Why use GPUs instead of CPUs?

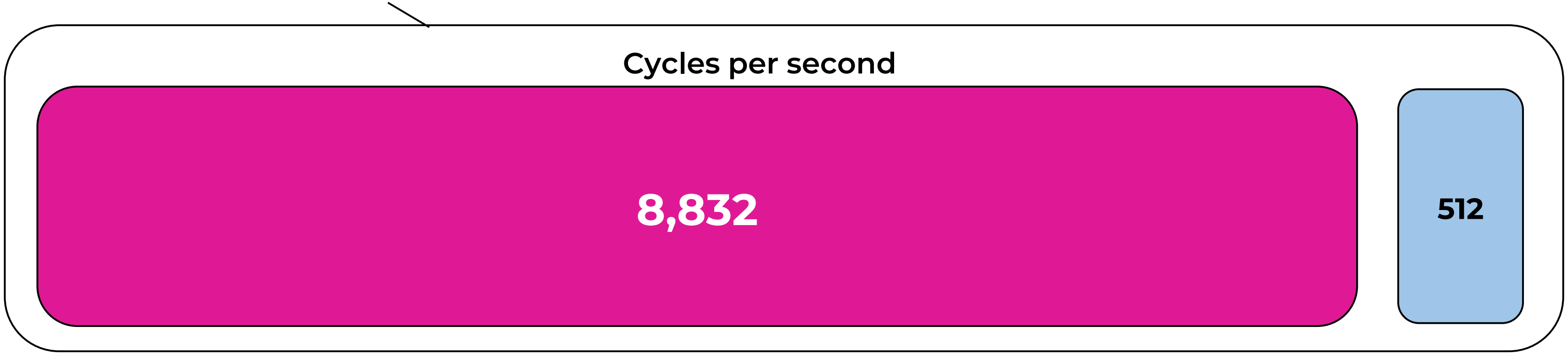
Short Answer:

They are faster. ***Way*** faster.

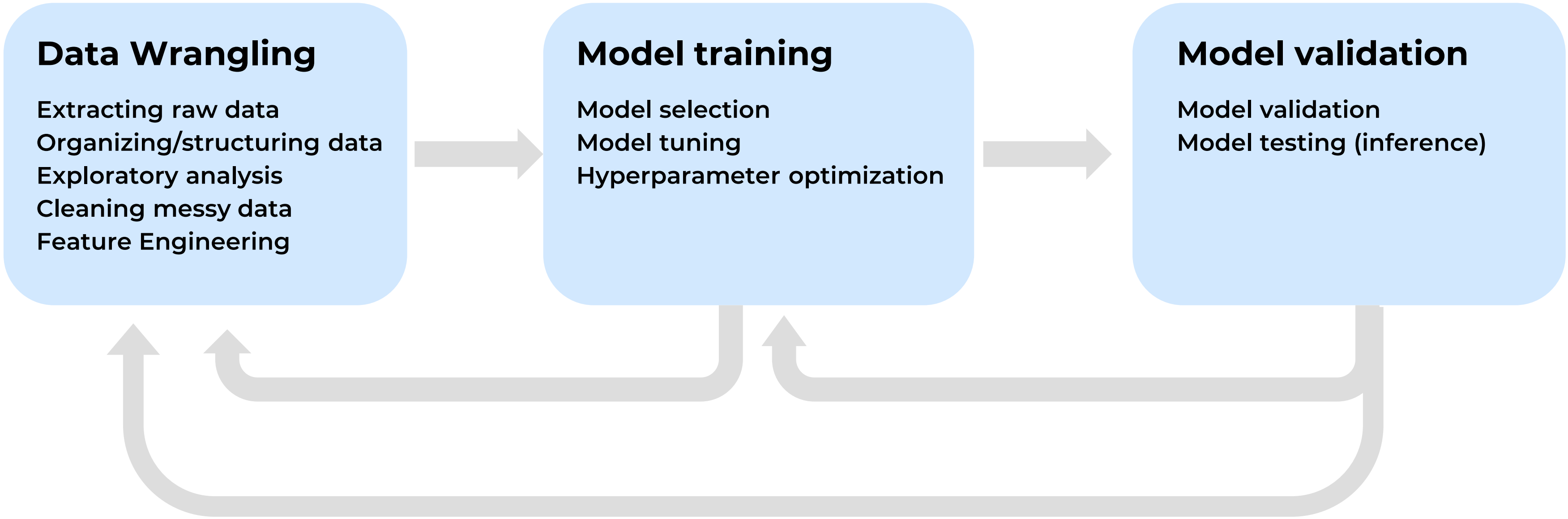
The “fastest” and “largest” **CPUs** today have speeds around **4 GHs** and about **128 cores**, and costs about **$5,000**

An “average” GPU like the RTX 3080, has core speeds of **1.5 GHs** and **5,888 cores**, and costs about **$1,000**

Size **not** to scale



Typical Data Science Workflows



Data Wrangling

Extracting raw data
Organizing/structuring data
Exploratory analysis
Cleaning messy data
Feature Engineering

