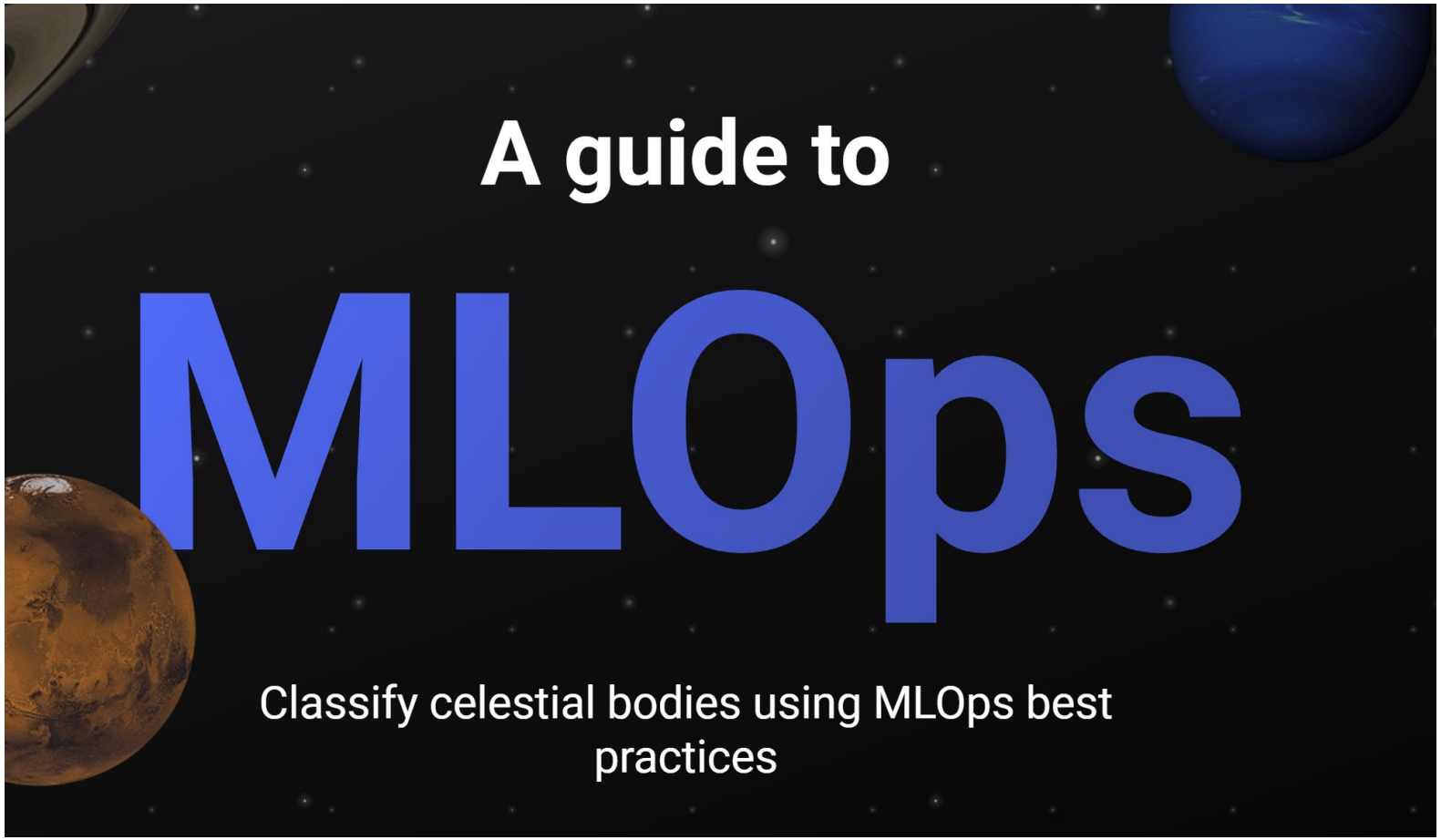
Machine Learning and Data in Operation

Bertil Chapuis

Beat Wolf

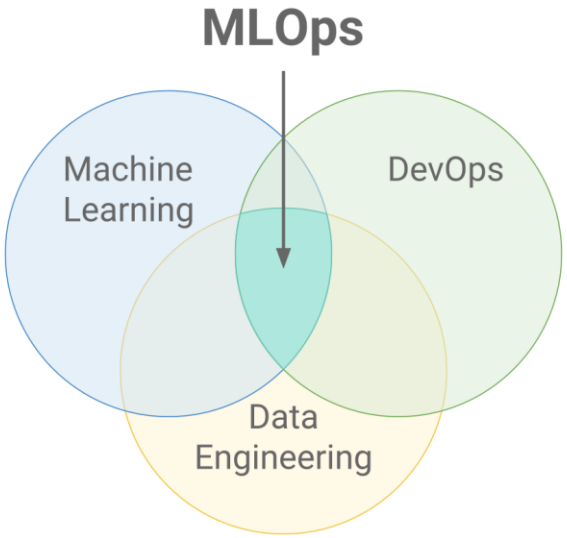
Discussion

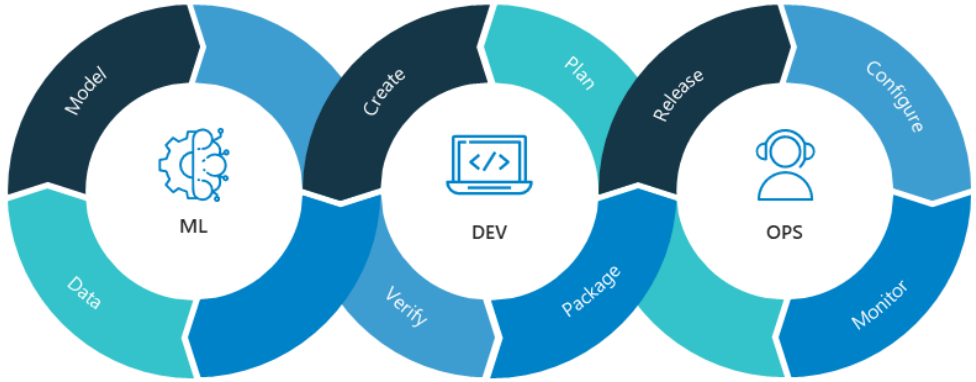

A guide to
MLOps
Classify celestial bodies using MLOps best practices

MLOps

MLOps overview

- MLOps streamlines and automates the **development**, **monitoring** and **management** of machine learning models in production to ensure **reliable** and **scalable** operations
- It is not a technology, but a philosophy
  - What exactly an MLOps approach is, can vary for every project





MLOps readiness level

- Microsoft defines 5 levels of MLOps maturity
  - 0: No MLOps
    - Manual builds, trainings and testing, no centralized tracking of model performance
  - 1: DevOps but no MLOps
    - Automated builds and tests, but still manual training, testing and tracking
  - 2: Automated training:
    - Automated model training, centralized tracking of model performance, model management, easy to reproduce models, manual but low friction releases
  - 3: Automated model deployment
    - Integrated A/B testing, full traceability from deployment to original data, low friction releases
  - 4: Full MOPs automated operations
    - Automated model training and testing, verbose centralized metrics for deployed model, full system is automated, production systems help to improve models

https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/mlops-maturity-model

MLOps readiness level, how?

