Model pipeline

Sprint 4 - Week 8

INFO 9023 - Machine Learning Systems Design

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

2025 Spring

Agenda

What will we talk about today

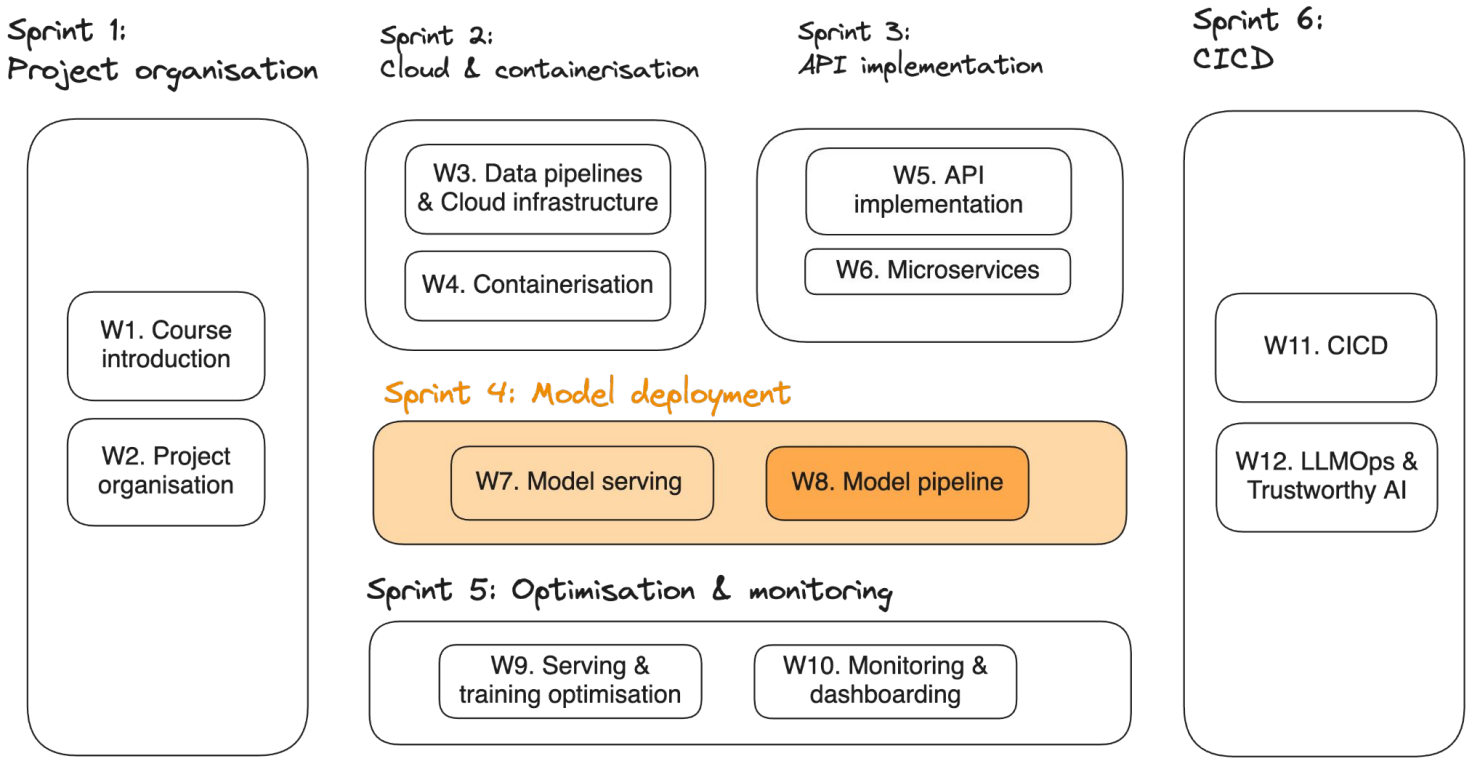
Lecture

1. Parallel and Distributed Training
2. Model complexity optimisation
3. Model serving optimisation
4. *ML model pipeline* → If we have enough time
5. *ML platforms & orchestrators*

Directed Work

1. ~~Vertex Pipeline~~

Status on our overall course roadmap



Group projects

Collaboration rules recap

- Final grades for the project are **personal**
- They take into account personal **contribution** to the project

Therefore, make sure that each team members **demonstrate works** on the project

- Split *presentation time* and *answering questions* during MS presentations
- Be active in pushing codes to github

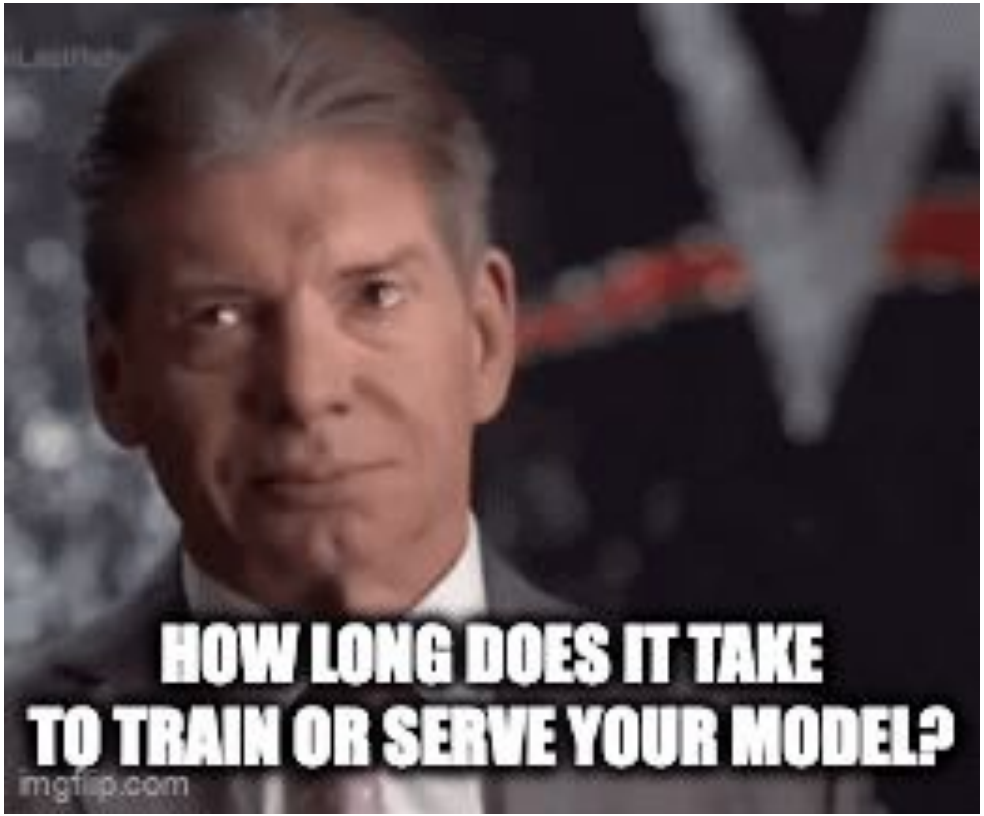
Additional directed work

Collaboration rules recap

Today will be another directed work! You'll get a pass/fail which will count for your final grade.

That means there will be more than the originally announced 5 directed works.

But all directed works will still sum up to 20% of your grade.


HOW LONG DOES IT TAKE
TO TRAIN OR SERVE YOUR MODEL?
imgflip.com

Breakout exercise

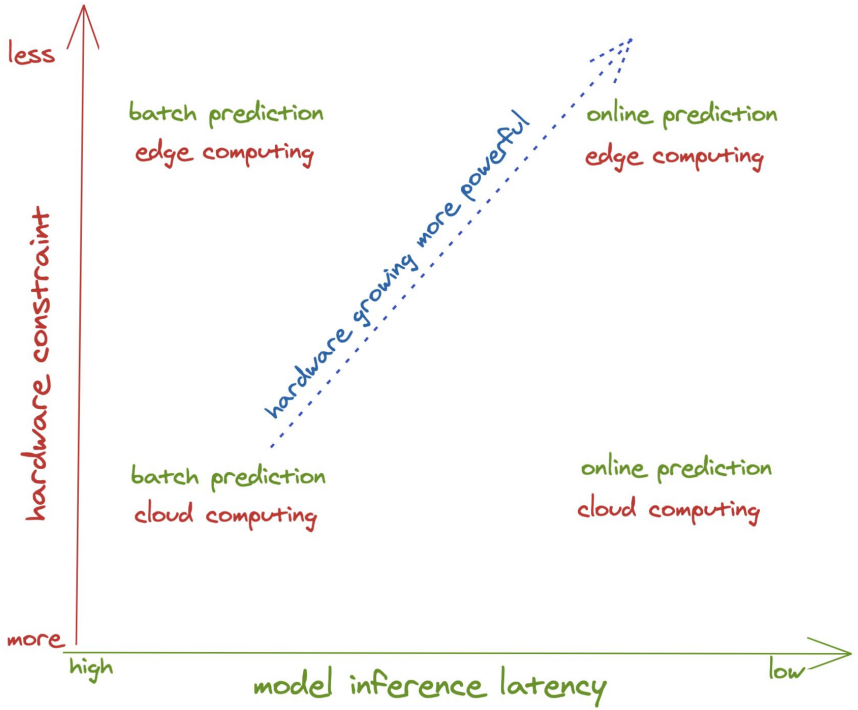
Group of ~4, duration 10 minutes

Identify 2 applications for each quadrant.

How do you determine:

- Batch vs. online prediction
- Edge vs. cloud

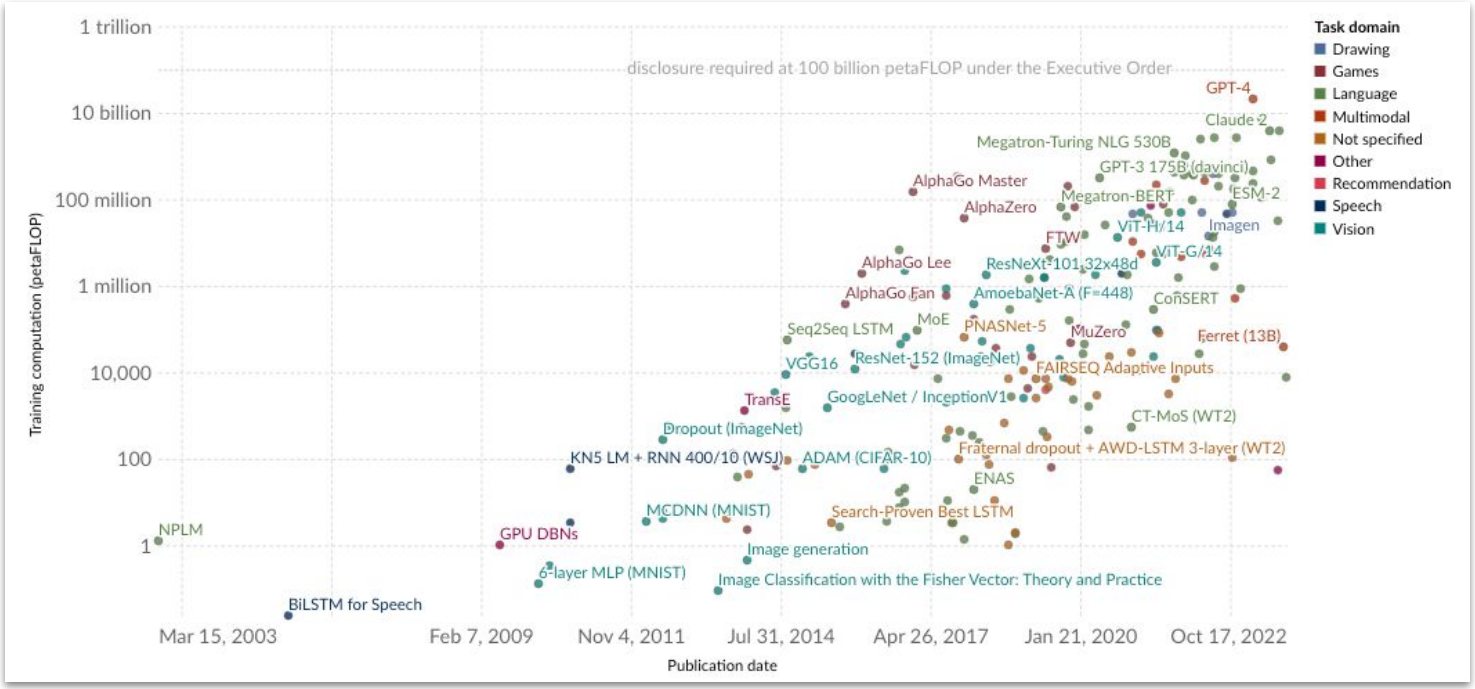
Hints: Look at some of the applications on your phone.



Computational cost of training ML models

Training ML models is often a computationally heavy task

LLMs are pushing the boundaries



https://ourworldindata.org/grapher/artificial-intelligence-training-computation?time=2003-03-15..latest

Calculate the size of an ML training job

Method 1: Counting operations in the model

How do we represent a single operation?

Floating Point Operation (FLOP): A single operation on a floating variables. Typically used to calculate computer performance (FLOP / seconds = FLOPS) and the total number of operations for a specific computational job.

PetaFLOP: ~ 10^{15} FLOPs.

PetaFLOP-day: The number of FLOP done by a machine doing 1 petaFLOPS if it was running for a full day.

Name	Unit	Value
kiloFLOPS	kFLOPS	10^3
megaFLOPS	MFLOPS	10^6
gigaFLOPS	GFLOPS	10^9
teraFLOPS	TFLOPS	10^{12}
petaFLOPS	PFLOPS	10^{15}
exaFLOPS	EFLOPS	10^{18}
zettaFLOPS	ZFLOPS	10^{21}
yottaFLOPS	YFLOPS	10^{24}
ronnaFLOPS	RFLOPS	10^{27}
quettaFLOPS	QFLOPS	10^{30}

Apple M2 GPU = 3.6 TFLOPS

LIÈGE université

https://en.wikipedia.org/wiki/Floating_point_operations_per_second

Calculate the size of an ML training job

Method 1: Counting operations in the model

How do we count the number of operations?

$$\text{training\_compute} = (\text{ops\_per\_forward\_pass} + \text{ops\_per\_backward\_pass}) * \text{n\_passes}$$

$$\text{n\_passes} = \text{n\_epochs} * \text{n\_examples}$$

Layer	# parameters	# floating point operations
Fully connected layer from N neurons to M neurons	N*M + M ≈ N*M	2*N*M+M+M ≈ 2*N*M
	WEIGHTS BIASES	NONLINEARITIES

Number of operations in a single forward pass.

https://epochai.org/blog/estimating-training-compute

Calculate the size of an ML training job

Method 2: GPU time

GPU-days describe the accumulated number of days a single GPU has been used for the training.

If the training lasted 5 days and a total of 4 GPUs were used, that equals 20 GPU-days.

Downside: Dependent of what type of GPUs were used…

https://epochai.org/blog/estimating-training-compute

Calculate the size of an ML training job

Method 2: GPU time

From GPU-days to FLOP: Let's take Image GPT

*"[..]iGPT-L was trained for roughly **2500 V100-days** […]"*

- **Tensor performance:** We see in the V100 specification that it performs at **125 TeraFLOPS**
- **Usage**: Hard to make full usage of the tensor performance, we assume **30%**
- **Time**: It ran for **2500 days**. And each day is made of **86400 seconds**

SPECIFICATIONS

	V100 PCIe	V100 SXM2	V100S PCIe
GPU Architecture		NVIDIA Volta	
NVIDIA Tensor Cores		640	
NVIDIA CUDA® Cores		5,120	
Double-Precision Performance	7 TFLOPS	7.8 TFLOPS	8.2 TFLOPS
Single-Precision Performance	14 TFLOPS	15.7 TFLOPS	16.4 TFLOPS
Tensor Performance	112 TFLOPS	125 TFLOPS	130 TFLOPS

NVIDIA V100 specifications

Total number of FLOP to train Image GPT =

30% x 125e12 FLOPS x 2500 days x 86400 sec/day = **8.1e21 = 8.1e6 PetaFLOP = 8.1 ZettaFLOP**

https://epochai.org/blog/estimating-training-compute

Parallel and Distributed Training

Why should we look at parallelisation?

Model training is too slow? You have multiple GPUs? ⇒ Use **parallelisation**

An ML model training works as different iterations of updating the model weights/parameters.

How to parallelise this process?

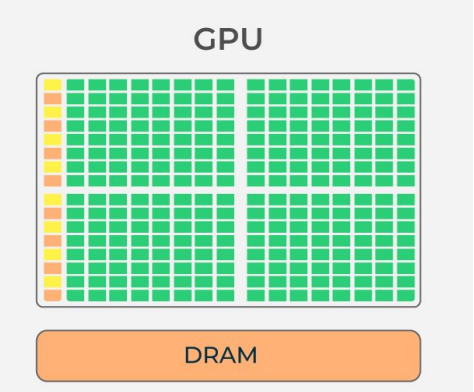
```
# Essential working of ML model training

for step in range(training_steps):
  # Each loops changes the weights to reduce loss,
  # based on a data sample
  weights, loss = update(weights, data)

  # If the loss is low enough the training stops
  if loss <= exit_loss:
    break
```

Why should we look at parallelisation?

Both due to model architecture and GPU hardware design

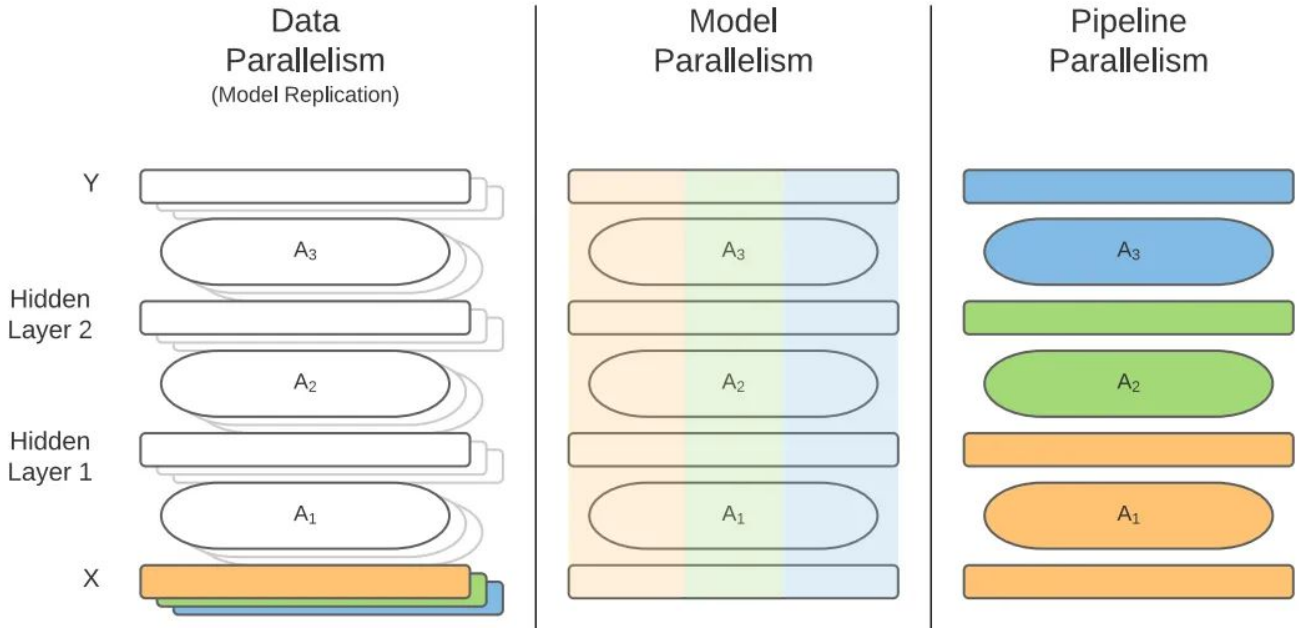
A **GPU** (Graphics Processing Unit) is designed for massively **parallel computation**. CPU has a few powerful cores that are designed for general-purpose, Sequential Processing. GPU has many small cores that are designed especially for tasks like graphics rendering and parallel computation.

Pros	Cons
• Parallel Processing • High Throughput • Graphics Rendering	• Not Versatile (cannot be used for tasks which require processing, such as operating systems or everyday applications) • Power Consumption • Cost



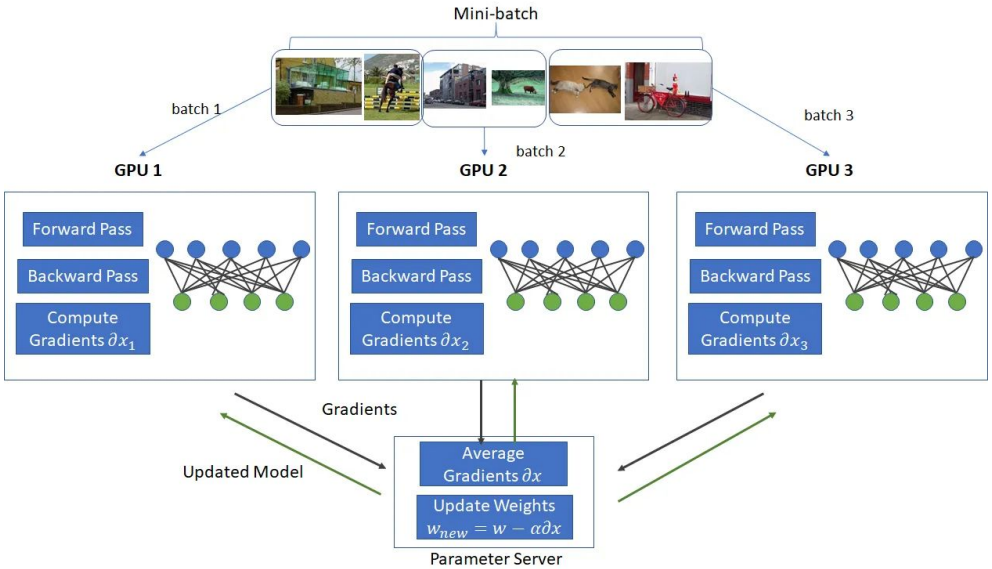
https://www.geeksforgeeks.org/difference-between-cpu-and-gpu/

Overview of the different methods for model parallel training



https://betterprogramming.pub/parallel-and-distributed-training-in-deep-learning-for-beginners-part-1-introduction-612a4534a117

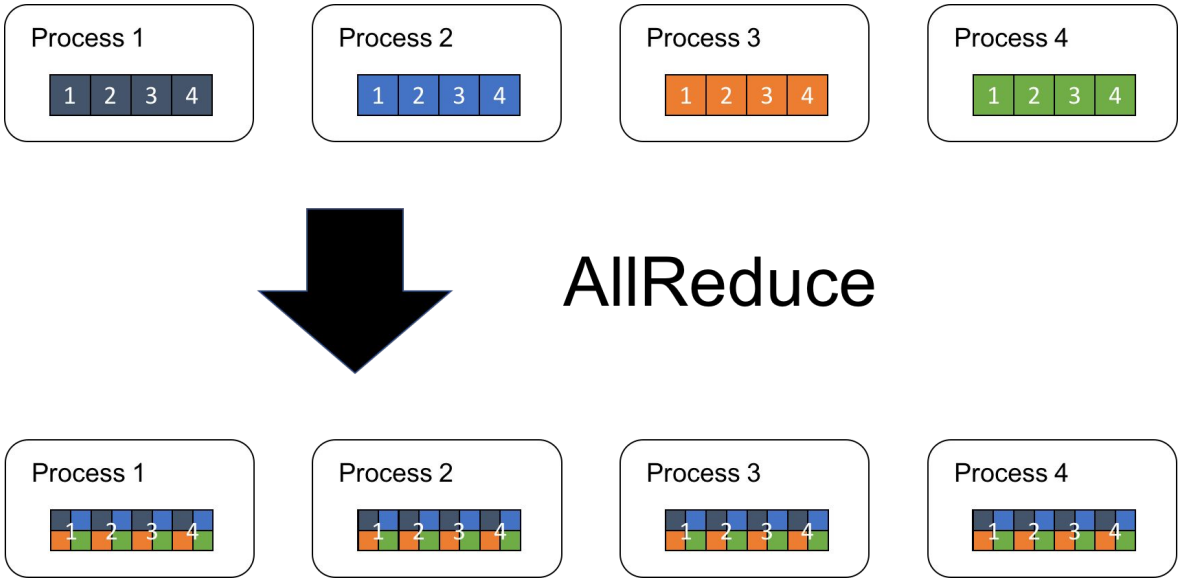
1. Data parallelism



Split a single batch into equal size chunks called **mini-batches (MBS)**, or micro-batches.
Compute the forward and backward passes on all MBS in parallel on **different workers/machines**.
Use (e.g.) an all-reduce algorithm to aggregate the results of each MBS and compute the final weights.

https://www.telesens.co/2017/12/25/understanding-data-parallelism-in-machine-learning/

1. Data parallelism



The parameter server then updates the weights for all workers before starting a new step with a new data batch.

https://tech.preferred.jp/en/blog/technologies-behind-distributed-deep-learning-allreduce/

1. Data parallelism

Pros	Cons
• Easy to implement, usually has available implementations already — DP and DDP in PyTorch. Or MirroredStrategy in Tensorflow. • Fully model-agnostic — works with anything (CNNs, Transformers, GANs, etc.) without modifications. • Easy to predict speed improvements before running (e.g. if we use 4 GPUs, then the convergence will be at most 4x quicker).	• Requires the model to fit into the memory of a single worker. • Scalable only up to a point (without further tricks and optimizations, such as LAMB) 4 → 8 batch size = Optimisation 1024 → 2048 batch size = Each worker get 2x more compute so no optimisation • Big communication payload if you have a large models with a lot of parameters. You need to send each parameters back and forth to all workers at each batch… ResNet50 = 24M parameters = 1.47GB • Soft requirement for GPUs to be on the same machine or on a local network

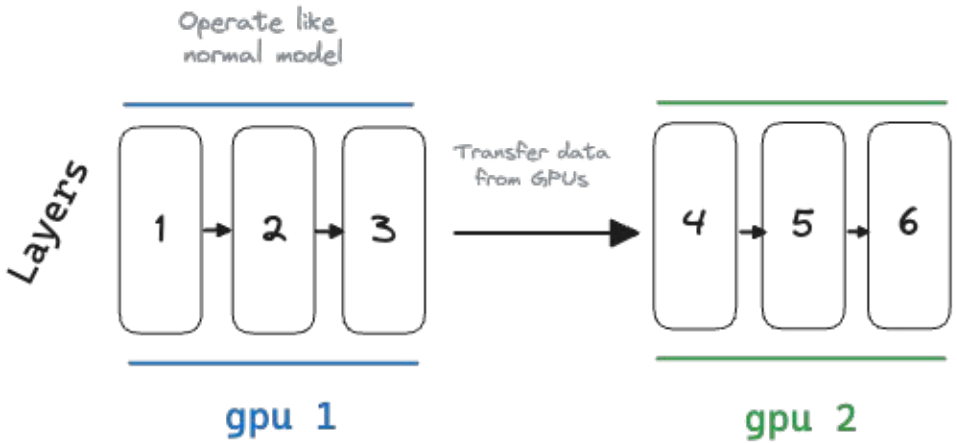
https://betterprogramming.pub/parallel-and-distributed-training-in-deep-learning-for-beginners-part-1-introduction-612a4534a117

2. Naive model parallelism

Naive (or vertical) model parallelism spreads groups of **model layers across multiple GPUs**.

Transmit information at each pass between GPUs. Because to compute layer l, you typically need all outputs from layer l-1, so you need to wait until all of the workers computing layer l-1 have finished.

It is almost exclusively used within a single physical cluster due to high bandwidth between devices (GPUs) the motherboard provides, rather than across the network.



https://huggingface.co/docs/transformers/v4.15.0/en/parallelism

2. Naive model parallelism

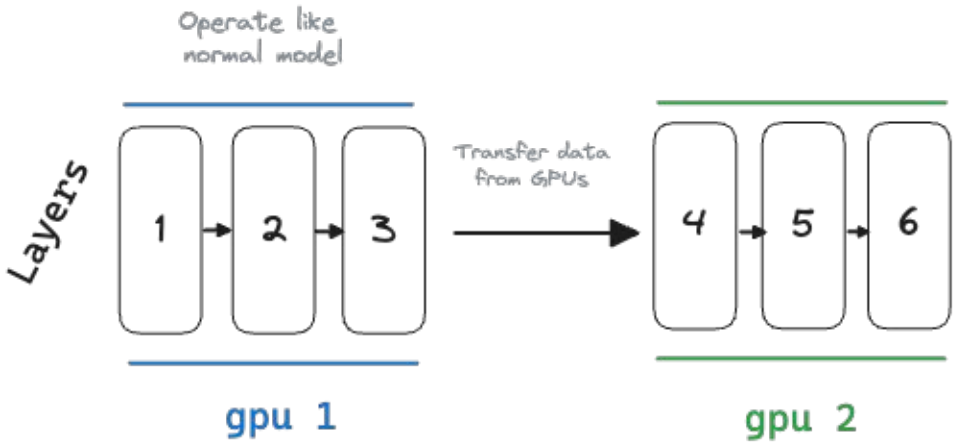
Overhead in copying data:

E.g.

1x 24GB card = 4x 6GB cards using naive MP

↓

Slower due to data transfer



https://huggingface.co/docs/transformers/v4.15.0/en/parallelism

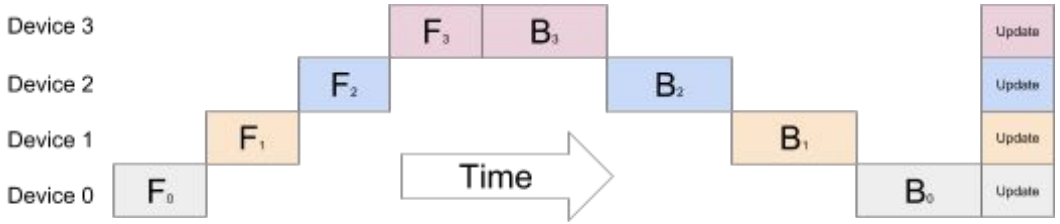
2. Naive model parallelism

Pros	Cons
● Greatly reduced memory usage, proportional to the number of devices used. Relevant in the era of Large models ● Can use NVIDIA's Megatron-LM library	● Model-dependent — different way to divide the model which is dependent on the model architecture ● Expensive communications ● Idle time for your GPUs while other groups of layers are being processed ● It's hard! Implementing Model Parallelism in your ML program is typically done by hand, requires knowledge of your model and its layers, as well as how they interact with your hardware

https://betterprogramming.pub/parallel-and-distributed-training-in-deep-learning-for-beginners-part-1-introduction-612a4534a117

2. Naive model parallelism

Long idle time of your GPUs.



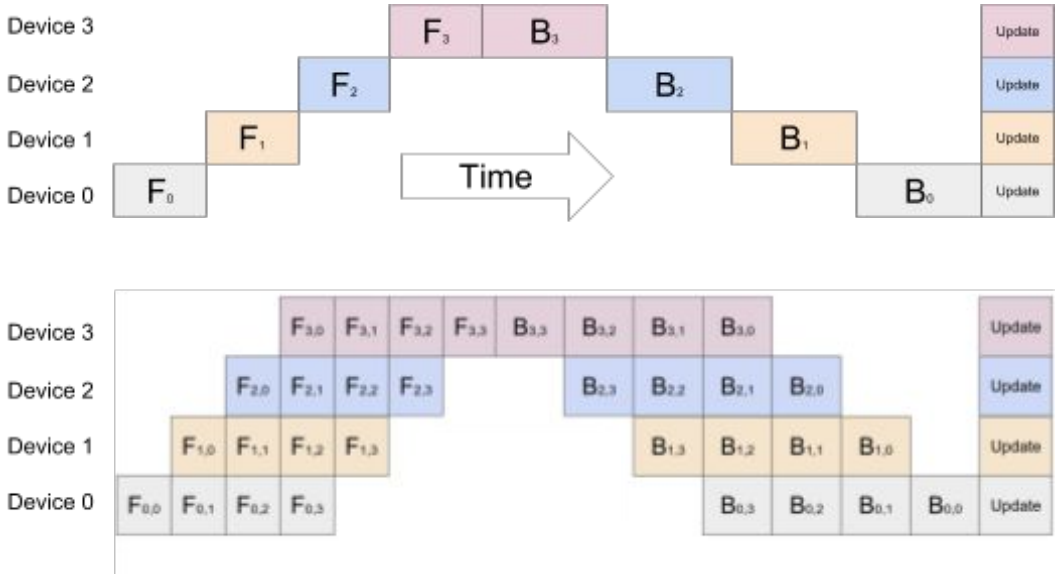
Model parallelism

https://blog.research.google/2019/03/introducing-gpipe-open-source-library.html

3. Pipeline parallelism

Solution: Mix of data and model parallelism.

Split data in mini-batches and compute passes for different groups of layers sequentially.



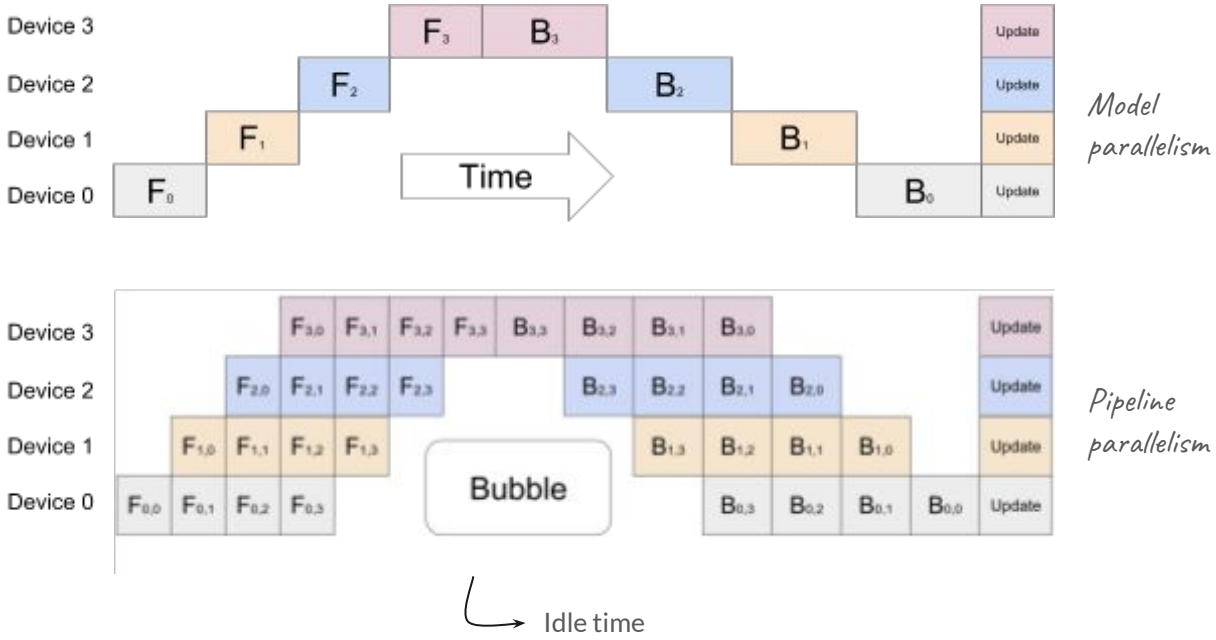
https://blog.research.google/2019/03/introducing-gpipe-open-source-library.html

3. Pipeline parallelism

Still some idle time called the **bubble**.

Traditional Pipeline API solutions:

- PyTorch
- FairScale
- DeepSpeed
- Megatron-LM



https://blog.research.google/2019/03/introducing-gpipe-open-source-library.html

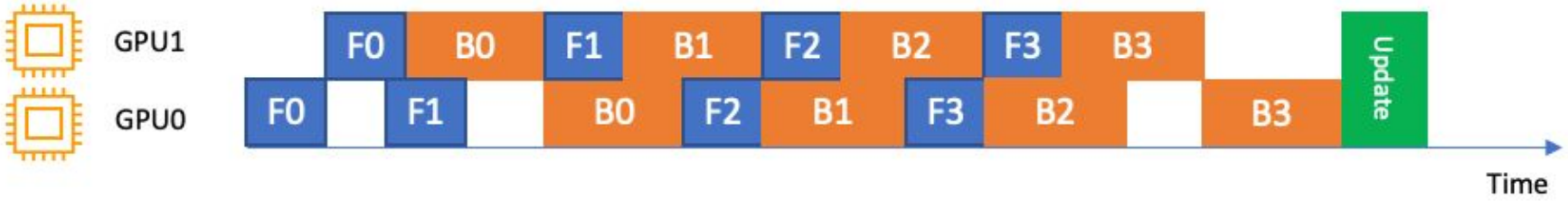
3. Pipeline parallelism

Pros	Cons
• Reduces memory usage **and** GPU idle time	• More complexity (extra hyperparameter to optimise) • Limited framework. PyTorch requires a single Tensor or a tuple of Tensors as the only input and output. No other python variables can be passed. • Have to arrange each layer so that the output of one model becomes an input to the other model.

https://huggingface.co/docs/transformers/v4.15.0/en/parallelism

3. Pipeline parallelism

Interleaved Pipeline: Further reduce the bubble (idle) time by prioritising backward passes.



Supported by DeepSpeed, Varuna and SageMaker.

https://docs.aws.amazon.com/sagemaker/latest/dg/model-parallel-core-features.html

When to use which parallelisation technique?

Technique	Implementation complexity	Model size & structure	Hardware
Data parallelism	**Simple**. Integrated in frameworks like Pytorch or Tensorflow.	Your **model fits into a single GPU** (usually small batch size - e.g. 64).	Works on both **single-machine** and **multi-machine** configurations.
Model parallelism	**More complex**.	Model **does not fit into a single device**.	Recommended on **single machine** due to **high communication costs**.
Pipeline parallelism	**Complex** but supported out-of-the-box by some frameworks	Model **does not fit into a single device**. Works well for sequential models like CNN and transformers.	Recommended on **single machine** due to **high communication costs**. High machine utilization and memory efficiency.

https://betterprogramming.pub/parallel-and-distributed-training-in-deep-learning-for-beginners-part-1-introduction-612a4534a117

Ray

Ray is a framework that provides a simple, universal API to build distributed ML applications. It is designed to scale Python applications from a single computer to a cluster with ease.

- **Does not provide hardware**: Ray itself doesn't provide the physical compute resources (like CPUs or GPUs).
- **Scaling**: Mostly used when transitioning from a single machine to distributed training.
- **Model integration**: Integrates directly with Pytorch, Huggingface or Scikit-learn.
- **Cloud Integration**: You can deploy Ray easily on Azure, GCP or AWS.

Model complexity optimisation

Difference between large and simple models

Large models

- More features, hidden units in Neural Network, trees for decision trees, parameters, …
- Capture more complex relationship in the data
- Generalises better
- More memory intensive

⇒ Slower!

For accuracy sensitive complex use cases where a lot of training data is available.

VS

Simple models

- Less features, smaller Neural Network, less trees, less parameters, …
- Fits less well on complex relationship in the data
- For simpler tasks
- Requires less memory

⇒ Faster!

For low-latency or cost efficient models capturing simpler systems.

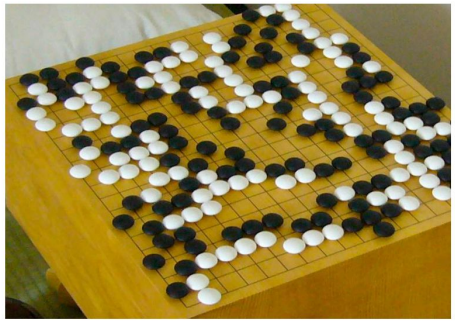
Clear objective: What are your speed requirements?

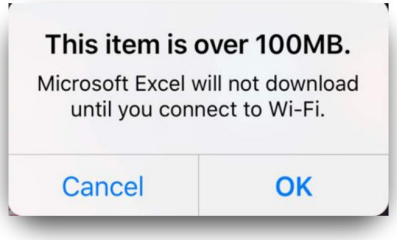
- Satisficing metrics: E.g. prediction latency, cost, … ⇒ Set a threshold
- Optimising metrics: Accuracy, precision, recall, … ⇒ Optimise

Nevertheless, sometimes you start with a large model which you want to reduce size and complexity.

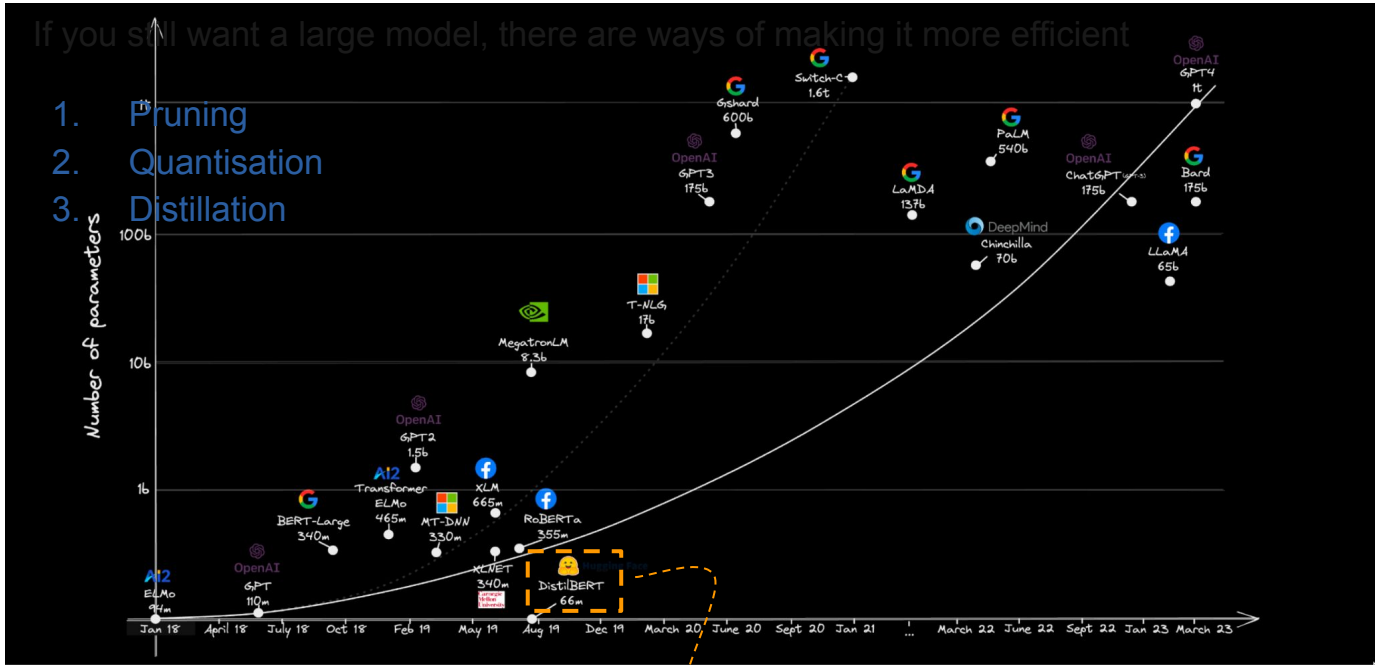
- Model speed
  - *E.g. not detecting pedestrian in time*
- Model size
  - *E.g. uploading/updating model over network*
- Energy efficiency
  - *E.g. AlphaGo: 1920 CPUs and 280 GPUs*

⇒ $3000 electric bill per game





Optimising your model can be done through model optimisation.

