Introduction to Efficient Data Pipelines

Efficient training pipelines

DeepLearning.AI

In Module 4 you'll dive into:

Optimization

Images

Text

Efficiency

This module is about optimizing training time

Build efficient data pipelines

This module is about optimizing training time

Build efficient data pipelines

Profile training loops

This module is about optimizing training time

Build efficient data pipelines

Profile training loops

Apply optimization techniques

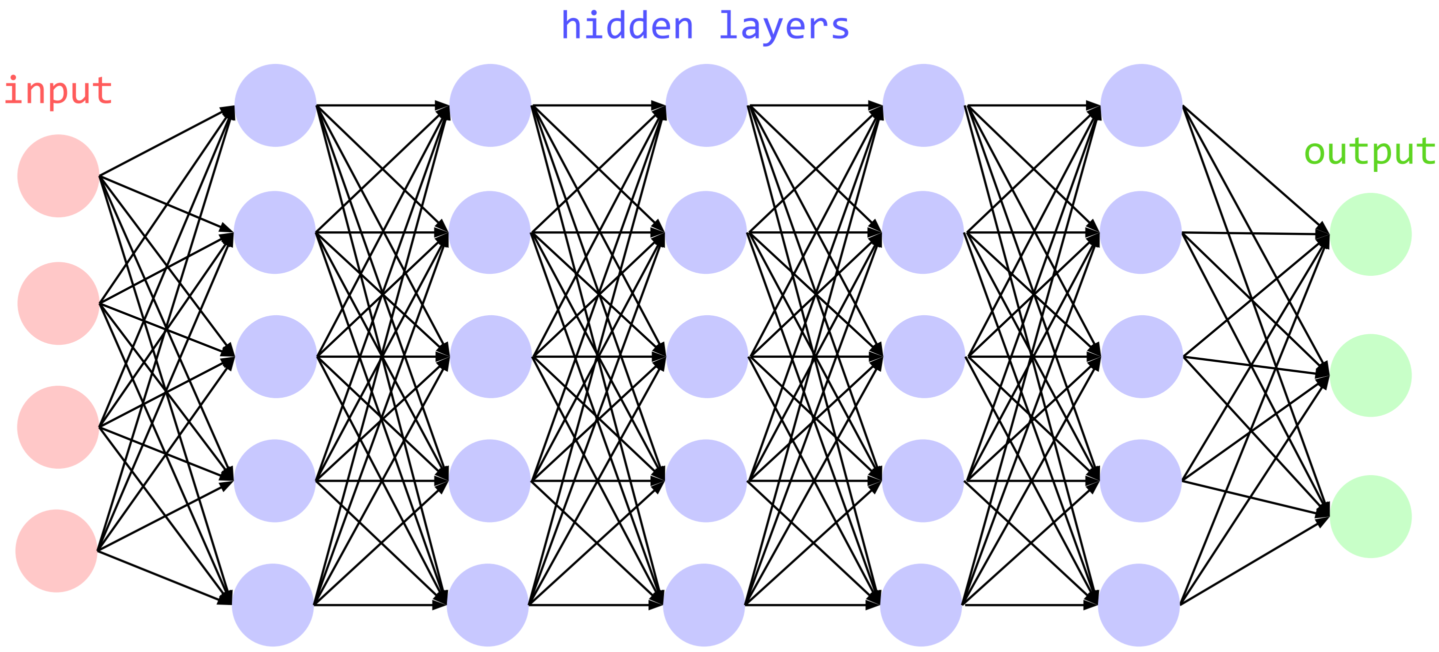
This module is about optimizing training time

Build efficient data pipelines

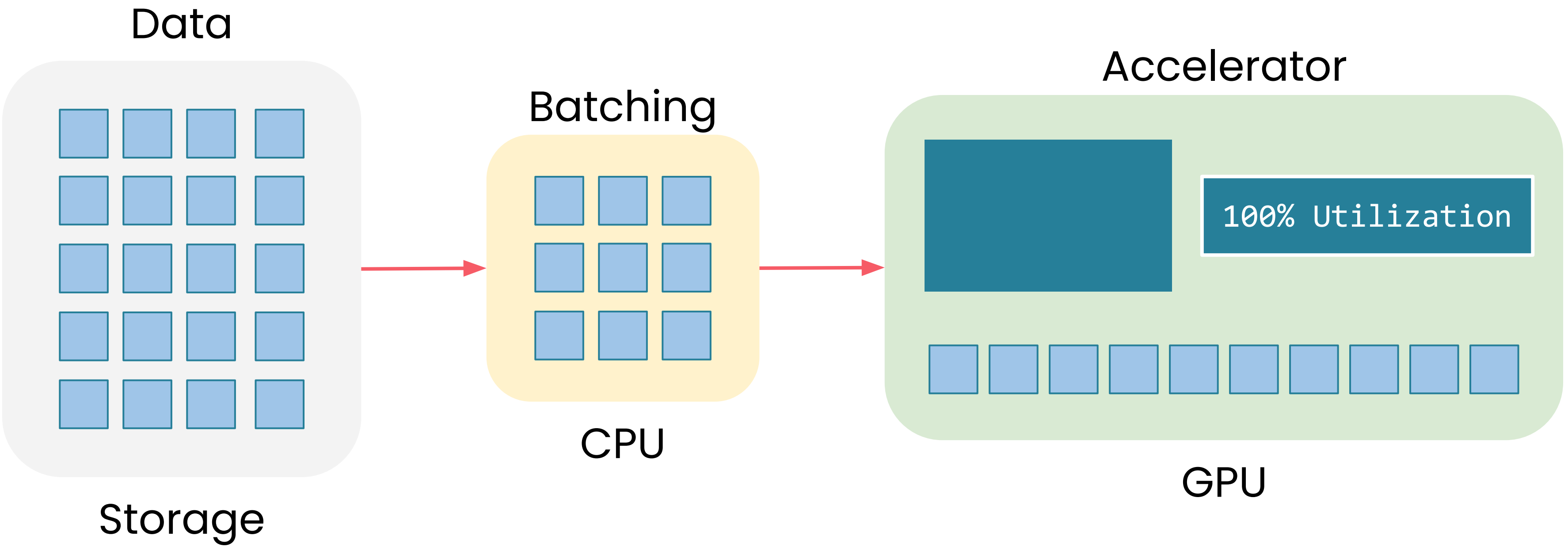
Profile training loops

Apply optimization techniques

Is training painfully slow?



