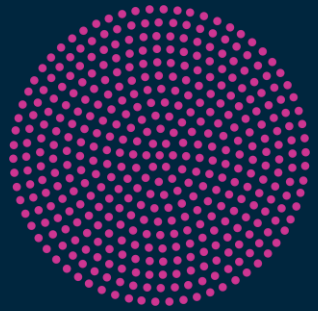
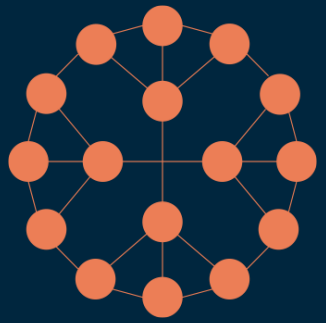
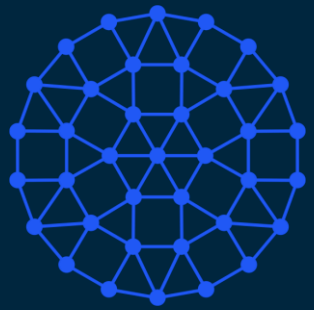


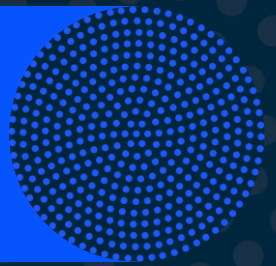




Why run:ai?

November | 2022

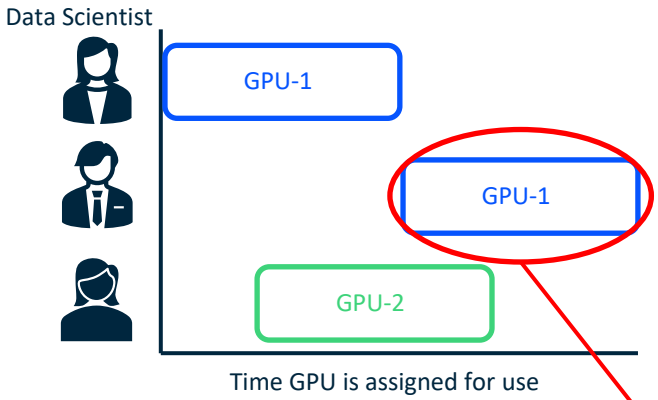
Utilization vs Allocation



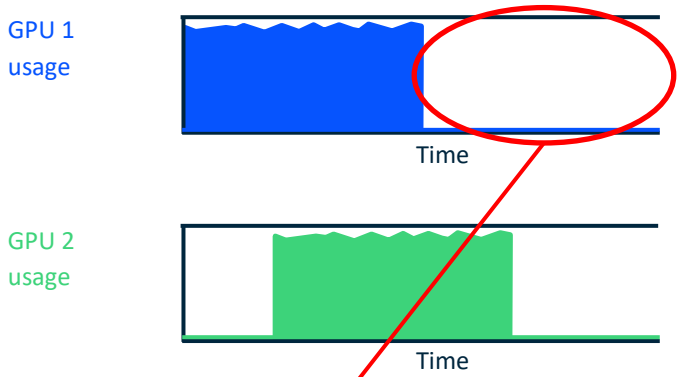
Utilization vs Allocation

People sometimes confuse utilization for allocation, but they are not the same.

Allocation refers to the ***assignment*** of a certain resource (GPUs, in our case) for use.



Utilization refers to the ***use*** of a certain resource (GPUs, in our case) while in allocated.



