RAPIDS: Accelerated Data Science and Model Inference

Roman Enzmann, Solutions Architect at NVIDIA

Agenda

- Why RAPIDS in Data Science
- How is speedup achieved?
- Success Stories
- Data Loading & ETL (cuDF and Spark RAPIDS)
- Machine Learning (cuML)
- Graph Analytics (cuGraph)
- Triton FIL
- Fraud Detection With RAPIDS
- How to Get Started

Data Center Traffic Continues to Expand

Cyberattacks Increased by 72% from 2021 to 2023[1]

Explosion of Data

16B
Connected Things (IoT)[2]

5B+
Internet Users[3]

291ZB
of data generated by 2027[4]

Consequences of Cyberattacks

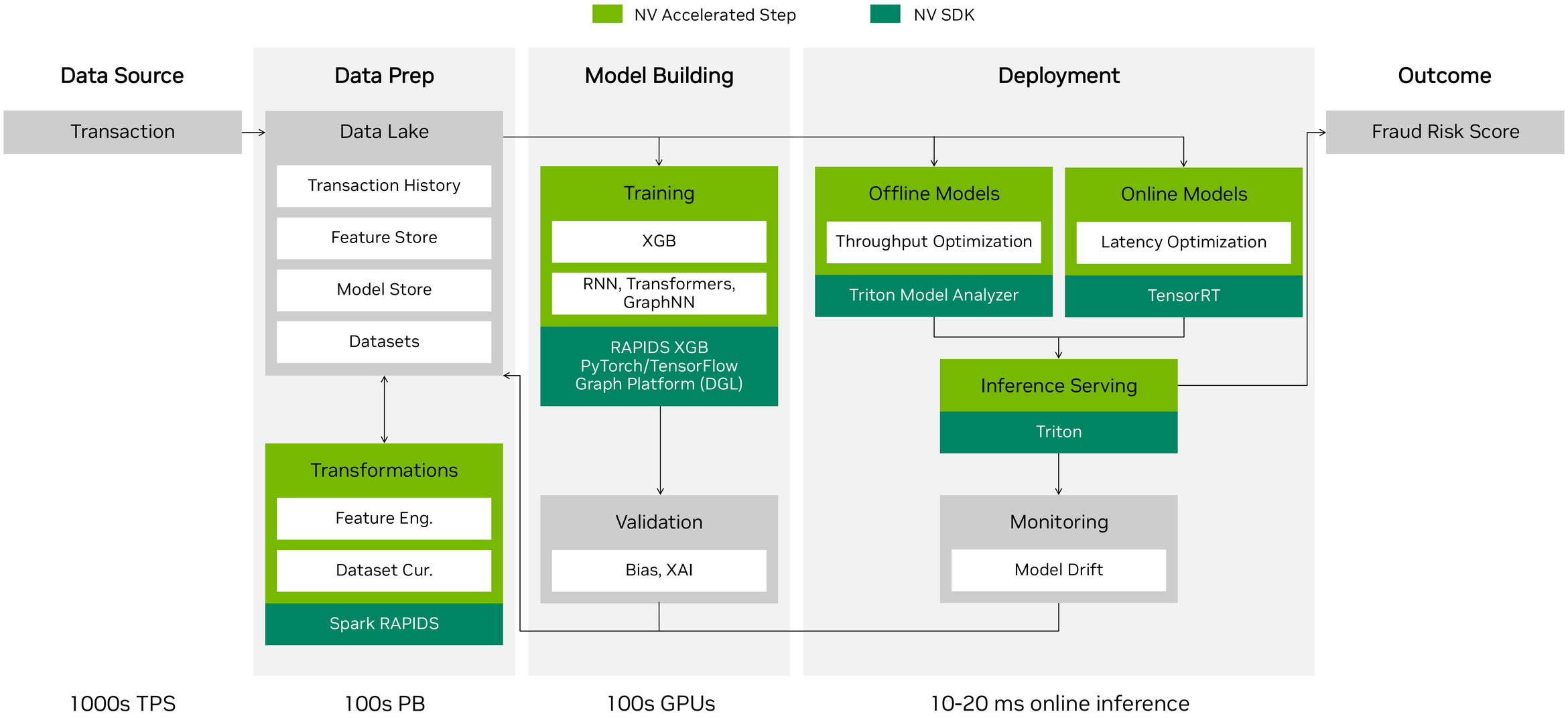
$4.45M
Average cost of a breach[5]

277
days on average to identify and contain a breach[5]

343M
Victims of 2,365 cyberattacks in 2023[6]

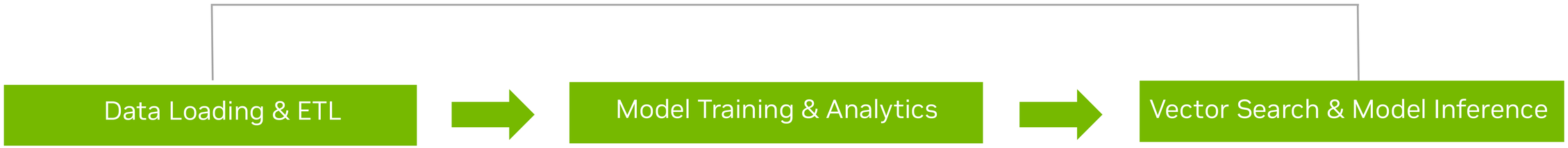
1. Forbes | 2. IoT Analytics, 2023 | 3. Digital 2022 April Global Statshot Report | 4. IDC, Revelations in the Global DataSphere, August 2023 | 5. IBM Security 2023 | 6. Forbes

Transaction Fraud Detection Workflow Example



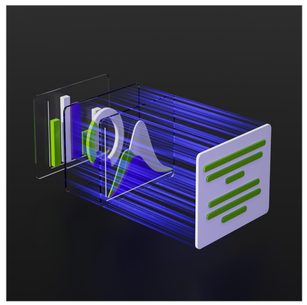
Modelling Process

Simple three step process at face value

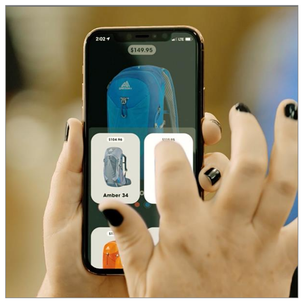


Modern Enterprise Applications Need Accelerated Computing

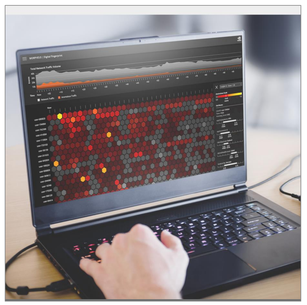
Internet scale data | Massive models | Real-time performance



LLMs



Recommenders

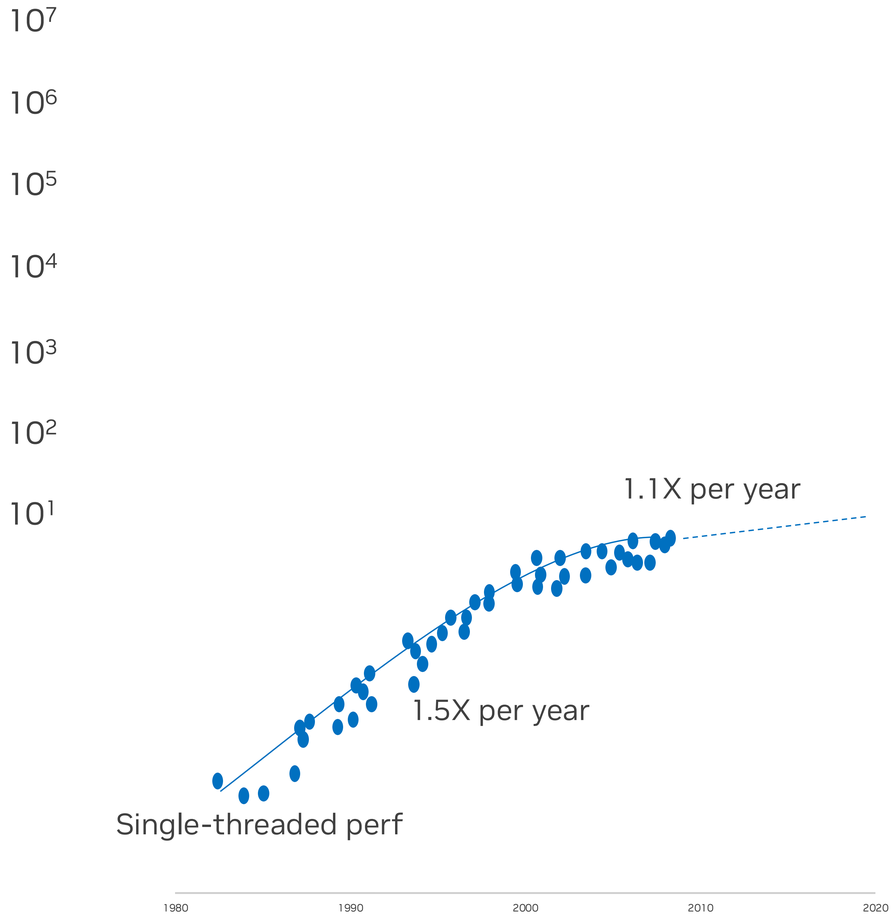


Cybersecurity

Fraud detection

Moore’s Law

Neural scaling law

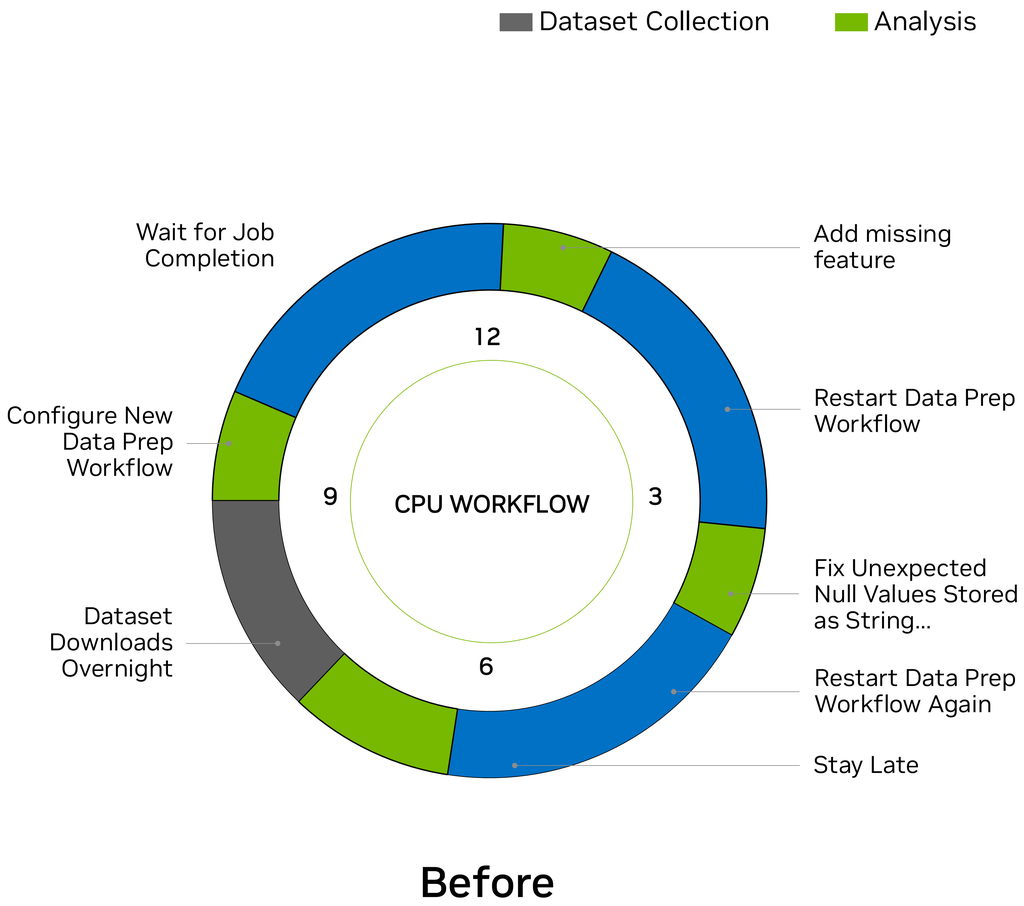


Drivers:

1. Data availability
2. Model complexity
3. Transaction speed

Accelerated Performance

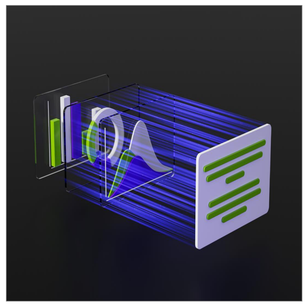
Accelerated Computing Empowers Modern Data Science Teams



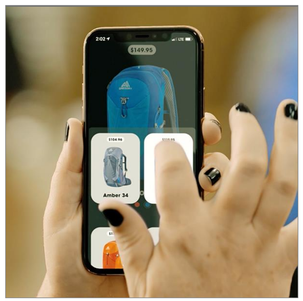
Before

Modern Enterprise Applications Need Accelerated Computing

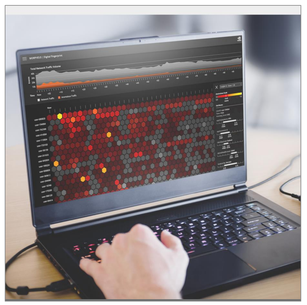
Internet scale data | Massive models | Real-time performance



LLMs



Recommenders

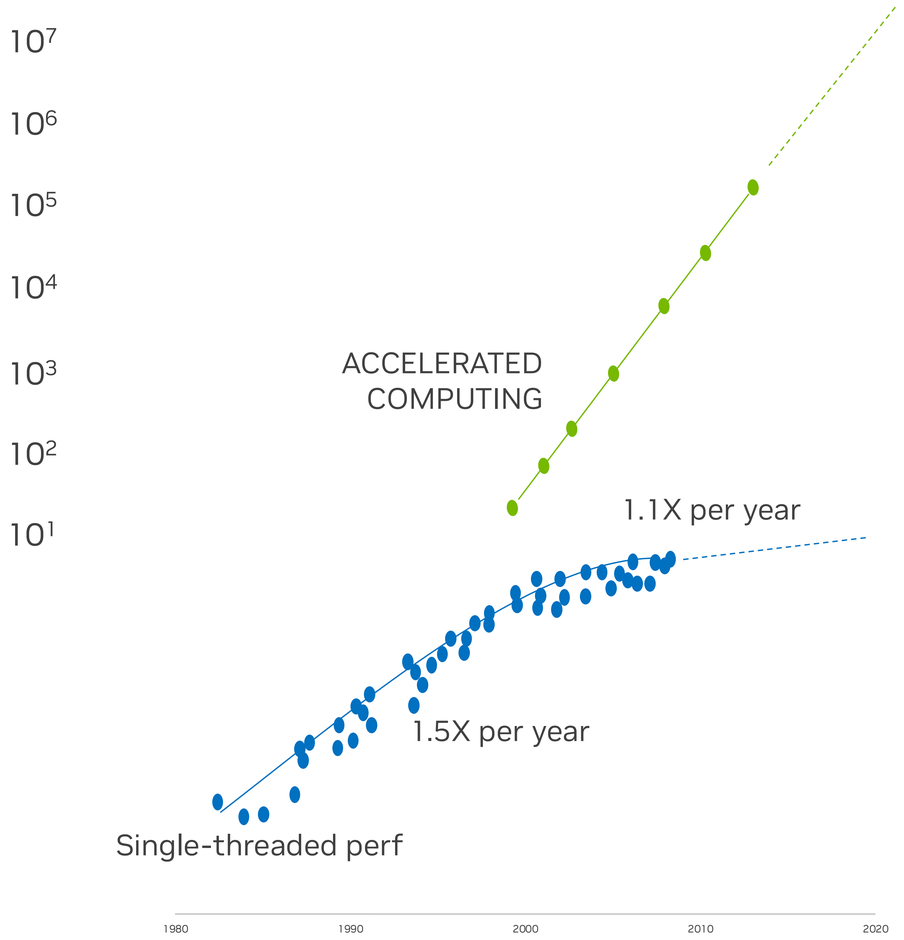


Cybersecurity

Fraud detection

Moore’s Law

Neural scaling law



Drivers:

1. Data availability
2. Model complexity
3. Transaction speed

Accelerated Performance

Accelerated Computing Empowers Modern Data Science Teams

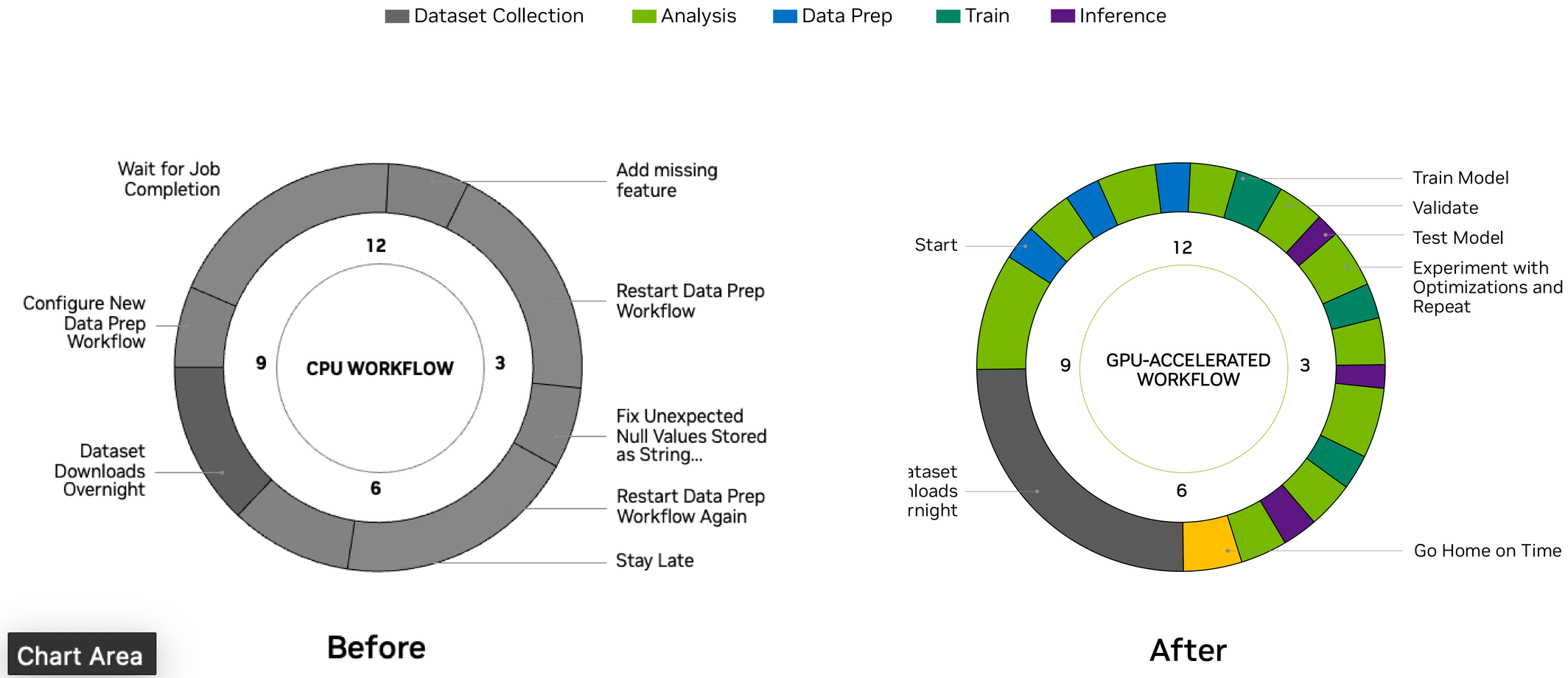


Chart Area

How RAPIDS Fits into Data Science

NVIDIA RAPIDS

Improve Existing Data Processing Workflows

Faster Results

Accelerate data replication and AI pipelines

Lower Costs

Save on infrastructure

Reduce power consumption and carbon footprint

Quick Time-to-Value

No code changes required

RAPIDS Accelerates Data Science End-to-End

Data Loading & ETL

Model Training & Analytics

Vector Search & Model Inference

RAPIDS

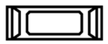
APACHE Spark™

RAPIDS for Apache Spark
Apache Spark Accelerator

NVIDIA AI Enterprise
Development Tools | Cloud Native Management and Orchestration | Infrastructure Optimization