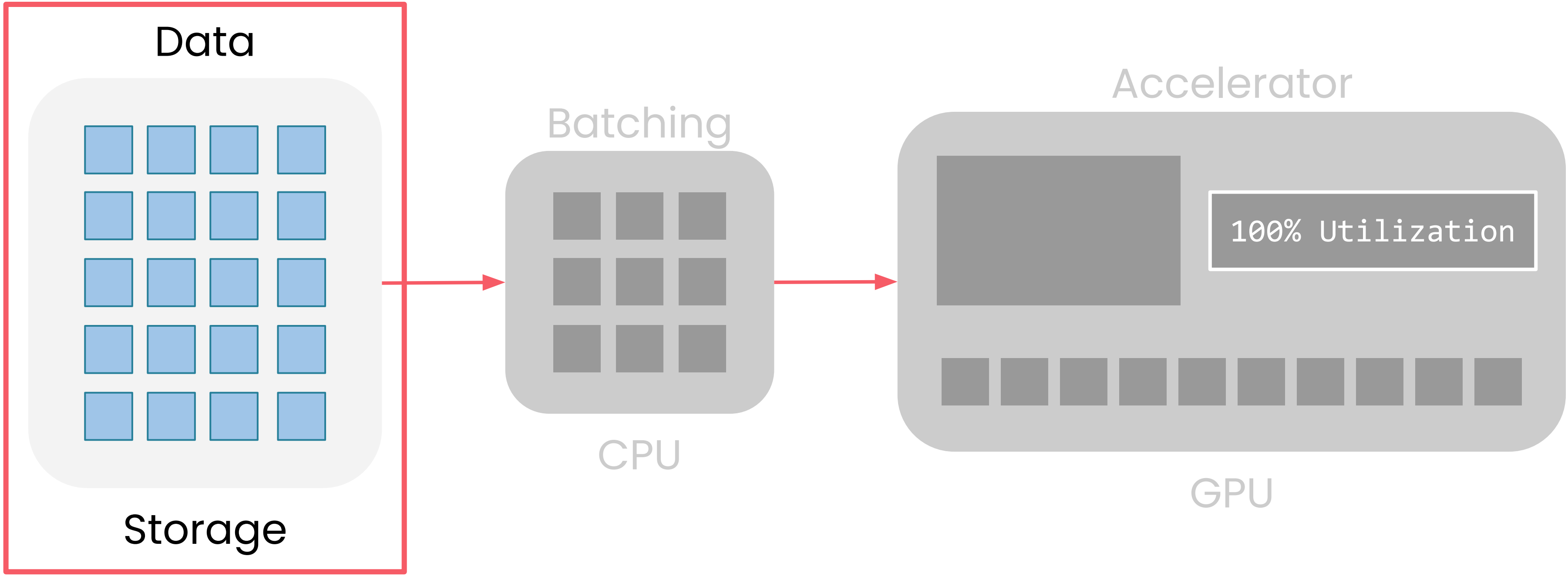
The data bottleneck



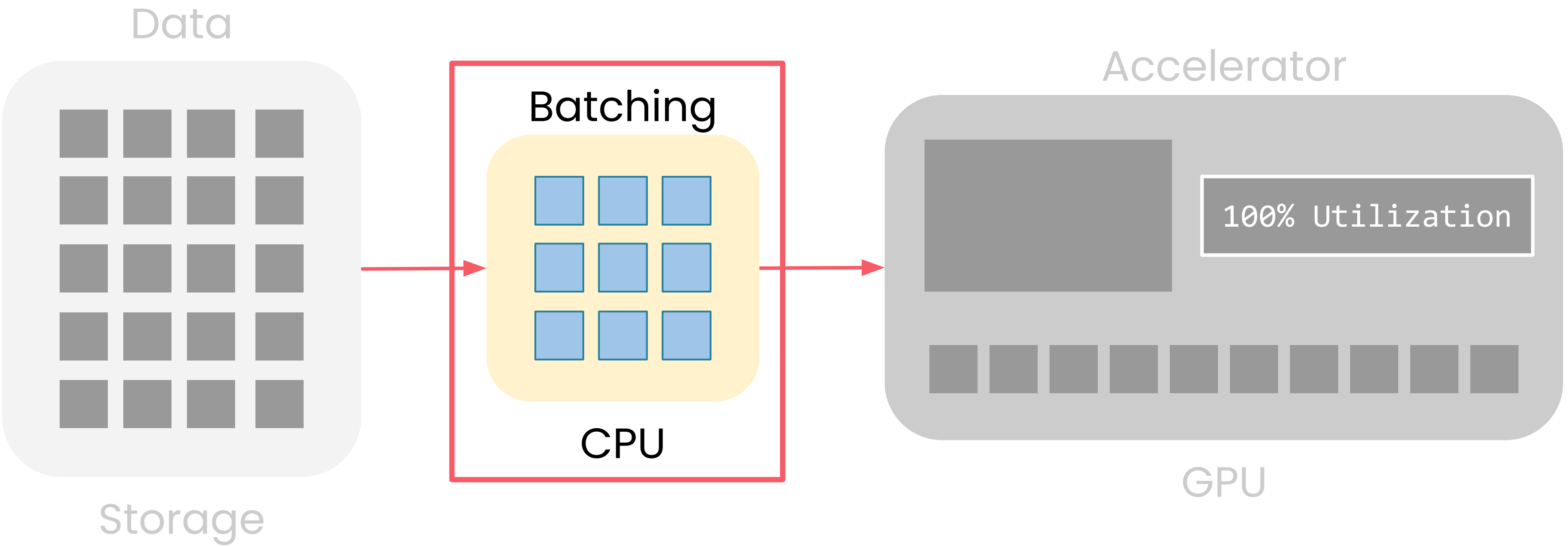
DeepLearning.AI

Laurence Moroney

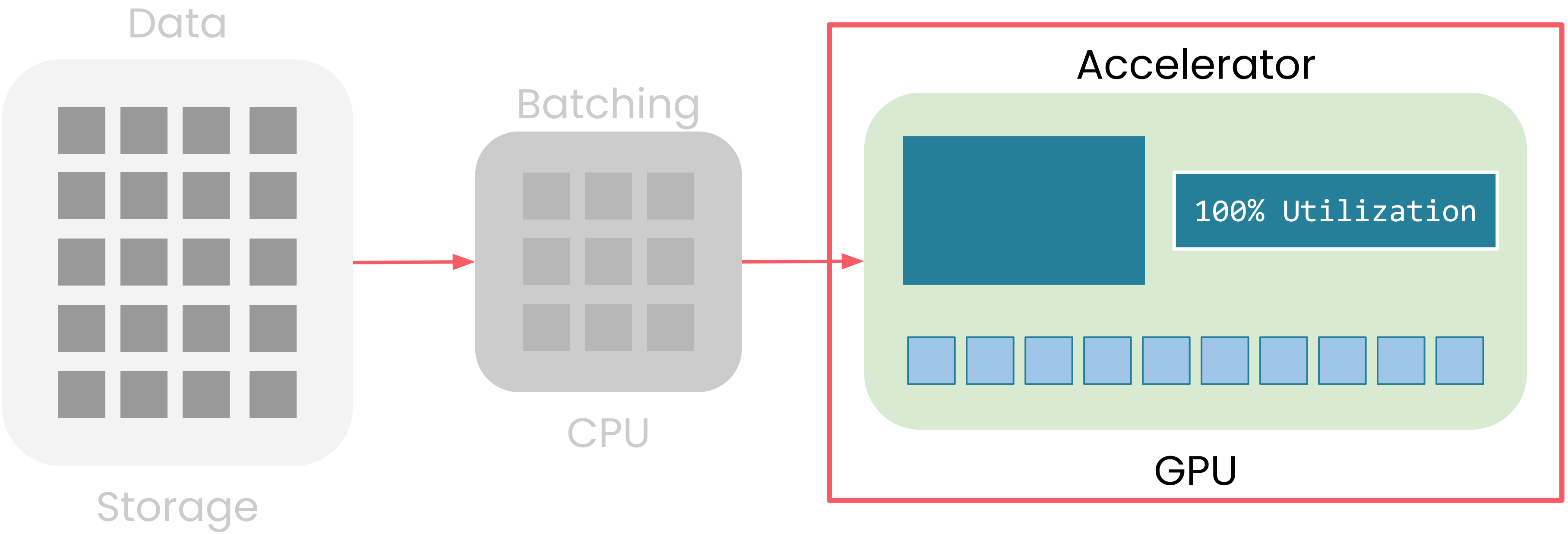
The data bottleneck



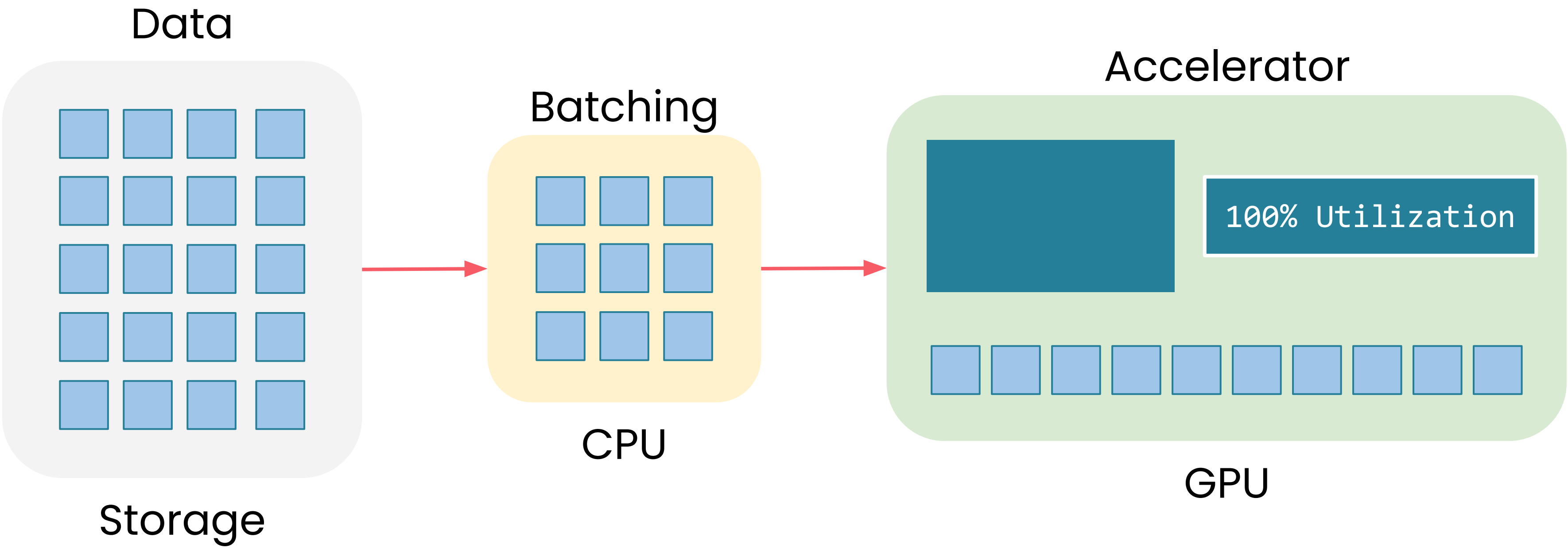
The data bottleneck



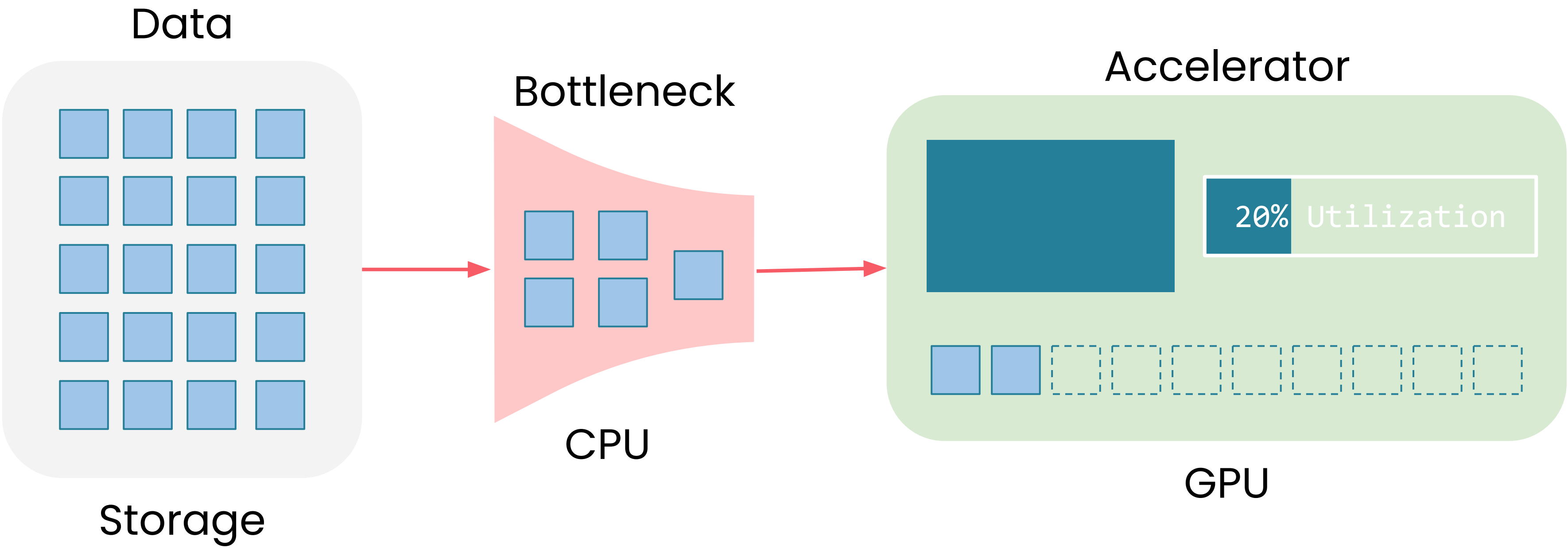
The data bottleneck



The data bottleneck



The data bottleneck



DeepLearning.AI

Laurence Moroney

Key tools for solving the bottleneck

`Dataset`

`DataLoader`

Dataset: The blueprint for your data

```python
from torch.utils.data import Dataset

class CustomDataset(Dataset):

   def __init__(self, data_path):
       # Load data, preprocessing, etc.
       pass

   def __len__(self):
       # Return the size of your dataset
       return len(self.data)

   def __getitem__(self, idx):
       # Return sample by index
       return self.data[idx]
```

Dataset: The blueprint for your data

```
from torch.utils.data import Dataset

class CustomDataset(Dataset):

    def __init__(self, data_path):
        # Load data, preprocessing, etc.
        pass

    def __len__(self):
        # Return the size of your dataset
        return len(self.data)

    def __getitem__(self, idx):
        # Return sample by index
        return self.data[idx]
```

Dataset: The blueprint for your data

```
from torch.utils.data import Dataset

class CustomDataset(Dataset):

    def __init__(self, data_path):
        # Load data, preprocessing, etc.
        pass

    def __len__(self):
        # Return the size of your dataset
        return len(self.data)

    def __getitem__(self, idx):
        # Return sample by index
        return self.data[idx]
```

Dataset: The blueprint for your data

```python
from torch.utils.data import Dataset

class CustomDataset(Dataset):

    def __init__(self, data_path):
        # Load data, preprocessing, etc.
        pass

    def __len__(self):
        # Return the size of your dataset
        return len(self.data)

    def __getitem__(self, idx):
        # Return sample by index
        return self.data[idx]
```


Laurence Moroney

DataLoader: Batching, shuffling, and loading

```python
from torch.utils.data import DataLoader

dataset = CustomDataset('path/to/data')
dataloader = DataLoader(
    dataset,
    batch_size=32,
    shuffle=True,
    num_workers=4
)

for batch in dataloader:
    # Training loop
    pass
```

DataLoader: Batching, shuffling, and loading

```
from torch.utils.data import DataLoader

dataset = CustomDataset('path/to/data')
dataloader = DataLoader(
    dataset,
    batch_size=32,
    shuffle=True,
    num_workers=4
)

for batch in dataloader:
    # Training loop
    pass
```


Laurence Moroney

DataLoader: Batching, shuffling, and loading

```
from torch.utils.data import DataLoader

dataset = CustomDataset('path/to/data')
dataloader = DataLoader(
    dataset,
    batch_size=32,
    shuffle=True,
    num_workers=4
)

for batch in dataloader:
    # Training loop
    pass
```

DataLoader: Batching, shuffling, and loading

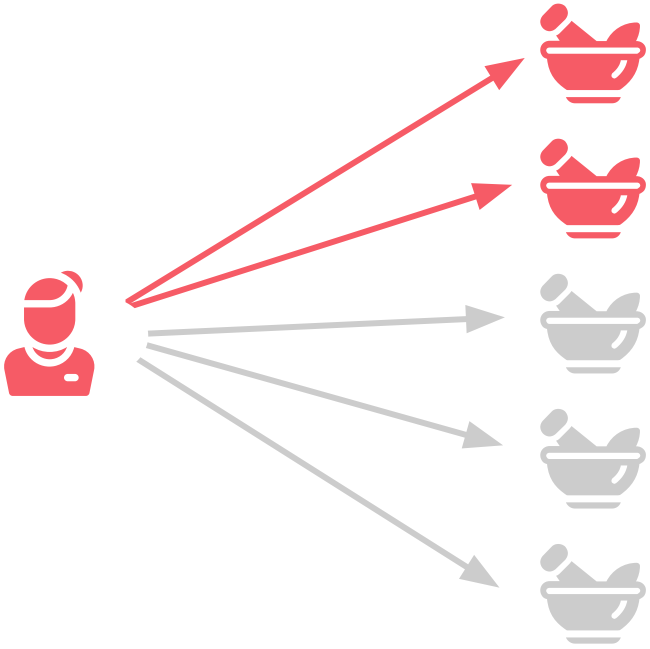
```
from torch.utils.data import DataLoader

dataset = CustomDataset('path/to/data')
dataloader = DataLoader(
    dataset,
    batch_size=32,
    shuffle=True,
    num_workers=4
)

for batch in dataloader:
    # Training loop
    pass
```

Number of workers: A critical parameter

Controls how many CPU processes are used in parallel to load data

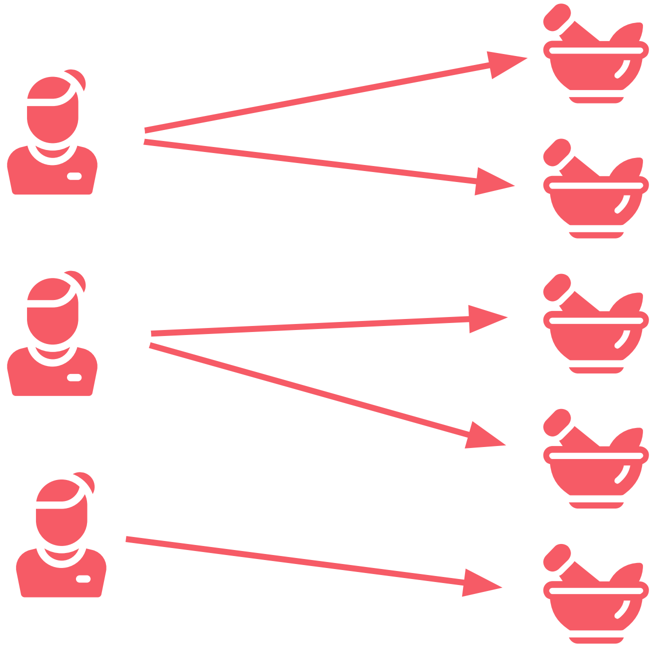
Number of workers: A critical parameter

Controls how many CPU processes are used in parallel to load data



Number of workers: A critical parameter

Controls how many CPU processes are used in parallel to load data



Experimenting with number of workers

```python
trainset = helper_utils.download_and_load_cifar10()
workers_to_test = [0, 2, 4, 6, 8, 10]

def experiment_workers(workers_to_test, trainset, device):

   worker_times = {}

   for nw in workers_to_test:
       print(f"--- Testing Number of Workers = {nw} ---")

       loader = DataLoader(trainset,
                           batch_size=32,
                           shuffle=True,
                           num_workers=nw
                          )
```

Experimenting with number of workers

```python
trainset = helper_utils.download_and_load_cifar10()
workers_to_test = [0, 2, 4, 6, 8, 10]

def experiment_workers(workers_to_test, trainset, device):

    worker_times = {}

    for nw in workers_to_test:
        print(f"--- Testing Number of Workers = {nw} ---")

        loader = DataLoader(trainset,
                            batch_size=32,
                            shuffle=True,
                            num_workers=nw
                           )
```

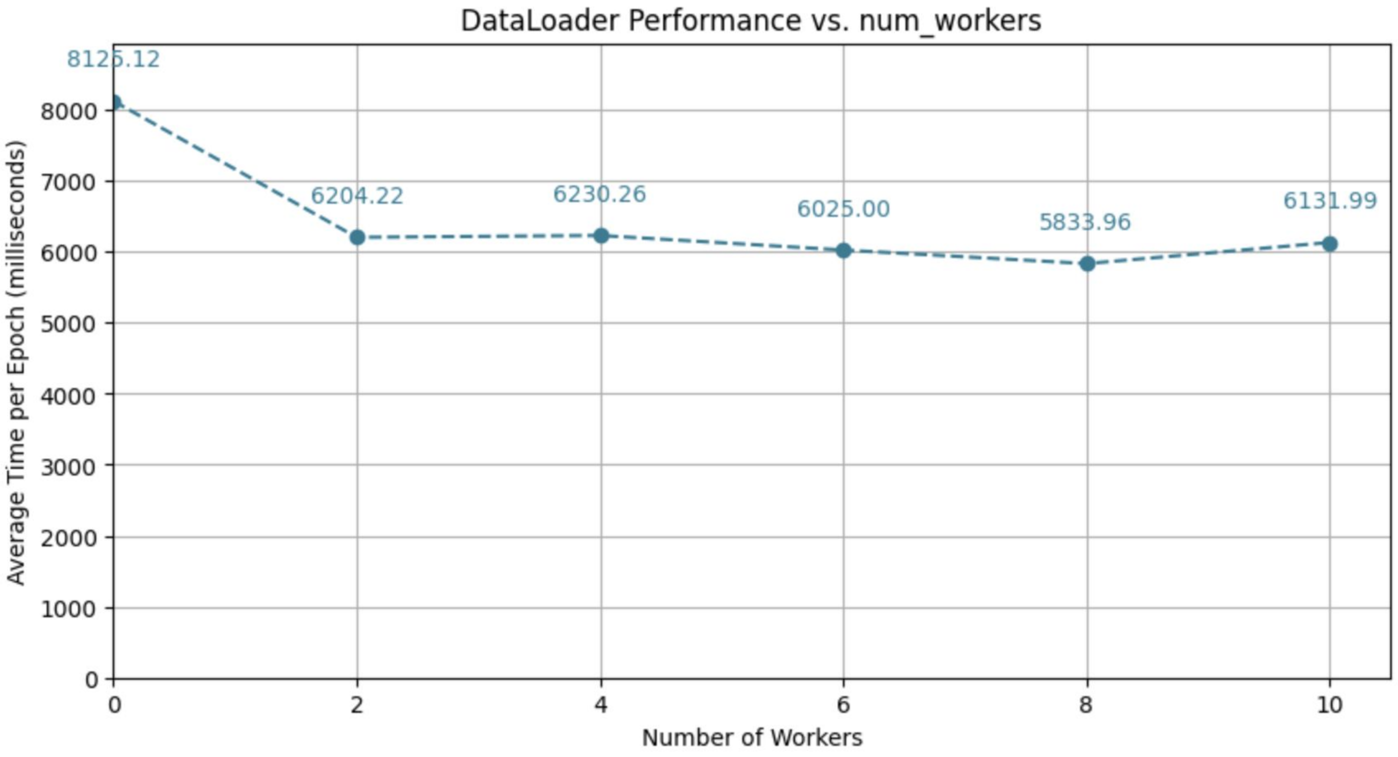
Experimenting with number of workers

```
trainset = helper_utils.download_and_load_cifar10()
workers_to_test = [0, 2, 4, 6, 8, 10]

def experiment_workers(workers_to_test, trainset, device):

    worker_times = {}

    for nw in workers_to_test:
        print(f"--- Testing Number of Workers = {nw} ---")

        loader = DataLoader(trainset,
                            batch_size=32,
                            shuffle=True,
                            num_workers=nw
                           )
```

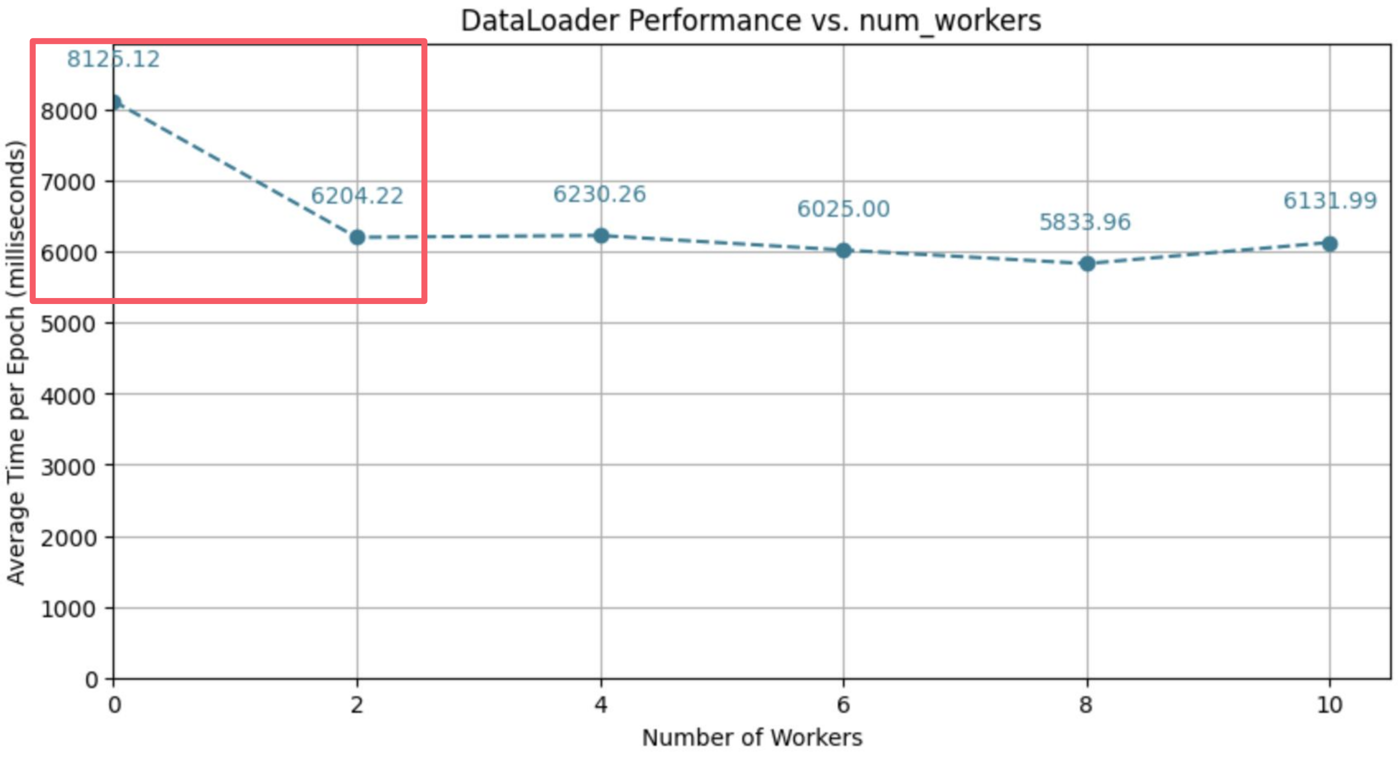
```python
for nw in workers_to_test:

    ...

    try:
        worker_times[nw] = helper_utils.measure_average_epoch_time(loader, device)
    except RuntimeError as e:
        print(f"\n❌ ERROR with {nw} workers. Likely a shared memory issue.")
        worker_times[nw] = float('inf')

    del loader
    gc.collect()

    if torch.cuda.is_available():
        torch.cuda.empty_cache()

return worker_times
```

```python
for nw in workers_to_test:

    ...

    try:
        worker_times[nw] = helper_utils.measure_average_epoch_time(loader, device)
    except RuntimeError as e:
        print(f"\nX ERROR with {nw} workers. Likely a shared memory issue.")
        worker_times[nw] = float('inf')

    del loader
    gc.collect()

    if torch.cuda.is_available():
        torch.cuda.empty_cache()

return worker_times
```

