

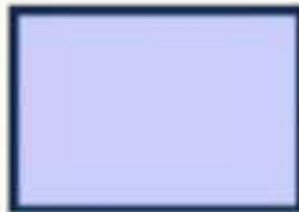
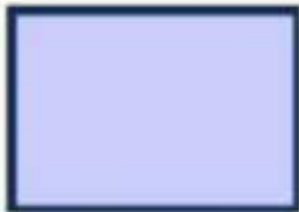


Deep Learning with Google Cloud

Google Cloud Platform

Robert Saxby
Khalid Salama
Brian Gibson

Big Data Product Specialist, Google Cloud
Machine Learning Product Specialist, Google Cloud
EMEA Higher Education Lead, Google Cloud



Dialing...



Hitting the limits early on



The Anatomy of a Large-Scale Hypertextual Web Search Engine

Authors: Sergey Brin and Lawrence Page
Date: 1996
Source: Computer Science Department, Stanford University

Abstract
This document is an early draft of a paper describing the design and implementation of the Google search engine. It is intended as a technical reference for those interested in the design of a large-scale search engine. The paper describes the architecture of the search engine, the algorithms used for indexing and searching, and the hardware and software requirements. It also discusses the challenges of scaling the search engine to handle a large volume of requests and the need for a distributed architecture.


Keywords: Web, Search Engines, Information Retrieval, Algorithms, Design

1. Introduction
The purpose of this document is to describe the design and implementation of the Google search engine. The search engine is designed to handle a large volume of requests and to provide fast and accurate search results. The design is based on a distributed architecture and uses a variety of algorithms to index and search the web. The hardware and software requirements are also discussed.

The Anatomy of a Large-Scale Hypertextual Web Search Engine

1996, Sergey Brin and Lawrence Page
Computer Science Department, Stanford University, Stanford,
CA 94305

Google

Google Search

I'm Feeling Lucky

The background of the image is a vast, multi-level data center. The ceiling is a complex network of dark metal beams and pipes, with numerous lights. Below, rows of server racks are visible, some with glowing blue and yellow lights. The floor is a light-colored, polished tile. The overall atmosphere is industrial and high-tech.

Google



Google Search

I'm Feeling Lucky

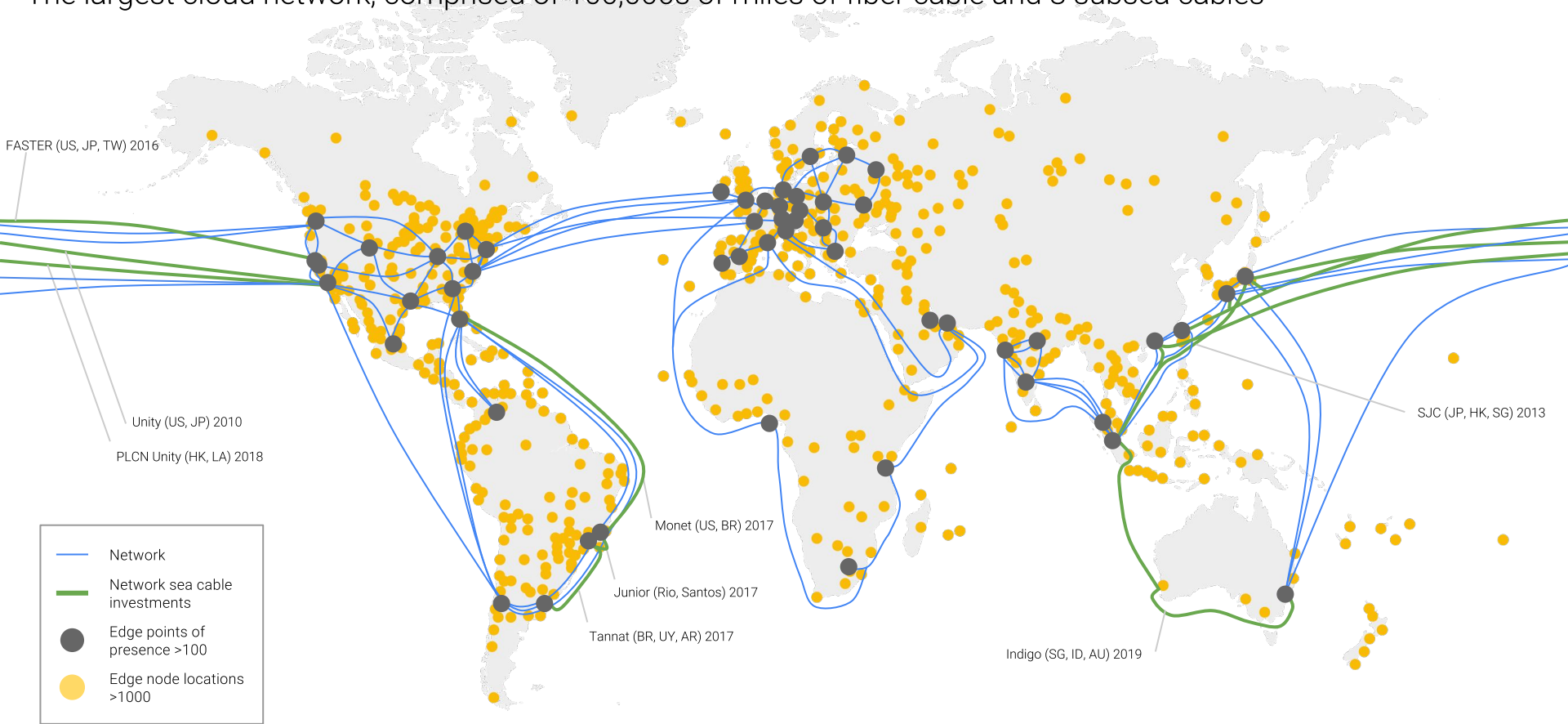
Google



7 Cloud products with 1 billion users

Google Network

The largest cloud network, comprised of 100,000s of miles of fiber cable and 8 subsea cables



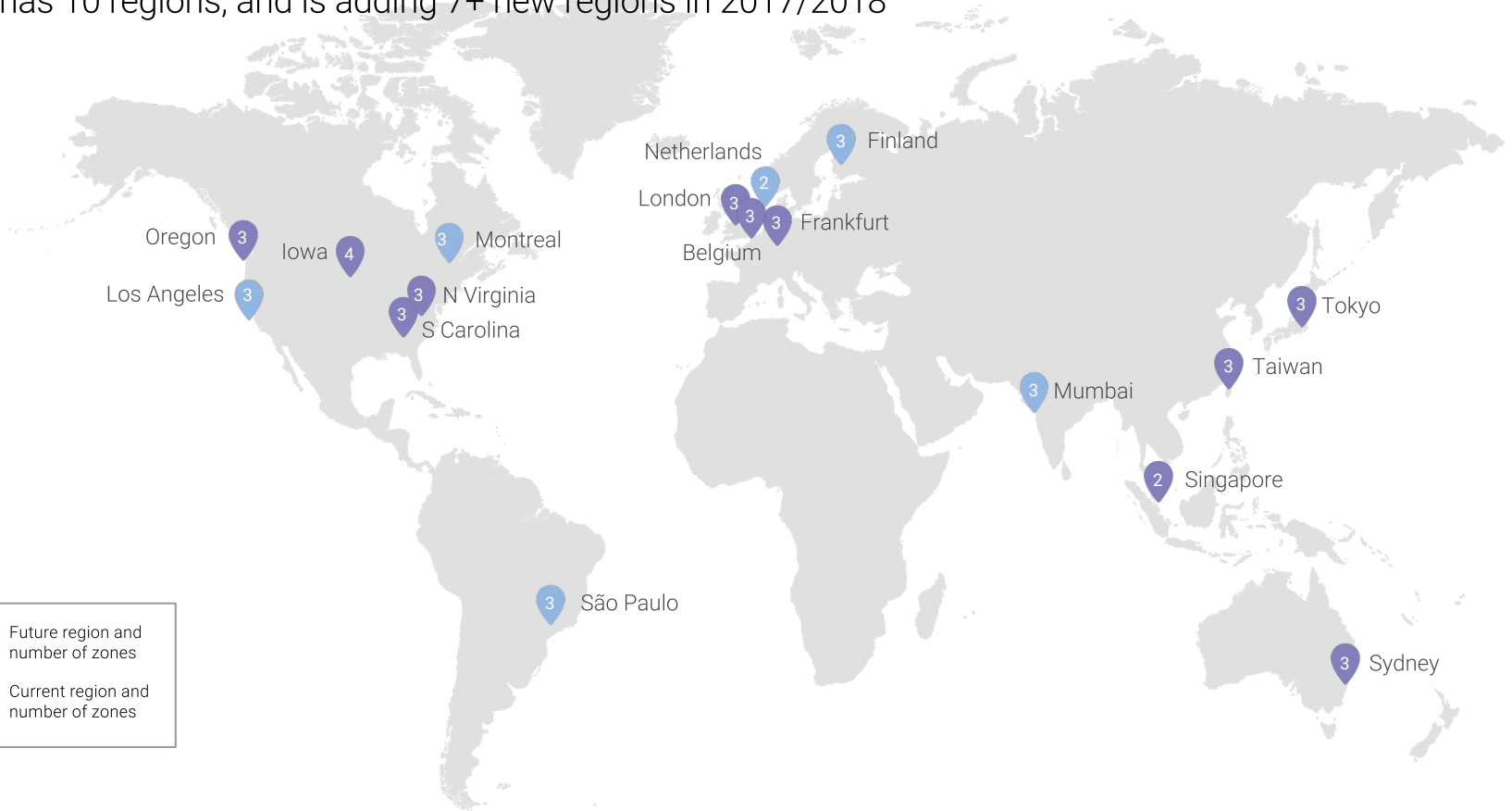
A dark grey world map is shown with a network of thin, light-colored lines connecting various points across the globe. These points are represented by small, solid yellow circles. The connections are most dense in North America and Europe, with lines extending across the Atlantic and Pacific Oceans. The overall aesthetic is technical and global.

\$29.4 Billion

3 Year Trailing CAPEX Investment

GCP Regions

GCP has 10 regions, and is adding 7+ new regions in 2017/2018

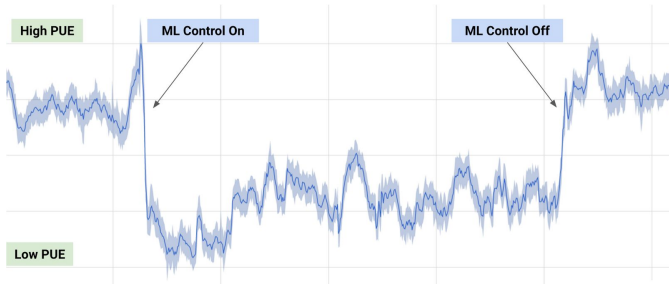


Light blue pin: Future region and number of zones
Dark blue pin: Current region and number of zones

Sustainability

Google datacenters have **half the overhead** of typical industry data centers

Largest private investor in renewables: \$2 billion generating 3.2 GW



Applying Machine Learning produced **40% reduction in cooling energy**



Advancing the state of the art...

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in this paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

1 Introduction

Over the past five years, the authors and many others at Google have implemented hundreds of special-purpose computations that process large amounts of raw data, such as crawled documents, web request logs, etc., to compute various kinds of derived data, such as inverted indices, various representations of the graph structure of web documents, summaries of the number of pages crawled per host, the set of most frequent queries in a

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures combine to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then applying a *reduce* operation to all the values that shared the same key, in order to combine the derived data appropriately. Our use of a functional model with user-specified *map* and *reduce* operations allows us to parallelize large computations easily and to re-execute as the primary mechanism for fault tolerance.

The major contributions of this work are a simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs.

Section 2 describes the basic programming model and gives several examples. Section 3 describes an implementation of the MapReduce interface tailored towards our cluster-based computing environment. Section 4 describes several refinements of the programming model that we have found useful. Section 5 has performance measurements of our implementation for a variety of tasks. Section 6 explores the use of MapReduce within Google including our experiences in using it as the basis

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber

{fay,jeff,sanjay,wilson,hser,m,burrows,tushar,fikes,gruber}@google.com

Google, Inc.

Abstract

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. In this paper we describe the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format, and we describe the design and implementation of Bigtable.

1 Introduction

Over the last two and a half years we have designed, implemented, and deployed a distributed storage system for managing structured data at Google called Bigtable. Bigtable is designed to replace a variety of petabytes of data and thousands of machines. Bigtable has achieved several goals: wide applicability, scalability, high performance, and high availability. Bigtable is used by more than sixty Google products and projects, including Google Analytics, Google Finance, Orkut, Personalized Search, Writely, and Google Earth. These products use Bigtable for a variety of demanding workloads, which range from throughput-oriented batch-processing jobs to latency-sensitive serving of data to end users. The Bigtable clusters used by these products span a wide range of configurations, from a handful to thousands of servers, and store up to several hundred terabytes of data. In many ways, Bigtable resembles a database; it shares many implementation strategies with databases. Parallel databases [14] and main-memory databases [13] have

achieved scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Data is indexed using row and column names that can be arbitrary strings. Bigtable also treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings. Clients can control the locality of their data through careful choices in their schemas. Finally, Bigtable schemes parameters for clients dynamically control whether to serve data out of memory or from disk.

Section 2 describes the data model in more detail, and Section 3 provides an overview of the client API. Section 4 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 5 describes the fundamentals of the Bigtable implementation, and Section 6 describes some of the refinements that we made to improve Bigtable's performance. Section 7 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 8, and discuss some lessons we learned in designing and supporting Bigtable in Section 9. Finally, Section 10 describes related work, and Section 11 presents our conclusions.

2 Data Model

A Bigtable is a sparse, distributed, persistent multi-dimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.

(row:string, column:string, time:int64) -> string

Dremel: Interactive Analysis of Web-Scale Datasets

Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassiliakis

Google, Inc.

