Model serving

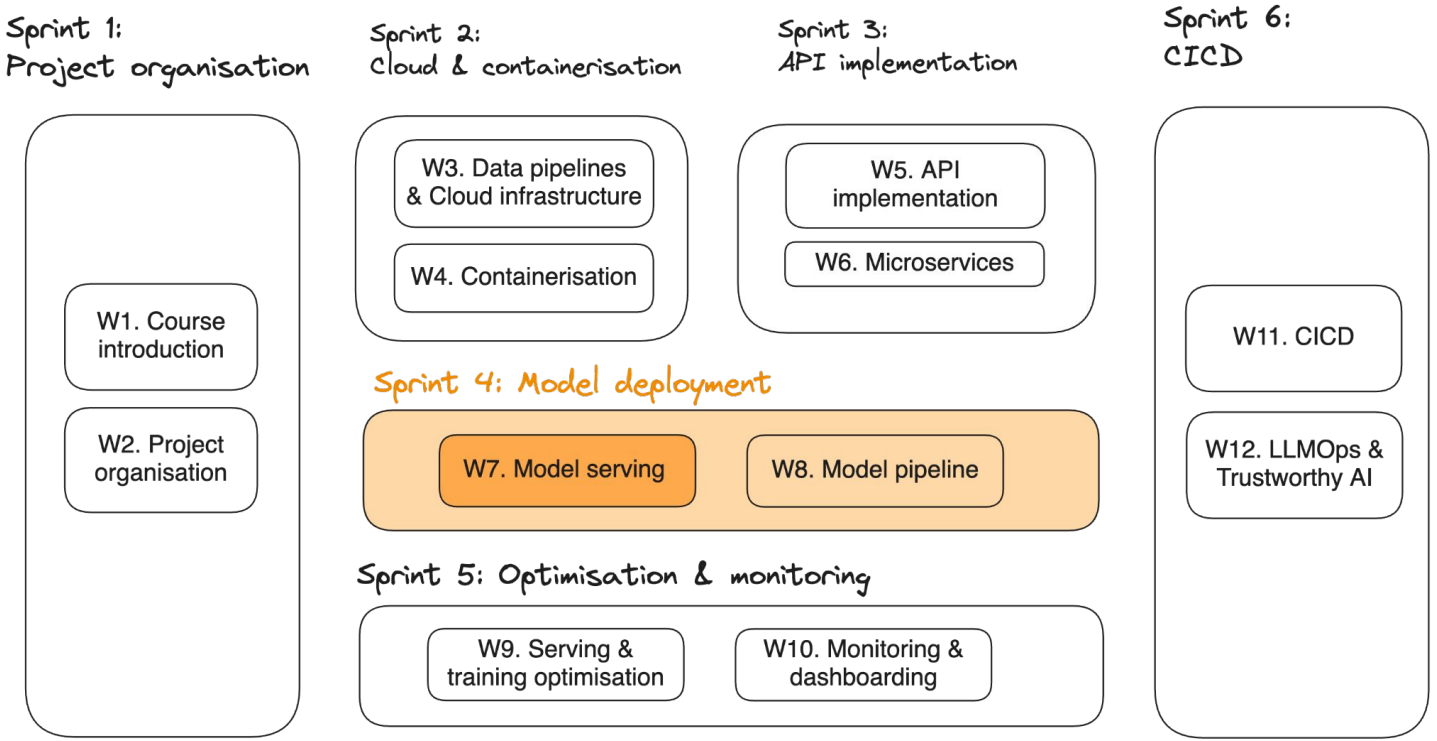
Sprint 4 - Week 7

INFO 9023 - Machine Learning Systems Design

Thomas Vrancken (t.vrancken@uliege.be)
Matthias Pirlet (matthias.pirlet@uliege.be)

2025 Spring

Status on our overall course roadmap




YOU'RE DONE TRAINING YOUR MODEL?
TIME TO START SERVING IT !
imgflip.com

Agenda

What will we talk about today

Lecture

1. Use case deep dive: Rug cutting
2. ML Model serving
3. Cloud vs on-prem deployment

Guest lecture

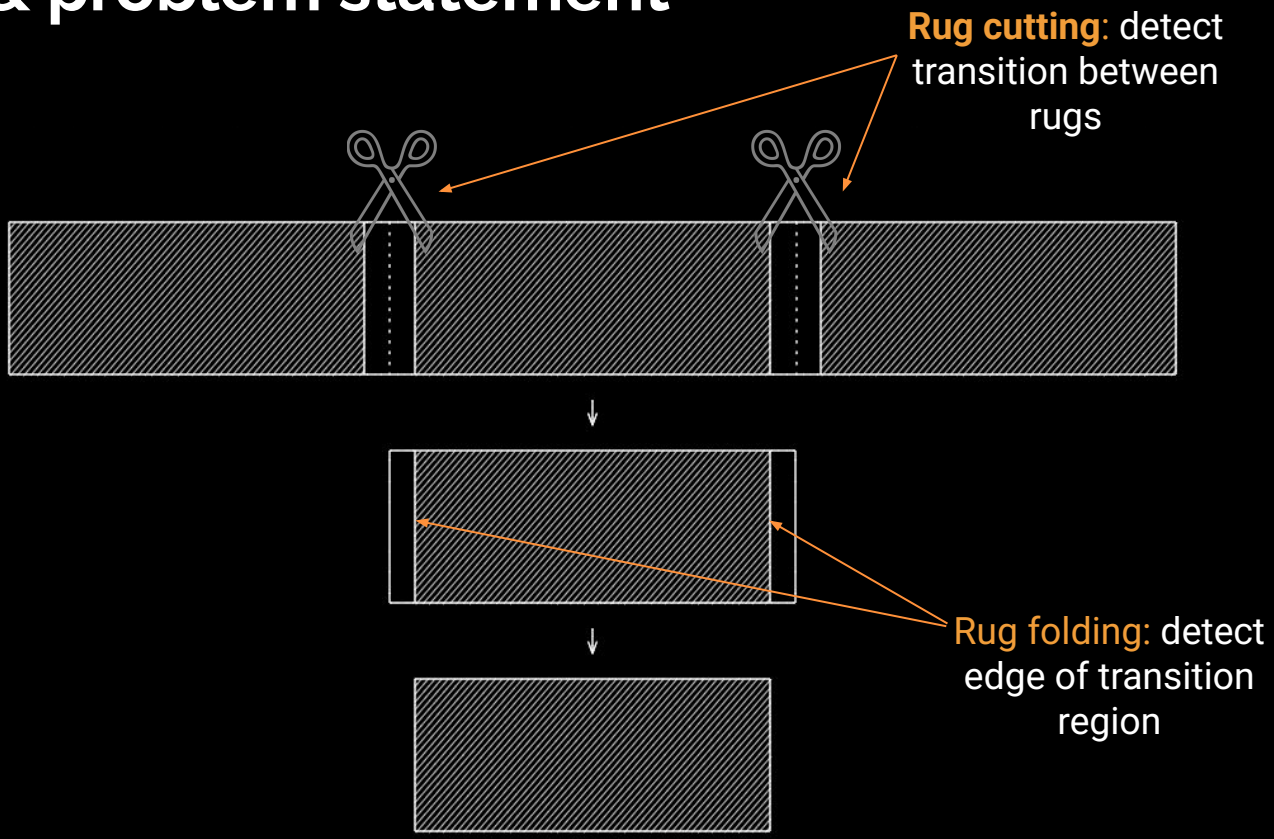
4. Connexion

Demo

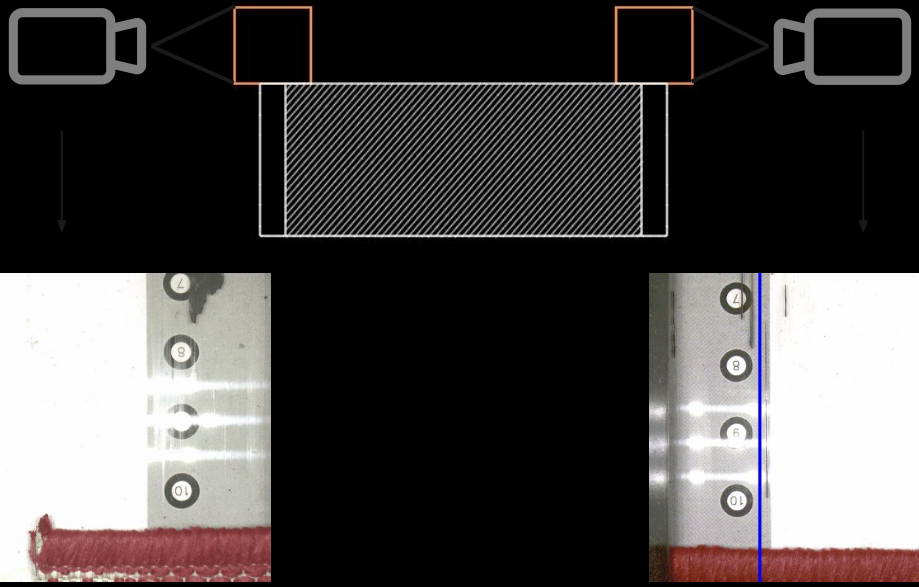
5. vLLM on the Cloud with GPUs

ML Model serving

Context & problem statement



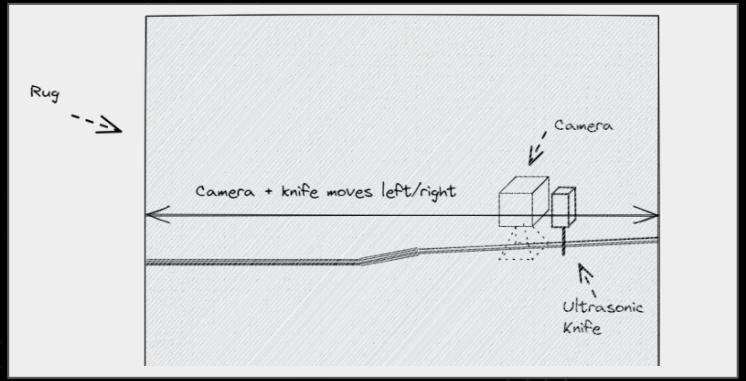
Rug folding

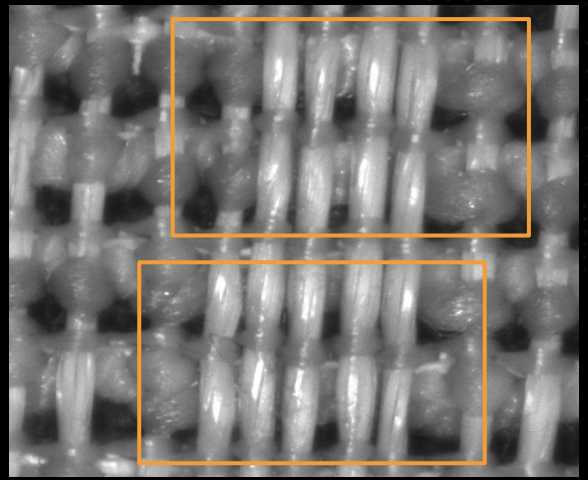


Rug cutting: 2 AI models

- Line Search: The rug must be stopped when the outer edge of the cutting line is in view of the camera. An approximate distance of where to stop is known, but a visual check based on image processing is necessary for accurate localization.

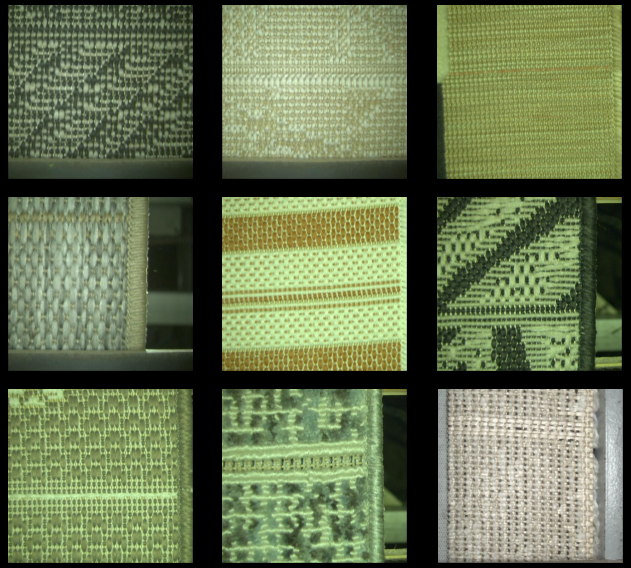
- Line Tracking: Once the outer edge of the cutting line is found and centered, A camera and an ultrasonic knife move along the width of the rug while adjusting the rug position based on the images to make sure that a cut is made at the correct position.



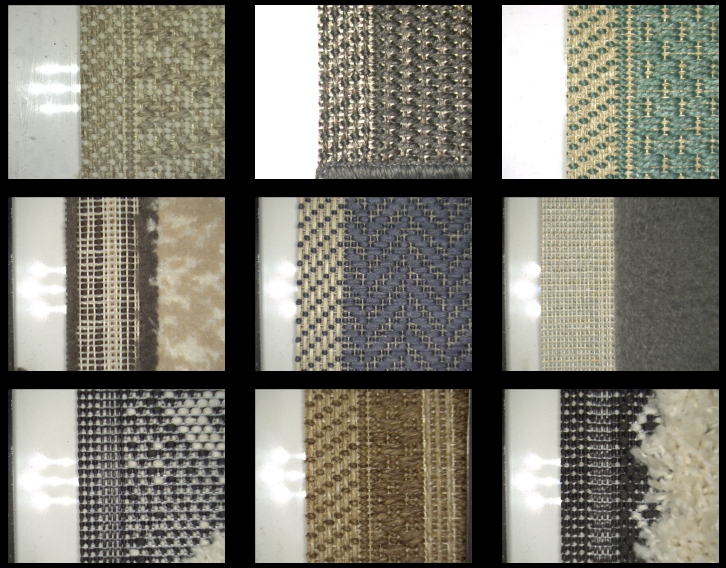


Rug cutting: different carpet types

S1 carpets

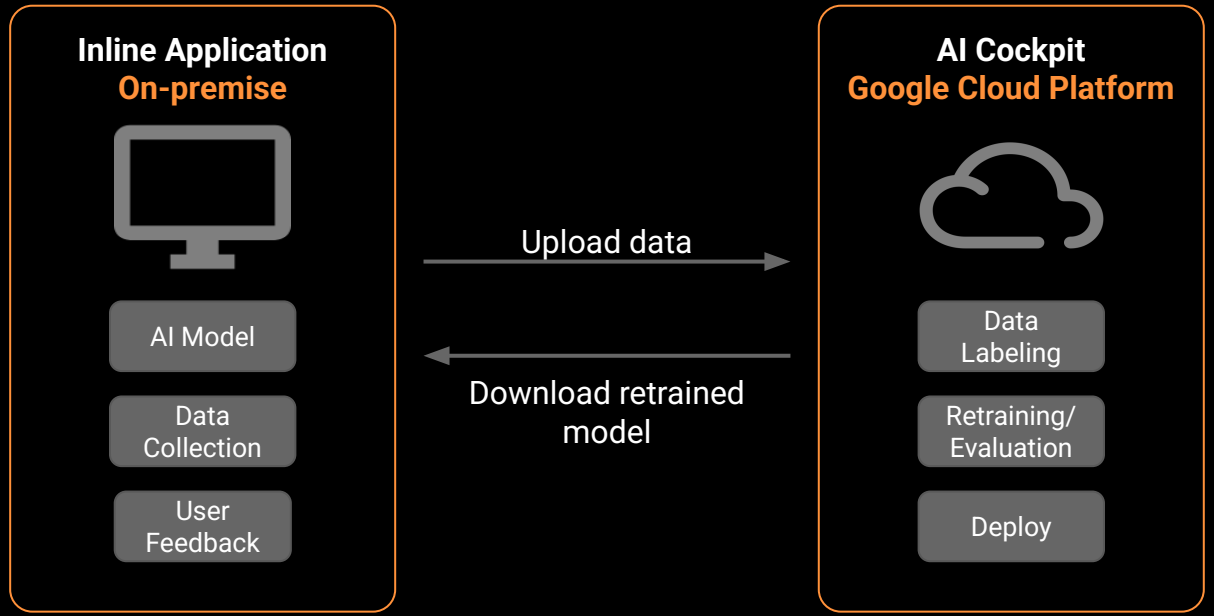


L1 carpets

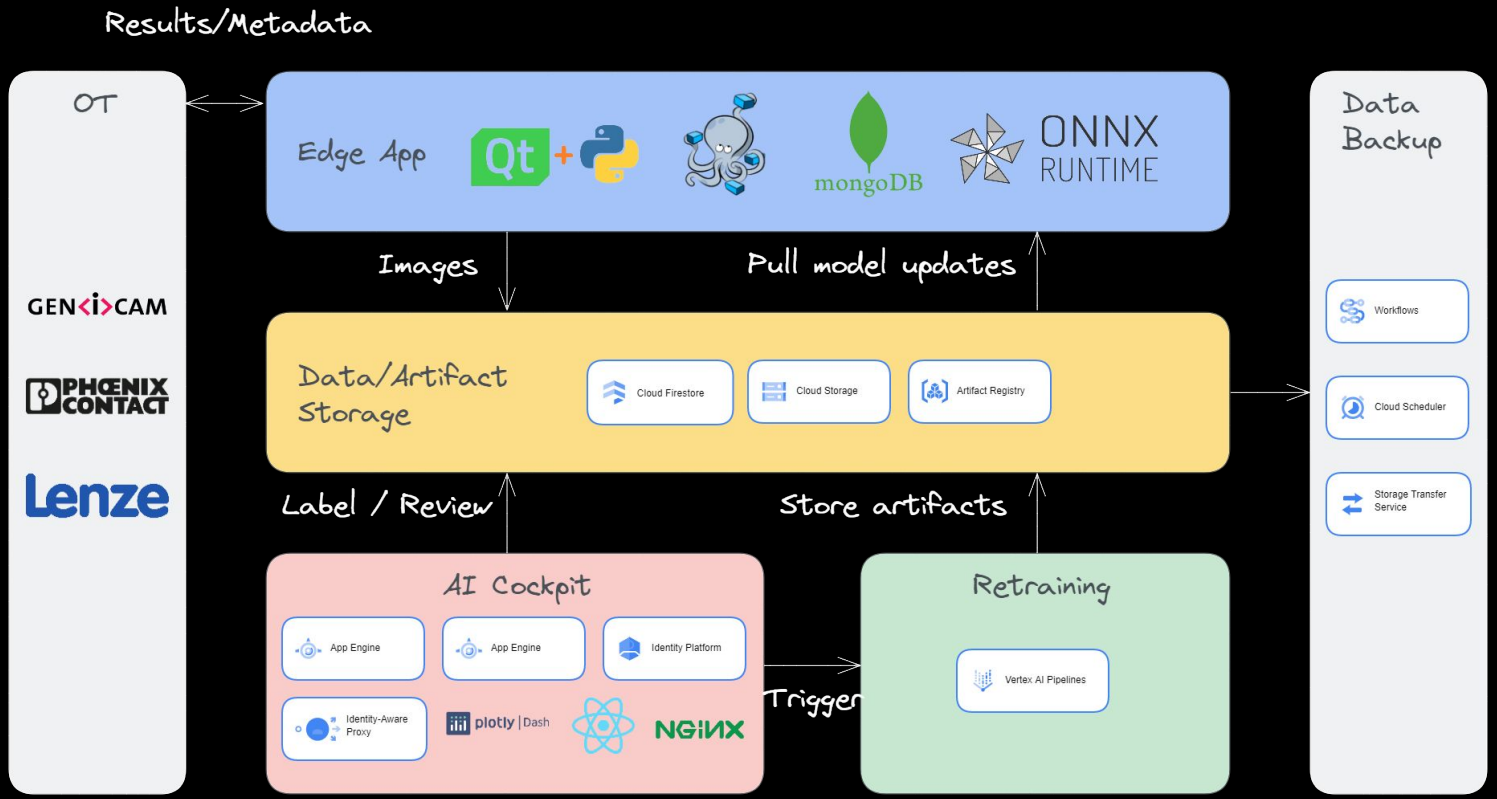


Solution helicopter view

Hybrid architecture



High level architecture and tooling.



The integration study has produced three key decisions

Compute Device

Camera Type

PLC-centric vs PC-centric

Traditional Windows Desktop

- Remotely maintainable by IT
- Known environment

- Windows OS not suited for real-time industrial applications
- Increases development difficulty
- Long lead times for purchase

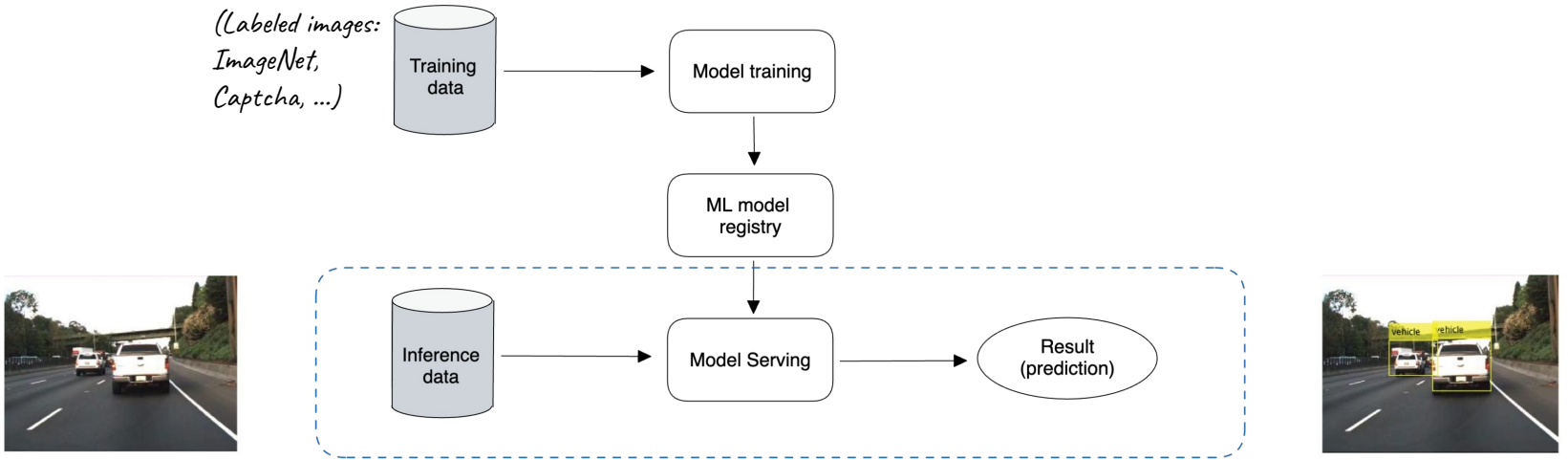
Nvidia Jetson

- Industrial single-board Linux computer with GPU
- Small form factor (DIN rail mount)
- Faster availability (+/- 1 month)
- Enables standardization to speed up more followup use-cases
- Learning curve because of Linux OS