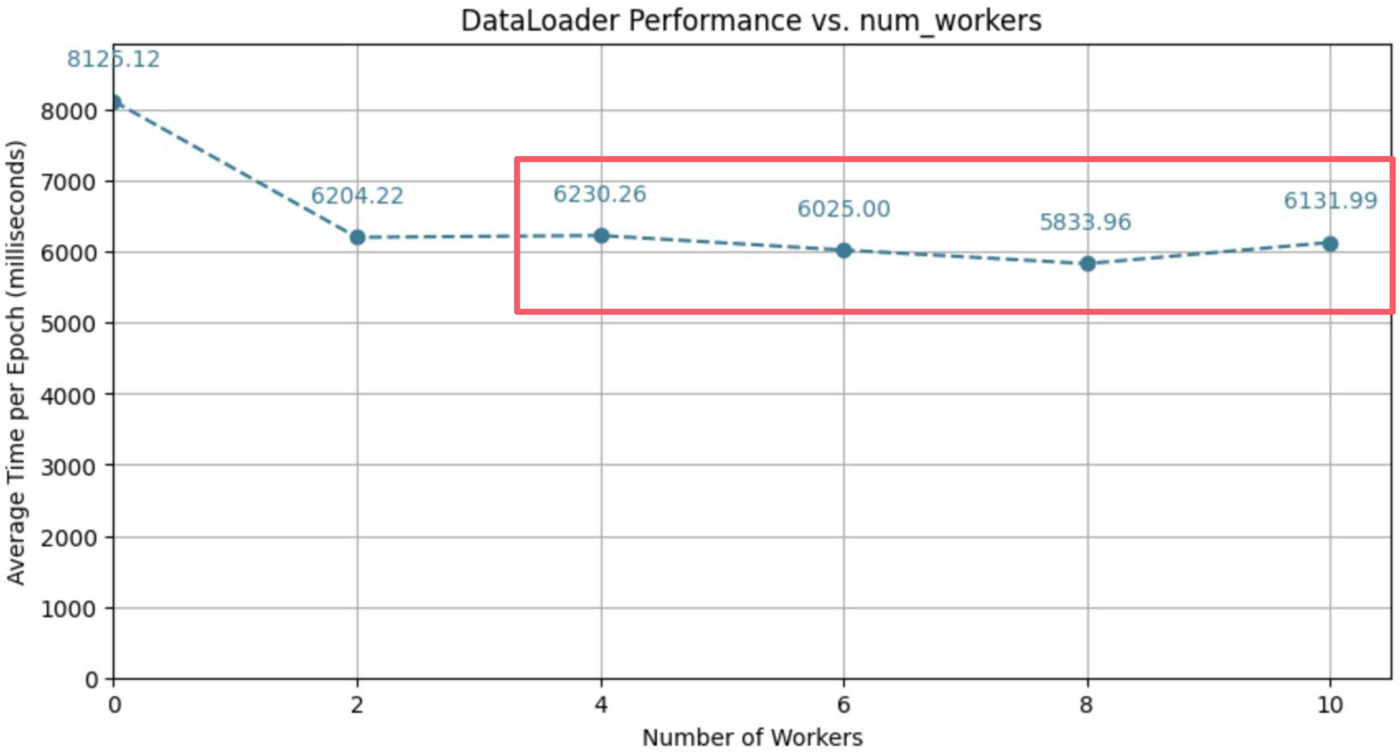
Effect of `num_workers` on time per epoch



Effect of `num_workers` on time per epoch



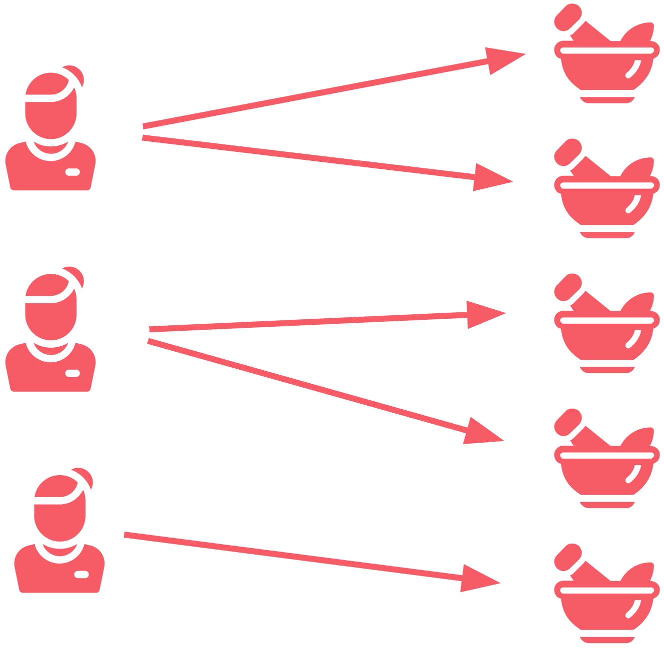
Effect of `num_workers` on time per epoch



DeepLearning.AI

Laurence Moroney

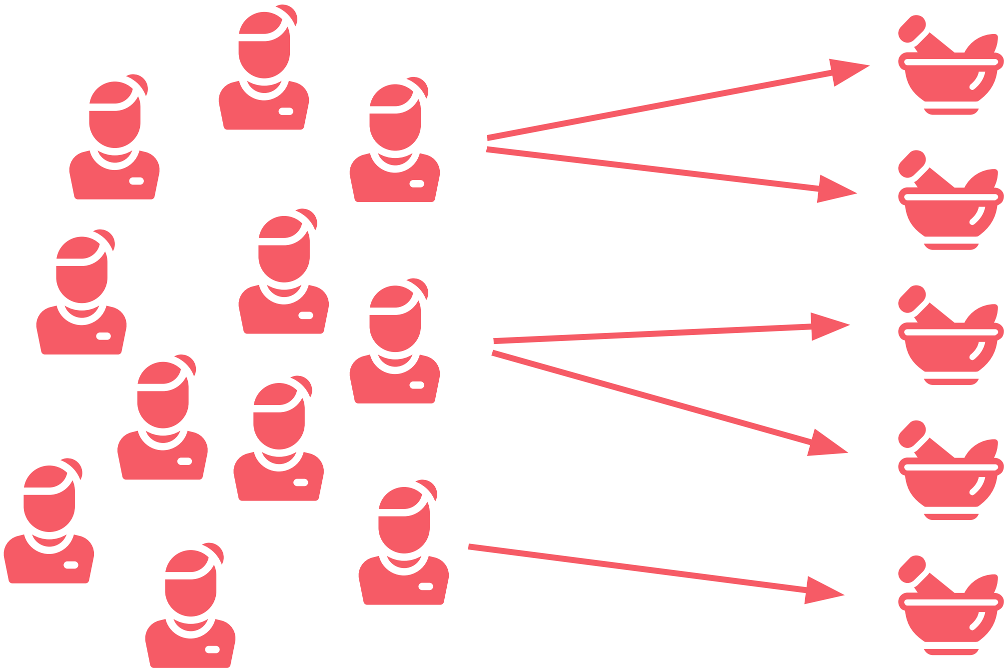
Too many workers create a new bottleneck



DeepLearning.AI

Laurence Moroney

Too many workers create a new bottleneck



Beware! Results will vary depending on:

Dataset

Model architecture

Hardware

Find the number of CPU cores in your machine

```python
cpu_cores = os.cpu_count()

print(f"Number of available CPU cores: {cpu_cores}")
```