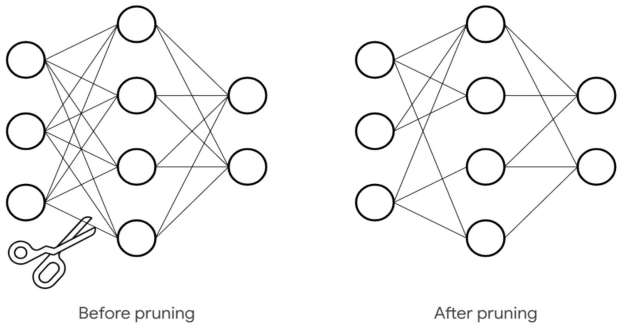


97% of the accuracy for 50% of the size of BERT - its original model

LIÈGE université

Pruning for model compression.

Pruning: Removing underutilised weight connections in a network to increase inference speed and decrease model storage size.



Before pruning

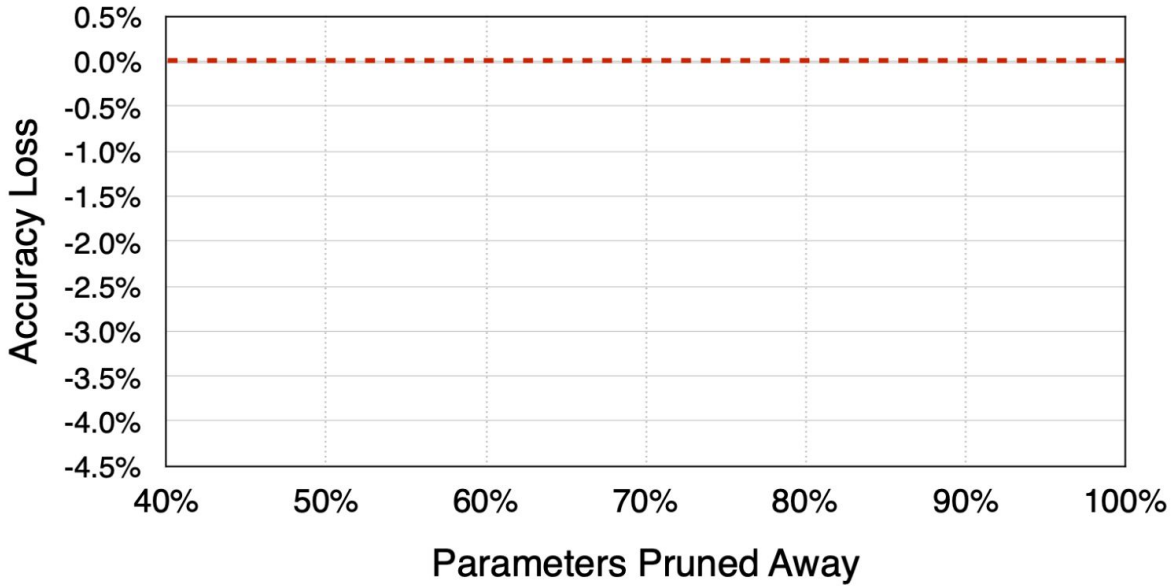
After pruning

Model	Non-sparse accuracy (float)	Sparse accuracy (float)	Number of non-zero parameters (NNZ) in sparse models
InceptionV3	78.1%, 27.1M parameters	78.0% @ 50% sparsity	13.6M
		76.1% @ 75% sparsity	6.8M
		74.6% @ 87.5% sparsity	3.3M
GNMT EN-DE	26.77 BLEU, 211 M parameters	26.86 BLEU @ 80% sparsity	44 M
		26.52 BLEU @ 85% sparsity	33 M
		26.19 BLEU @ 90% sparsity	22 M
GNMT DE-EN	29.47 BLEU, 211 M parameters	29.50 BLEU @ 80% sparsity	44 M
		29.24 BLEU @ 85% sparsity	33 M
		28.81 BLEU @ 90% sparsity	22 M

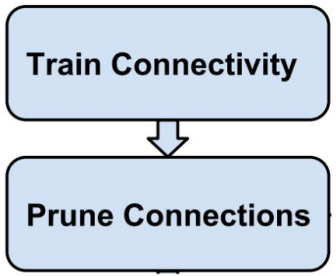
TensorFlow Model Optimization Toolkit

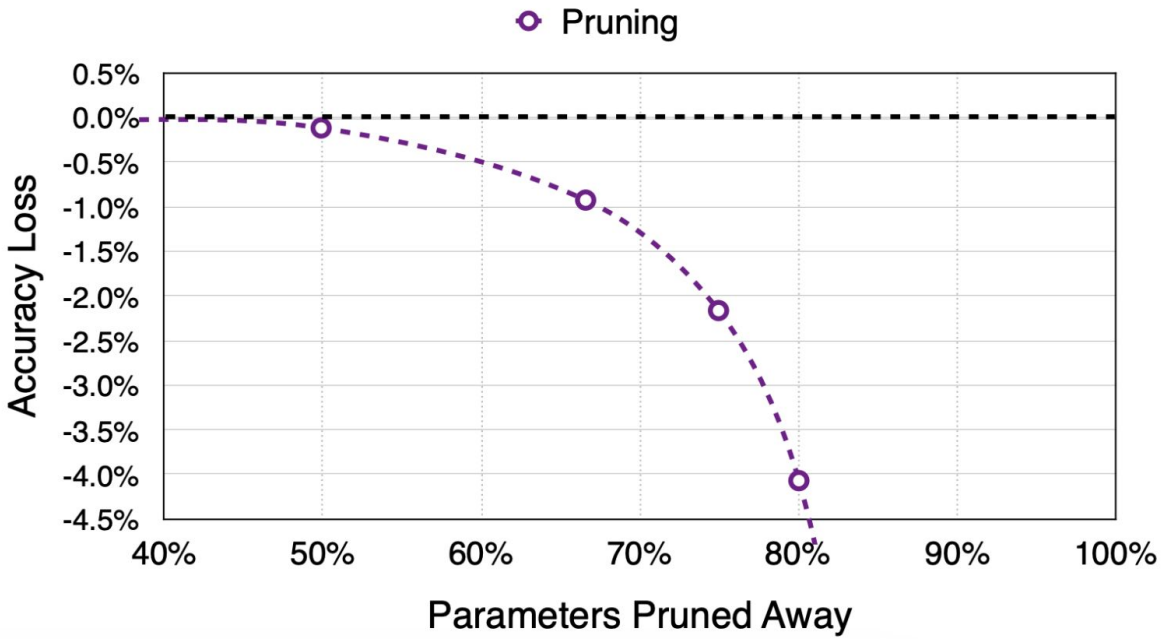
Pruning for model compression.

Train Connectivity

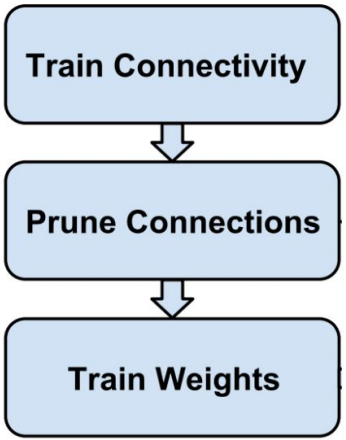


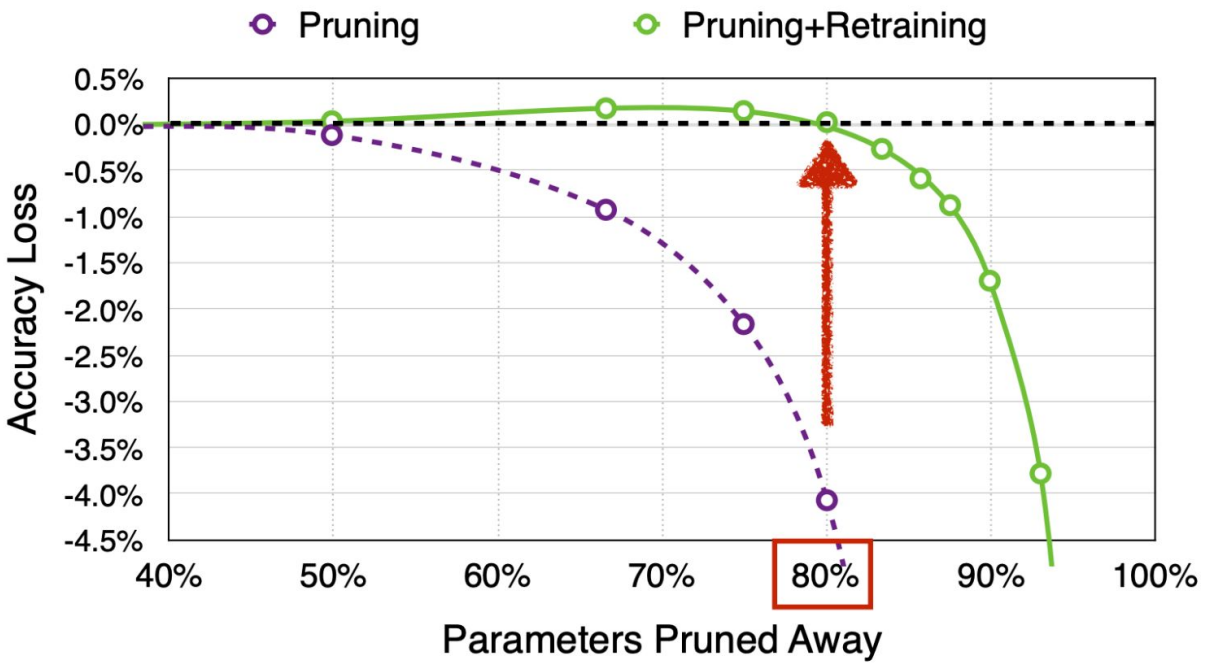
Pruning for model compression.



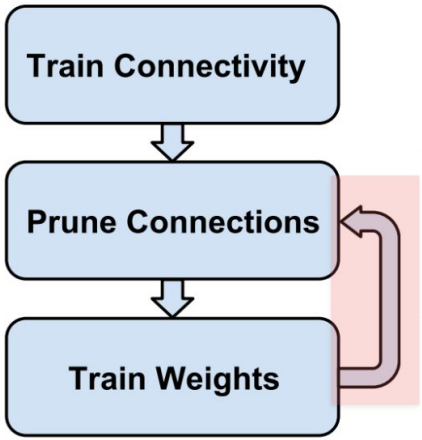


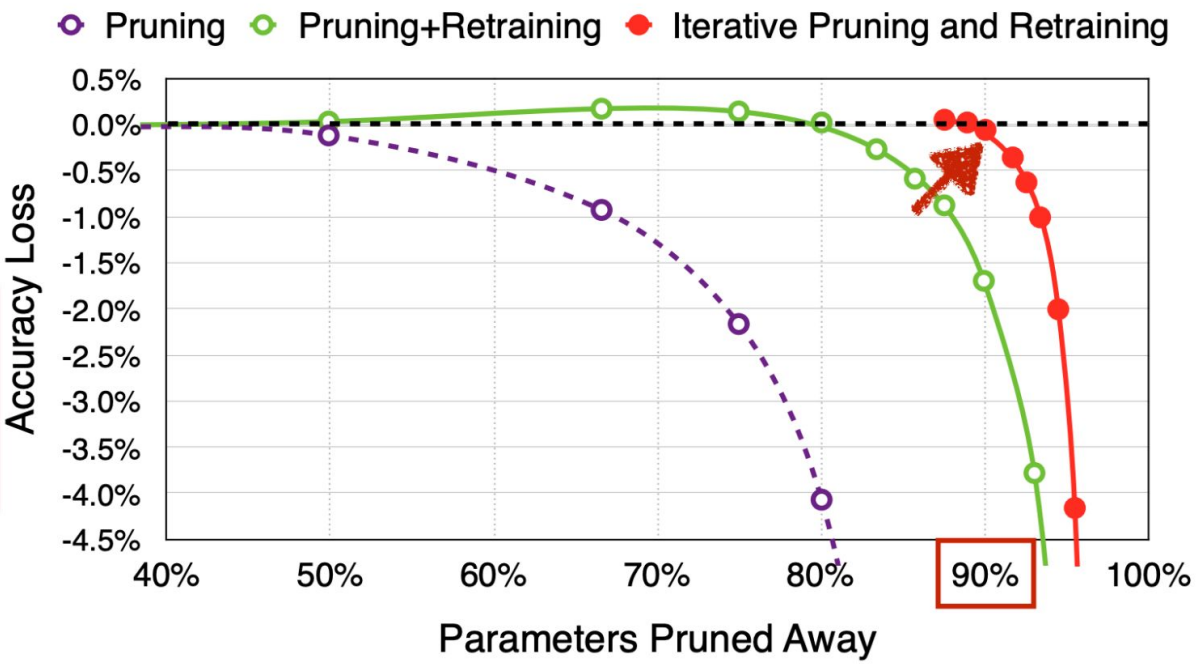
Pruning for model compression.





Pruning for model compression.





Pruning happens in human brain.

50 Trillion Synapses

This image is in the public domain

Newborn

1000 Trillion Synapses

This image is in the public domain

1 year old

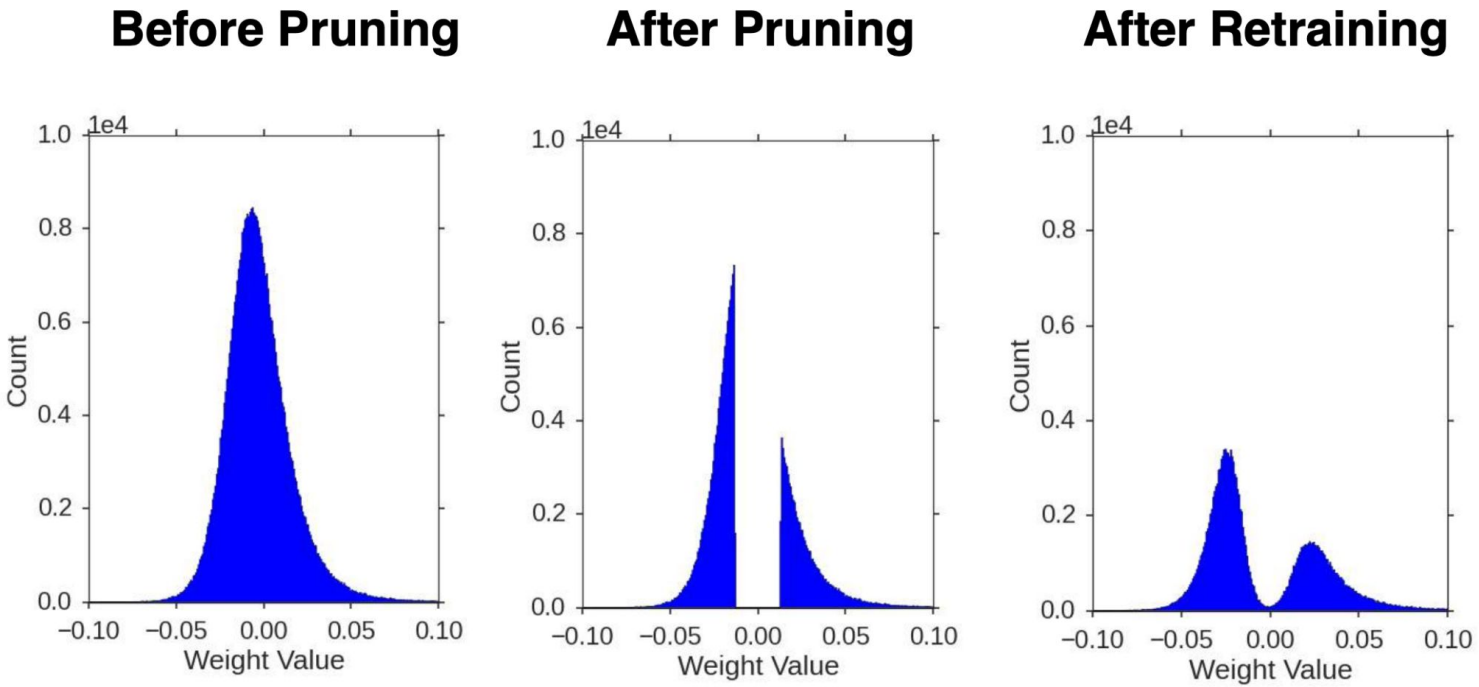
500 Trillion Synapses

This image is in the public domain

Adolescent

Christopher A Walsh. Peter Huttenlocher (1931-2013). Nature, 502(7470):172–172, 2013.

Pruning changes weight distributions.

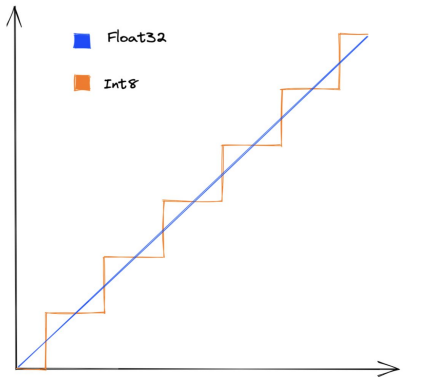


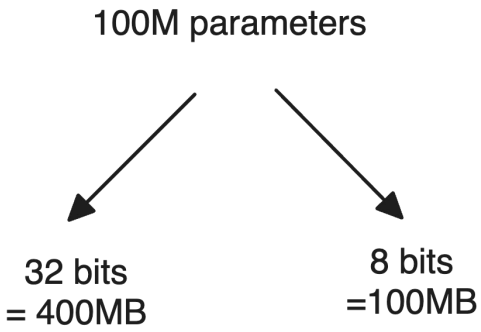
Christopher A Walsh. Peter Huttenlocher (1931-2013). Nature, 502(7470):172–172, 2013.

Quantisation

Quantisation: Reduction in precision of the numbers used to represent a models parameters.

E.g. Change the weights of a model from float32 to int8.





Quantisation

- **Post-training quantization**: This involves applying quantization to a fully trained model. The weights and activations are converted from floating-point to lower-precision formats, typically 8-bit integers.
- **Quantization-aware training**: This involves simulating low-precision weights and activations during the training process itself. By incorporating quantization into the training, the model can adapt to the reduced precision and often results in higher accuracy compared to post-training quantization.
- **Dynamic quantization**: This approach quantizes the weights of the model ahead of time but quantizes the activations dynamically at runtime. This is particularly useful for models with recurrent layers, where the input data shape varies from one batch to another.

Quantisation

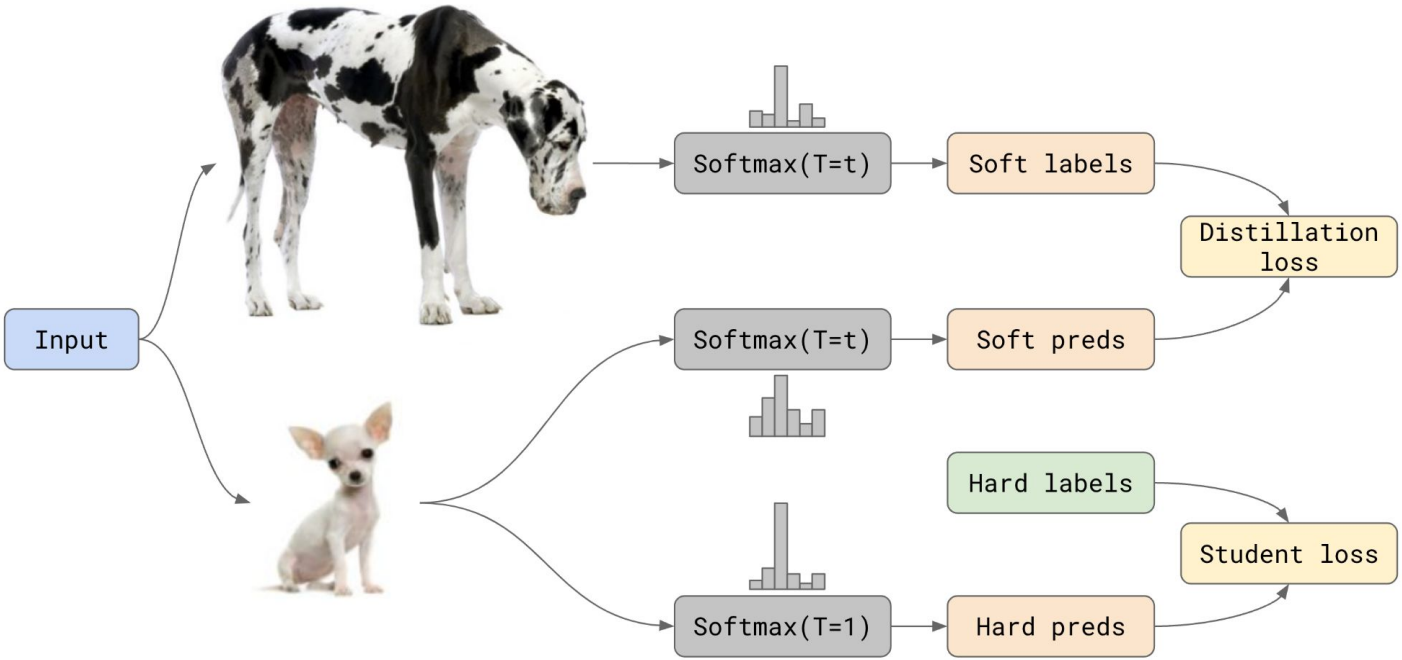
Model	Top-1 Accuracy (Original)	Top-1 Accuracy (Post Training Quantized)	Top-1 Accuracy (Quantization Aware Training)	Latency (Original) (ms)	Latency (Post Training Quantized) (ms)	Latency (Quantization Aware Training) (ms)	Size (Original) (MB)	Size (Optimized) (MB)
Mobilenet-v1-1-224	0.709	0.657	0.70	124	112	64	16.9	4.3
Mobilenet-v2-1-224	0.719	0.637	0.709	89	98	54	14	3.6
Inception_v3	0.78	0.772	0.775	1130	845	543	95.7	23.9
Resnet_v2_101	0.770	0.768	N/A	3973	2868	N/A	178.3	44.9

TensorFlow Lite documentation

Quantisation

Pros	Cons
● Reduce memory footprint ● Increase computation speed ● Bigger batch size ● Computation on 16 bits is faster than on 32 bits	● Smaller range of values ● Values rounded to 0 ● Need an efficient rounding technique

Knowledge distillation: training a smaller model to mimic a larger one based on high Softmax temperature.