- How we achieve a certain readiness level depends on the project
  - The technologies might be different
  - The priorities migh vary
    - Is it more important to monitor models in production or being able to A/B test new models?
  - The way to MLOps can be gradual
- We will now discuss some options in terms of
  - Infrastructure / Technology
  - Model optimisations for production
  - Model/Data drift
  - Monitoring
  - Continous learning

Infrastructure

Infrastructure

- When developing applications with an MLOps approach, there are many options for:
  - Infrastructure
  - Technologies
- Depending on the project requirements the right choice might be different
- An MLOps system also has various parts, with their individual needs
  - Data storage/Versioning
  - Model training/Experiments
  - Serving models in production

Infrastructure options

- *What are the infrastructure options we have?*

- *What are the advantages and disadvantages of those infrastructure options?*

Infrastructure options

- *What are the infrastructure options we have?*
  - Cloud based (AWS, Azure etc.)
    - Custom deployment or services
  - On-premise servers
  - Edge computing (browser, app, dedicated hardware)
  - A mix
- *What are the advantages and disadvantages of those infrastructure options?*
  - Cost
  - Privacy
  - Scalability
  - Technological complexity

Cloud

- Often, MLOps application are deployed into a public cloud
  - VM, Docker images etc.
- The advantages are well known
  - Reduced technological complexity
  - "Infinite" scaling
  - Low initial costs
- Some of the platforms even propose MLOps specific infrastructure

Cloud services

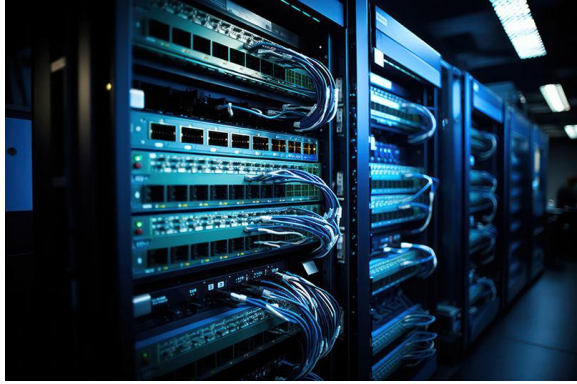
- Sometimes, part of our system is a standard model
  - Object detection, LLM etc.
- There are various services providing paying APIs to use those standard models
  - Either from specialized companies or the big cloud providers themselves

- *When should we use this approach, instead of hosting the models ourselves?*

On-premise

vmware®

kubernetes



- For privacy or cost reasons, the deployment on-premise might be preferred
  - Either at your company or at the clients company
- This requires in-house knowhow and hardware
  - Scaling can become difficult
- But we control the dataflow, which is sometimes a legal requirement
  - Medical data etc.
- In some cases, you can reproduce a cloud like environment
  - For example kubernetes

Edge

TensorFlow.js

ONNX

- Edge computing can be considered a special case of on-premise deployments
- Edge devices are usually specialized hardware, but also "normal"
  - IoT
  - Mobile devices
  - FPGA, Smaller GPUs (Jetson etc.)
  - Web browsers
- Special considerations need to be applied on the models that are run
  - Smaller models
  - Quantized or pruned models
- Special frameworks like TensorflowLite, Pytorch Mobile exist

Mixture

- In a complex MLOps system, you can mix the infrastructure options.
- Example:
  - MLOps pipeline in a public cloud with the model being served on an IoT device
  - Base models trained on-premise because of private data and then deployed in the cloud for inference
  - Kubernetes based infrastructure, with some parts using a commercial API for inference

Technologies

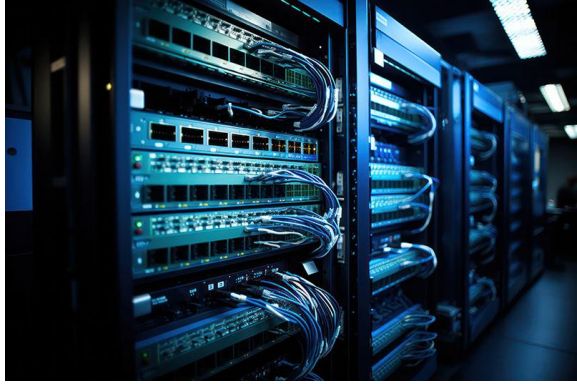
Technologies

- Especially in the cloud there are various ways to deploy ML workloads
  - Dedicated servers/services (microservice or similar)
  - Serverless architectures (Lambdas functions)
  - Batch processing (Kubernetes tasks, CI/CD Runners, Ray etc.)
  - Stream processing (Kafka etc.)
- Again, you might not want the same approach for the production model and development of the production model

Technology requirements

- When looking at what technology to use when deploying individual parts, we need to define our requirements
  - Latency (time to response)
  - Throughput (requests per second)
  - *Bandwidth (Input and output data size)*
- Examples:
  - LLM assistant
  - Periodic model training
  - Realtime object detection

Dedicated servers



- One common way to deploy ML solutions are "dedicated" servers
- Those can be
  - Actual physical machines
  - Virtual machines
  - Containers
- The particularity is, we have reserved resources for our model in advance
  - It is always available with no waiting time
- Scaling is done in steps (1, 2, 3+ servers)
  - This is often not very efficient if you have few requests

Serverless systems

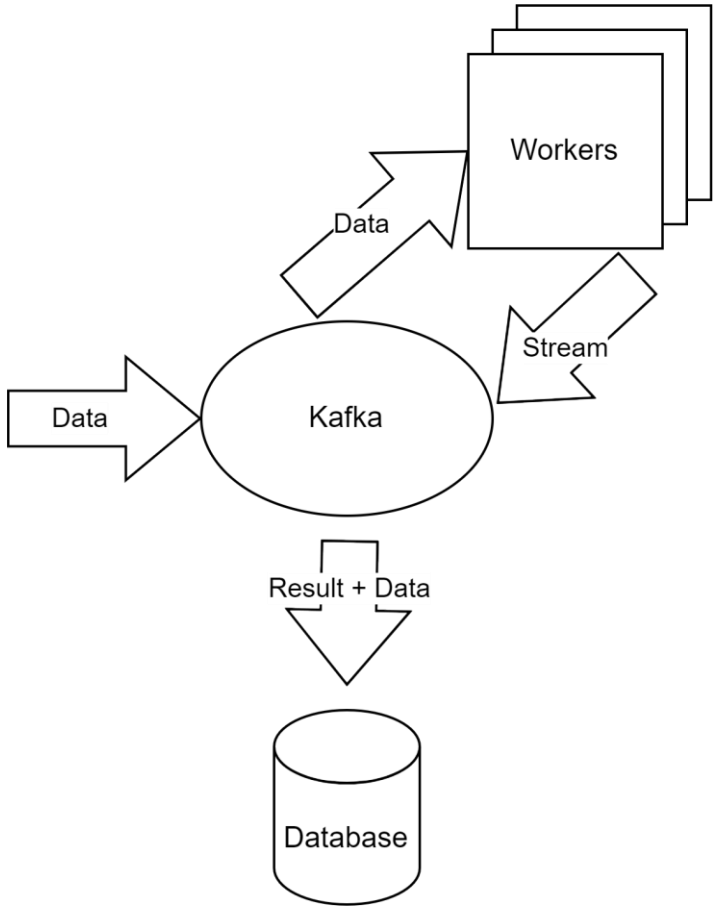
- Having a server dedicated to our model(s) can be expensive in a low usage environment
- Serverless systems, like amazon lambda, allow you to execute specific code without a dedicated server
  - You are billed per usage/call
- There are limits in terms of model size and reactivity
  - Model need to be loaded into memory, for a cold start
- Serverless functions are essentially docker containers started on demand

Batch processing

- When reactivity is not important, it is sometimes interesting to process ML workloads in batches
  - Analyse all the new user-data from today
  - Train a new model
  - Etc.
- Frameworks like Ray or Kubernetes Tasks allow you to do this
  - You define a "task" when is then executed when resources are available
    - For example periodically, or when there is enough work
  - The result is stored somewhere (Database, S3 etc.)
  - A group of multi-purpose worker nodes share the workload

Stream processing

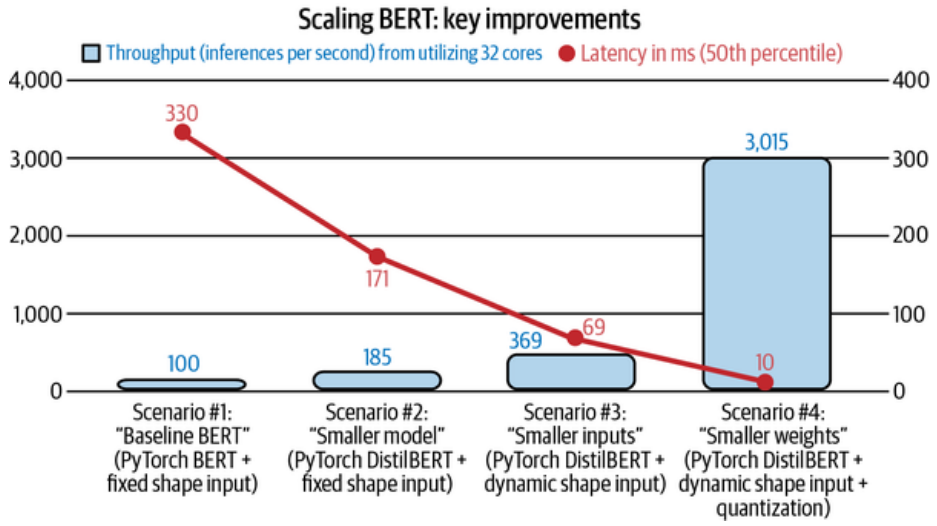
- Stream processing of AI workloads handles more or less frequent streams of data
  - This is "non interactive" usually
- Frameworks like Kafka are often central to this type of workload
- Similar to batch processing, you can scale with worker nodes to handle the stream processing workload
- The frameworks for Batch and Stream processing have a high overlap