Output

```
Number of available CPU cores: 48
```

GPU efficiency is a key driver

Idle GPU

Training slows down

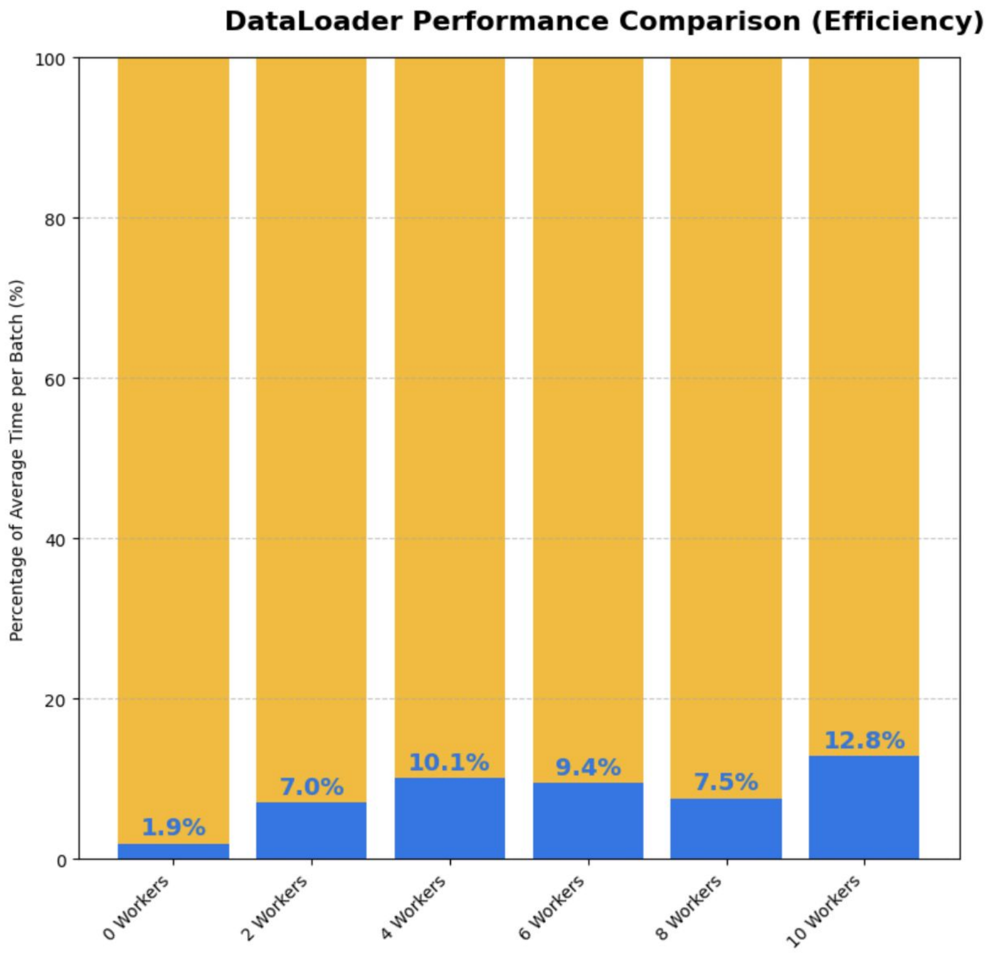
GPU efficiency is a key driver

Idle GPU

Training slows down

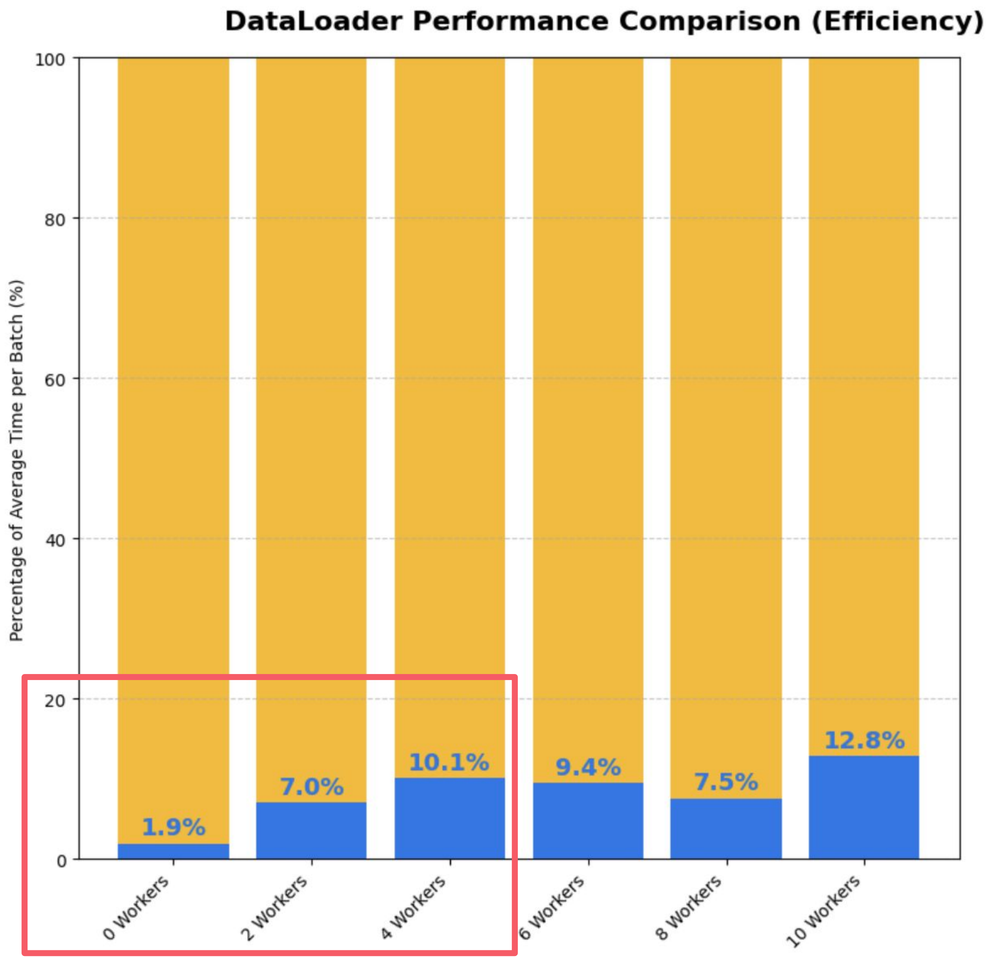
Active GPU

Training speeds up



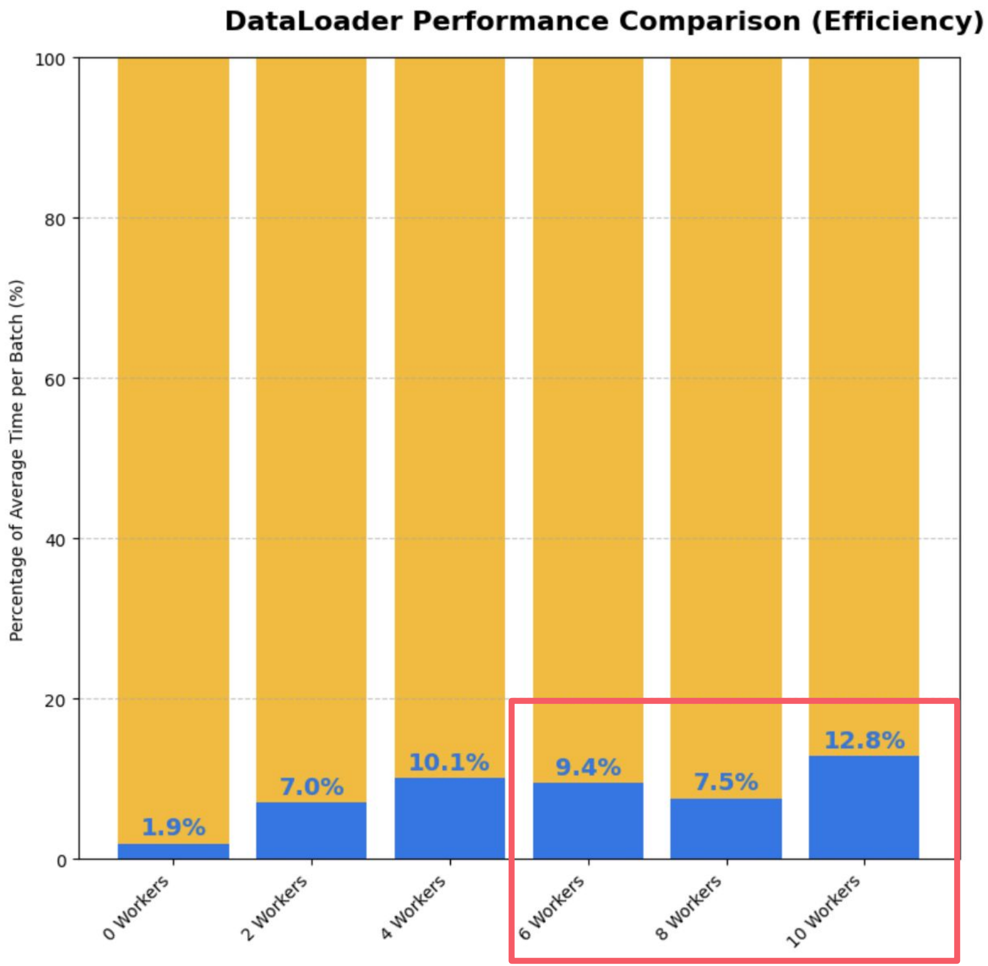
DeepLearning.AI

Laurence Moroney



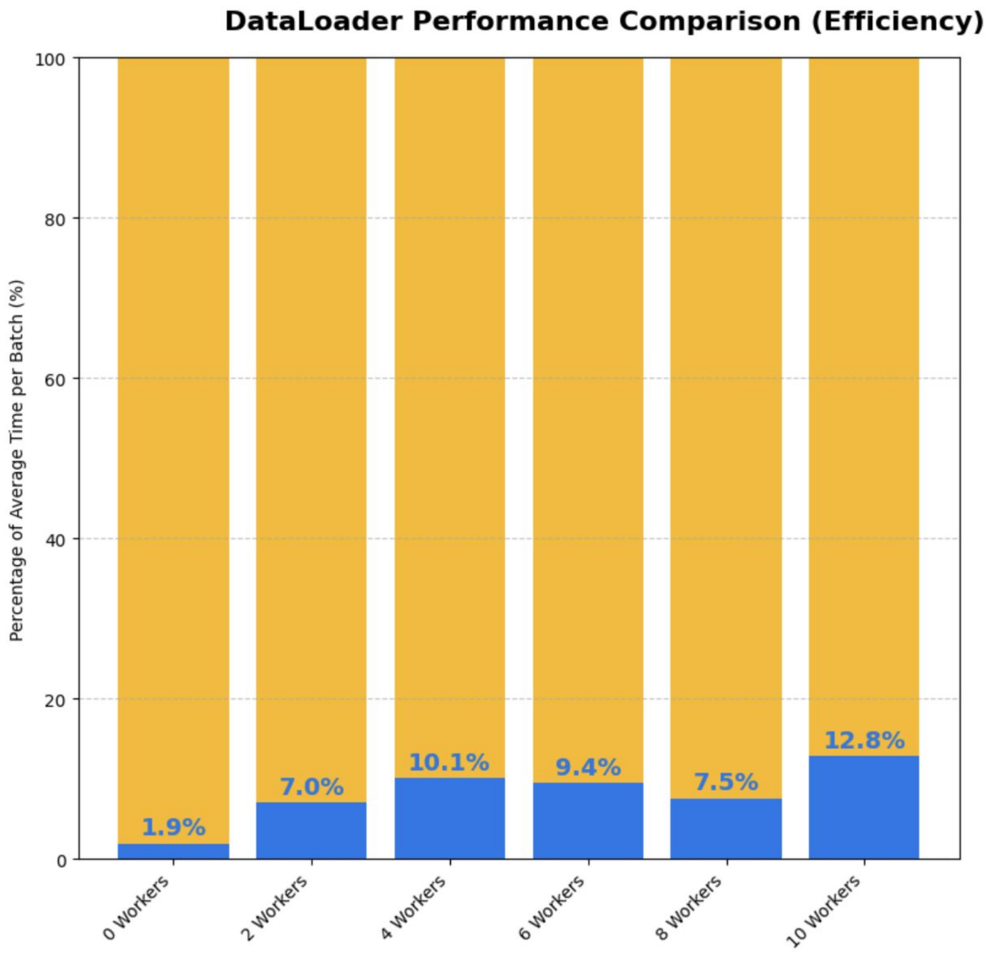
DeepLearning.AI

Laurence Moroney



DeepLearning.AI

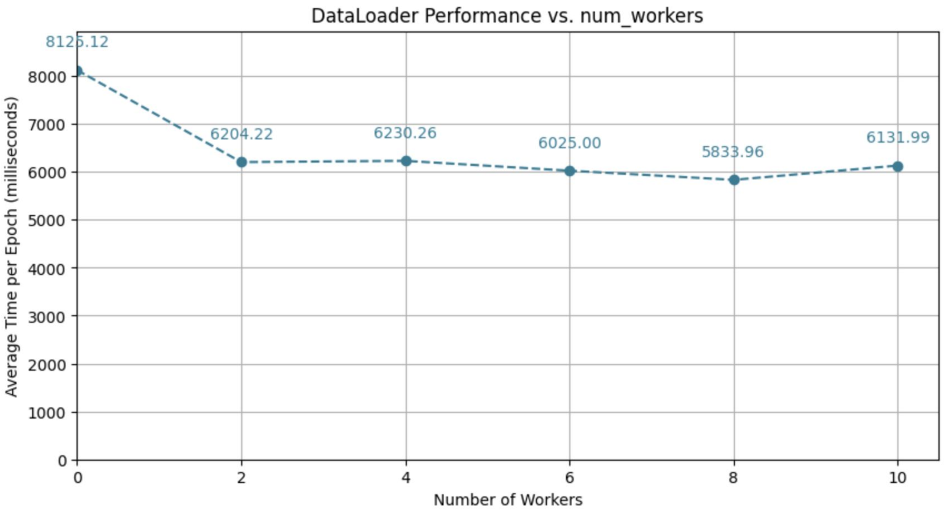
Laurence Moroney

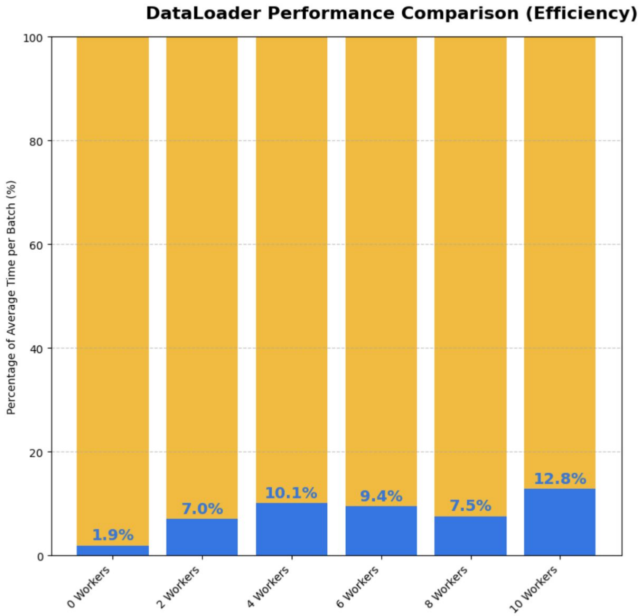


DeepLearning.AI

Laurence Moroney

Run experiments and observe trends





Batching and Other DataLoader Settings

Efficient training pipelines

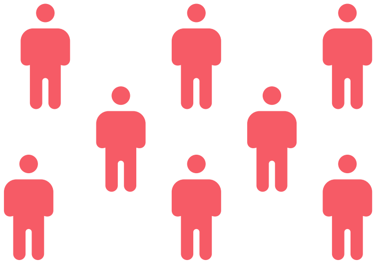
DeepLearning.AI

Batching is essential

Allows your model to process multiple data samples at the same time

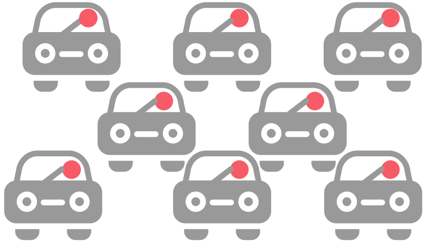
Batching is essential

Allows your model to process multiple data samples at the same time



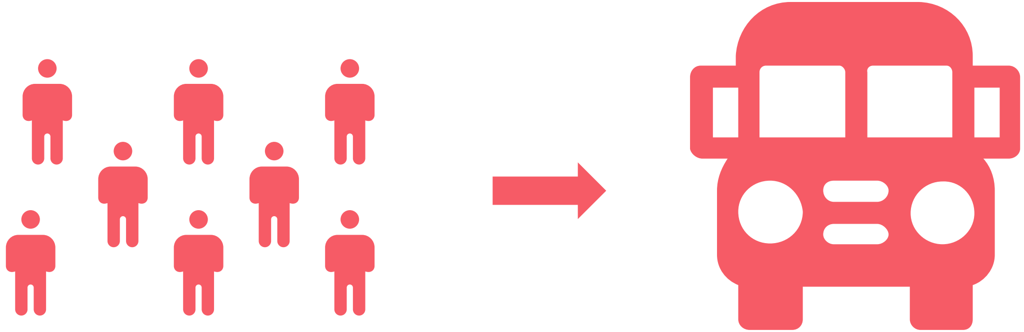
Batching is essential

Allows your model to process multiple data samples at the same time



Batching is essential

Allows your model to process multiple data samples at the same time



An experiment with 6 different batch sizes

```
batch_sizes_to_test = [16, 32, 64, 128, 256, 512]

def experiment_batch_sizes(batch_sizes_to_test, trainset, device):

   batch_size_times = {}

   for bs in batch_sizes_to_test:
       print(f"--- Testing Batch Size = {bs} ---")

       loader = DataLoader(trainset,
                           batch_size=bs,
                           shuffle=True,
                           num_workers=6
                       )
```

An experiment with 6 different batch sizes

```
batch_sizes_to_test = [16, 32, 64, 128, 256, 512]

def experiment_batch_sizes(batch_sizes_to_test, trainset, device):

    batch_size_times = {}

    for bs in batch_sizes_to_test:
        print(f"--- Testing Batch Size = {bs} ---")

        loader = DataLoader(trainset,
                            batch_size=bs,
                            shuffle=True,
                            num_workers=6
                        )
```

DeepLearning.AI

Laurence Moroney

An experiment with 6 different batch sizes

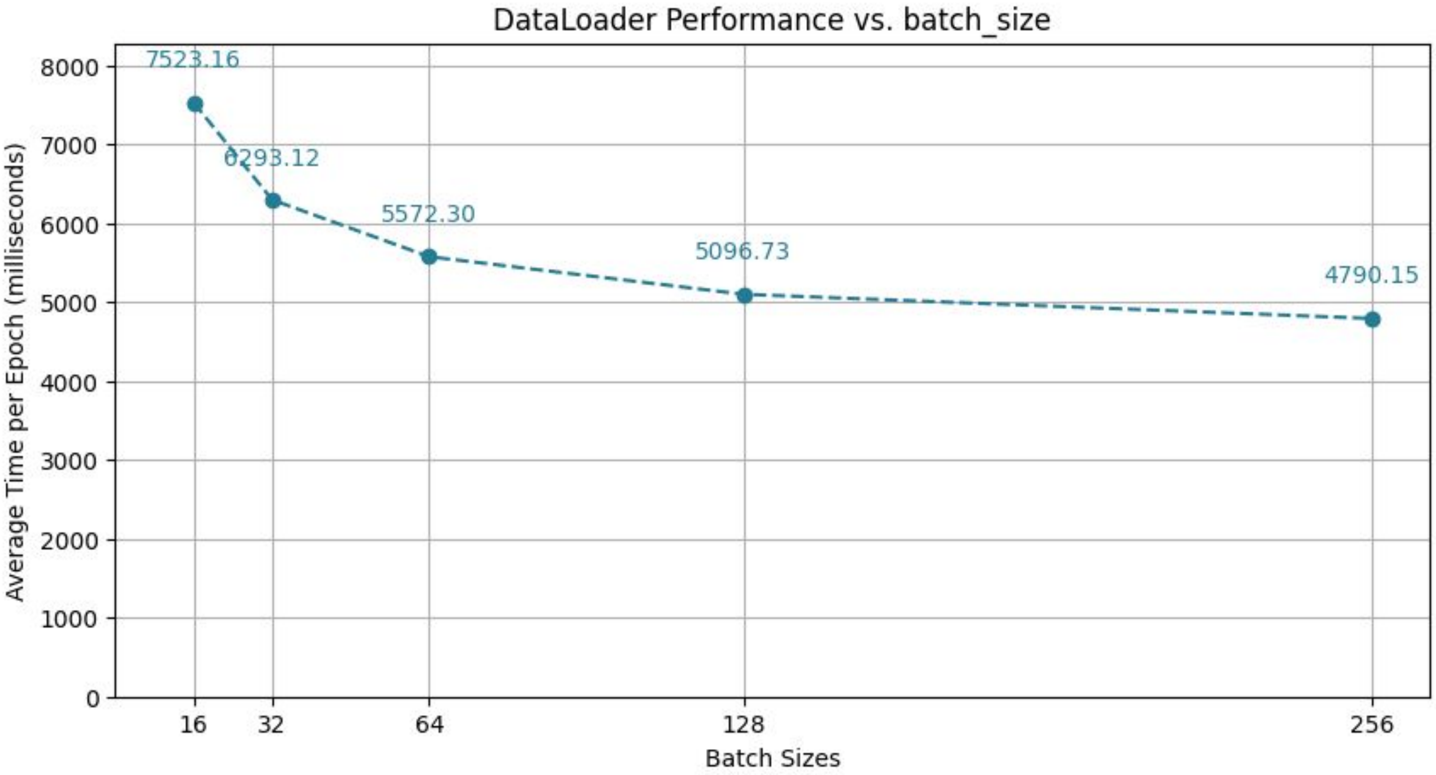
```
batch_sizes_to_test = [16, 32, 64, 128, 256, 512]

def experiment_batch_sizes(batch_sizes_to_test, trainset, device):

    batch_size_times = {}

    for bs in batch_sizes_to_test:
        print(f"--- Testing Batch Size = {bs} ---")

        loader = DataLoader(trainset,
                            batch_size=bs,
                            shuffle=True,
                            num_workers=6
                    )
```


Laurence Moroney

```python
for bs in batch_sizes_to_test:

    ...

    try:
        batch_size_times[bs] = helper_utils.measure_average_epoch_time(loader, device)
    except RuntimeError as e:
        print(f"\n❌ ERROR with batch size {bs}. Likely a GPU memory issue.")
        batch_size_times[bs] = float('inf')

    del loader
    gc.collect()

    if torch.cuda.is_available():
        torch.cuda.empty_cache()

return batch_size_times
```

```python
for bs in batch_sizes_to_test:

    ...

    try:
        batch_size_times[bs] = helper_utils.measure_average_epoch_time(loader, device)
    except RuntimeError as e:
        print(f"\nX  ERROR with batch size {bs}. Likely a GPU memory issue.")
        batch_size_times[bs] = float('inf')

    del loader
    gc.collect()

    if torch.cuda.is_available():
        torch.cuda.empty_cache()

return batch_size_times
```