Cloud


Data Center


Edge


RTX Laptop

RAPIDS Accelerates Data Science End-to-End

Data Loading & ETL

RAPIDS cuDF
Tabular Data Loading and Processing

RAPIDS for Apache Spark
Apache Spark Accelerator

RAPIDS Dask
Dask Accelerator

Model Training & Analytics

RAPIDS cuGraph
Large Scale Graph Analytics

RAPIDS cuML
Machine Learning Toolkit

Deep Learning
cuGraph-powered GNNs

RAPIDS for Apache Spark
Apache Spark ML Accelerator

Vector Search & Model Inference

RAPIDS cuVS
Vector Search & ML

Triton FIL
Optimized tree model inference

NVIDIA AI Enterprise

Development Tools | Cloud Native Management and Orchestration | Infrastructure Optimization

Cloud

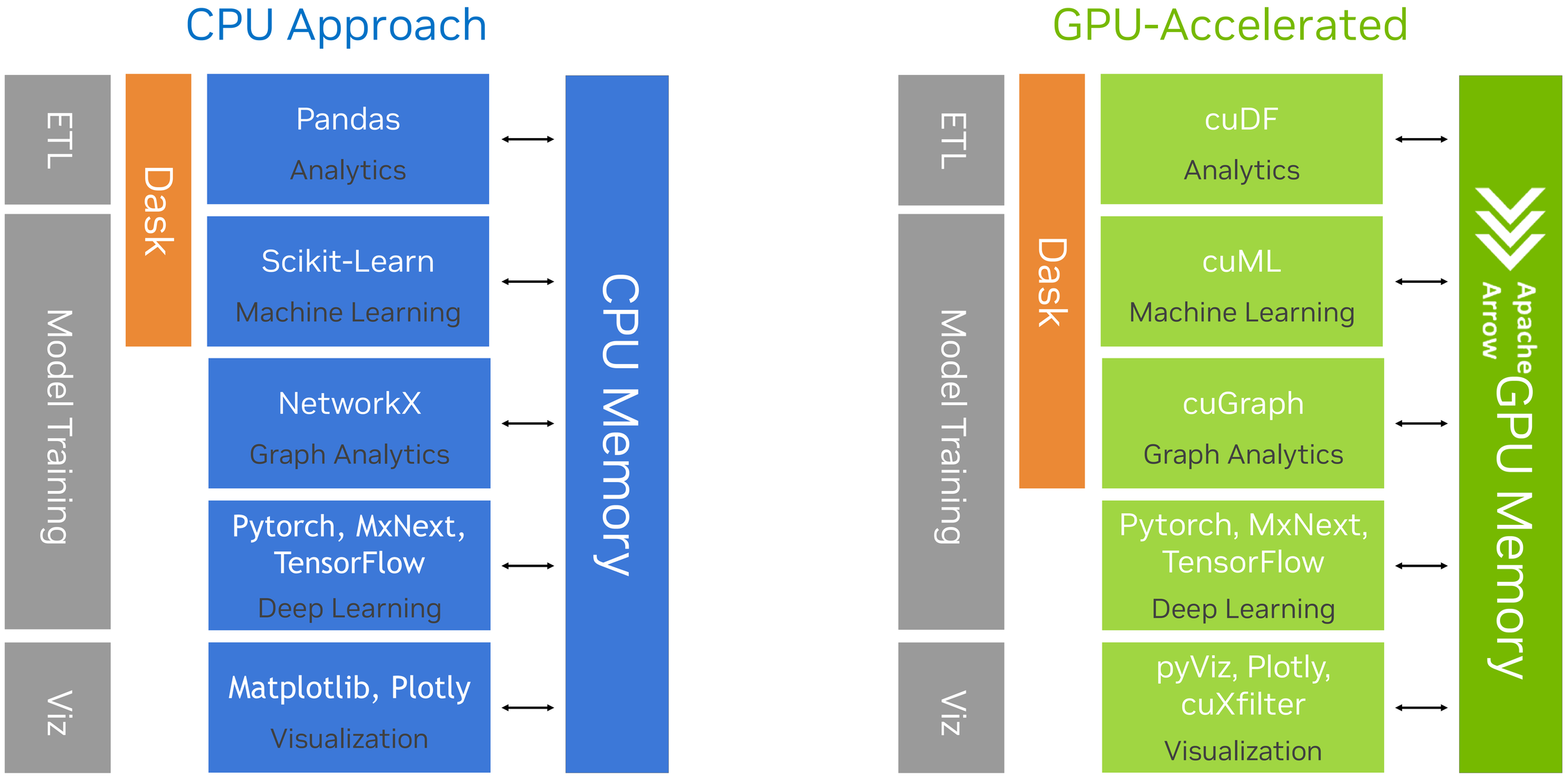
Data Center

Edge

RTX Laptop

Side by Side Comparison

Same Familiar Python APIs – No/Low Code Change



Minor Code Changes for Major Benefits

Abstracting Accelerated Compute through Familiar Interfaces

CPU

scikit-learn

```
from sklearn.cluster import DBSCAN

clustering = DBSCAN(eps=3.1).fit(x)
```

XGBoost

```
from xgboost import XGBRegressor

XGBRegressor(tree_method="hist",
                    device="cpu")
```

NetworkX

```
import networkx as nx

pr = nx.pagerank(graph)
```

Spark

```
spark.sql("""
    select order,
        count(*) as order_count
    from orders""")
```

GPU

cuML

```
from cuml.cluster import DBSCAN

clustering = DBSCAN(eps=3.1).fit(x)
```

Average Speed-Ups: 50x

XGBoost

```
from xgboost import XGBRegressor

XGBRegressor(tree_method="hist",
                    device="cuda")
```

Average Speed-Ups: 10x

cuGraph

```
import cugraph

pr = cugraph.pagerank(graph)
```

Average Speed-Ups: 250x

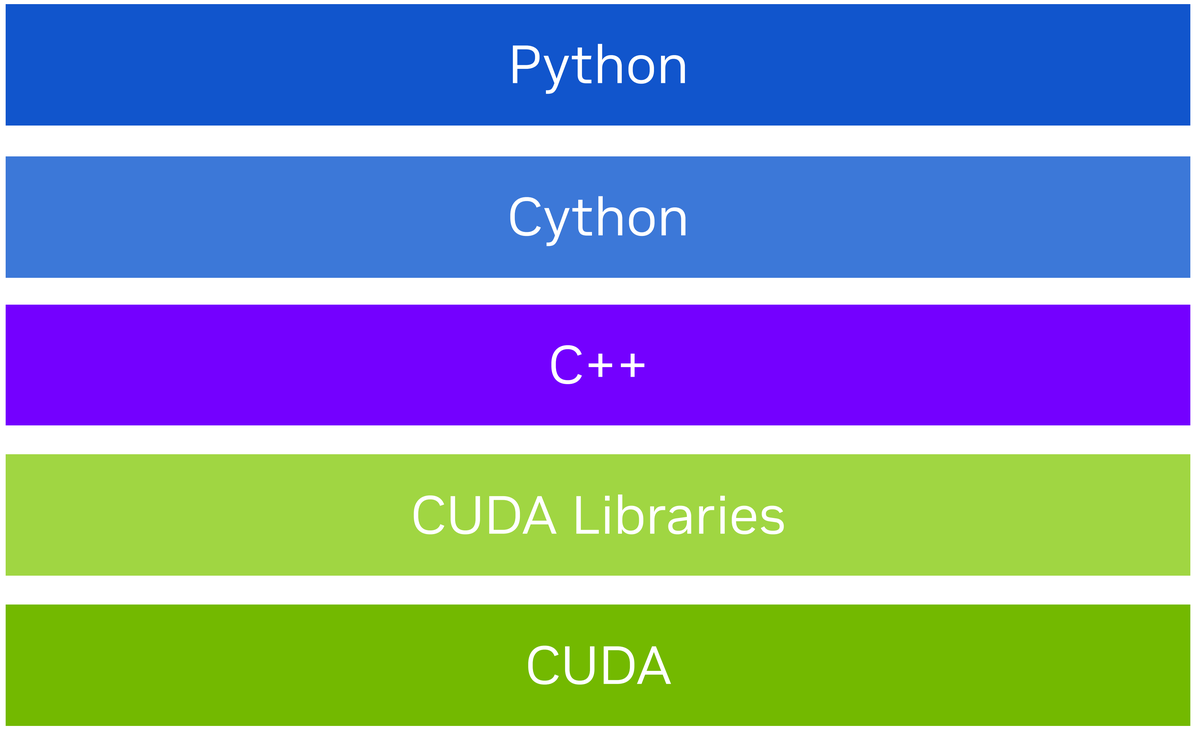
Spark

```
spark.conf.set("spark.plugins","com.nvidia.spark.SQLPlugin")
spark.sql("""
    select order,
        count(*) as order_count
    from orders""")
```

Average Speed-Ups: 10x

Technology Stack

RAPIDS



Accelerated Computing Swim Lanes

Making accelerated computing more seamless while enabling deep customization for maximum performance

Easier to Use

Maximum Performance

Zero Code Change: Acceleration Plugins *(no-code change)*

cudf.pandas: Accelerated Pandas, Accelerated NetworkX nx-cugraph, RAPIDS Spark Accelerator, Array-API backed Scikit-learn...

Hybrid CPU/GPU libraries *(minimal change)*

Pytorch, FAISS, Tensoflow, XGBoost, cuML-CPU ...

Dask, pySpark...

GPU Python Libraries *(GPU Python code)*

RAPIDS core libraries (cuDF, cuML, cuGraph, cuVS), RMM, CuPy, Numba, OpenAI Triton ...

Python/CUDA libraries *(Hybrid Python / CUDA code)*

CuPy RawKernels, Numba CUDA, Cython wrappers for CUDA ...

C++/CUDA high level *(High-level C++/CUDA code)*

RAFT, CCCL: Thrust, CUB ...

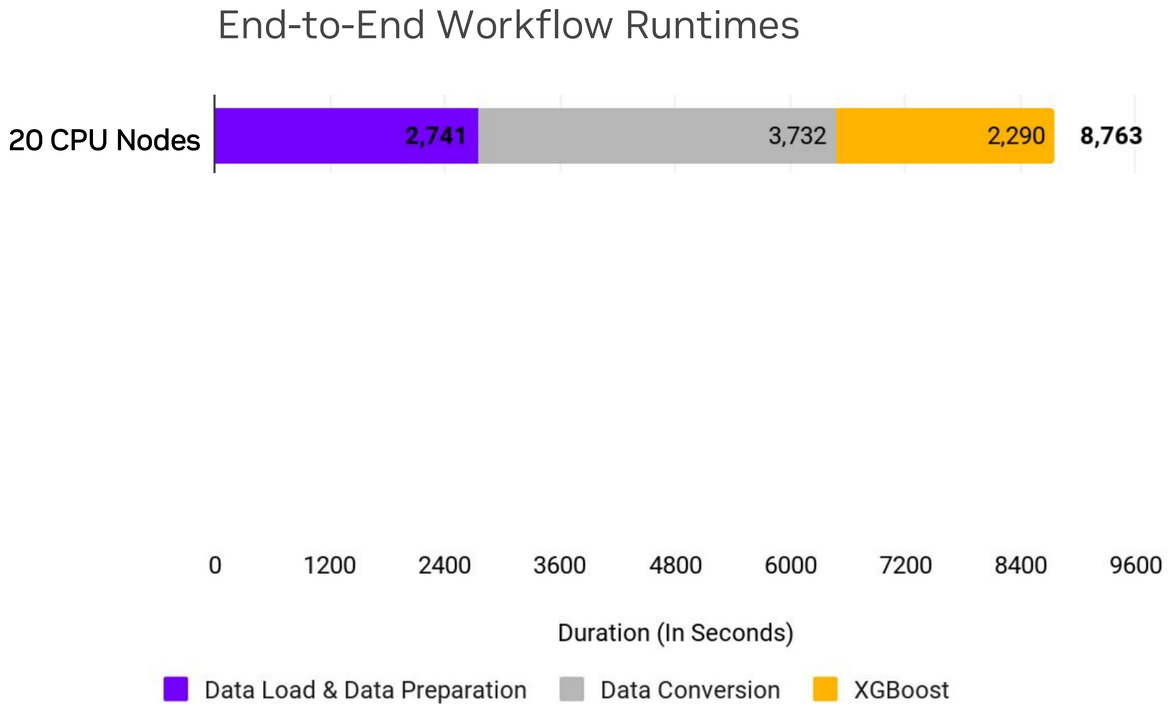
CUDA Toolkit *(C++/CUDA code and kernels)*

cuBLAS, cuDNN, cuSolver, cuSPARSE, ...

NVIDIA

Lightning-fast End-to-end Performance

Reducing Data Science Processes from Hours to Seconds



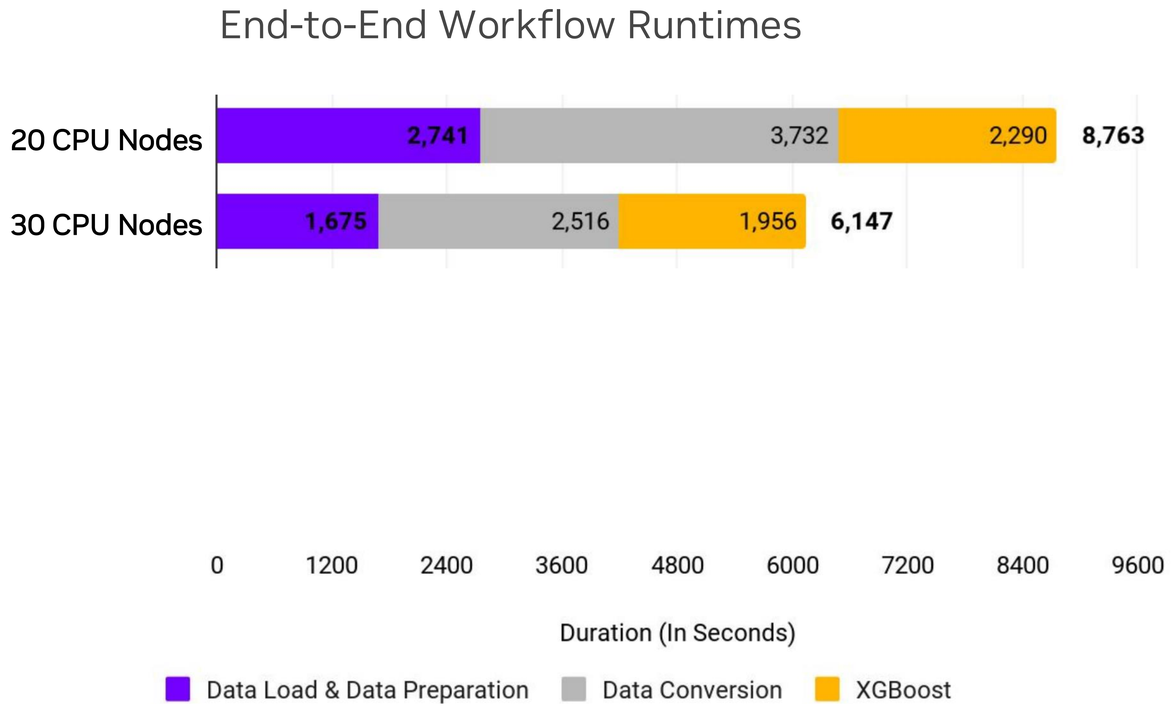
16 GPUs Provide More Power than 100 CPU Nodes

70x Faster Performance than Similar CPU Configuration

20x More Cost-Effective than Similar CPU Configuration

Lightning-fast End-to-end Performance

Reducing Data Science Processes from Hours to Seconds



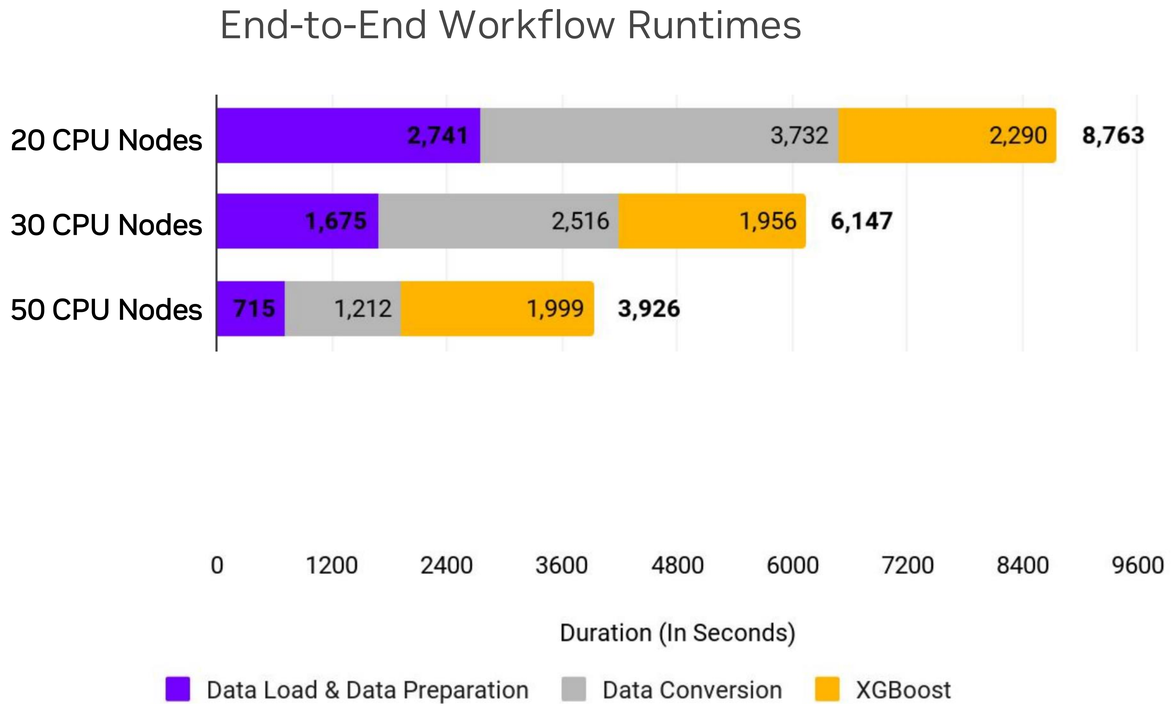
16 GPUs Provide More Power than 100 CPU Nodes

70x Faster Performance than Similar CPU Configuration

20x More Cost-Effective than Similar CPU Configuration

Lightning-fast End-to-end Performance

Reducing Data Science Processes from Hours to Seconds



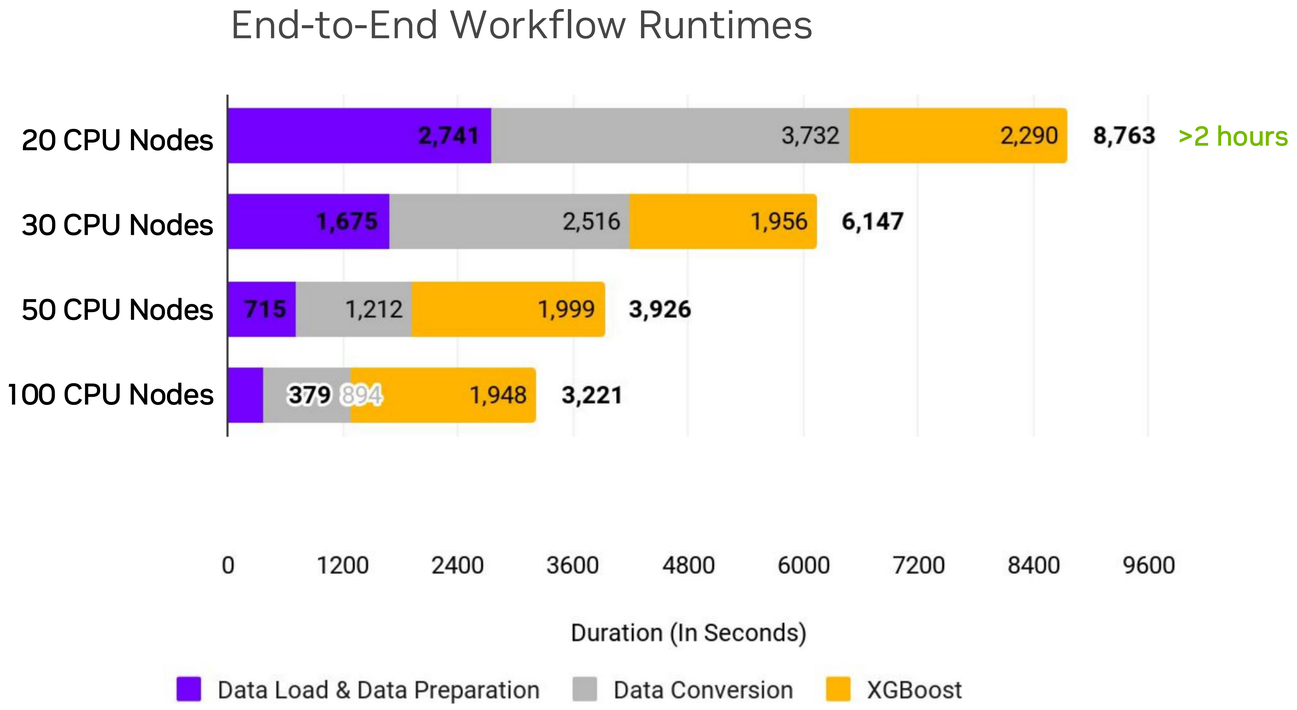
16 GPUs Provide More Power than 100 CPU Nodes