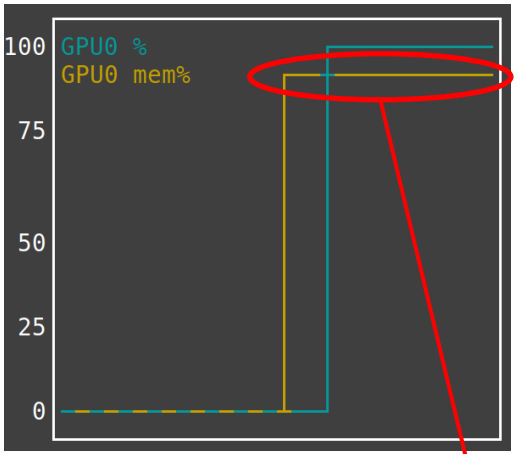
A GPU can be allocated…

…but not utilized

Example of Utilization vs Allocation, with TensorFlow

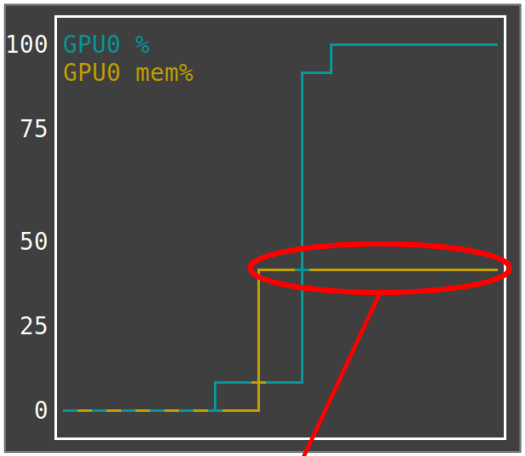
By default, TensorFlow *allocates* all GPU memory to itself, regardless of how much it needs.

(default)



Only by setting the TF_FORCE_GPU_ALLOW_GROWTH variable to true, will TensorFlow only allocate needed GPU memory.

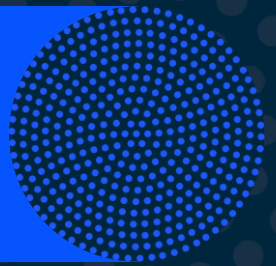
```
export TF_FORCE_GPU_ALLOW_GROWTH=true
```



One might think they are fully utilizing their GPU memory….

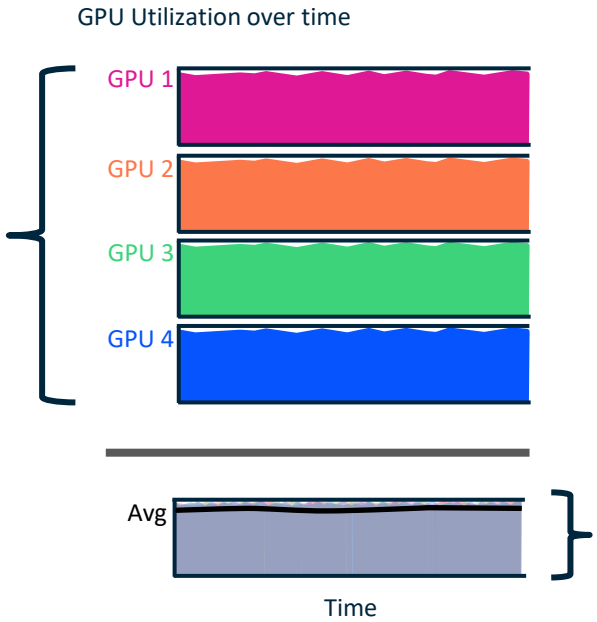
…when in fact, they are not.

Ideal GPU Utilization



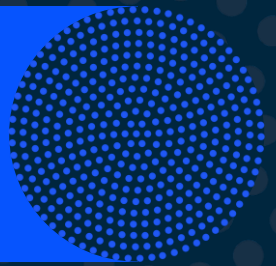
In a perfect world…

We would *fully* utilize *all* GPUs, at *all times.*



Therefore, our metric of interest is:
Average GPU utilization over time

Sub-optimal GPU Utilization



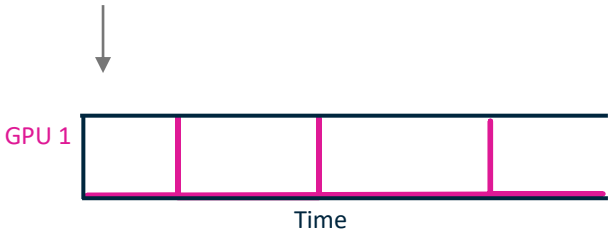
In real life…

GPUs are *ridiculously* underutilized! Avg Util Over Time is almost always less than 10%! Why?

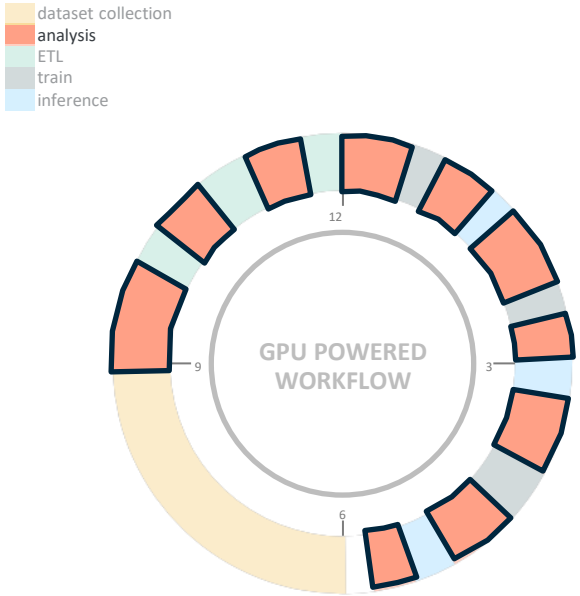
Example: GPU usage during ‘analysis’

During Analysis,

- most time is spent **writing code**.
- *Intermittently* run segments of code to **test**.



During analysis, **most of my time** I am **not utilizing** the **GPU to its fullest capacity.**



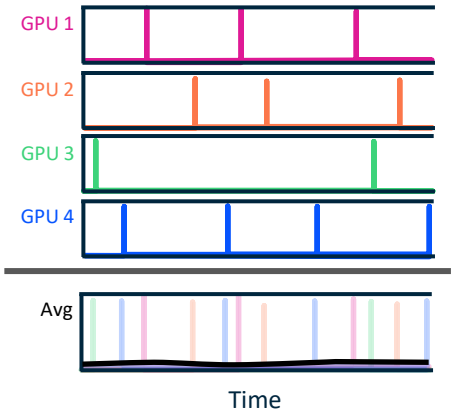
developer.nvidia.com/blog/gpu-accelerated-analytics-rapids/

In real life…

GPUs are *ridiculously* underutilized! Avg Util Over Time is almost always less than 10%! Why?

During initial analysis/prototyping, most time is spent writing and developing code/scripts. The only time the GPU is used is during sporadic testing of code.

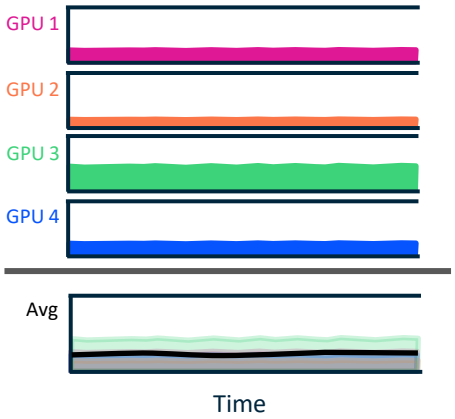
GPU Utilization over time



This is where we see the absolute lowest Average Utilization.

Sometimes the model(s) we are training is very small (or very simple), or the ETL pipeline we are running is not complex.

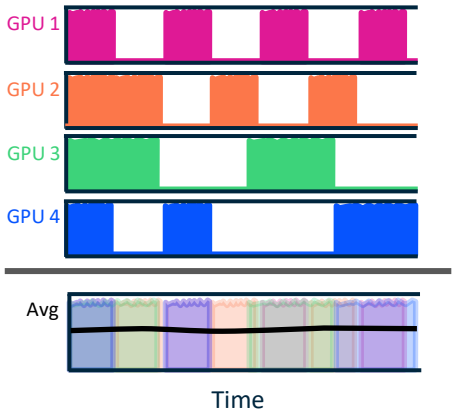
GPU Utilization over time



In these cases, we get more consistent utilization, but still do not saturate our GPUs.

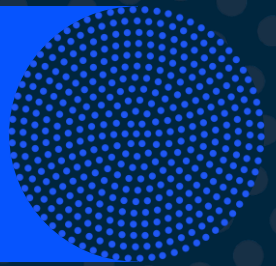
We might sometimes need to train a series of large/complex models, or complex ETL pipelines, but we don't run them back-to-back because we get busy (or some other reason).