Model optimizations

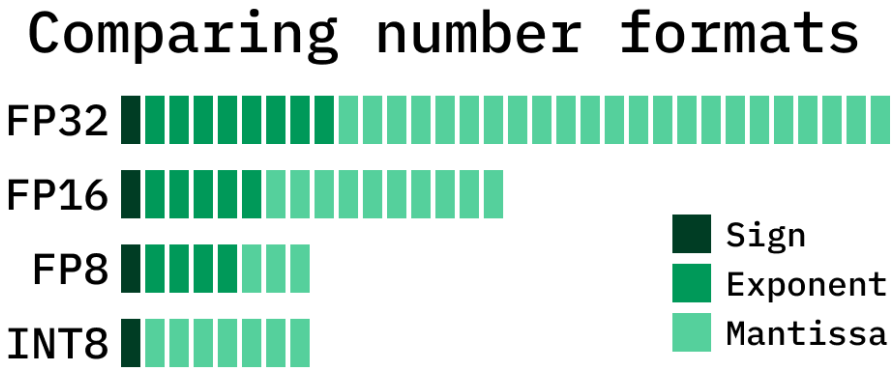
Model optimizations

- When deploying a model, we want to do so using the least amount of resources necessary
- Various techniques exist to improve reduce the hardware requirements of the models, while increasing their speed



Quantization

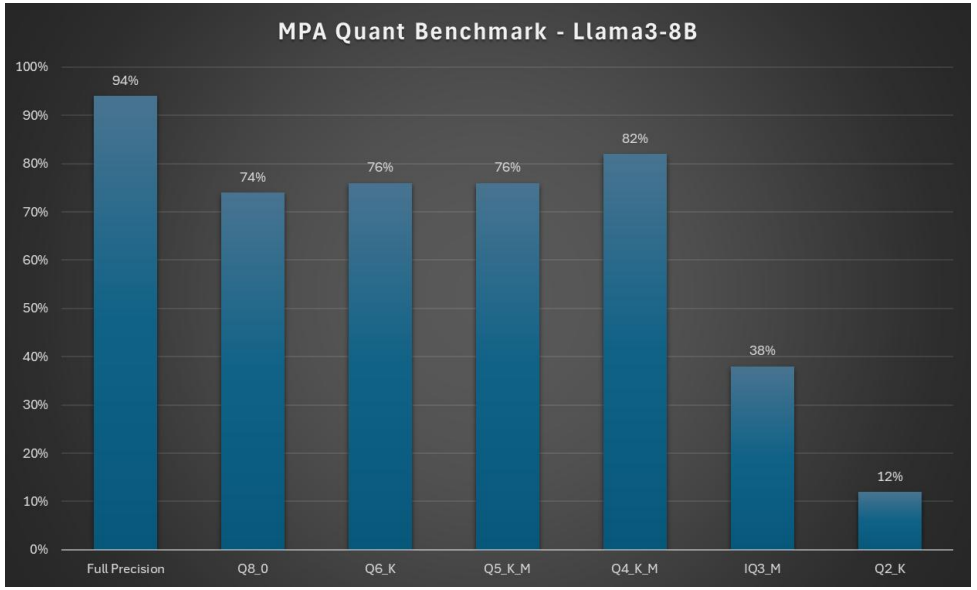
- When training models, we often use 16 or 32bit floats as weights
- In practice, we can often retain a similar level of accuracy, with smaller numbers
- Reducing the weights accuracy reduces the memory requirements and speeds up computations
- Not all GPUs support all data formats



	Supported CUDA Core Precisions									Supported Tensor Core Precisions								
	FP8	FP16	FP32	FP64	INT1	INT4	INT8	TF32	BF16	FP8	FP16	FP32	FP64	INT1	INT4	INT8	TF32	BF16
NVIDIA Tesla P4	No	No	Yes	Yes	No	No	Yes	No	No	No	No	No	No	No	No	No	No	No
NVIDIA P100	No	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No	No	No	No	No	No
NVIDIA Volta	No	Yes	Yes	Yes	No	No	Yes	No	No	No	Yes	No	No	No	No	No	No	No
NVIDIA Turing	No	Yes	Yes	Yes	No	No	Yes	No	No	No	Yes	No	No	Yes	Yes	Yes	No	No
NVIDIA A100	No	Yes	Yes	Yes	No	No	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
NVIDIA H100	No	Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	Yes	No	Yes	No	No	Yes	Yes	Yes

Quantization

- One popular domain for quantized models are LLMs
  - They are large and slow
- There are actually different ways to quantize your model
  - They do not all perform the same and not all frameworks support them all

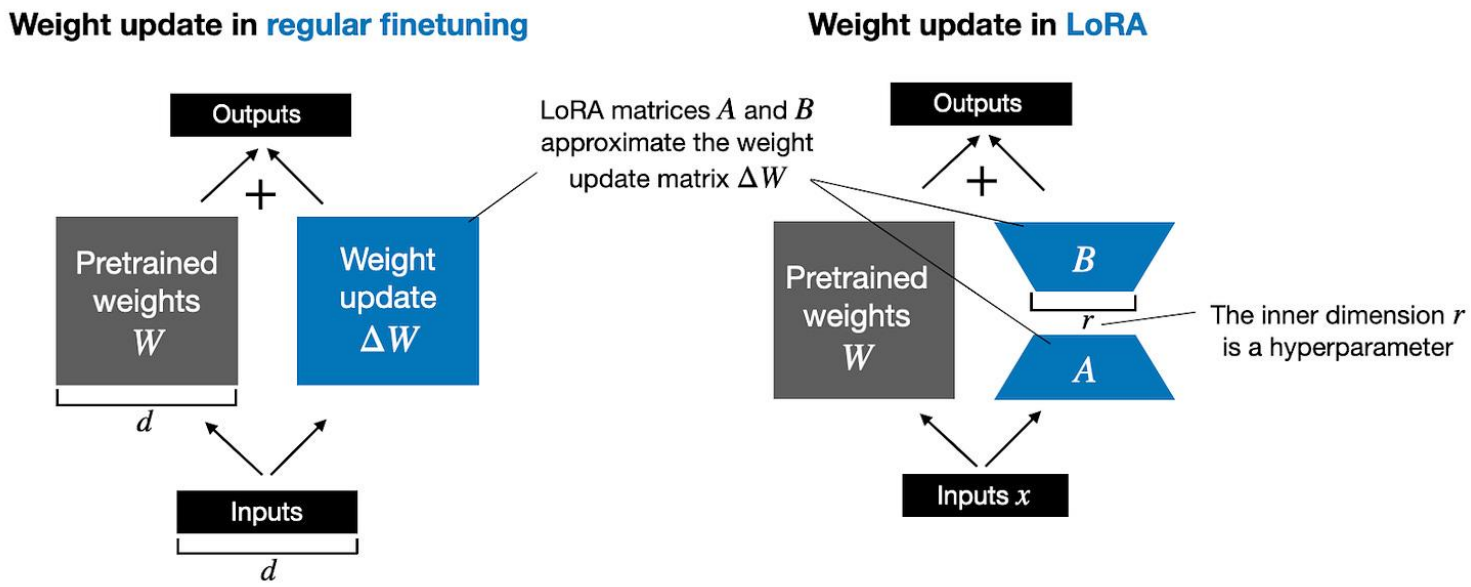


Quantization

- To get a quantized model, two approaches are possible
  - **Post-Training Quantization (PTQ)**
    - The model is quantized after being trained
  - **Quantization-Aware Training (QAT)**
    - Train the model with quantization fake quantization to minimize accuracy loss
- PTQ is a good approach for simple models and if you can accept some accuracy loss, it is easy to put in place
  - Frameworks like pytorch and tensorflow cna do this for you
- QAT is more complex and costly, but will give you the best results in terms of accuracy