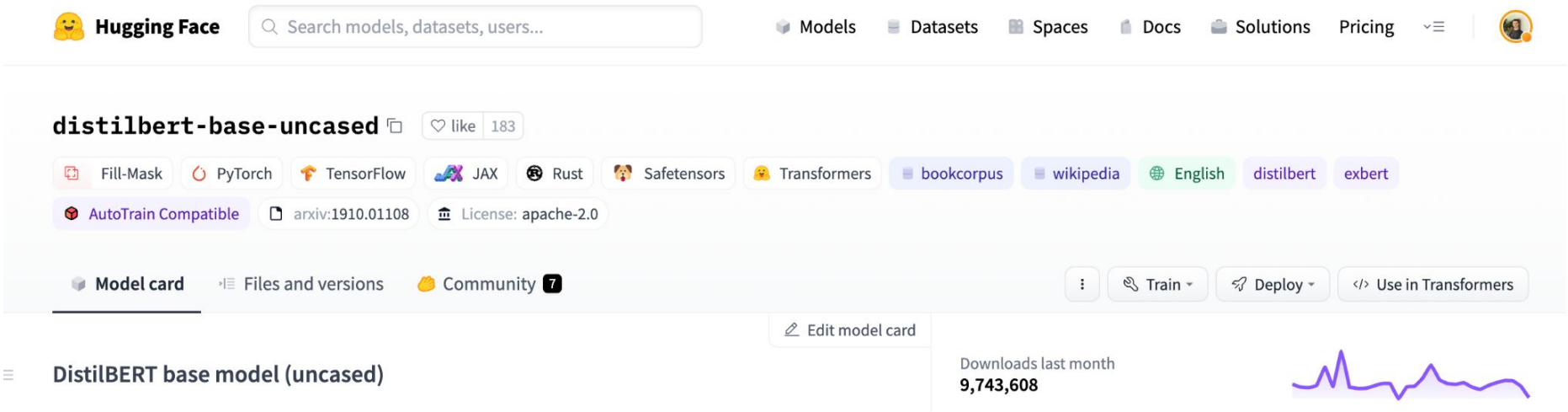
Interesting example: DistilBERT

Knowledge distillation: training a smaller model to mimic a larger one based on high Softmax temperature.

Pros	Cons
● Fast to train student network if teacher is pre-trained. ● Teacher and student can be completely different architectures.	● If teacher is not pre-trained, may need more data & time to first train teacher. ● Sensitive to applications and model architectures.

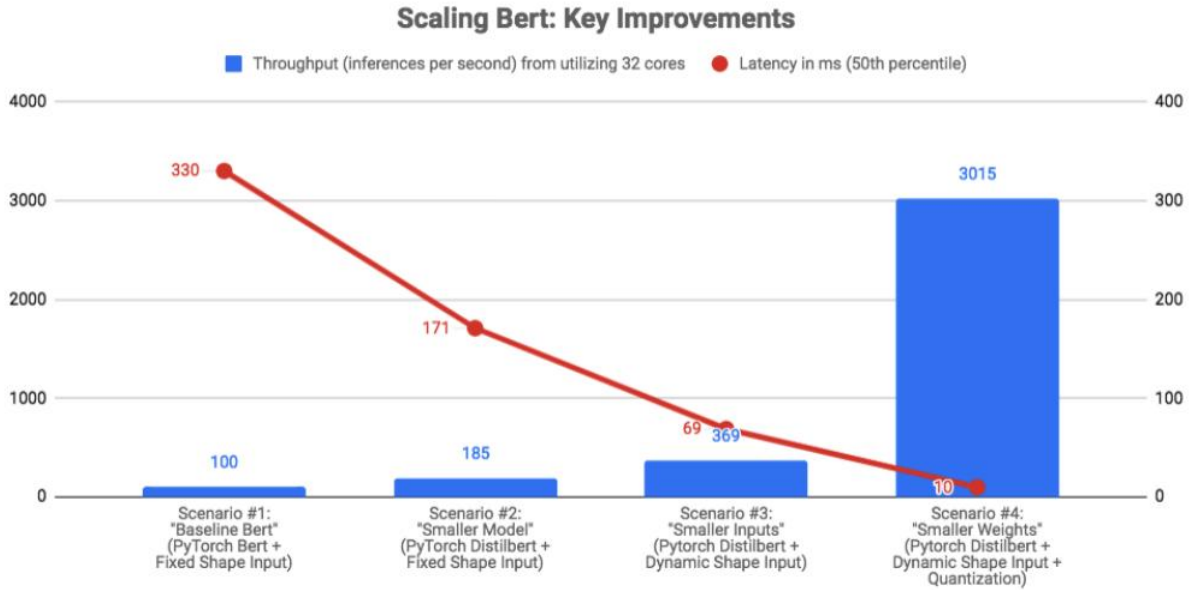
Interesting example: DistilBERT

DistilBERT is a good example of distilled model.



LIÈGE université

Improvement in terms of latency and throughput is significant.

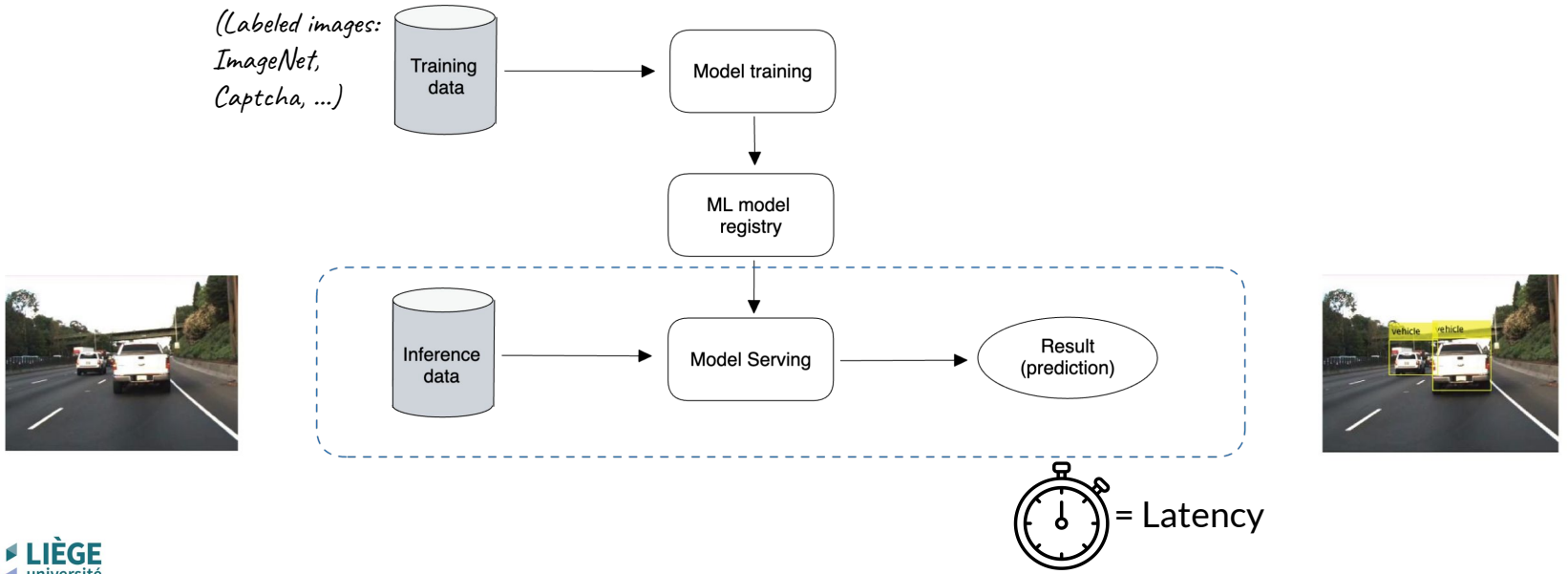


Model serving optimisation

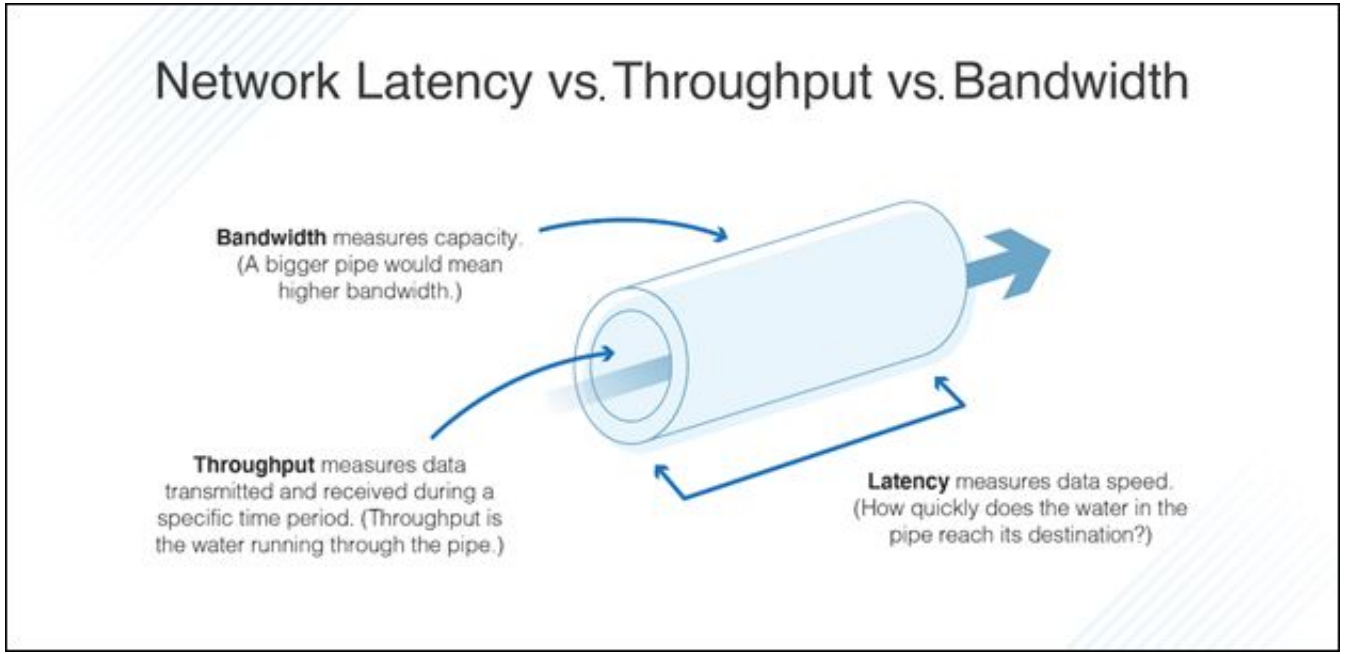
What is model serving and latency?

An ML model is first trained on a (usually labeled) data set. It is then called on **inference** by users during **model serving**.

Latency is the delay between an ML model receiving inference data and producing a specific result.



Different serving metrics



Despite closely related, the performance in each metric is not linear (e.g. can have a low latency but not a great throughput).

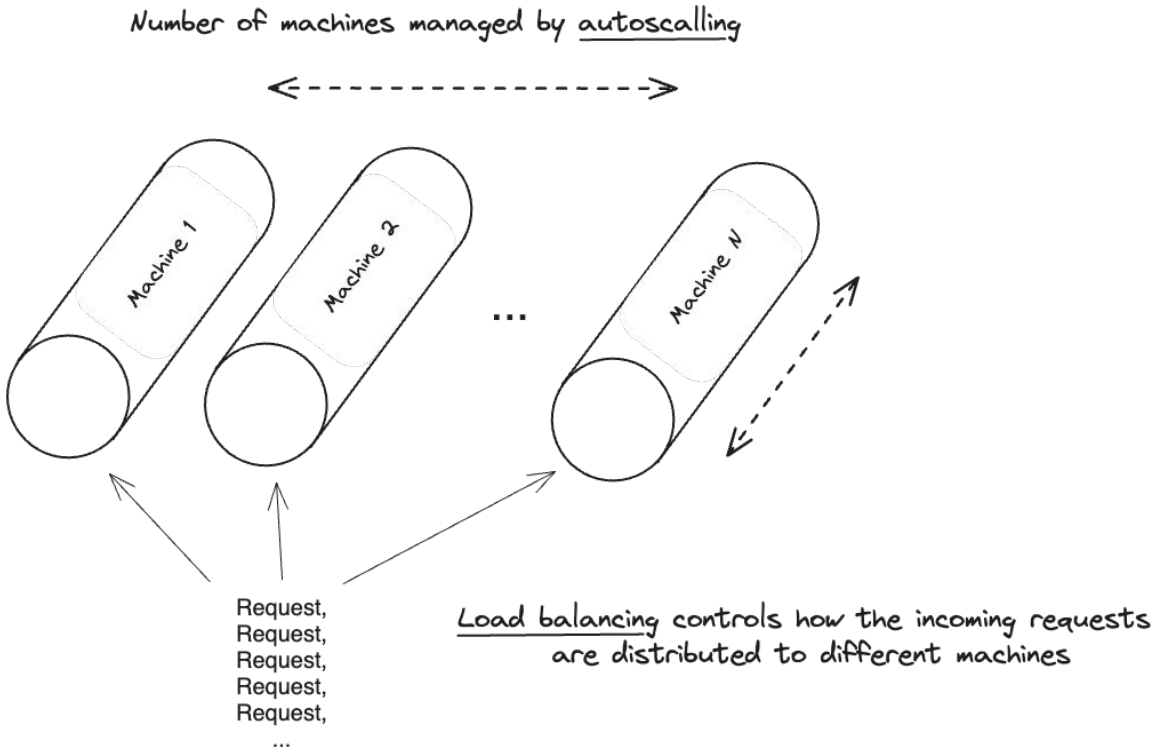
https://shardeum.org/blog/throughput-vs-bandwidth/

Cloud deployment: Autoscalling

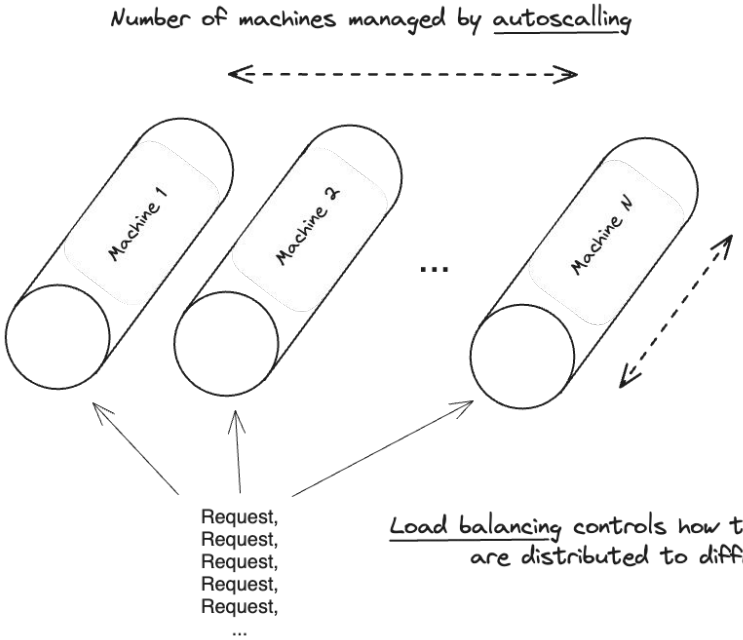
Different Cloud platforms allow for **auto scaling** and **load balancing**.

They will spin more machines to handle incoming requests in parallel.

It is detrimental to handling a large amount of incoming requests.



Autoscaling will improve bandwidth and throughput (latency of a single request will not be impacted).



Helps keeping throughput and bandwidth to a good level

Latency remains the same

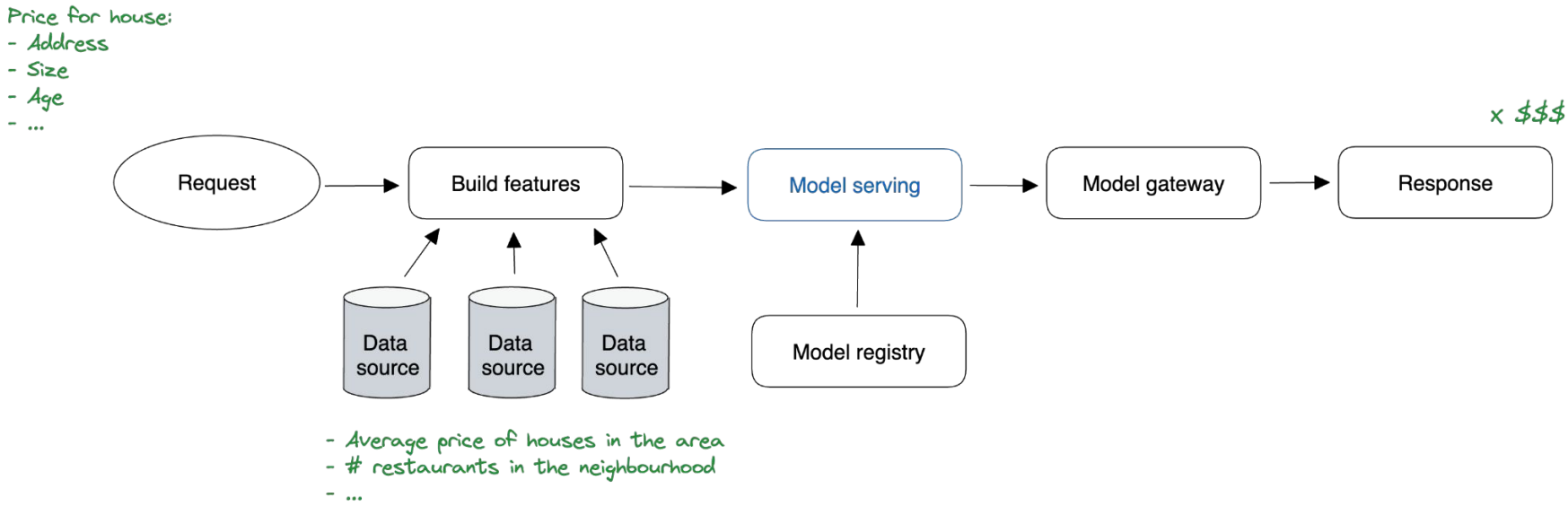
⇒ *Some applications can* ***scale to 0***

Ways of reducing your serving latency without changing your ML model

We look at three different methods:

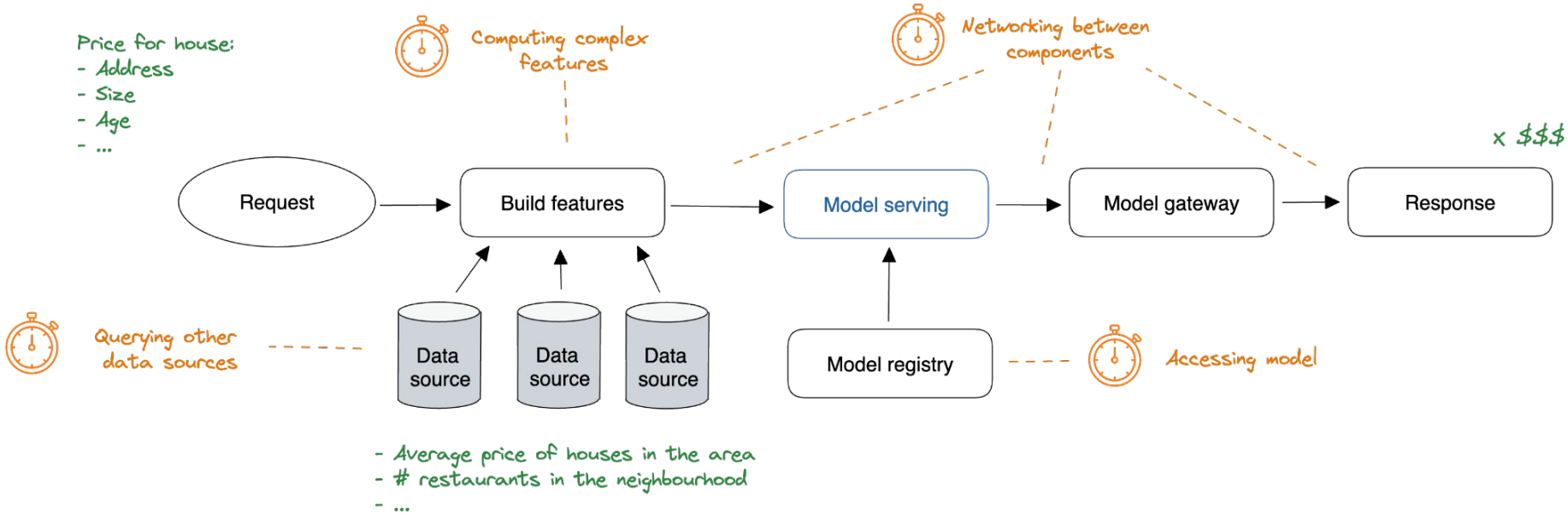
1. Optimise the **pipeline** around your model
2. Optimise the **hardware** used by your model
3. Optimise the **framework** used by your model

Look at different parts of your pipeline.



