oAdapting to model Change

Ketika dihadapkan pada suatu perubahan banyak hal yang mungkin terjadi, especially machine learning

1. Upstream Model
2. **Source data yang** di maintain **Tim** lain
3. Relationship antara **Features dengan labels**
4. **Distributions input**

Changing Distribution → model inputs mengalami perubahan (I.e Distribusi nya berbeda dari sebelumnya), dan ini bisa disebabkan oleh banyak **Hal**

1. **Waktu**
2. **Trends**
3. **Dan beberapa constraint lainnya**

Dan tentu, perubahan itu bisa datang mulai dari **Labelnya,** sampai ke **Featurenya**.

Extrapolasi → Generalisasi distribusi data yang berbeda pada training

Interpolasi →Generalisasi distribusi data pada training

Beberapa cara untuk **Protect** datamu dari **Changing distribustion adalah**

1. Monitoring (Misalnya Mean/STDnya tiba tiba berubah secara signifikan)
2. Cek residual (Maksudnya adalah cek **Error** dari predictions dan **labelnya,**) ini gambarannya

3. **Regulary Retraining modelnya**
4. **Lakukan** Data **Recency (** Data terbaru, adalah data yang most important), Yep lakukan yang namanya pembobotan pada lossfunctionnya

Notes penting :

Data dependencies yang tidak diperlukan, meliputi

Legacy features → Valueable **pada waktu tertentu,** Namun seiring berjalannya waktu, fitur tersebut redundant

Bundled Features → **Secara kolektif** Valuable, namun jika dipisah tidak berguna (Ini maksudnya fiturnya saling **dependent**)

Right **And** Wrong **Decision**

Konsep ini sebenernya berkaitan dengan yang namanya **Data leakage** , **models** learning **unacceptable strategies,** misalnya

1. Prediksi **Majority class (kasus imbalance data**
2. **Menggunakan sebuah fitur yang tidak diketahui.** Misalnya **Empty string**

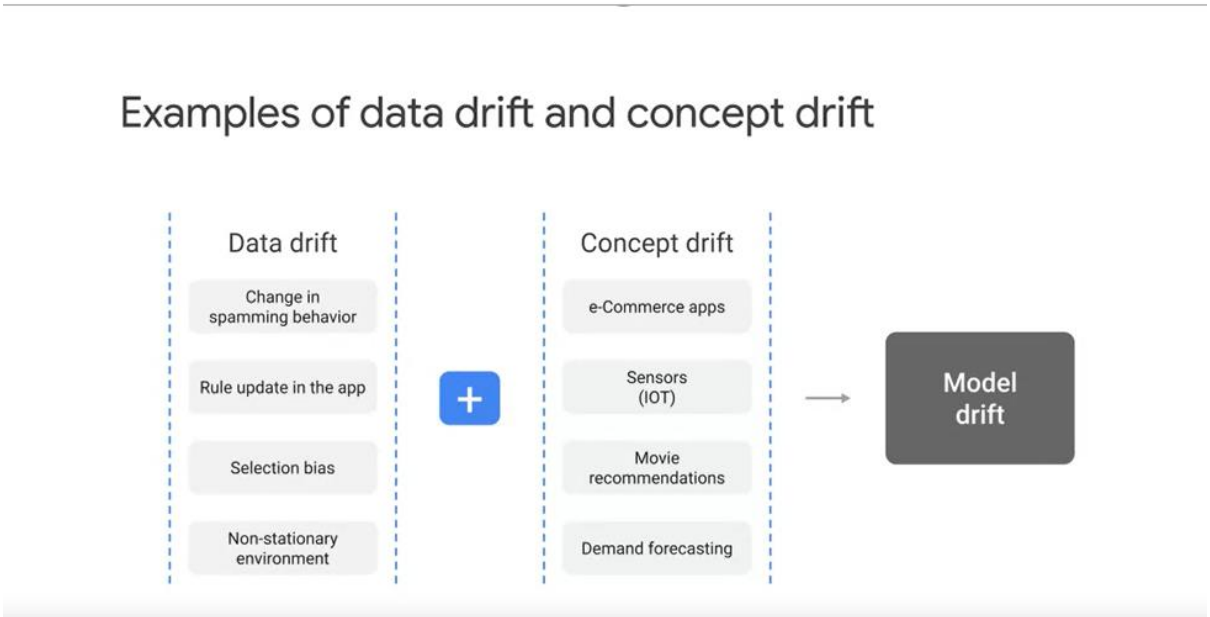
Selama melakukan Maintanability terhadap system failure yang dapat disebabkan baik dari sisi Server ataupun model

1. Siapkan Infrastuktur yang dapat **Rollback state model** jika terjadi kesalahan
2. Selalu sajikan model dalam keadaan terbaru

Cold start problem → Dimana model sudah usang (Ga sesuai lagi prediksinya), solusinya

1. Retraini model
2. Memahami limit dari modelnya

Data drift + Concept drift → **Model drift**



Waktu adalah permainan penting **didalam concept drift**

Aksi yang bisa dilakukan Untuk **Mitigasi concept drift**

Kalau Data Drift , Cari data lagi untuk **Introduce new classes** (jika ada)/ update the data dan retrained model

Concept Drift, Data yang lama mengalami perubahan distribusi, dan relationship dengan si label sehingga perlu pelabelan ulang, pembaruan data, dan retraining model

Selain, kedua hal diatas, lu bisa melakukan **Design system yang dapat deteksi perubahan**

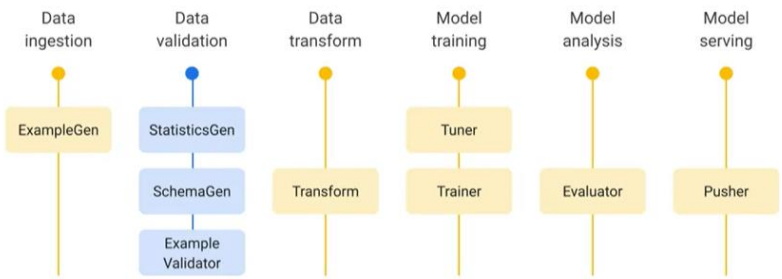
a. Reivent the model
b. Atau mulai Training dari checkpoint

Bisa juga menggunakan **An ensemble approach to train**

Maksudnya adalah **Melakukan training data baru dan menggunakan knowledge** data lama**,** untuk **mendapatkan knowledge baru yang ditransfer tanpa** mengabaikan **knowledge sebelumnya**. Pada saat melakukan retraining model. DIVERSITY dari Old concept may apply tapi Pastikan Konsep baru yang hendak di training menggunakan **low diversity**