LoRA

- Many models are too large to train
  - Again, LLMs are a prime example
- Low-rank adaptation is a technique where we create a companion model, which influences how the original model works



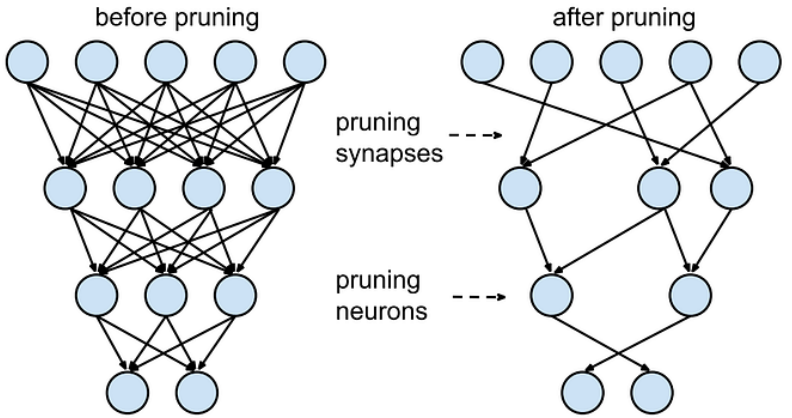
LoRA

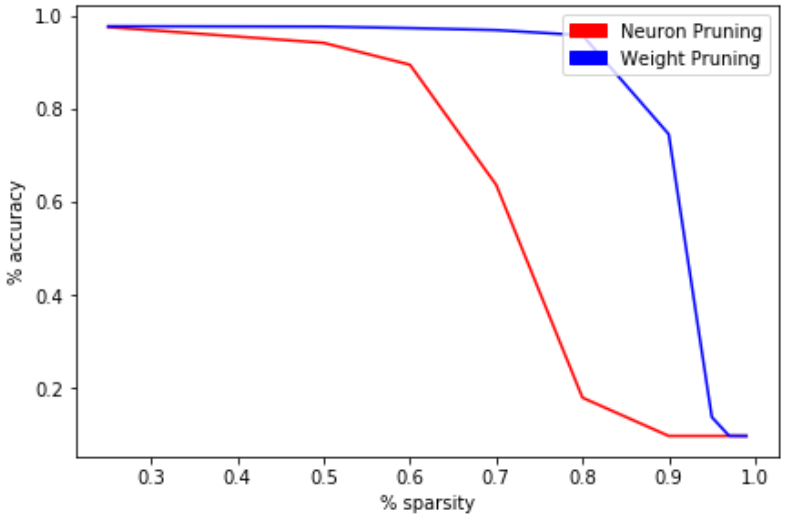
- LoRAs do have a reduced accuracy compared to a full fine-tune
- But the require significantly less resources

- You can have use-case or user specific LoRAs
  - Load the main model in memory
  - Swap the LoRA based on the current use-case or client

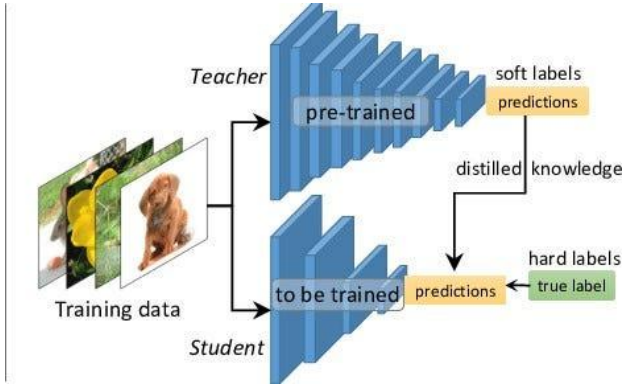
Prunning

- Most models do not need all the weights and connections to work well
- With prunning we simplify the original model, while keeping ist accuracy mostly intact
- The speedups vary depending on the architecture of the model
- There is a trade-off with accuracy





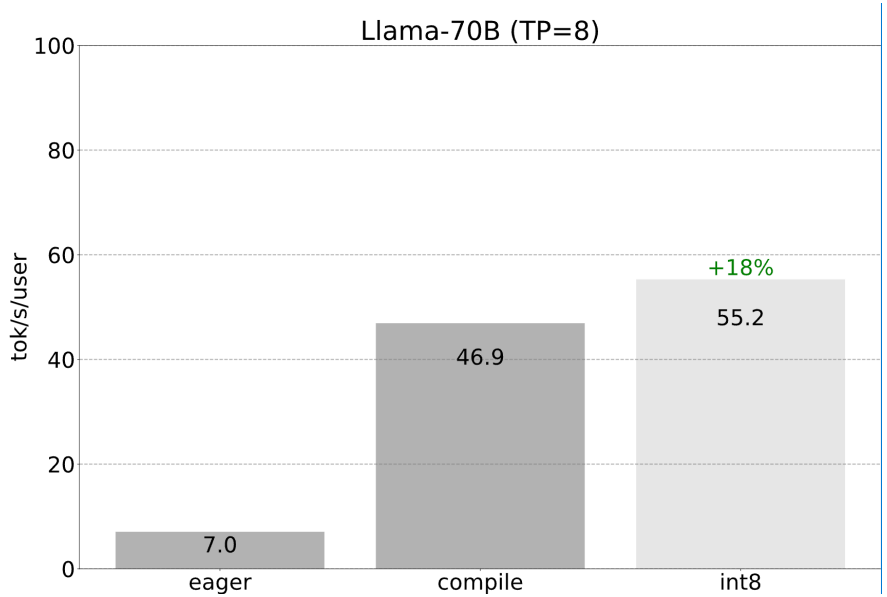
Distillation



- Model distillation allows you to take a large model and reduce its size
- The idea is to train a small model (student), based on the outputs of a large model (teacher model)
- In normal training, we calculate the loss by comparing the predictions with the groundthruth
- During distillation, we train on the predictions of the teacher model, not the groundthruth
  - The student learns to reproduce [0.7, 0.2, 0.1] instead of [1,0,0]
- The results are often much better than training the small model with only the training data

Model compilation and frameworks

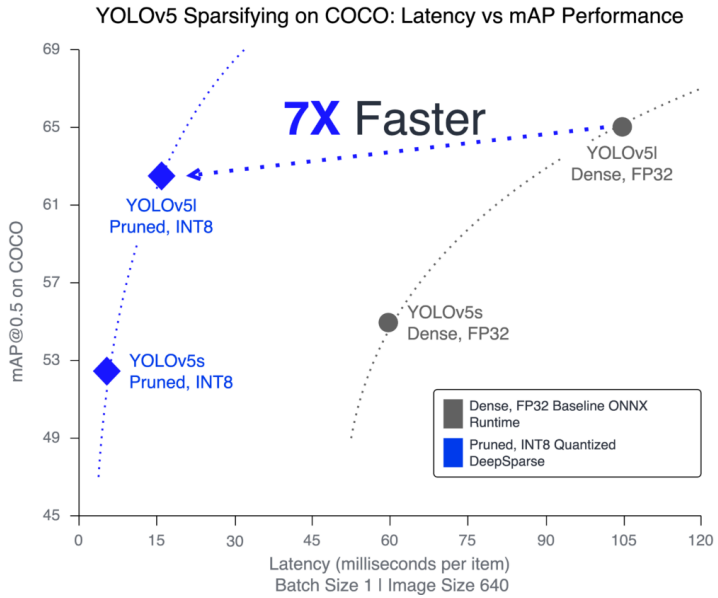
- Specific compilers and runtime environments exist to execute models faster than standard frameworks
- This is a fast evolving field, but it is worth a look when deploying a model
- One example is pytorch
  - Pytorch compile
  - Pytorch autoquant
  - Etc



https://pytorch.org/blog/accelerating-generative-ai-2/

Example

- A popular example of multiple techniques being combined is the Yolo model
- YouOnlyLookOnce is a object detection model, focused on speed
- They employ all the available techniques to speed up their model architecture
- In this particular example, CPU specific methods have been applied