{melnik, andrey.jlong, gromer, shiva, mtolton, theo.v}@google.com

ABSTRACT

Dremel is a scalable, interactive ad-hoc query system for analysis of read-only nested data. By combining multi-level execution trees and columnar data layout, it is capable of running aggregation queries over trillion-row tables in seconds. The system scales to thousands of CPUs and petabytes of data, and has thousands of users at Google. In this paper, we describe the architecture and implementation of Dremel, and explain how it complements MapReduce-based computing. We present a novel columnar storage representation for nested records and discuss experiments on few-thousand node instances of the system.

1. INTRODUCTION

Large-scale analytical data processing has become widespread in web companies and across industries, not least due to low-cost storage that enabled collecting vast amounts of business-critical data. Putting this data at the fingertips of analysts and engineers has grown increasingly important: interactive response times often make a qualitative difference in data exploration, monitoring, online customer support, rapid prototyping, debugging of data pipelines, and other tasks.

Performing interactive data analysis at scale demands a high degree of parallelism. For example, reading one terabyte of compressed data in one second using today's commodity disks would require tens of thousands of disks. Similarly, CPU-intensive queries may need to run on thousands of cores to complete within seconds. At Google, massively parallel computing is done using shared clusters of commodity machines [5]. A cluster typically hosts a multitude of distributed applications that share resources, have widely varying workloads, and run on machines with different hardware parameters. An individual worker in a distributed application may take much longer to execute a given task than others, or may never complete due to failures or preemption by the cluster management system. Hence, dealing with stragglers and failures is essential for achieving fast execution and fault tolerance [10].

First, its architecture borrows the concept of a serving tree used in distributed search engines [11]. Just like a web search request, a query gets pushed down the tree and is rewritten at each step. The result of the query is then assembled by aggregating the replies received from lower levels of the tree. Second, Dremel provides a high-level, SQL-like language to express ad-hoc queries. In contrast to layers such as Pig [18] and Hive [16], it executes queries entirely without translating them into MR jobs.

Lastly, and importantly, Dremel uses a column-stripped storage representation, which enables it to read less data from secondary storage than a row-stripped representation. This is particularly important for queries that primarily rely on their speed as opposed to torque. We use this name for an internal project only.

exchanged by distributed systems, structured documents, etc. lend themselves naturally to a *nested* representation. Normalizing and recombinning such data at web scale is usually prohibitive. A nested data model underlies most of structured data processing at Google [21] and reportedly at other major web companies.

This paper describes a system called Dremel¹ that supports interactive analysis of very large datasets over shared clusters of commodity machines. Unlike traditional databases, it is capable of operating on its *own* nested data. *In situ* refers to the ability to access data 'in place', e.g., in a distributed file system (like GFS [14] or another storage layer (e.g., Bigtable [8]). Dremel can execute many queries over each data that would ordinarily require a sequence of MapReduce (MR [22]) jobs, but at a fraction of the execution time. Dremel is not intended as a replacement for MR and is often used in conjunction with it to analyze outputs of MR pipelines or rapidly prototype larger computations.

Dremel has been in production since 2006 and has thousands of users within Google. Multiple instances of Dremel are deployed in the company, ranging from tens to thousands of nodes. Examples of using the system include:

- Analysis of crawled web documents.
- Tracking install data for applications on Android Market.
- Crash reporting for Google products.
- OCR results from Google Books.
- Spam analysis.
- Debugging of map files on Google Maps.
- Table migrations in managed Bigtable instances.
- Results of tests run on Google's distributed build system.
- Disk I/O statistics for hundreds of thousands of disks.
- Resource monitoring for jobs run in Google's data centers.
- Symbolic and dependencies in Google's codebase.

Dremel builds on ideas from web search and parallel DBMSs. First, its architecture borrows the concept of a serving tree used in distributed search engines [11]. Just like a web search request, a query gets pushed down the tree and is rewritten at each step. The result of the query is then assembled by aggregating the replies received from lower levels of the tree. Second, Dremel provides a high-level, SQL-like language to express ad-hoc queries. In contrast to layers such as Pig [18] and Hive [16], it executes queries entirely without translating them into MR jobs.

Lastly, and importantly, Dremel uses a column-stripped storage representation, which enables it to read less data from secondary storage than a row-stripped representation. This is particularly important for queries that primarily rely on their speed as opposed to torque. We use this name for an internal project only.

1 To appear in OSDI 2004

1

1 To appear in OSDI 2006

1

2004

2005

2006

@bagibson

...and creating a revolution

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized across a large cluster of commodity machines. The system takes care of the details of replication, fault-tolerance, data distribution and load balancing in a library. Our abstraction is in-



MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

1 Introduction

Over the past five years, the authors and many others at Google have implemented hundreds of special-purpose computations that process large amounts of raw data, such as crawled documents, web request logs, etc., to compute various kinds of derived data, such as inverted indices, various representations of the graph structure of web documents, summaries of the number of pages crawled per host, the set of most frequent queries in a

Translated in OSDI 2004

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures combine to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is in-

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Translated in OSDI 2004

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber

{fay,jeff,sanjay,wilson,hsieh,deborah,wallach,mike,rob}@google.com

Google, Inc.

Abstract

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving).

Bigtable is designed to reliably scale to petabytes of data and thousands of machines. Bigtable has achieved several goals: wide applicability, scalability, high performance, and high availability. Bigtable is used by more than sixty Google products and projects, including Google Analytics, Google Finance, Orkut, Personalized Search, Writely, and Google Earth. These products use Bigtable for a variety of demanding workloads, which range from throughput-oriented batch-processing jobs to latency-sensitive serving of data to end users.

The Bigtable clusters used by these products span a wide range of configurations, from a handful to thousands of servers, and store up to several hundred terabytes of data. In many ways, Bigtable resembles a database; it shares many implementation strategies with databases. Parallel databases [14] and main-memory databases [13] have

achieved scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Data is indexed using row and column names that can be arbitrary strings. Bigtable also treats data as interpreted strings, although clients often serialize data in forms of structured and semi-structured data as text strings. Clients can control the locality of their data through careful placement of data on servers. Bigtable also supports a rich set of operations for querying and analyzing data. Bigtable has grown increasingly important to Google products and projects, and has grown increasingly important to Google products and projects, and has grown increasingly important to Google products and projects.

To appear in OSDI 2006

Dremel: Interactive Analysis of Web-Scale Datasets

Georgy Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassiliadis

Google, Inc.

{melnik, andrey.jlong, gromer, shiva, mtolton, theo.v}@google.com

ABSTRACT

Dremel is a scalable, interactive ad-hoc query system for analysis of read-only nested data. By combining multi-level execution trees and columnar data layout, it is capable of running aggregation queries over trillion-row tables in seconds. The system scales to thousands of CPUs and petabytes of data, and has thousands of users at Google. In this paper, we describe the architecture and implementation of Dremel, and explain how it complements MapReduce-based computing. We present a novel columnar storage representation for graph data and discuss experiments on few-thousand node graph data analysis systems.

1. INTRODUCTION

Large-scale analytical data has become an important asset for many companies and academic institutions. This data is often stored in low-cost storage that enabled columnar data layouts and columnar data access. During this data access, the system has grown increasingly important to Google products and projects, and has grown increasingly important to Google products and projects.

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving).

2 Data Model

A Bigtable is a sparse, distributed, persistent multi-dimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.

Dremel is a scalable, interactive ad-hoc query system for analysis of read-only nested data. By combining multi-level execution trees and columnar data layout, it is capable of running aggregation queries over trillion-row tables in seconds. The system scales to thousands of CPUs and petabytes of data, and has thousands of users at Google. In this paper, we describe the architecture and implementation of Dremel, and explain how it complements MapReduce-based computing. We present a novel columnar storage representation for graph data and discuss experiments on few-thousand node graph data analysis systems.

To appear in OSDI 2006

exchanged by distributed systems, structured documents, etc. tend themselves naturally to a *nested* representation. Normalizing and recomputing such data at web-scale is usually prohibitive. A nested data model underlies most of structured data processing at Google [21] and reportedly at other major web companies.

This paper describes a system called Dremel that supports interactive analysis of very large datasets over shared clusters of commodity machines. Unlike traditional databases, it is capable of operating on its own nested data. *As-is* refers to the ability to access data “in place”, e.g., in a distributed file system (like GFS [14]) or another storage layer (e.g., Bigtable [5]). Dremel can execute many queries over such data that would ordinarily require a sequence of MapReduce [12] jobs.

Dremel is a scalable, interactive ad-hoc query system for analysis of read-only nested data. By combining multi-level execution trees and columnar data layout, it is capable of running aggregation queries over trillion-row tables in seconds. The system scales to thousands of CPUs and petabytes of data, and has thousands of users at Google. In this paper, we describe the architecture and implementation of Dremel, and explain how it complements MapReduce-based computing. We present a novel columnar storage representation for graph data and discuss experiments on few-thousand node graph data analysis systems.

- Tracking install data for applications on Android Market.
- Crash reporting for Google products.
- OCR results for Google Books.
- Spam analysis.
- Debugging of my files on Google Maps.
- Tablet migrations in managed Bigtable instances.
- Results of tests run on Google's distributed build system.
- Disk I/O statistics for hundreds of thousands of disks.
- Resource monitoring for jobs run in Google's data centers.
- Symbols and dependencies in Google's codebase.

Dremel builds on ideas from web search and parallel DBMSs. First, its architecture borrows the concept of a serving tree used in distributed search engines [11]. Just like a web search request, a query gets pushed down the tree and is rewritten at each step. The result of the query is assembled by aggregating the replies received from lower levels of the tree. Second, Dremel provides a high-level, SQL-like language to express ad-hoc queries. In contrast to layers such as Pig [10] and Hive [16], it executes queries natively without translating them into MR jobs.

Lastly, and importantly, Dremel uses a column-stripped storage representation, which enables it to read less data from secondary storage than would be required by a traditional row-oriented system. We use this name for an internal project only.