Model training

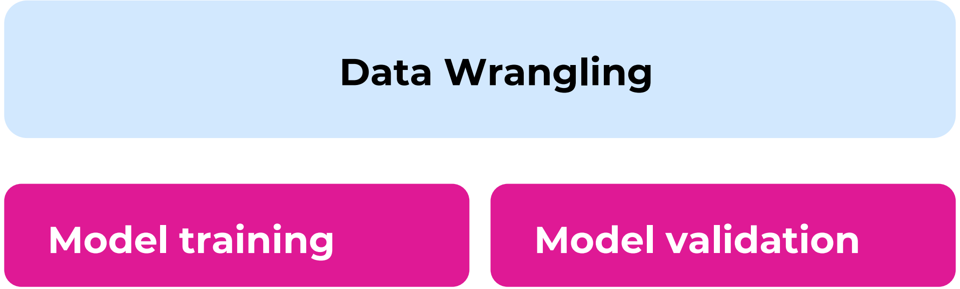
Model selection
Model tuning
Hyperparameter optimization

Model validation

Model validation
Model testing (inference)

GPU Acceleration

GPU acceleration is commonly concentrated in the modeling portions.



Data Wrangling

Model training

Model validation

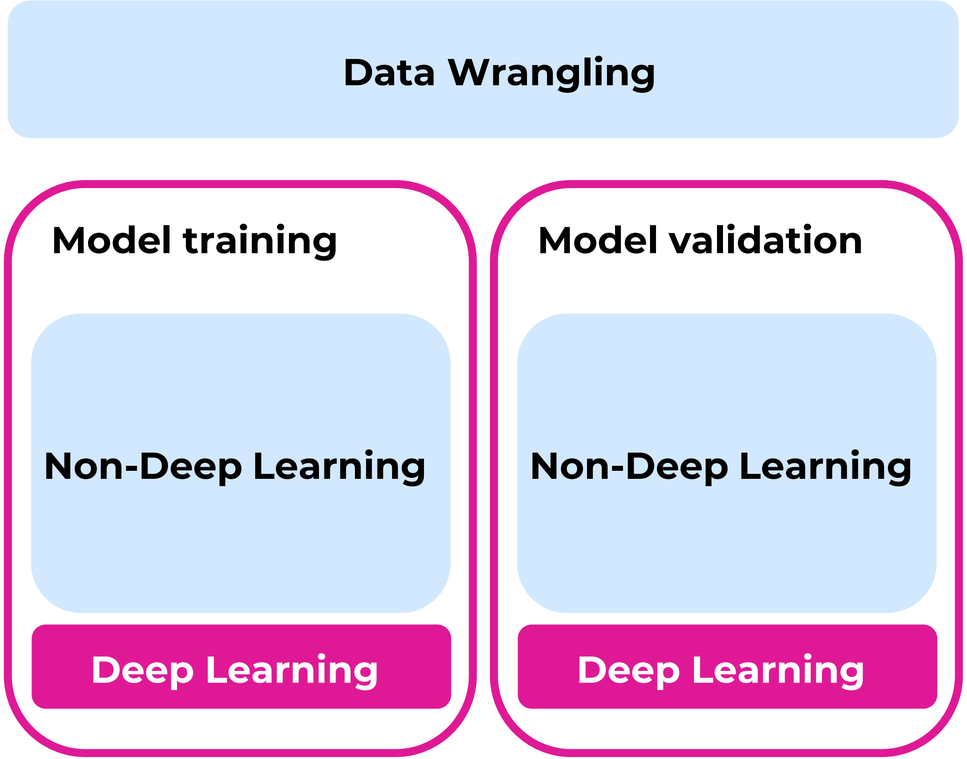
GPU Acceleration

GPU acceleration is commonly concentrated in the modeling portions.

Furthermore, that **concentration** tends to be in the **Deep Learning** and **Neural Networks** subfields.

This isn’t necessarily a bad thing; training a model in a matter of hours or days versus weeks is a huge improvement.

Still, it would be nice to be able to **leverage the GPU for non-deep learning tasks**....