70x Faster Performance than Similar CPU Configuration

20x More Cost-Effective than Similar CPU Configuration

Lightning-fast End-to-end Performance

Reducing Data Science Processes from Hours to Seconds



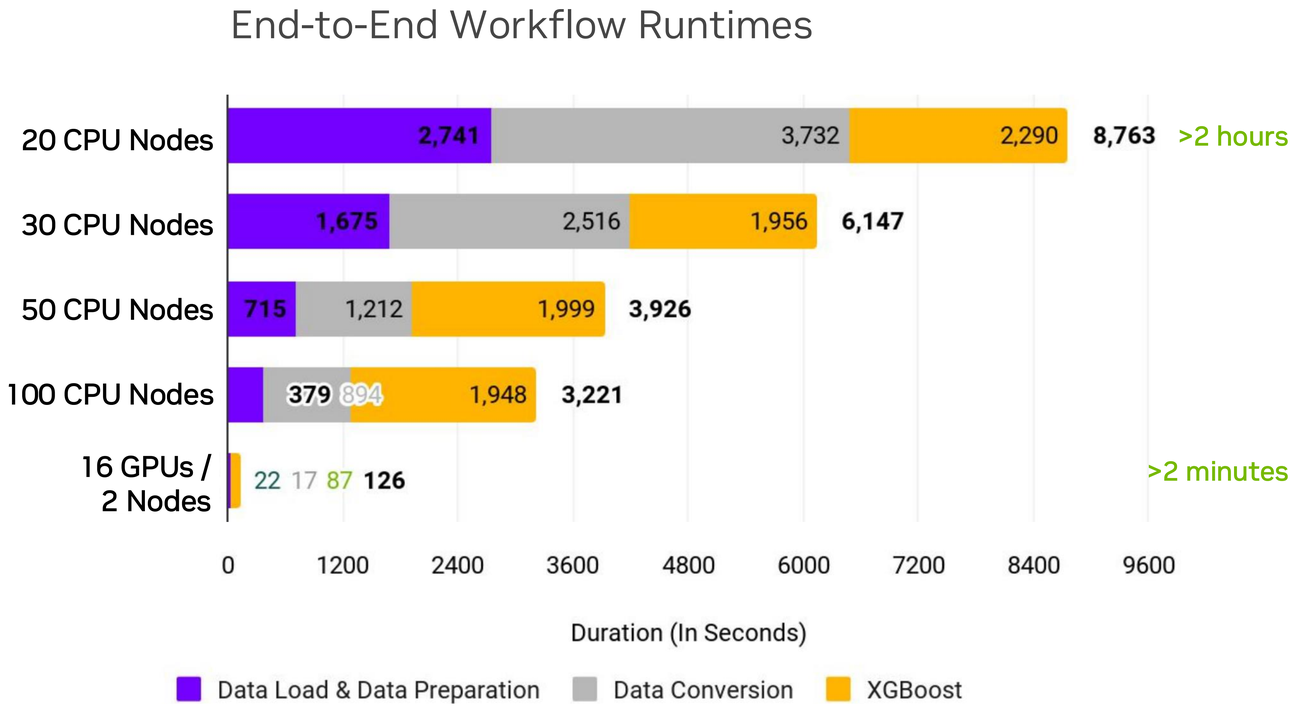
16 GPUs Provide More Power than 100 CPU Nodes

70x Faster Performance than Similar CPU Configuration

20x More Cost-Effective than Similar CPU Configuration

Lightning-fast End-to-end Performance

Reducing Data Science Processes from Hours to Seconds



16 GPUs Provide More Power than 100 CPU Nodes

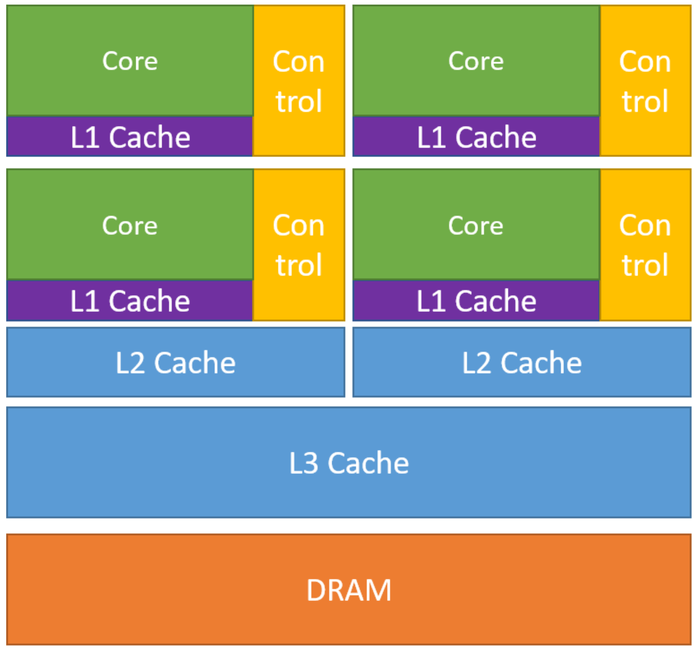
70x Faster Performance than Similar CPU Configuration

20x More Cost-Effective than Similar CPU Configuration

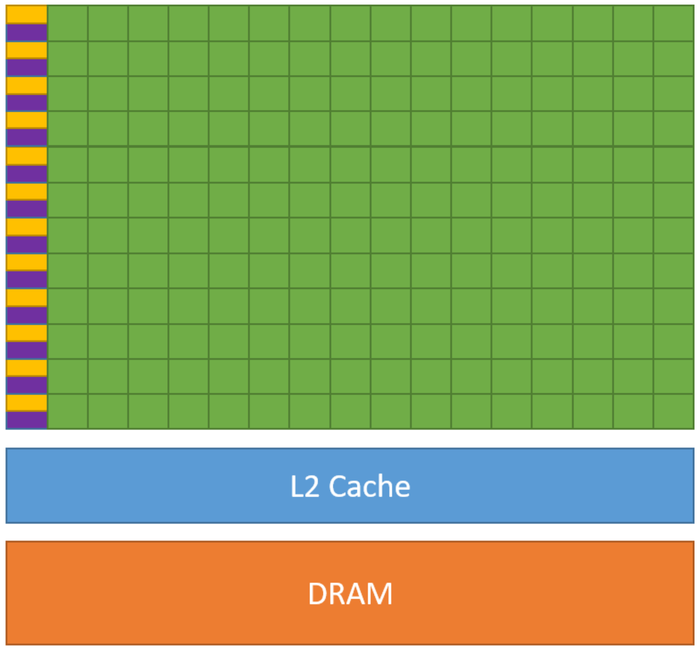
Speedups and Architecture

GPU vs CPU

RAPIDS



CPU



GPU

CUDA

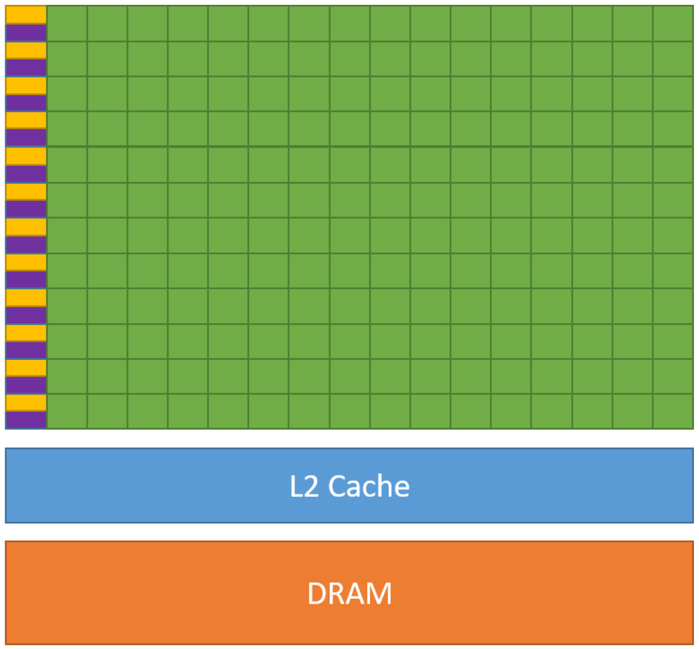
Sequential operations (tens of threads)

Parallel operations (thousands of threads)

GPU Computing Applications

GPU vs CPU

RAPIDS



GPU

Multiple streaming processors (SPs) (32 or 48 CUDA cores) make up 1 streaming multiprocessor (SM)

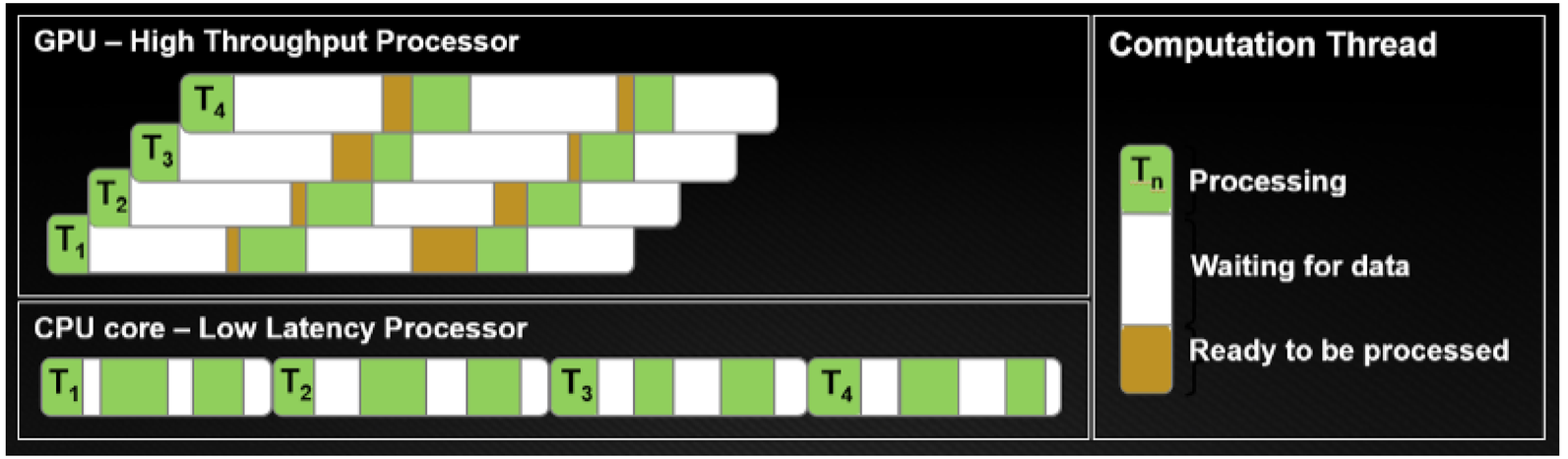
At time 0 a single warp of 32 threads is executed

All threads in a warp execute the same instruction

Each thread performs instruction on its own piece of data

GPU vs CPU

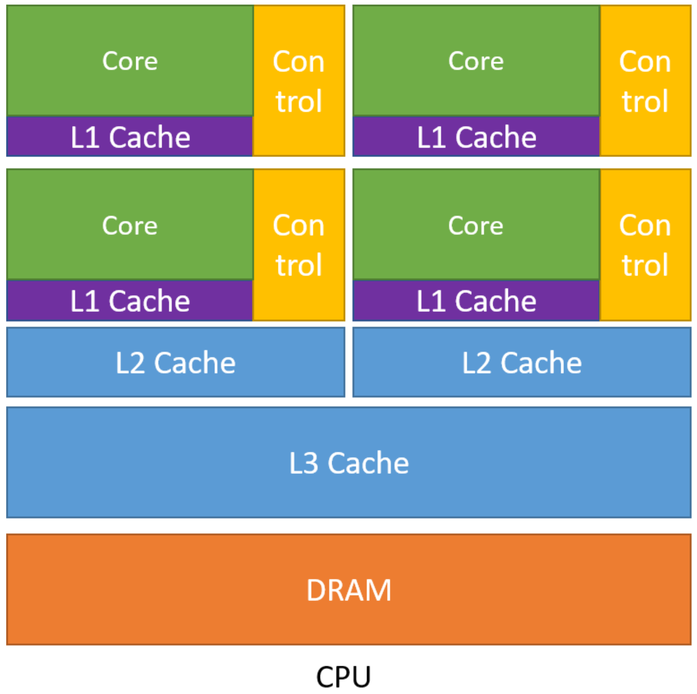
RAPIDS



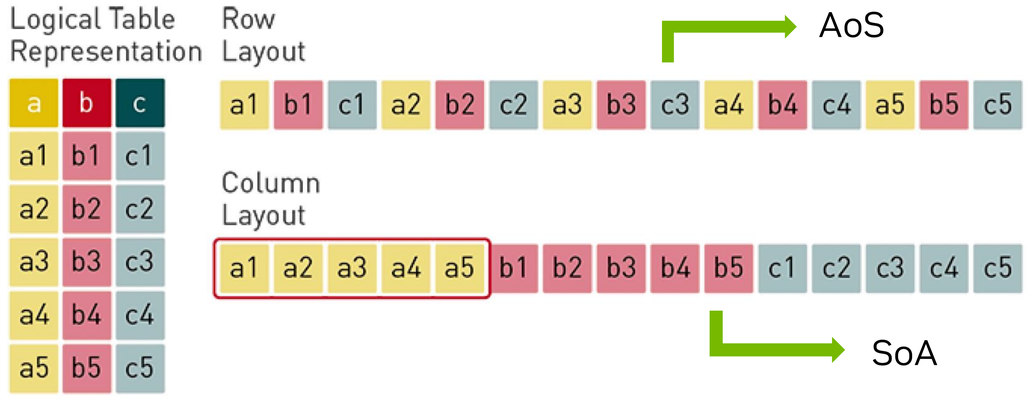
Keep GPUs busy

GPU vs CPU

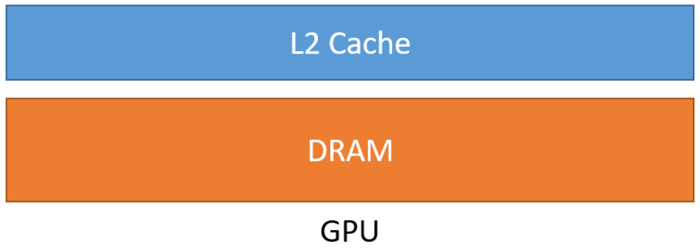
RAPIDS



CPU



CUDA



GPU

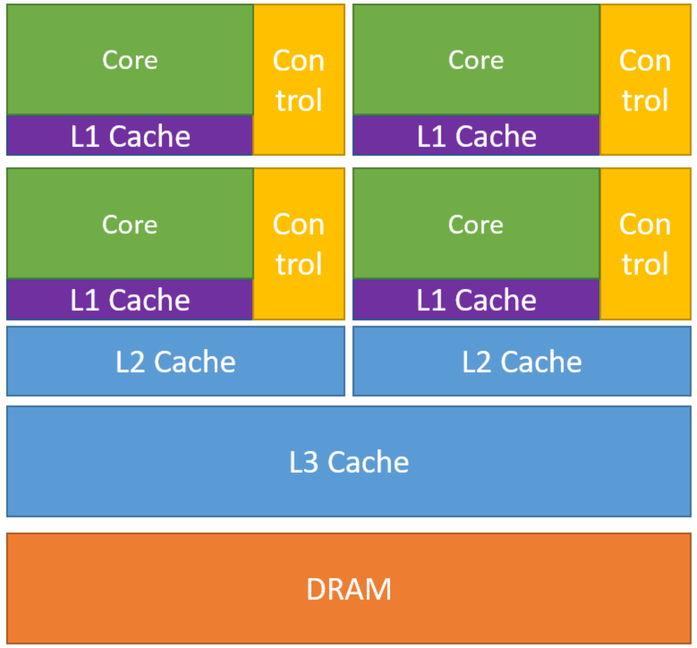
Sequential operations (tens of threads)

Parallel operations (thousands of threads)

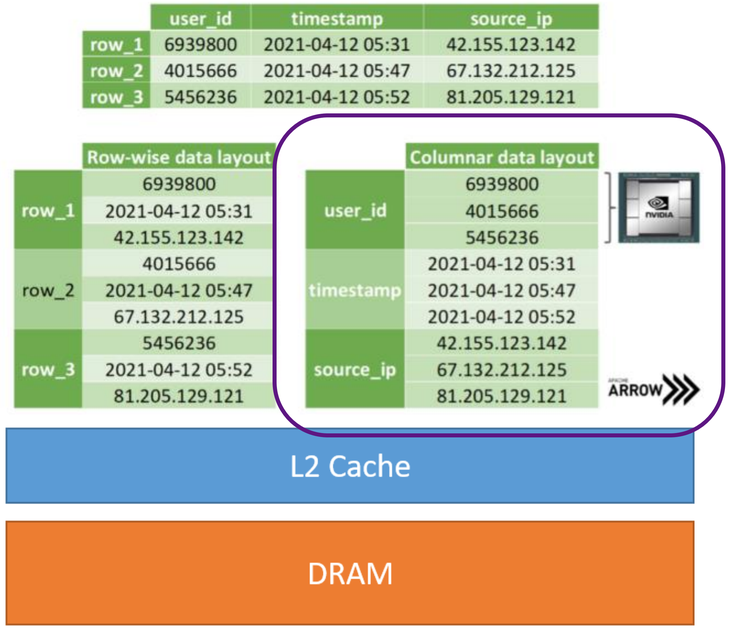
GPU Computing Applications

GPU vs CPU

RAPIDS



CPU



GPU

CUDA

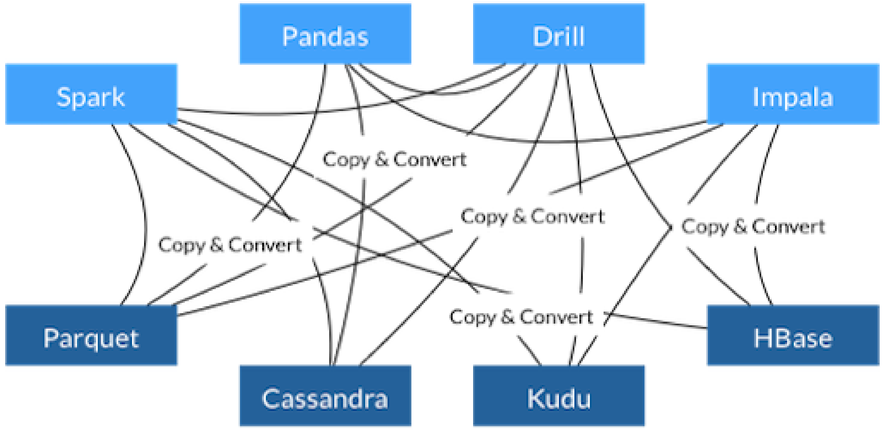
Sequential operations (tens of threads)