GPU Utilization over time



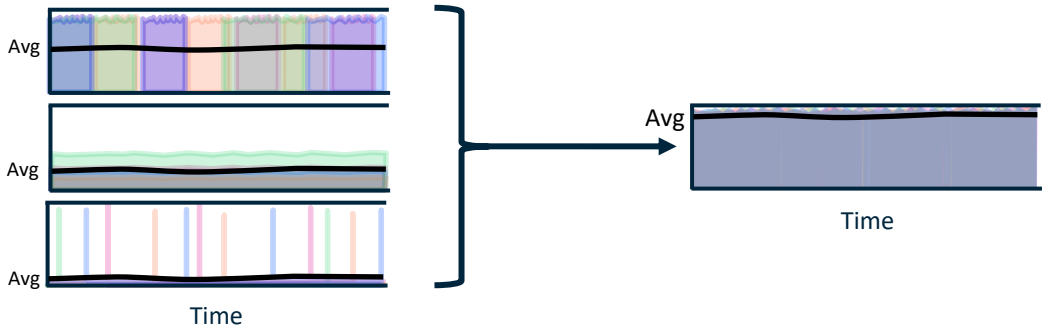
Here, there are long periods between high saturation periods where our GPUs do nothing.

Optimizing GPU Utilization with run:ai



Goal: Maximize average GPU utilization over time

run:ai is **Kubernetes** based **software** specifically designed to **maximize GPU utilization**



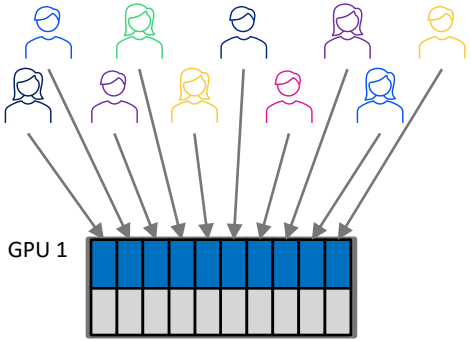
How?

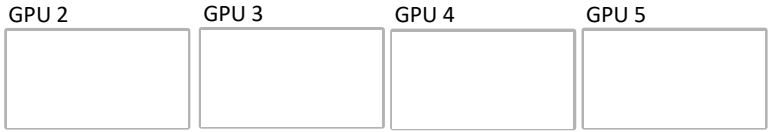
1. Fractional GPUs
2. Dynamic scheduler

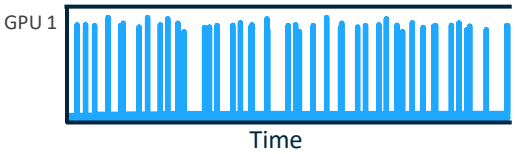
Addressing ‘analysis’ low GPU usage

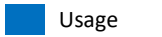
With run.ai **fractional GPU**

- **Each** data **scientist** can be **allocated a *fraction* of a GPU**
- Prevents GPU compute power from being taken up by low usage tasks
- **Frees up GPU resources for other tasks** that use them heavily



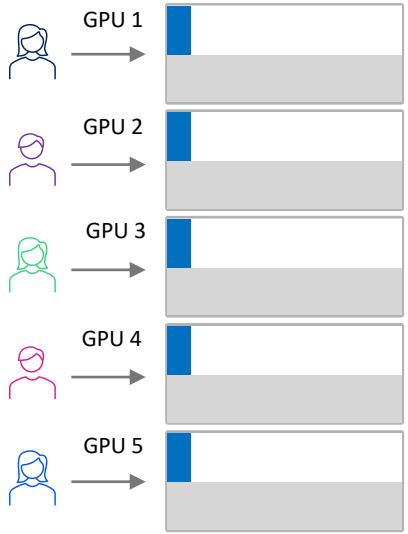


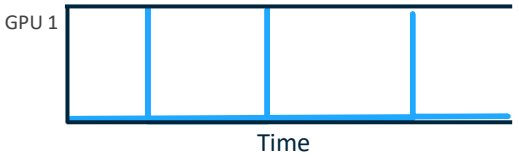




Traditional

- **Each** data **scientist** must be **allocated an entire GPU**
- Because most of a data scientist’s day does *not* consist of training inference or ETL, **GPU usage tends to be** low and Most **GPU compute power sits idle** most of the time
- **Less GPU resources** are **available** for other tasks that use them heavily





Addressing low GPU saturation

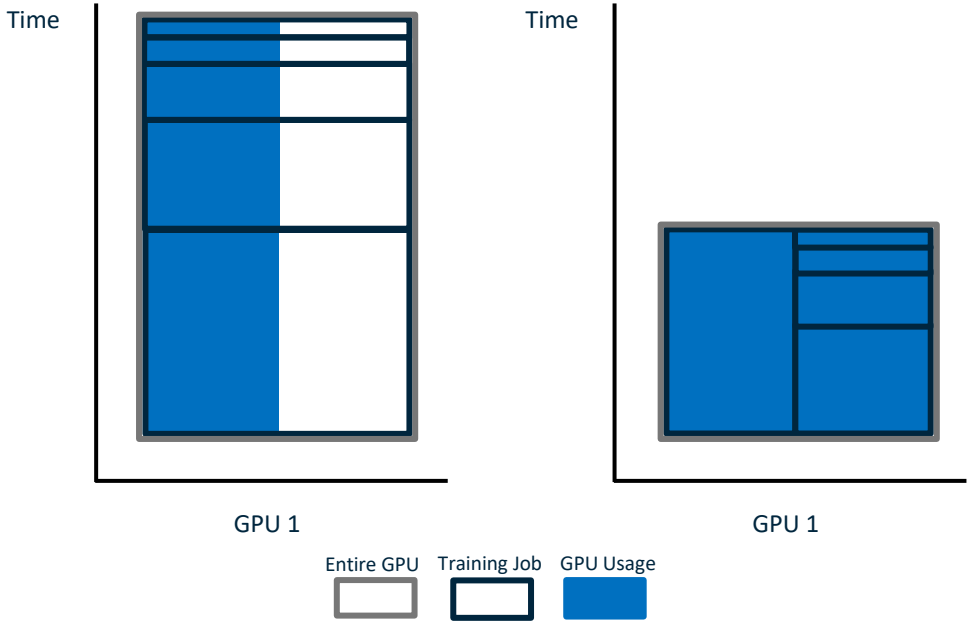
With run:ai factional GPU

We can also address the low-GPU-saturation problem with fractional GPUs.

Workloads that do not saturate the GPU can be combined, and run on the same GPU, simultaneously

Without Run:ai, an entire GPU had to be dedicated to each job.

With Run:ai, a fraction of a GPU (50% in this case) could be dedicated to each job, and all jobs would finish sooner



Addressing idle GPU time

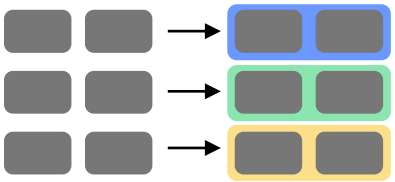
With run:ai dynamic scheduler

We can address the idle GPU problem with the Dynamic Scheduler.

Dynamic scheduler example:

A

[1] A guaranteed-quota of 2 GPUs per project, is set up



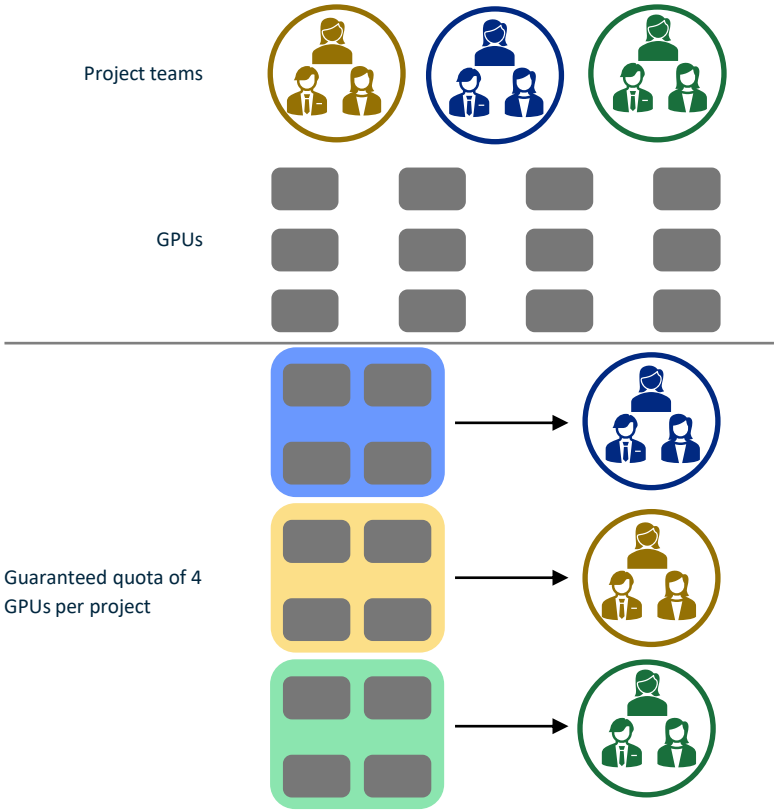
B

C

What is “guaranteed-quota?”

Ensures a project always has a **minimum level of GPU resources available** to them.

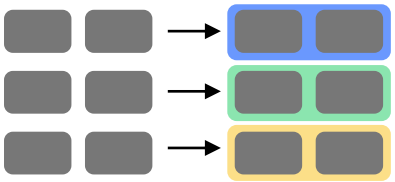
- Guaranteed-quota GPUs are dynamic, not static. i.e. any GPU from the GPU pool could be allocated to anyone.



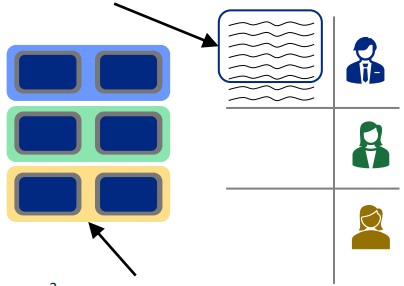
Dynamic scheduler example:

A

[1] A guaranteed-quota of 2 GPUs per project, is set up

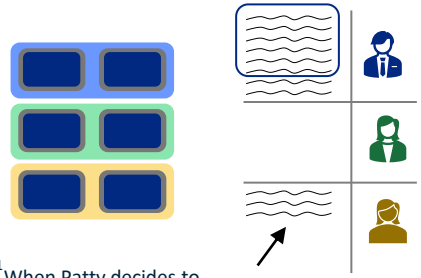


[2] If Bob is the only one with jobs in the queue....



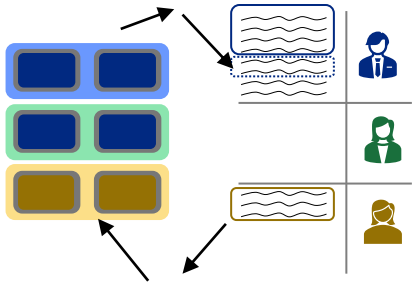
[3] ... Bob's jobs will be run on all available GPUs

B



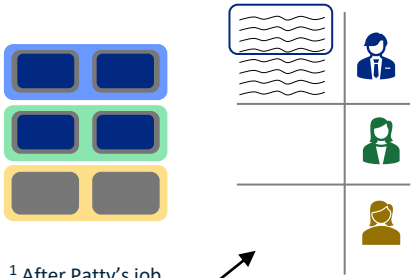
[1] When Patty decides to submits some jobs...

[2] ... some of Bob's jobs will stopped and put back into the queue....

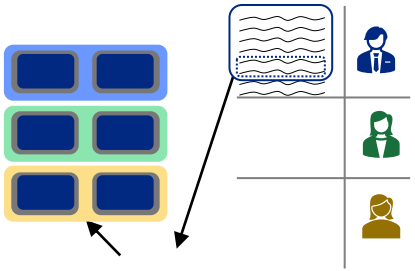


[3] ... and Patty's jobs will run

C



[1] After Patty's job finish running...



[2] ... Bob's previously stopped jobs will resume.