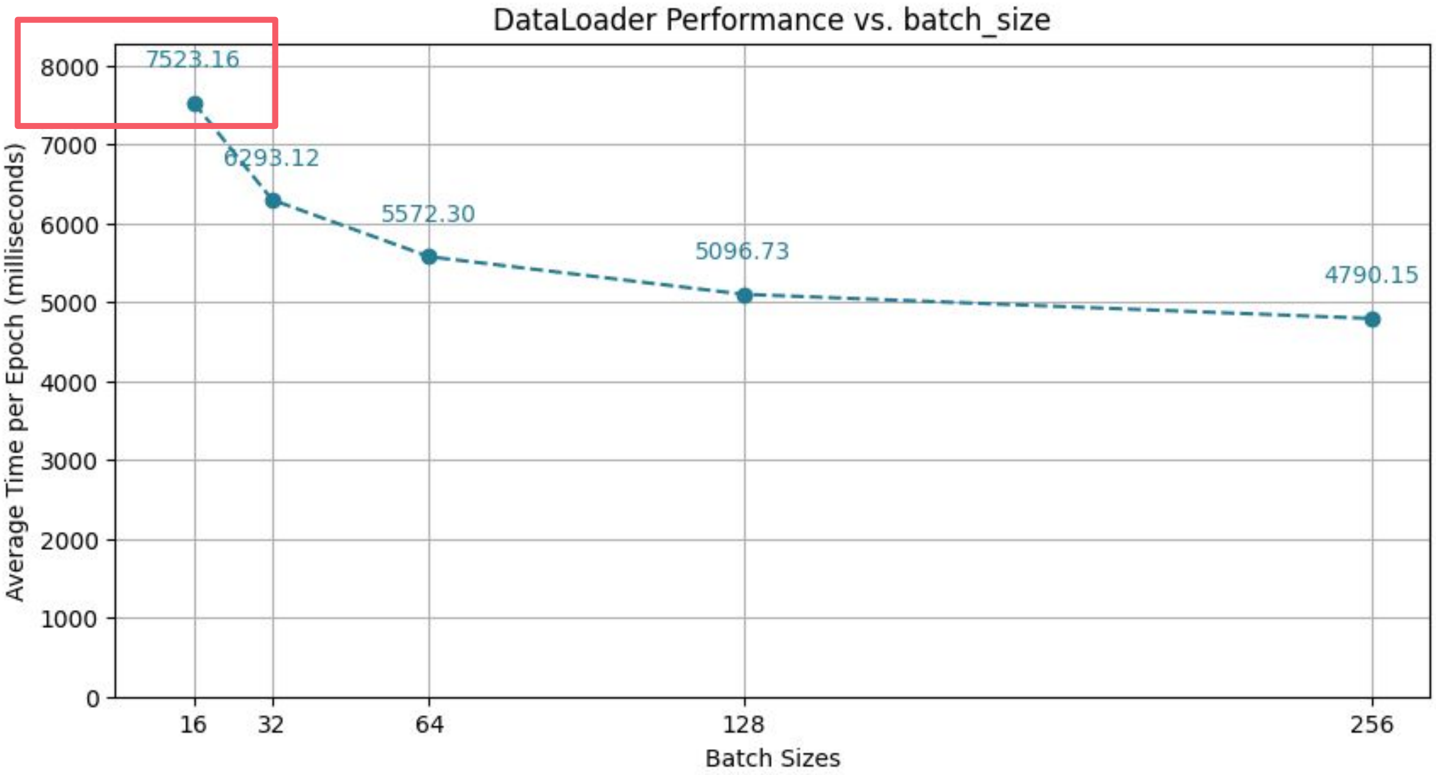
Effect of batch size on time per epoch



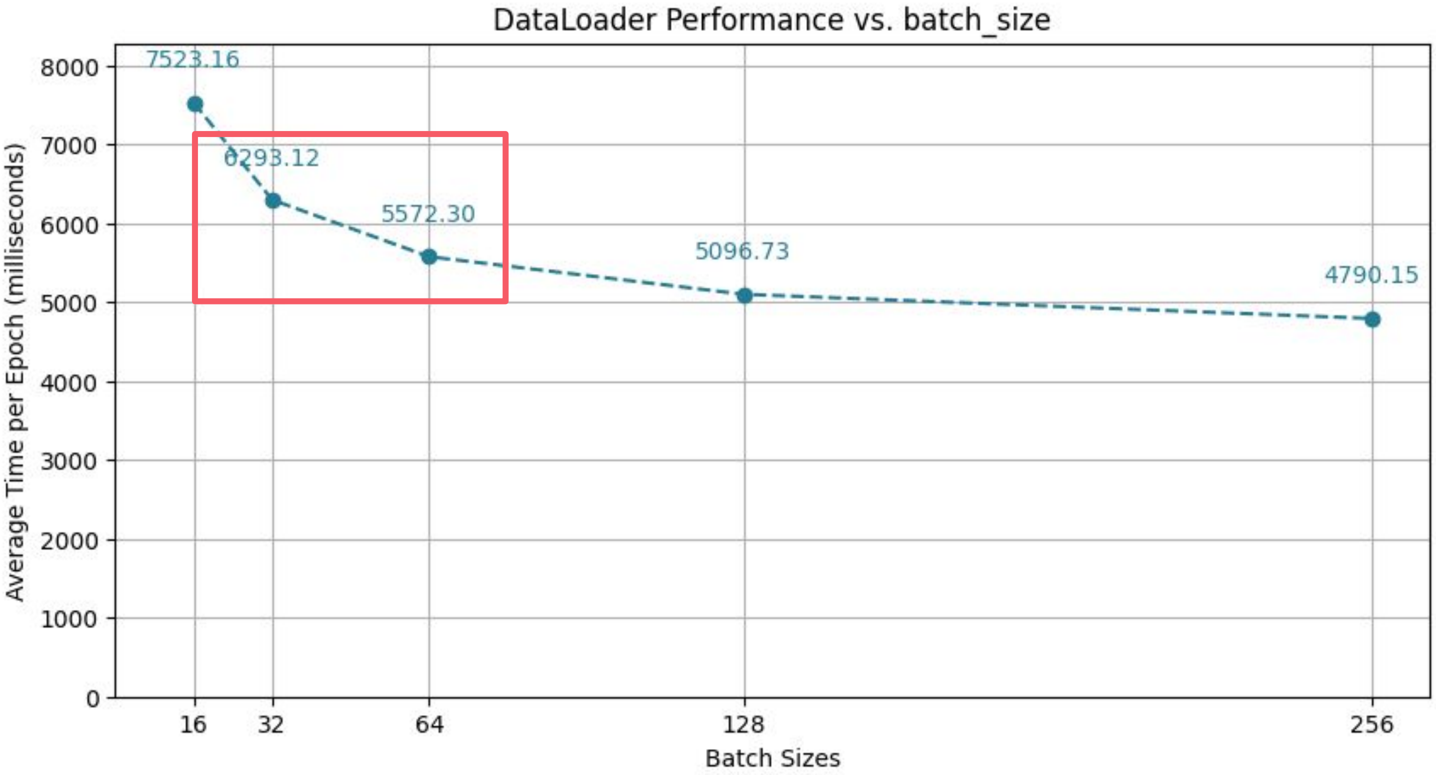
DeepLearning.AI

Laurence Moroney

Effect of batch size on time per epoch



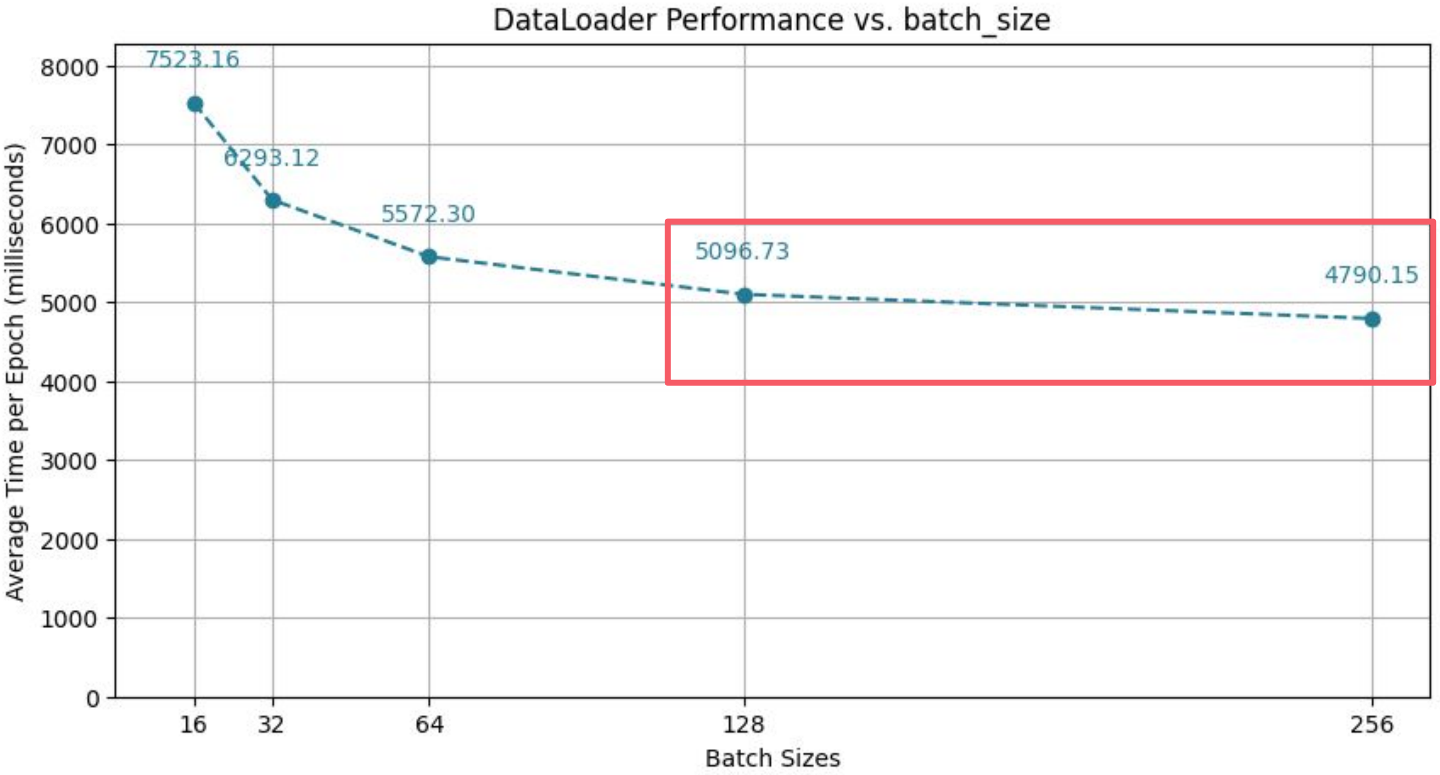
Effect of batch size on time per epoch



DeepLearning.AI

Laurence Moroney

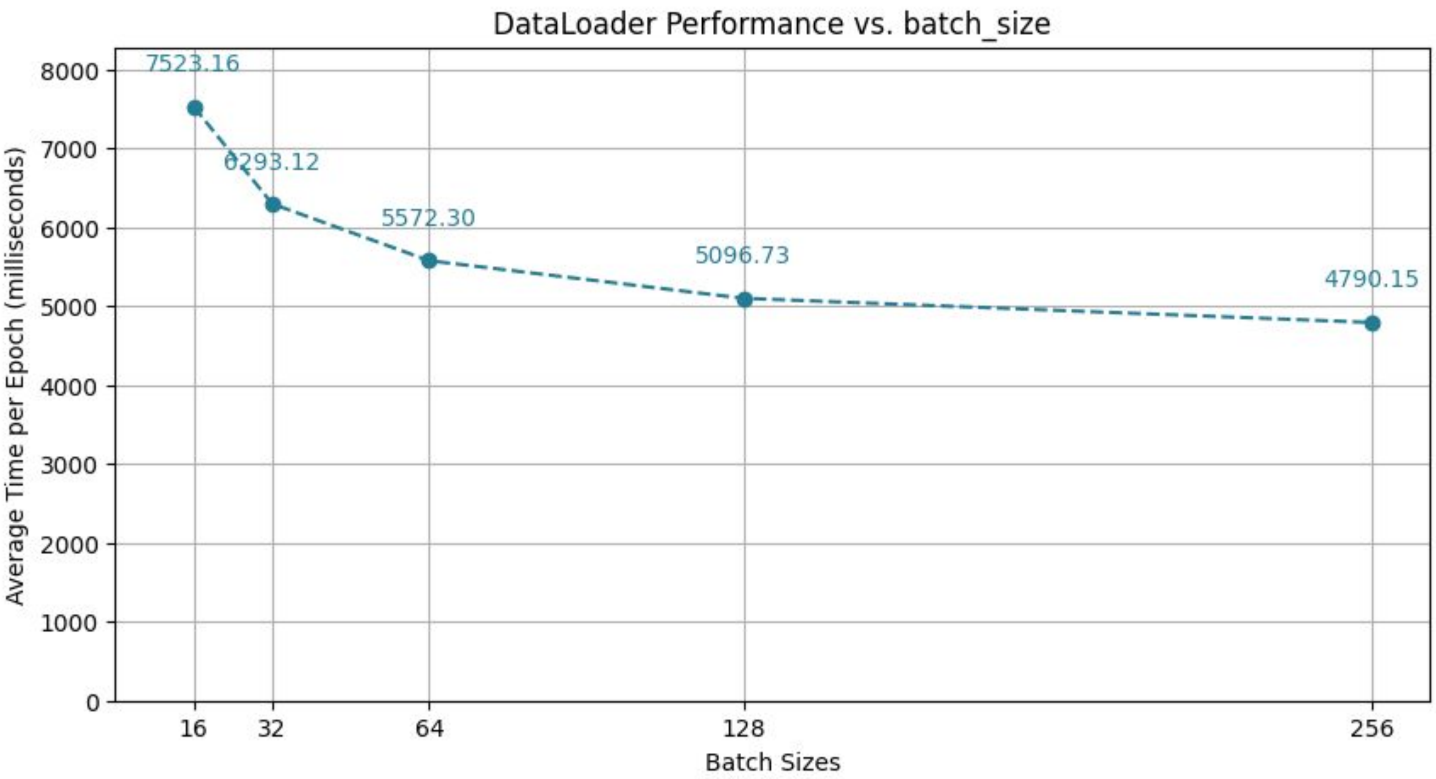
Effect of batch size on time per epoch



DeepLearning.AI

Laurence Moroney

Effect of batch size on time per epoch



Other parameters: `pin_memory`

Whether to use a special region of RAM that allows faster data transfer

Other parameters: `pin_memory`

Whether to use a special region of RAM that allows faster data transfer

Off

GPU does an extra copy step

Other parameters: `pin_memory`

Whether to use a special region of RAM that allows faster data transfer

Off

GPU does an extra copy step

On

GPU can access memory directly

Experimenting with pin_memory

```python
pin_memory_settings = [False, True]

def experiment_pin_memory(pin_memory_settings, trainset, device):

   pin_memory_times = {}

   for setting in pin_memory_settings:
       print(f"--- Testing with pin_memory = {setting} ---")

       loader = DataLoader(trainset,
                           batch_size=256,
                           num_workers=6,
                           shuffle=True,
                           pin_memory=setting
                       )
```

Experimenting with pin_memory

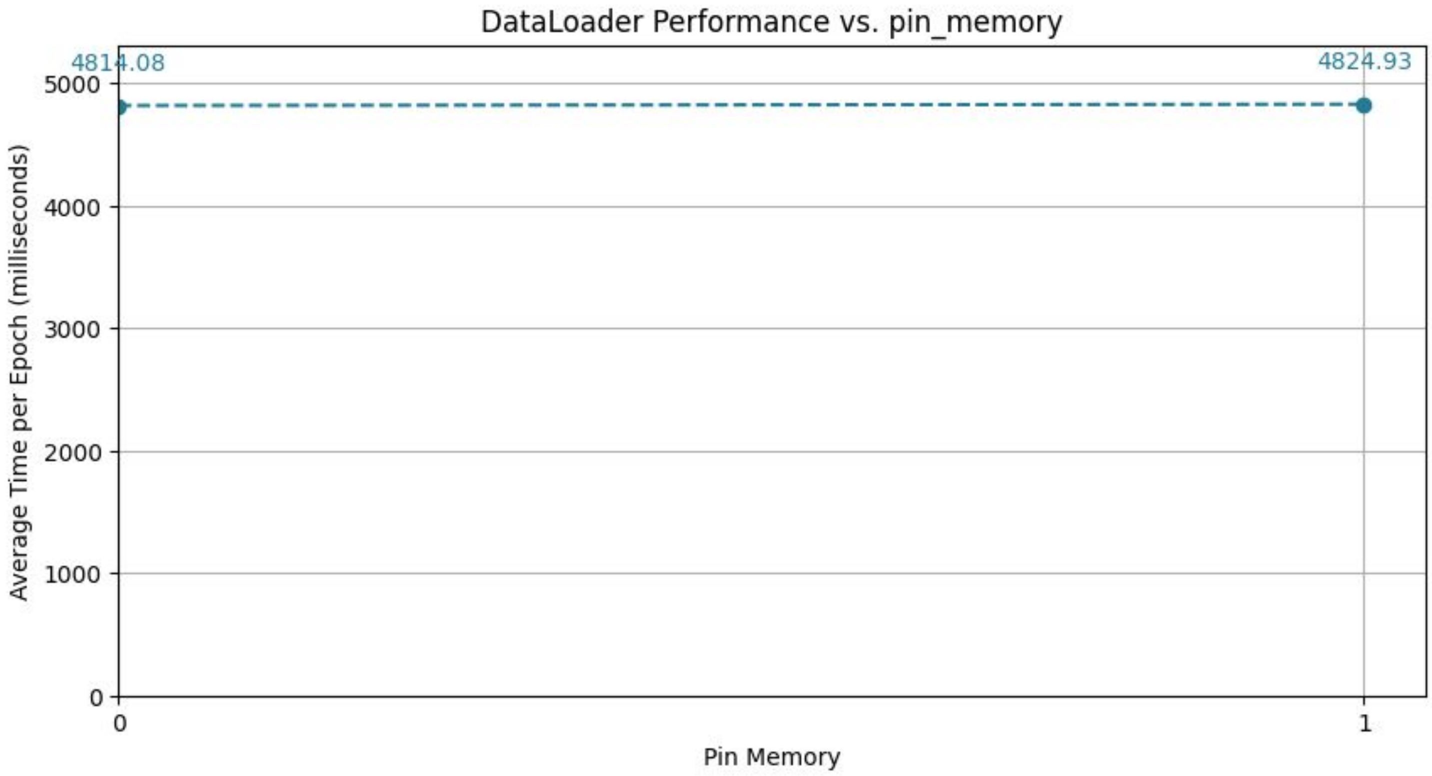
```
pin_memory_settings = [False, True]

def experiment_pin_memory(pin_memory_settings, trainset, device):

    pin_memory_times = {}

    for setting in pin_memory_settings:
        print(f"--- Testing with pin_memory = {setting} ---")

        loader = DataLoader(trainset,
                            batch_size=256,
                            num_workers=6,
                            shuffle=True,
                            pin_memory=setting
                       )
```

DeepLearning.AI

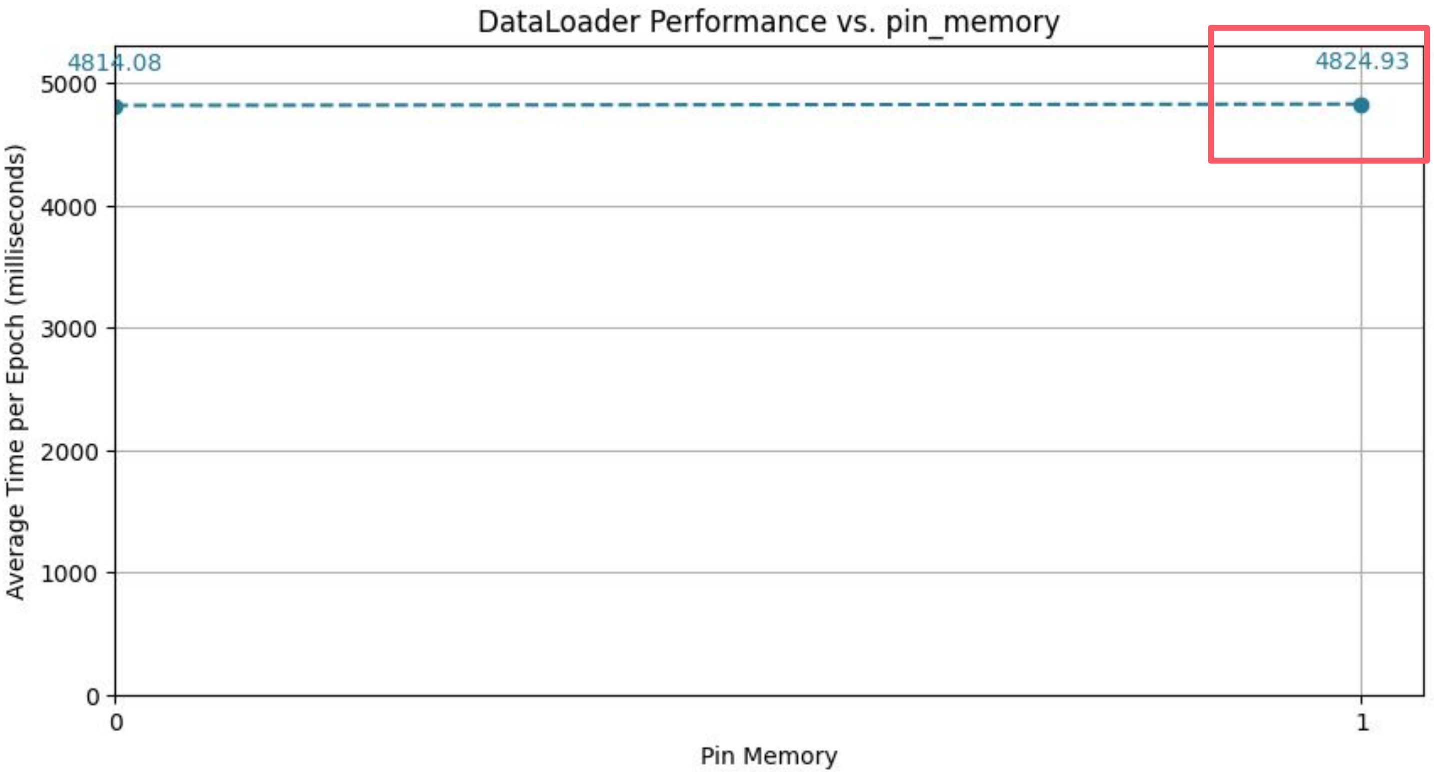
Laurence Moroney

```python
for setting in pin_memory_settings:

    ...

    try:
        pin_memory_times[setting] = helper_utils.measure_average_epoch_time(loader, device)
    except RuntimeError as e:
        print(f"\n❌ An error occurred with pin_memory = {setting}: {e}")
        pin_memory_times[setting] = float('inf')

    del loader
    gc.collect()
    if torch.cuda.is_available():
        torch.cuda.empty_cache()

return pin_memory_times
```

Effect of `pin_memory` on time per epoch



Effect of `pin_memory` on time per epoch



DeepLearning.AI

Laurence Moroney

Is `pin_memory` worth exploring?

Is `pin_memory` worth exploring?

Yes! Especially when combined with multiple workers

Other parameters: `prefetch_factor`

Controls how many batches each worker preloads into memory

Other parameters: `prefetch_factor`

Controls how many batches each worker preloads into memory

Default

```
prefetch_factor = 2
```

Other parameters: `prefetch_factor`

Controls how many batches each worker preloads into memory

Default

`prefetch_factor = 2`

`num_workers = 6`

12 batches preloaded

Experimenting with `prefetch_factor`

```
prefetch_factors_to_test = [2, 4, 6, 8, 10, 12]

def experiment_prefetch_factor(prefetch_factors_to_test, trainset, device):

    prefetch_factor_times = {}

    for pf in prefetch_factors_to_test:
        print(f"--- Testing prefetch_factor = {pf} ---")

        loader = DataLoader(trainset,
                            batch_size=256,
                            shuffle=True,
                            num_workers=6,
                            pin_memory=False,
                            prefetch_factor=pf
                  )
```

Experimenting with prefetch_factor

```
prefetch_factors_to_test = [2, 4, 6, 8, 10, 12]

def experiment_prefetch_factor(prefetch_factors_to_test, trainset, device):

    prefetch_factor_times = {}

    for pf in prefetch_factors_to_test:
        print(f"--- Testing prefetch_factor = {pf} ---")

        loader = DataLoader(trainset,
                            batch_size=256,
                            shuffle=True,
                            num_workers=6,
                            pin_memory=False,
                            prefetch_factor=pf
                        )
```

Experimenting with `prefetch_factor`

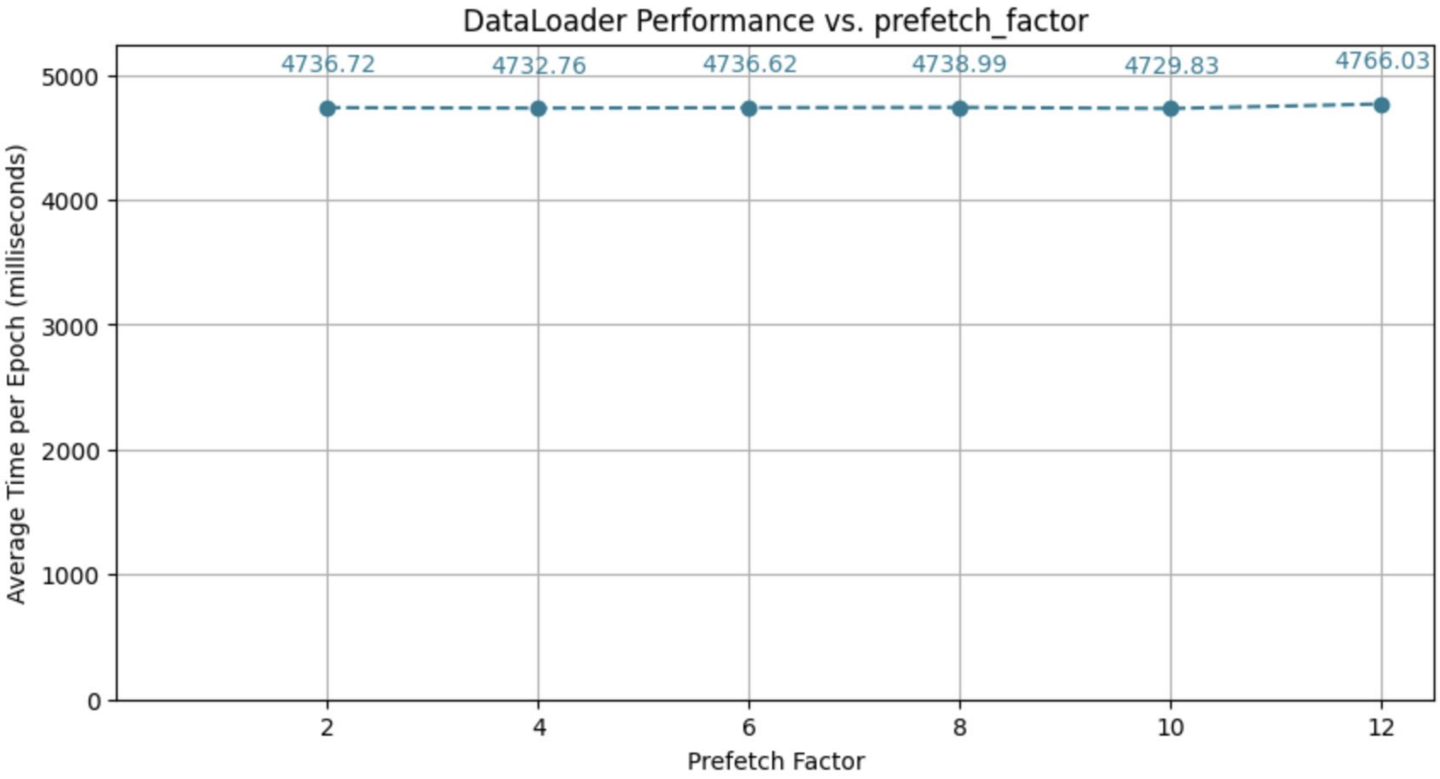
```
prefetch_factors_to_test = [2, 4, 6, 8, 10, 12]

def experiment_prefetch_factor(prefetch_factors_to_test, trainset, device):

   prefetch_factor_times = {}

   for pf in prefetch_factors_to_test:
       print(f"--- Testing prefetch_factor = {pf} ---")

       loader = DataLoader(trainset,
                           batch_size=256,
                           shuffle=True,
                           num_workers=6,
                           pin_memory=False,
                           prefetch_factor=pf
                      )
```


Laurence Moroney

```python
for pf in prefetch_factors_to_test:

    ...

        try:
            prefetch_factor_times[pf] = helper_utils.measure_average_epoch_time(loader,
device)
        except RuntimeError as e:
            print(f"\n❌ ERROR with prefetch_factor {pf}: {e}")
            prefetch_factor_times[pf] = float('inf')

        del loader
        gc.collect()

        if torch.cuda.is_available():
            torch.cuda.empty_cache()

    return prefetch_factor_times
```

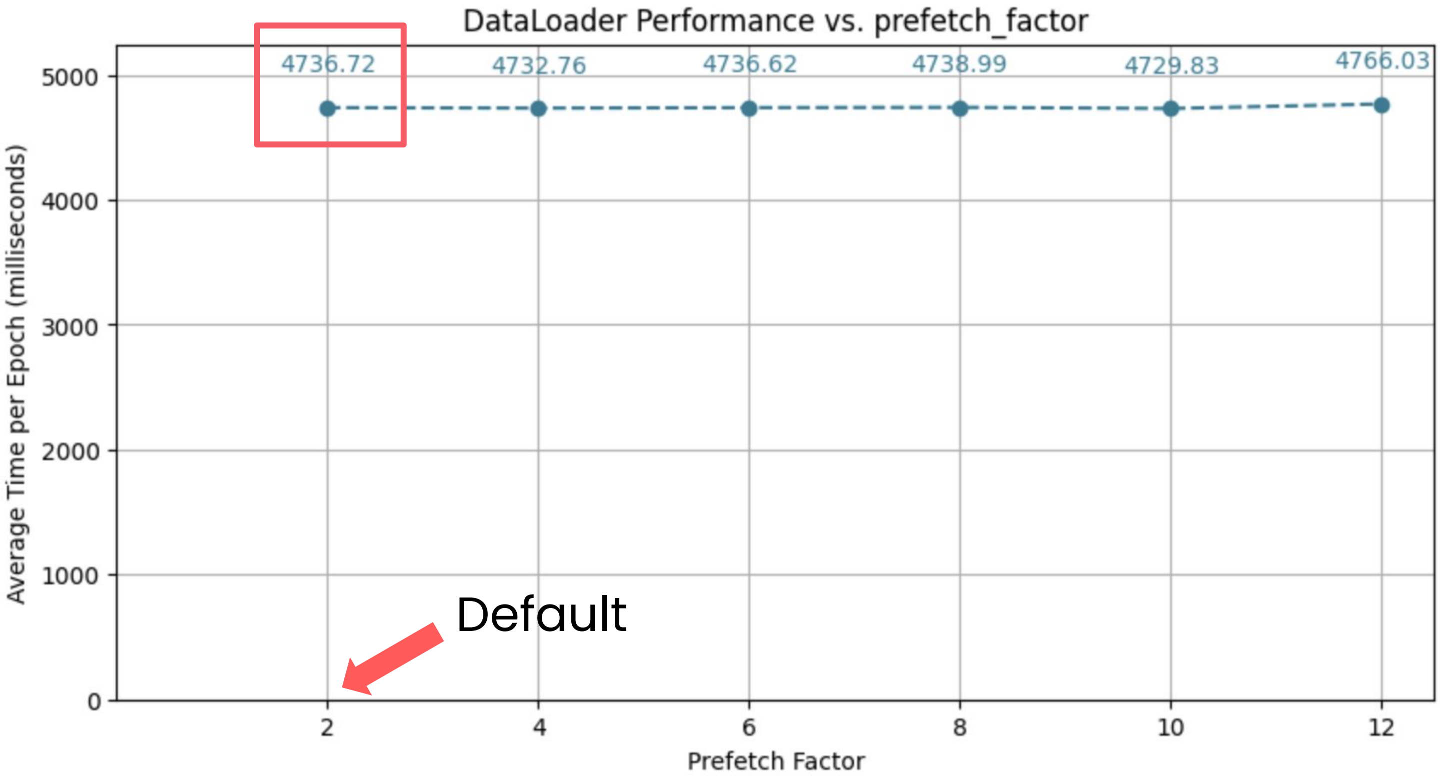
```python
    for pf in prefetch_factors_to_test:

        ...

        try:
            prefetch_factor_times[pf] = helper_utils.measure_average_epoch_time(loader, 
device)
        except RuntimeError as e:
            print(f"\nX ERROR with prefetch_factor {pf}: {e}")
            prefetch_factor_times[pf] = float('inf')

        del loader
        gc.collect()

        if torch.cuda.is_available():
            torch.cuda.empty_cache()

    return prefetch_factor_times
```

Effect of `prefetch_factor` on time per epoch



Laurence Moroney

Effect of `prefetch_factor` on time per epoch



Profiling

Efficient training pipelines

DeepLearning.AI

This module is about optimizing training time

Build efficient data pipelines

Profile training loops

Apply optimization techniques

Train faster without sacrificing accuracy

Train faster without sacrificing accuracy

Reduce
operation time

Train faster without sacrificing accuracy

Reduce
operation time

Better
GPU use

Train faster without sacrificing accuracy

Reduce operation time

Better GPU use

Avoid memory waste

Train faster without sacrificing accuracy

Reduce operation time

Better GPU use

Avoid memory waste

Identify where training is slow

The PyTorch Profiler is a diagnostic tool

It helps answer critical questions like:

The PyTorch Profiler is a diagnostic tool

It helps answer critical questions like:

Which operations take the most time?

The PyTorch Profiler is a diagnostic tool

It helps answer critical questions like:

Which operations take the most time?

Is the GPU fully in use?

The PyTorch Profiler is a diagnostic tool

It helps answer critical questions like:

Which operations take the most time?

Is the GPU fully in use?

Are there memory bottlenecks?

The PyTorch Profiler is a diagnostic tool

It helps answer critical questions like:

- Which operations take the most time?
- Is the GPU fully in use?
- Are there memory bottlenecks?
- Which lines of code cause issues?

Lightning: A framework to simplify training

Manual profiling

Need to repeat boilerplate code

Lightning: A framework to simplify training

Manual profiling

Need to repeat boilerplate code

Lightning

Takes care of the engineering details

Setting up the Profiler with Lightning

```
log_dir = "./profiler_output"

profiler = PyTorchProfiler(
    dirpath=log_dir,
    filename="profile_report",
    schedule=schedule(wait=2, warmup=2, active=10, repeat=1),
    profile_memory=True
)
```

Setting up the Profiler with Lightning

```
log_dir = "./profiler_output"

profiler = PyTorchProfiler(
    dirpath=log_dir,
    filename="profile_report",
    schedule=schedule(wait=2, warmup=2, active=10, repeat=1),
    profile_memory=True
)
```

Setting up the Profiler with Lightning

```
log_dir = "./profiler_output"

profiler = PyTorchProfiler(
    dirpath=log_dir,
    filename="profile_report",
    schedule=schedule(wait=2, warmup=2, active=10, repeat=1),
    profile_memory=True
)
```

Setting up the Profiler with Lightning

```
log_dir = "./profiler_output"

profiler = PyTorchProfiler(
    dirpath=log_dir,
    filename="profile_report",
    schedule=schedule(wait=2, warmup=2, active=10, repeat=1),
    profile_memory=True
)
```

Setting up the Profiler with Lightning

```
log_dir = "./profiler_output"

profiler = PyTorchProfiler(
    dirpath=log_dir,
    filename="profile_report",
    schedule=schedule(wait=2, warmup=2, active=10, repeat=1),
    profile_memory=True
)
```

Setting up the Profiler with Lightning

```
log_dir = "./profiler_output"

profiler = PyTorchProfiler(
    dirpath=log_dir,
    filename="profile_report",
    schedule=schedule(wait=2, warmup=2, active=10, repeat=1),
    profile_memory=True
)
```

Setting up the Profiler with Lightning

```
log_dir = "./profiler_output"

profiler = PyTorchProfiler(
    dirpath=log_dir,
    filename="profile_report",
    schedule=schedule(wait=2, warmup=2, active=10, repeat=1),
    profile_memory=True
)
```

Setting up the Profiler with Lightning

```
log_dir = "./profiler_output"

profiler = PyTorchProfiler(
    dirpath=log_dir,
    filename="profile_report",
    schedule=schedule(wait=2, warmup=2, active=10, repeat=1),
    profile_memory=True
)
```

Setting up the Profiler with Lightning

```
log_dir = "./profiler_output"

profiler = PyTorchProfiler(
    dirpath=log_dir,
    filename="profile_report",
    schedule=schedule(wait=2, warmup=2, active=10, repeat=1),
    profile_memory=True
)
```

Setting up the Profiler with Lightning

```
log_dir = "./profiler_output"

profiler = PyTorchProfiler(
    dirpath=log_dir,
    filename="profile_report",
    schedule=schedule(wait=2, warmup=2, active=10, repeat=1),
    profile_memory=True
)
```

Setting up the Profiler with Lightning

```
trainer = pl.Trainer(
    profiler=profiler,
    max_steps=14,
    accelerator="auto",
    devices=1,
    logger=False,
    enable_model_summary=False
    enable_checkpointing=False
)
```

Setting up the Profiler with Lightning

```
trainer = pl.Trainer(
    profiler=profiler,
    max_steps=14,
    accelerator="auto",
    devices=1,
    logger=False,
    enable_model_summary=False
    enable_checkpointing=False
)
```