Data Wrangling

Model training

Non-Deep Learning

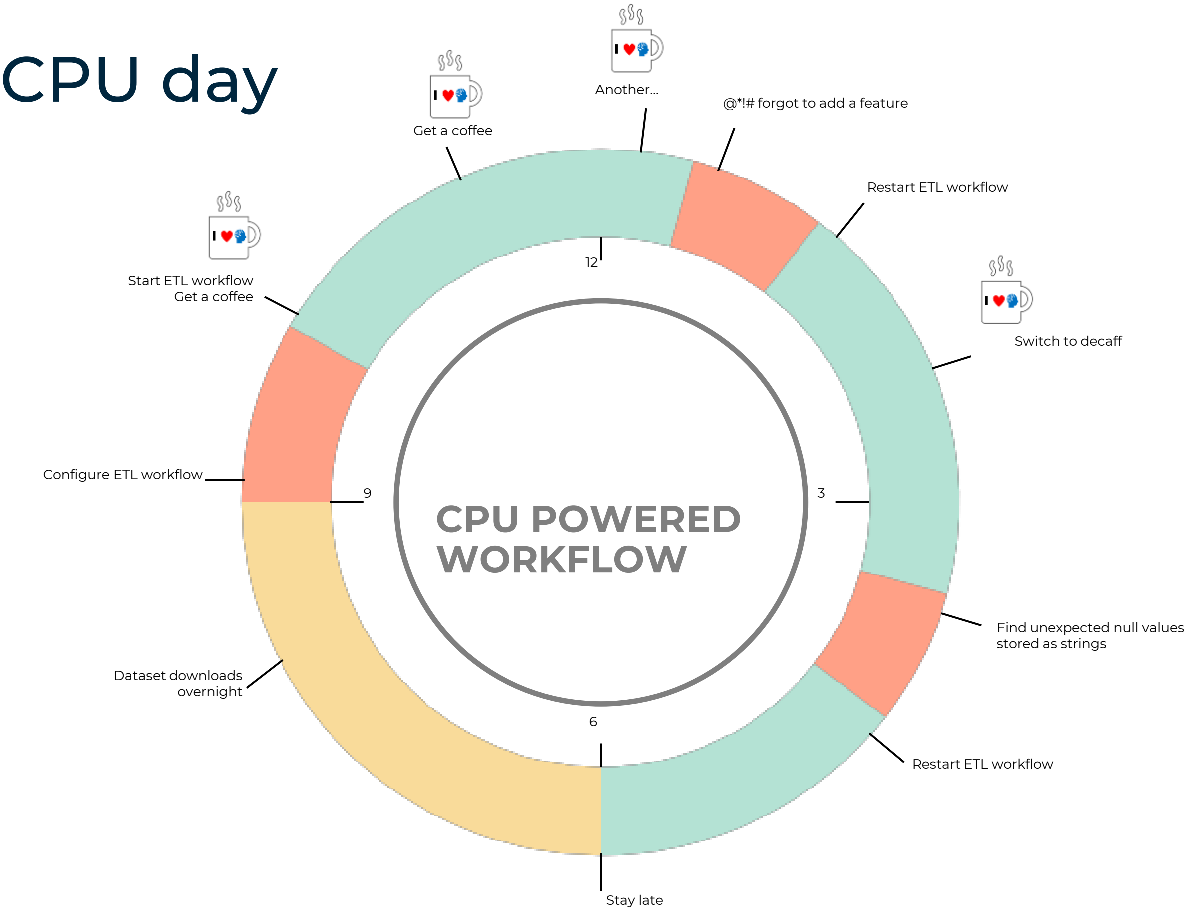
Deep Learning

Model validation

Non-Deep Learning

Deep Learning

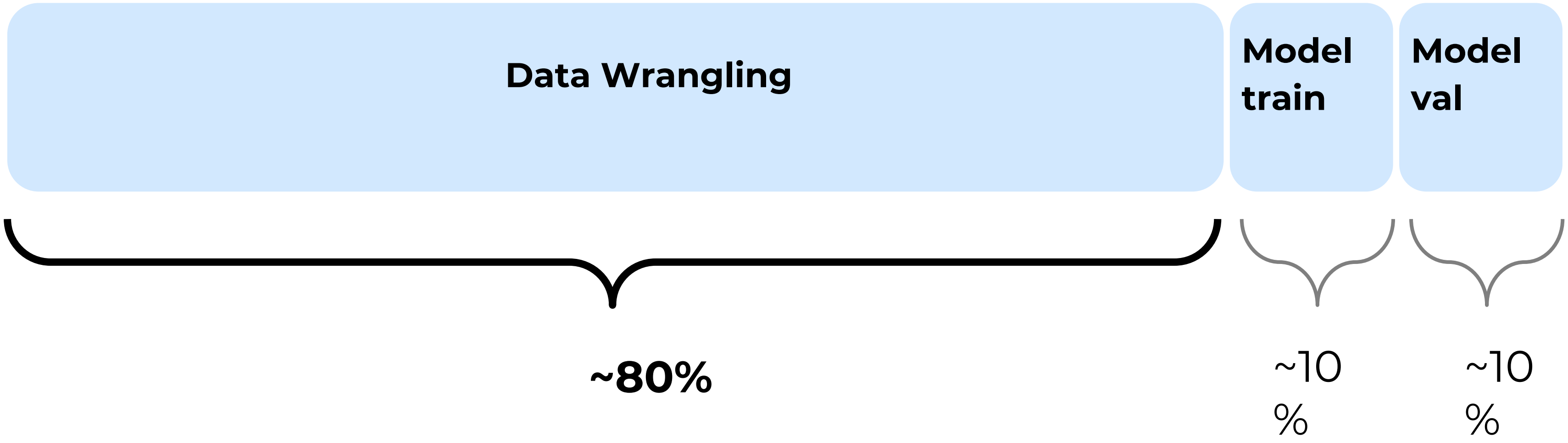
Typical CPU day



- 🟨 dataset collection
- 🟥 analysis
- 🟩 ETL
- ⬛ train
- 🟦 inference

https://developer.nvidia.com/blog/gpu-accelerated-analytics-rapids/

Points of frustration



Data Wrangling

Model train

Model val

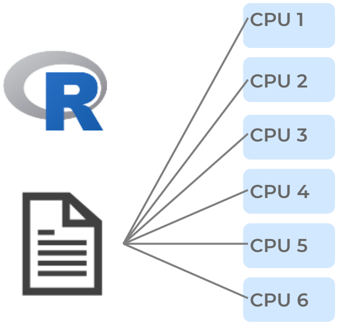
~80%

~10 %

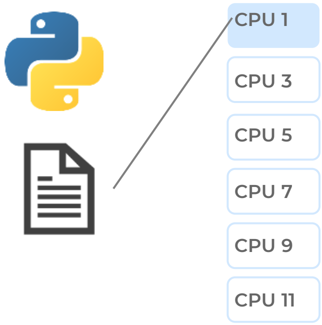
~10 %

At least **80% of my time** was spend wrestling with the data!
Very little was left for model training and validation!
A huge bottleneck in productivity!

Points of frustration



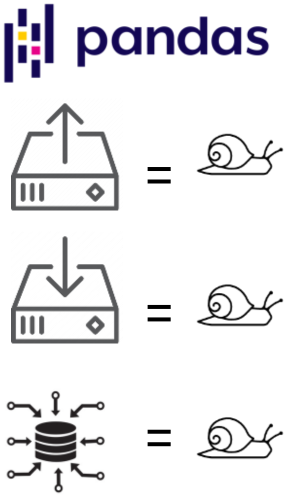
CPU 1
CPU 2
CPU 3
CPU 4
CPU 5
CPU 6



CPU 1
CPU 3
CPU 5
CPU 7
CPU 9
CPU 11