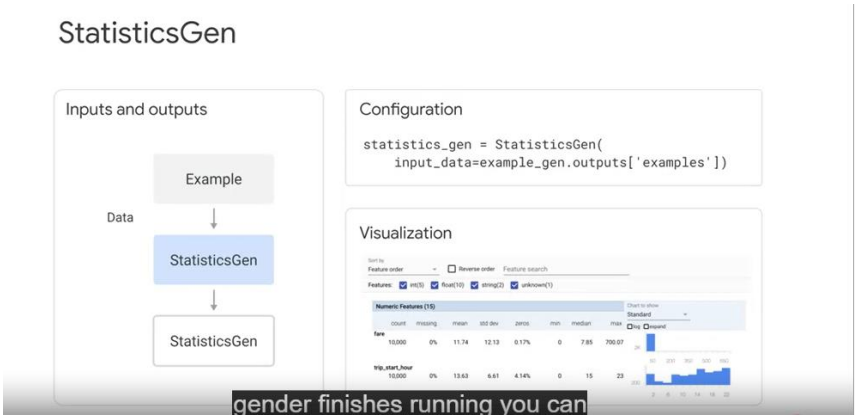
Tensorflow data validation

Components Dari tensorflow data validation



Statistic Gen

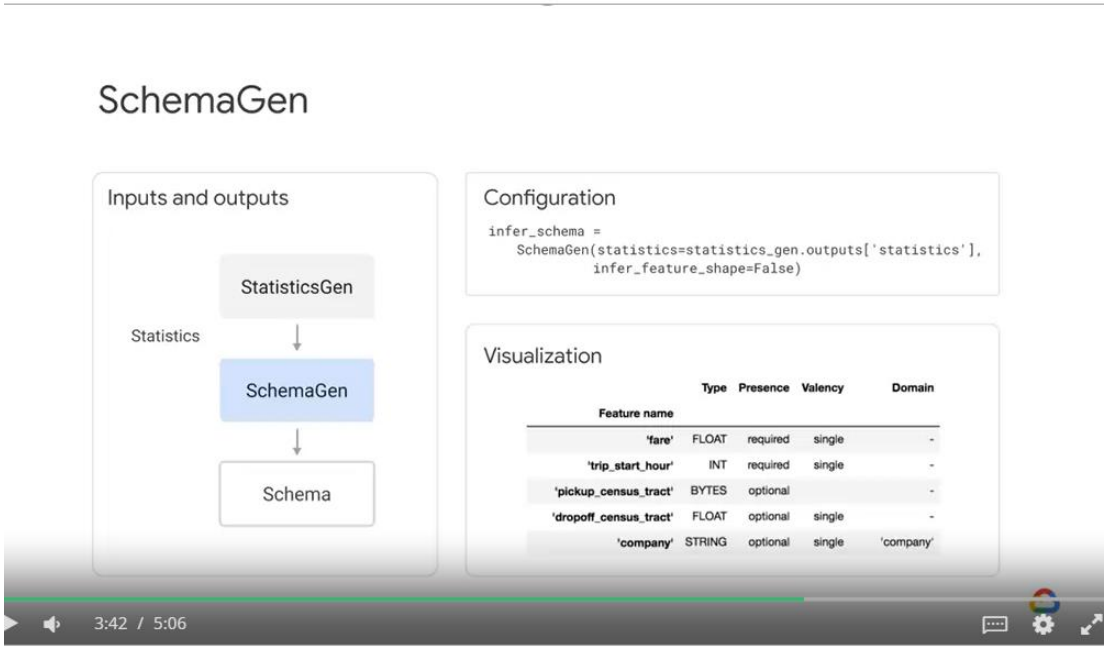
1. Ngebuat informasi statistik mengenai data untuk kemudian **digunakan visualisasi dan validasi**



Dalam statistic gen, Beriku ini adalah beberapa **Validasi data checks yang dapat dilakukan**

1. **Fitur** Mean, Max, Min, Mode, Median
2. **Korelasi fitur**
3. **Imbalance Class**
4. **Missing values**
5. **Histogram (Numeric & categorical)**

Schema Gen → Inisiasi Skema dari dataset (Mirip kek Skema database)



Type Columns → Indikasi Tipe data variable

Presence columns → Indikasi apakah sebuah fitur harus ada dalam sebuah instance / row

Valency → Indikasi jumlah value dari fitur tersebut per instance

Domain → Indikasi Domain dan valuenya, Maksudnya ini untuk setiap kolom kategorikal, dibutuhkan satu nilai kategorikal (LIST ACCEPTABLE VALUES)

Kapan menggunakan TVDF ?

1. Validasi Data baru Untuk **Inference** , memastikan bahwasanya tidak ada **bad features**
2. **Validasi** Data baru untuk **Inference**, Memastikan bahwasanya **Model yang dilatih** Memiliki informasi untuk melakukan decision berdasarkan data baru (DIstribusinya tidak berubah Signifikan)
3. **Validasi** Data **sehabis melakukan transformasi dan feature engineering** (Make sure everything goes well

Kenapa perlu menggunakan TFDV ? Dikarenakan agar Errornya **roughly the same**, Maksudnya adalah

1. Numerical Features antara training, eval, dan serving (memiliki Mean, median , std) yang roughly the same.
2. Categorical features juga ya kek gitu Ga ada Imbalance antara