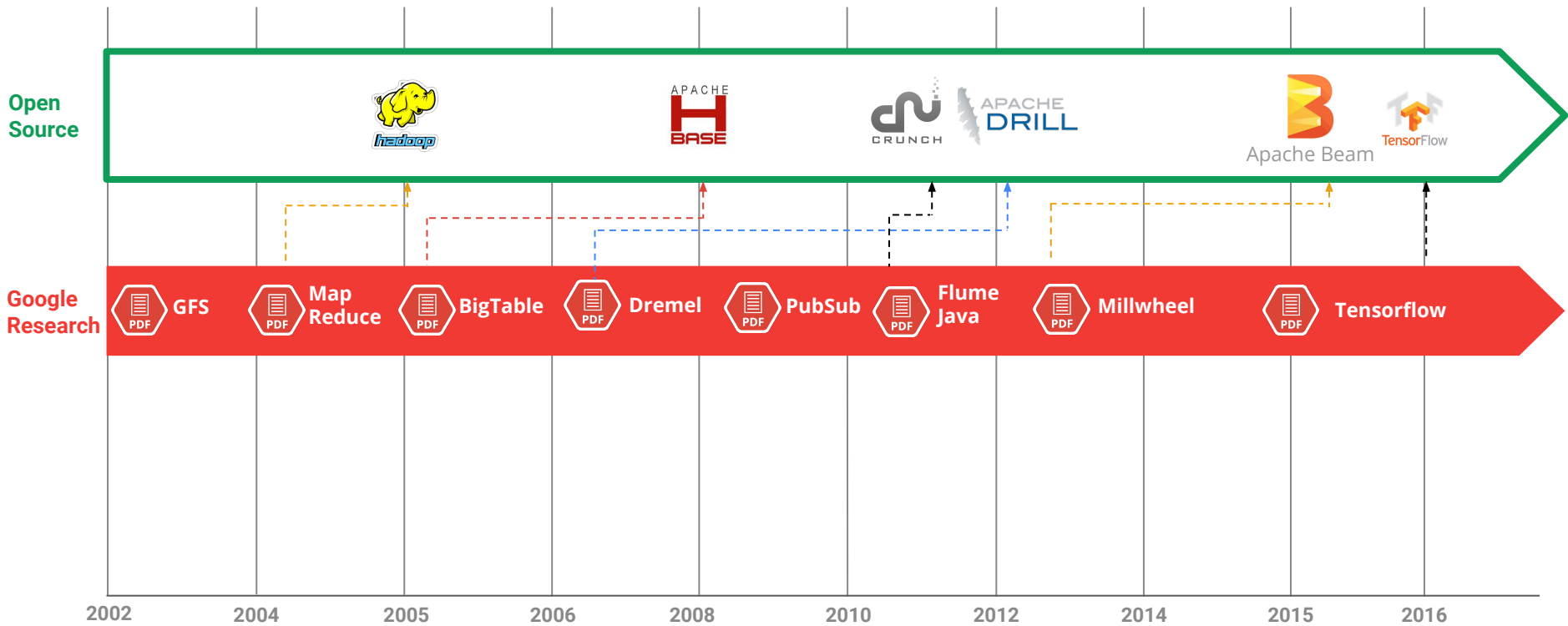
To appear in OSDI 2006

2005

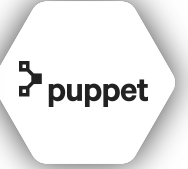
2008

2012

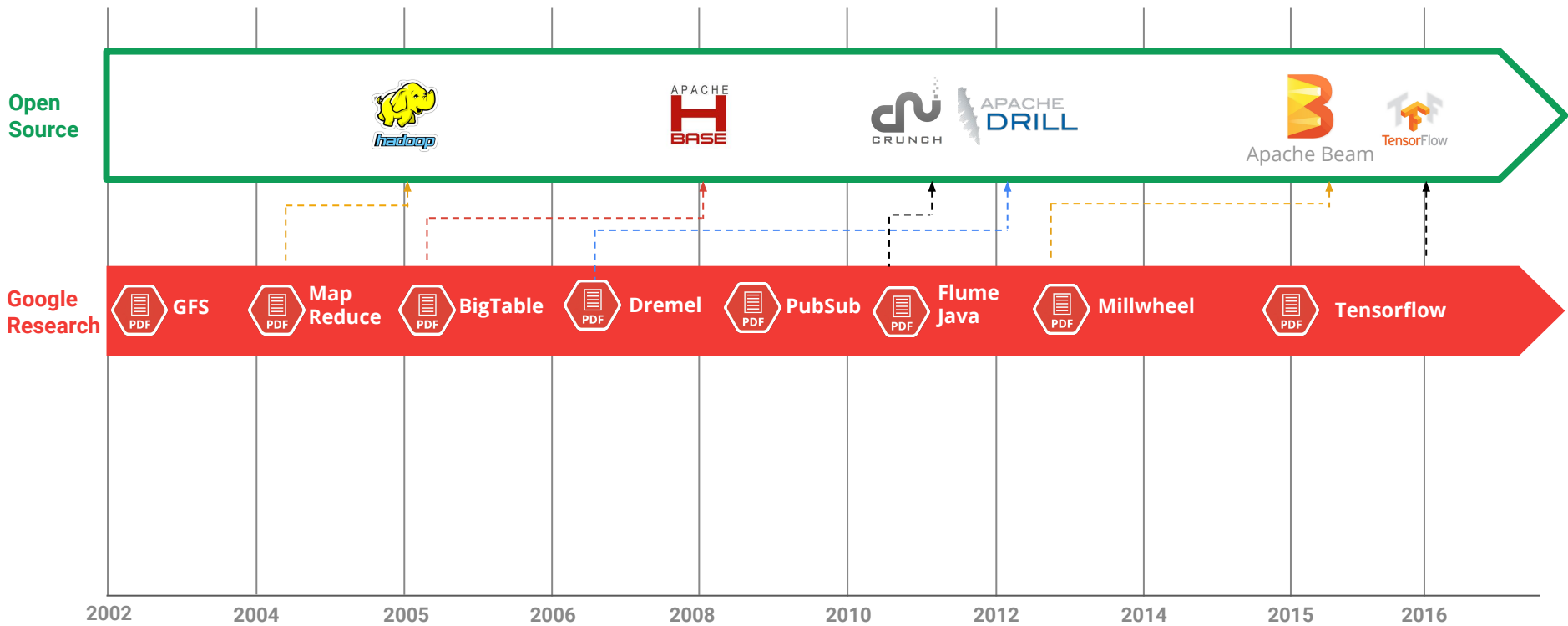
15+ years of solving Data Problems



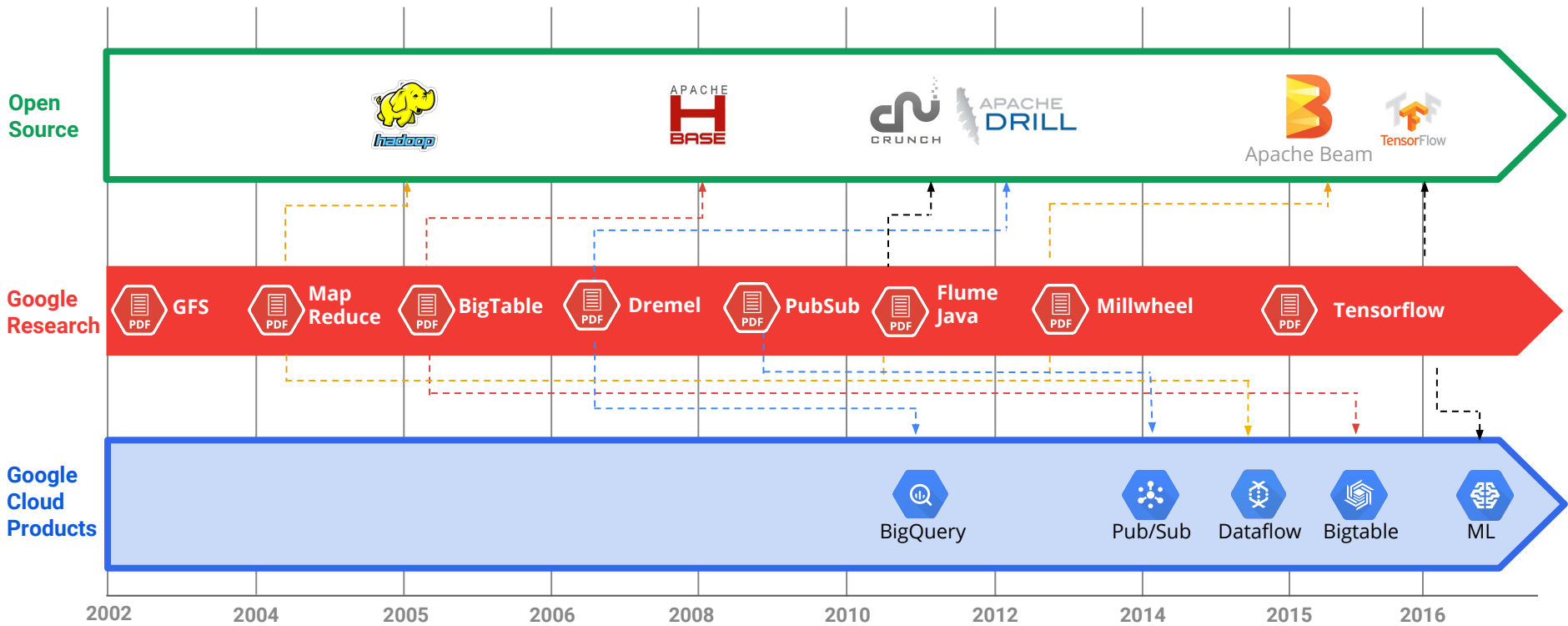
Walking the walk



15+ years of solving Data Problems



15+ years of solving Data Problems



More than 60 Google Cloud Platform services

Compute



Compute Engine



App Engine



Container Engine



Container Registry



Cloud Functions

Storage and Databases



Cloud Storage



Cloud Bigtable



Cloud Datastore



Cloud SQL



Cloud Spanner



Persistent Disk

Networking



Virtual Private Cloud



Cloud Load Balancing



Cloud CDN



Cloud Router



External IP



Cloud Interconnect



Cloud DNS



Networking



VPN

Big Data



BigQuery



Cloud Dataflow



Cloud Dataproc



Cloud Datalab



Cloud Pub/Sub



Genomics

Machine Learning



Cloud Machine Learning



Cloud Vision API



Cloud Speech API



Cloud Natural Language API



Cloud Translation API



Cloud Jobs API

Identity & Security



Cloud IAM



Cloud Resource Manager



Cloud Security Scanner



Key Management Service



Firewall



BeyondCorp



Data Loss Prevention



Identity-Aware Proxy



Security Key Enforcement

More than 60 Google Cloud Platform services

Management Tools



Stackdriver



Monitoring



Logging



Error Reporting



Trace



Debugger



Cloud Deployment Manager



Cloud Endpoints



Cloud Console



Cloud Shell



Cloud Mobile App



Cloud Billing API



Cloud APIs

Developer Tools



Cloud SDK



Cloud Deployment Manager



Cloud Source Repositories



Cloud Tools for Android Studio



Cloud Tools for IntelliJ



Cloud Tools for PowerShell




Cloud Tools for Visual Studio



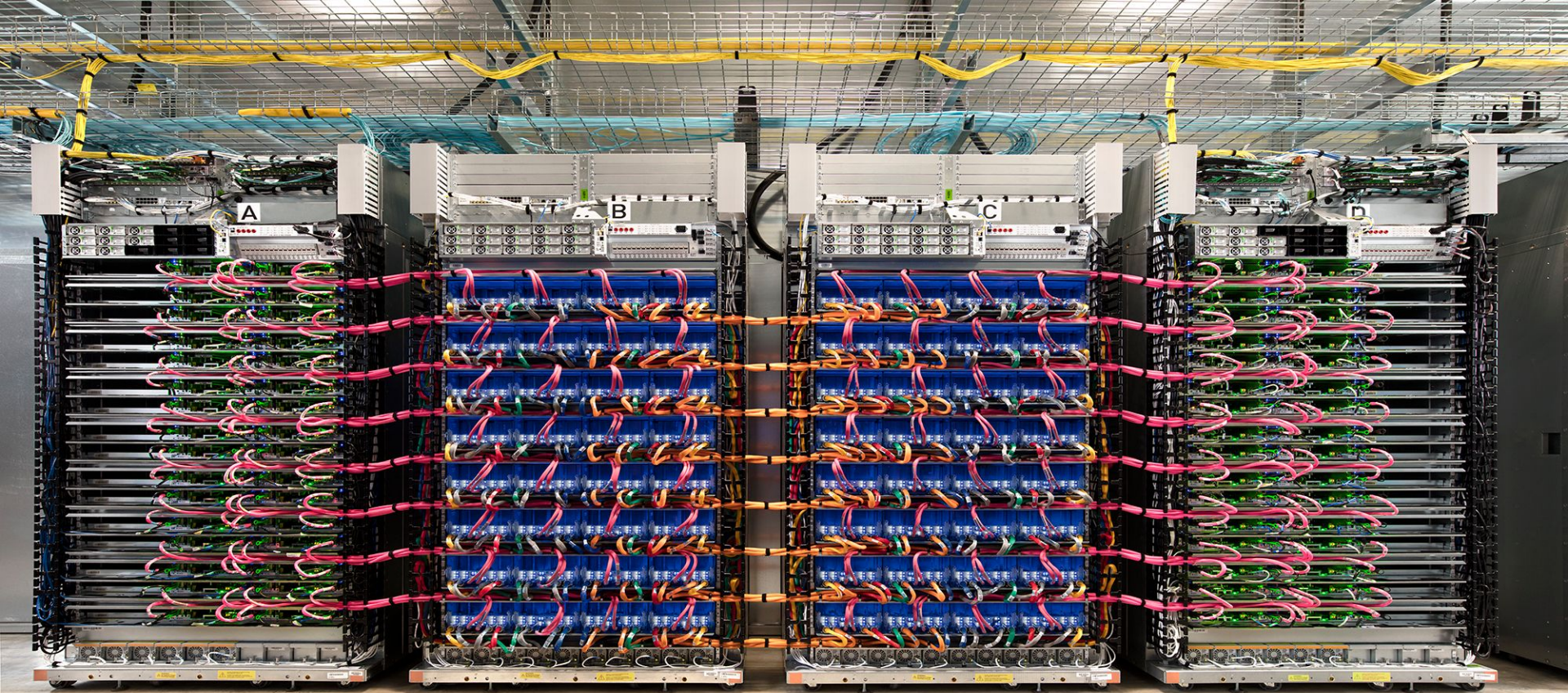
Google Plug-in for Eclipse



Cloud Test Lab

A person's hands are shown holding a small, detailed globe of the Earth. The hands are positioned over a tablet or screen that displays various digital data visualizations, including bar charts, line graphs, and network diagrams. The background is dark and filled with these digital elements, creating a high-tech, futuristic atmosphere. The text is overlaid in the center of the image.

Helping students,
researchers, and practitioners
build what's next



Big Data and Machine Learning

Khalid Salama - Machine Learning Product Specialist

Robert Saxby - Big Data Product Specialist

Two ways we can help you benefit from ML

Use your own data to train models



TensorFlow



Cloud Machine Learning Engine

Ready to use Machine Learning models



Cloud Vision API



Cloud Speech API



Cloud Jobs API



Cloud Translation API



Cloud Natural Language API

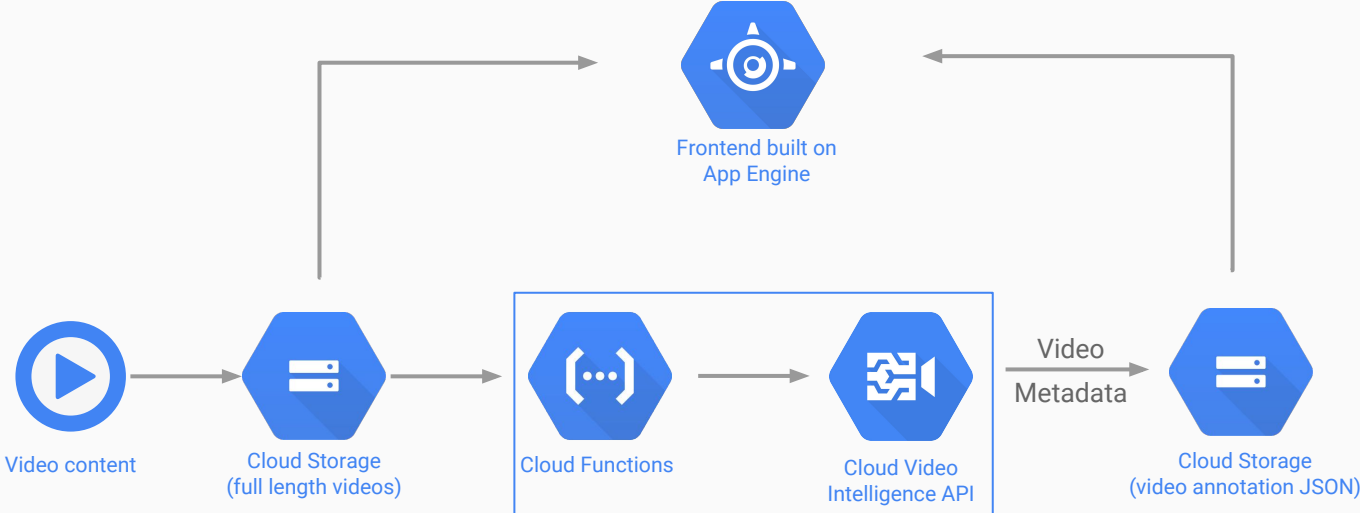


Cloud Video Intelligence API



Cloud Video Intelligence API demo

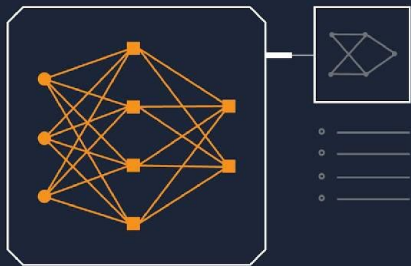
How the demo works



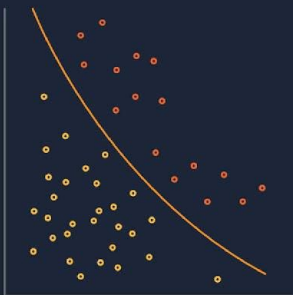
Built by @SRobTweets and @AlexWolfe



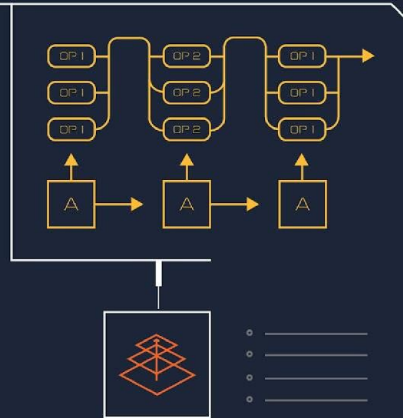
NLP demo



- _____
- _____
- _____



TensorFlow



What is TensorFlow?