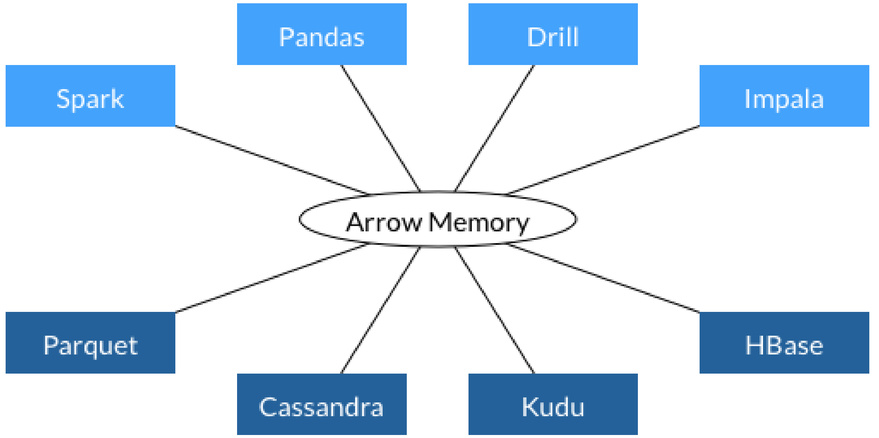
Parallel operations (thousands of threads)

GPU Computing Applications

Apache Arrow

Cross-language Development Platform for In-Memory Analytics





- Each system has its own internal memory format
- 70-80% computation wasted on serialization and deserialization
- Similar functionality implemented in multiple projects

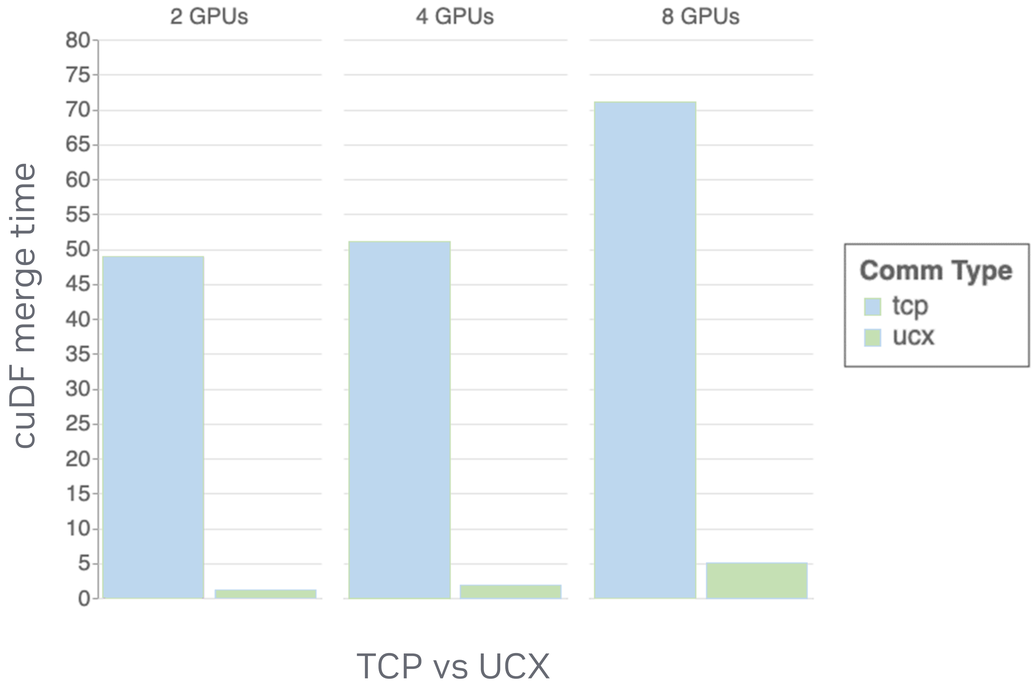
- All systems utilize the same memory format
- No overhead for cross-system communication
- Projects can share functionality (e.g., Parquet-to-Arrow reader)

Source: From Apache Arrow Home Page - https://arrow.apache.org/

Unified Communication X (UCX)

Bringing Hardware Accelerated Communications to Dask

- Dask-CUDA supports integration with UCX
  - Only two-line change necessary
- UCX provides uniform access to transports:
  - NVLink (direct GPU-to-GPU) and Infiniband (direct GPU-Fiber)
  - TCP, Shared memory
- Zero-copy GPU memory transfers over RDMA



Scaling Up and Out

RAPIDS + Dask + UCX

Scale Up / Accelerate

RAPIDS and Others

Accelerated on single GPU

CuPy/PyTorch/..
cuDF
cuML
cuGraph
Numba

RAPIDS

Python Data Ecosystem

NumPy, Pandas, Scikit-Learn, Numba, etc

Single CPU core
In-memory data

Numba
NumPy
pandas
PyData
scikit learn
NetworkX

RAPIDS + Dask + UCX

Multi-GPU
On single Node (DGX)
Or across a cluster

RAPIDS
DASK

Dask

Multi-cpu and multi-node distribution

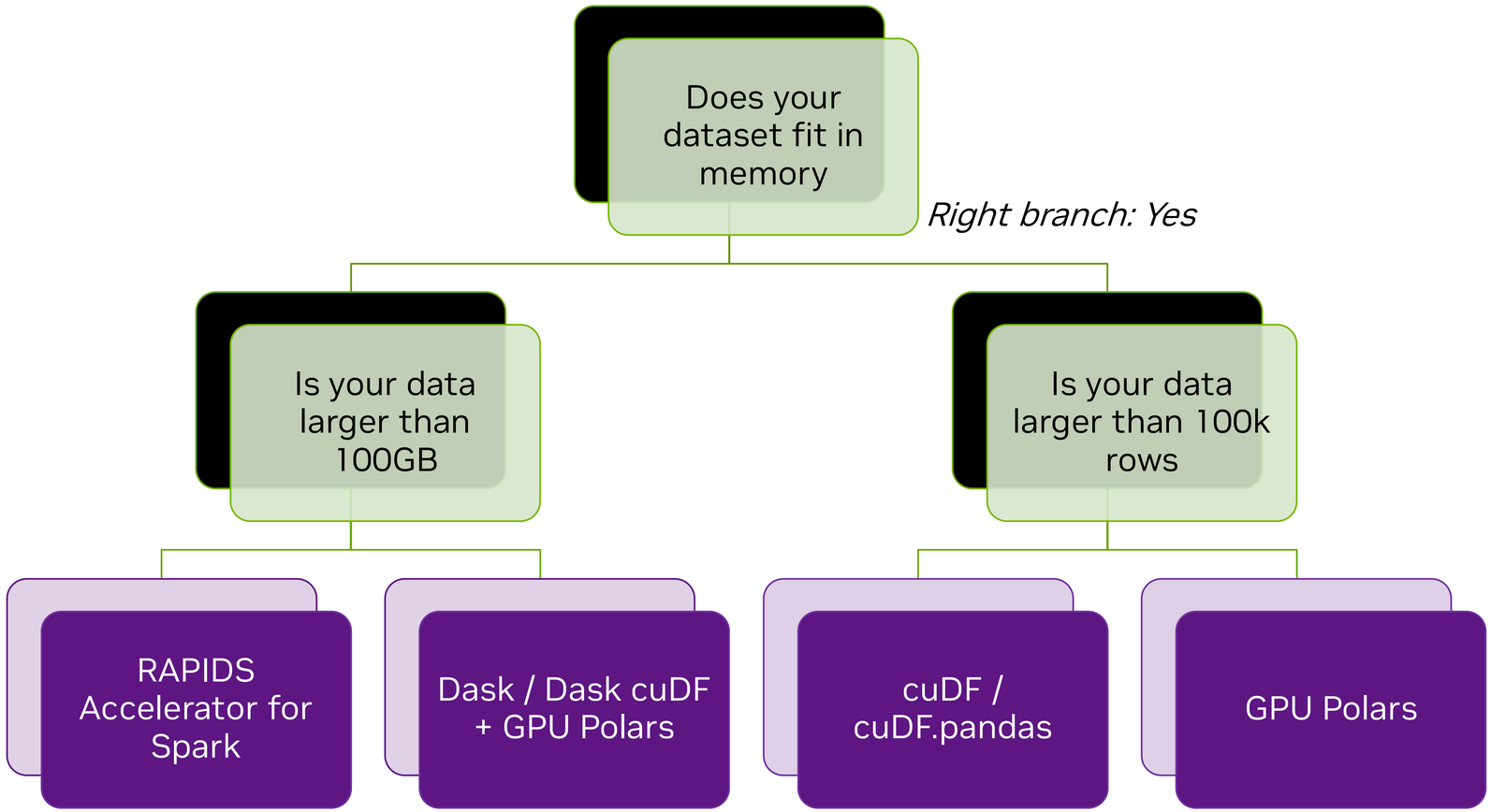
Dask Array
Dask DataFrame
Dask-ML

DASK

Scale Out / Parallelize

How to Choose The Correct API

Different APIs are specialized for different workloads



Success Stories

Customer Spotlight: PayPal

Accelerating computing with NVIDIA drives tangible business benefits

With hundreds of thousands of Apache Spark jobs running hourly, CPU based compute is both expensive and time-consuming

Customer Pain Point

Running Spark jobs in the cloud is expensive and time-consuming

By focusing on operations and queries that were computation-bound, PayPal was able to utilize GPU architecture to accelerate operations

NVIDIA Solution

Migration to GPUs allows PayPal to leverage compute

NVIDIA technology improved business outcomes by decreasing run time, requiring fewer machines, and increasing machine utilization

Tangible Business Benefit

Spark RAPIDS and GPUs drove ~70% cost savings

PayPal

NVIDIA

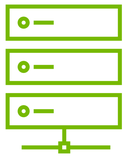
Customer Spotlight: Bunq

Digital bank debunks financial fraud with generative AI

With more than 12 million customers and 8 billion euros' worth of deposits made to date, bunq, an online bank, has become one of the largest neobanks in the EU

Customer Pain Point

Financial fraud and money laundering increasingly prevalent



Using supervised and unsupervised learning, bunq's AI-powered transaction-monitoring system is completely automated and easily scalable

NVIDIA Solution

NVIDIA hardware and GPU-accelerated data science libraries

Beyond model training, bunq utilizes NVIDIA solutions for Finn, bunq's personal AI-assistant, automated marketing efforts, and handling customer service tickets

Realized Business Benefits

Data processing accelerated 5x;
fraud detection model trained 100x faster

bunq

NVIDIA

NVIDIA RAPIDS Acceleration Libraries

Data Preparation
cuDF

Less Waiting. More Building.

> Data preparation and exploration takes 60 to 80 percent of the analytical pipeline in a typical machine learning or deep learning project

Accelerated data processing

Optimized libraries for various data sizes

100k of rows

More than 100k of rows

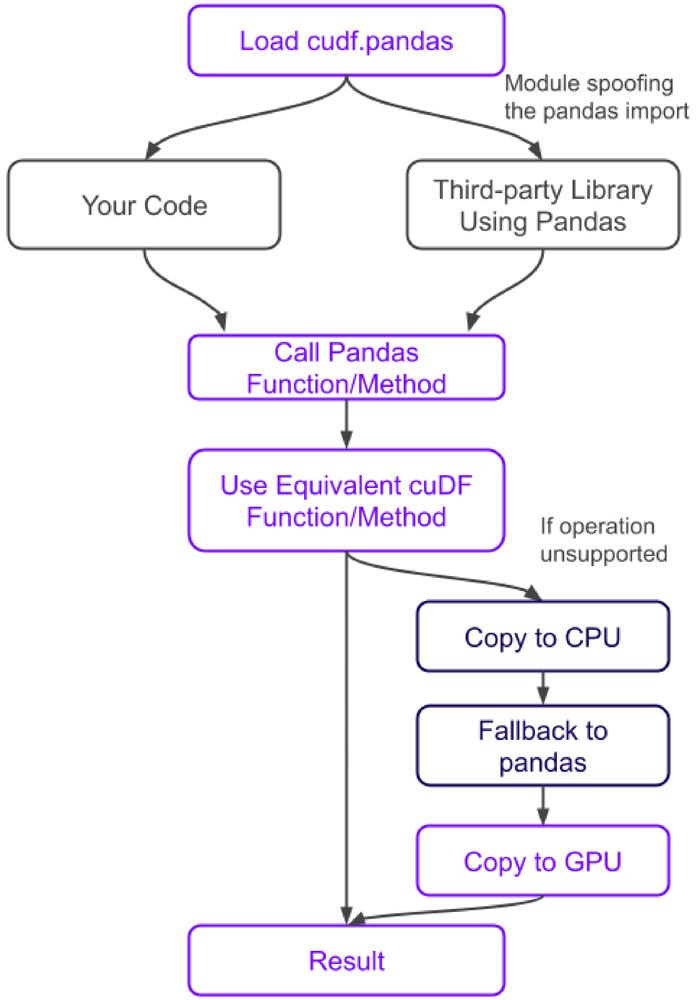
More than 100GB of data

Accelerated pandas

cudf.pandas: the zero code change GPU accelerator for pandas built on cuDF

- Requires *no changes* to existing pandas code. Just
  - `%load_ext cudf.pandas`
  - `$ python -m cudf.pandas <script.py>`
- 100% of the pandas API
- Accelerates workflows up to 150x using the GPU
- Compatible with code that uses third-party libraries
- Falls back to using pandas on the CPU for unsupported functions and methods

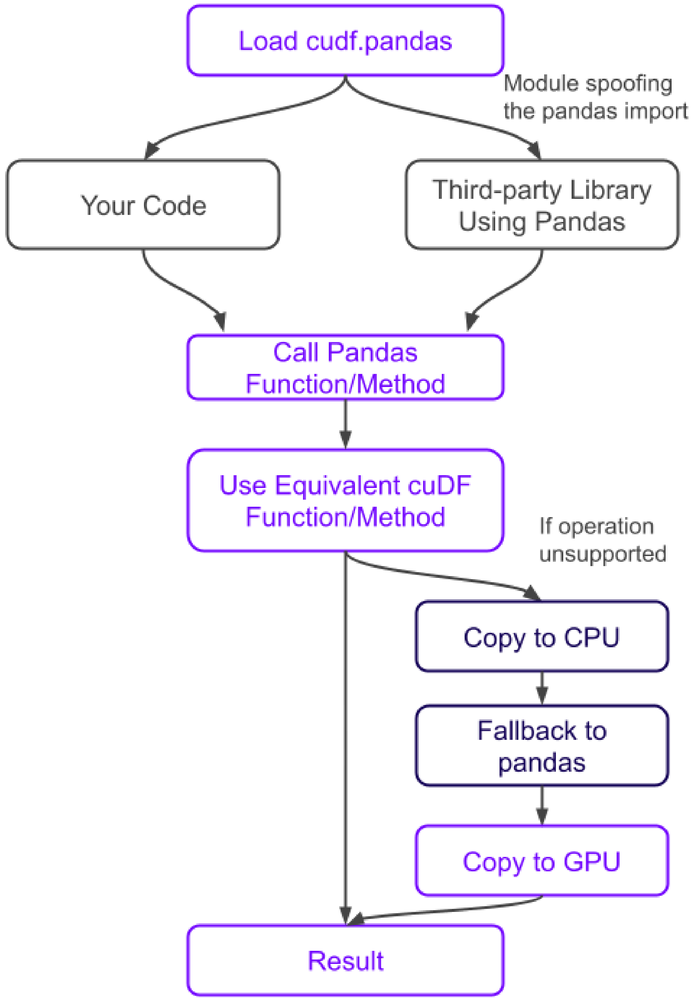
```
[ ]:
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

data = pd.read_parquet("data.parquet")
subset = data.index.indexer_between_time("09:30", "16:00")
data = data.iloc[subset]
results = data.groupby(pd.Grouper(freq="1D")).mean()

sns.lineplot(results)
plt.xticks(rotation=30)
```



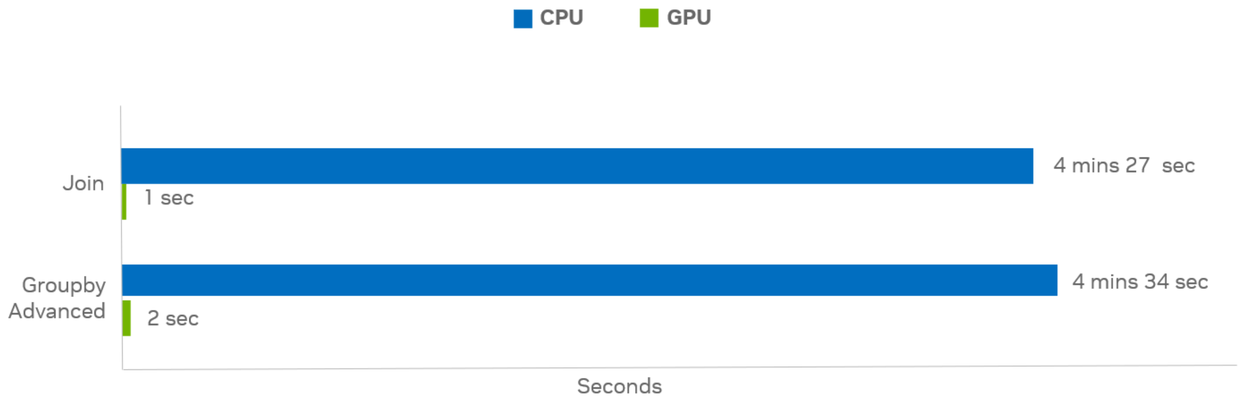
Accelerated pandas

cudf.pandas: the zero code change GPU accelerator for pandas built on cuDF

- Requires *no changes* to existing pandas code. Just
  - `%load_ext cudf.pandas`
  - `$ python -m cudf.pandas <script.py>`
- 100% of the pandas API
- Accelerates workflows up to 150x using the GPU
- Compatible with code that uses third-party libraries
- Falls back to using pandas on the CPU for unsupported functions and methods



150x Faster pandas



Standard DuckDB Data Benchmark (5 GB) performance comparison between cuDF.pandas and traditional pandas v2.2

cuDF pandas accelerator mode (cudf.pandas)

Bringing the speed of cuDF to every Pandas user

GPU

```
%load_ext cudf.pandas
```

```
%%time
import pandas as pd

weekday_names = {
    0: "Monday",
    1: "Tuesday",
    2: "Wednesday",
    3: "Thursday",
    4: "Friday",
    5: "Saturday",
    6: "Sunday",
}

df = pd.read_parquet("/tmp/nyc_parking_violations_2022.parquet")
df["Issue Date"] = df["Issue Date"].astype("datetime64[ms]")
df["issue_weekday"] = df["Issue Date"].dt.weekday.map(weekday_names)
df.groupby(["issue_weekday"])["Summons Number"].count().sort_values()
```

```
CPU times: user 601 ms, sys: 141 ms, total: 742 ms
Wall time: 720 ms
issue_weekday
Sunday        462992
Saturday     1108385
Monday       2488563
Wednesday    2760088
Tuesday      2809949
Friday       2891679
Thursday     2913951
Name: Summons Number, dtype: int64
```

- Constantly ensured full Pandas compatibitlity:
  - Unit-testing with Pandas test suite
  - Integration test for key PyData libraries

cuDF pandas accelerator mode (cudf.pandas)

Bringing the speed of cuDF to every Pandas user

Profiler

```
%%cudf.pandas.profile
import pandas as pd

weekday_names = {
    0: "Monday",
    1: "Tuesday",
    2: "Wednesday",
    3: "Thursday",
    4: "Friday",
    5: "Saturday",
    6: "Sunday",
}

df = pd.read_parquet("/tmp/nyc_parking_violations_2022.parquet")
df["Issue Date"] = df["Issue Date"].astype("datetime64[ms]")
df["issue_weekday"] = df["Issue Date"].dt.weekday.map(weekday_names)
df.groupby(["issue_weekday"])["Summons Number"].count().sort_values()
```

Total time elapsed: 1.854 seconds
14 GPU function calls in 1.200 seconds
2 CPU function calls in 0.027 seconds

Stats

Function	GPU ncalls	GPU cumtime	GPU percall	CPU ncalls	CPU cumtime	CPU percall
read_parquet	1	0.841	0.841	0	0.000	0.000
DataFrame.__getitem__	2	0.002	0.001	0	0.000	0.000
NDFrame.astype	1	0.043	0.043	0	0.000	0.000
DataFrame.__setitem__	2	0.006	0.003	0	0.000	0.000
Series	1	0.000	0.000	0	0.000	0.000
Series.map	1	0.253	0.253	0	0.000	0.000
DataFrame.groupby	1	0.001	0.001	0	0.000	0.000
DataFrameGroupBy.__getitem__	1	0.002	0.002	0	0.000	0.000
GroupBy.count	1	0.028	0.028	0	0.000	0.000
Series.sort_values	1	0.005	0.005	0	0.000	0.000
Series.__repr__	1	0.017	0.017	0	0.000	0.000
Series.to_frame	1	0.003	0.003	0	0.000	0.000
DataFrame._repr_html_	0	0.000	0.000	1	0.020	0.020
NDFrame._repr_latex_	0	0.000	0.000	1	0.007	0.007

Not all pandas operations ran on the GPU. The following functions required CPU fallback:

- DataFrame._repr_html_
- NDFrame._repr_latex_

To request GPU support for any of these functions, please file a Github issue here:
https://github.com/rapidsai/cudf/issues/new/choose.