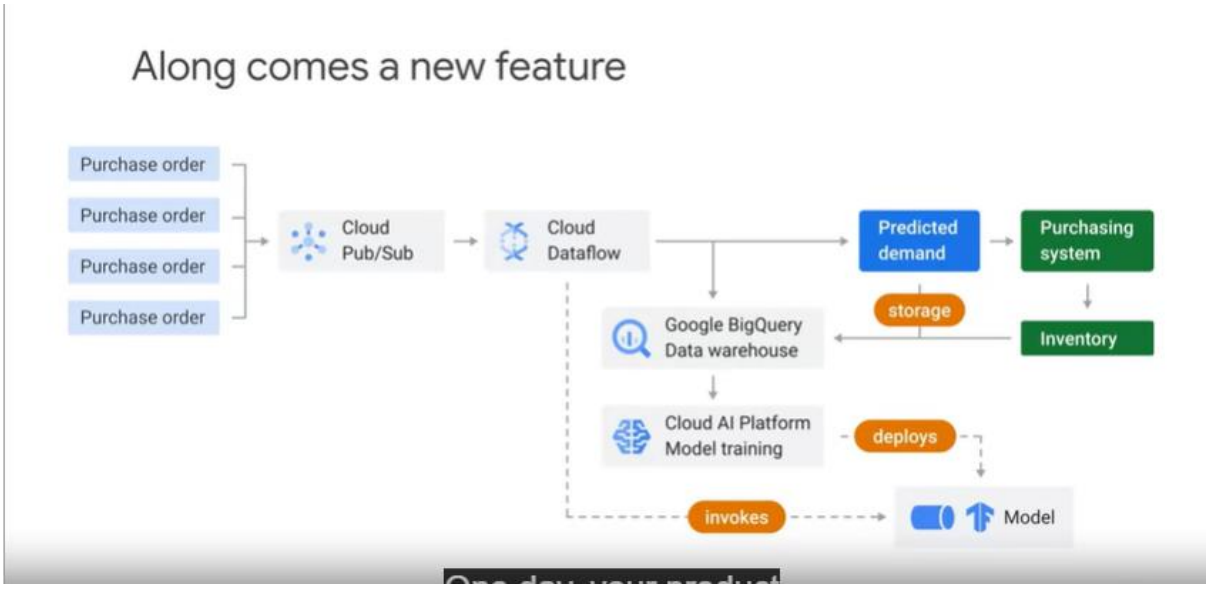
Skew

1. Schema Skew → Ketika Schema Training Different Dengan Eval, dan Serving
2. Feature Skew → Terjadinya perbedaan **nilai fitur** pada saat training dengan serving (Misalnya (terjadi perubahan logika transformasi data)
3. **Distribution skew →** Yep Distribusi dari training dengan eval dan serving totally different . Yang ini disebeabkan oleh beberapa hal (data sourcesnya, Different code mechanism , dan the world)

Training/Serving skew terjadi karena terdapat perbedaan Perfomances antara Training dengan serving yang disebabkakn oleh salah satu hal dibawah ini

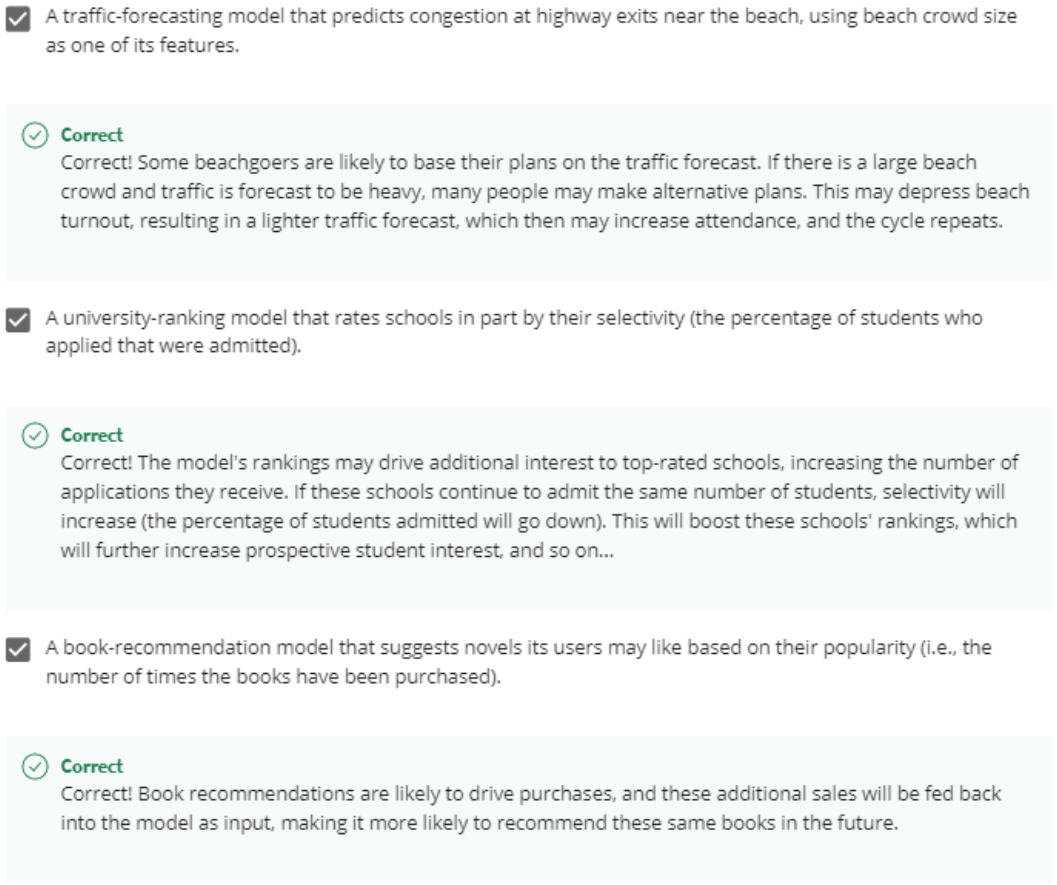
1. Perubahan yang terdapat diantara data training dengan data serving
2. Perbedaan Handle data pada training dengan serving pipelines