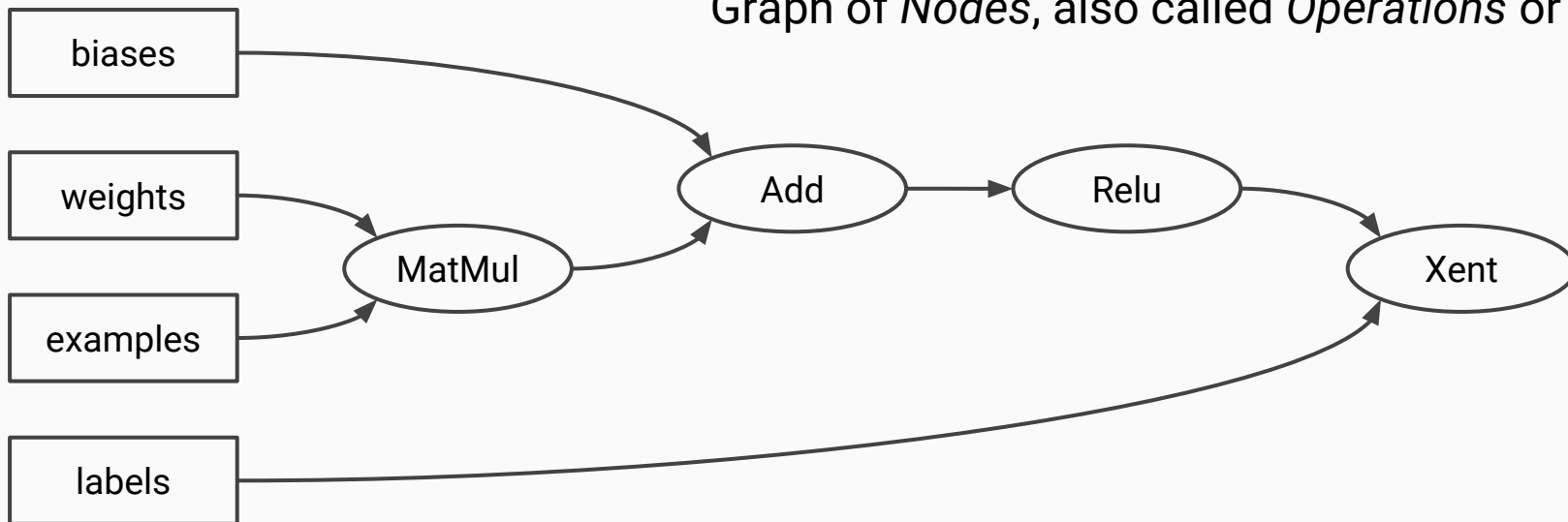


- A system for distributed, parallel machine learning
- It's based on general-purpose dataflow graphs
- It targets heterogeneous devices
 - A single PC with CPU
 - A single PC with GPU(s)
 - A mobile device
 - Clusters of 100s or 1000s of CPUs, GPUs and TPUs

Another data flow system



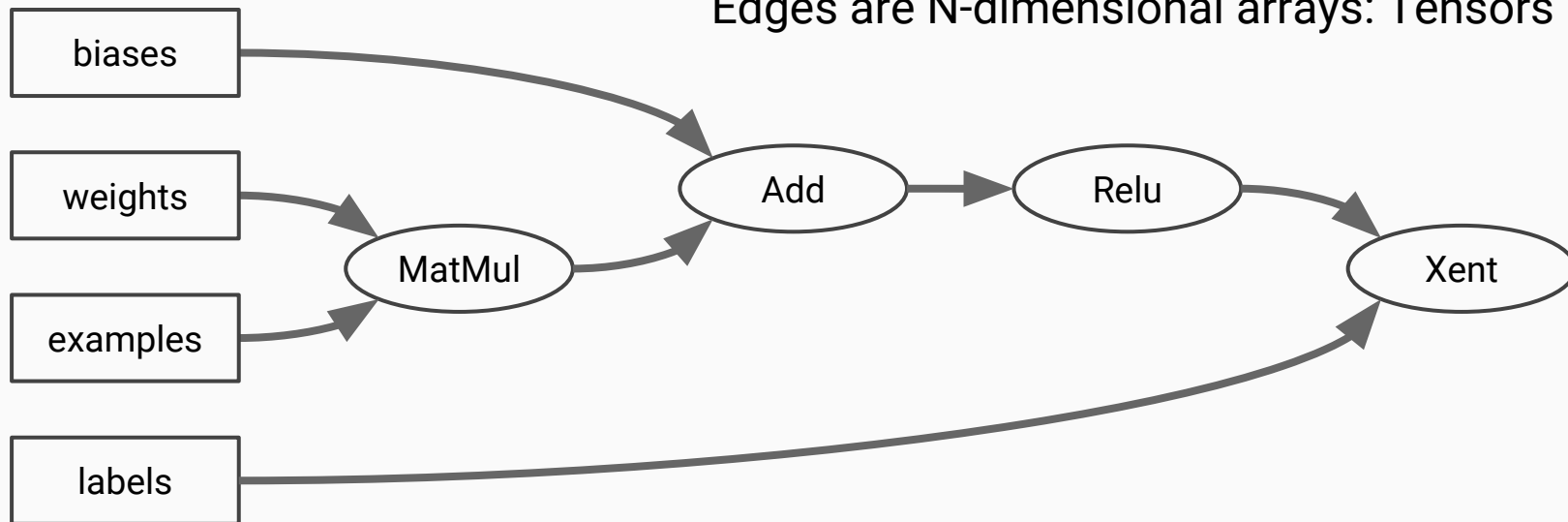
Graph of *Nodes*, also called *Operations* or *ops*



With tensors



Edges are N-dimensional arrays: Tensors



What's in a name?



0 Scalar (magnitude only)

$s = 483$

1 Vector (magnitude and direction)

$v = [1.1, 2.2, 3.3]$

2 Matrix (table of numbers)

$m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]$

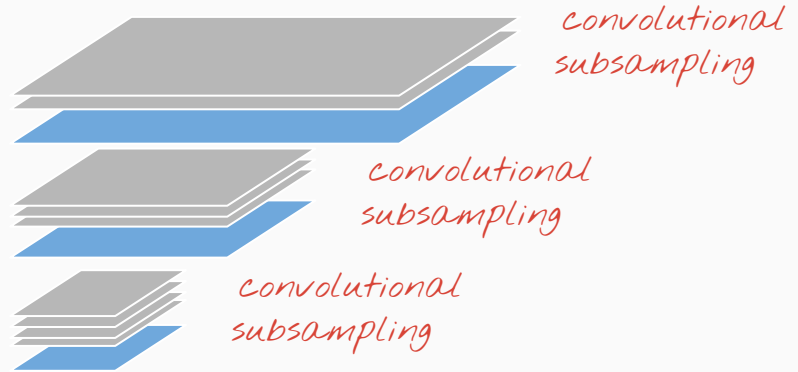
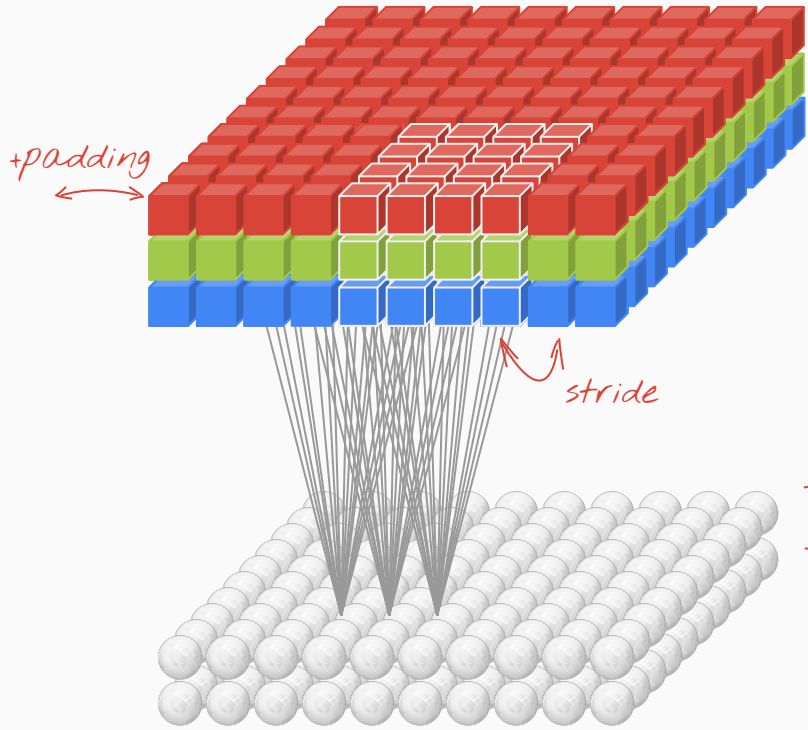
3 3-Tensor (cube of numbers)

$t = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]$

4 n-Tensor (you get the idea)

....

Convolutional layer



$$\begin{array}{l}
 W_1[4, 4, 3] \\
 W_2[4, 4, 3]
 \end{array}
 \left|
 \begin{array}{l}
 W[4, 4, 3, 2] \\
 \text{filter size} \\
 \text{input channels} \\
 \text{output channels}
 \end{array}
 \right.$$