https://neuralmagic.com/blog/benchmark-yolov5-on-cpus-with-deepsparse/

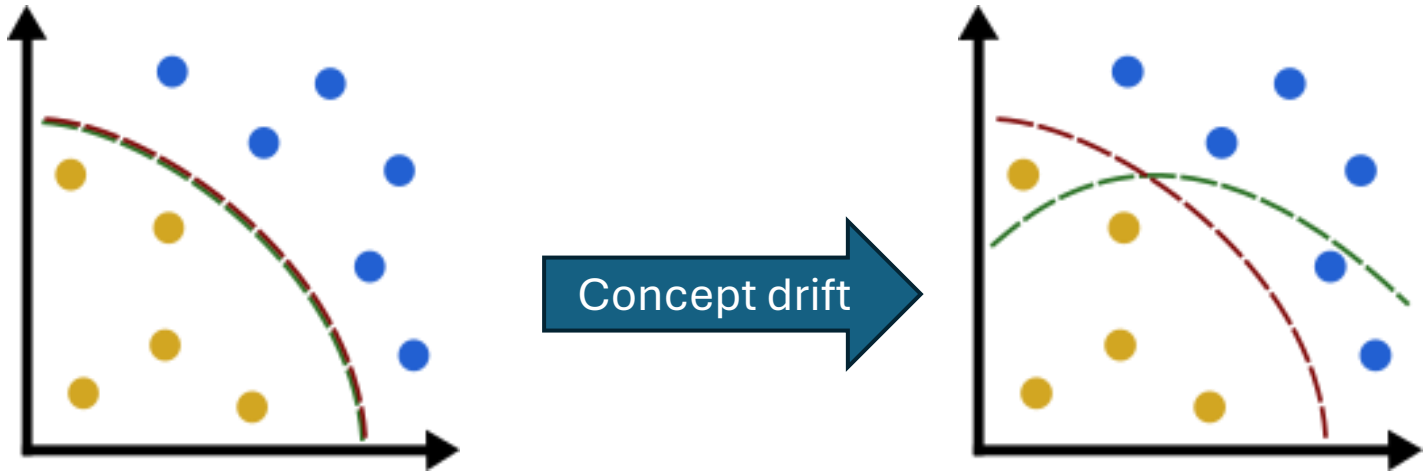
Model Drift

Model drift

- When we create a model, we know its performance on a test set
- In production and over time this performance can change
  - This is called **model drift**, a change in performance (worse) of our model
- The cause of model drift can be various
  - The concept we are modeling changed over time
  - The data we analyse no longer follows the training distribution
    - This includes data quality
- The world we live in is not static, monitoring our model is essential

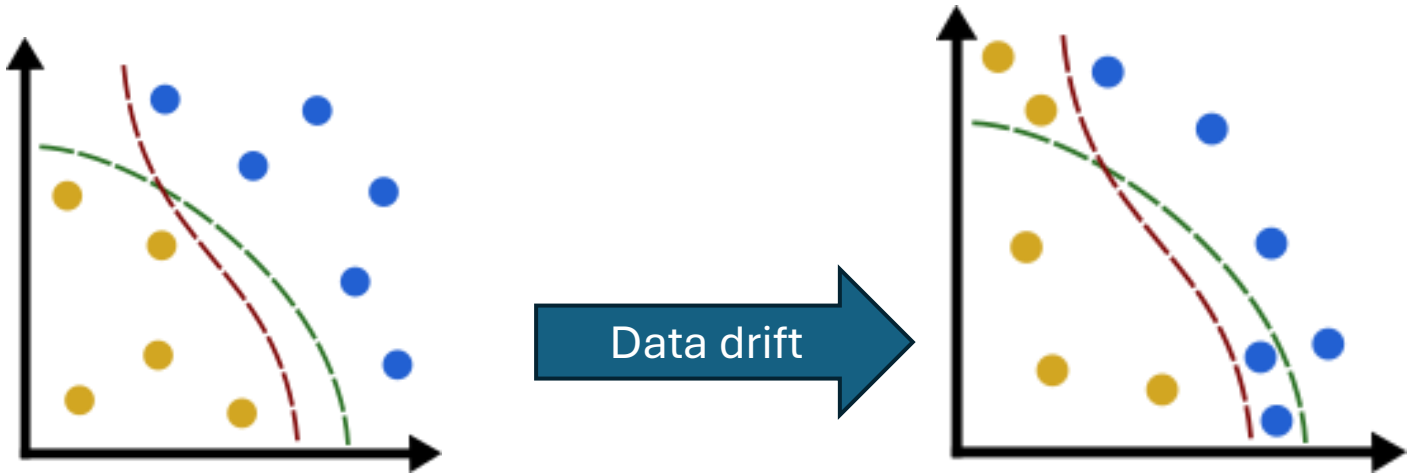
Concept drift

- The concept that the model of what we model changed over time (green decision boundary)
  - Eg. High risk / low risk genetic markers
- We still have the same data (train vs production) but some are now wrong because the decision boundary moved (green) but not the model decision boundary (red)



Data drift

- Data drift does not change the concept we model
  - But our initial decision boundary (green) did not represent the "real" decision boundary (red)
- In production, we see data outside our training distribution, revealing this error

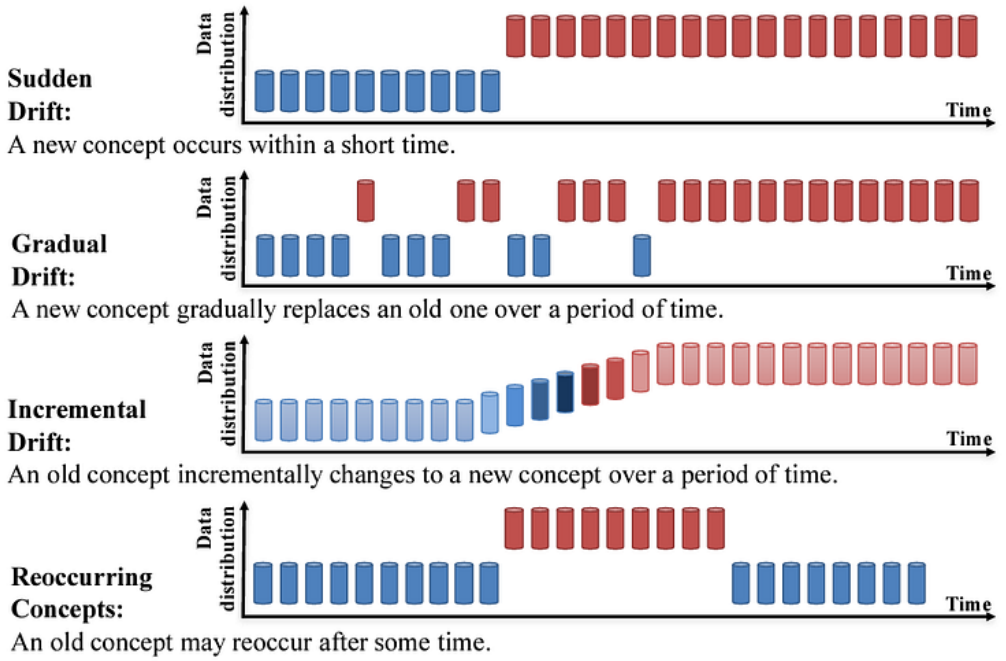


Data drift – Label vs Feature drift

- Feature drift represents a change in input features
  - P(X) changed (probability to see a certain input feature)
  - More clients from a specific region
- Label drift
  - P(Y) changed (probability to see a certain model output)
    - This usually implies a feature drift as well
  - We see more pneumonia cases because of a seasonal change

Rate of drift

- Model drift can occur at various speed
  - Sudden drift
    - Covid
  - Gradual drift
  - Incremental drift
  - Recurring drift
    - Seasonal changes



https://medium.com/the-modern-scientist/understanding-data-drift-in-machine-learning-51bf0022ee14

- We need to be aware of those possibilities to better detect them
  - Sudden drift are easy, the others are trickier
- It will also inform our retraining strategy (frequency, trigger etc.)

Data drift monitoring

- If for some reason we know the true label after a prediction, we can track the accuracy metrics of the model over time
  - But this requires a source for the true label, usually the user
- Otherwise, we need to monitor the data drift
  - Change in distribution of the input and/or the output of the model
  - This includes a change in data quality
- Libraries exist, such as Alibi-detect, to simplify the calculation of metrics related to drift detection

Data drift

- Basic tracking of data drift can be achieved easily with individual input / output values
  - Track ratio of categorical values
  - Track distribution of numerical values
- Classic statistical tests can be used (Fisher, Chi-squared etc.)