Diagnosing A **Production model**



Dari sini, sebenarnya kita bisa dapat bahwasanya nanti si Input data itu akan masuk **Telebih dahulu ke cloud pub/sub** data di pass ke Cloud DataFlow , nanti si cloud dataflow akan memanggil **Recent model untuk melakukan prediksi**, Setiap **Prediksi, data input dan inventory** AKAN DISIMPAN ke **Data warehouse (meanse that setiap transaksi data itu** Disimpan)

Feedback Loop Problem → Rare instance muncul, dan diperlukan bantuan manusia untuk membetulkannya



Pengantar Feedback Loop

(HUMAN IN THE LOOP)

HITL refers to systems that allow humans to give direct feedback to a model for predictions below a certain level of confidence.

Jadi FeedbackLoop Problem itu seperti **Rare dataset** muncul , dan dijadikan input (Kalau kita ngga ngasuh tau model yang ada Akurasi prediksinya menurun

What is high performance ML ?

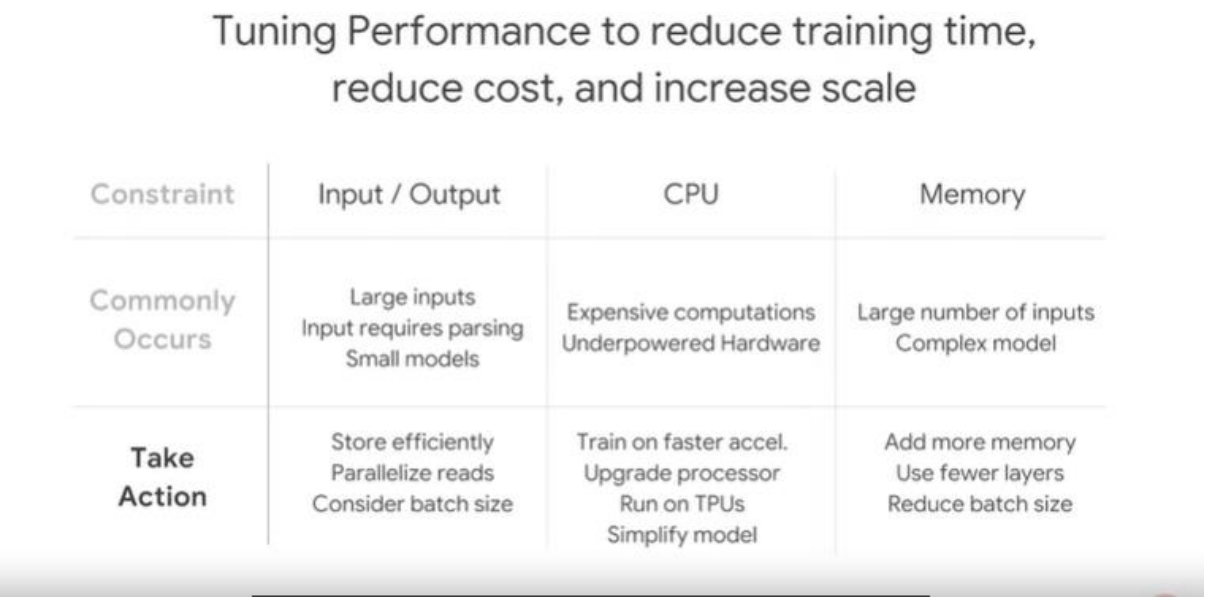
Kalau discourse ini lebih ke **Infrastruktur performance (training time)** Untuk metrics lainnya (AKurasi, budget) itu ga diperimbangkan mendentail

