BLAZINGDB
Data Lake to AI on GPUs
CPUs can no longer handle the growing data demands of data science workloads.

- **Slow Process**
  - Preparing data and training models can take days or even weeks.

- **Suboptimal Infrastructure**
  - Hundreds to **tens of thousands of CPU servers** are needed in data centers.
GPUs are well known for accelerating the training of machine learning and deep learning models.

Performance improvements increase at scale.

40x Improvement over CPU.
But data preparation still happens on CPUs, and can’t keep up with GPU accelerated machine learning.

- Apache Spark
  - Query
  - ETL
  - ML Train

- Apache Spark + GPU ML
  - Query
  - ETL
  - ML Train

Enterprise GPU users find it challenging to "Feed the Beast".
An end-to-end analytics solution on GPUs is the only way to maximize GPU power.

**Expertise:**
- GPU DBMS
- GPU Columnar Analytics
- Data Lakes

**Expertise:**
- CUDA
- Machine Learning
- Deep Learning

**Expertise:**
- Python
- Data Science
- Machine Learning
RAPIDS, the end-to-end GPU analytics ecosystem

A set of open source libraries for GPU accelerating data preparation and machine learning.

```
import cudf
from cuml import KNN
import numpy as np

np_float = np.array([
    [1,2,3],  # Point 1
    [1,2,3],  # Point 2
    [1,2,3]   # Point 3
]).astype('float32')

gdf_float = cudf.DataFrame()
gdf_float['dim_0'] = np.ascontiguousarray(np_float[:,0])
gdf_float['dim_1'] = np.ascontiguousarray(np_float[:,1])
gdf_float['dim_2'] = np.ascontiguousarray(np_float[:,2])

print('n_samples = 3, n_dims = 3')
print(gdf_float)

knn_float = KNN(n_gpus=1)
knn_float.fit(gdf_float)
Distance,Index = knn_float.query(gdf_float,k=3)
# Get 3 nearest neighbors
print(Index)
print(Distance)
```
BlazingSQL: The GPU SQL Engine on RAPIDS

A SQL engine built on RAPIDS. Query enterprise data lakes lightning fast with full interoperability with the RAPIDS stack.
BlazingSQL, The GPU SQL Engine for RAPIDS

A SQL engine built on RAPIDS. Query enterprise data lakes lightning fast with full interoperability with RAPIDS stack.

```python
from blazingsql import BlazingContext
bc = BlazingContext()

# Register Filesystem
bc.hdfs('data', host='129.13.0.12', port=54310)

# Create Table
bc.create_table('performance', file_type='parquet', path='hdfs://data/performance/

# Execute Query
result_gdf = bc.run_query('SELECT * FROM performance WHERE YEAR(maturity_date)>2005')
print(result_gdf)
```
Getting Started Demo

Getting Started with BlazingSQL on RAPIDS AI

Import Package
```python
import blazing.sql as blazingsql
```

Create a BlazingSQL Context
```python
bc = blazingsql.make_context()
```

Register a FileSystem
```python
bc = bc.append_filesystem("workspace", path="...", file_type="...")
```

Create a Table
```python
bc.create_table("netflow", '/.../...')
```

Query a Table
```python
result = bc.execute("SELECT * FROM main.netflow", ["netflow"]
result.select_columns
print(result.csv)
```
BlazingSQL + XGBoost Loan Risk Demo

Train a model to assess risk of new mortgage loans based on Fannie Mae loan performance data

**Mortgage Data**
4.22M Loans
148M Perf. Records
CSV Files on HDFS

**ETL/Feature Engineering**

**XGBoost Training**

**Spark**
4 Nodes
8 vCPUs per node
30GB RAM

**blazingSQL**
1 Node
16 vCPUs per node
1 Tesla T4 GPU
2560 CUDA Cores
16GB VRAM
RAPIDS + BlazingSQL outperforms traditional CPU pipelines

Demo Timings (ETL Phase)

<table>
<thead>
<tr>
<th>Dataset Size</th>
<th>System</th>
<th>Time in Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.8GB</td>
<td>blazingSQL (1 x T4)</td>
<td>0''</td>
</tr>
<tr>
<td>3.8GB</td>
<td>Spark (4 Nodes)</td>
<td>1000''</td>
</tr>
<tr>
<td>15.6GB</td>
<td>blazingSQL (1 x T4)</td>
<td>2000''</td>
</tr>
<tr>
<td>15.6GB</td>
<td>Spark (4 Nodes)</td>
<td>3000''</td>
</tr>
</tbody>
</table>
Scale up the data on a DGX
4 x V100 GPUs
BlazingSQL + Graphistry Netflow Analysis

Visually analyze the VAST netflow data set inside Graphistry in order to quickly detect anomalous events.

Netflow Data
- 65M Events
- 1,440 Devices
- 2 Weeks

ETL

Visualization
Benchmarks

Netflow Demo Timings (ETL Only)

- BlazingSQL
- Spark Tool
- Pandas
Benefits of BlazingSQL

**Data Lake to RAPIDS**
Query data from Data Lakes directly with SQL in to GPU memory, let RAPIDS do the rest.

**Blazing Fast.**
Massive time savings with our GPU accelerated ETL pipeline.

**Minimal Code Changes Required.**
RAPIDS with BlazingSQL mirrors Pandas and SQL interfaces for seamless onboarding.

**Stateless and Simple.**
Underlying services being stateless reduces complexity and increase extensibility.
Upcoming BlazingSQL Releases

- **V0.1**: Query GDFs
  Use the PyBlazing connection to execute SQL queries on GDFs that are loaded by the cuDF API.

- **V0.2**: Direct Query Flat Files
  Integrate FileSystem API, adding the ability to directly query flat files (Apache Parquet & CSV) inside distributed file systems.

- **V0.3**: String Support
  String support and string operation support.

- **V0.4**: Distributed Scheduler
  SQL queries are fanned out across multiple GPUs and servers.

- **V0.5**: Physical Plan Optimizer
  Partition culling for where clauses and joins.
BlazingSQL is quick to get up and running using either DockerHub or Conda Install: