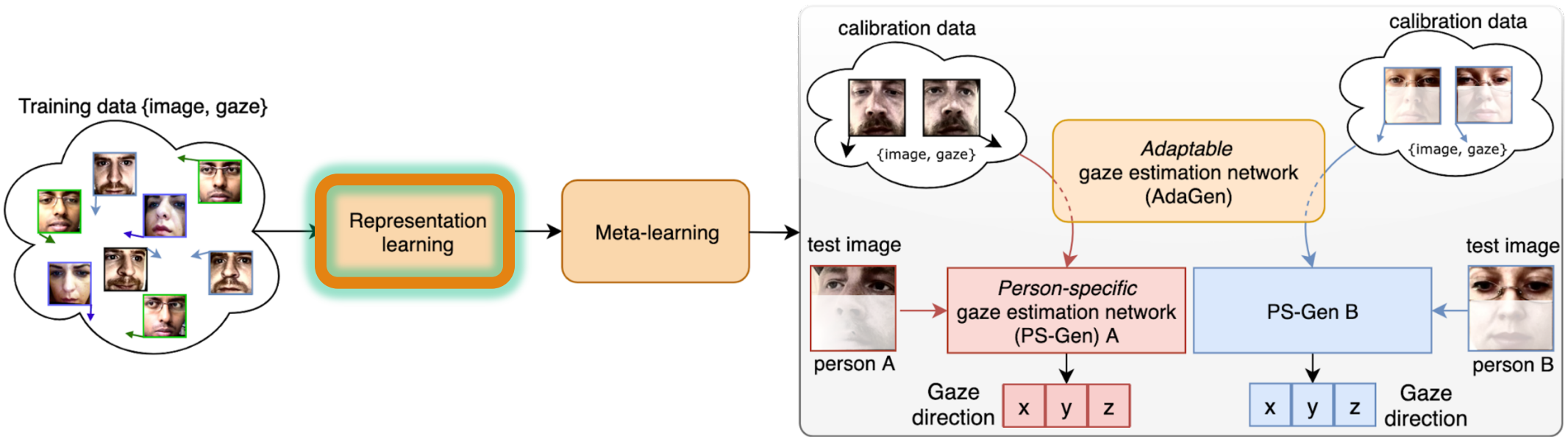
MIXED PRECISION TRAINING FOR FAZE: FEW-SHOT ADAPTIVE GAZE ESTIMATION

Shalini De Mello

with Sangkug Lym and Dusan Stosic

APPLICATION

FAZE: Few-shot Adaptive Gaze Estimation*✧

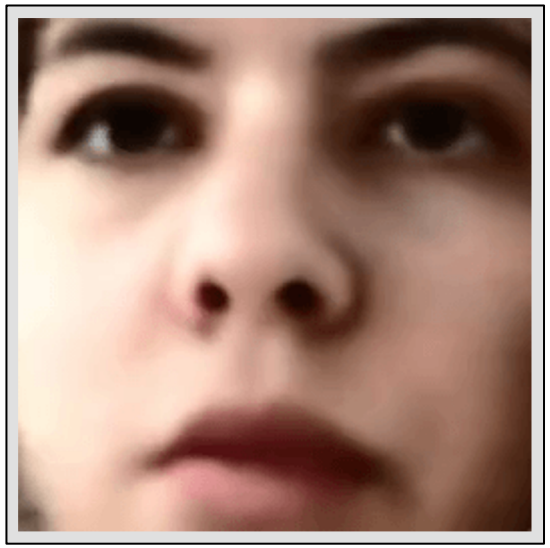


*Park *et al.*, ICCV 2019, ✧Code: https://github.com/NVlabs/few_shot_gaze

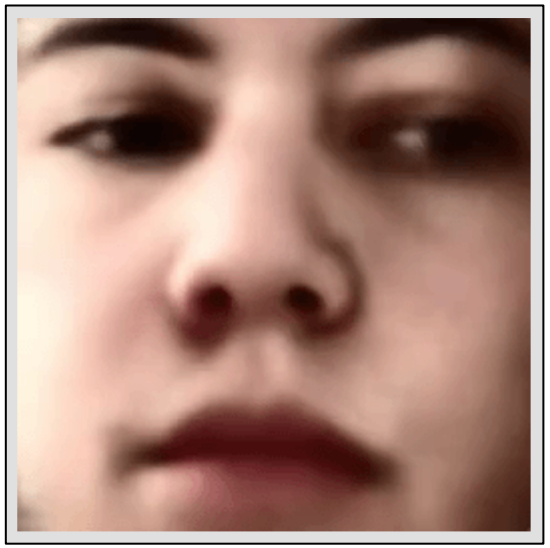
REPRESENTATION LEARNING

Gaze and Head Pose Re-direction

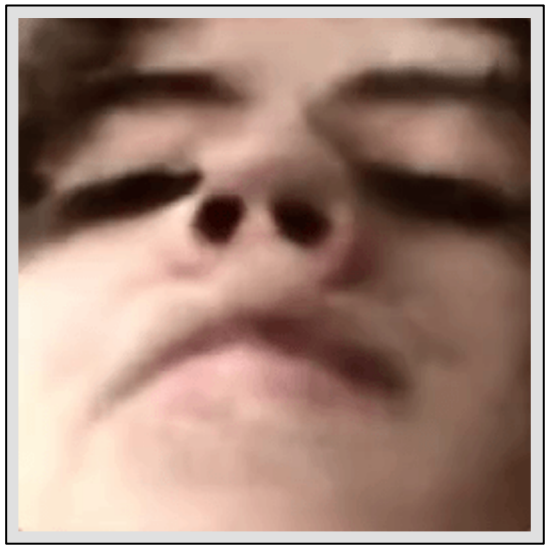
GAZE DIRECTION (UP-DOWN)



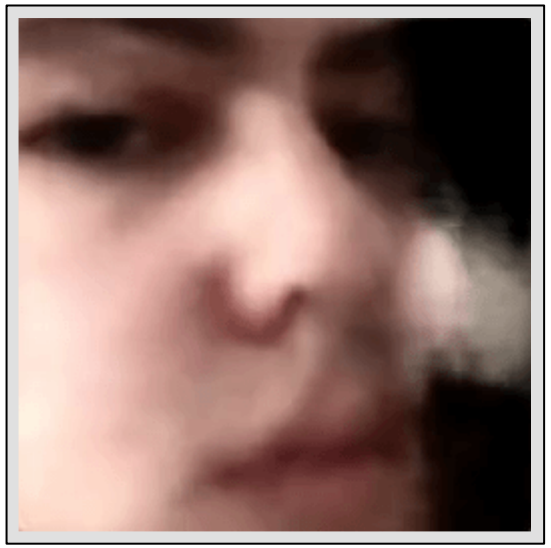
GAZE DIRECTION (RIGHT-LEFT)



HEAD ROTATION (UP-DOWN)



HEAD ROTATION (RIGHT-LEFT)



LATENT SPACE WALK IN [-45, 45 DEG]

EVALUATED ON THE MPIIGAZE SET

MODEL DETAILS

Decoding Transforming Encoder-Decoder (DT-ED) Architecture

x_a

DenseNET Encoder
(conv)

$\mathbf{R}^g_{ba}$

$\mathbf{R}^h_{ba}$

DenseNET Decoder
(deconv)

$\hat{x}_b$

$\mathcal{L}_1$

x_b

$\mathcal{L}_{gaze} + \mathcal{L}_{EC}$

16x1 Appearance Code

16x3 Gaze Direction Code

16x3 Head Rotation Code

1.8M training images!

TRAINING OPTIMIZATIONS

Hardware: V100 16/32 GB, Docker: nvidia/pytorch:20.03-py3

Baseline: V100 x 8 (Data Parallel), batch size 2048, image size 64 x 256 (input and output)

- + Mixed Precision Training -- NVIDIA AMP (O1)
- + Replace with Distributed Data Parallel (DDP)
- + FusedSGD
- + cuDNN Algorithm Autotuning

TRAINING OPTIMIZATIONS

Mixed Precision Training - NVIDIA AMP (O1)

- Install NVIDIA APEX: https://github.com/NVIDIA/apex#quick-start
- Import amp (also native support from PyTorch 1.6)

```
from apex import amp
```

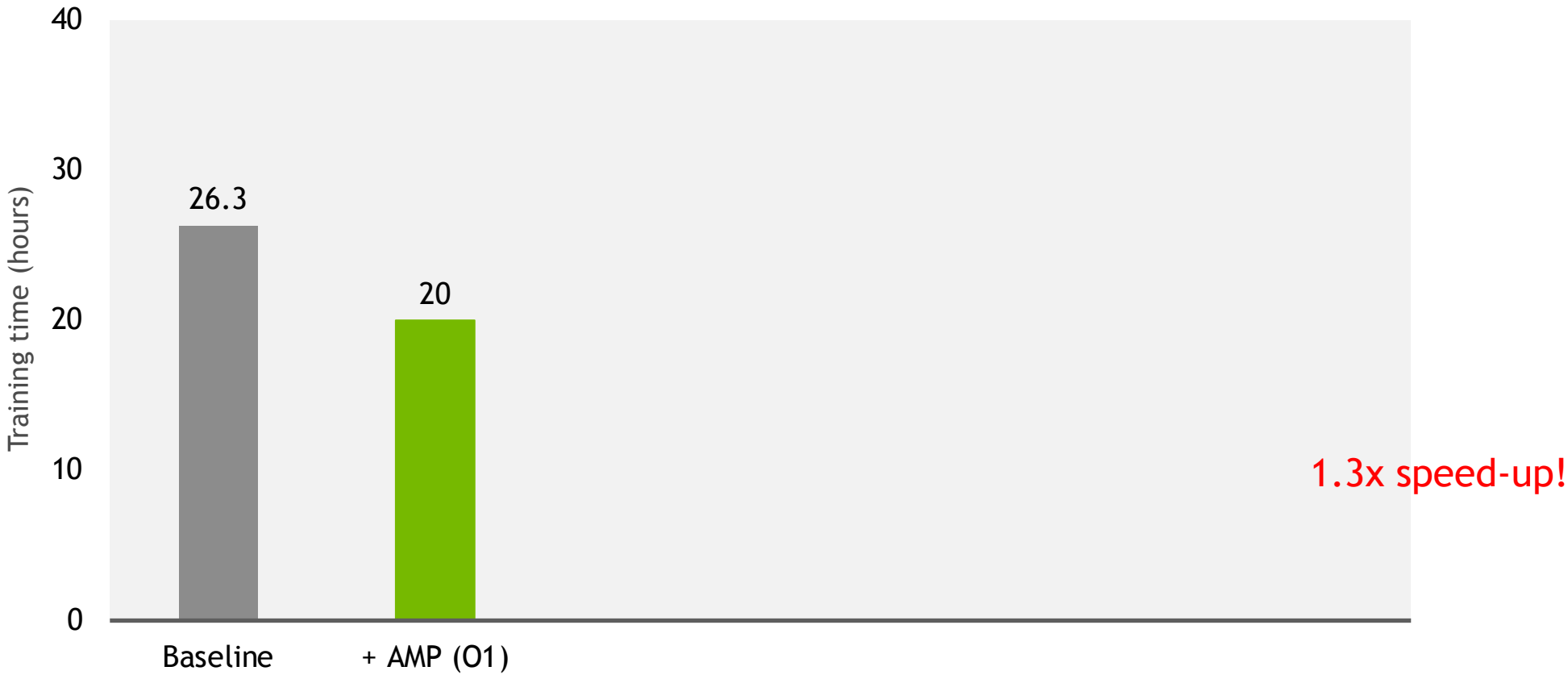
- Wrap network and optimizer

```
network, optimizers = amp.initialize(network, optimizers,
                                     opt_level='O1', num_losses=len(optimizers))
```

- Scale loss

```
with amp.scale_loss(loss_to_optimize, optimizer) as scaled_loss:
    scaled_loss.backward()
```

TRAINING TIME



TRAINING OPTIMIZATIONS

Distributed Data Parallel (DDP)

Separate CPU processes for each GPU
(https://pytorch.org/docs/master/generated/torch.nn.parallel.DistributedDataParallel.html)

- Import

```
from torch.utils.data.distributed import DistributedSampler
from torch.nn.parallel import DistributedDataParallel as DDP
```

- Instantiate multiple CPU processes
  - For single node training

```
parser.add_argument('--local_rank', default = 0, type = int)

world_size = torch.distributed.get_world_size()

torch.cuda.set_device(args.local_rank)
torch.distributed.init_process_group(backend = 'nccl', init_method = 'env://')
```

  - For multi-node training

```
global rank = node_rank x torch.cuda.device_count() + local_rank
```

TRAINING OPTIMIZATIONS

Distributed Data Parallel (DDP)

- Distribute network and data

```
network = DDP(network, device_ids=[args.local_rank])

train_sampler = DistributedSampler(train_dataset, num_replicas=world_size,
                                   rank=args.local_rank, shuffle=True)
train_dataloader = DataLoader(train_dataset, batch_size=args.batch_size,
                              drop_last=True, num_workers=args.num_data_loaders,
                              pin_memory=True, sampler=train_sampler)
```

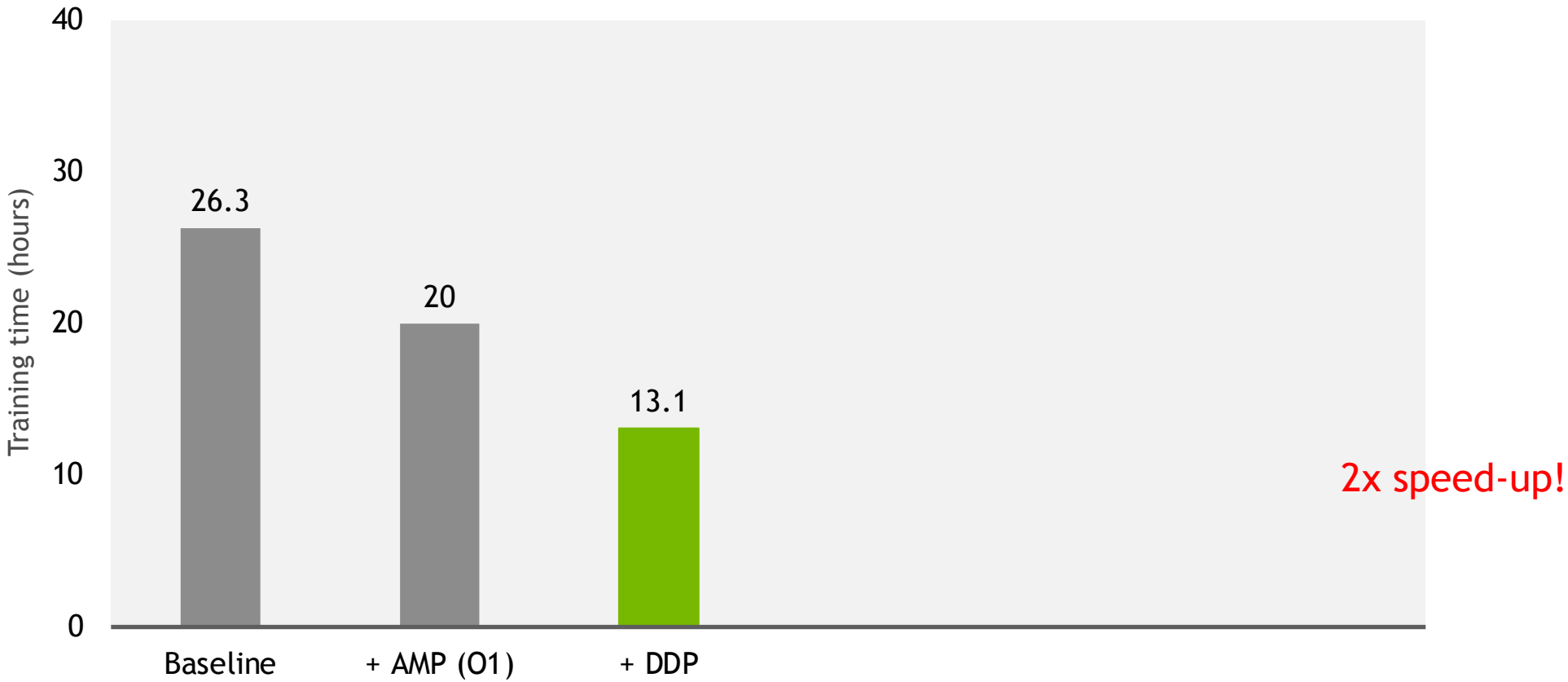
- Combine distributed loss if needed (for logging only)

```
def reduce_loss(loss):
    loss_clone = loss.clone()
    torch.distributed.all_reduce(loss_clone, op = torch.distributed.ReduceOp.SUM)
    avg_loss = loss_clone / world_size
    return avg_loss
```