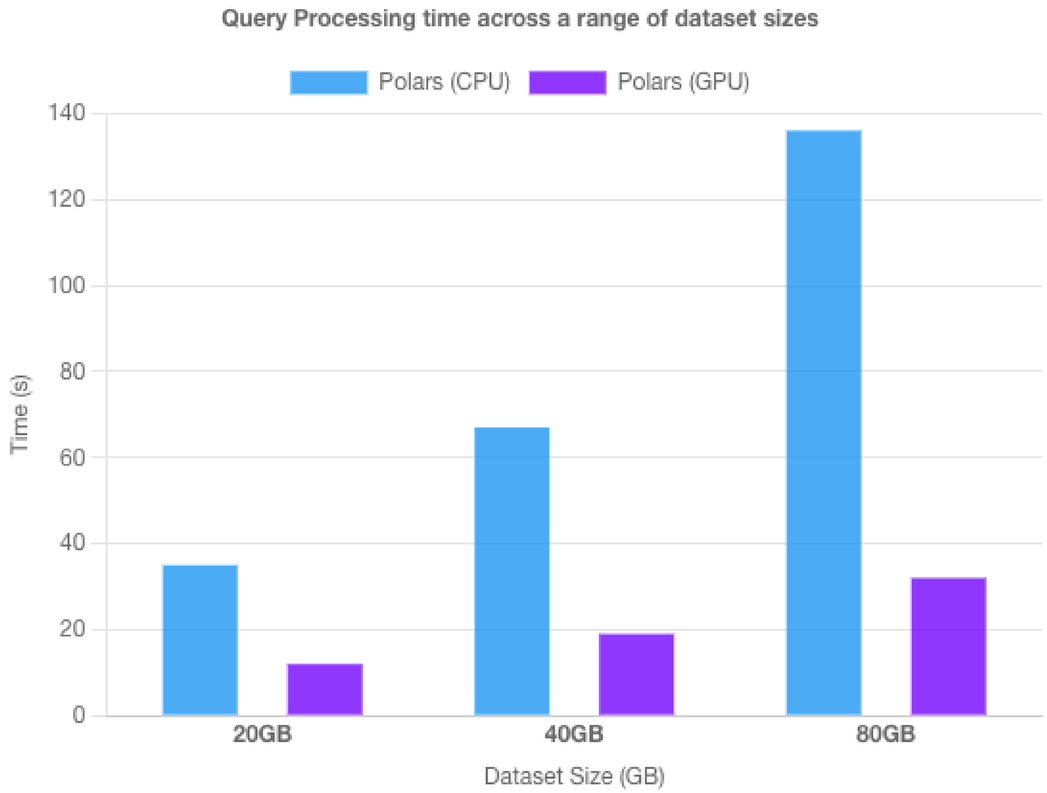
- Profiler built in
- Accessible in command line and notebook
- Possible to see how much time each function spends divided on CPU and GPU

Accelerated polars

Zero code change accelerator for Polars

- Zero code change to existing Polars code
- Polars attempts to execute operations on GPU first and falls back to CPU if necessary
- Inspection of optimized query plan to attempt to build execution plan. If fails → fallback to CPU

```
import polars as pl

ldf = pl.LazyFrame({"a": [1.242, 1.535]})

print(
    ldf.select(
        pl.col("a").round(1)
    ).collect(engine="gpu")
)
```



Accelerated Dask

Just set “cudf” as the backend and use Dask-CUDA Workers

- Configurable Backend and GPU-Aware Workers
- Memory Spilling (GPU->CPU->Disk)
- Optimized Memory Management
- Accelerated RDMA and Networking (UCX)

```
import dask
from dask_cuda import LocalCUDACluster
from dask.distributed import Client
import dask.dataframe as dd

dask.config.set({"dataframe.backend": "cudf"})

cluster = LocalCUDACluster(...)
client = Client(cluster)
```

```
from dask_cuda import LocalCUDACluster
cluster = LocalCUDACluster(...)
cluster
```

LocalCUDACluster

382454ff

Dashboard: http://127.0.0.1:8787/status	**Workers:** 1
Total threads: 1	**Total memory:** 0.98 TiB
Status: running	**Using processes:** True

▸ Scheduler Info

Data Preparation
NVIDIA Spark RAPIDS

How Spark RAPIDS Works

Key technologies for GPU acceleration

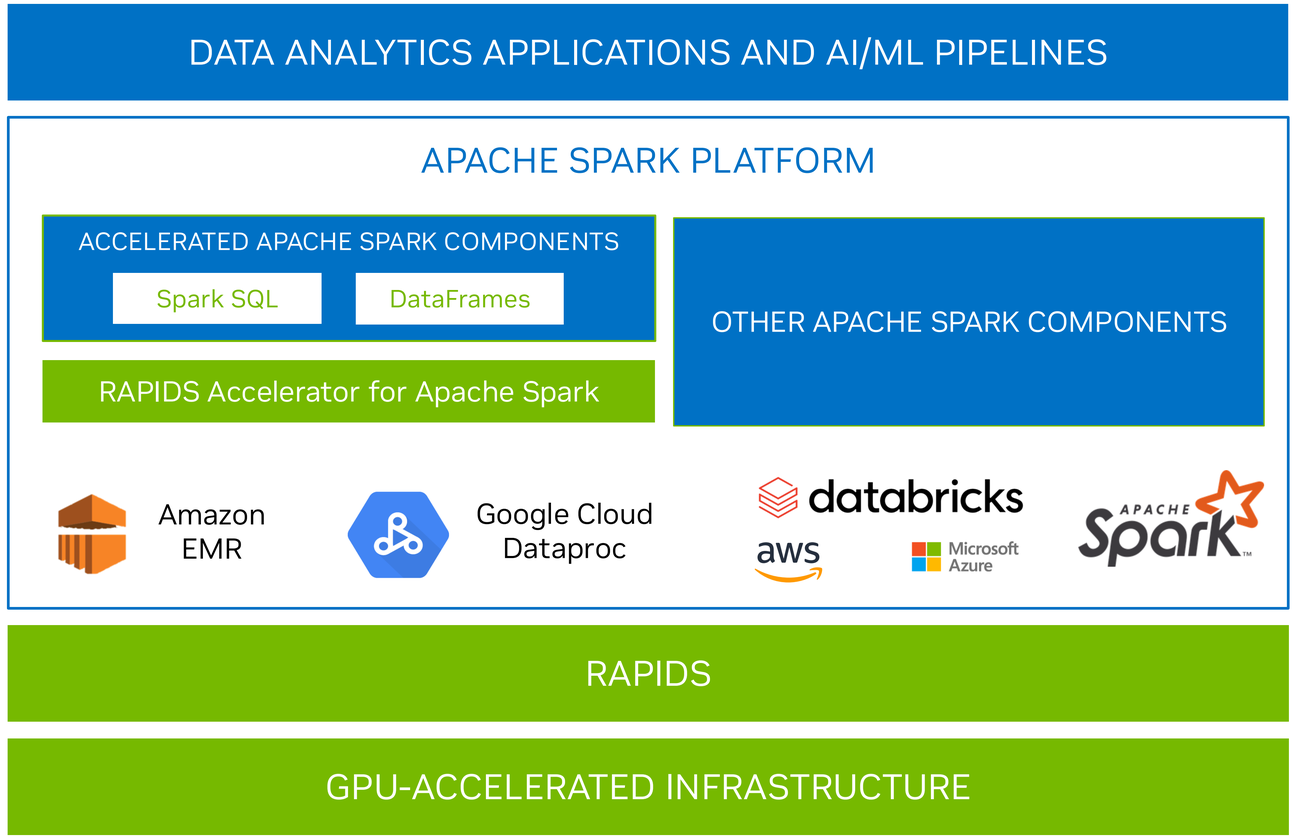
NVIDIA RAPIDS Accelerator

- Operates as a software plugin to popular Apache Spark platforms
  - Automatically accelerates supported operations
  - Requires no code changes
- Operations currently accelerated
  - Spark SQL
  - DataFrame
- Works with Spark standalone, YARN clusters, Kubernetes clusters

Key Spark 3 Innovations

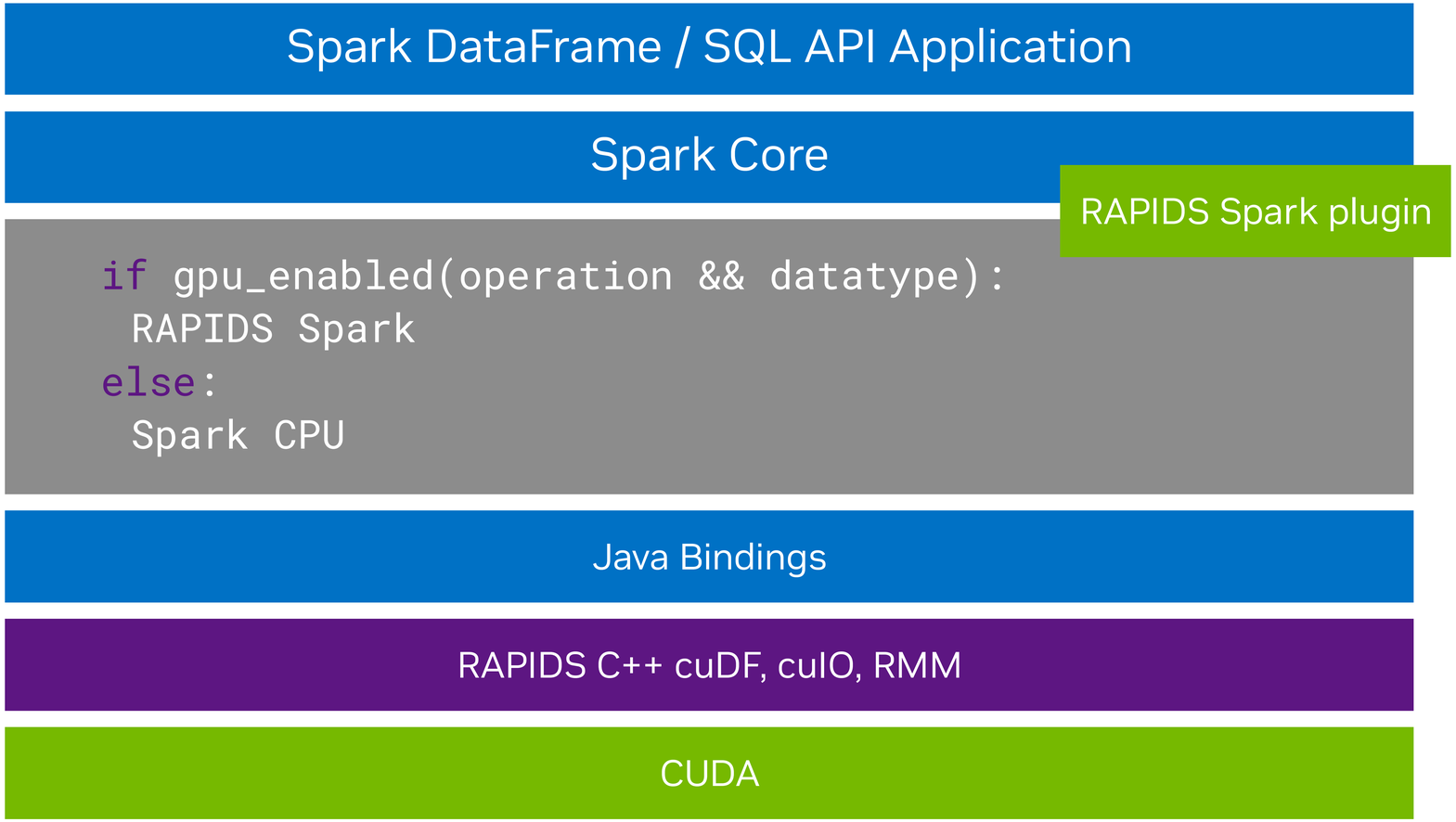
Columnar processing support in the Catalyst query optimizer – allows efficient GPU acceleration

GPU-aware scheduling of executors with a specified number of GPUs and how many GPUs for each task



RAPIDS Accelerator for Apache Spark

Spark Plugin for GPU Acceleration



Accelerated Apache Spark

Zero code change acceleration for Spark DataFrames and SQL

CPU Spark

```
spark.sql("""
select
    order
    count(*) as order_count
from
    orders"""
)
```

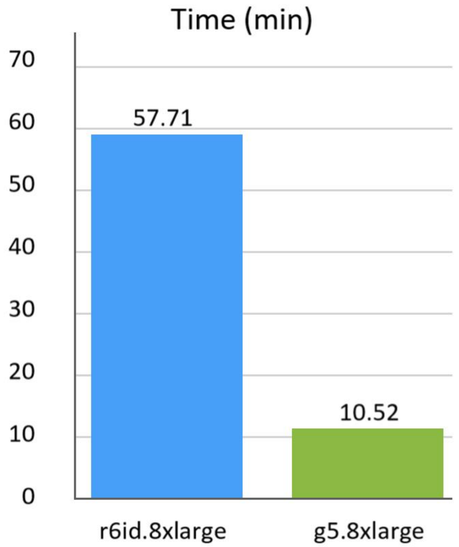
GPU Spark

```
spark.conf.set("spark.plugins",
"com.nvidia.spark.SQLPlugin")

spark.sql("""
select
    order
    count(*) as order_count
from
    orders"""
)
```

Average Speed-Ups: >5x

- RAPIDS operates as a software plugin
- Automatically accelerates supported operations (with CPU fallback if needed)
- Works with Spark standalone, YARN clusters, Kubernetes clusters
- Compatible with PySpark, SparkR, Java, Scala and other DataFrame-based APIs
- How to run
  - Add jar to classpath and set spark.plugins config

NVIDIA Decision Support Benchmark 3TB (Public Cloud)



5.5x faster

80% cost savings

Apache Spark 3.4.1, RAPIDS Spark release 24.04
See GTC session S62257 for details

Hybrid CPU/GPU libraries

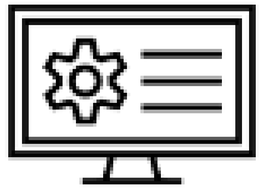
NVIDIA

Simple Steps to Savings

Guided by Accelerated Spark Analysis Tool

Qualification

Identify and estimate the cost savings and acceleration potential of your Spark workloads based on an analysis of the log files.



Bootstrap

Get recommended configuration parameters for GPU acceleration on your Spark cluster. Use them for initial run.

Profiling / Tuning

See optimized configuration per application based on the initial (bootstrap) job run.

Machine Learning
cuML

cuML

Accelerated machine learning with a scikit-learn API

CPU

Scikit-learn

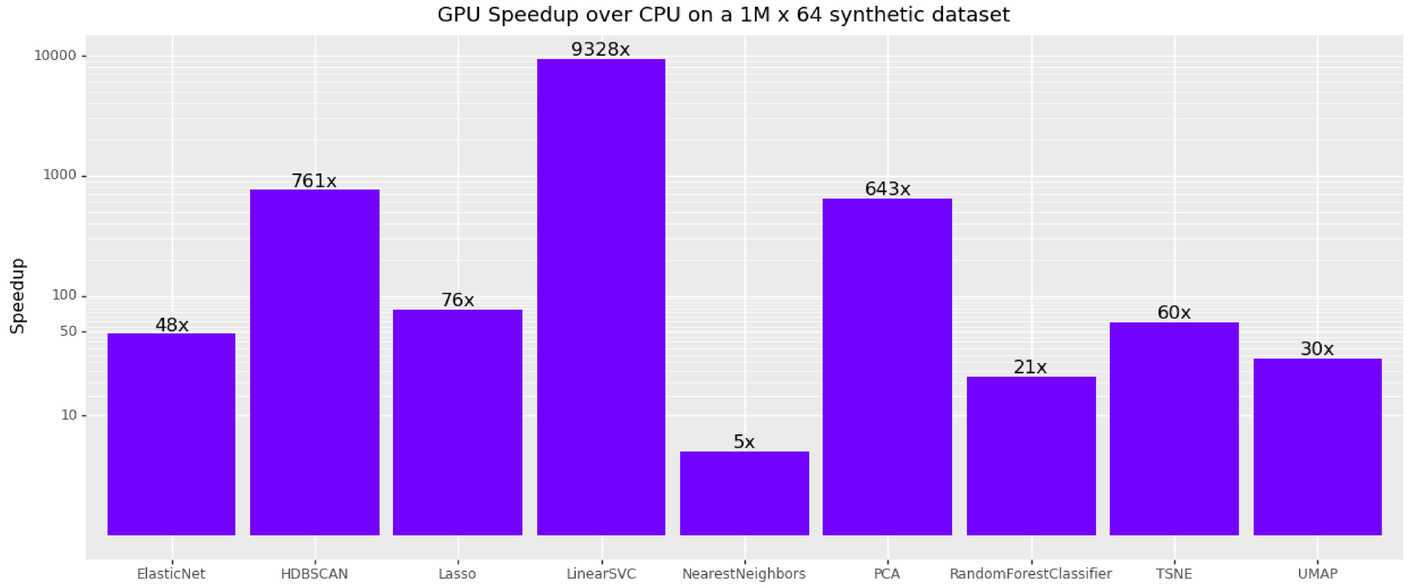
```
>>> from sklearn.ensemble import
RandomForestClassifier
>>> clf = RandomForestClassifier()
>>> clf.fit(x, y)
```

GPU

cuML

```
>>> from cuml.ensemble import
RandomForestClassifier
>>> clf = RandomForestClassifier()
>>> clf.fit(x, y)
```

50+ GPU-Accelerated Algorithms



Time Series	Classification	Regression	Clustering	Preprocessing
Cross Validation	Tree Models	Dimensionality Reduction	Explainability	

A100 GPU vs. AMD EPYC 7642 (96 logical cores)
cuML 23.04, scikit-learn 1.2.2, umap-learn 0.5.3

cuML-CPU

CPU version of cuML

```
Conda install –c rapidsai –c nvidia –c conda-forge cuml-cpu=23.10
```

CPU

Scikit-learn

```
>>> from sklearn.ensemble import
RandomForestClassifier
>>> clf = RandomForestClassifier()
>>> clf.fit(x, y)
```

GPU

cuML

```
>>> from cuml.ensemble import
RandomForestClassifier
>>> clf = RandomForestClassifier()
>>> clf.fit(x, y)
```

CPU

cuML

```
>>> from cuml.ensemble import
RandomForestClassifier
>>> clf = RandomForestClassifier()
>>> clf.fit(x, y)
```

GPU

cuML

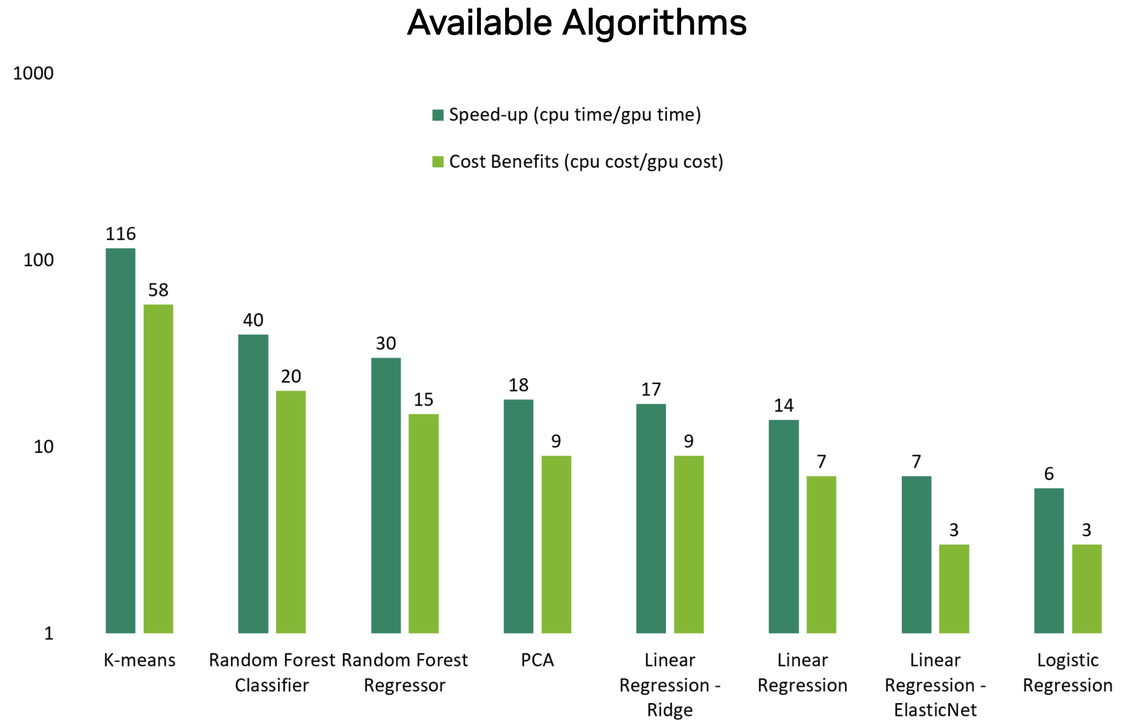
```
>>> from cuml.ensemble import
RandomForestClassifier
>>> clf = RandomForestClassifier()
>>> clf.fit(x, y)
```