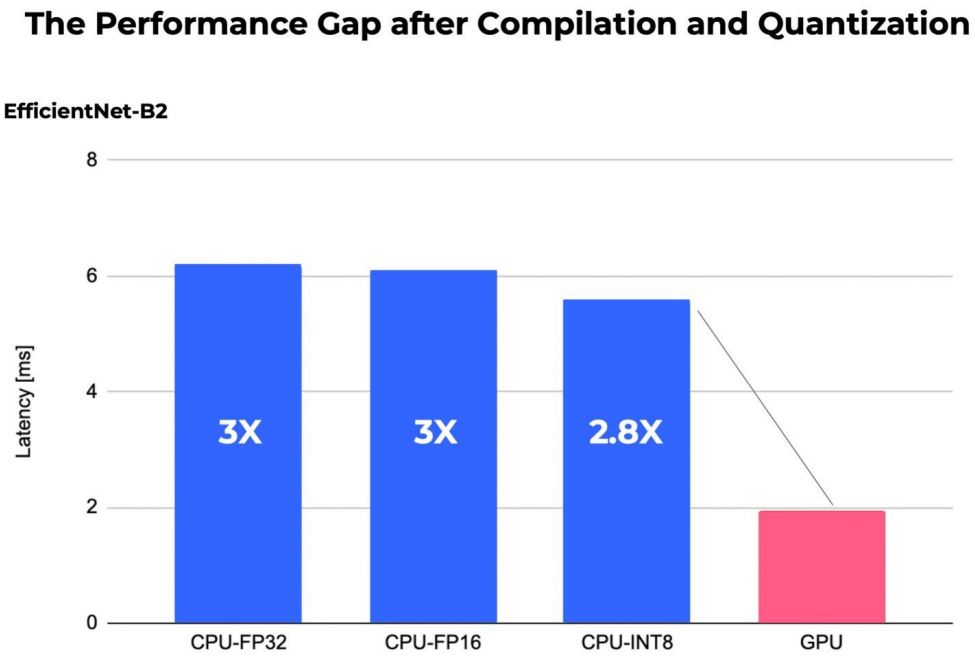
Look at different parts of your pipeline.

You ML model might not be the bottleneck!



Optimise the hardware used by your ML model.

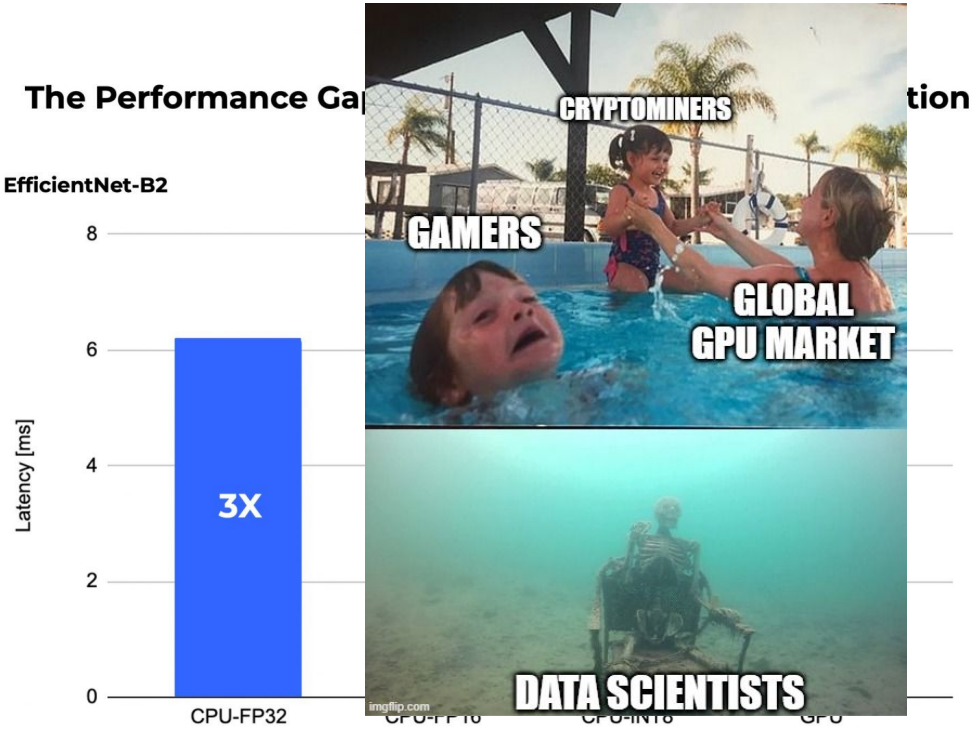
Hardware



Credit: deci.ai

Optimise the hardware used by your ML model.

Hardware



Credit: deci.ai

Tensor Processing Unit (TPU)

Processor	*mm^2*	*Clock MHz*	*TDP Watts*	*Idle Watts*	*Memory GB/sec*	*Peak TOPS/chip*	
						8b int.	*32b FP*
CPU: Haswell (18 core)	662	2300	145	41	51	2.6	1.3
GPU: Nvidia K80 (2 / card)	561	560	150	25	160	--	2.8
TPU	<331*	700	75	28	34	91.8	--

*TPU is less than half die size of the Intel Haswell processor

K80 and TPU in 28 nm process; Haswell fabbed in Intel 22 nm process

These chips and platforms chosen for comparison because widely deployed in Google data centers

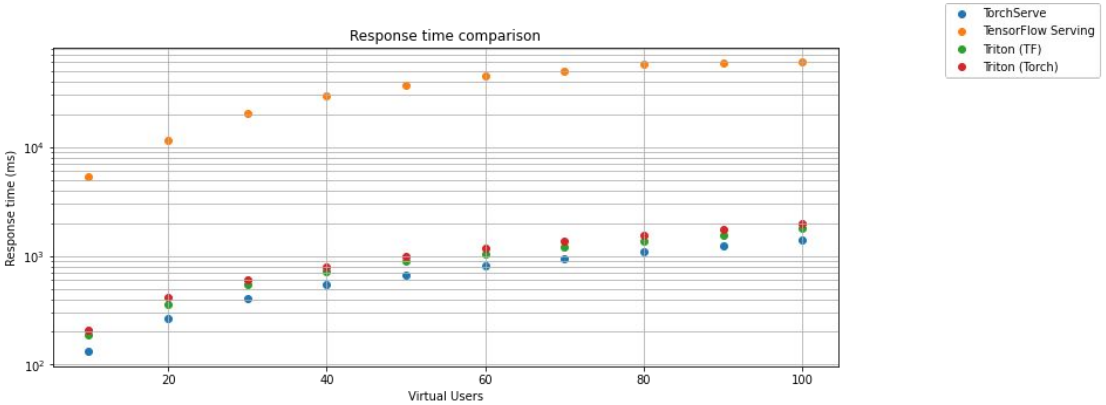
David Patterson and the Google TPU Team, In-Data Center Performance Analysis of a Tensor Processing Unit - ***2017***

You can wrap your model using different model frameworks.

TorchServe: flexible and easy-to-use tool for serving PyTorch models created by Facebook.

vLMM: Enhance the inference and serving efficiency of large language models (LLMs). Using PagedAttention and Continuous Batching, vLLM optimizes memory management and accelerates inference speeds, supporting distributed inference across multiple GPUs.

Triton™ Inference Server: NVIDIA's Triton Inference Server provides a cloud and edge inferencing solution optimized for both CPUs and GPUs.



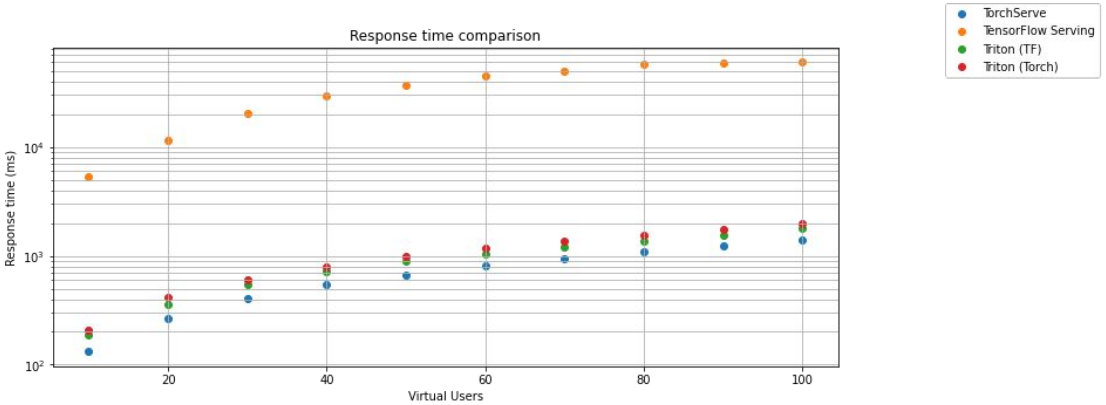
https://biano-ai.github.io/research/2021/08/16/quantitative-comparison-of-serving-platforms-for-neural-networks.html

You can wrap your model using different model frameworks.

~~Triton™ Inference Server~~

NVIDIA Dynamo: Announced on March 18, 2025, at GTC 2025, NVIDIA Dynamo is an open-source, low-latency inference framework designed to serve generative AI models in distributed environments. It builds upon the success of Triton by introducing a modular architecture tailored for large-scale AI deployments.

Dynamo is designed to be inference engine agnostic (supports TRT-LLM, vLLM, SGLang or others).



https://biano-ai.github.io/research/2021/08/16/quantitative-comparison-of-serving-platforms-for-neural-networks.html

NVIDIA Dynamo: Zoom-in

Features.

Disaggregated Serving

Separates LLM context (prefill) and generation (decode) phases across distinct GPUs, enabling tailored model parallelism and independent GPU allocation to increase requests served per GPU.

GPU Planner

Monitors GPU capacity in distributed inference environments and dynamically allocates GPU workers across context and generation phases to resolve bottlenecks and optimize performance.

Smart Router

Routes inference traffic efficiently, minimizing costly recomputation of repeat or overlapping requests to preserve compute resources while ensuring balanced load distribution across large GPU fleets.

NIXL Low-Latency Communication Library

Accelerates data movement in distributed inference settings while simplifying transfer complexities across diverse hardware, including GPUs, CPUs, networks, and storage.

"When serving the open-source DeepSeek-R1 671B reasoning model on NVIDIA GB200 NVL72, NVIDIA Dynamo increased the number of requests served by up to 30x..."

LIÈGE université

https://www.nvidia.com/en-us/ai/dynamo/

NVIDIA Dynamo: Zoom-in

Installation

The following examples require a few system level packages. Recommended to use Ubuntu 24.04 with a x86_64 CPU. See support_matrix.md

```
apt-get update
DEBIAN_FRONTEND=noninteractive apt-get install -yq python3-dev python3-pip python3-venv libucx
python3 -m venv venv
source venv/bin/activate

pip install ai-dynamo[all]
```

Running and Interacting with an LLM Locally

To run a model and interact with it locally you can call `dynamo run` with a hugging face model. `dynamo run` supports several backends including: `mistralrs`, `sglang`, `vllm`, and `tensorrtllm`.

Example Command

```
dynamo run out=vllm deepseek-ai/DeepSeek-R1-Distill-Llama-8B
```

```
? User › Hello, how are you?
✔ User · Hello, how are you?
Okay, so I'm trying to figure out how to respond to the user's greeting. They said, "Hello, ho
```

https://github.com/ai-dynamo/dynamo

LLM Serving

Dynamo provides a simple way to spin up a local set of inference components including:

- **OpenAI Compatible Frontend** – High performance OpenAI compatible http api server written in Rust.
- **Basic and Kv Aware Router** – Route and load balance traffic to a set of workers.
- **Workers** – Set of pre-configured LLM serving engines.

To run a minimal configuration you can use a pre-configured example.

Start Dynamo Distributed Runtime Services

First start the Dynamo Distributed Runtime services:

```
docker compose -f deploy/docker-compose.yml up -d
```

Start Dynamo LLM Serving Components

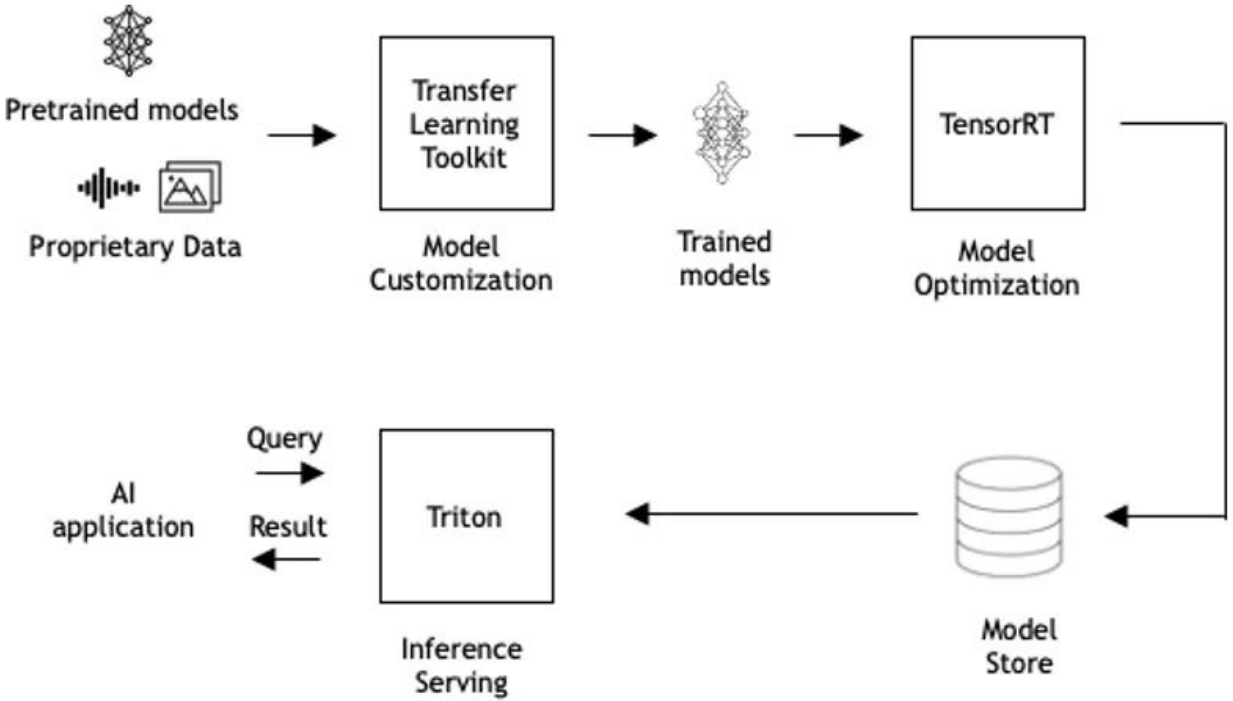
Next serve a minimal configuration with an http server, basic round-robin router, and a single worker.

```
cd examples/llm
dynamo serve graphs.agg:Frontend -f configs/agg.yaml
```

Send a Request

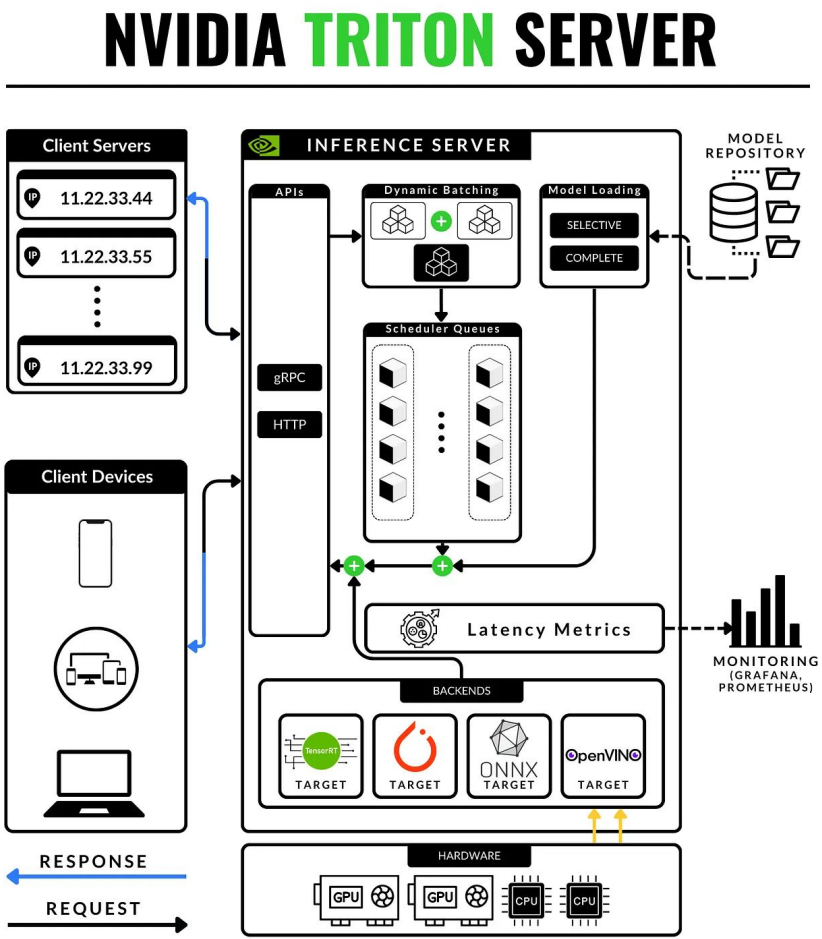
```
curl localhost:8000/v1/chat/completions   -H "Content-Type: application/json"   -d '{
    "model": "deepseek-ai/DeepSeek-R1-Distill-Llama-8B",
    "messages": [
    {
        "role": "user",
        "content": "Hello, how are you?"
    }
    ],
    "stream":false,
    "max_tokens": 300
  }' | jq
```

What is NVIDIA Triton Inference Server (T.I.S) ?



https://medium.com/@tam.tamanna18/deploying-an-onnx-model-with-triton-inference-server-929484be7d2b

What is NVIDIA Triton Inference Server (T.I.S) ?



https://decodingml.substack.com/p/dml-how-to-deploy-deep-learning-models

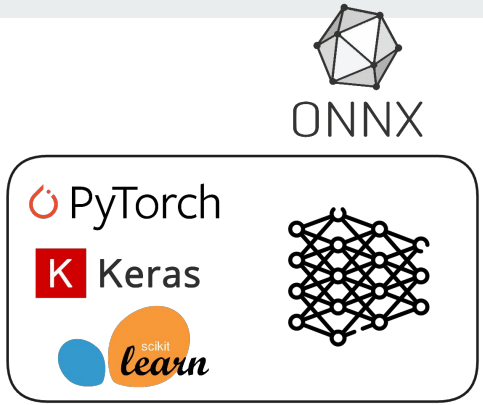
Why use Triton servers ?

- **Optimisation**: Nvidia Triton is optimized for GPU and CPU performance. This includes support for NVIDIA GPUs for faster computation, which can significantly reduce inference times, especially for deep learning models.
- **Scalability**: It is designed to scale across multiple nodes and GPUs, allowing you to serve high volumes of inference requests efficiently.
- **Cloud and Edge Deployment**: It can be deployed in the cloud, on-premises, or at the edge, providing flexibility depending on your infrastructure needs.
- **Ease of use**: Supports multiple frameworks: TensorRT, TensorFlow SavedModel, ONNX, PyTorch TorchScript
- **Monitoring:** Automatically provides metrics on *:8002* port in Prometheus data format which includes GPU utilization, server throughput, latency, and many more.
- **Auto-scaling**: You can control gpu_clusters and model_replicas from within the model configuration files.
- **Online testing**: Deploy multiple version of a model for A/B testing.

https://decodingml.substack.com/p/dml-how-to-deploy-deep-learning-models

What is ONNX ?

Open Neural Network Exchange



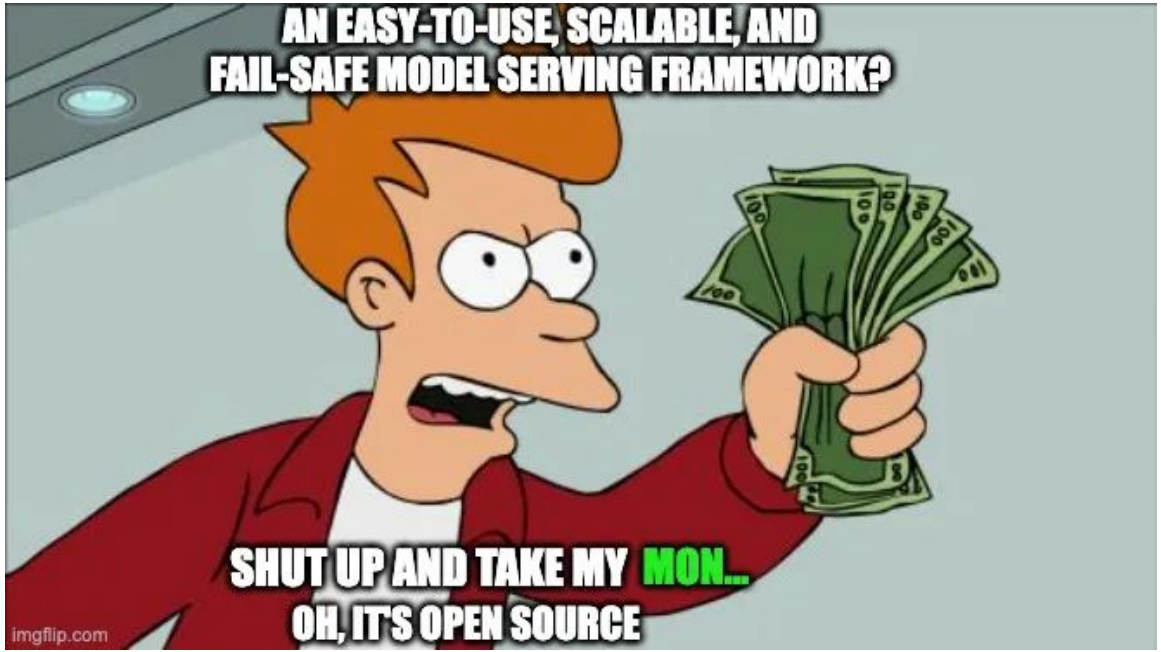
ONNX defines a common **set of operators** - the building blocks of machine learning and deep learning models - and a **common file format** to enable AI developers to use models with a variety of **frameworks**, **tools**, **runtimes**, and **compilers**.

Key benefits:

- **Interoperability**: Develop in your preferred framework without worrying about downstream inferencing implications. ONNX enables you to use your preferred framework with your chosen inference engine.
- **Hardware Access**: ONNX makes it easier to access hardware optimizations. Use ONNX-compatible runtimes and libraries designed to maximize performance across hardware.

https://onnx.ai/

What is NVIDIA Triton Inference Server (T.I.S) ?



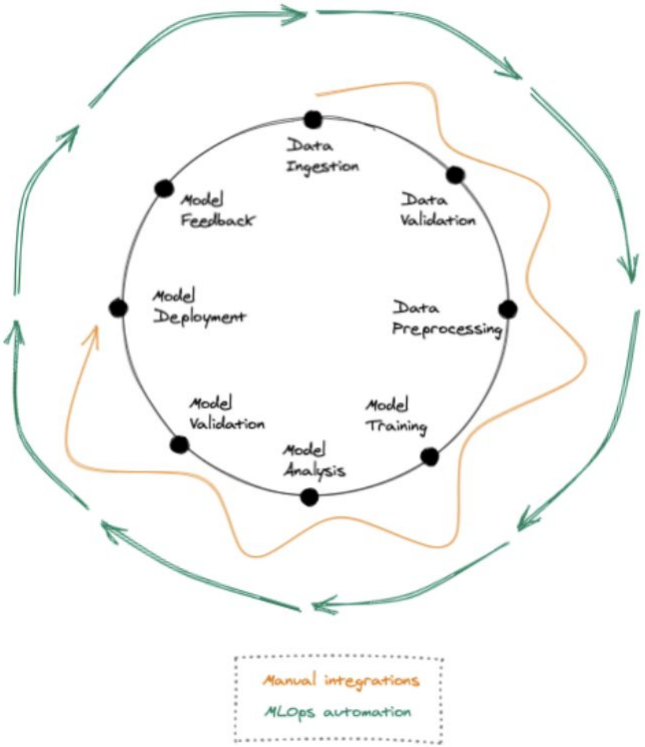
https://decodingml.substack.com/p/dml-how-to-deploy-deep-learning-models

Zoom on how requests are processed by the Triton server

1. **Client** packs the input data (image, text, audio).
2. **Client** specifies which model to use (by name and version)
3. **Client** sends the inference request to the server via either HTTP or gRPC.
4. **Server** receives a request and places it in a queue as Triton is designed to handle multiple requests simultaneously.
5. **Server** retrieves the specified model from the model repository and performs inference.
6. **Server** sends the response back to the client using the same protocol gRPC or HTTP.
7. **Client** receives the response and extracts the result tensor() → numpy().

ML model pipeline

Why do we need ML pipelines?



The general idea is to not treat **ML workflows** as a one-off, but to treat them in a **reliable** and **reproducible** way.

Why do we need ML pipelines?

The Spotify experience

The Winding Road to Better Machine Learning Infrastructure Through Tensorflow Extended and Kubeflow

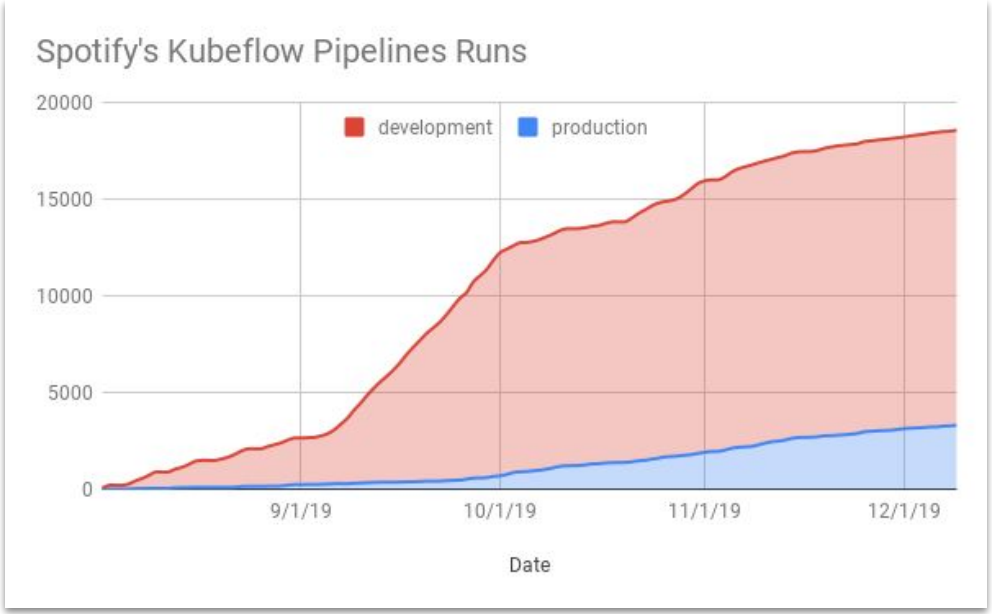
Posted on December 13, 2019 by josh baer and samuelngahane

"As we built these new Machine Learning systems, we started to hit a point where **our engineers spent more of their time maintaining data and backend systems in support of the ML-specific code** than iterating on the model itself. We realized we needed to standardize best practices and build tooling to bridge the gaps between data, backend, and ML: we needed a Machine Learning platform. "

https://labs.spotify.com/2019/12/13/the-winding-road-to-better-machine-learning-infrastructure-through-tensorflow-extended-and-kubeflow/

Why do we need ML pipelines?

Spotify accomplished an astonishing 7x for running machine learning pipelines

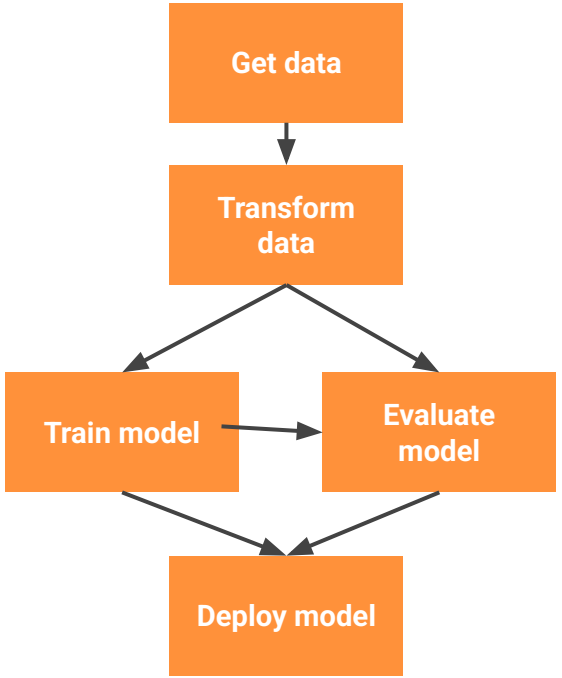


https://labs.spotify.com/2019/12/13/the-winding-road-to-better-machine-learning-infrastructure-through-tensorflow-extended-and-kubeflow/

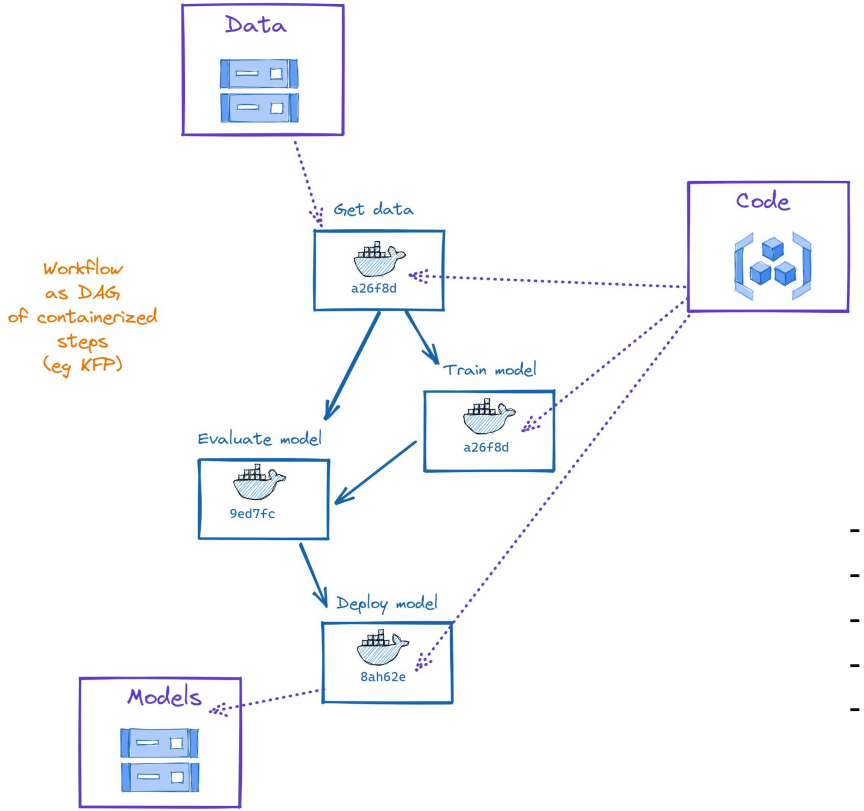
Represent your entire ML workflow as a DAG

Think about your workflow in terms of Directed Acyclic Graph (**DAGs**):

- What needs to be done sequentially vs in parallel
- Does the step process something itself or call an external service?
- Process first, implementation second



Represent your entire ML workflow as a DAG

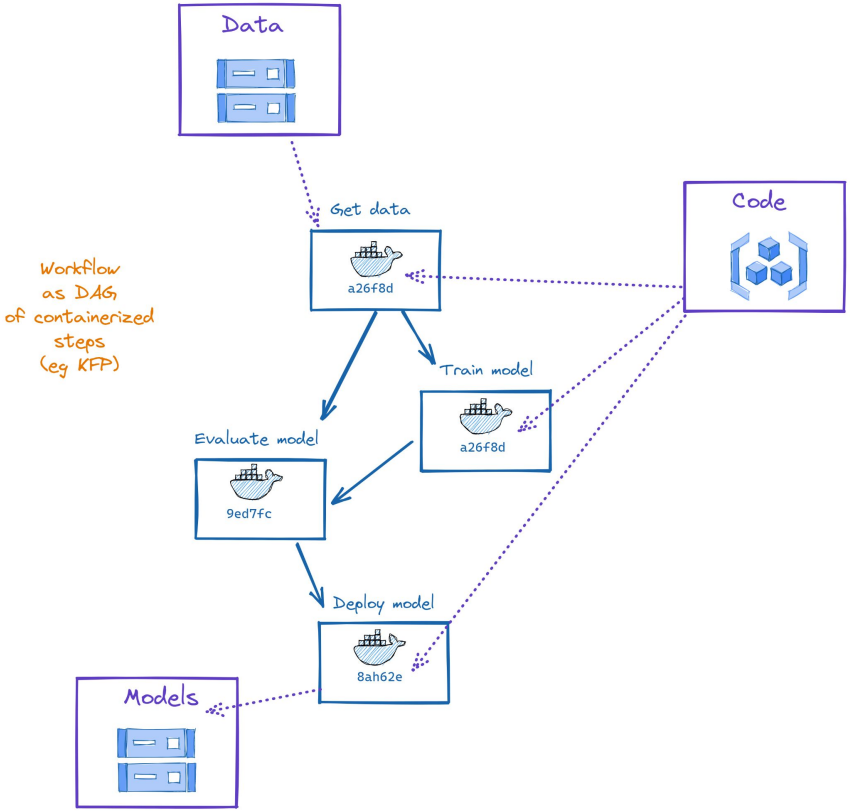


- Kubeflow pipelines
- Sagemaker pipelines
- Vertex pipelines
- Valohai
- MLFlow

LIÈGE université

Definition of a ML Pipeline

“A machine learning pipeline is a series of interconnected data processing and modeling steps designed to automate, standardize and streamline the process of building, training, evaluating and deploying machine learning models.”



https://www.ibm.com/topics/machine-learning-pipeline

What are components of an ML pipeline?

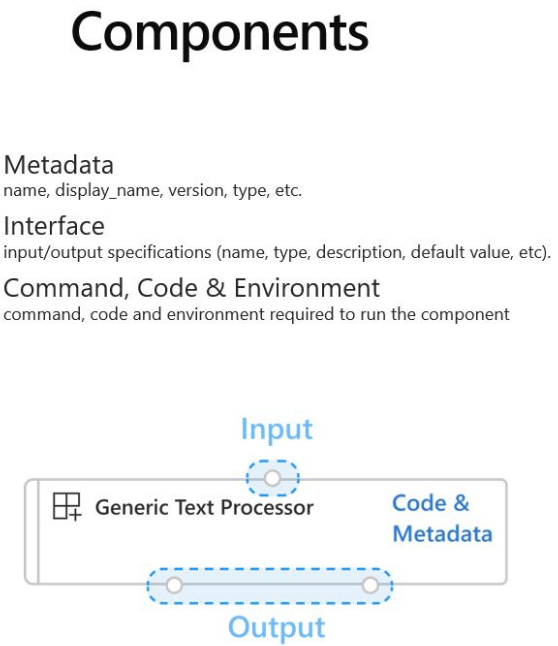
A machine learning pipeline is the workflow for a full **machine learning task**.

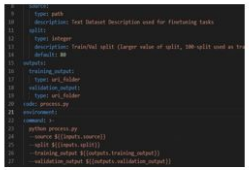
Components are the building blocks of a machine learning pipeline.

They contain:

- Metadata
- Interface (input/output specifications)
- Command, Code & Environment.

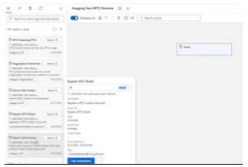
Build as **Docker images** or as **python functions** (with specific decorators and configurations).





CLI

SDK

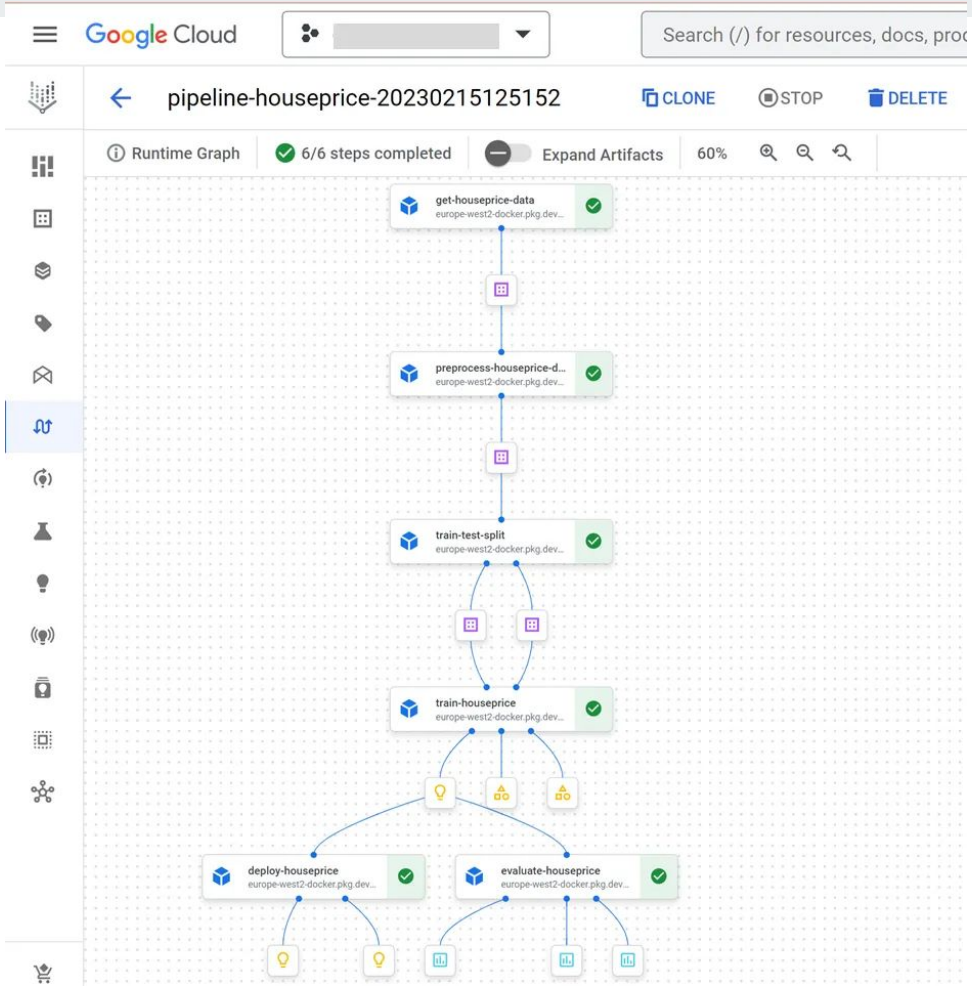


UI

https://learn.microsoft.com/en-us/azure/machine-learning/concept-component?view=azureml-api-2

Example ML Pipeline

Predict house prices with GCP Vertex AI

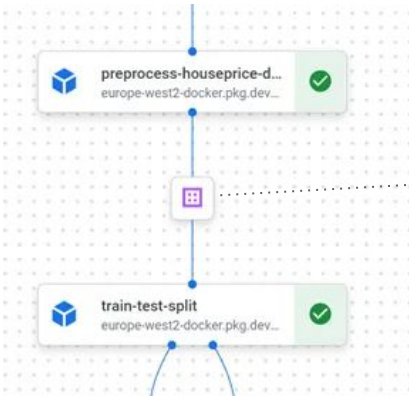


https://medium.com/@wardarahim25/step-by-step-guide-to-creating-and-deploying-custom-ml-pipelines-with-gcp-vertex-ai-part-2-3be6e314bc48

How to pass data between components?

Data staging

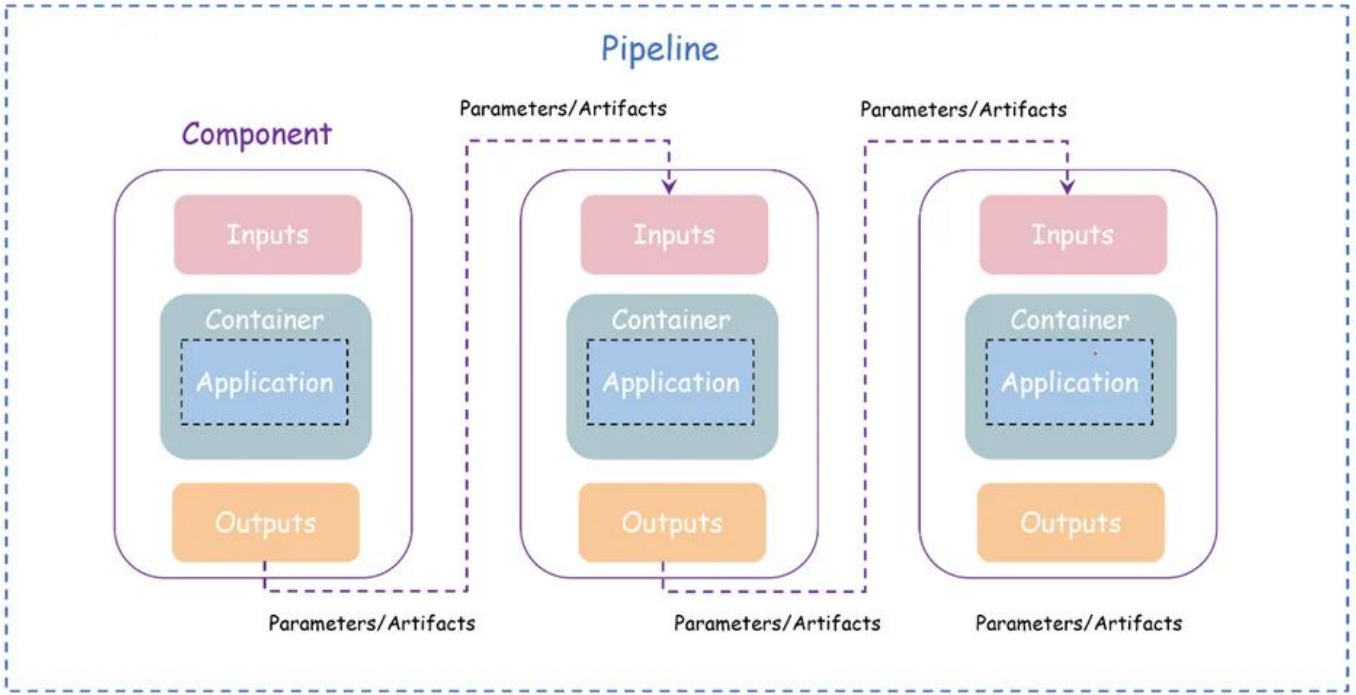
Data needs to be **staged** between components, stored into a Cloud service at the end of a component so it can be taken up by the next component.