ML Model serving

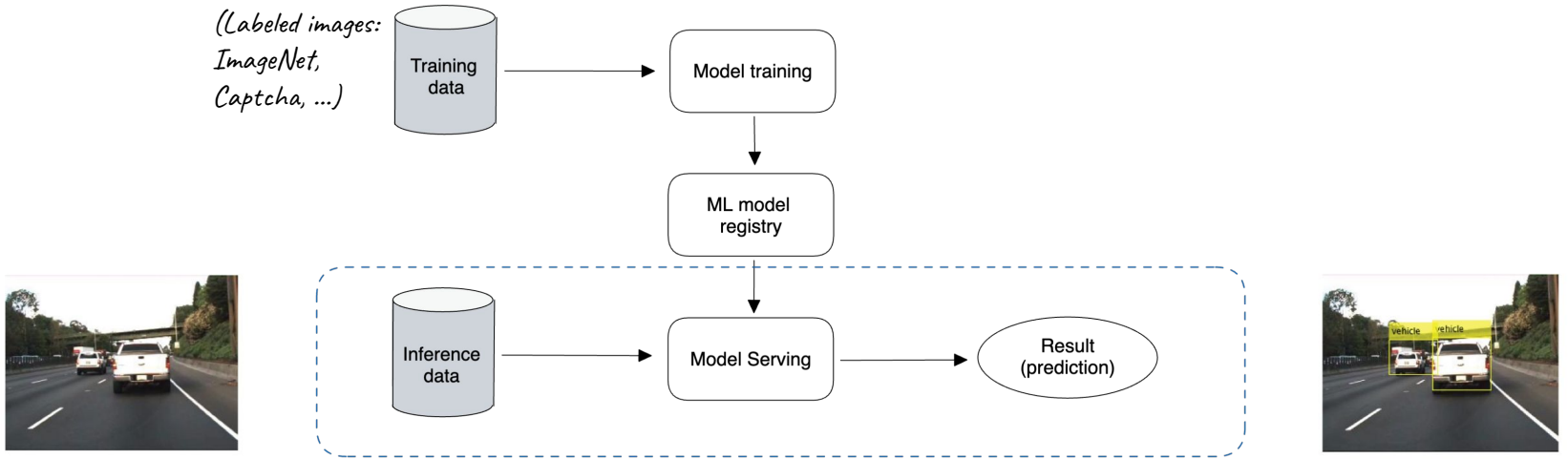
What is model serving?

After training an ML model, it can be called on **inference** by users during **model serving**.



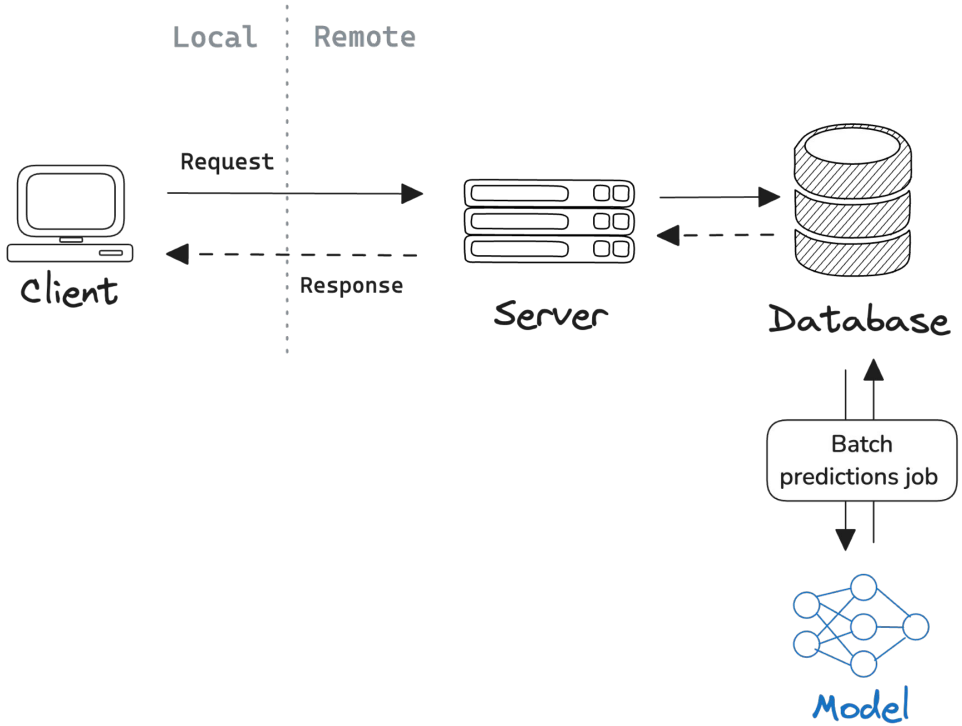
What is model serving?

After training an ML model, it can be called on **inference** by users during **model serving**.



Latency = ⏱️ time between request & response

Offline predictions



Offline predictions

Offline serving (aka **batch serving**): Model is used in a batch job on a large number of historical data points.

Prediction can be triggered by:

- **Scheduler:** periodically (daily, weekly, ...) on all new or relevant data points.
- **Triggered**: Upon a specific event (new data batch uploaded)

Models have **lower latency** but **higher throughput** requirements.

Optimisation techniques can still be valuable for cost/compute benefits.

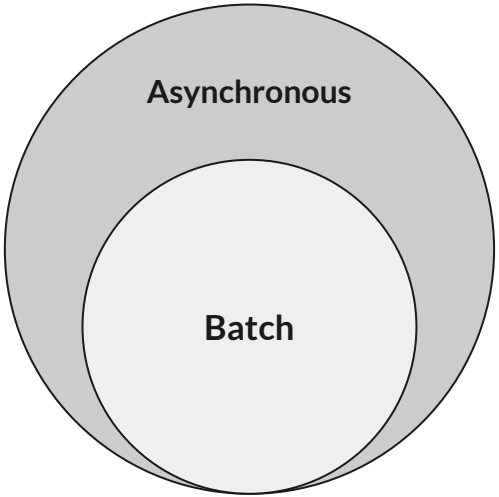
Examples:

- A lot of forecasting models (periodical run when you have new data - daily, weekly, …)
- Document AI on archive
- Sentiment analysis on customer feedback

Batch predictions

Pros	Cons
● Low Latency (⚠️ for pre-computed systems) ● Simple to implement ● Less downtime risks	● Can only produce predictions for a fixed list of inputs ● Doesn't scale to complex input types ● Users don't get the most up-to-date predictions

Synchronous and asynchronous.

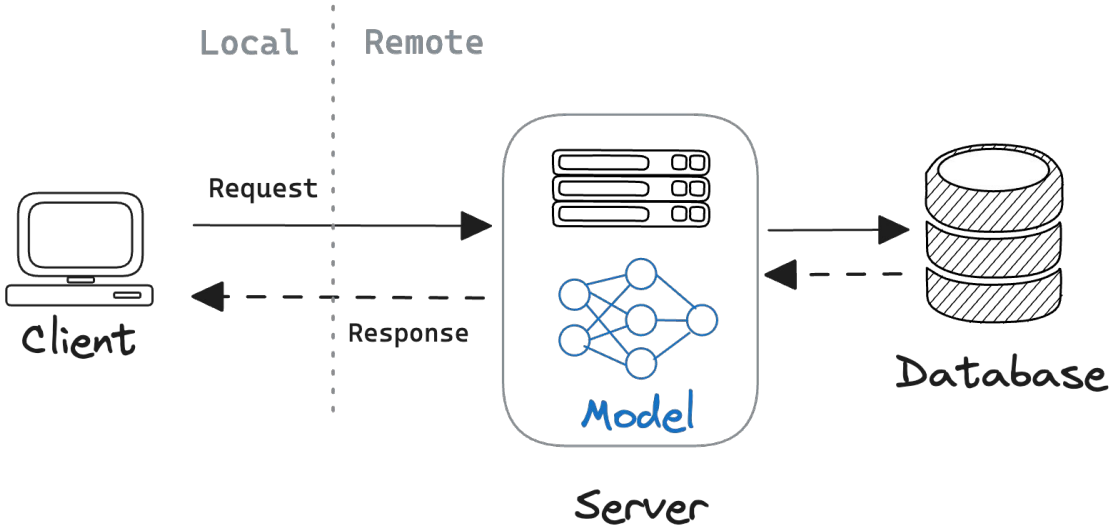


Asynchronous: The request for prediction (+ subsequent action) and the model computing result do not need to happen at the same time

- **Push**: The model generates predictions which are pushed as notifications to the user (e.g. fraud detection)
- **Pull**: The results are computed and stored in a database, allowing retrieval on demand for subsequent actions

⚠️ Batch serving is *an example* of asynchronous pull serving. However, it focuses on the aspect of making large batches of predictions in one job.

Online predictions



Online predictions

Online serving (aka **real-time**): Model is used in a real-time by being called on a single data point and directly returning the result. All input data (features) need to be made available to the model.

Low latency is often a hard requirement for online serving.

Examples: Detecting car traffic objects in images, spam detection, …

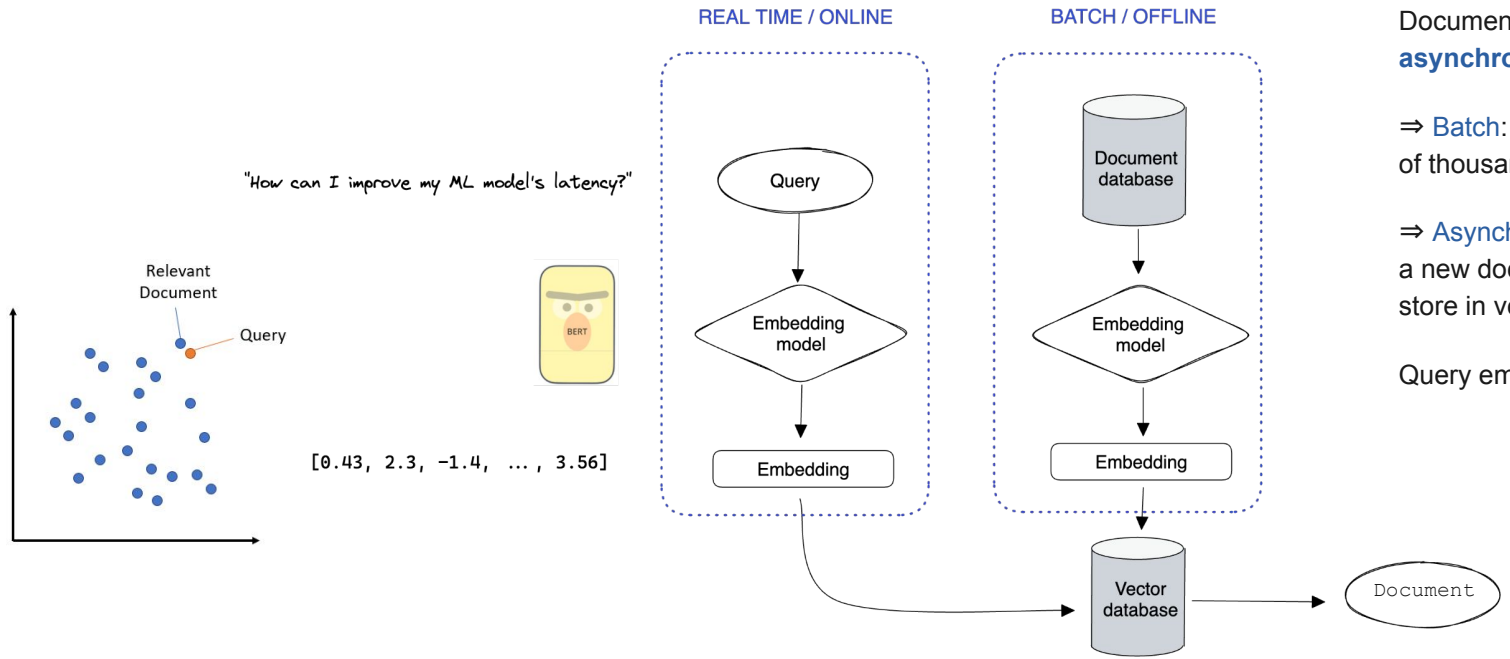
Attention: Online ≠ On the Cloud. You can do real-time (online) serving but on prem, so not on the Cloud

Online predictions

Pros	Cons

● Personalization & generalisation: Data point-specific predictions
● Latest possible prediction
● No scheduled pipeline or prediction data storage necessary
	● Requires low latency model
	● High uptime required for your model
	● Serving infrastructure overhead
	  ○ GPUs provisioning
	  ○ Scalability
	  ○ Load balancing
	  ○ Cold start

Search engine: Example of hybrid model serving.



Document embeddings: **Batch + asynchronous**

⇒ Batch: Historical load (big job of thousands of documents)

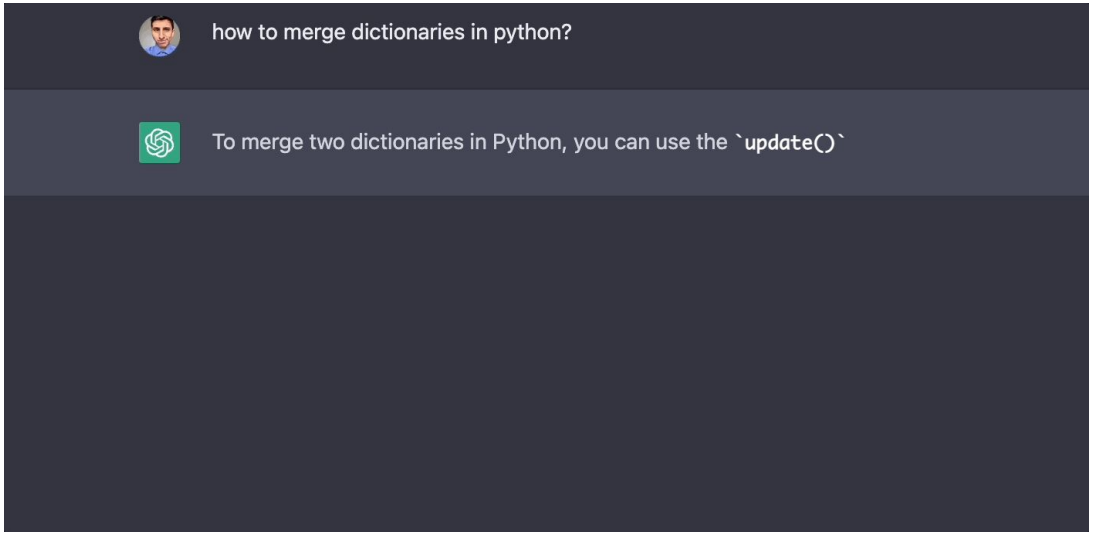
⇒ Asynchronous: If you receive a new document (embed it and store in vector DB)

Query embeddings: **Real-time**

Streaming response

Some models produce outputs as a continuous stream of information.

Think of LLMs producing answers words per words instead of a full text after ~10 seconds.



Streaming response

Some models produce outputs as a continuous stream of information.

Think of LLMs producing answers words per words instead of a full text after ~10 seconds.=

Websocket

- WebSockets allow full-duplex (bidirectional) communication between the client and server.
- Unlike REST, WebSockets keep the connection persistent, enabling real-time interaction.
- Typically used in chatbot applications, multiplayer games, and live notifications.