Accelerated Spark Machine Learning

SparkML built on tops of RAPIDS cuML allows no-change speedups for Spark

- Compatible with pyspark.ml DataFrame APIs
- Built on top of RAPIDS
- Requires no application code change
- Package import change:

```
from spark_rapids_ml.clustering import Kmeans

kmeans_estm = KMeans()\
.setK(100)\
.setFeaturesCol("features")\
.setMaxIter(30)

kmeans_model = kmeans_estm.fit(pyspark_data_frame)

kmeans_model.write().save("saved-model")

transformed = kmeans_model.transform(pyspark_data_frame)
```



Accelerated XGBoost

"XGBoost is All You Need" – Bojan Tunguz, 4x Kaggle Grandmaster

CPU

XGBoost

```
>>> from xgboost import XGBClassifier
>>> clf = XGBClassifier()
>>> clf.fit(x, y)
```

GPU

XGBoost

```
>>> from xgboost import XGBClassifier
>>> clf = XGBClassifier(device="cuda")
>>> clf.fit(x, y)
```

Up to 20x Speedups

- One line of code change to unlock up to 20x speedups with GPUs
- Scalable to the world's largest datasets with Dask and PySpark
- Built-in SHAP support for model explainability
- Deployable with Triton for lighting-fast inference in production
- RAPIDS helps maintain the XGBoost project

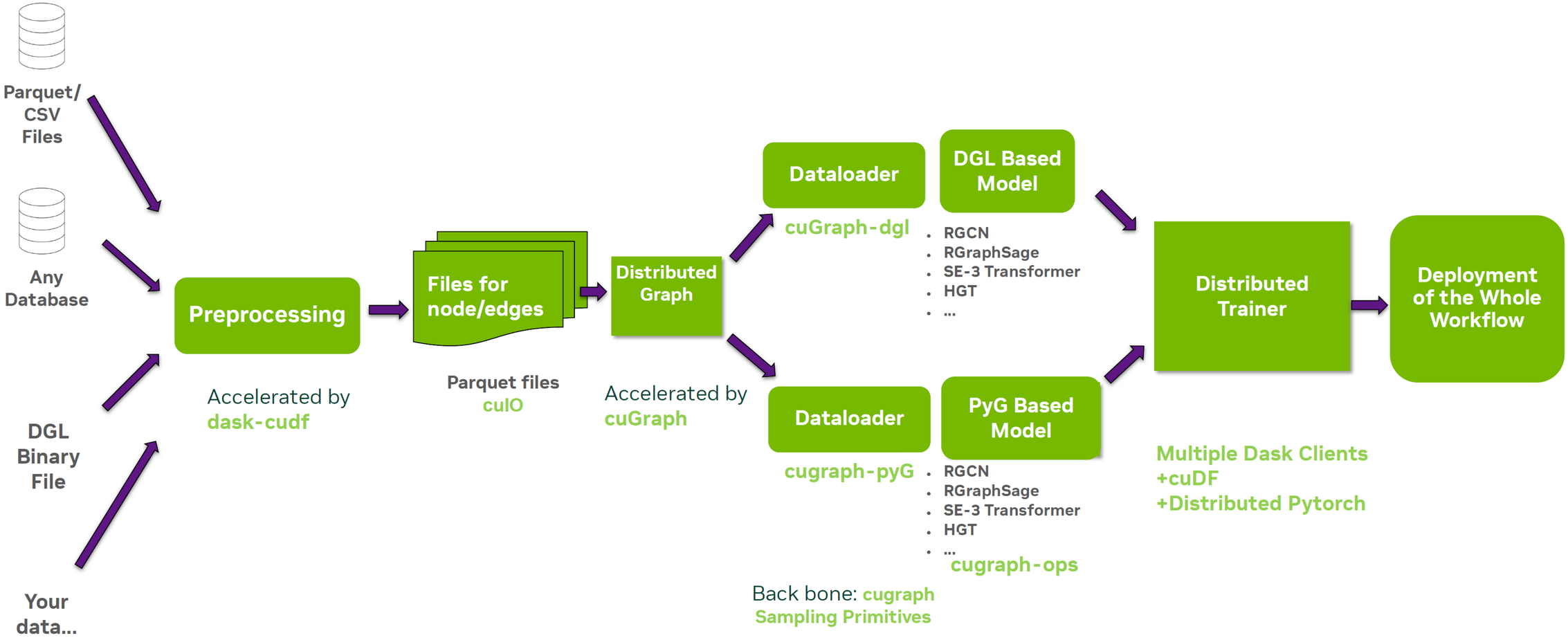
XGBoost 2.0 improvements:

- Column-based split for federated learning
- Multi-target trees with vector-leaf outputs
- `approx` tree method is now GPU accelerated
- Improved external memory support

Graph Analytics cuGraph

cuGraph + GNN Frameworks

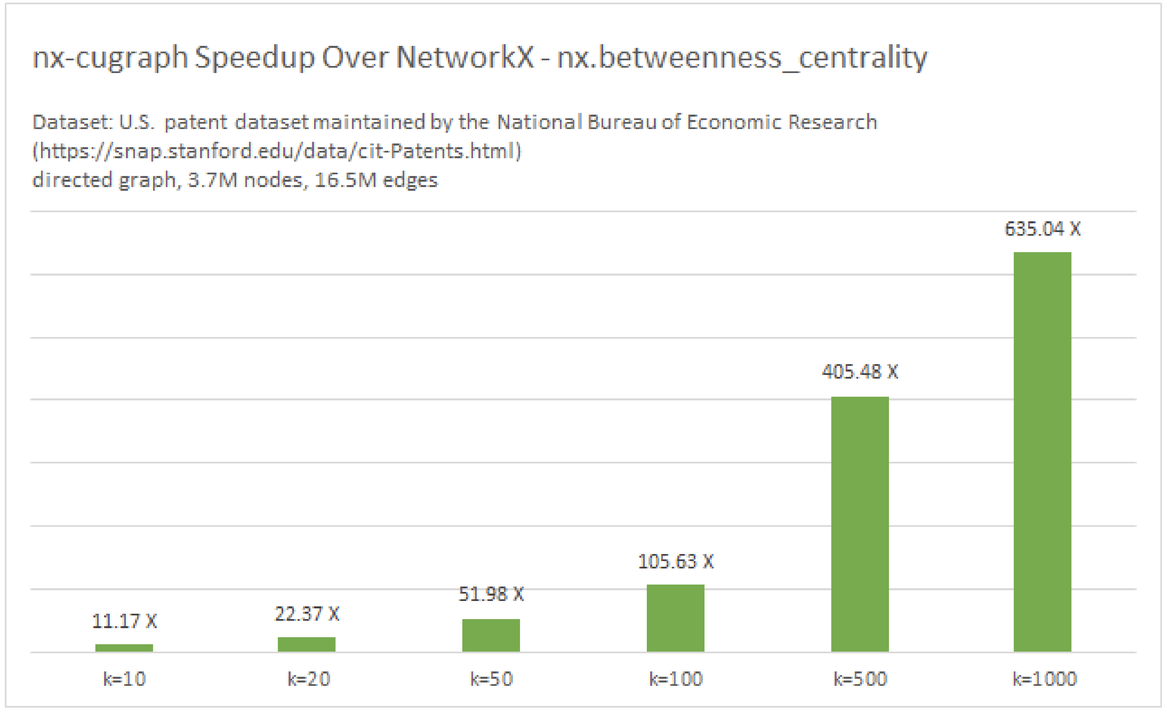
End-to-End Workflows



Accelerated NetworkX

Zero code change acceleration for NetworkX, powered by cuGraph

- Zero code change GPU-acceleration for NetworkX code
- Accelerates algorithms up to 600x, based on algorithm and graph size
- Support for 60 popular graph algorithms and growing
- Fallback to using CPU NetworkX for any unsupported algorithms



NetworkX 3.2. CPU: Intel(R) Xeon(R) Platinum 8480CL 2TB. GPU: NVIDIA H100 80GB

NetworkX
Network Analysis in Python

NVIDIA

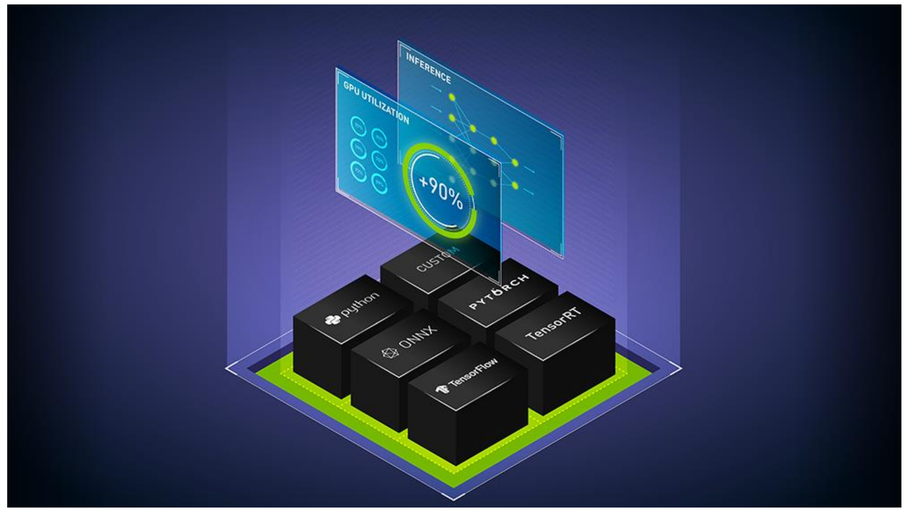
```python
import pandas as pd
import networkx as nx

url = "https://data.rapids.ai/cugraph/datasets/cit-Patents.csv"
df = pd.read_csv(url, sep=" ", names=["src", "dst"], dtype="int32")
G = nx.from_pandas_edgelist(df, source="src", target="dst")

%time result = nx.betweenness_centrality(G, k=10)
```

```
user@machine:/# ipython bc_demo.ipy
CPU times: user 7min 38s, sys: 5.6 s, total: 7min 44s
Wall time: 7min 44s

user@machine:/# NETWORKX_BACKEND_PRIORITY=cugraph ipython bc_demo.ipy
CPU times: user 18.4 s, sys: 1.44 s, total: 19.9 s
Wall time: 20 s
```

pip install nx-cugraph-cu12 --extra-index-url https://pypi.nvidia.com
conda install -c rapidsai -c conda-forge -c nvidia nx-cugraph

Accelerated Inference

Triton Inference Server



- Part of NVIDIA AI platform
- Available with NVAIE
- Open-source software
- Standardizes and unifies AI model deployment and execution

Concurrent execution

- Enables high GPU utilization and throughput

Dynamic batching

- Increases throughput under latency constraints

Model analyzer

- Finds best configuration for models to maximize target function (possible targets: latency, throughput, memory footprint)

Multi-GPU Multi-node inference

- Inter-layer parallelism
- Intra-layer parallelism

RAPIDS Forest Inference Library backend

- Backend support for tree-based ML workloads

Amazon SageMaker integration

- Native integration of NVIDIA Triton Inference Server in Amazon SageMaker

RAPIDS FOREST INFERENCE LIBRARY (FIL)

Inference on Tree-Based ML Models

Native support

- XGBoost
- LightGBM

Support with Treelite serialization

- GradientBoostinClassifier / -Regressor
- IsolationForest
- RandomForestRegressor
- ExtraTreesClassifier / -Regressor
- Any tree model when exported to Treelite's checkpoint serialization format

Supported serialization formats

- XGBoost JSON (< 1.7)
- XGBoost Binary
- LightGBM Text
- Treelite binary checkpoint

Deploying Models in Triton

1. Create model repository directory
2. Serialize models
   1. Either on disk
   2. or cloud
3. Add configuration files to describe model and how to use it (batch size, number of features, number of outputs, type of model, instance kind (GPU / CPU) etc.)
4. Start server
5. Submit inference requests
6. Optimize performance (throughput, latency, device utilization etc.)

By deploying a model on GPU more accurate models can be used in production without going over latency budget

Documentation: https://github.com/triton-inference-server/fil_backend

Optimized Inference for Tree-based Models

Forest Inference Library (FIL) Backend for both CPUs and GPUs

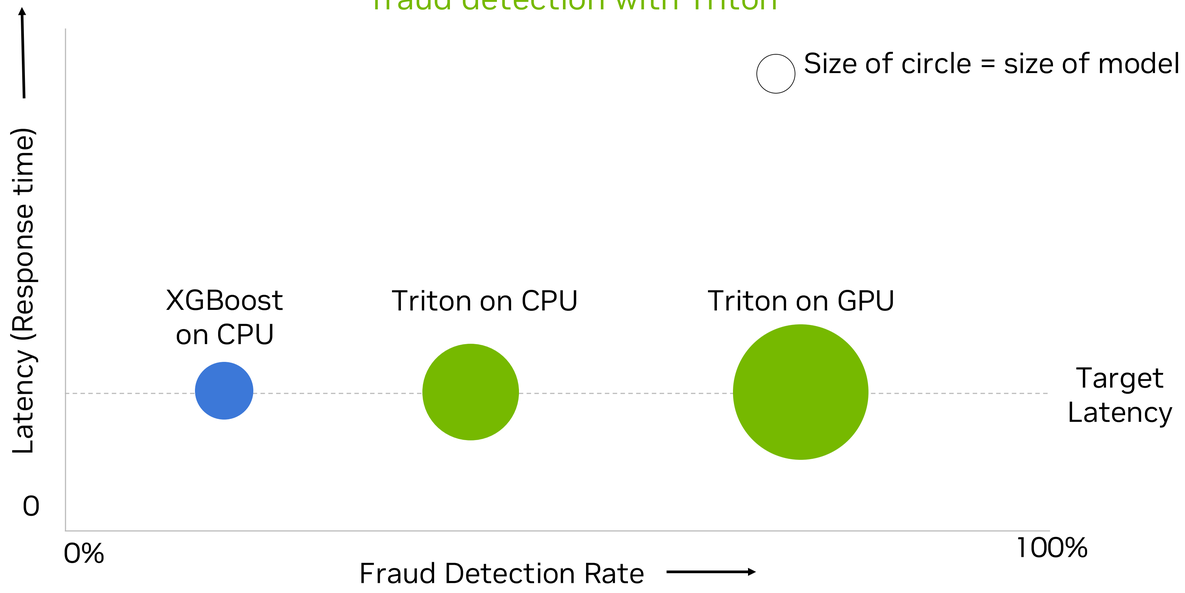
Real time inference for models of all sizes

Model explainability with SHAP values

Run multiple tree-based models concurrently on the same CPU/GPU

Fraud Detection Example

Run bigger models with low latency for improved fraud detection with Triton



dmlc XGBoost

scikit learn RandomForest

LightGBM

RAPIDS cuML RandomForest

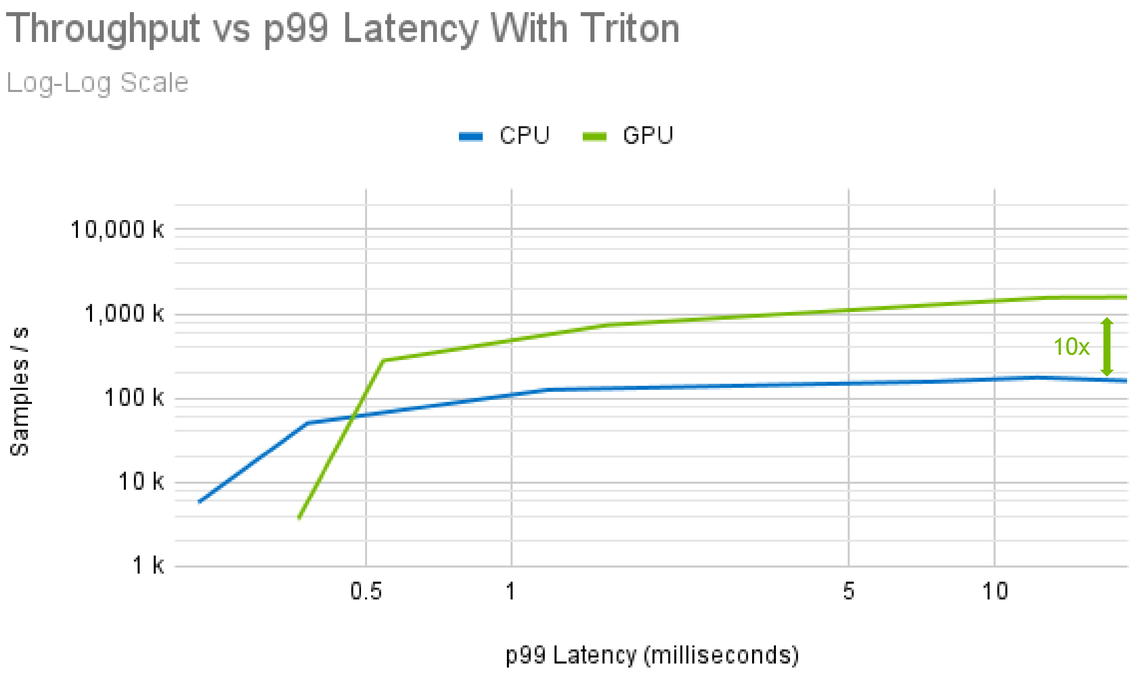
Blog and Notebook
NVIDIA Launchpad lab

Hybrid CPU/GPU libraries

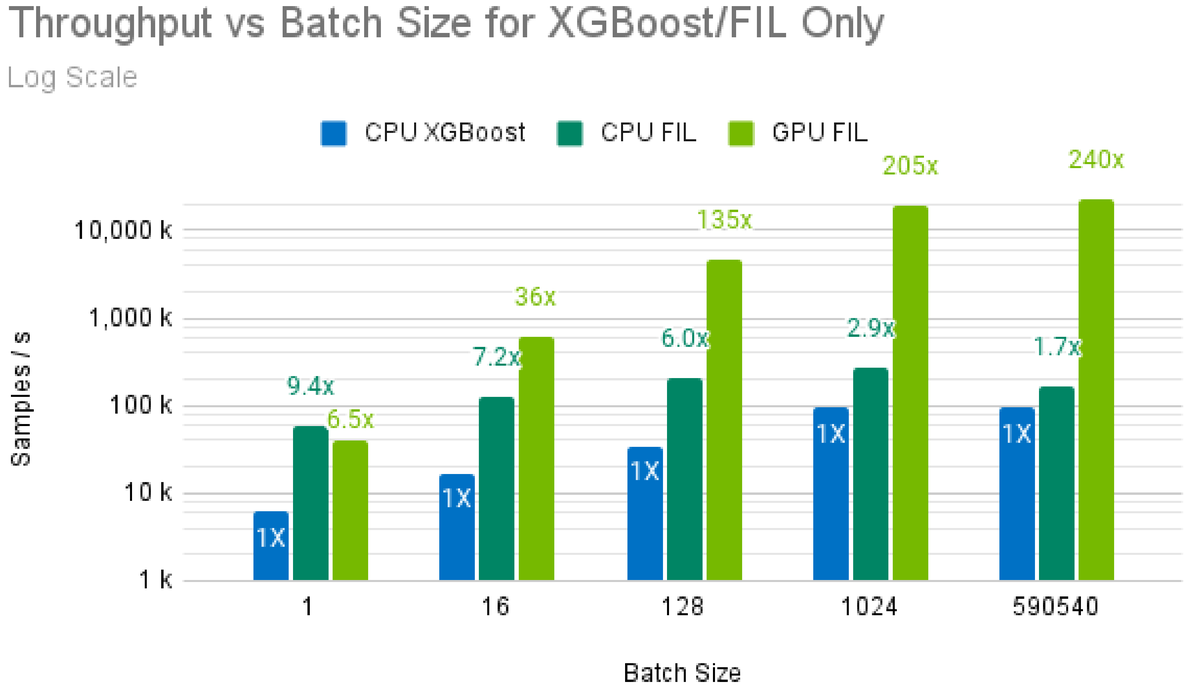
NVIDIA

Optimizing Throughput and Latency with Triton and FIL

Get maximum throughput for your latency budget with Triton



Throughput/latency curve for serving a ~200K node XGBoost model over GRPC with Triton