Tips **Optimizing training budget**

1. **Time** → Seberapa lama waktu yang dialokasikan untuk training ? Berapa kali training yang dibutuhkan per satuan waktu ? 1 Jam ? 2 Jam ? Model memiliki waktu update selama berapa waktu ? 4 jam sebelum Hari H ?

   Maksud waktu disini adalah (Seberapa lama waktu yang lu butuhkan untuk melakukan training, deploy, dan kapan model latest tersebut harus tersedia)

2. **Cost → Computing Cost n**ya sanggup berapa untuk dapat balance dengan time dan costnya
3. **Scale → Lu mau training di** Compute instance yang bagus but expensive, atau comput instances yang murah tapi banyak ? Drawbacksnya adalah **Cost dan time**.



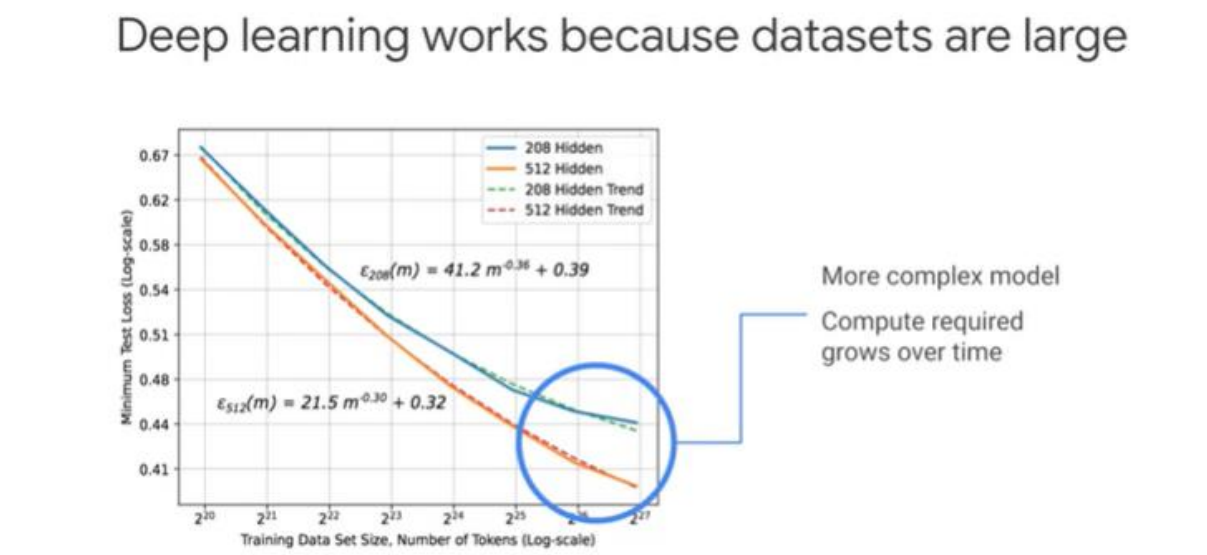
Input/Output → Large inputs Kek streaming data atau batch yang sangat besar

CPU → Model lu Punya Komputasi yang **WEDAN Beratnya**

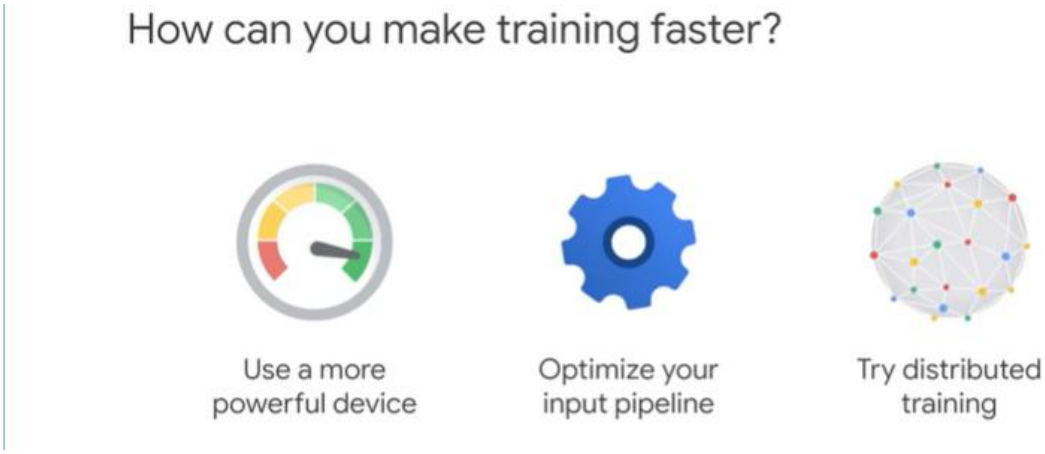
Memory → Weight yang bisa dihold oleh system memorynya

WHY Distributed Training Dibutuhkan ?

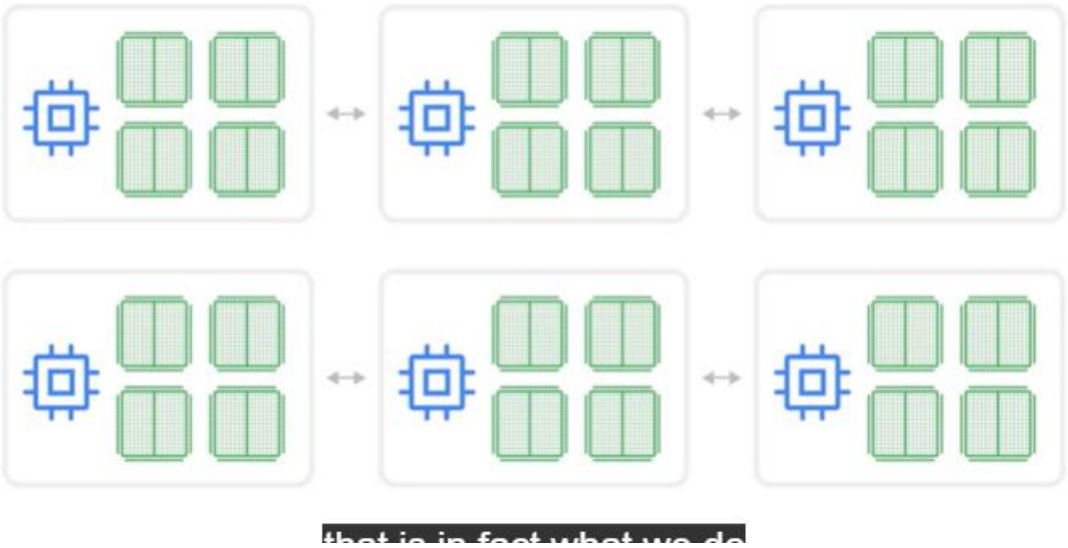
Pada Dasarnya semakin banyak datanya, maka model **Deeplearning semakin** baik. Dan tentu didukung oleh kompleksitas model. But Drawbacksnya ya **Compute powernya besar**



Ini lah mengapa **Distributed system dibutuhkan** Untuk training machine learning