Setting up the Profiler with Lightning

```
trainer = pl.Trainer(
    profiler=profiler,
    max_steps=14,
    accelerator="auto",
    devices=1,
    logger=False,
    enable_model_summary=False
    enable_checkpointing=False
)
```

Setting up the Profiler with Lightning

```
trainer = pl.Trainer(
    profiler=profiler,
    max_steps=14,
    accelerator="auto",
    devices=1,
    logger=False,
    enable_model_summary=False
    enable_checkpointing=False
)
```

Setting up the Profiler with Lightning

```
trainer = pl.Trainer(
    profiler=profiler,
    max_steps=14,
    accelerator="auto",
    devices=1,
    logger=False,
    enable_model_summary=False
    enable_checkpointing=False
)
```


Laurence Moroney

Setting up the Profiler with Lightning

```
trainer = pl.Trainer(
    profiler=profiler,
    max_steps=14,
    accelerator="auto",
    devices=1,
    logger=False,
    enable_model_summary=False
    enable_checkpointing=False
)
```

Setting up the Profiler with Lightning

```
trainer = pl.Trainer(
    profiler=profiler,
    max_steps=14,
    accelerator="auto",
    devices=1,
    logger=False,
    enable_model_summary=False
    enable_checkpointing=False
)
```

Running a diagnosis

```
# Create an instance of the LightningModule.
model_baseline = CIFAR10LightningModule()

# Instantiate the DataModule (2 workers).
dm_loader = CIFAR10DataModule(num_workers=2)

# Start the training and profiling run.
trainer.fit(model_baseline, dm_loader)

# Print a confirmation message when done.
print("\nProfiling Complete!\n")
```

Running a diagnosis

```
# Create an instance of the LightningModule.
model_baseline = CIFAR10LightningModule()

# Instantiate the DataModule (2 workers).
dm_loader = CIFAR10DataModule(num_workers=2)

# Start the training and profiling run.
trainer.fit(model_baseline, dm_loader)

# Print a confirmation message when done.
print("\nProfiling Complete!\n")
```

Running a diagnosis

```
# Create an instance of the LightningModule.
model_baseline = CIFAR10LightningModule()

# Instantiate the DataModule (2 workers).
dm_loader = CIFAR10DataModule(num_workers=2)

# Start the training and profiling run.
trainer.fit(model_baseline, dm_loader)

# Print a confirmation message when done.
print("\nProfiling Complete!\n")
```

Running a diagnosis

```
# Create an instance of the LightningModule.
model_baseline = CIFAR10LightningModule()

# Instantiate the DataModule (2 workers).
dm_loader = CIFAR10DataModule(num_workers=2)

# Start the training and profiling run.
trainer.fit(model_baseline, dm_loader)

# Print a confirmation message when done.
print("\nProfiling Complete!\n")
```

Profiling results

Information on operations

- How long they took
- How many times they were called

Profiling results

Information on operations

- How long they took
- How many times they were called

Memory usage

- CPU-GPU transfer latency
- Stack traces

Profiling results

Row	Operation Sequence	Action	Total Time (ms)	Calls
1	1	ProfilerStep*	315.034008	10
2	92	aten::convolution_backward	4.439983	30
3	154	aten::conv2d	4.393448	30
4	35	autograd::engine::evaluate_function: AddmmBackward0	2.990377	20
5	36	AddmmBackward0	2.041308	20

Profiling results

Row	Operation Sequence	Action	Total Time (ms)	Calls
1	1	ProfilerStep*	315.034008	10
2	92	aten::convolution_backward	4.439983	30
3	154	aten::conv2d	4.393448	30
4	35	autograd::engine::evaluate_function: AddmmBackward0	2.990377	20
5	36	AddmmBackward0	2.041308	20

Profiling results

Row	Operation Sequence	Action	Total Time (ms)	Calls
1	1	ProfilerStep*	315.034008	10
2	92	aten::convolution_backward	4.439983	30
3	154	aten::conv2d	4.393448	30
4	35	autograd::engine::evaluate_function: AddmmBackward0	2.990377	20
5	36	AddmmBackward0	2.041308	20

Try a more efficient model

Parameter	Baseline Model	Efficient Model
conv_channels	256, 512, 1024	32, 64, 128
linear_features	2048	512

Try a more efficient model

Parameter	Baseline Model	Efficient Model
conv_channels	256, 512, 1024	32, 64, 128
linear_features	2048	512

Try a more efficient model

Parameter	Baseline Model	Efficient Model
conv_channels	256, 512, 1024	32, 64, 128
linear_features	2048	512

Baseline vs. efficient model

Metric	Baseline Model	Efficient Model
ProfilerStep* Time (ms)	315.034008	189.495232

Baseline vs. efficient model

Metric	Baseline Model	Efficient Model
ProfilerStep* Time (ms)	315.034008	189.495232
Training Accuracy (%)	87.77	80.31
Validation Accuracy (%)	75.68	74.74

Optimizing Training Loops

Efficient training pipelines

DeepLearning.AI

Try two optimization techniques:

Mixed precision

- Combine 16-bit and 32-bit
- Reduce memory
- Preserve accuracy

Try two optimization techniques:

Mixed precision

- Combine 16-bit and 32-bit
- Reduce memory
- Preserve accuracy

Gradient accumulation

- Larger effective batch size
- Several mini-batches
- Accumulate gradients
- Single weight update

Setting the stage for optimization experiments

A custom callback

Setting the stage for optimization experiments

A custom callback

A training function

Setting the stage for optimization experiments

A custom callback

A training function

A `run_optimization()` function

Test 6 different configurations

Name	precision	grad_accum	batch_size	Effective batch size
Standard	32-true	1	256	256
Mixed precision	16-mixed	1	256	256
Gradient accumulation 256-128	32-true	2	128	256
Gradient accumulation 256-64	32-true	4	64	256
Combined 256-128	16-mixed	2	128	256
Combined 256-64	16-mixed	4	64	256

Test 6 different configurations

Name	precision	grad_accum	batch_size	Effective batch size
Standard	32-true	1	256	256
Mixed precision	16-mixed	1	256	256
Gradient accumulation 256-128	32-true	2	128	256
Gradient accumulation 256-64	32-true	4	64	256
Combined 256-128	16-mixed	2	128	256
Combined 256-64	16-mixed	4	64	256

Test 6 different configurations

Name	precision	grad_accum	batch_size	Effective batch size
Standard	32-true	1	256	256
Mixed precision	16-mixed	1	256	256
Gradient accumulation 256-128	32-true	2	128	256
Gradient accumulation 256-64	32-true	4	64	256
Combined 256-128	16-mixed	2	128	256
Combined 256-64	16-mixed	4	64	256

Test 6 different configurations

Name	precision	grad_accum	batch_size	Effective batch size
Standard	32-true	1	256	256
Mixed precision	16-mixed	1	256	256
Gradient accumulation 256-128	32-true	2	128	256
Gradient accumulation 256-64	32-true	4	64	256
Combined 256-128	16-mixed	2	128	256
Combined 256-64	16-mixed	4	64	256

Test 6 different configurations

Name	precision	grad_accum	batch_size	Effective batch size
Standard	32-true	1	256	256
Mixed precision	16-mixed	1	256	256
Gradient accumulation 256-128	32-true	2	128	256
Gradient accumulation 256-64	32-true	4	64	256
Combined 256-128	16-mixed	2	128	256
Combined 256-64	16-mixed	4	64	256

Test 6 different configurations

Name	precision	grad_accum	batch_size	Effective batch size
Standard	32-true	1	256	256
Mixed precision	16-mixed	1	256	256
Gradient accumulation 256-128	32-true	2	128	256
Gradient accumulation 256-64	32-true	4	64	256
Combined 256-128	16-mixed	2	128	256
Combined 256-64	16-mixed	4	64	256

Test 6 different configurations

Name	precision	grad_accum	batch_size	Effective batch size
Standard	32-true	1	256	256
Mixed precision	16-mixed	1	256	256
Gradient accumulation 256-128	32-true	2	128	256
Gradient accumulation 256-64	32-true	4	64	256
Combined 256-128	16-mixed	2	128	256
Combined 256-64	16-mixed	4	64	256

```python
# Data module setup
batch_size = 128
data_module = CIFAR10DataModule(batch_size=batch_size, num_workers=num_workers)

# Gradient accumulation setup
effective_batch_size = 256
grad_accum = effective_batch_size // batch_size

res, p_data = run_optimization(
    name="Gradient Accumulation (Effective BS: 256-128)",
    precision="32-true",
    grad_accum=grad_accum,
    data_module=data_module,
    num_epochs=num_epochs,
)

results.append(res)
plot_data.append(p_data)
```

```python
# Data module setup
batch_size = 128
data_module = CIFAR10DataModule(batch_size=batch_size, num_workers=num_workers)

# Gradient accumulation setup
effective_batch_size = 256
grad_accum = effective_batch_size // batch_size

res, p_data = run_optimization(
    name="Gradient Accumulation (Effective BS: 256-128)",
    precision="32-true",
    grad_accum=grad_accum,
    data_module=data_module,
    num_epochs=num_epochs,
)

results.append(res)
plot_data.append(p_data)
```

```
# Data module setup
batch_size = 128
data_module = CIFAR10DataModule(batch_size=batch_size, num_workers=num_workers)

# Gradient accumulation setup
effective_batch_size = 256
grad_accum = effective_batch_size // batch_size

res, p_data = run_optimization(
   name="Gradient Accumulation (Effective BS: 256-128)",
   precision="32-true",
   grad_accum=grad_accum,
   data_module=data_module,
   num_epochs=num_epochs,
)

results.append(res)
plot_data.append(p_data)
```

```python
# Data module setup
batch_size = 128
data_module = CIFAR10DataModule(batch_size=batch_size, num_workers=num_workers)

# Gradient accumulation setup
effective_batch_size = 256
grad_accum = effective_batch_size // batch_size

res, p_data = run_optimization(
    name="Gradient Accumulation (Effective BS: 256-128)",
    precision="32-true",
    grad_accum=grad_accum,
    data_module=data_module,
    num_epochs=num_epochs,
)

results.append(res)
plot_data.append(p_data)
```

```
# Data module setup
batch_size = 128
data_module = CIFAR10DataModule(batch_size=batch_size, num_workers=num_workers)

# Gradient accumulation setup
effective_batch_size = 256
grad_accum = effective_batch_size // batch_size

res, p_data = run_optimization(
    name="Gradient Accumulation (Effective BS: 256-128)",
    precision="32-true",
    grad_accum=grad_accum,
    data_module=data_module,
    num_epochs=num_epochs,
)

results.append(res)
plot_data.append(p_data)
```

```python
# Data module setup
batch_size = 128
data_module = CIFAR10DataModule(batch_size=batch_size, num_workers=num_workers)

# Gradient accumulation setup
effective_batch_size = 256
grad_accum = effective_batch_size // batch_size

res, p_data = run_optimization(
    name="Gradient Accumulation (Effective BS: 256-128)",
    precision="32-true",
    grad_accum=grad_accum,
    data_module=data_module,
    num_epochs=num_epochs,
)

results.append(res)
plot_data.append(p_data)
```

```python
# Data module setup
batch_size = 128
data_module = CIFAR10DataModule(batch_size=batch_size, num_workers=num_workers)

# Gradient accumulation setup
effective_batch_size = 256
grad_accum = effective_batch_size // batch_size

res, p_data = run_optimization(
    name="Gradient Accumulation (Effective BS: 256-128)",
    precision="32-true",
    grad_accum=grad_accum,
    data_module=data_module,
    num_epochs=num_epochs,
)

results.append(res)
plot_data.append(p_data)
```

```python
# Data module setup
batch_size = 128
data_module = CIFAR10DataModule(batch_size=batch_size, num_workers=num_workers)

# Gradient accumulation setup
effective_batch_size = 256
grad_accum = effective_batch_size // batch_size

res, p_data = run_optimization(
    name="Gradient Accumulation (Effective BS: 256-128)",
    precision="32-true",
    grad_accum=grad_accum,
    data_module=data_module,
    num_epochs=num_epochs,
)

results.append(res)
plot_data.append(p_data)
```

```
# Data module setup
batch_size = 128
data_module = CIFAR10DataModule(batch_size=batch_size, num_workers=num_workers)

# Gradient accumulation setup
effective_batch_size = 256
grad_accum = effective_batch_size // batch_size

res, p_data = run_optimization(
    name="Gradient Accumulation (Effective BS: 256-128)",
    precision="32-true",
    grad_accum=grad_accum,
    data_module=data_module,
    num_epochs=num_epochs,
)

results.append(res)
plot_data.append(p_data)
```