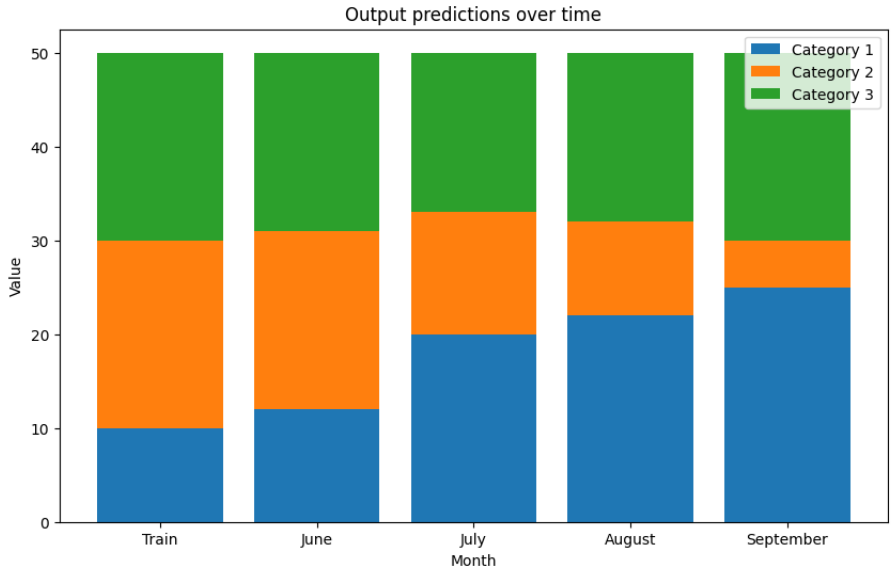
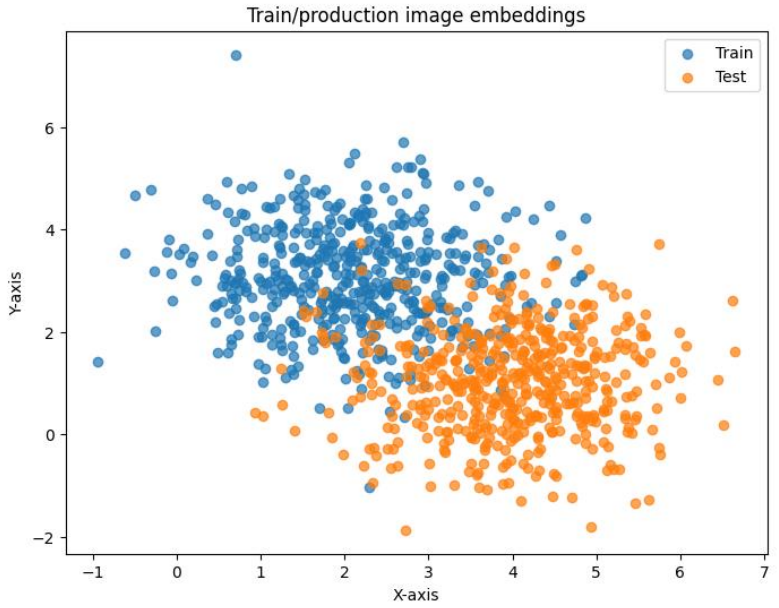
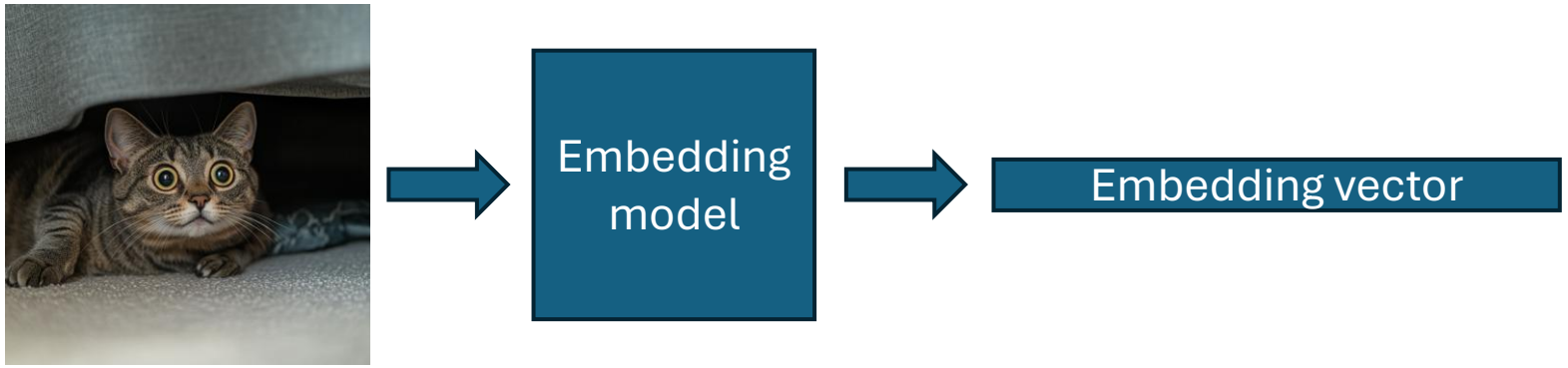


Image data drift

- For tabular data, detecting distribution drifts is "easy"
- Complex data, such as images or text require different strategies
- One strategy is to track the embedding vectors





Monitoring

Monitoring

- A central part of MLOps is monitoring your solution (like for DevOps)
- What to monitor depends on your application and needs

- *What are the metrics we should monitor in a MLOps environment?*

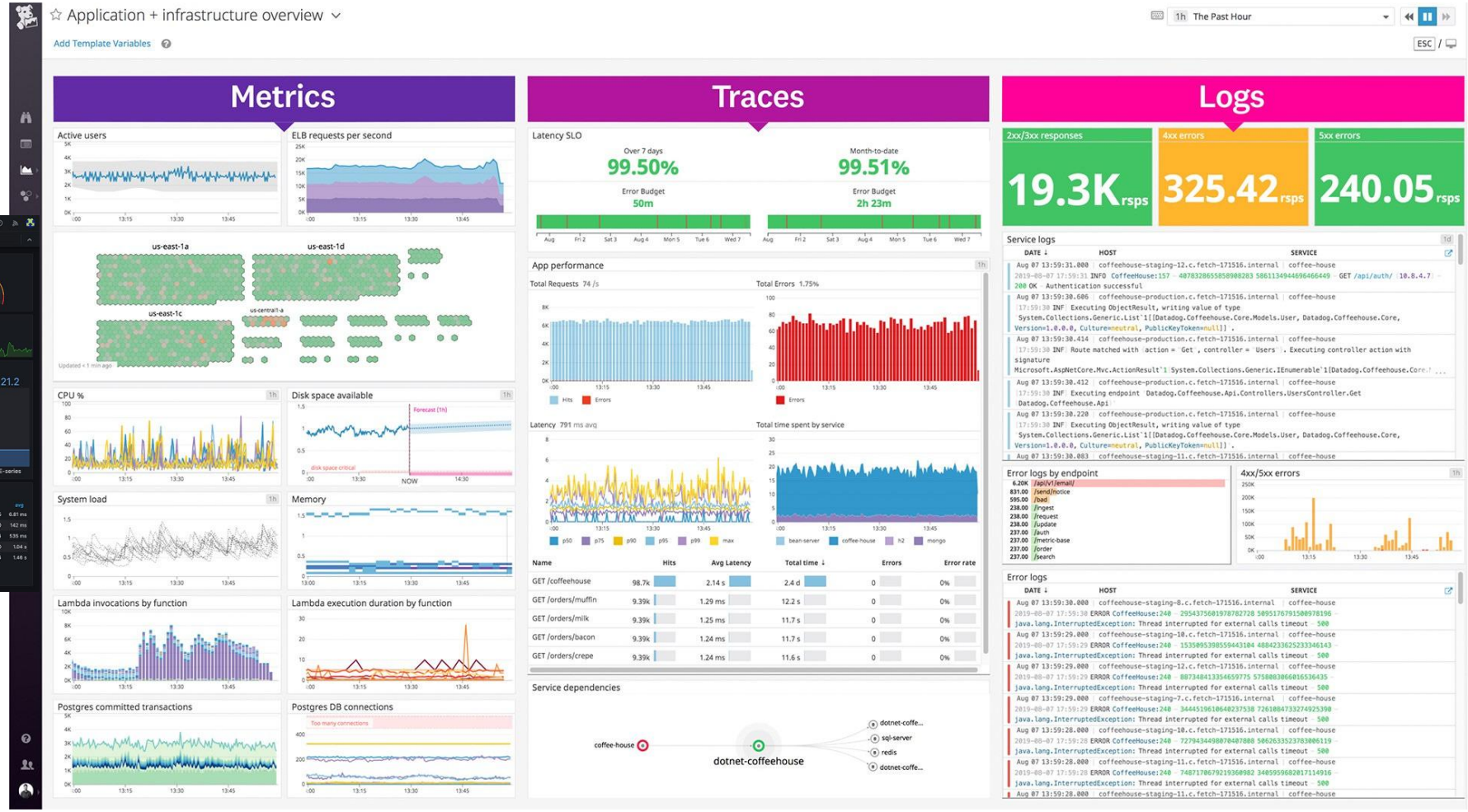
What to monitor (non exhaustive)