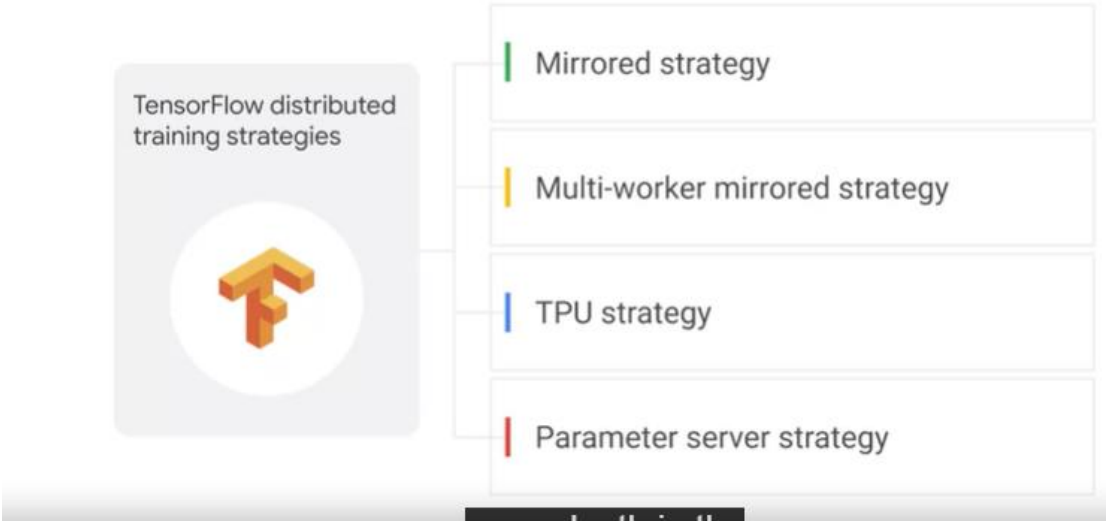


Ini gambarannya



Distributed Training Acrchitectures

Tensorflow distributed training strategies

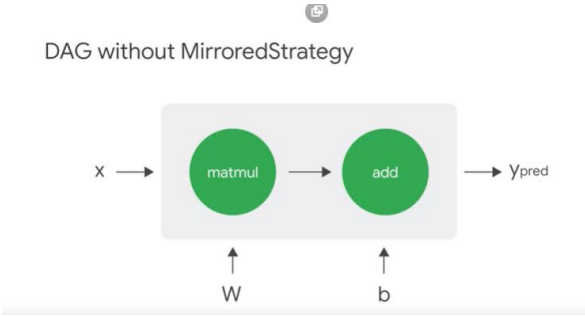


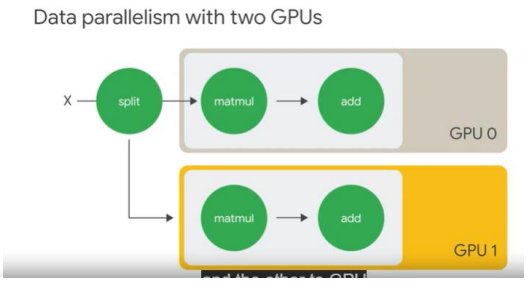
Mirrored Strategy (SNychronous Distributed training)

Single machine, multiple GPU

Step yang dilakukan oleh **mirrored strategy adalah**

1. Ngebuat Replica Model Untuk Setiap **GPU**
2. **Setiap** Training **1 minibatch** Di **Split sebanyak n Gpu**
3. **Mirrored strategy bakalan Manajemen** Gradient dan Update di seluruh GPU yang digunakan

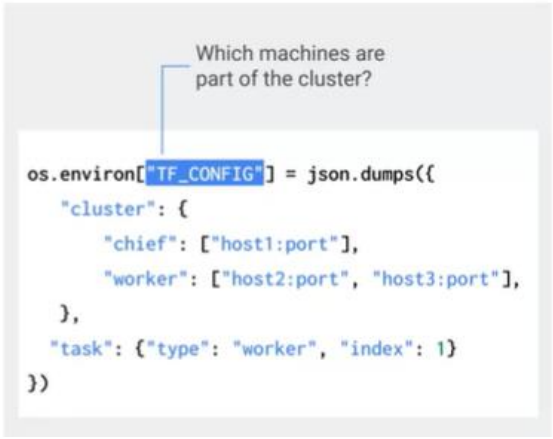




Multiworker-mirrored strategy (Synchronous Distributed training) (Data parallelism)

Mirip dengan yang diatas, perbedaannya hanya disini

1. Tensorflow perlu tau **Machines mana yang masuk kedalam Clusternya**



2. Dibutuhkan worker untuk eksekusi training
3. Dibutuhkan Chief untuk menyimpan training chekcpoints, Writing summary Files to Tensorboard
4. Synchronisasi Seluruh gradient hasil training
5. Saving
    a. Chief Worker Save **final model** ke **Desired model directory**
    b. **Other worker** Save ke **Temporary directory (Unique)**

Tpu strategy

More faster dari kedua hal diatas (Menggunakan TPU bkan GPU <training lebih cepat)

Parameter server strategy (Asynchronous Distributed training)

1. **Step yang dilakukan** Mirip dengan **step asyn**

2. **Setiap worker** akan update weight dll secara **Independent (tidak ada sinkronisasi)**
3.