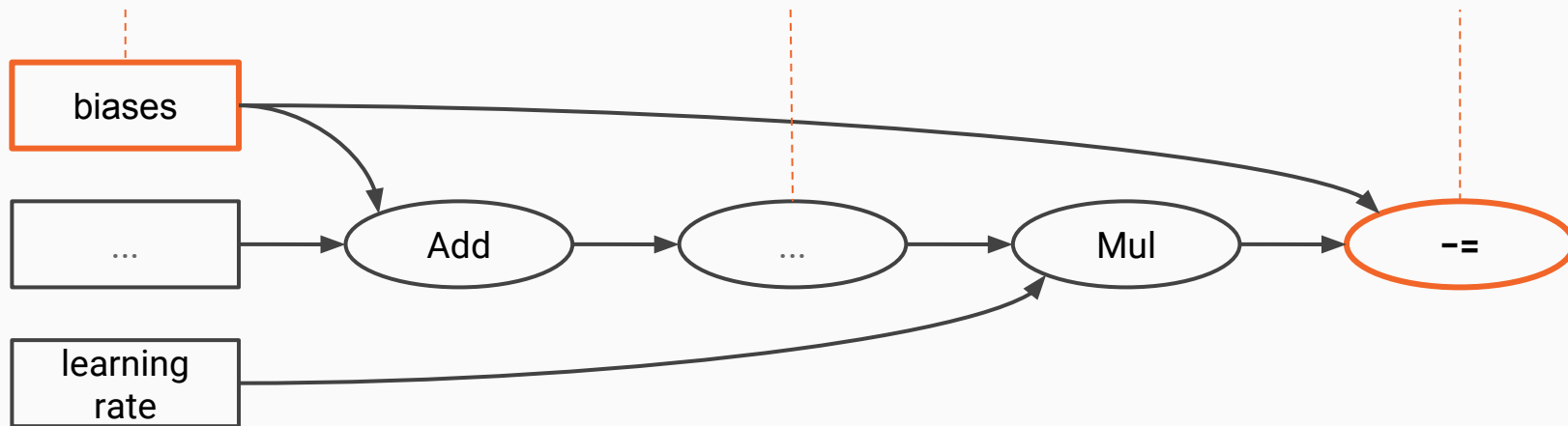
With state



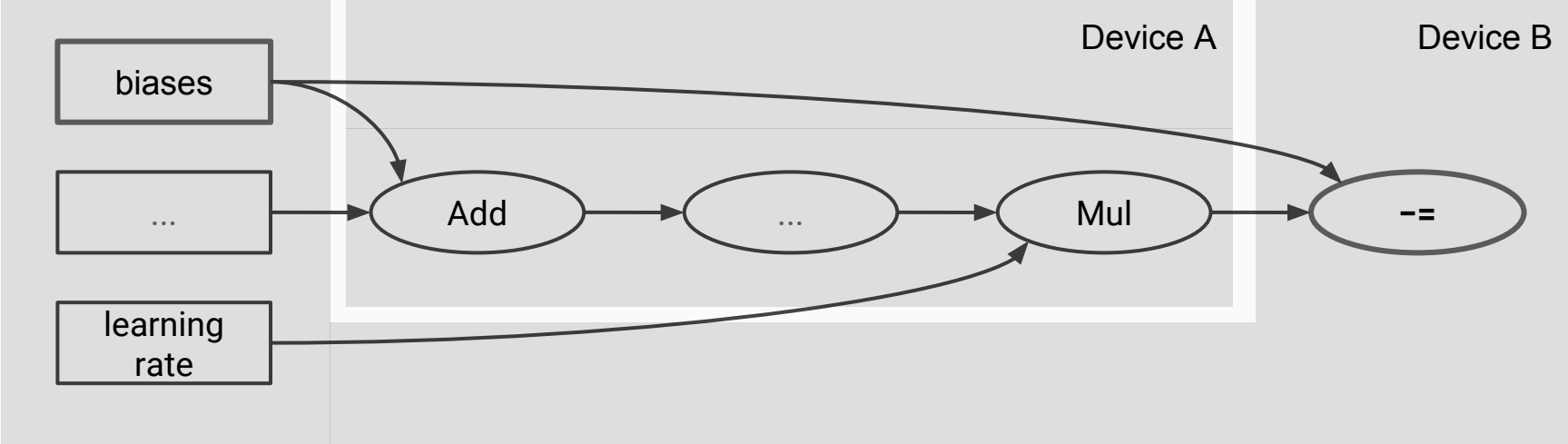
'Biases' is a variable

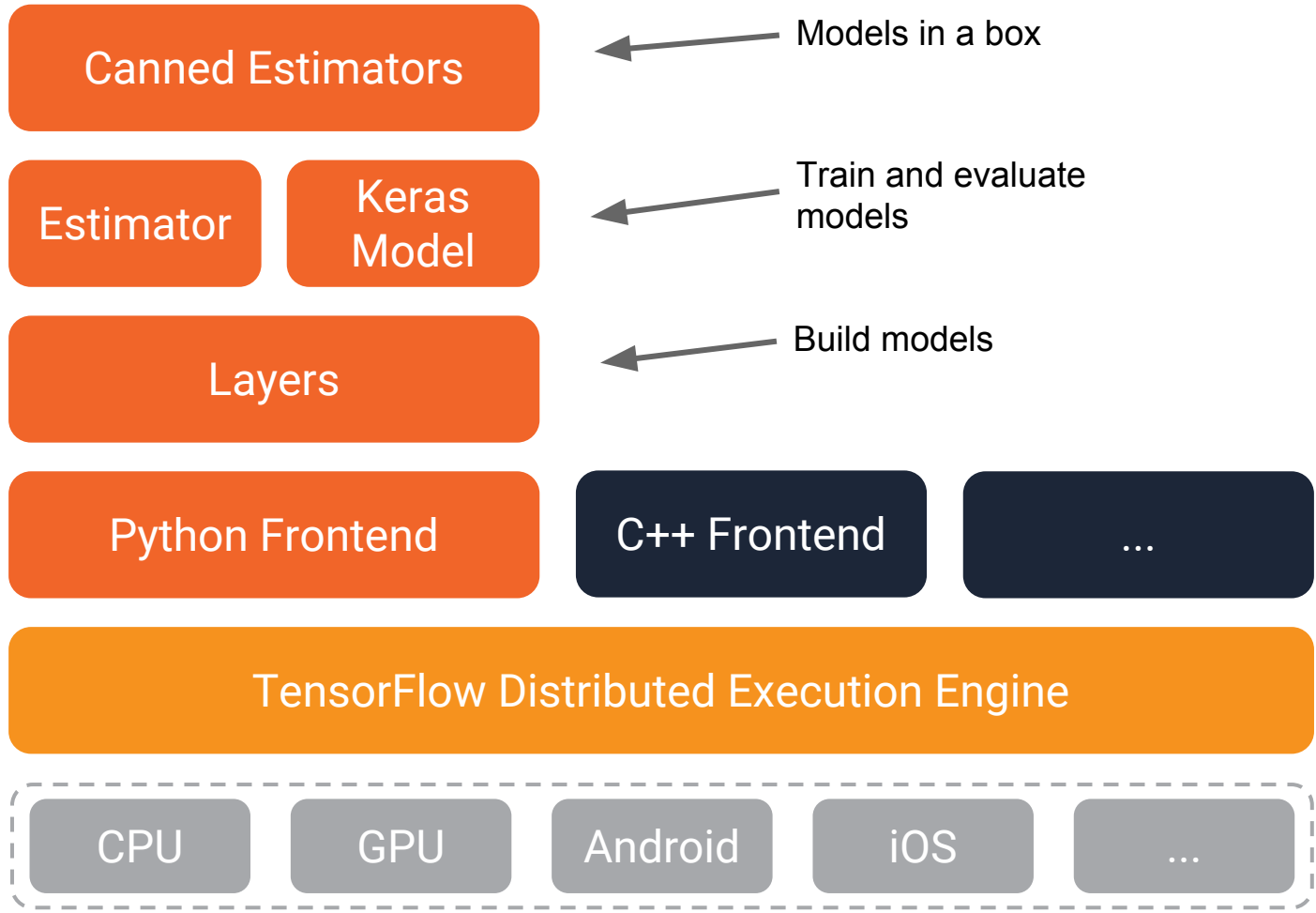
Some ops compute gradients

--= updates biases



And distributed





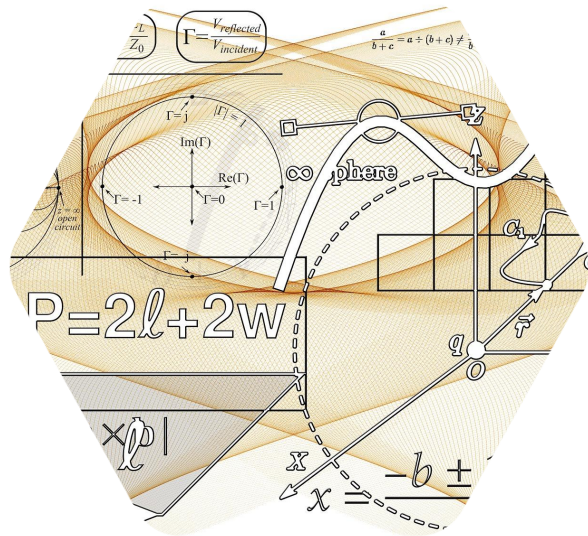
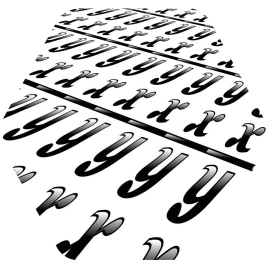
← Models in a box

← Train and evaluate models

← Build models



The popular imagination of what ML is



Lots of data

Complex mathematics in multidimensional spaces

Magical results

In reality, ML is



Define objectives



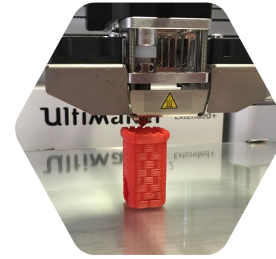
Collect data



Understand and prepare the data



Create the model

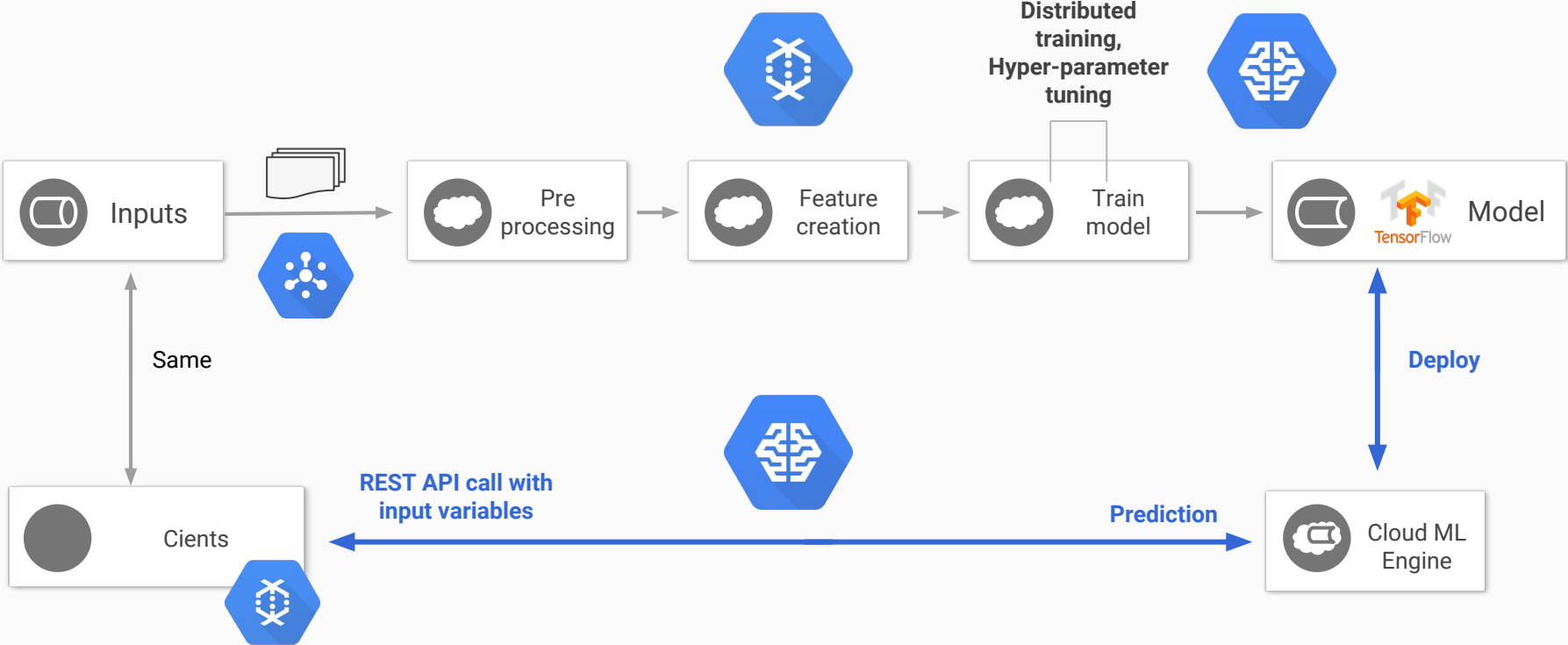


Refine the model



Serve the model

A useful and common design pattern



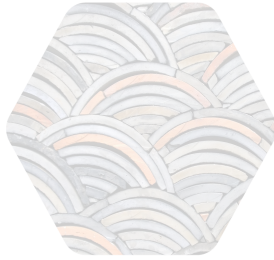
In reality, ML is



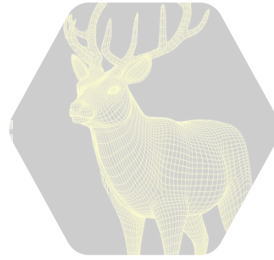
Define objectives



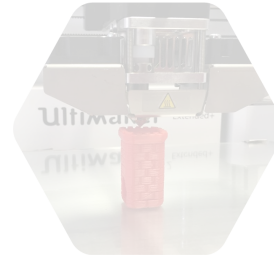
Collect data



Understand and prepare the data



Create the model



Refine the model

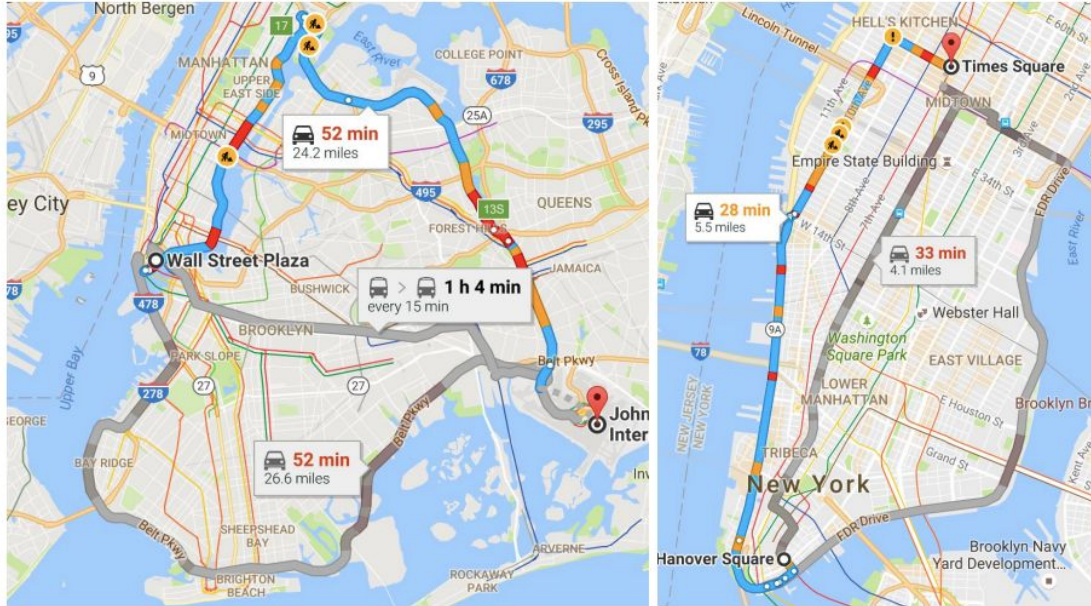


Serve the model



Estimate taxi fares

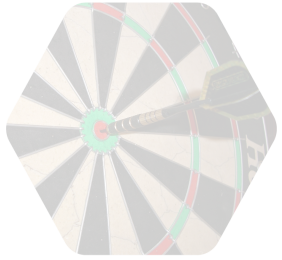
NYC Taxi & Limousine Commission



Goal is to **estimate the taxi fare** based on pickup and drop locations, as well as other trip information...

http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

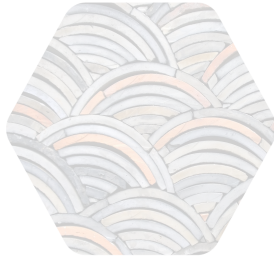
In reality, ML is



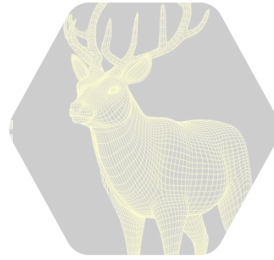
Define objectives



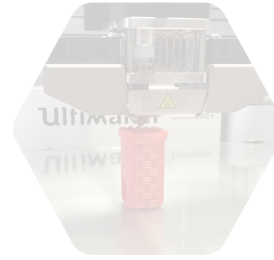
Collect data



Understand and prepare the data



Create the model



Refine the model



Serve the model

Serverless platform for all stages of the analytics data lifecycle

Ingestion



Cloud Pub/Sub



Cloud Storage

Processing



Cloud Dataflow

Analysis



BigQuery

No manual sharding
No capacity guessing
No idle resources
No maintenance window
No manual scaling
No file management

Introducing: self-service data preparation



Cloud Dataprep

- ✓ Visually explore and interact with data
- ✓ Instantly understand data distribution and patterns
- ✓ Quickly identify data quality issues
- ✓ Get automatic data transformation suggestions
- ✓ Standardize, structure and join datasets easily with a guided approach