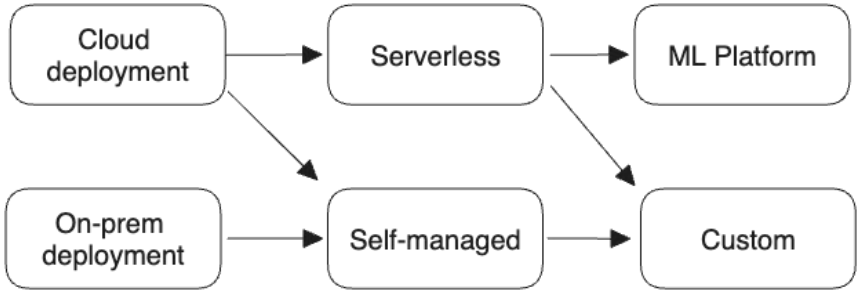
Hybrid approach: Use both online and offline

Pre-compute and store common queries

For example:

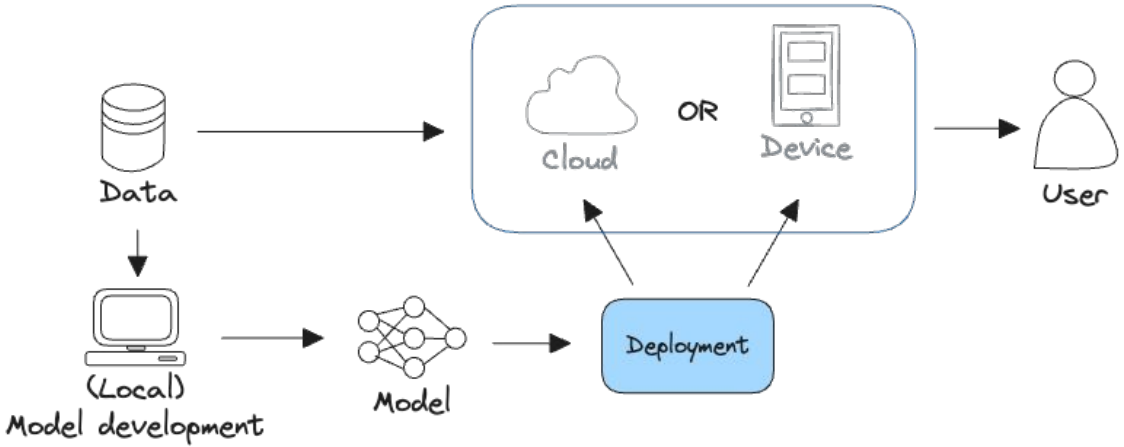
- DOORDASH
  - Restaurant recommendations use batch predictions
  - Within each restaurant, item recommendations use online predictions
- NETFLIX
  - Title recommendations use batch predictions
  - Row orders use online predictions

Cloud vs on-prem deployment



Model deployment

You need to store your model somewhere so it can be called to do predictions on new data.

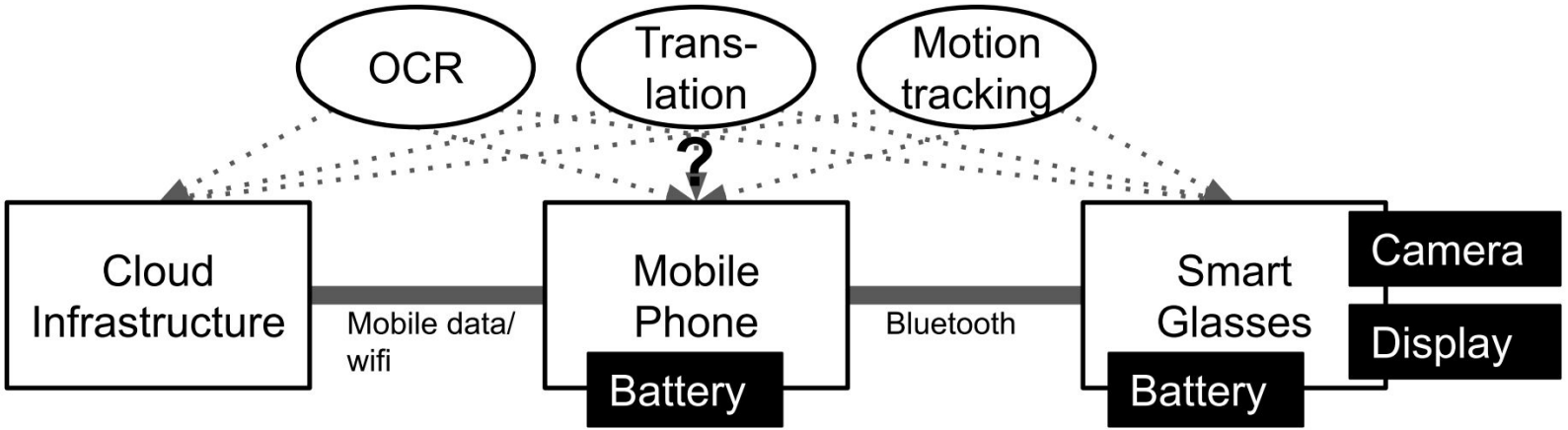


Example: Augmented reality translation

https://mlip-cmu.github.io/s2023/slides/09_deploying_a_model/deployment.pdf

Example: Augmented reality translation

Where should the models live?



https://mlip-cmu.github.io/s2023/slides/09_deploying_a_model/deployment.pdf

Considerations

- How much data is needed as input for the model?
- How much output data is produced by the model?
- How fast/energy consuming is model execution?
- What latency is needed for the application?
- How big is the model? How often does it need to be updated?
- Cost of operating the model? (distribution + execution)
- What happens if users are offline?

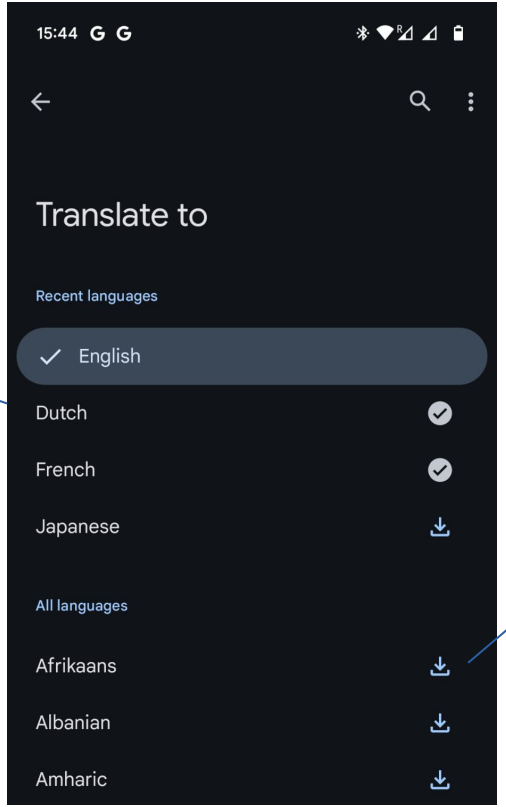
Cloud vs on-prem computing

	Cloud computing	**On-prem computing**
Computations	Done on **cloud** (servers). Model is stored on the Cloud and connected to through internet.	Model is stored on **on-prem** devices (own server, Jetson Nano, browsers, phones, smart watches, car, ...). Does not require internet connection, though it can help to update and maintain models.
Examples	• Most APIs • ChatGPT • Email spam filtering • …	• Next word predictions when typing in your smartphone • Self driving cars • Camera to detect where a rug needs to be cut

Example: Google Translate

Model downloaded on your phone (on-prem)!

Works offline.



Model on the cloud!

Needs internet.

LIÈGE université

Why on-prem deployment?

Pros	Cons
Privacy ● Sometimes hard requirement on data not leaving a specific location (hospitals, or personal data can't leave the EU due to GDPR) ● Less risk of data to be intercepted over network **Latency** ● No networking latency - can be faster ● Usually more of a monolith architecture which doesn't rely on microservices communicating to each other ● Even though **Unstable connexion** ● Can work where there is no internet connexion (e.g. self-driving car)	**Privacy** ● Actually pretty unsafe as someone can just physically intercept the data (steal the device) **Hardware** ● Can be really expensive to buy a performant hardware setup ● Don't have the "pay for what you use" - your training GPUs are wasted when nothing is running **Development** ● Often have to physically connect to the device ● Maintenance overhead (someone needs to physically go there if something crashes). **Scalability** ● Hard to increase to 20 new plants

Cloud deployment: Custom vs ML Platform deployment

Custom deployment

Containerise your model inference code as a **microservice** (as an **API**).

Spin up a compute instance on the Cloud and **host it yourself**.

Pros: Much more flexibility. Can add custom logic. Cheap.

Cons: Infrastructure implementation overhead. Time to deployment.

ML Platform deployment

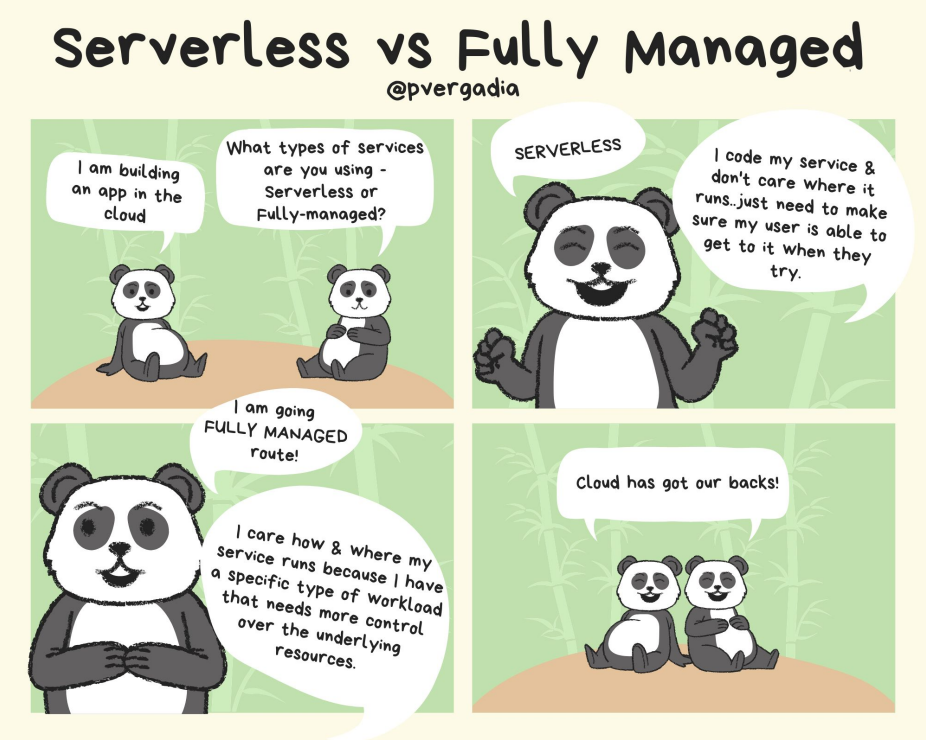
Use a **standard model framework** (Pytorch, Tensorflow, Huggingface, …) which can be deployed on a **ML Platform as a Service (Paas)** such as Vertex, Azure ML or Sagemaker.

The **infrastructure** scaling and overhead is **managed** by the **service provider**.

Pros: Less overhead, easy access to optimise infrastructure.

Cons: Limited customisation/flexibility. Hard to include custom logic. Expensive.

Cloud custom deployment: Serverless



Cloud custom deployment: Serverless

Serverless

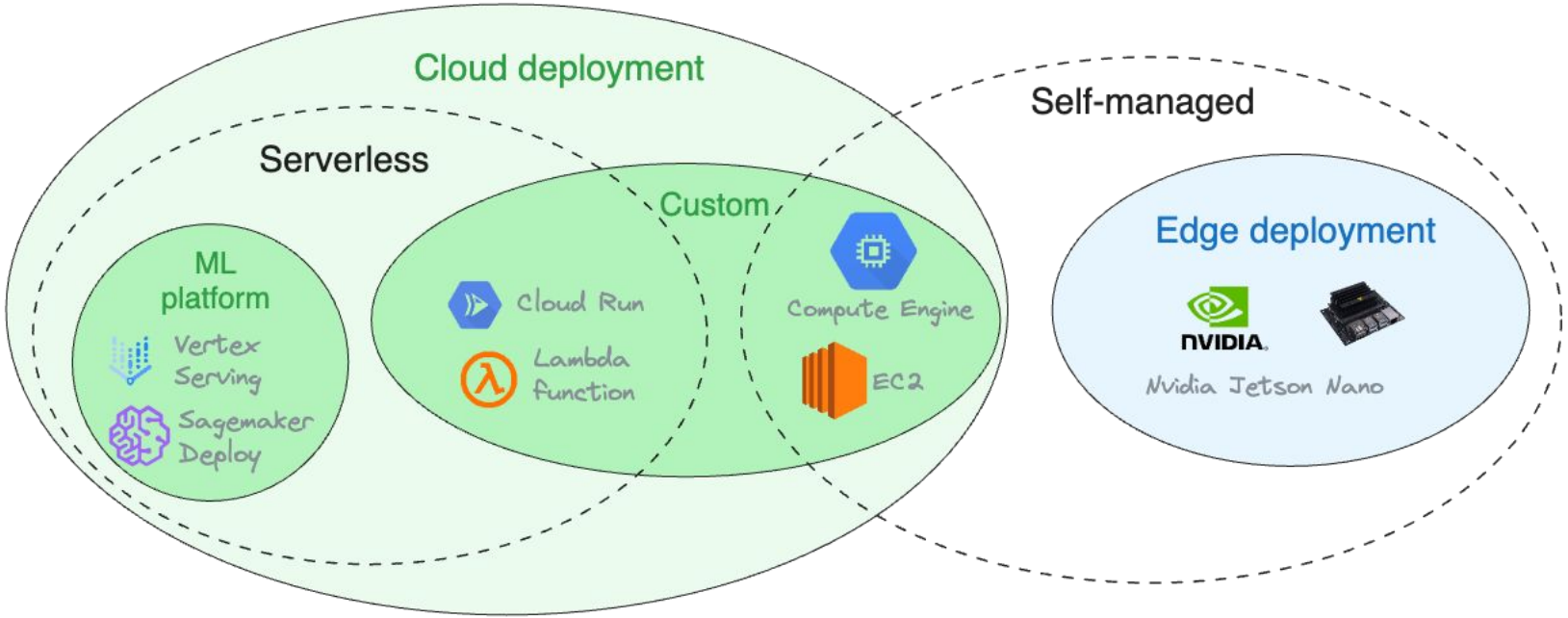
- Cloud-native
- Developers don’t have to manage servers
- Cloud provider abstracts away the provisioning, maintaining, and scaling the server infrastructure
- Developers just have to package their codes in a container
- Opposite is **self-managed**

Pros	Cons

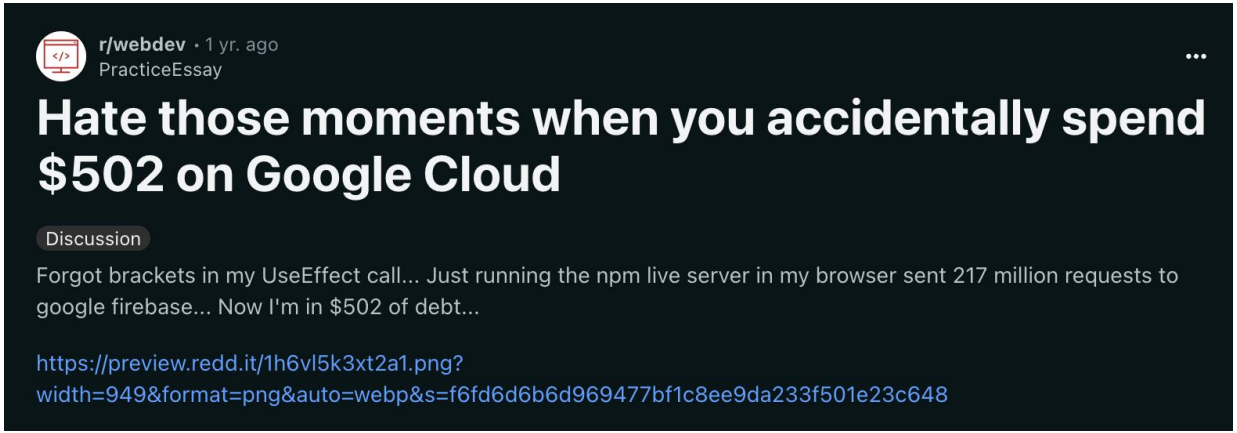
● More efficient use of time for developers
● Often cost effective (pay for what you use)
● Simplified operations
● Better adoption of DevOps / MLOps practices
  ○ Team collaboration / rotation
  ○ Stateless containerised application
	● Less control over the server architecture
	  ○ Type of compute
	  ○ Interaction between components
	● Vendor lock-in
	● Debugging