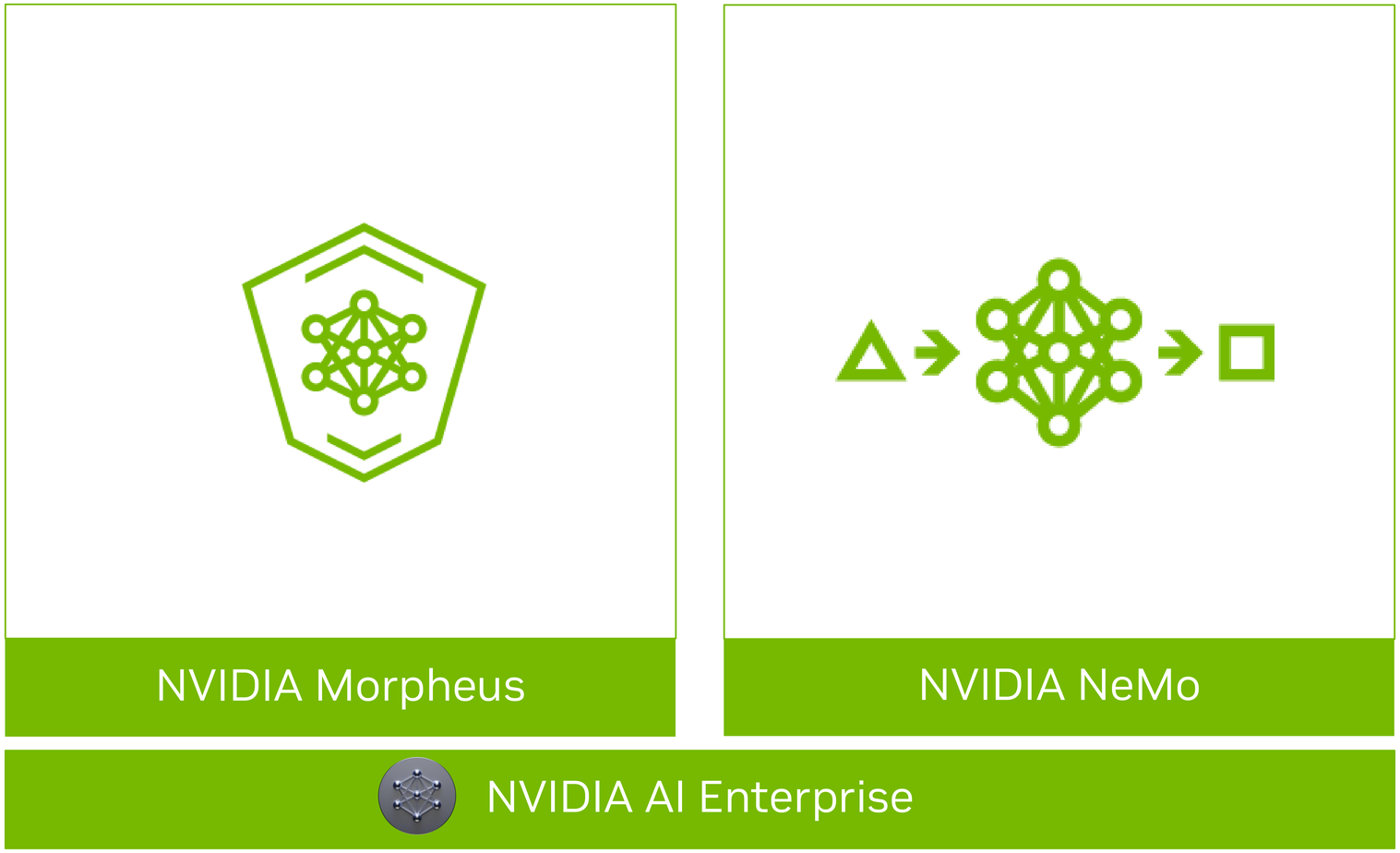
Performance of FIL library relative to CPU XGBoost

GPU benchmarks run on a GCP a2 instance with single A100; CPU benchmarks run on a GCP n2-standard-16
Max model depth: 10, Features: 393, Trees: 1000
Model trained on IEEE Kaggle Fraud Detection dataset

NVIDIA Software Stack

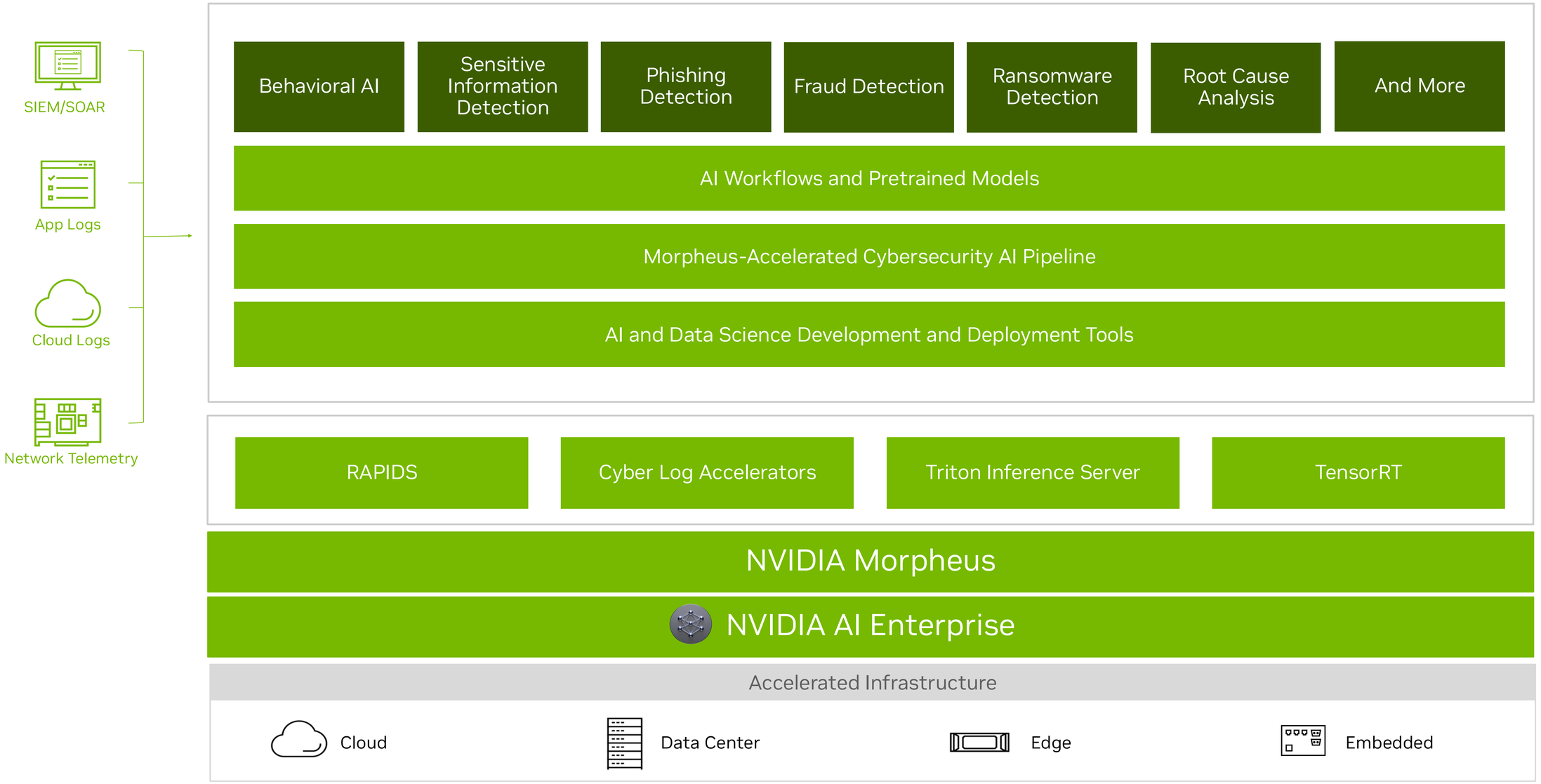
Overcome Challenges in Cybersecurity

Generative AI unlocks new possibilities in modern cyber defense



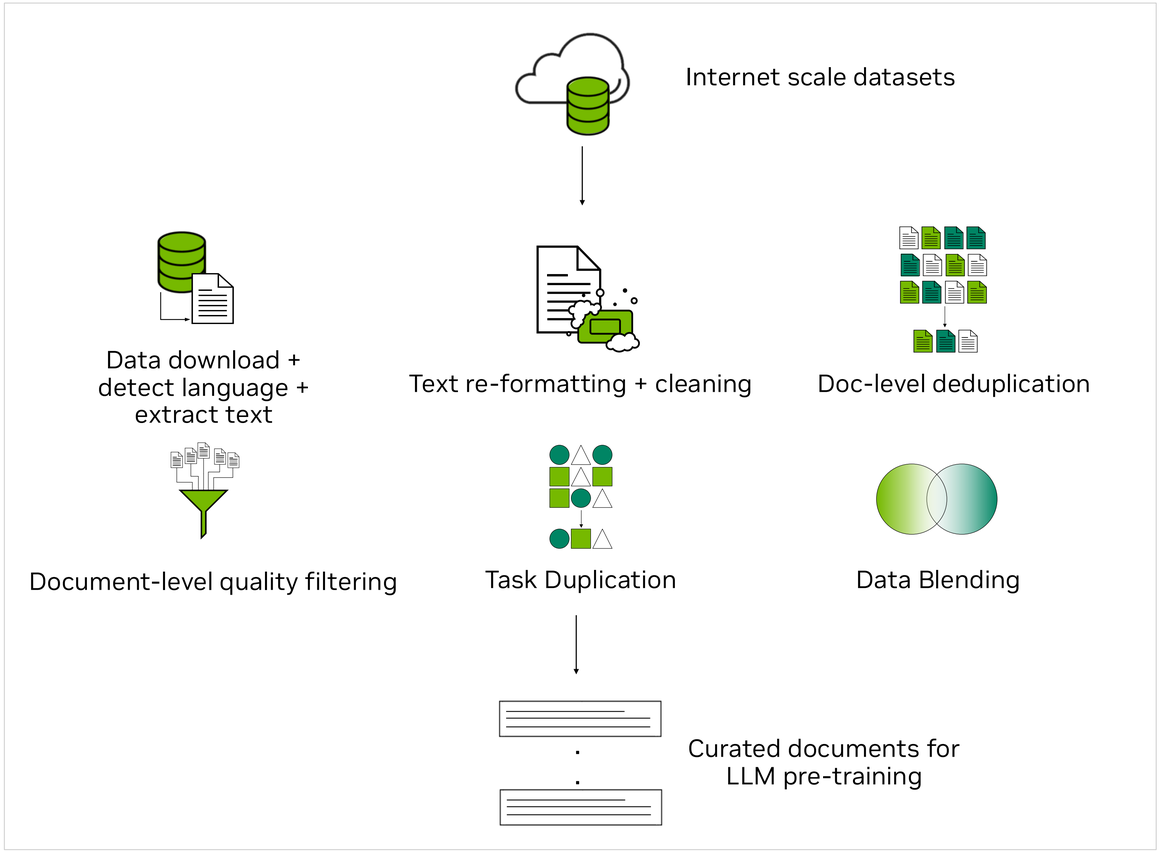
NVIDIA Morpheus Cybersecurity AI Framework

Automated, real-time threat detection at scale

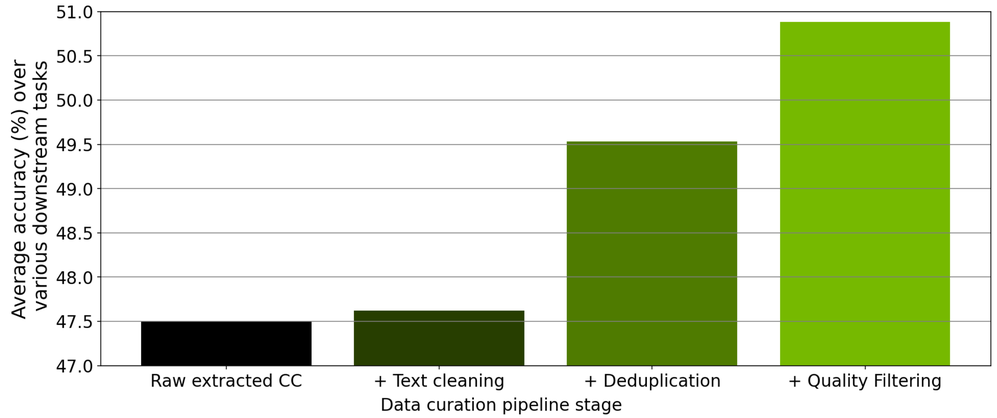


NeMo Curator

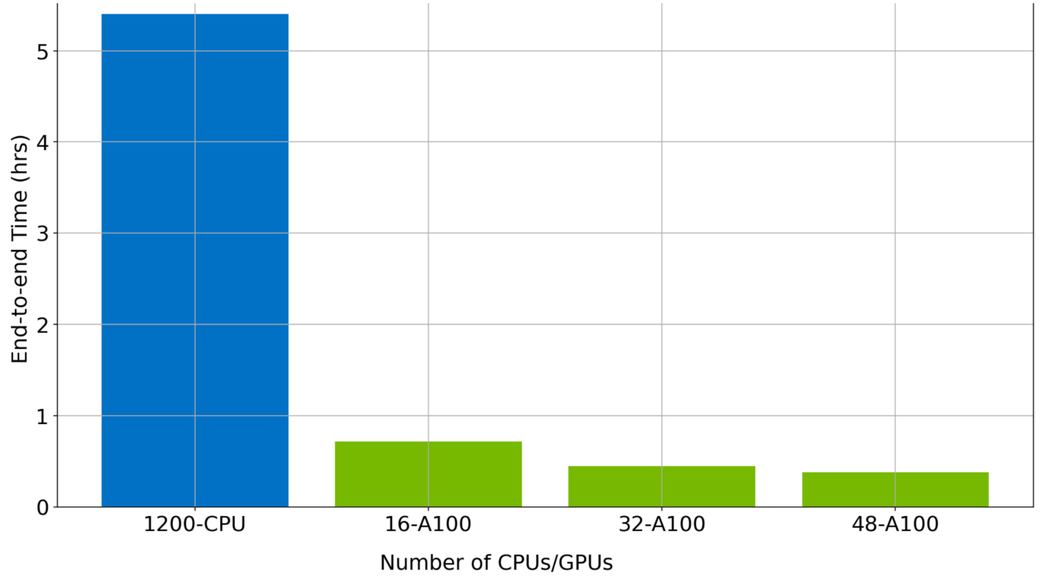
Python library of scalable data-mining modules for curating language data for training LLMs



Downstream task performance



Fuzzy Deduplication of 500 GB of Text Data

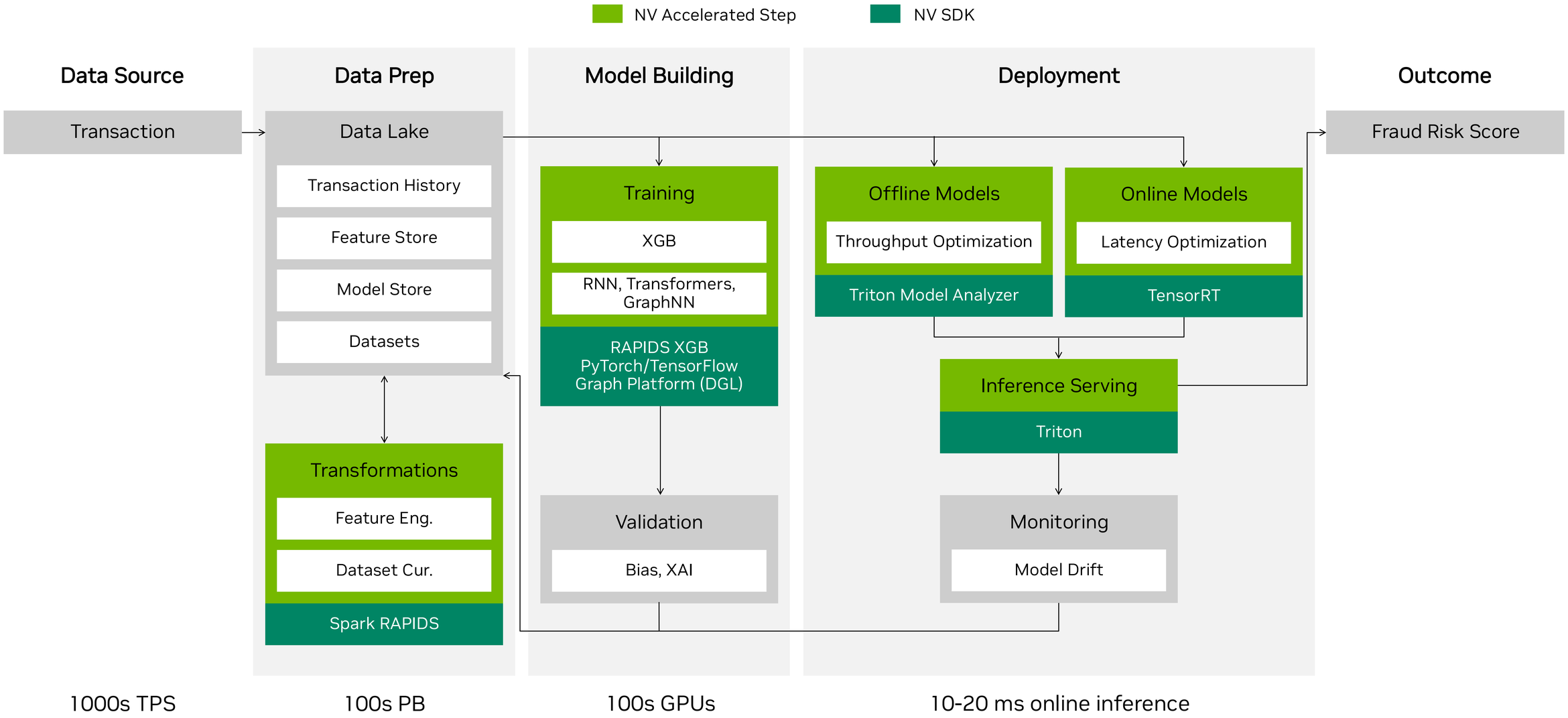


https://github.com/NVIDIA/NeMo-Curator

https://catalog.ngc.nvidia.com/orgs/nvidia/containers/nemo

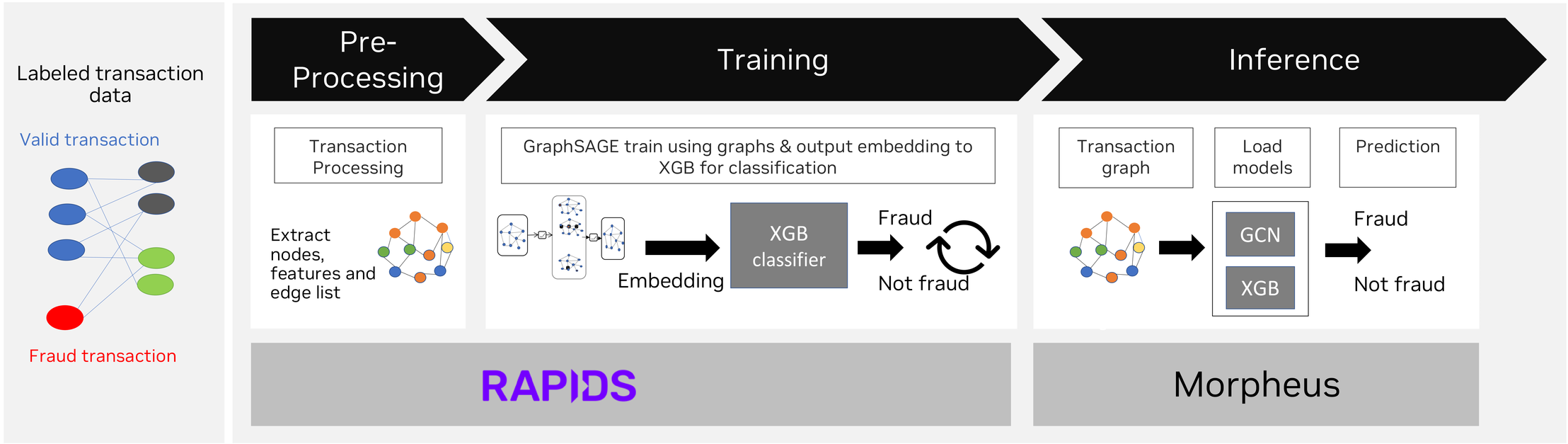
Accelerating Fraud Detection

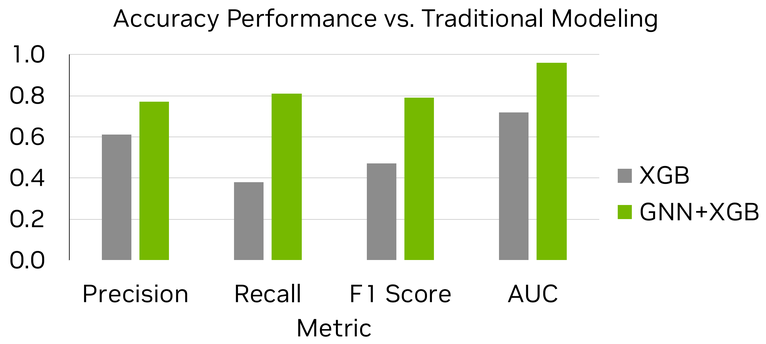
Transaction Fraud Detection Workflow Example