Dataprep Example

Two datasets: Retail transactions & Ad impression logs

Goal: Find purchases that followed an ad for that product to that user

Retail purchase transactions

0-57	[{"salesOrderLineKey": "1", "productKey": "1110067200000-182", "dueDate": "2005-10-07T00:00:00Z", "shipDate": "2005-10-07T00:00:00Z"}]
0-588	[{"salesOrderLineKey": "1", "productKey": "1145836800000-296", "dueDate": "2006-09-13T00:00:00Z", "shipDate": "2006-09-13T00:00:00Z"}]
0-503	[{"salesOrderLineKey": "1", "productKey": "1289606400000-221", "dueDate": "2011-10-30T00:00:00Z", "shipDate": "2011-10-30T00:00:00Z"}]
0-623	[{"salesOrderLineKey": "1", "productKey": "1213660800000-460", "dueDate": "2008-09-11T00:00:00Z", "shipDate": "2008-09-11T00:00:00Z"}]

Ad impressions

4:00	https://theglobeandmail.com/sed/accumsan/felis/ut.json?pid=1157241600000-568
0:00	http://cam.ac.uk/neque/vestibulum/eget/vulputate.aspx?pid=1109721600000-237
8:00	https://odnoklassniki.ru/maecenas.jsp?pid=1281312000000-61
9:00	https://ft.com/hac.png?pid=1140393600000-91



Cloud Dataprep demo

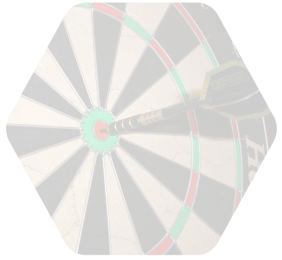
BigQuery: 100% serverless data warehouse



Google BigQuery

- ✓ Google Cloud's Enterprise Data Warehouse for Analytics
- ✓ Petabyte-Scale and Fast Convenience of SQL
- ✓ Encrypted, Durable and Highly Available
- ✓ Fully Managed and Serverless

In reality, ML is



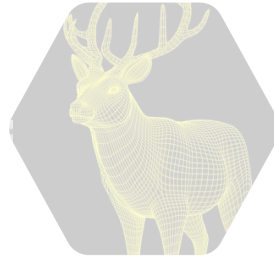
Define objectives



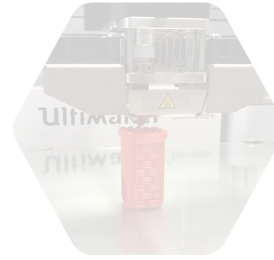
Collect data



Understand and prepare the data



Create the model



Refine the model



Serve the model

Powerful Data Exploration



Cloud Datalab

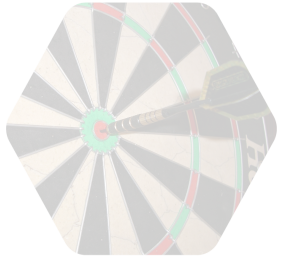
- ✓ Notebook interface
- ✓ Leverage existing Jupyter modules and knowledge
- ✓ Suitable to interactive data science and machine learning
- ✓ Closely integrated with BigQuery and Cloud ML



Understanding & Preparing data

- Describe dataset
- Explore and visualise the data
- Create training, validation, and test datasets
- Set a baseline

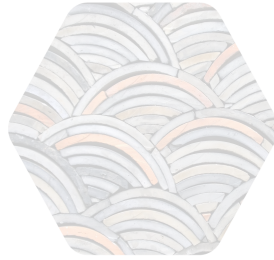
In reality, ML is



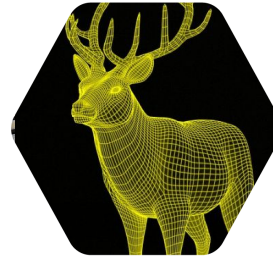
Define objectives



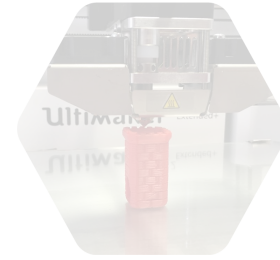
Collect data



Understand and prepare the data



Create the model



Refine the model



Serve the model

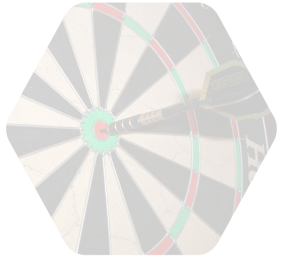


First experiments

- Basic features
- Small training set
- Simple Linear Regression model

I need coffee!

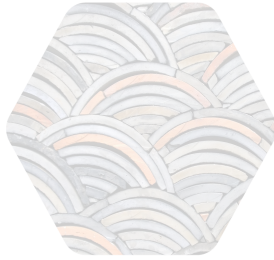
In reality, ML is



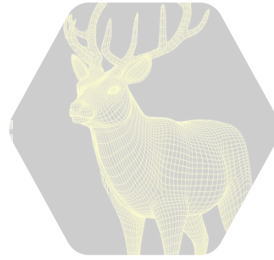
Define objectives



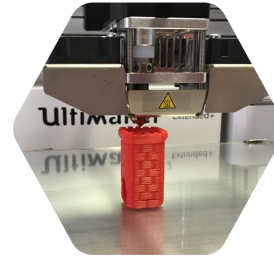
Collect data



Understand and prepare the data



Create the model



Refine the model



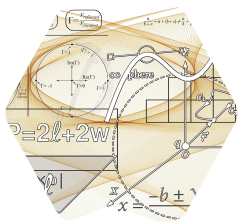
Serve the model

What do you think we can do to improve the model?

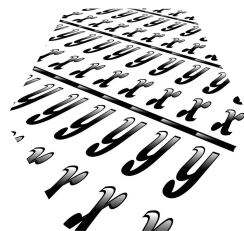
Refine the model



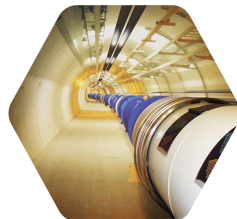
Feature engineering



Better algorithms



More examples, more data

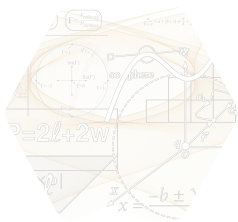


Hyperparameter tuning

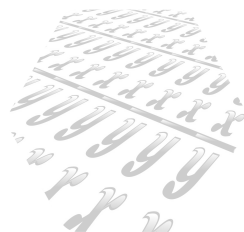
Refine the model



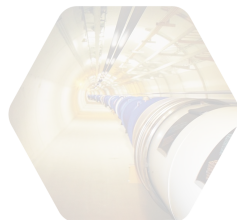
Feature engineering



Better algorithms



More examples, more data



Hyperparameter tuning

Feature engineering

Cleanse

Rows
Invalid data
Several missing features

Columns
(Near) zero variance
Many missing values
Many distinct values
(consider grouping or encoding)

Tune

Scaling
Handling missing values
Handling outliers
Handling skewed distribution

Transform

Discretization (binning)
Equal width
Equal Size
Clustering
Supervised

Encoding
Indicators (one-hot)
Learning with Counts
Embedding

Extract

(kernel-based) PCA
PLS (supervised)
Embedding
(Auto-encoders)
Hash Bucketization

Select

Filter methods
Wrapper methods

Construct

Polynomial expansion
Interactions (Crossing)
Multiplication, ratios
AND, OR, NOT

Feature comparison flags

Data-specific
Text
Image
Audio

Recency, Frequency,
Intensity (RFI) metric
Datetime elements
Distances with locations

Feature engineering

Cleanse

Rows
Invalid data
Several missing features

Columns
(Near) zero variance
Many missing values
Many distinct values
(consider grouping or encoding)

Tune

Scaling
Handling missing values
Handling outliers
Handling skewed distribution

Transform

Discretization (binning)
Equal width
Equal Size
Clustering
Supervised

Encoding
Indicators (one-hot)
Learning with Counts
Embedding

Extract

(kernel-based) PCA
PLS (supervised)
Embedding
(Auto-encoders)
Hash Bucketization

Select

Filter methods
Wrapper methods

Construct

Polynomial expansion
Interactions (Crossing)
Multiplication, ratios
AND, OR, NOT

Feature comparison flags

Data-specific
Text
Image
Audio

Recency, Frequency,
Intensity (RFI) metric
Datetime elements
Distances with locations

Feature engineering

Cleanse

Rows
Invalid data
Several missing features

Columns
(Near) zero variance
Many missing values
Many distinct values
(consider grouping or encoding)

Tune

Scaling
Handling missing values
Handling outliers
Handling skewed distribution

Transform

Discretization (binning)
Equal width
Equal Size
Clustering
Supervised

Encoding
Indicators (one-hot)
Learning with Counts
Embedding

Extract

(kernel-based) PCA
PLS (supervised)
Embedding
(Auto-encoders)
Hash Bucketization

Select

Filter methods
Wrapper methods

Construct

Polynomial expansion
Interactions (Crossing)
Multiplication, ratios
AND, OR, NOT

Feature comparison flags

Data-specific
Text
Image
Audio

Recency, Frequency,
Intensity (RFI) metric
Datetime elements
Distances with locations

Feature engineering

Cleanse

Rows

Invalid data
Several missing features

Columns

(Near) zero variance
Many missing values
Many distinct values
(consider grouping or encoding)

Tune

Scaling
Handling missing values
Handling outliers
Handling skewed distribution

Transform

Discretization (binning)

Equal width
Equal Size
Clustering
Supervised

Encoding

Indicators (one-hot)
Learning with Counts
Embedding

Extract

(kernel-based) PCA
PLS (supervised)
Embedding
(Auto-encoders)
Hash Bucketization

Select

Filter methods
Wrapper methods

Construct

Polynomial expansion

Interactions (Crossing)

Multiplication, ratios
AND, OR, NOT

Feature comparison flags

Data-specific

Text
Image
Audio

Recency, Frequency,
Intensity (RFI) metric
Datetime elements
Distances with locations

Feature engineering

Cleanse

Rows
Invalid data
Several missing features

Columns
(Near) zero variance
Many missing values
Many distinct values
(consider grouping or encoding)

Tune

Scaling
Handling missing values
Handling outliers
Handling skewed distribution

Transform

Discretization (binning)
Equal width
Equal Size
Clustering
Supervised

Encoding
Indicators (one-hot)
Learning with Counts
Embedding

Extract

(kernel-based) PCA
PLS (supervised)
Embedding
(Auto-encoders)
Hash Bucketization

Select

Filter methods
Wrapper methods

Construct

Polynomial expansion
Interactions (Crossing)
Multiplication, ratios
AND, OR, NOT

Feature comparison flags

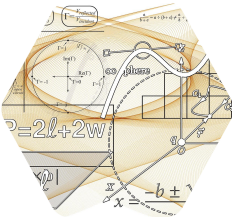
Data-specific
Text
Image
Audio

Recency, Frequency,
Intensity (RFI) metric
Datetime elements
Distances with locations

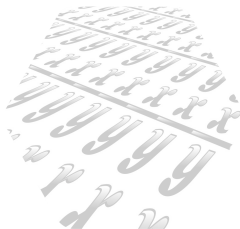
Refine the model



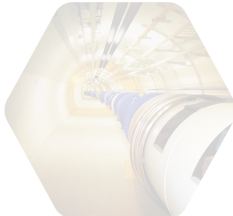
Feature engineering



Better algorithms



More examples, more data

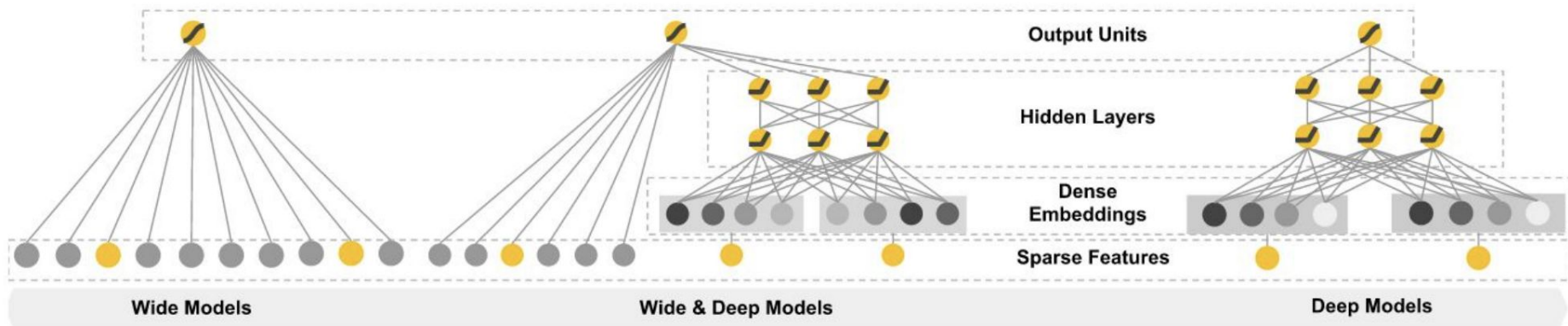


Hyperparameter tuning

Better algorithms: DNN

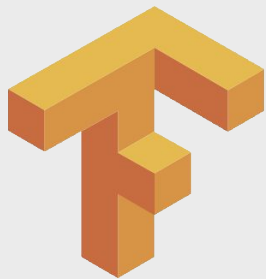
- ✓ **Consider different optimisation algorithms:** Adam, Adagrade, derivative-free methods, etc.
- ✓ **Consider different activation functions:** relu, radial basis functions (RDF), etc.
- ✓ **Consider different model architectures:** number of layers and nodes per layer, connectivity, etc.
- ✓ **Consider different loss metric:** quadratic loss, cross entropy, bayesian information reward, etc.
- ✓ **Consider handling overfitting:** regularisation, early-stopping, dropouts, etc.
- ✓ **Consider Linear Combined DNN:** for deep (dense) and wide (sparse) features

Linear combined DNN



<https://research.googleblog.com/2016/06/wide-deep-learning-better-together-with.html>

When do you think neural networks are not the best option for ML?