- SLA (Service Level Agreement)
  - Uptime
- Performance
  - Time to answer / Latency
  - Number of requests
  - Resources (GPU, CPU, Memory, I/O)
- Distributions
  - Model/data drift
- Data quality
  - Missing values, invalid values and outliers
- Model
  - Accuracy
  - Prediction confidence
- Model training
  - Accuracy, etc.
- Other
  - Costs (API, hosting)

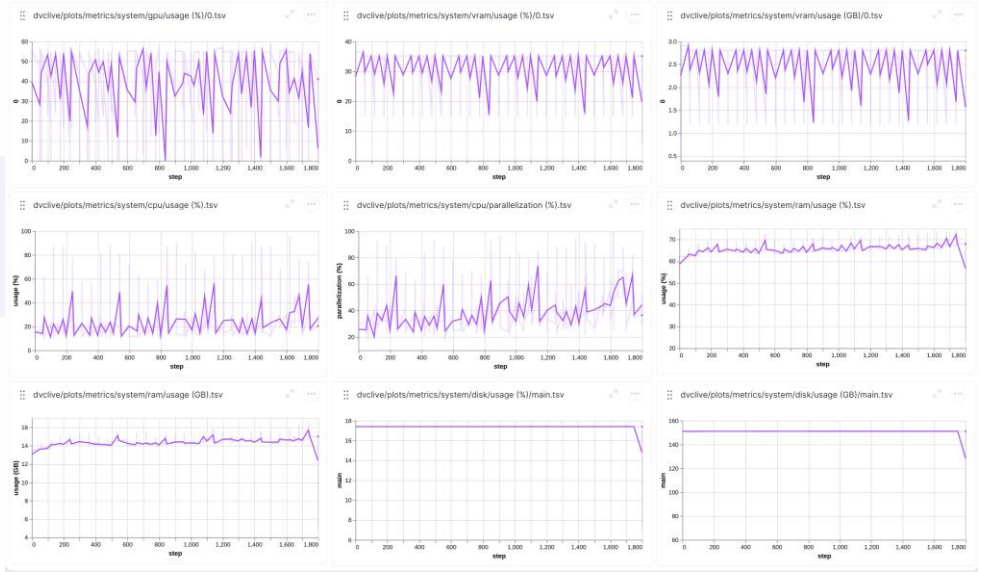
Frameworks

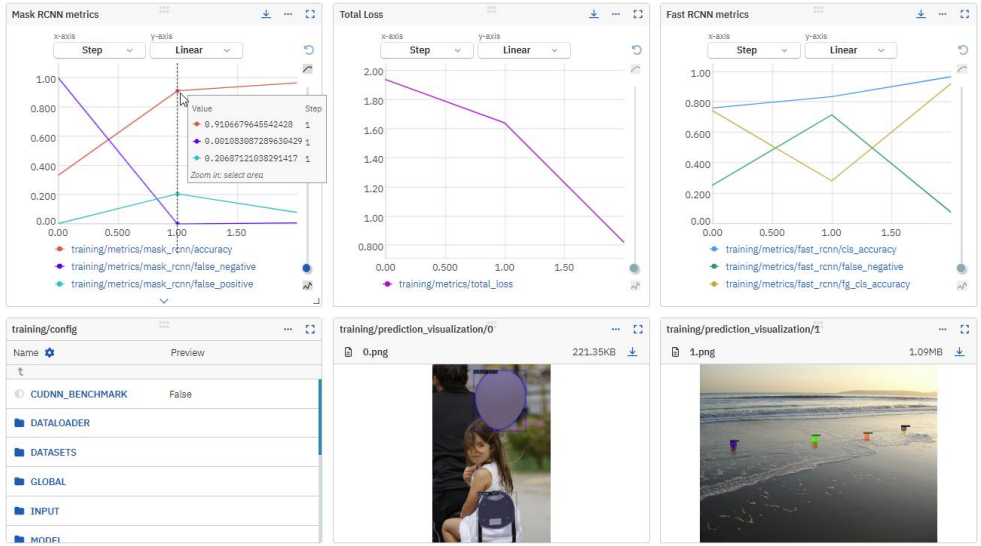
- Various logging frameworks exist, a simple graphana is often enough
- Depending on what we log and the use-case the choice is different:
  - Dashboards
  - Alerts
  - Model training/experiments
  - Triggering new model trainings
    - Manual or automatic
- Specalized tools, such as Weights&Biases for model training, Evidently.AI for drift detection and other exist