Some languages, like R, will use all available CPU cores. Python, by purposeful design, is limited to a single CPU.

I can get around this by using python's multiprocessing module, which *mimics* multiprocessing, but it's still an extra step.



pandas

Pandas has notoriously **slow** read, write, and aggregation speeds

A wonderful unified solution

The Rapids.ai is a suite of GPU accelerated libraries for python and include:

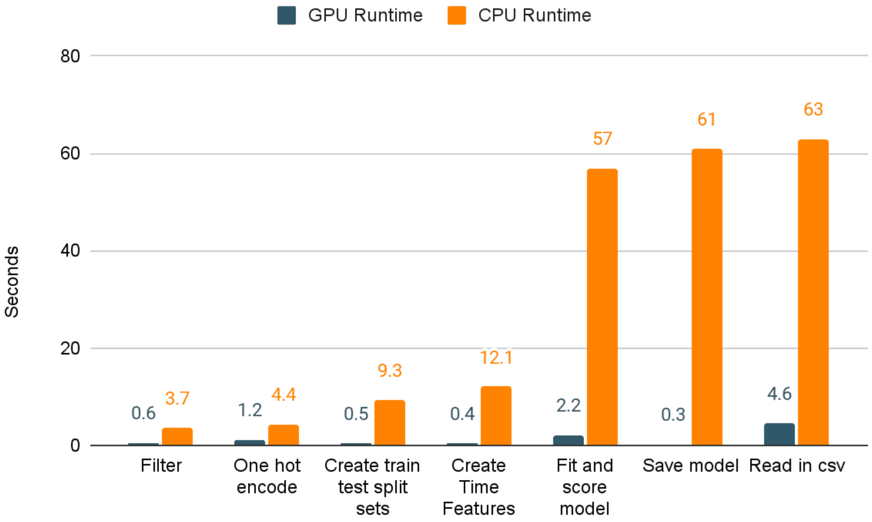
- cuDF: mirroring Pandas
- cuML: mirroring Scikit-learn
- cuSignal: mirroring SciPy
- cuXFilter: framework for visualization
- cuGraph: mirroring NetworkX
- and more!

In addition XGBoost, Dask, and Spark all have GPU implementations that utilize the Rapids engine.

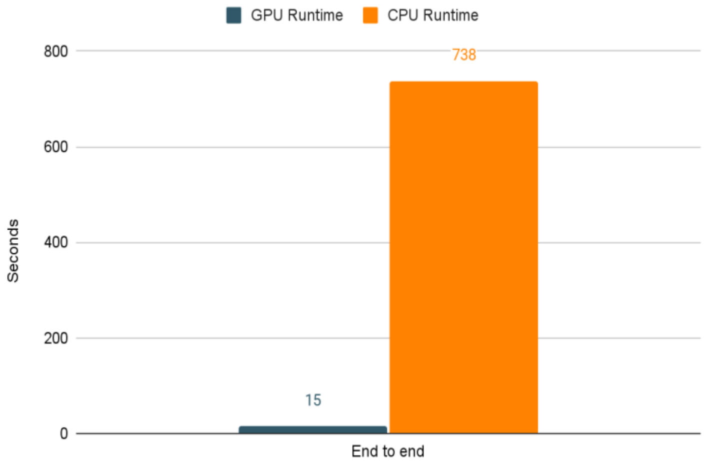
Comparing speeds: trip data for NYC taxis