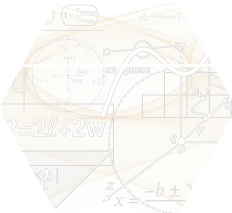
Second experiment

- Feature engineering
 - Datetime info
 - Bucketization
 - Crossed features
- Better algorithms
 - Linear Combined DNN
 - Feature embedding

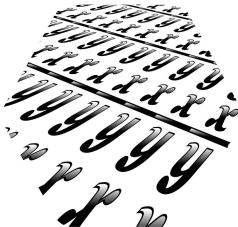
Refine the model



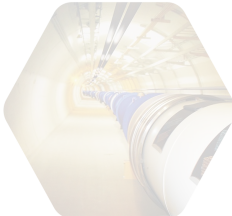
Feature engineering



Better algorithms



More examples, more data



Hyperparameter tuning

“ It's not who has the best algorithm who wins, it's who has the most data.”

— Andrew Ng , Co-Founder of Google Brain



Fully-managed data processing service



Cloud Dataflow

- ✓ Combines batch and streaming models
- ✓ Autoscaling and dynamic task sharding
- ✓ Connectors to Bigtable, BigQuery, Cloud Storage
- ✓ Portable via open-source Apache Beam SDK

Apache Beam & Cloud Dataflow

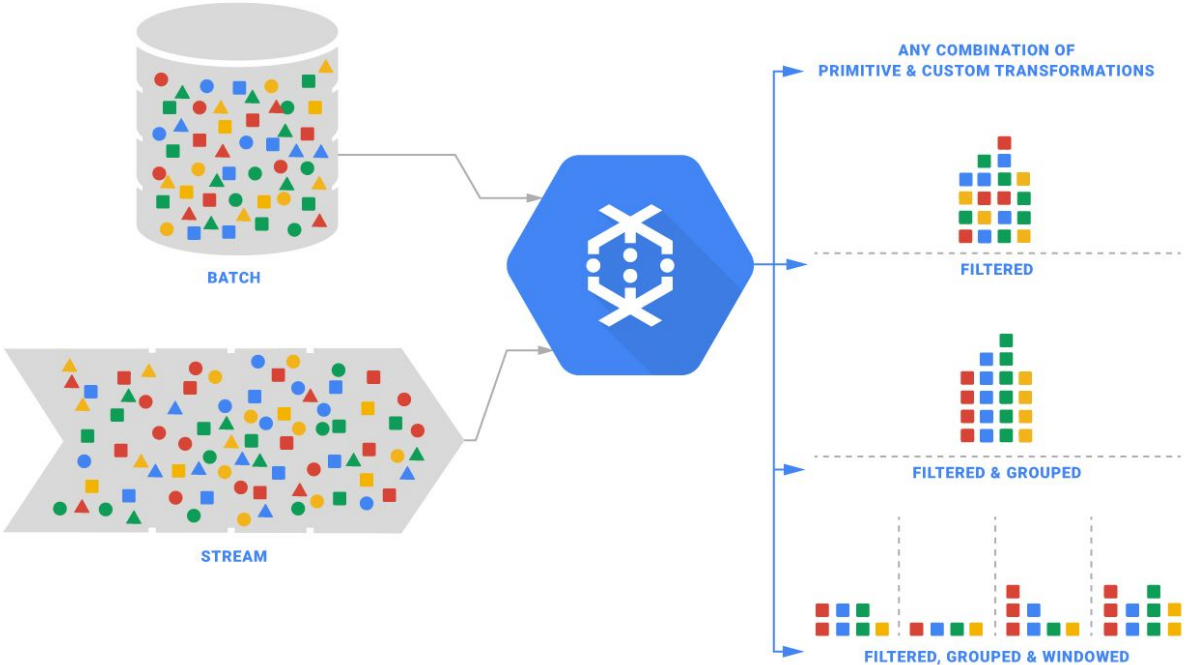


Apache Beam is a collection of SDKs for **building** streaming data processing pipelines.



Cloud Dataflow is a fully managed (no-ops) and integrated service for **executing** optimized parallelized data processing pipelines.

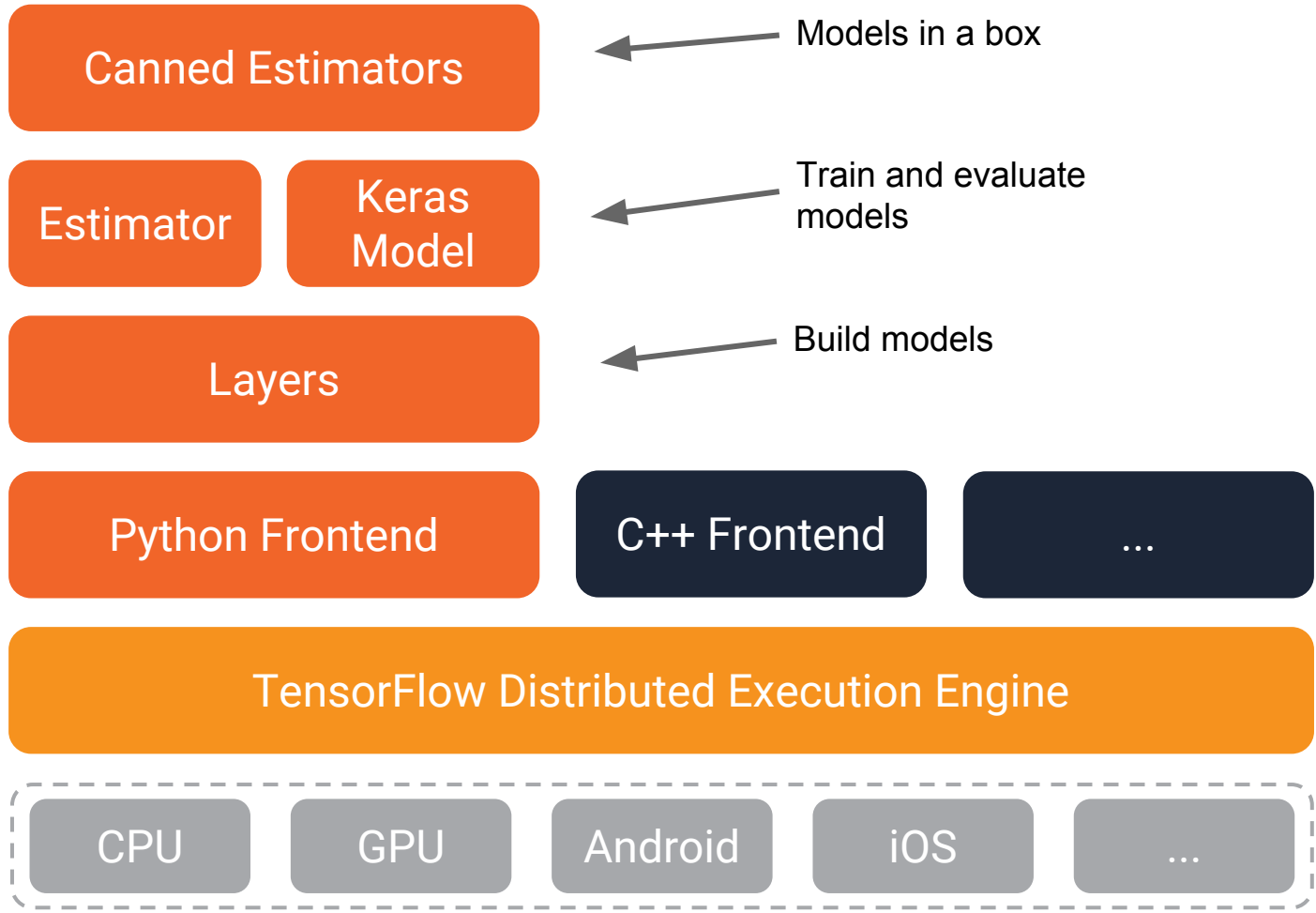
Unified batch and stream processing





Preparing big data for training

- Dataflow to export data from BigQuery to GCS



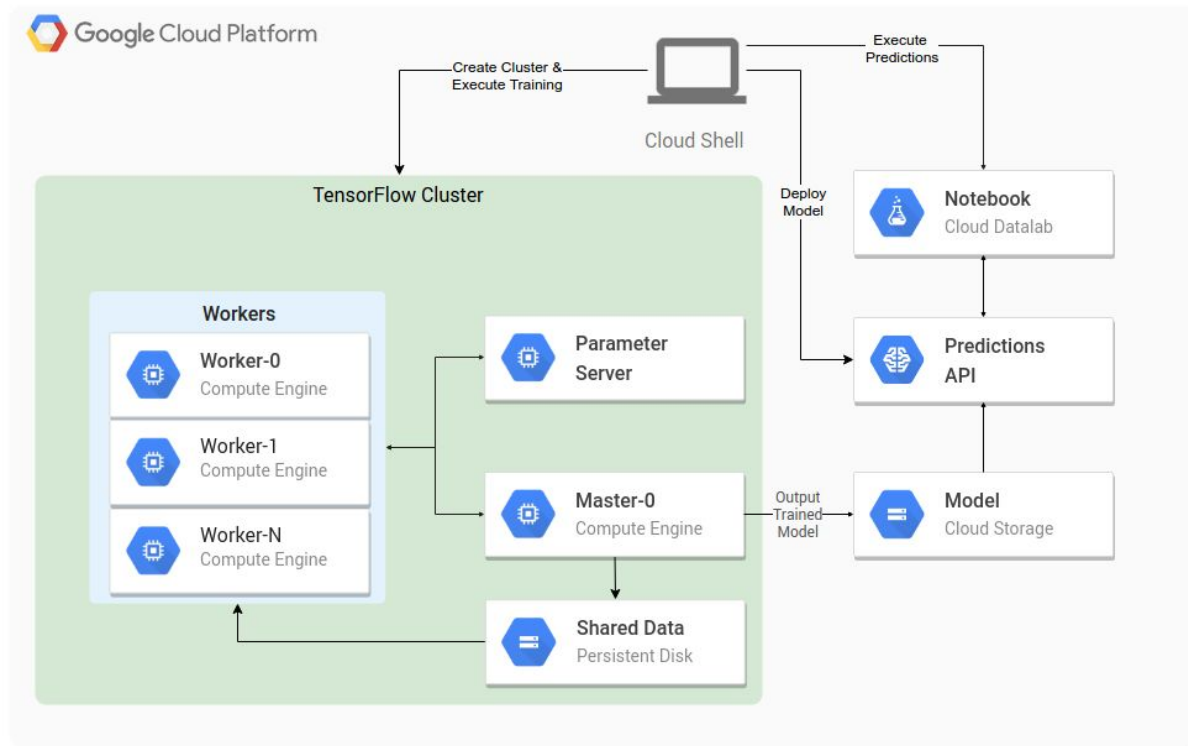
← Models in a box

← Train and evaluate models

← Build models



Distributed TensorFlow on Compute Engine



<https://cloud.google.com/solutions/running-distributed-tensorflow-on-compute-engine>

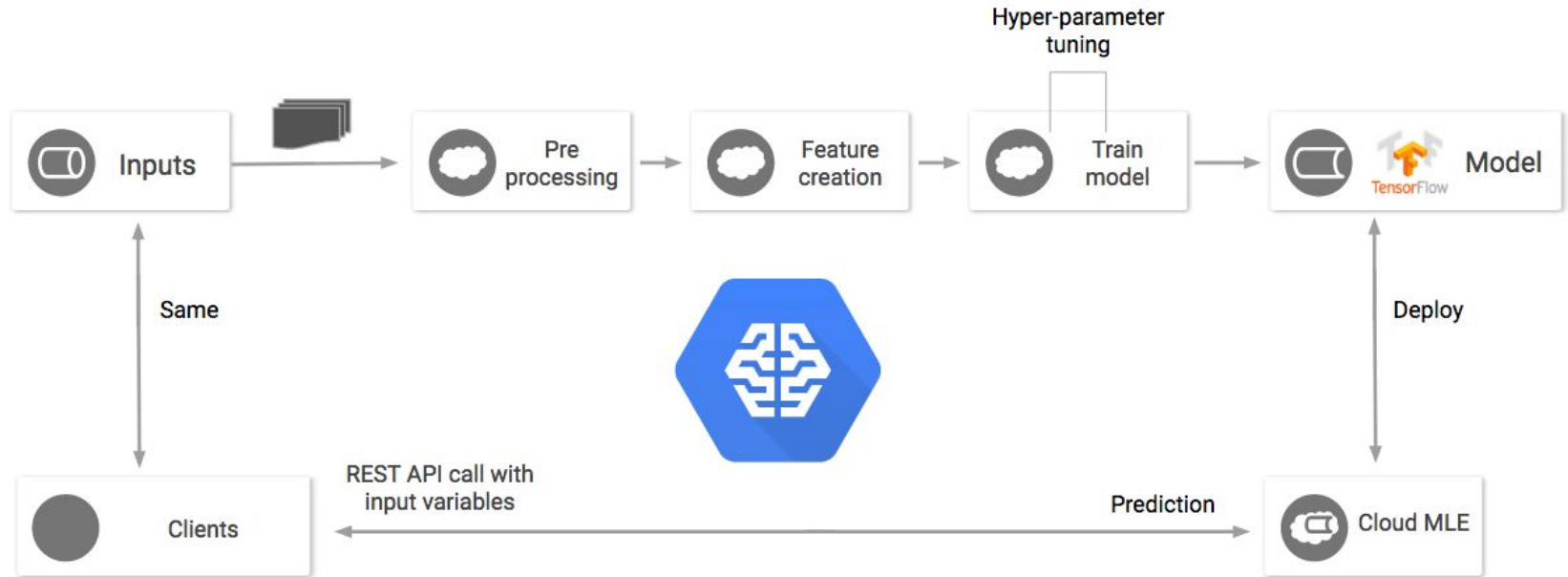
Machine Learning on any data, of any size



Cloud ML Engine

- ✓ Services are designed to work together
- ✓ Managed distributed training infrastructure that supports CPUs and GPUs
- ✓ Automatic hyperparameter tuning
- ✓ Portable models with TensorFlow

