zu reduzieren*

Vielen Dank für Ihre Aufmerksamkeit

www.regiocom.com

Mario Wolframm,
Marcus Petersen

KI-Anwendungen

mario.wolframm@regiocom.com,
marcus.petersen@regiocom.com