https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless

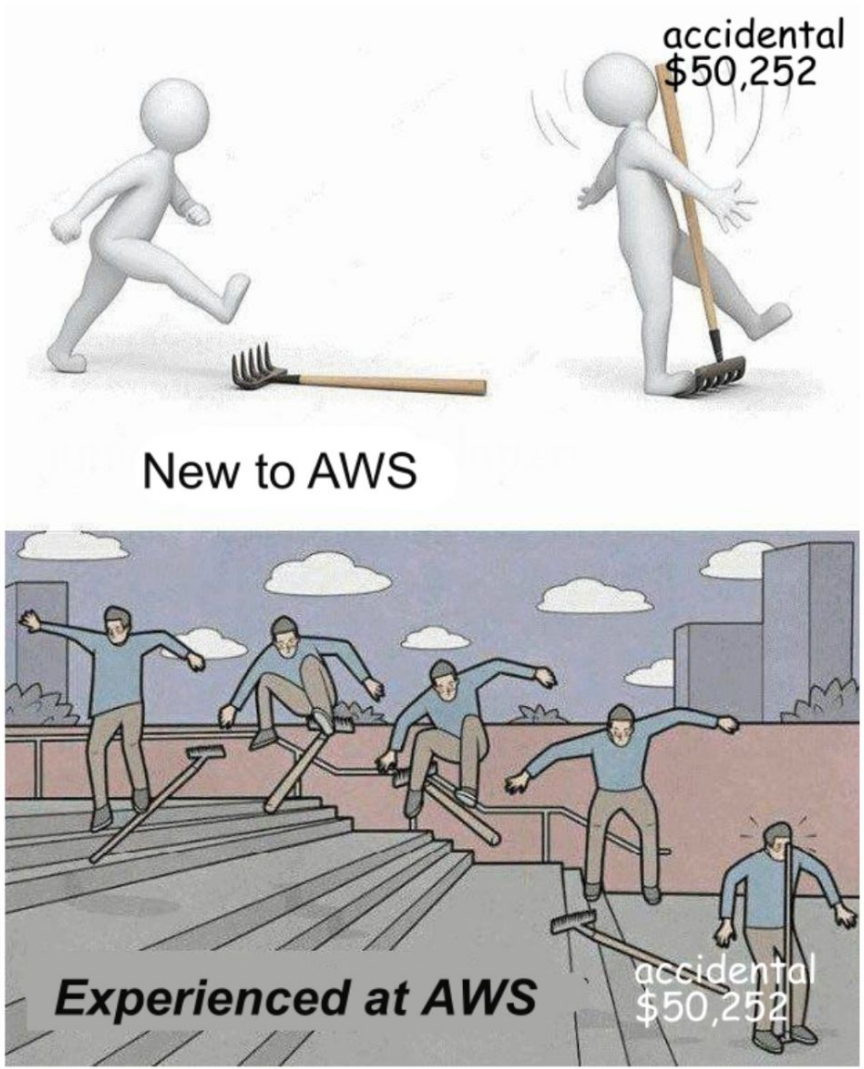
Example of different tools for different types of deployments



Parentheses: Cloud infrastructure incurs costs!



Parentheses: Clo



r/webdev · 1 y
PracticeEssay
Hate tho
$502 on
Discussion
Forgot brackets in r
google firebase... N
https://preview.redd
width=949&format

...
end
ests to

Hacker News new | past | comments |
login

appstorelottery on March 29, 2020 | parent | conte
A few years ago my startup was killed by a AV
figured he'd never make a $80k mistake again
overnight we were out of money. The mistake
hosting. Not kidding.

same provisioning mistake at his previous job - so I
did help shell out with me to cover the cost &
it me so hard I went back in time to PHP and shared

LIÈGE université

Parentheses: Ways to manage Cloud costs

- Make sure to shutdown idle instances!
- Select coherent compute instance size
- Use services that **scale to 0**
  - Spins an instance only when getting requests. Shuts it down after a while.
- You can set parameters to your autoscaling :
  - Max concurrent instances
  - Max instance size
  - Max cloud spend
- You can set alerts (emails etc)
- Good to frequently have a look at your Cloud cost breakdown and identify avoidable costs
- Use batch instead of streaming when possible
- Use custom serving instead of managed serving

GCP Cloud Run Pricing

Let’s look at some use cases where model latency is the difference between success or not…

Optimising latency in object detection on live combine harvest.

Finding Needles (and broken rice, leaves and ticks) in a literal haystack... but fast.

Use case: Installing a device on combines to automatically segment larger wheat objects and steering combine settings.

Type of serving: ??

LIÈGE université

Optimising latency in object detection on live combine harvest.

Finding Needles (and broken rice, leaves and ticks) in a literal haystack... but fast.

Use case: Installing a device on combines to automatically segment larger wheat objects and steering combine settings.

Type of serving: Real-time (synchronous) and on-edge

Improvements: ??

LIÈGE université

Optimising latency in object detection on live combine harvest.

Finding Needles (and broken rice, leaves and ticks) in a literal haystack... but fast.

Use case: Installing a device on combines to automatically segment larger wheat objects and steering combine settings.

Type of serving: On-edge and real-time (synchronous).

Improvements:

- Simplified model
- Optimised model (quantisation and pruning)
- Optimised framework (Tensorflow Lite)
  - (Deprecated)

Outcome:

- Faster than our competitors
- Cheaper hardware (200k $ difference)

LIÈGE université

Transcribing an archive database of call center audio recordings.

Use case: Large archive of call center a recordings to be transcribed using speech-to-text. Gain insights into phone calls in order to support and facilitate operators' tasks.

Type of serving: ??

Transcribing an archive database of call center audio recordings.

Use case: Large archive of call center a recordings to be transcribed using speech-to-text. Gain insights into phone calls in order to support and facilitate operators' tasks.

Type of serving: Batch.

Improvements:

- ??

Transcribing an archive database of call center audio recordings.

Use case: Large archive of call center a recordings to be transcribed using speech-to-text. Gain insights into phone calls in order to support and facilitate operators' tasks.

Type of serving: Batch.

Improvements:

- Not much possible with whisper out of the box
- Use lighter open source model (e.g. wav2vec)
- Hardware + cloud infrasctructure

Outcome: Blocking factor in the current track

https://huggingface.co/jonatasgrosman/wav2vec2-large-xlsr-53-english https://openai.com/research/whisper

Guest lecture: Connexion

LIÈGE université

vLLM on the Cloud with GPUs

Wrap-up

Lecture summary

Topic	Concepts	To know for…	
		Project	Exam
ML Model serving	• Batch vs real-time • Asynchronous vs synchronous • Push vs pull		Yes
ML Model deployment	• Cloud vs on-premise		Yes
Guest lecture	• Connexion		
Demo: vLLM		Possibly	

Project objective for sprint 4

#	Week	Work package	Requirement
4.1	W08	Build a **pipeline** to automatically run different sequential components such as training your model and deploying your model. For it you can use orchestrated pipeline tools such as Kubeflow Pipelines, AWS Sagemaker or GCP Vertex. **Attention**: If you run this pipeline in the Cloud it can incur Cloud **costs**. Make sure to use a platform where you have credits and not burn through them. You can ask for support from the teaching staff in that regard.	Optional

That’s it for today!

LIÈGE université