Distributed TensorFlow with Cloud ML Engine and Cloud Datalab



<https://cloud.google.com/ml-engine/docs/tutorials/distributed-tensorflow-mnist-cloud-datalab>

Running locally



train locally

```
gcloud ml-engine local train \
```

```
--module-name trainer.task --package-path trainer/ \
```

```
-- \
```

```
--train-files $TRAIN_DATA --eval-files $EVAL_DATA --train-steps 1000 --job-dir $MODEL_DIR
```

output directory

training data

evaluation data



Single trainer running in the cloud



train in the cloud

region

```
gcloud ml-engine jobs submit training $JOB_NAME --job-dir $OUTPUT_PATH \  
  --runtime-version 1.0 --module-name trainer.task --package-path trainer/ --region $REGION \  
  -- \  
  --train-files $TRAIN_DATA --eval-files $EVAL_DATA --train-steps 1000 --verbosity DEBUG
```

*Google cloud storage
location*



Distributed training in the cloud



```
gcloud ml-engine jobs submit training $JOB_NAME --job-dir $OUTPUT_PATH \  
  --runtime-version 1.0 --module-name trainer.task --package-path trainer/ --region $REGION \  
  --scale-tier STANDARD_1 distributed \  
  \  
  --train-files $TRAIN_DATA --eval-files $EVAL_DATA --train-steps 1000 --verbosity DEBUG
```



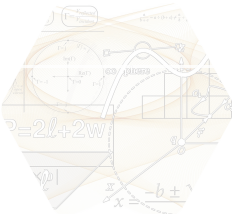


Training with Cloud ML Engine

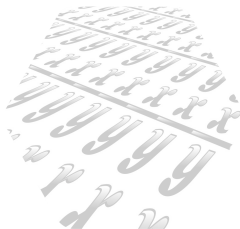
Refine the model



Feature engineering



Better algorithms

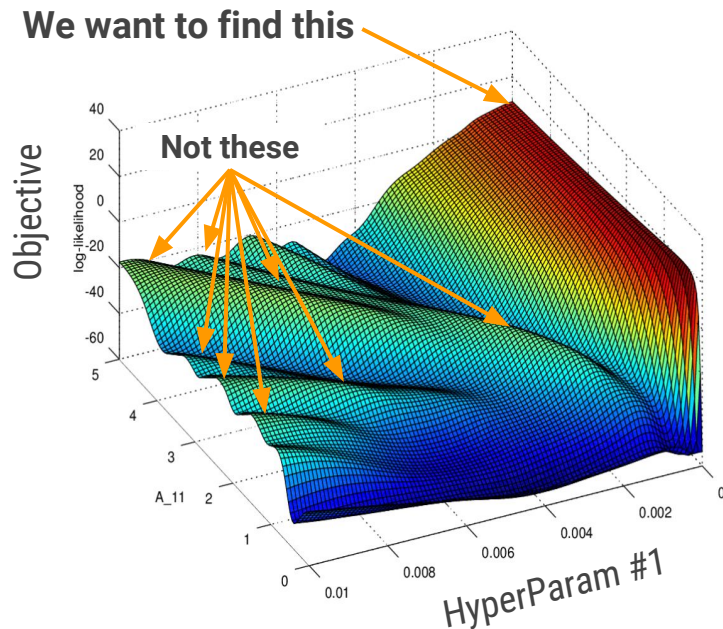


More examples, more data



Hyperparameter tuning

Hyperparameter tuning



- Automatic hyperparameter tuning service
- Build better performing models faster and save many hours of manual tuning
- Google-developed search (Bayesian Optimisation) algorithm efficiently finds better hyperparameters for your model/dataset

<https://cloud.google.com/blog/big-data/2017/08/hyperparameter-tuning-in-cloud-machine-learning-engine-using-bayesian-optimization>

Hyperparameter tuning



```
gcloud ml-engine jobs submit training $JOB_NAME --job-dir $OUTPUT_PATH \  
  --runtime-version 1.0 --module-name trainer.task --package-path trainer/ --region $REGION \  
  --scale-tier STANDARD_1 --config $HPTUNING_CONFIG hypertuning \  
  \  
  --train-files $TRAIN_DATA --eval-files $EVAL_DATA --train-steps 1000 --verbosity DEBUG
```



Hyperparameter tuning



hptuning_config.yaml

```
trainingInput:
  hyperparameters:
    goal: MAXIMIZE
    hyperparameterMetricTag: accuracy
    maxTrials: 4
    maxParallelTrials: 2
    params:
      - parameterName: first-layer-size
        type: INTEGER
        minValue: 50
        maxValue: 500
        scaleType: UNIT_LINEAR_SCALE
  ...
```

task.py

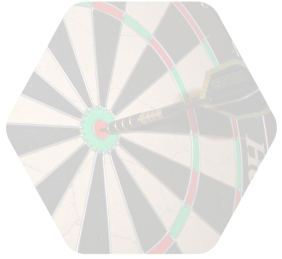
```
...
# Construct layers sizes with exponential decay
hidden_units=[
    max(2, int(hparams.first_layer_size *
               hparams.scale_factor**i))
    for i in range(hparams.num_layers)
],
...
parser.add_argument(
    '--first-layer-size',
    help='Number of nodes in the 1st layer of the DNN',
    default=100,
    type=int
)
...
```





Results comparison

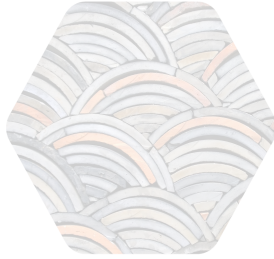
In reality, ML is



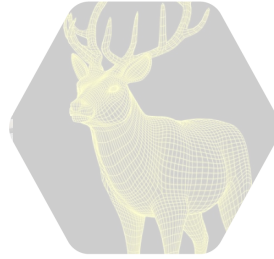
Define
objectives



Collect
data



Understand
and prepare
the data



Create the
model



Refine the
model



Serve the
model

Deploying the model



Creating model

```
gcloud ml-engine models create $MODEL_NAME --regions=$REGION
```

Creating versions

```
gcloud ml-engine versions create v1 --model $MODEL_NAME --origin $MODEL_BINARIES \  
  --runtime-version 1.0
```

```
gcloud ml-engine models list
```



Predicting



```
gcloud ml-engine predict --model $MODEL_NAME --version v1 --json-instances ../test.json
```

Using REST:

```
POST https://ml.googleapis.com/v1/{name=projects/\*\*}:predict
```

JSON format (in this case):

```
{"age": 25, "workclass": "private", "education": "11th", "education_num": 7, "marital_status":  
"Never-married", "occupation": "machine-op-inspector", "relationship": "own-child", "gender": "  
male", "capital_gain": 0, "capital_loss": 0, "hours_per_week": 40, "native_country": "  
United-States"}
```





Predicting taxi fares



TensorFlow and GPUs

Create a VM with a GPU

Using the gcloud command line tool:

```
gcloud beta compute instances create vm-gpu-tf \  
  --machine-type n1-standard-2 --zone europe-west1-b --accelerator type=nvidia-tesla-k80,count=1 \  
  --image-family ubuntu-1604-lts --image-project ubuntu-os-cloud --maintenance-policy TERMINATE \  
  --restart-on-failure
```


Installing CUDA

SSH into the machine and run the following script:

```
#!/bin/bash
echo "Checking for CUDA and installing."
# Check for CUDA and try to install.
if ! dpkg-query -W cuda; then
    # The 16.04 installer works with 16.10.
    curl -O http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/cuda-repo-ubuntu1604_8.0.61-1_amd64.deb
    dpkg -i ./cuda-repo-ubuntu1604_8.0.61-1_amd64.deb
    apt-get update
    apt-get install cuda -y
fi
```

Installing CUDA

Verify and set CUDA environment variables

```
nvidia-smi
```

```
echo 'export CUDA_HOME=/usr/local/cuda' >> ~/.bashrc
```

```
echo 'export PATH=$PATH:$CUDA_HOME/bin' >> ~/.bashrc
```

```
echo 'export LD_LIBRARY_PATH=$CUDA_HOME/lib64' >> ~/.bashrc
```

```
source ~/.bashrc
```

Installing cuDNN

Download and install:

```
tar xzvf cudnn-8.0-linux-x64-v6.0.tgz
```

```
sudo cp cuda/lib64/* /usr/local/cuda/lib64/
```

```
sudo cp cuda/include/cudnn.h /usr/local/cuda/include/
```

```
rm -rf ~/cuda
```

```
rm cudnn-8.0-linux-x64-v6.0.tgz
```

<https://developer.nvidia.com/developer-program>

Installing Tensorflow

tensorflow-gpu ensures that the GPU used for the operations where applicable

```
sudo apt-get install python-dev python-pip libcupti-dev
```

```
sudo pip install tensorflow-gpu
```

Testing

tensorflow-gpu ensures that the GPU used for the operations where applicable

```
import tensorflow as tf

with tf.device('/gpu:0'):
    a_g = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a-gpu')
    b_g = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b-gpu')
    c_g = tf.matmul(a_g, b_g, name='c-gpu')

with tf.Session(config=tf.ConfigProto(log_device_placement=True)) as sess:
    print (sess.run(c_g))
```

<https://medium.com/google-cloud/using-a-gpu-tensorflow-on-google-cloud-platform-1a2458f42b0>

Further reading and useful links

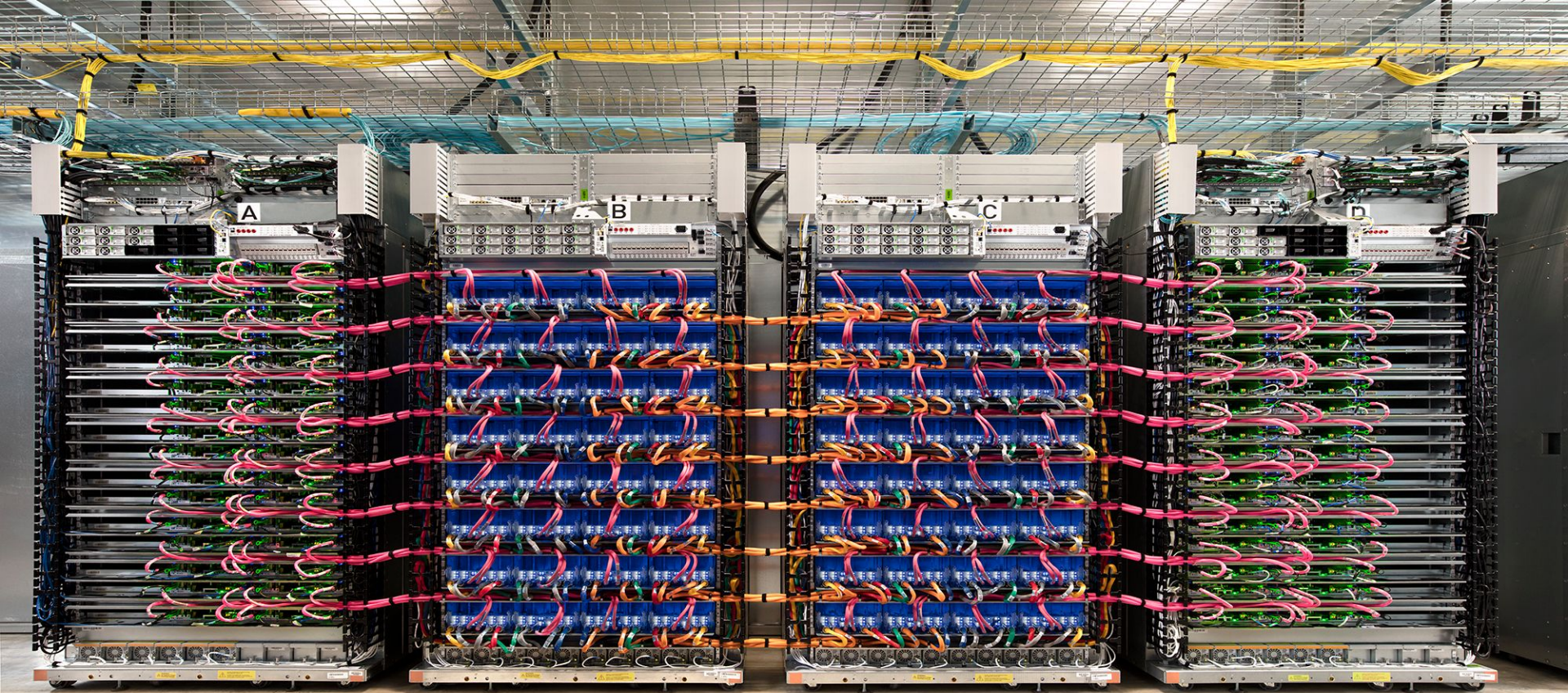
- <https://cloud.google.com/solutions/running-distributed-tensorflow-on-compute-engine>
- <https://cloud.google.com/ml-engine/docs/tutorials/distributed-tensorflow-mnist-cloud-datalab>
- <https://codelabs.developers.google.com/?cat=TensorFlow>
- <https://codelabs.developers.google.com/codelabs/cloud-ml-engine-sd-regression>
- <https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/>
- <https://medium.com/google-cloud/using-a-gpu-tensorflow-on-google-cloud-platform-1a2458f42b0>
- <https://medium.com/google-cloud/running-jupyter-notebooks-on-gpu-on-google-cloud-d44f57d22dbd>
- https://www.tensorflow.org/install/install_linux
- <https://research.googleblog.com/2016/06/wide-deep-learning-better-together-with.html>
- <https://cloud.google.com/genomics/> https://youtu.be/ExNxi_X4qug
- <https://www.kaggle.com/>

2nd Generation Tensor Processing Unit (TPU) Custom ASIC built and optimized for TensorFlow



Used in production at Google for over 16 months

180 Teraflops!! & 15-30x faster than CPU or GPU



A "TPU pod" built with 64 second-generation TPUs delivers up to 11.5 petaflops of machine learning acceleration.

Thank you!

Survey and follow up request:
<https://goo.gl/Rtkn8i>