Google Cloud Storage

Vertex pipelines integrate with Google Cloud Storage out of the box.

How to pass data between components?



https://medium.com/@wardarahim25/step-by-step-guide-to-creating-and-deploying-custom-ml-pipelines-with-gcp-vertex-ai-part-1-61185bc24f6a

How to pass data between components?

Data staging

```python
from kfp.v2 import dsl, compiler
from kfp.v2.dsl import component, Input, Output, Dataset

@component
def generate_data(output_data: Output[Dataset]):
    import pandas as pd
    df = pd.DataFrame({'numbers': [1, 2, 3, 4, 5]})
    df.to_csv(output_data.path, index=False)

@component
def process_data(input_data: Input[Dataset], output_data: Output[Dataset]):
    import pandas as pd
    df = pd.read_csv(input_data.path)
    df['squared'] = df['numbers'] ** 2
    df.to_csv(output_data.path, index=False)
```

Creates the following GCS objects

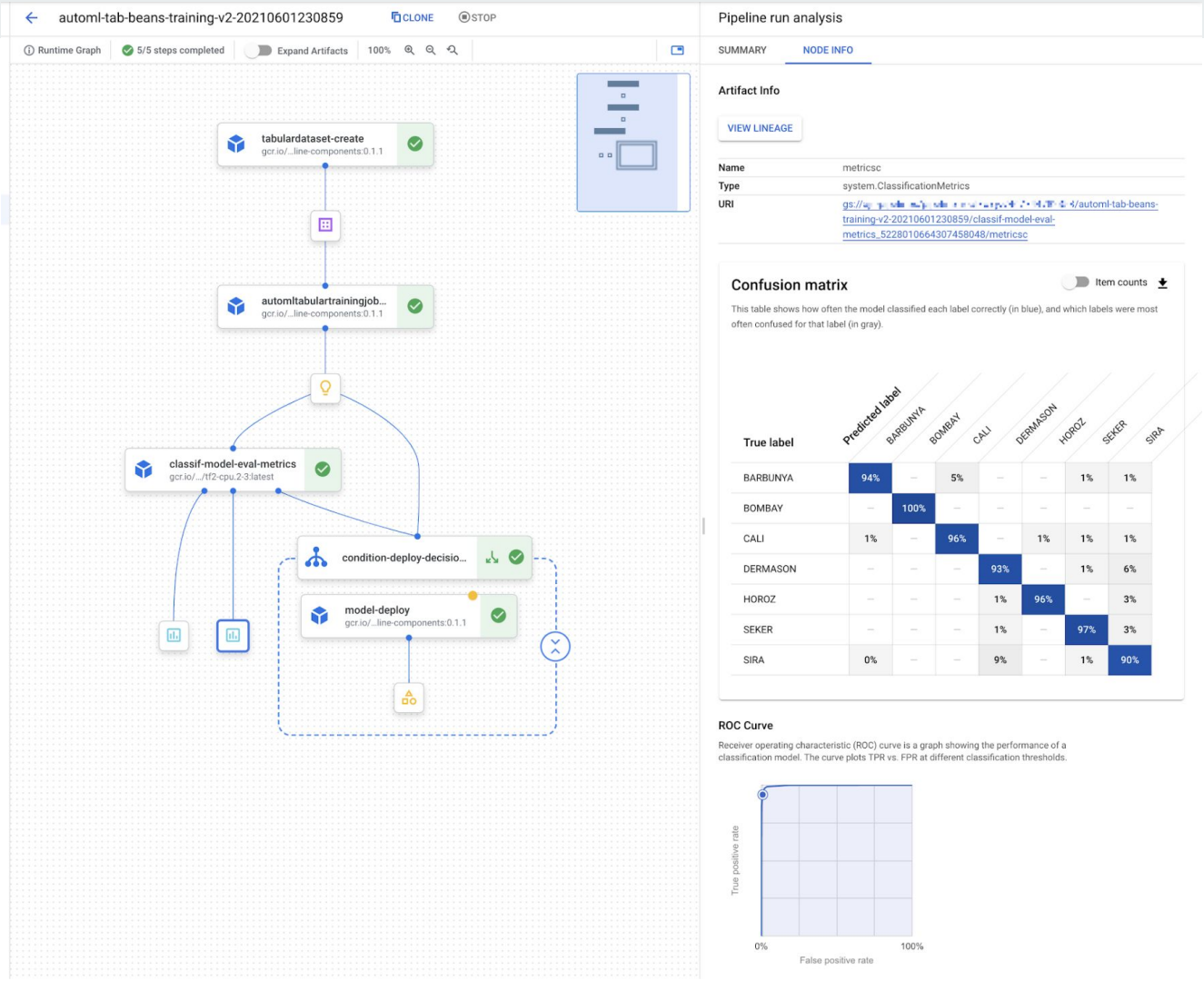
```
gs://your-bucket/pipeline-root/runs/{run-id}/generate_data/output_data/
gs://your-bucket/pipeline-root/runs/{run-id}/process_data/output_data/
```

LIÈGE université

Same example but using BigQuery for staging

```python
from kfp.v2 import dsl, compiler
from kfp.v2.dsl import component, Input, Output, Dataset, Model

BQ_DATASET = "your_bq_dataset"
BQ_SOURCE_TABLE = f"{PROJECT_ID}.{BQ_DATASET}.source_table"
BQ_TARGET_TABLE = f"{PROJECT_ID}.{BQ_DATASET}.processed_table"

@component
def create_bigquery_table():
    """Creates a BigQuery table and inserts sample data."""
    from google.cloud import bigquery

    client = bigquery.Client()
    schema = [
        bigquery.SchemaField("id", "INTEGER"),
        bigquery.SchemaField("value", "FLOAT"),
    ]

    table = bigquery.Table(BQ_SOURCE_TABLE, schema=schema)
    client.create_table(table, exists_ok=True)

    rows = [{"id": i, "value": float(i)} for i in range(1, 6)]

@component
def process_bigquery_data(bq_source: str, bq_target: str):
    """Reads from BigQuery, processes the data, and writes back to BigQuery."""
    from google.cloud import bigquery

    client = bigquery.Client()
    query = f"""
    SELECT id, value, value * value AS squared_value
    FROM `{bq_source}`
    """
    df = client.query(query).to_dataframe()

    job_config = bigquery.LoadJobConfig(write_disposition="WRITE_TRUNCATE")
    client.load_table_from_dataframe(df, bq_target, job_config=job_config)
```

Vertex lets you easily visualise artifacts - such as evaluation metrics



https://cloud.google.com/blog/topics/developers-practitioners/use-vertex-pipelines-build-automl-classification-end-end-workflow

LIÈGE université

Example ML Pipeline

Predict house prices with GCP Vertex AI

Based on Kubeflow Pipelines (KFP)

```
# IMPORT REQUIRED LIBRARIES
from kfp.v2 import dsl
from kfp.v2.dsl import (Artifact,
                        Dataset,
                        Input,
                        Model,
                        Output,
                        Metrics,
                        Markdown,
                        HTML,
                        component,
                        OutputPath,
                        InputPath)
from kfp.v2 import compiler
from google.cloud.aiplatform import pipeline_jobs
```

https://medium.com/@wardarahim25/step-by-step-guide-to-creating-and-deploying-custom-ml-pipelines-with-gcp-vertex-ai-part-2-3be6e314bc48

Example ML Pipeline

Predict house prices with GCP Vertex AI

Define **components**.

Can be separate **Docker containers** or **python functions**.

Can create a pipeline in one single notebook

```python
@component(
    base_image=BASE_IMAGE,
    output_component_file="get_data.yaml"
)

def get_houseprice_data(
    filepath: str,
    dataset_train: Output[Dataset],
):

    import pandas as pd

    df_train = pd.read_csv(filepath + '/train.csv')

    df_train.to_csv(dataset_train.path, index=False)
```

Data Ingestion component

https://medium.com/@wardarahim25/step-by-step-guide-to-creating-and-deploying-custom-ml-pipelines-with-gcp-vertex-ai-part-2-3be6e314bc48

Example ML Pipeline

Predict house prices with GCP Vertex AI

```python
@component(
    base_image=BASE_IMAGE,
    output_component_file="preprocessing.yaml"
)

def preprocess_houseprice_data(
    train_df: Input[Dataset],
    dataset_train_preprocessed: Output[Dataset],
):

    import pandas as pd
    from src.data_preprocessing.preprocessing import data_preprocessing_pipeline

    train_df = pd.read_csv(train_df.path)

    # data_preprocessing_pipeline creates a copy of the df, removes id col, converts to
    # subtracts YearSold from temporal features and cosine transforms cyclic features.
    train_df_preprocessed = data_preprocessing_pipeline(train_df)

    train_df_preprocessed.to_csv(dataset_train_preprocessed.path, index=False)
```

Data Preprocessing component

```python
@component(
    base_image=BASE_IMAGE,
    output_component_file="train_test_split.yaml",
)

def train_test_split(dataset_in: Input[Dataset],
                     dataset_train: Output[Dataset],
                     dataset_test: Output[Dataset],
                     test_size: float = 0.2):

    import pandas as pd
    from sklearn.model_selection import train_test_split

    df = pd.read_csv(dataset_in.path)
    df_train, df_test = train_test_split(df, test_size=test_size, random_state=42)

    df_train.to_csv(dataset_train.path, index=False)
    df_test.to_csv(dataset_test.path, index=False)
```

Train test split component

https://medium.com/@wardarahim25/step-by-step-guide-to-creating-and-deploying-custom-ml-pipelines-with-gcp-vertex-ai-part-2-3be6e314bc48

Example ML Pipeline

Predict house prices with GCP Vertex AI

...

```
14      import pandas as pd
15      import pickle
16      import shap
17      from src.modelling.train import HousePriceModel
18      from src.utils.utils import get_image_data
19
20      TARGET = 'SalePrice'
21
22      # Read train and test data
23      train_data = pd.read_csv(dataset_train.path)
24      test_data = pd.read_csv(dataset_test.path)
25
26      # Instantiate the model class
27      house_price_model = HousePriceModel(test_data.copy(),   #we perform hyperparameter t
28                                          target=TARGET,
29                                          n_kfold_splits=3,
30                                          n_trials=100,
31                                          random_state=42)
32
33      # Create X_train and y_train
34      X_train = train_data.drop(TARGET, axis=1)
35      y_train = train_data[TARGET]
```

...

Model training component

```
1   @component(
2       base_image=BASE_IMAGE,
3       output_component_file="model_evaluation.yaml"
4   )
5   def evaluate_houseprice(
6       houseprice_model: Input[Model],
7       metrics_baseline: Output[Metrics],
8       metrics_train: Output[Metrics],
9       metrics_test: Output[Metrics]):
10
11      import pickle
12
13      file_name = houseprice_model.path
14      with open(file_name, 'rb') as file:
15          model_data = pickle.load(file)
16
17      scores = model_data["scores_dict"]
18
19      def log_metrics(scores, metric):
20          for metric_name, val in scores.items():
21              metric.log_metric(metric_name, float(val))
22
23      log_metrics(scores["baseline_scores"], metrics_baseline)
24      log_metrics(scores["train_scores"], metrics_train)
25      log_metrics(scores["test_scores"], metrics_test)
```

Model evaluation component

LIÈGE université

https://medium.com/@wardarahim25/step-by-step-guide-to-creating-and-deploying-custom-ml-pipelines-with-gcp-vertex-ai-part-2-3be6e314bc48

Example ML Pipeline

Predict house prices with GCP Vertex AI

```python
from google.cloud import aiplatform as vertex_ai
from pathlib import Path

# Checks existing Vertex AI Enpoint or creates Endpoint if it is not exist.
def create_endpoint ():
    endpoints = vertex_ai.Endpoint.list(
    filter='display_name="{}"'.format(model_endpoint),
    order_by='create_time desc',
    project=gcp_project,
    location=gcp_region,
    )
    if len(endpoints) > 0:
        endpoint = endpoints[0] # most recently created
    else:
        endpoint = vertex_ai.Endpoint.create(
            display_name=model_endpoint,
            project=gcp_project,
            location=gcp_region
    )
    return endpoint
```

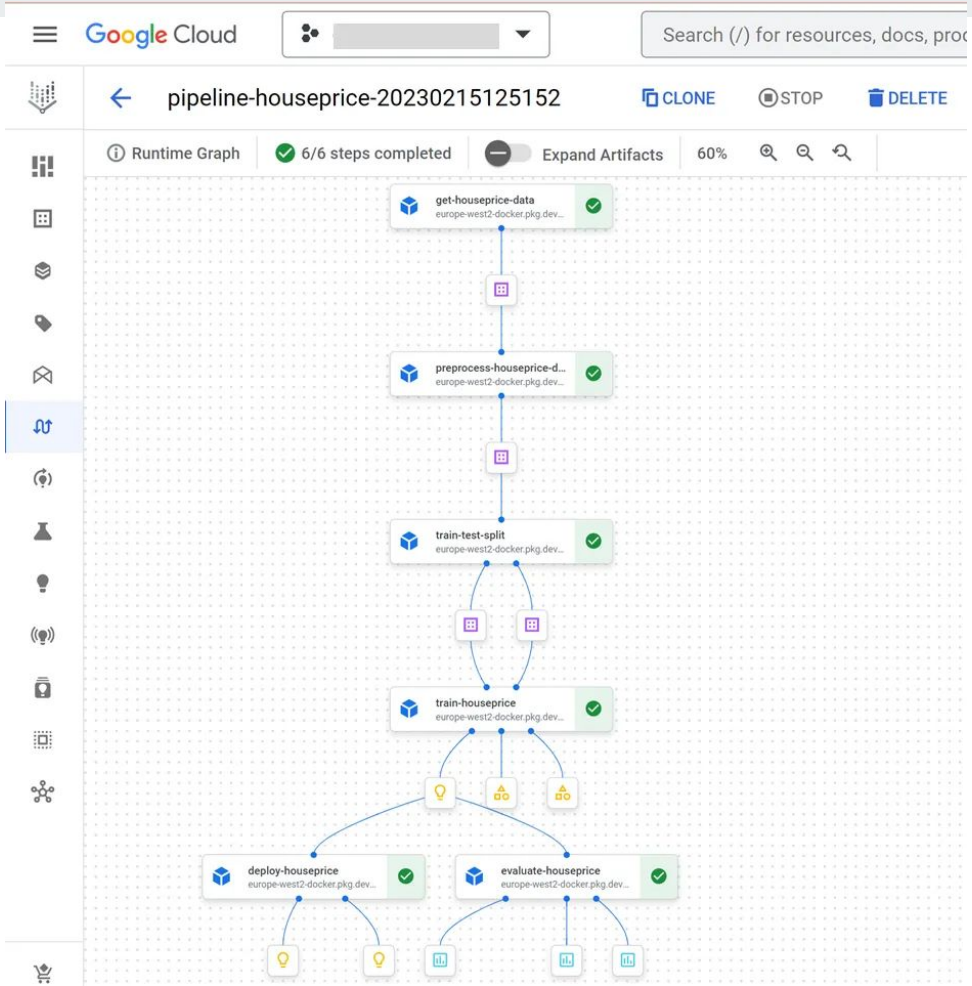
```python
# Uploads trained model to Vertex AI Model Registry or creates new model version in
def upload_model ():
    listed_model = vertex_ai.Model.list(
    filter='display_name="{}"'.format(display_name),
    project=gcp_project,
    location=gcp_region,
    )
    if len(listed_model) > 0:
        model_version = listed_model[0] # most recently created
        model_upload = vertex_ai.Model.upload(
                display_name=display_name,
                parent_model=model_version.resource_name,
                artifact_uri=str(Path(model.path).parent),
                serving_container_image_uri=serving_container_image_uri,
                location=gcp_region,
                serving_container_predict_route="/predict",
                serving_container_health_route="/health"
        )
```

Model deployment component

https://medium.com/@wardarahim25/step-by-step-guide-to-creating-and-deploying-custom-ml-pipelines-with-gcp-vertex-ai-part-2-3be6e314bc48

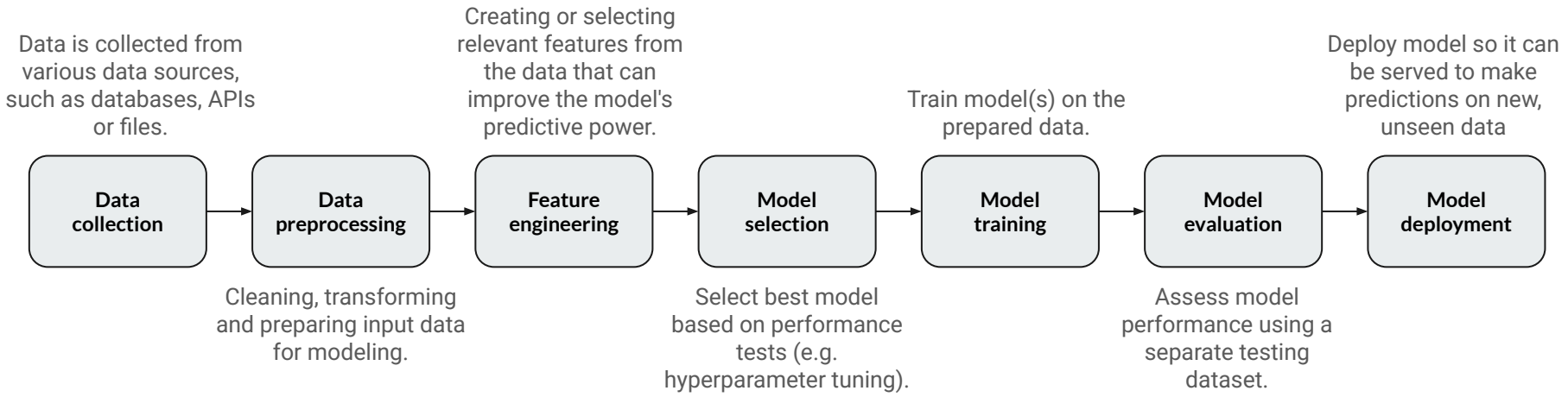
Example ML Pipeline

Predict house prices with GCP Vertex AI



https://medium.com/@wardarahim25/step-by-step-guide-to-creating-and-deploying-custom-ml-pipelines-with-gcp-vertex-ai-part-2-3be6e314bc48

Typical components of an ML pipeline



https://www.ibm.com/topics/machine-learning-pipeline

Avoiding the endless POC loop…

- Let’s change the model architecture
- Let’s try with another model
- Let’s gather more labeled data
- Ah wait, it was the wrong data, here is the new data, can we get results tomorrow?
- What were the accuracies again with the other model?
- Oops not sure what notebook was executed to get that model
- Could we try out the model to get feedback?
- …

*The key to a **successful POC outcome** is to **plan for production** during the proof of concept.*

Benefits of an ML Pipeline

- Modularization: Breaks ML workflows into reusable components, making development faster and easier to manage.
- Reproducibility: Ensures experiments and results can be consistently recreated by tracking configurations and data.
- Efficiency: Automates repetitive tasks like data preprocessing and model training, saving time and effort.
- Scalability: Seamlessly handles large datasets and complex models by leveraging cloud infrastructure.
- Experimentation: Enables quick testing of different models and hyperparameters without manual reconfiguration.
- Deployment: Streamlines moving models from development to production with automated workflows.
- Collaboration: Facilitates teamwork by structuring workflows and sharing components across teams.
- Version control and documentation: Tracks changes in datasets, models, and configurations, ensuring transparency and traceability.

https://www.ibm.com/topics/machine-learning-pipeline

Triggers

Timed

- **Schedule**: Pipeline runs repetitively in relation to the creation time of the Recurring Run. Set the unit (minutes, hours, etc.) and the scalar that goes with it

Event based

- **Manual**: Launch the pipeline manually
- **Triggered**: As part of another service (e.g. if users upload a new batch of training in a specific GCS bucket)

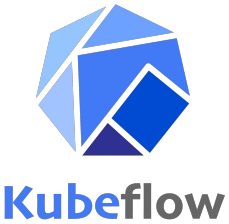
```
# ┌──────────── minute (0–59)
# │ ┌────────── hour (0–23)
# │ │ ┌──────── day of the month (1–31)
# │ │ │ ┌────── month (1–12)
# │ │ │ │ ┌──── day of the week (0–6) (Sunday to Saturday;
# │ │ │ │ │                                   7 is also Sunday on some systems)
# │ │ │ │ │
# │ │ │ │ │
# * * * * * <command to execute>
```

https://en.wikipedia.org/wiki/Cron

Microservices vs ML Pipeline

	Microservices	**ML Pipeline**
What is it?	One full application made of different services that are calling each other to serve the overall purpose of the application.	Ran a single workflow made of components that run sequentially.
Is made of…	Services (~= APIs)	Components. Single process that is ran once (e.g. data preparation, model training, evaluation and deployment).
Utilisation	The microservices stay up.	One time run (triggered/scheduled).
Purpose	All over software engineering.	ML.