I came across these on the internet…

Individual Tasks



Entire pipeline



…and had to test it out myself….

https://blog.cloudera.com/switching-from-cpus-to-gpus-for-nyc-taxi-fare-predictions-with-nvidia-rapids/

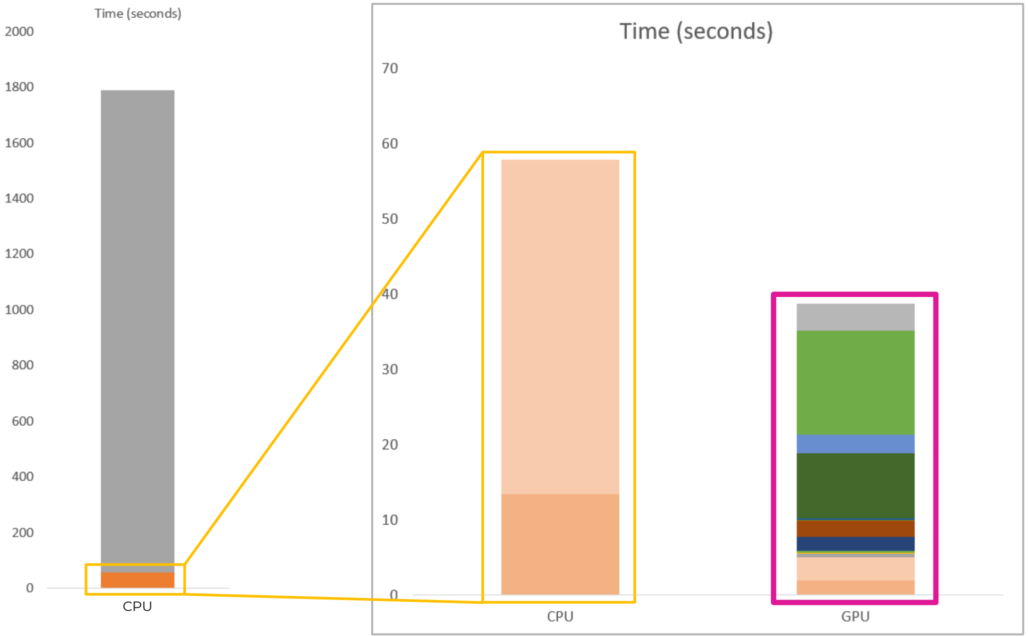
Benchmark: 20M rows & 10 columns

On CPU, in the time that it took to:

- ■ Load data
- ■ Write date

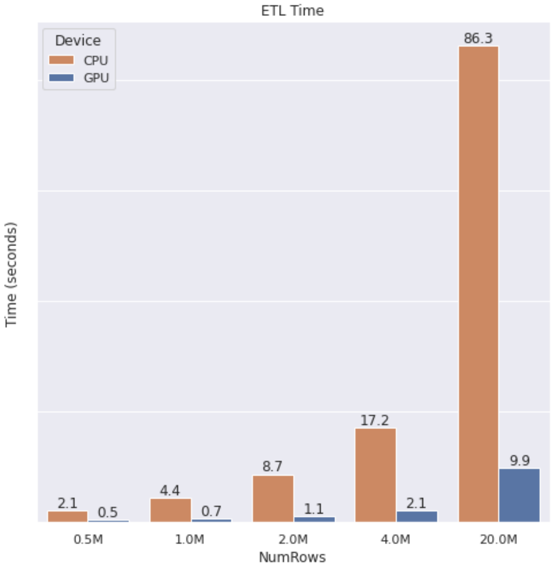
The GPU accomplished:

- ■ Load data
- ■ Write date
- ■ Describe dataframe
- ■ Set index on dataframe
- ■ Concat multiple dataframes
- ■ Groupby aggregation (mean)
- ■ Fit label encoder
- ■ Encode data
- ■ Scale data
- ■ split data
- ■ OLS Regression
- ■ Logistic Regression
- ■ K-Means
- ■ Random Forest Classifier
- ■ Gradient Boosting