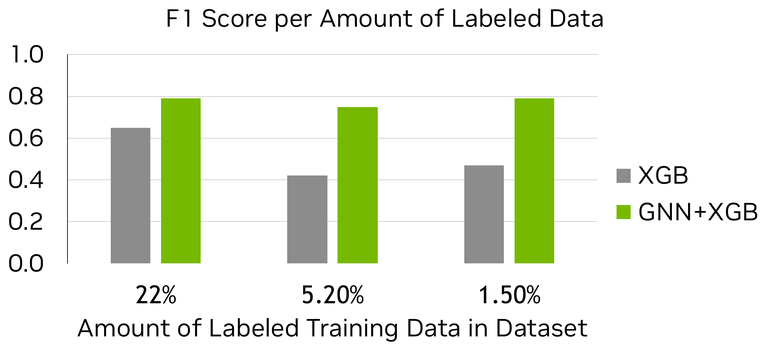


FRAUD DETECTION

Morpheus Use Case Using GNNs and XGBoost for Rapid Detection of Fraudulent Transactions





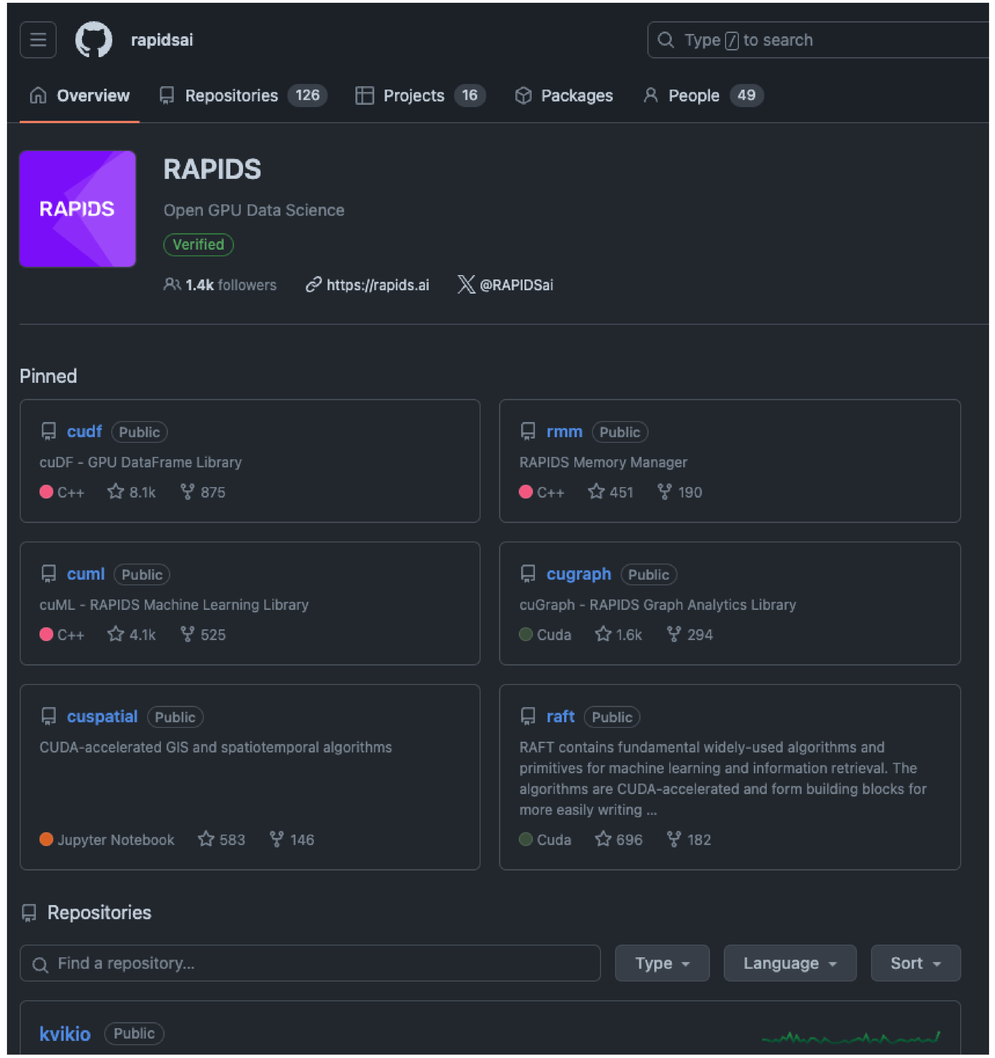


How to Get Started

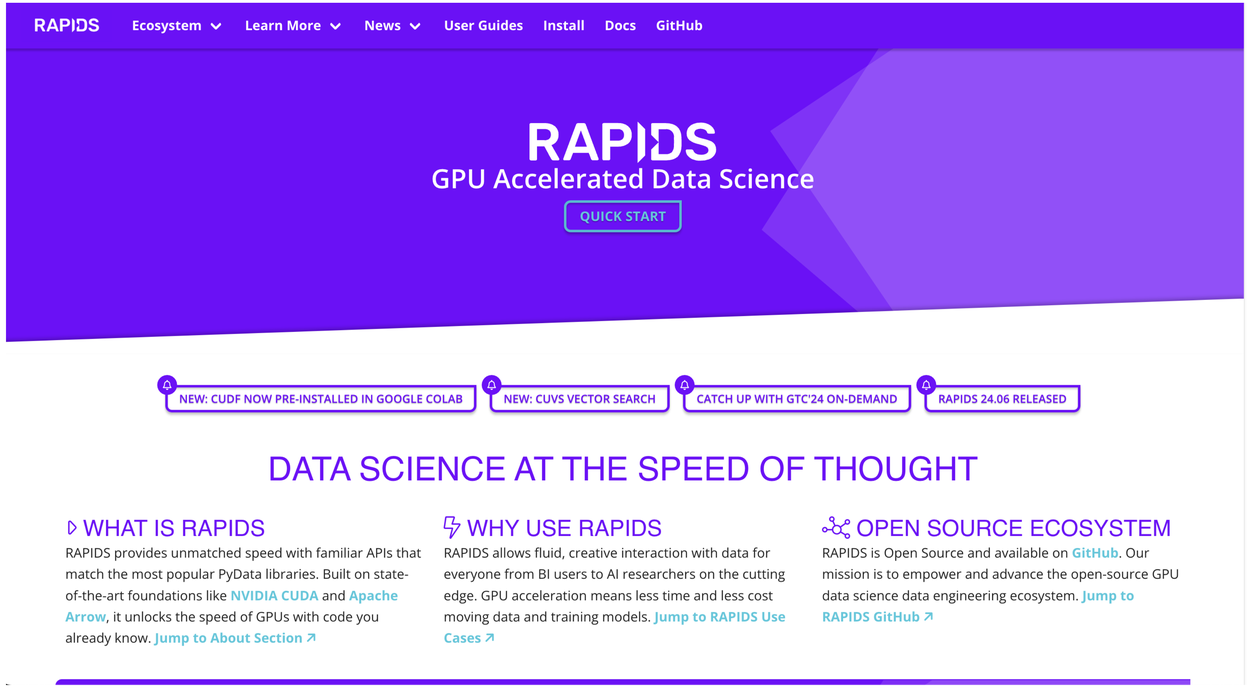
RAPIDS GitHub

Get started with the RAPIDS GitHub repository

https://github.com/rapidsai



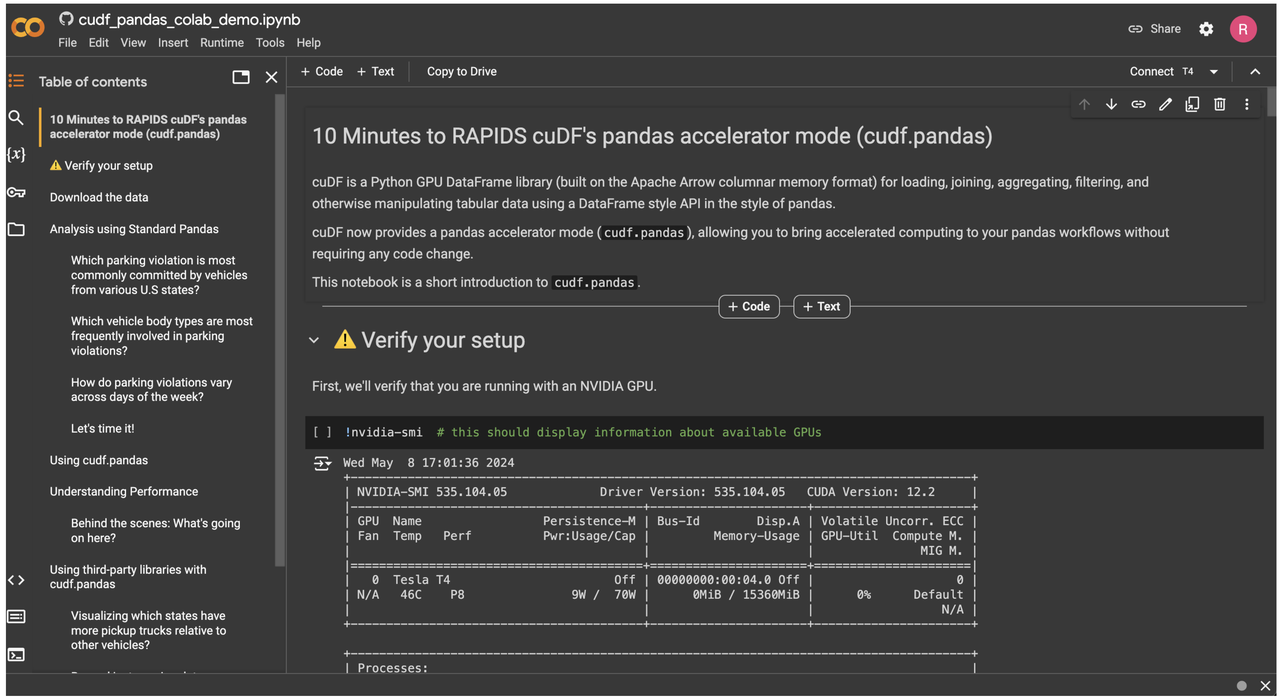
https://rapids.ai/



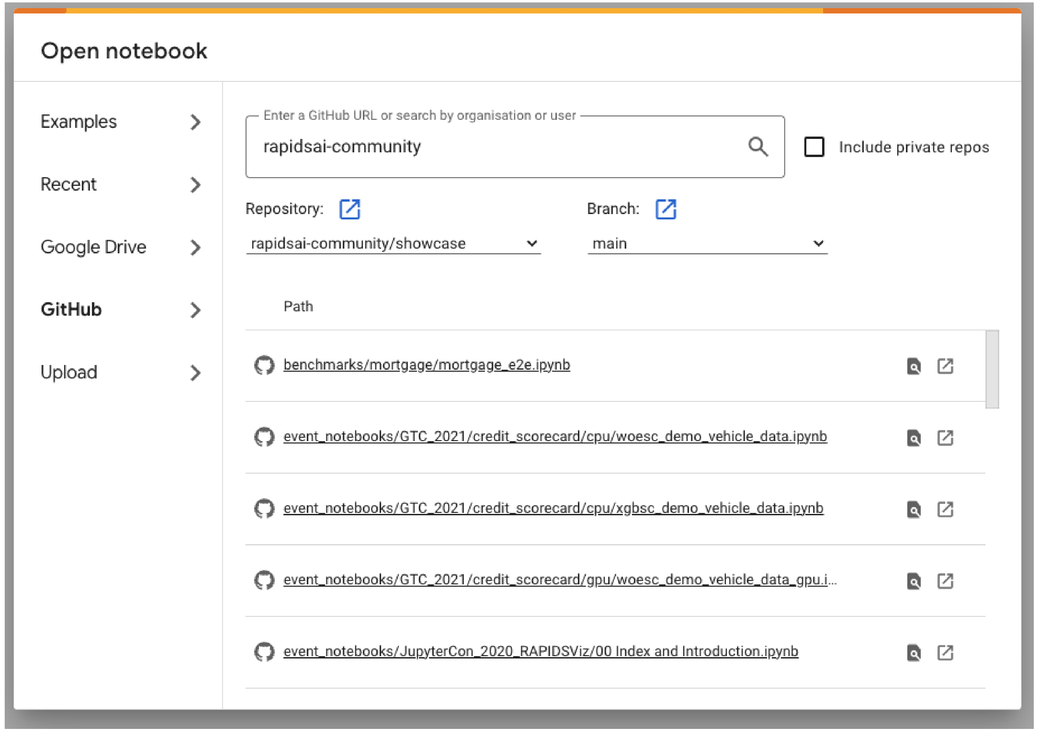
Google Colab

Deploy wherever you run

https://colab.research.google.com/github/rapidsai-community/showcase/blob/main/getting_started_tutorials/cudf_pandas_colab_demo.ipynb



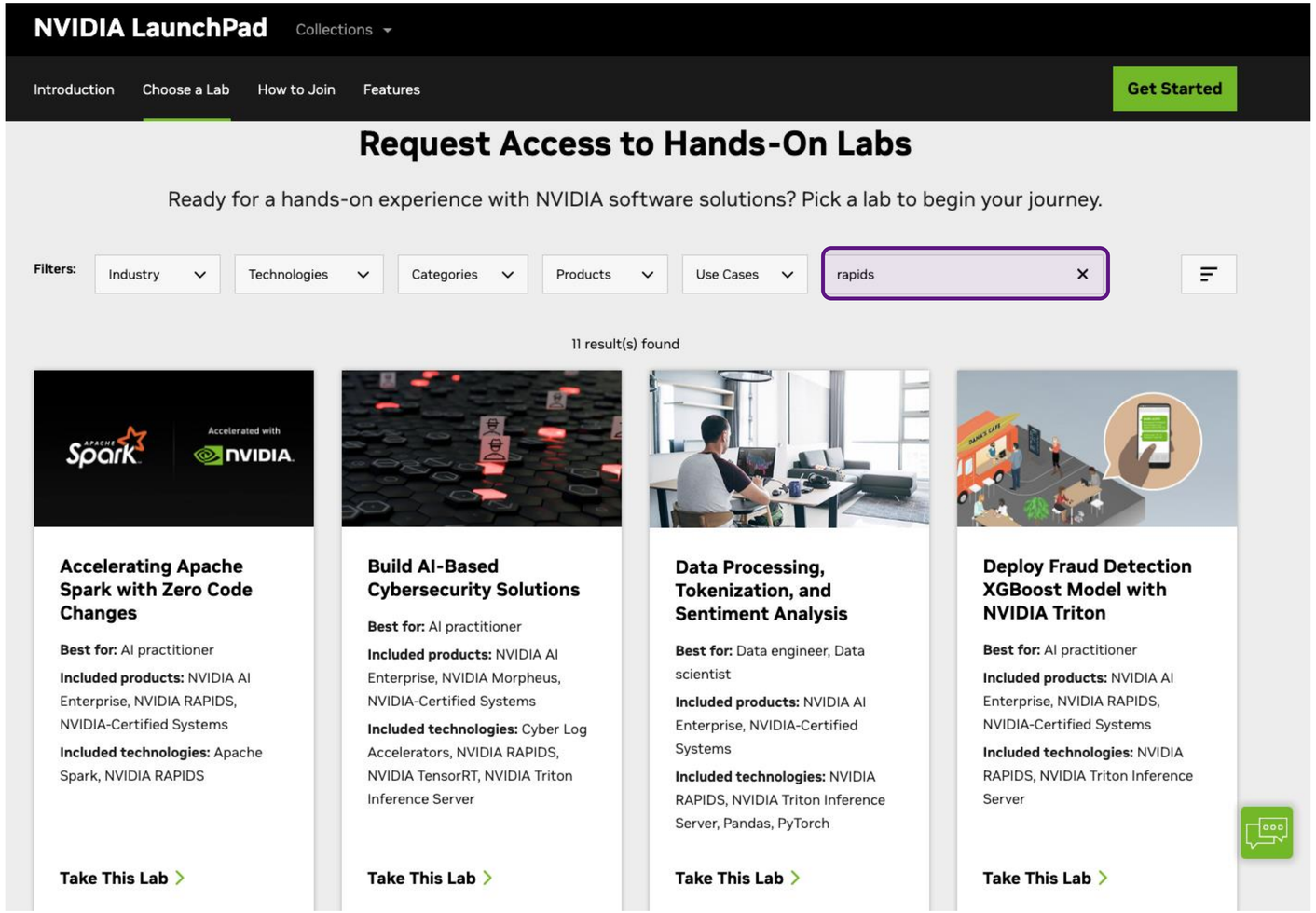
https://colab.research.google.com/github/rapidsai-community/showcase/blob/



NVIDIA Launchpad

Deploy wherever you run

https://www.nvidia.com/en-us/launchpad/



Smooth Installation and Packaging

Meeting a wide array of needs with easy to install and use distribution methods

Standard binary distributions

- One line installation methods on rapids.ai
  - *Conda*:
    - With conda-forge CUDA packages, and rapidsai RAPIDS packages
  - *Pip* installation
    - Custom index to support all RAPIDS ecosystem.

python™ Package Index

CONDA®

Containers

- Thin CUDA containers
- *RAPIDS* and other *application-level* containers
- *Devcontainers* for developers wanting to contribute
- *NVIDIA AI Enterprise*

NVIDIA NGC

docker

Source code on GitHub

https://github.com/rapidsai

GitHub

Cloud Deployment

aws

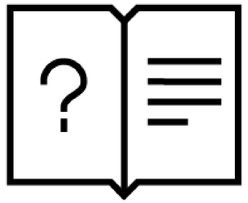
ORACLE Cloud Infrastructure

In the cloud

NVIDIA

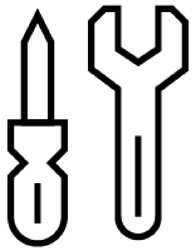
How to Get Started with RAPIDS

A Variety of Ways to Get Up & Running



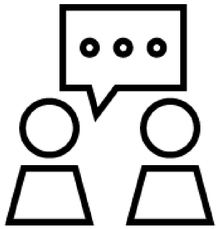
More about RAPIDS

- Learn more at RAPIDS.ai
- Read the API docs
- Check out the RAPIDS blog
- Read the NVIDIA DevBlog



Self-Start Resources

- Get started with RAPIDS
- Deploy on the Cloud today
- Start with Google Colab
- Look at the cheat sheets



Discussion & Support

- Check the RAPIDS GitHub
- Use the NVIDIA Forums
- Reach out on Slack
- Talk to NVIDIA Services

Get Engaged

@RAPIDSai

https://github.com/rapidsai

https://rapids.ai/slack-invite/

RAPIDS

https://rapids.ai

NVIDIA

renzmann@nvidia.com