Monitoring dashboards

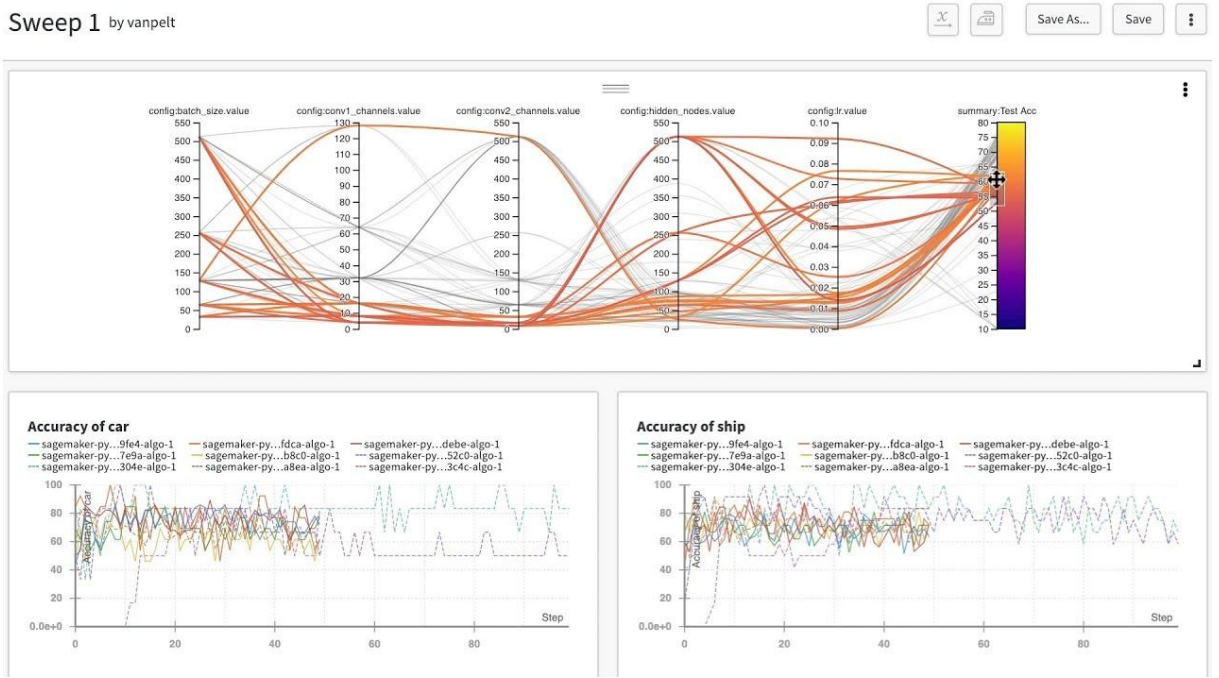


Experiments monitoring





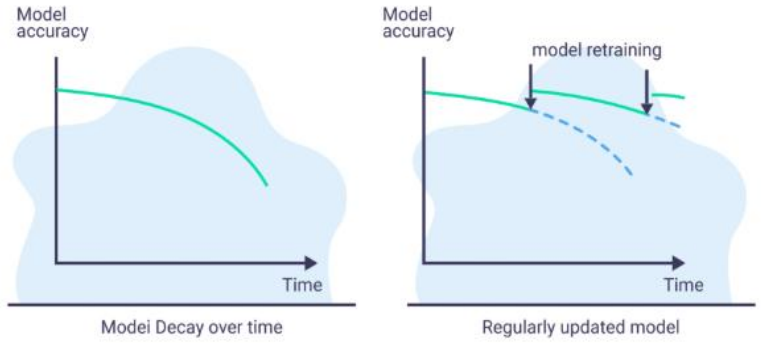
Weights & Biases



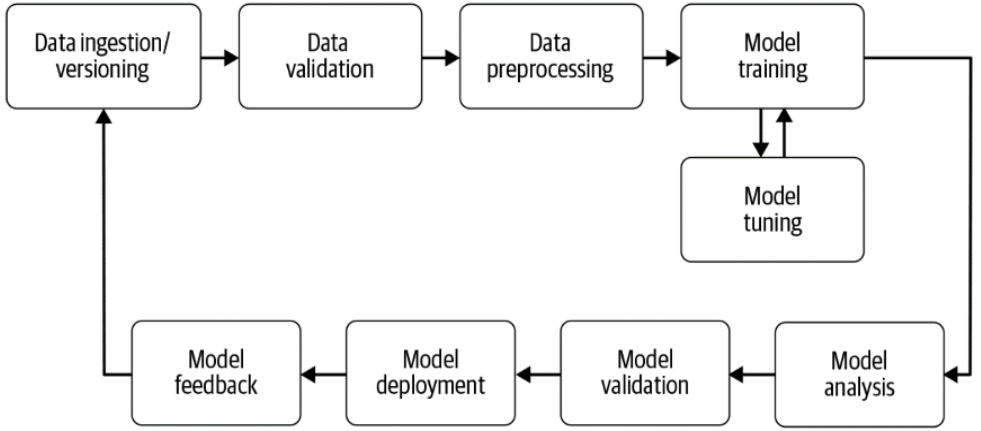
Continual learning

Continual learning

- Ideally, we never have to retrain our models
- But often we need to because:
  - We did not have enough data initially (cold start)
  - Model drift deteriorated our performance
    - Could be special events like black friday
- Continuous learning allows us to regularly update in production



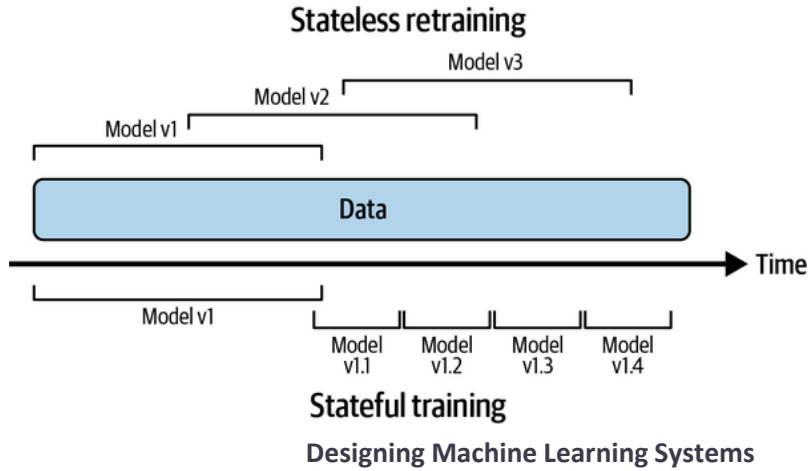
https://www.aporia.com/blog/concept-drift-in-machine-learning-101/



https://www.oreilly.com/library/view/building-machine-learning/9781492053187/

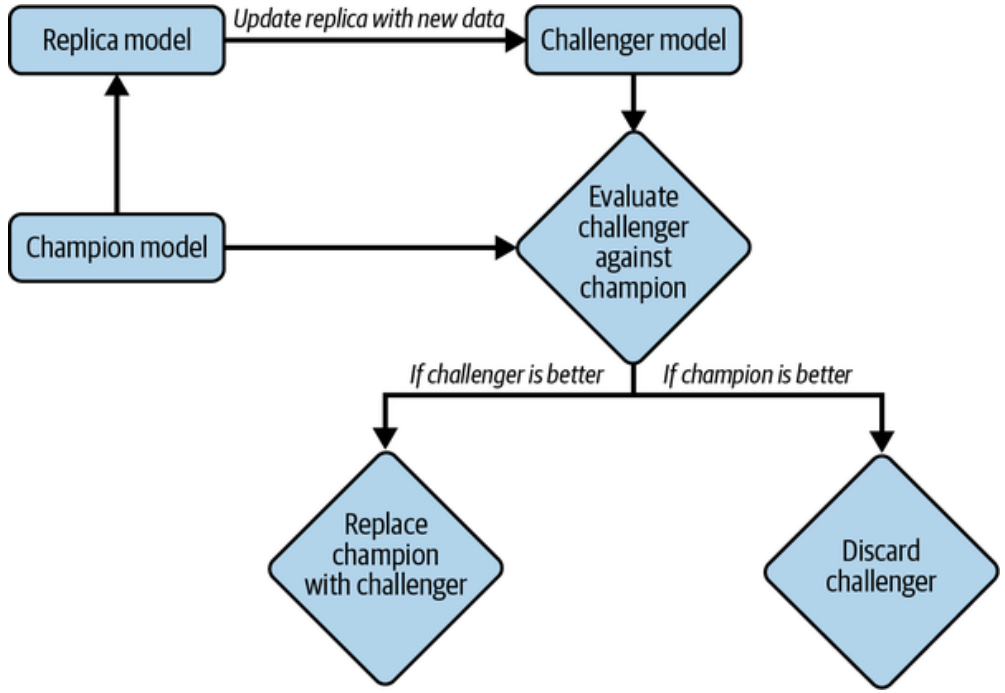
Training strategies

- Various strategies exist to define which data will be used to train a new model
  - Stateless training takes all data since a certan point in time
  - Stateful training finetunes a base model on a subset (usually most recent) of the data
- Stateful requires much less data, but makes it harder to make big changes to the model
- Combinations are possible, where sometimes a big base model is trained and that base model is then finetuned frequently



Not all models are equal

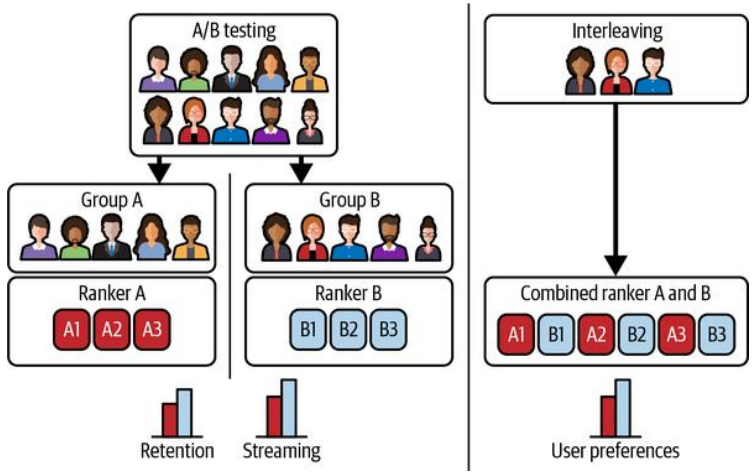
- New models may or may not be better than old models
- The new model is always benchmarked against the current model
  - Automated benchmarks
  - A/B testing
  - Etc.
- Only of it proves better, it will be deployed



Designing Machine Learning Systems

Deployment

- How to deploy your model without causing issues is key
  - **Shadow deployment**
    - Deploy your model in parallel with the existing and compare their output
  - **A/B testing**
    - Serve the new model to a certain percentage of users and compare the results
  - **Canary testing**
    - Similar to A/B testing, but increase percentage of users over time if satisfactory
  - **Interleaving experiments**
    - Useful for ranking systems. Combine output of old and new model



Other topics

Some MLOps topics we did not cover

- Some other topics not covered, but that you can cover!

- Hardware
  - GPUs, FPGAs, CPU
- Data versioning solutions
- Model registries