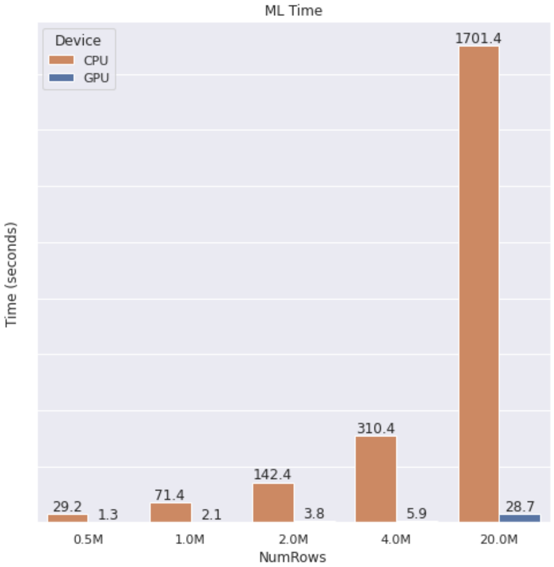


https://github.com/jonathancosme/GPU_vs_CPU

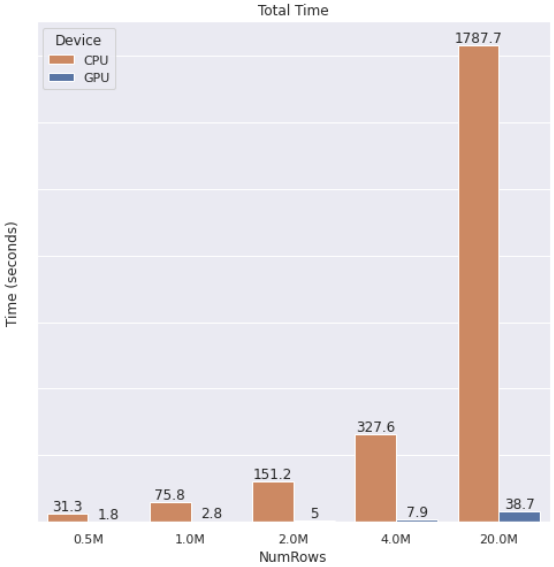
Benchmark: across different file sizes



For **ETL**, we see more than **85% reduction in time**



For **ML**, we see a more than **98% reduction in time**

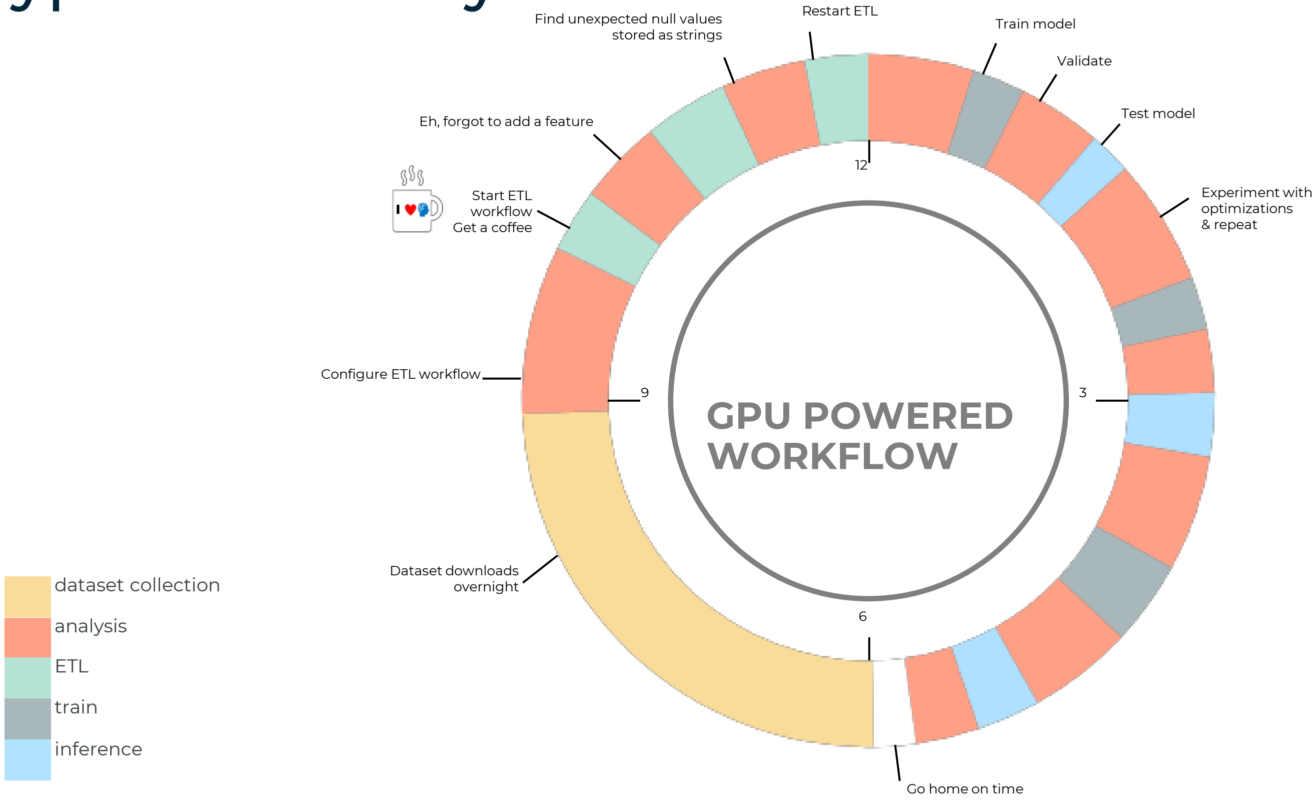


End-to-end, we see a more than **95% reduction in time**

The speed gain are **astounding**!