Optimization measured by 2 metrics

Optimization measured by 2 metrics

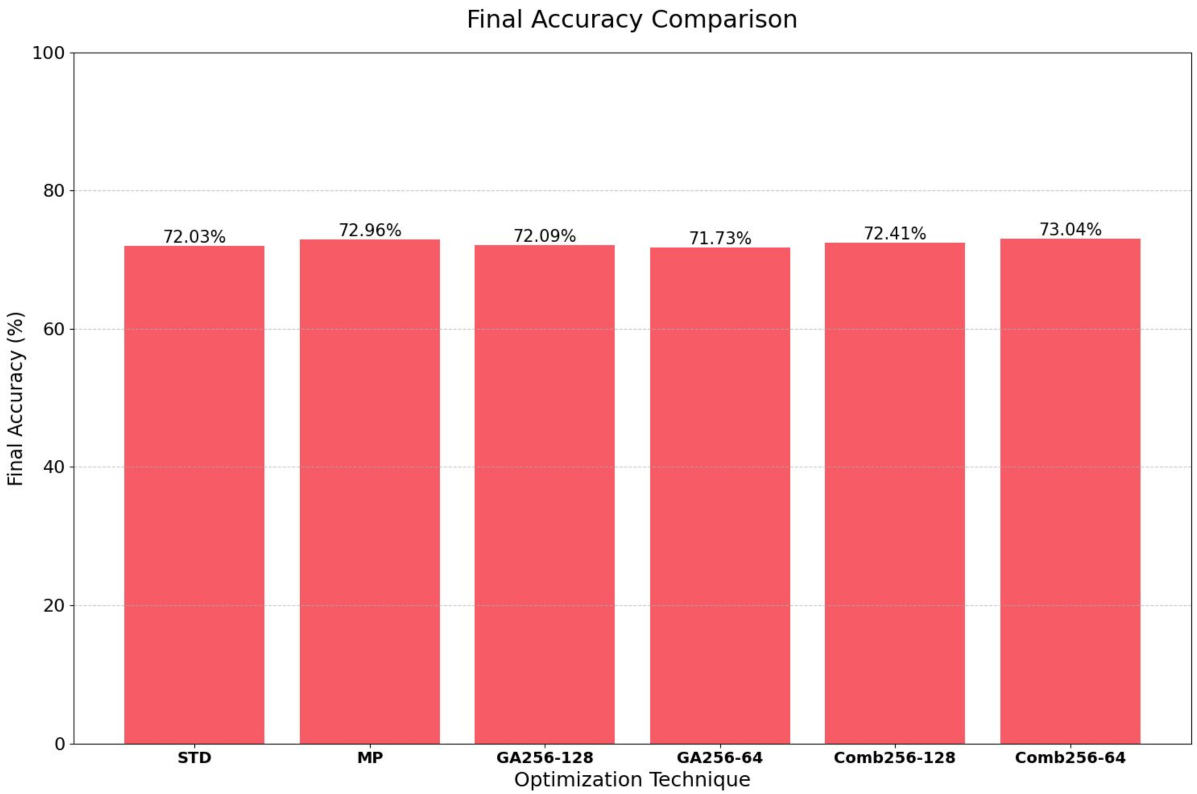
Validation
accuracy

Optimization measured by 2 metrics

Validation accuracy

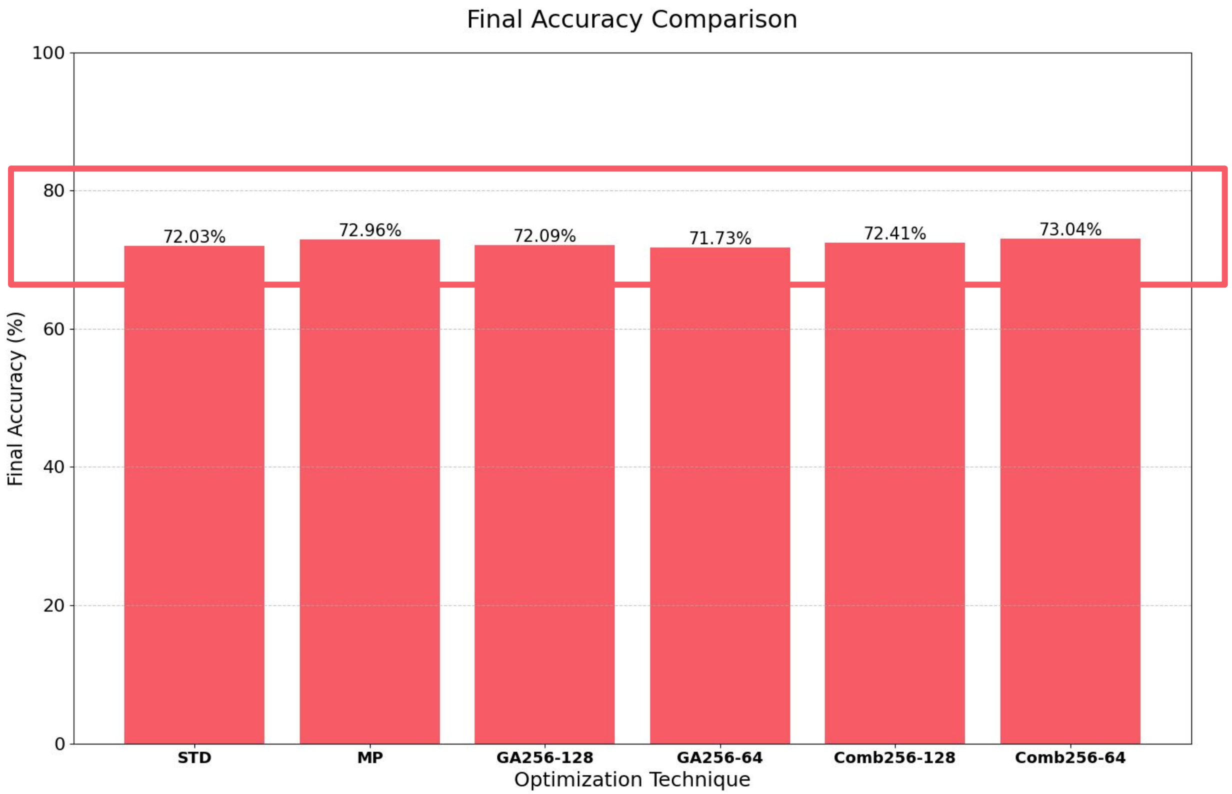
Peak memory usage

Validation accuracy



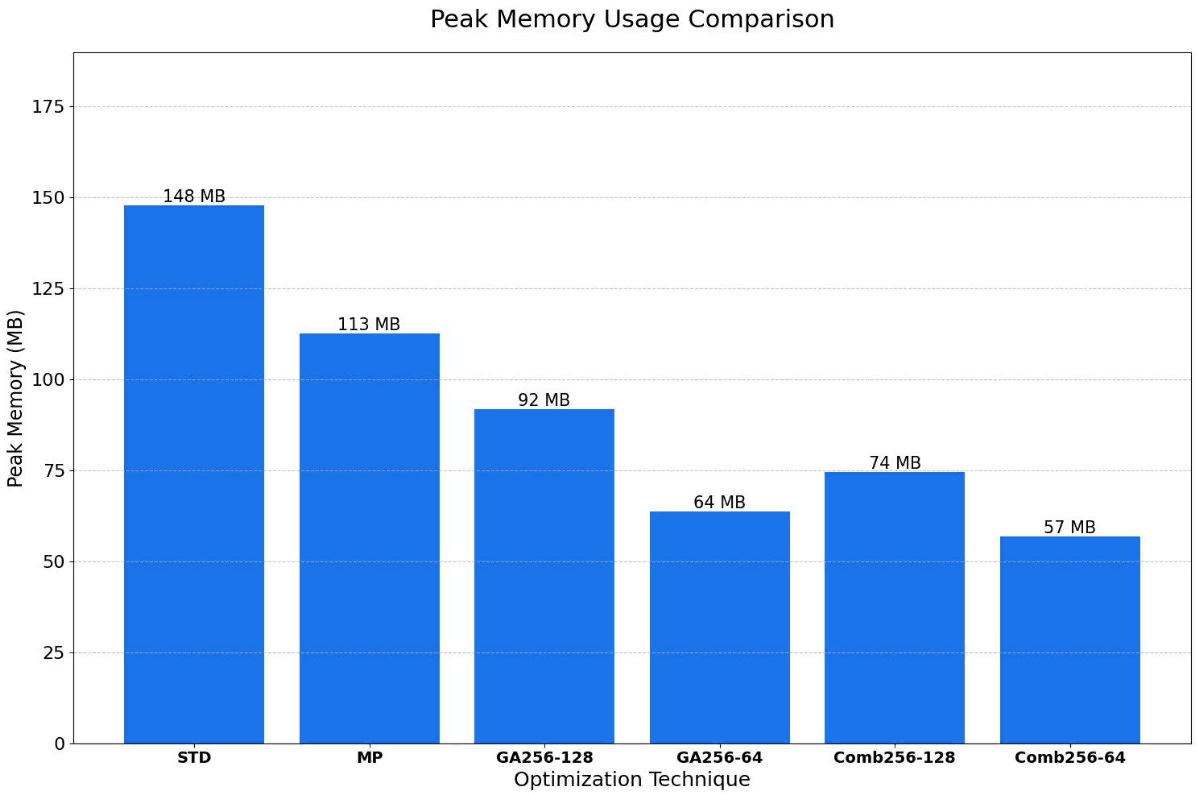
Label	Configuration
STD	Standard
MP	Mixed precision
GA256-128	Gradient accumulation (Effective BS: 256-128)
GA256-64	Gradient accumulation (Effective BS: 256-64)
Comb256-128	Combined (Effective BS: 256-128)
Comb256-64	Combined (Effective BS: 256-64)

Validation accuracy



Label	Configuration
STD	Standard
MP	Mixed precision
GA256-128	Gradient accumulation (Effective BS: 256-128)
GA256-64	Gradient accumulation (Effective BS: 256-64)
Comb256-128	Combined (Effective BS: 256-128)
Comb256-64	Combined (Effective BS: 256-64)

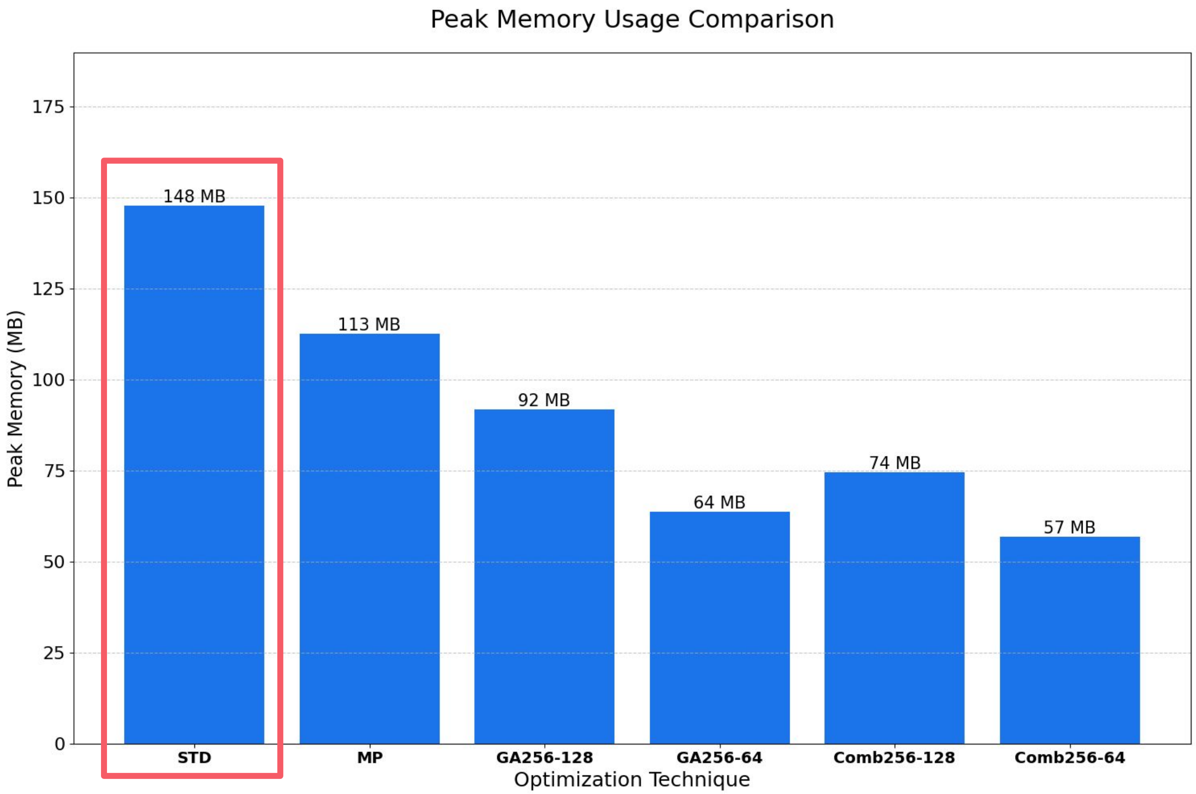
Peak memory usage



Label	Configuration
STD	Standard
MP	Mixed precision
GA256-128	Gradient accumulation (Effective BS: 256-128)
GA256-64	Gradient accumulation (Effective BS: 256-64)
Comb256-128	Combined (Effective BS: 256-128)
Comb256-64	Combined (Effective BS: 256-64)

Laurence Moroney

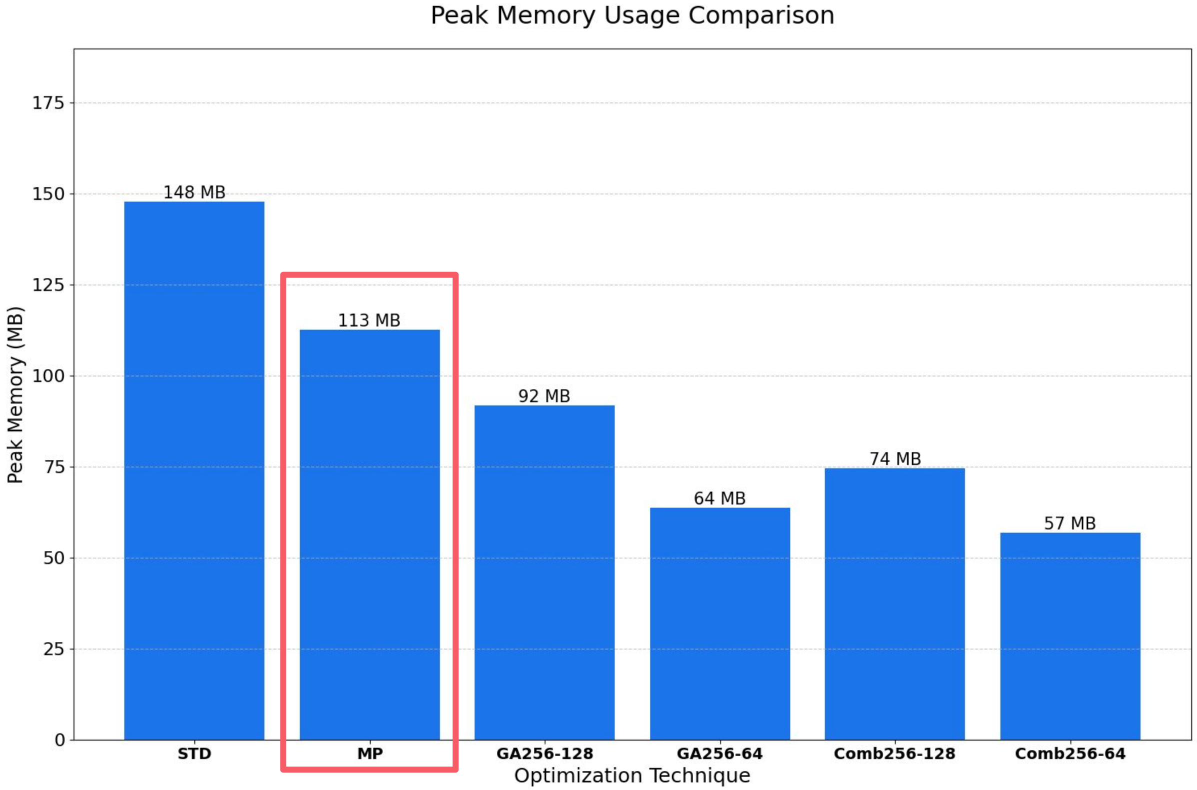
Peak memory usage



Label	Configuration
STD	Standard
MP	Mixed precision
GA256-128	Gradient accumulation (Effective BS: 256-128)
GA256-64	Gradient accumulation (Effective BS: 256-64)
Comb256-128	Combined (Effective BS: 256-128)
Comb256-64	Combined (Effective BS: 256-64)

Laurence Moroney

Peak memory usage

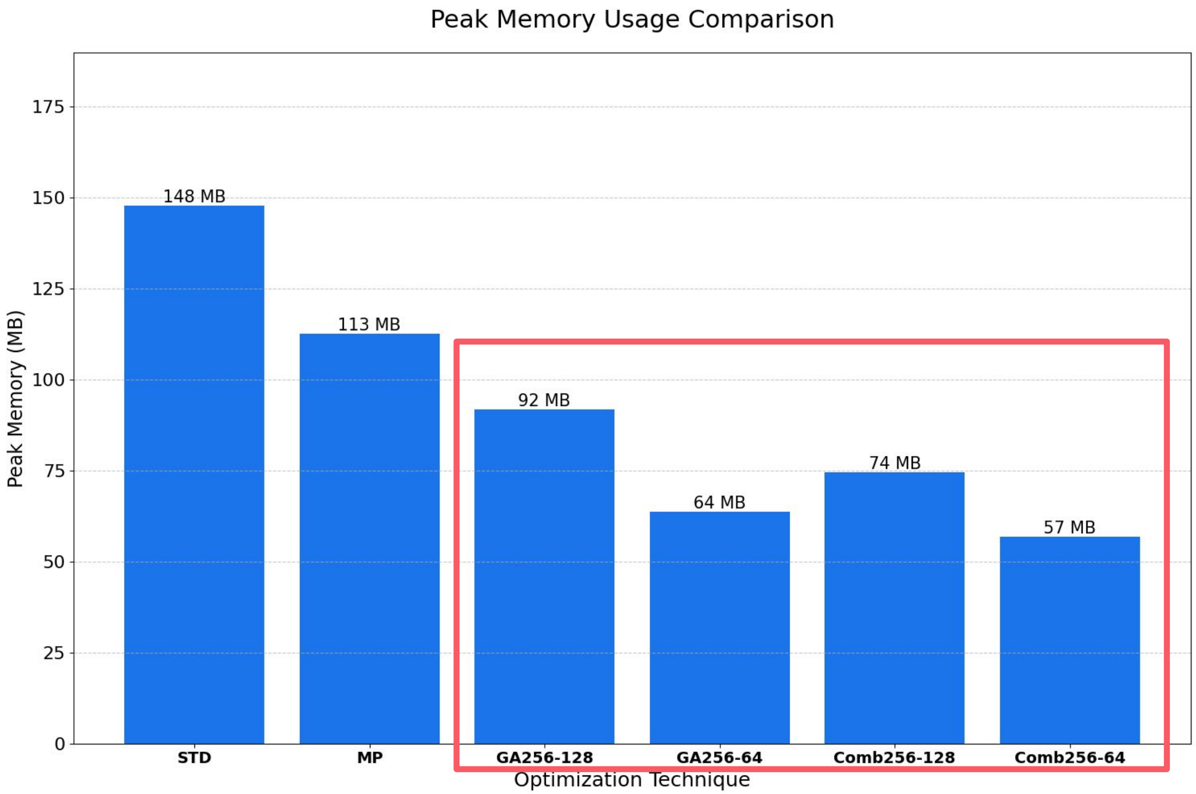


Label	Configuration
STD	Standard
MP	Mixed precision
GA256-128	Gradient accumulation (Effective BS: 256-128)
GA256-64	Gradient accumulation (Effective BS: 256-64)
Comb256-128	Combined (Effective BS: 256-128)
Comb256-64	Combined (Effective BS: 256-64)

DeepLearning.AI

Laurence Moroney

Peak memory usage



Label	Configuration
STD	Standard
MP	Mixed precision
GA256-128	Gradient accumulation (Effective BS: 256-128)
GA256-64	Gradient accumulation (Effective BS: 256-64)
Comb256-128	Combined (Effective BS: 256-128)
Comb256-64	Combined (Effective BS: 256-64)

What does this tell us?

Accuracy is resilient

What does this tell us?

Accuracy is resilient

Gradient accumulation reduces memory usage

Remember:

Results will vary when you run this experiment

DeepLearning.AI

What Else Can You Do with Lightning?

Efficient training pipelines

Why has Lightning become so prevalent?

Separate logic
from training
infrastructure

Why has Lightning become so prevalent?

Separate logic from training infrastructure

Consistency

Why has Lightning become so prevalent?

Separate logic from training infrastructure

Consistency

Built for scale

Why has Lightning become so prevalent?

Separate logic from training infrastructure

Consistency

Built for scale

Integration with tools

Why has Lightning become so prevalent?

Separate logic from training infrastructure

Consistency

Built for scale

Integration with tools

This is what you learned in this course:

This is what you learned in this course:

Module 1: **Hyperparameter Optimization**

This is what you learned in this course:

Module 1: **Hyperparameter Optimization**

Module 2: **Working with Images Using TorchVision**

This is what you learned in this course:

Module 1: **Hyperparameter Optimization**

Module 2: **Working with Images Using TorchVision**

Module 3: **Working with Text Using Hugging Face**

This is what you learned in this course:

Module 1: **Hyperparameter Optimization**

Module 2: **Working with Images Using TorchVision**

Module 3: **Working with Text Using Hugging Face**

Module 4: **Efficient Training Pipelines**

This is what you learned in this course:

Module 1: **Hyperparameter Optimization**

Module 2: **Working with Images Using TorchVision**

Module 3: **Working with Text Using Hugging Face**

Module 4: **Efficient Training Pipelines**

Get ready for the next course!

PyTorch: Advanced Architectures and Deployment

DeepLearning.AI

Laurence Moroney