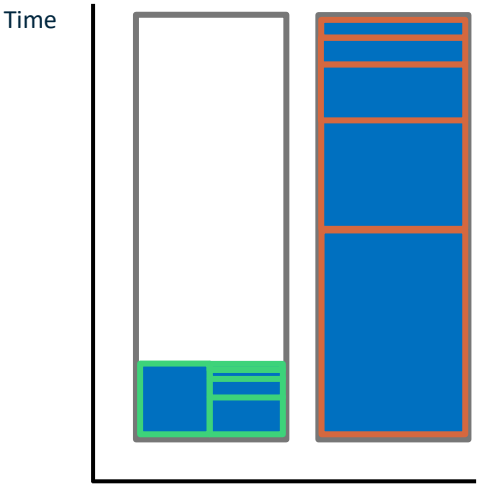
Addressing idle GPU time

With run:ai dynamic scheduler

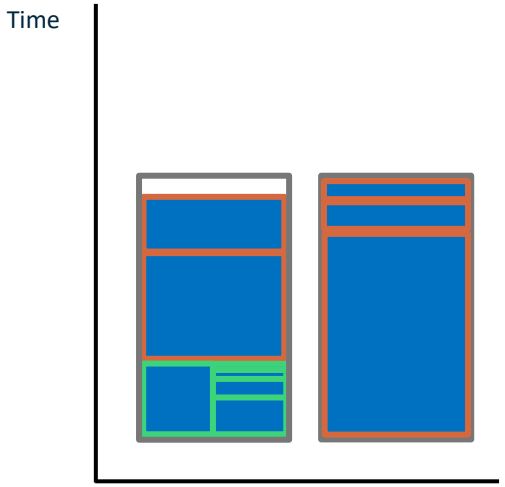
We can address the idle GPU problem with the Dynamic Scheduler.

As long as there are jobs in the queue, the scheduler will make sure all GPUs are being used.

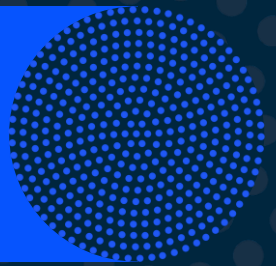
Without Run:ai, jobs are not automatically run on extra GPUs.



With Run:ai, the dynamics scheduler automatically runs job on extra GPU, when they are available.