https://github.com/jonathancosme/GPU_vs_CPU

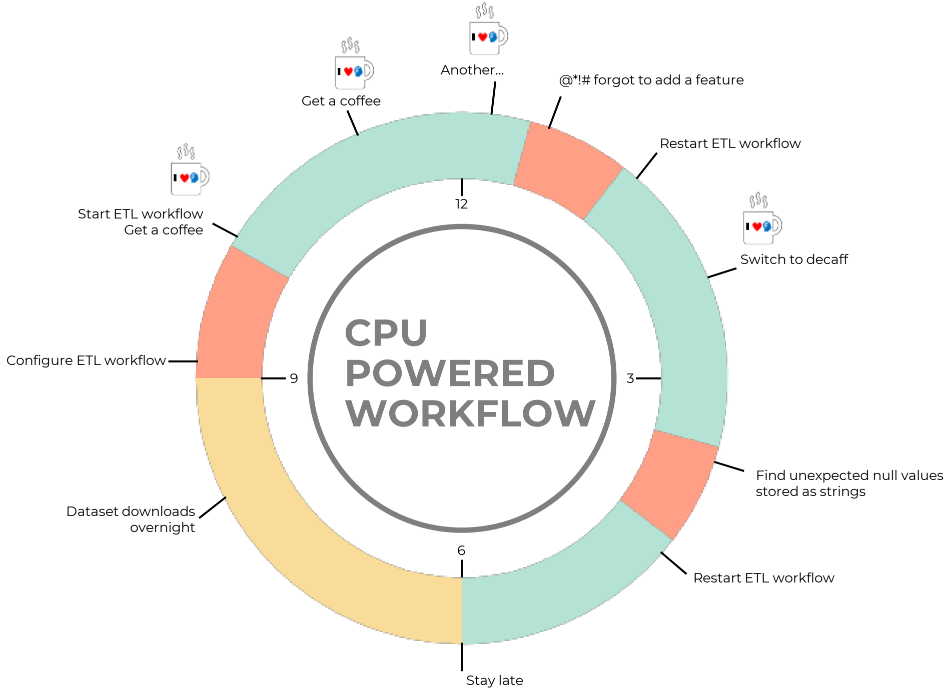
Typical GPU day

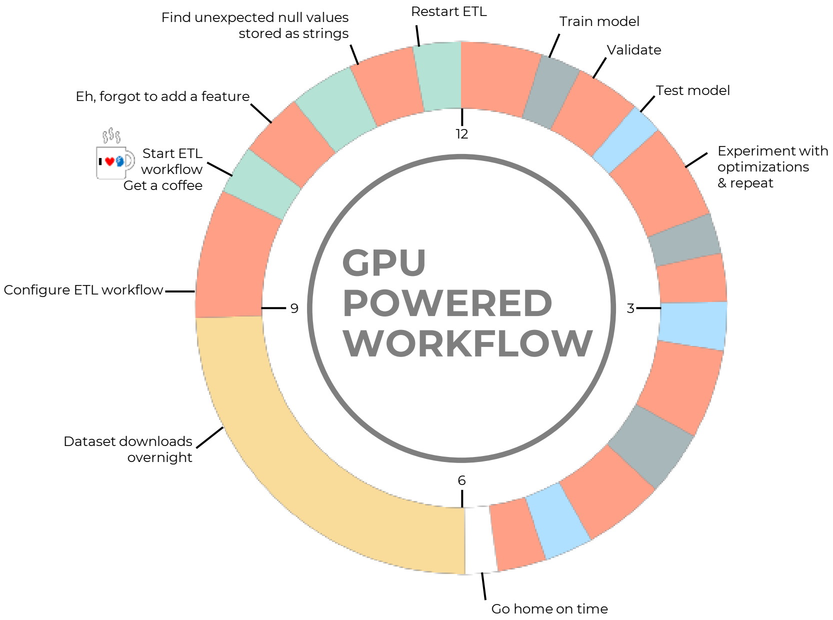


https://developer.nvidia.com/blog/gpu-accelerated-analytics-rapids/

Comparing CPU vs GPU day

- 🟨 dataset collection
- 🟧 analysis
- 🟩 ETL
- ⬛ train
- 🟦 inference

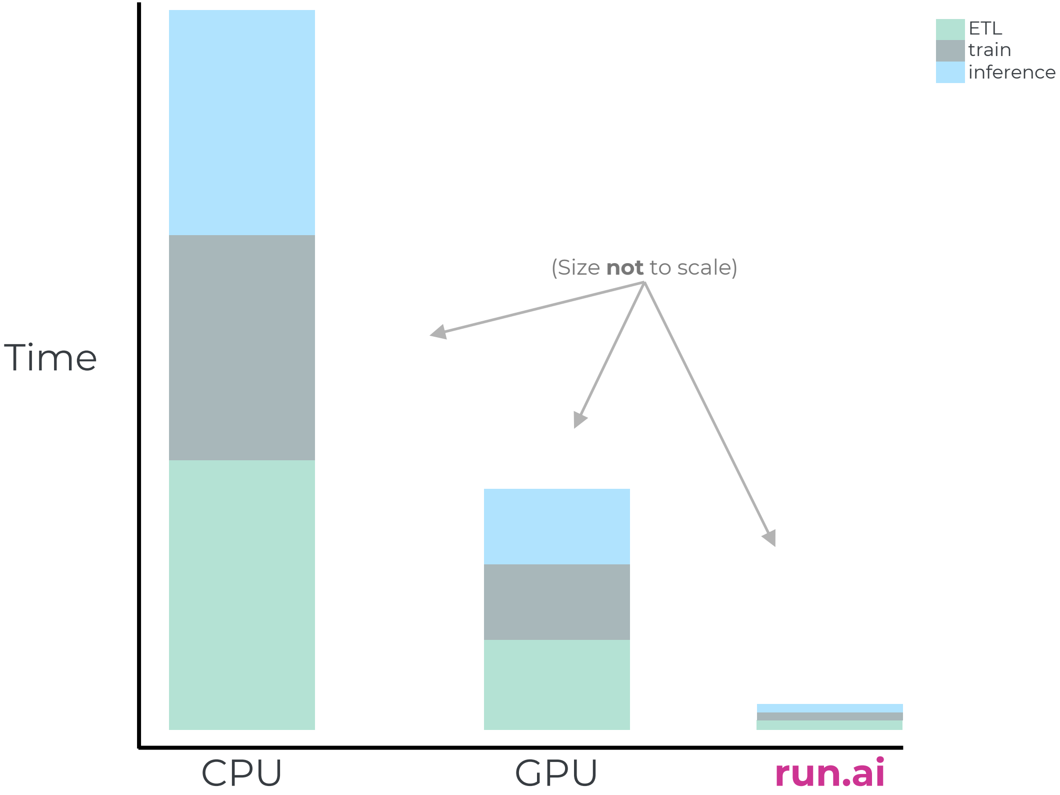




https://developer.nvidia.com/blog/gpu-accelerated-analytics-rapids/

Speed gains with run.ai

Using run.ai's state-of-the-art **scheduler** and **fractional GPU** capabilities, speed gains can be pushed to their maximum



FAQs

FAQs

All of my code is already in Pandas and Scikit-learn. Do I have to convert it all?

A valid concern!

cuDF was specifically build to mirror Pandas.

cuML was specifically build to mirror Scikit-Learn.

In nearly all cases, your Pandas/Scikit-Learn code is the same as the cuDF/cuML code!

FAQs

What if there’s something I *must* use pandas for?

No problem!
It's easy to convert back and forth between Pandas Dataframes and cuDF Dataframes!

```
# from cuDF to Pandas
df = df.to_pandas()

# your unique function
df = my_function(df)