ML (pipeline) platforms



ML pipeline platforms

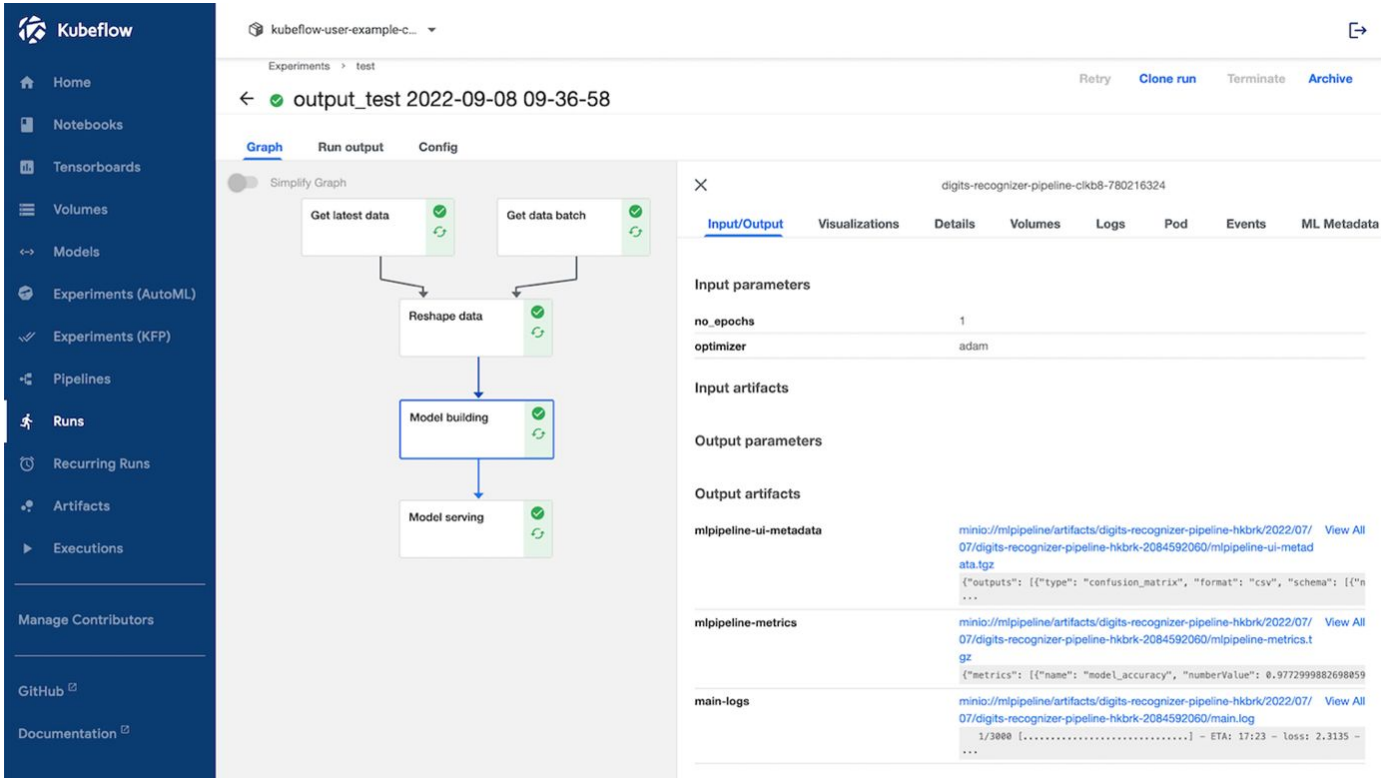
Amazo
SageM

ertex AI

Azure Machine

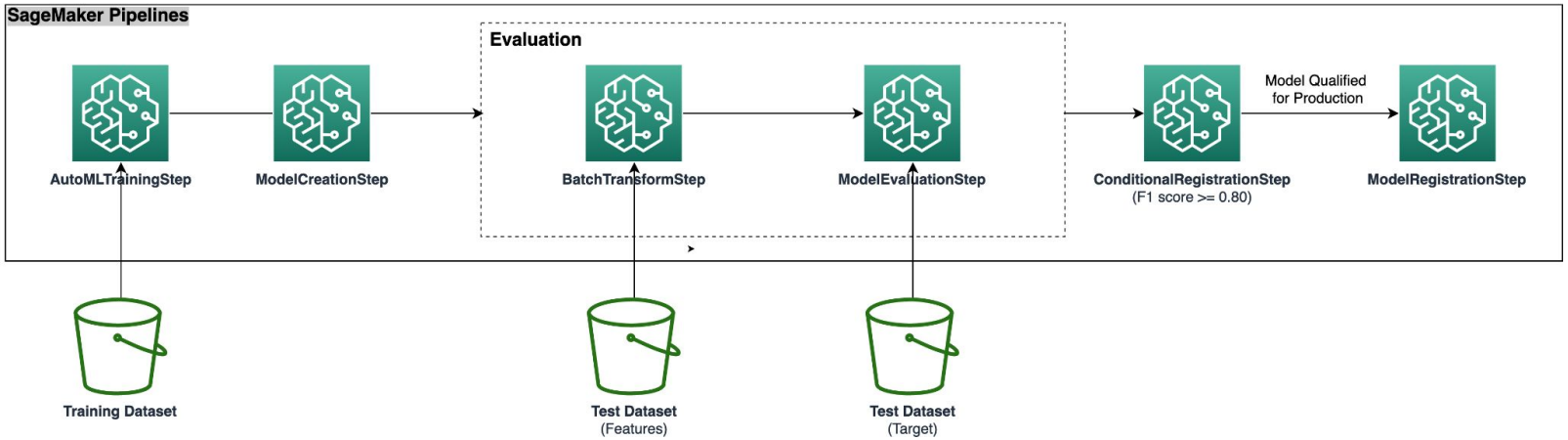
eflow

Example ML Pipeline (Kubeflow pipeline KFP)



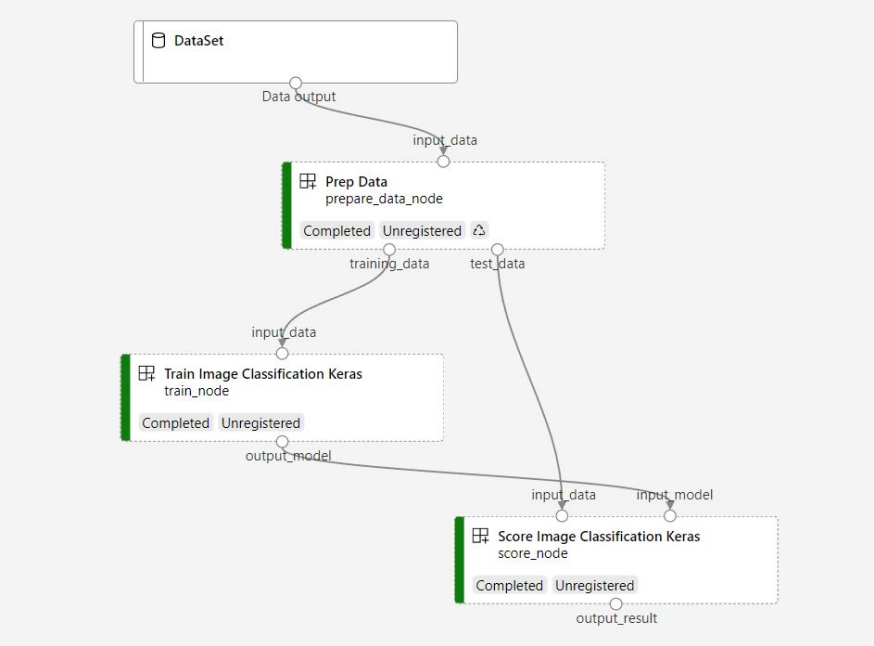
https://learn.microsoft.com/en-us/azure/machine-learning/how-to-create-component-pipeline-python?view=azureml-api-2

Example ML Pipeline (Sagemaker)



https://github.com/aws/amazon-sagemaker-examples/blob/main/sagemaker-pipelines/tabular/automl-step/sagemaker_autopilot_pipelines_native_auto_ml_step.ipynb

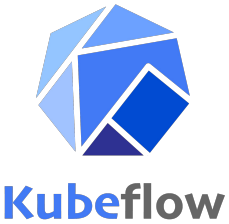
Example ML Pipeline (Azure ML)



https://learn.microsoft.com/en-us/azure/machine-learning/how-to-create-component-pipeline-python?view=azureml-api-2

ML Platforms

ML (pipeline) platforms



Sagemaker

Amazon SageMaker is an automated platform and comprehensive suite of tools that simplifies the development, training and deployment of machine learning (ML) models. It reduces the complexity of model development by providing a web-based interface for creating ML pipelines and pre-built algorithms.

Key features:

1. Notebooks
2. Training Job
3. Model registry
4. Prediction
5. Pipelines

https://www.missioncloud.com/blog/what-is-sagemaker-and-how-does-it-help-mlops

Sagemaker: Notebooks

Machine learning (ML) compute instance running the Jupyterlab. Use Jupyter notebooks in your notebook instance to prepare and process data, write code to train models, deploy models to SageMaker hosting, and test or validate your models.

Advantages:

- No use of data locally
- Select desired (GPUs)
- Better memory
- Manage access
- Collaborate as a team

Equivalent

Vertex Workbench

Azure Notebooks

https://docs.aws.amazon.com/sagemaker/latest/dg/gs-setup-working-env.html

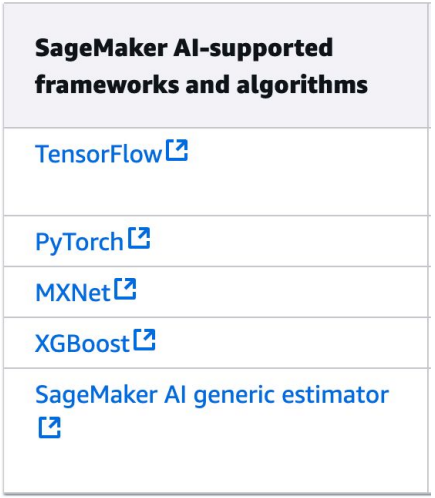
Sagemaker: Training Job

Managed way of launching a model training

1. Either be fully specify the ML algorithm, hyperparameters and input data location.
2. Or customize training job by selecting specific Sagemaker instance types and adding software libraries.

Advantages:

- Access more compute (GPUs)
- Access more memory
- Consistency
- Automation



Equivalent

Vertex Training

Azure ML Training

https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-training.html
https://docs.aws.amazon.com/sagemaker/latest/dg/debugger-supported-frameworks.html

Sagemaker: Model registry

SageMaker Model Registry is a service for packaging model artifacts with deployment information. That information includes your deployment code, what type of container to use and what type of instance to deploy. The Model Registry decreases time to deployment, as you already have the necessary information about how the model should be deployed.

Advantages:

- Consistency
- Automation
- Scalability
- Time to deployment

Equivalent

Vertex AI Model Registry

Azure ML Model Registry

https://www.missioncloud.com/blog/what-is-sagemaker-and-how-does-it-help-mlops

Sagemaker: Prediction

After you've trained and deployed your model in SageMaker, you can use it to generate predictions based on new data. There are two main ways to do this.

- Endpoint deployment: Deploy your model to an endpoint, allowing users and applications to send API requests to get model predictions in real time.
- Batch predictions: Generate predictions on large amounts of data without needing an immediate response..

Advantages:

- Consistency
- Automation
- Scalability
- Time to deployment

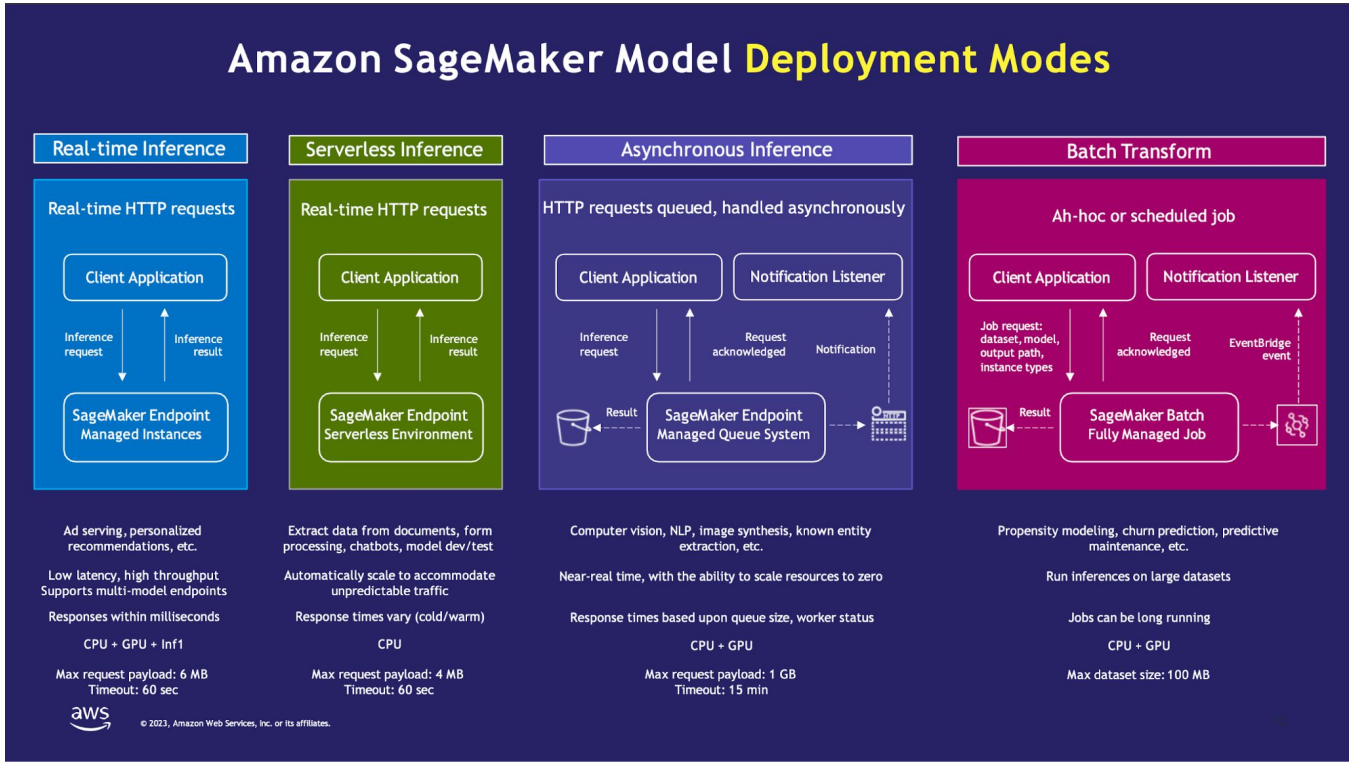
Equivalent

Vertex AI Prediction

Azure ML Prediction

https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-training.html

Sagemaker: Prediction



https://docs.aws.amazon.com/sagemaker/latest/dg/deploy-model-get-started.html

Sagemaker: Pipeline

SageMaker Pipelines is a tool for building, deploying and managing end-to-end ML workflows. Users can create automated workflows that cover data preparation, model training and deployment — all from a single interface. Because Pipeline logs everything, you can easily track and re-create models.

Advantages:

- Resource consumption
- Consistency
- Automation
- Scalability
- Time to deployment

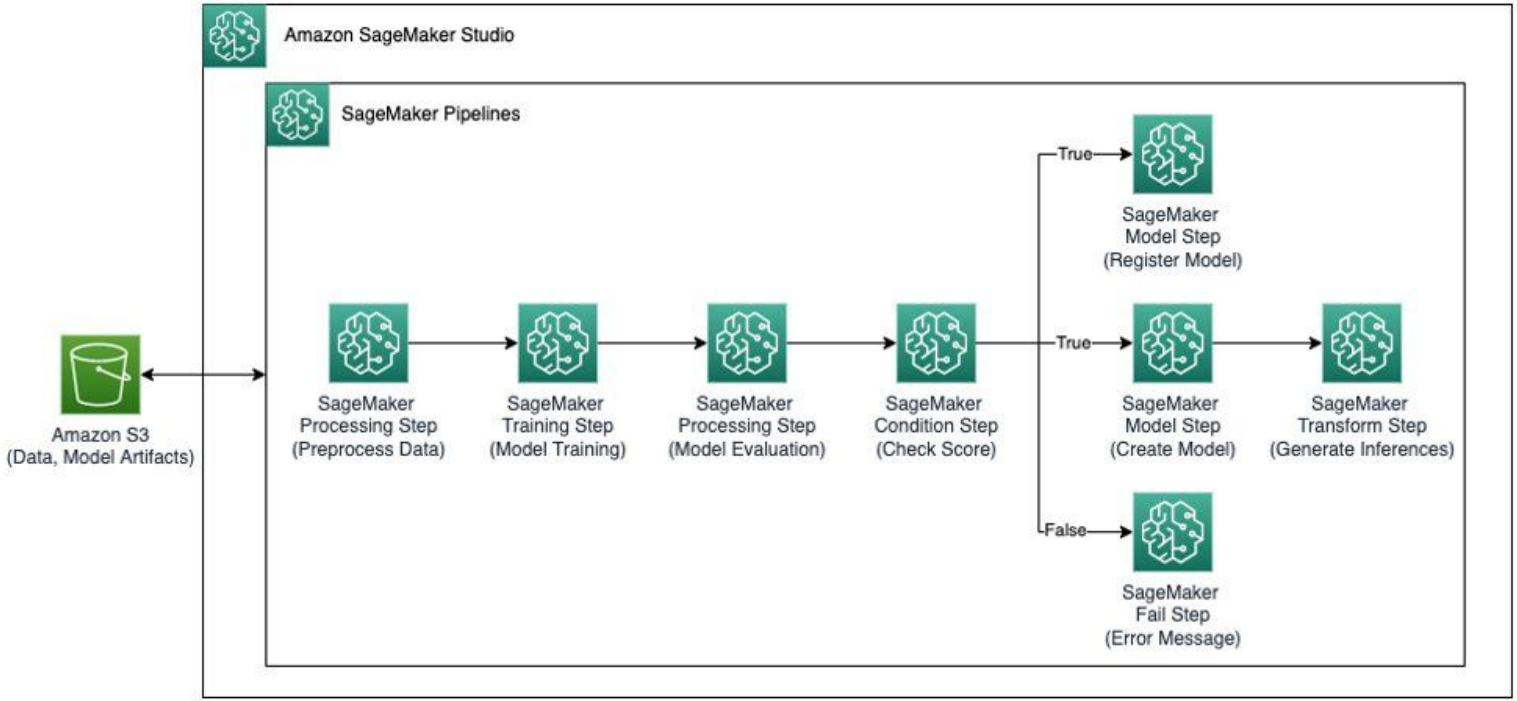
Equivalent

Vertex AI Pipeline

Azure ML Pipeline

https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-training.html

Sagemaker: Pipeline (example)



https://mkai.org/new-features-for-amazon-sagemaker-pipelines-and-the-amazon-sagemaker-sdk/

~~Lab: Vertex Pipeline~~

Wrap-up

Project milestones

MS 2

*Model **pipeline** is delayed 1 week. It's ok if you show it in the next milestone*

MS	Topics	Sprints
1	Present your general *use case*, the *data preparation* and the result from your model *experimentation*.	Sprint 1 & 2
2	Present your architecture for *model serving, model deployment* and (if possible) *model pipeline*.	Sprint 3 (& 4)
3	Present your overall project work. You can present a demo of your model/use case and any other topic you think is relevant.	All sprints

Project milestones

Practicals

- The milestone presentation will take 15min (10min presentation + 5min QA)
- MS 2 will be the week of the 31st of March. You have the choice to do it
  - In presence: Monday 31st of march afternoon
  - Online: Google Meets, other afternoons of the same week
- We will send a mail with a link to book a Google Meets slot

MS presentations

Rules of the game

- Share your PR (+ email) *before your* presentation
  - Document in the README what is important for the teaching staff to check!
- Make sure to be ready at the moment it starts
  - All team available
  - Slides ready
  - Mic check
  - Screen sharing check
- Spread the work 👬
  - Split the speaking time

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

Lecture summary

Topic	Concepts	To know for…	
		Project	Exam
Model serving optimisation	• Optimise different parts of model serving for latency optimisation		Yes
Parallel and Distributed Training	• Data parallelisation • Model parallelisation • Pipeline parallelisation		Yes
Model complexity optimisation	• What is model simplification and why it matters • Pruning, quantisation and distillation	Yes	

Lecture summary

(split over next week)

Topic	Concepts	To know for…	
		Project	Exam
Triton Inference Server	• What it is • ONNX • Lab on deploying a MobileNet model on a Triton server		
ML model pipeline	• What it is • Standard steps of ML pipelines		Yes
ML platforms & orchestrators	• Sagemaker offerings (workbench, training, registry. Predictions & pipelines)	Possibly (Vertex equivalent)	
Vertex Pipeline	• Build a vertex pipeline	Possibly	

That’s it for today!

LIÈGE université

1. Data parallelism

Given **p worker** devices/machines, and a batch of **K data samples**, the iterative-convergent formula would be:

$$A(t) = F(A(t-1), \sum_{p=1}^{P} \Delta_{\mathcal{L}}(A(t-1), \mathbf{x}_p))$$

Sum of update functions $\Delta_{\mathcal{L}}$ over parallel workers $p = 1, \ldots, P$

$\mathbf{x}_p$ is the subset of data assigned to worker p
Data indices at different workers do not overlap

https://tech.preferred.jp/en/blog/technologies-behind-distributed-deep-learning-allreduce/