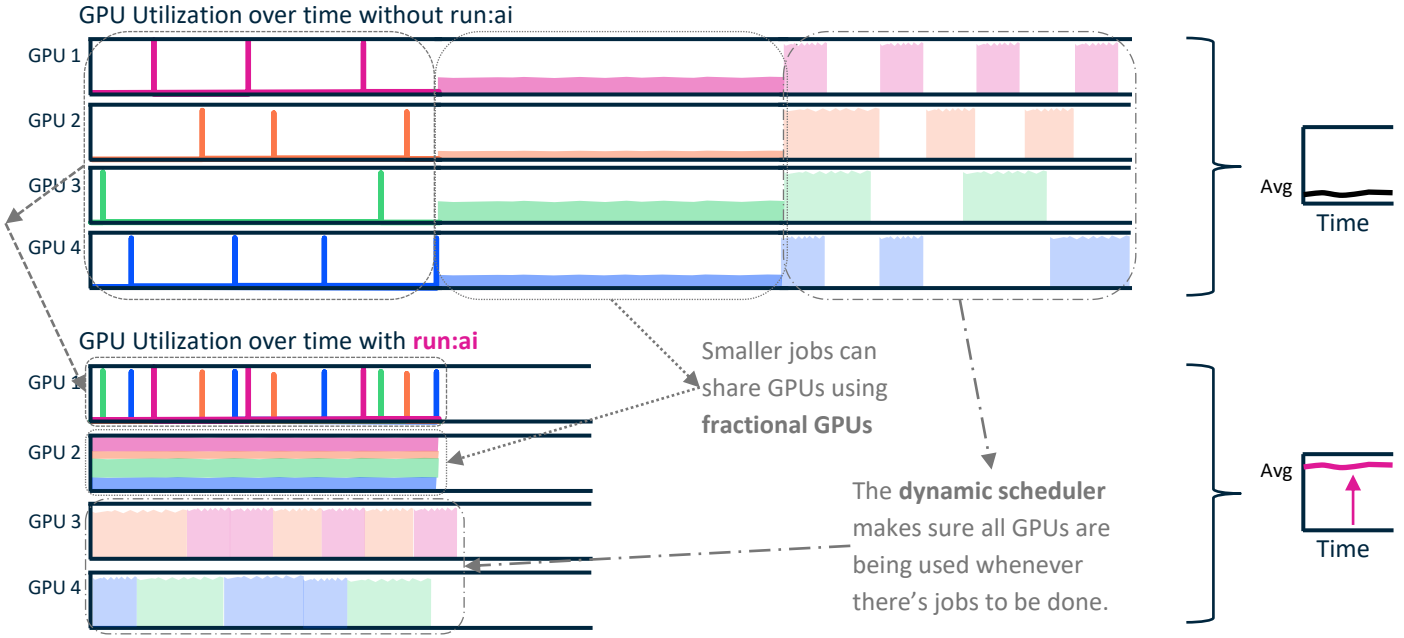


Putting it all together



Gains in GPU Utilization with run:ai

‘Analysis’ work to be done concurrently on one GPU with the **fractional GPUs** feature

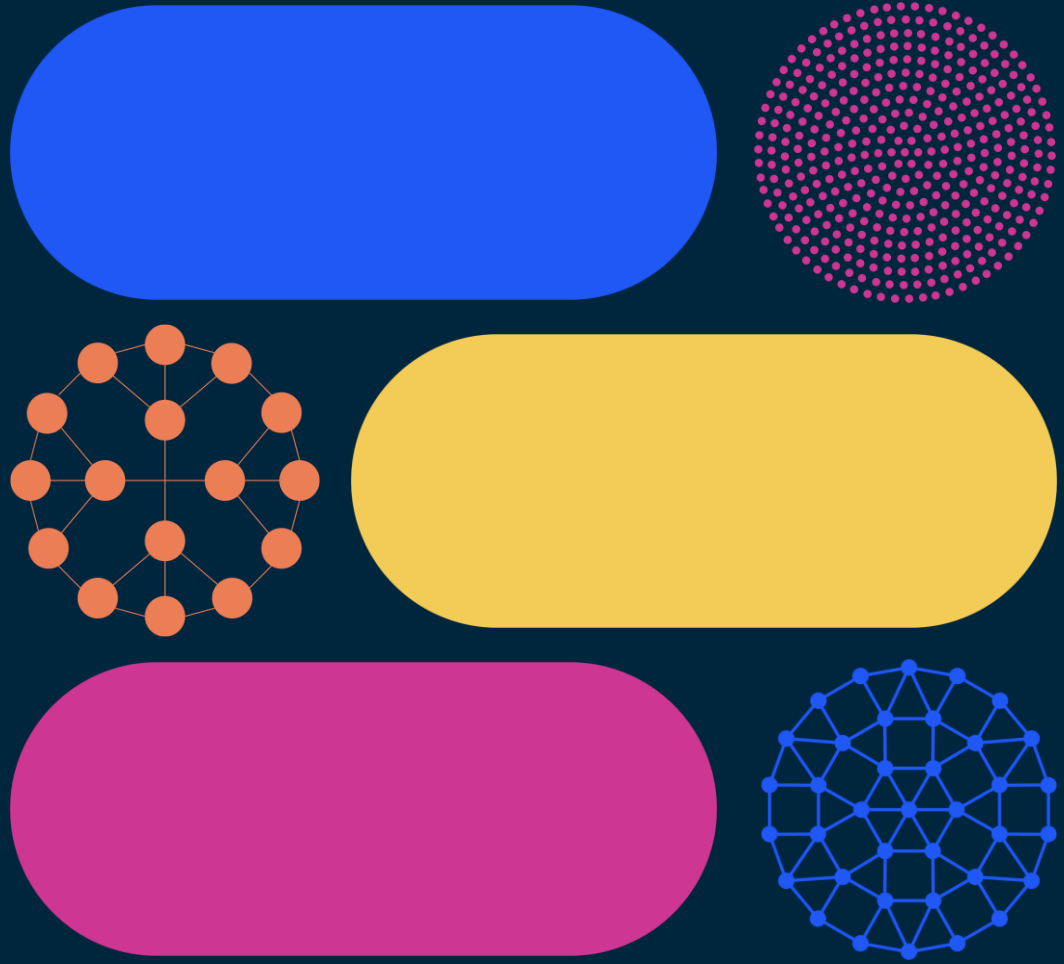


Net result of optimization with run:ai

- Resources become free much sooner
- More work gets done in less time

Use run:ai!

Thank you!



info@run.ai | www.run.ai