# from Pandas to cuDF
df = cudf.from_pandas(df)
```

FAQs

What if my data is too large for GPU memory?

No problem!

The Rapids ecosystem includes Dask-GPU, which is designed to handle extremely large datasets, in a distributed way.

Best of all, Dask was originally designed to be a distributed extension of Pandas, so implementing it will be very easy if you are used to pandas!

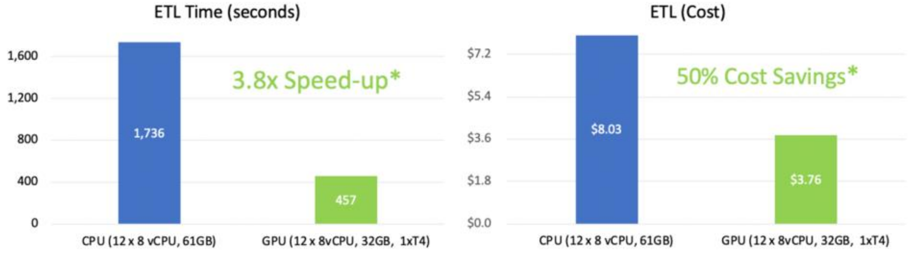
If you are used to using Spark, Apache has also incorporated the Rapids.ai engine to leverage GPU for ETL pipelines.

FAQs

GPU instances are expensive. Wouldn't it be cheaper to run a large CPU cluster rather than a GPU?

Actually, because a GPU can process data so much more quickly, the net cost of running a workflow on a GPU vs a large CPU cluster is actually less!

Rapids Accelerator for Apache Spark reaps the benefit of GPU performance while saving infrastructure costs.



*ETL for FannieMae Mortgage Dataset (~200GB) as shown in our demo. Costs based on Cloud T4 GPU instance market price & V100 GPU price on Databricks Standard edition

https://nvidia.github.io/spark-rapids/

Use GPUs!