- Launch python script

```
python -m torch.distributed.launch --nproc_per_node=8
```

TRAINING TIME



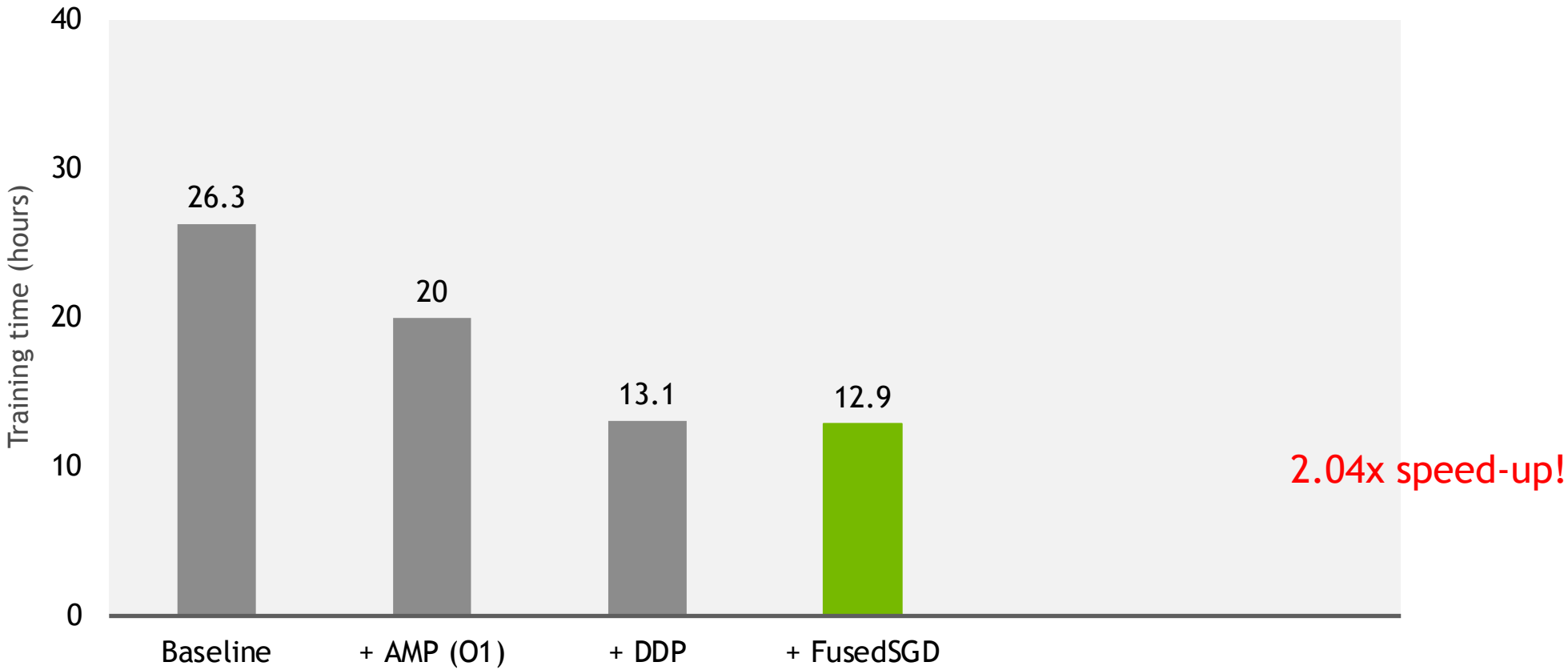
TRAINING OPTIMIZATIONS

FusedSGD

Fusion of SGD update's elementwise operations
(https://nvidia.github.io/apex/optimizers.html?highlight=fused#apex.optimizers.FusedSGD)

```
from apex.optimizers import FusedSGD
optimizer = FusedSGD(
    [
        {'params': [p for n, p in network.named_parameters() if not n.startswith('gaze')]},
        {
            'params': [p for n, p in network.named_parameters() if n.startswith('gaze')],
            'lr': gaze_lr,
        },
    ],
    lr=args.base_lr, momentum=0.9,
    nesterov=True, weight_decay=args.l2_reg)
```

TRAINING TIME



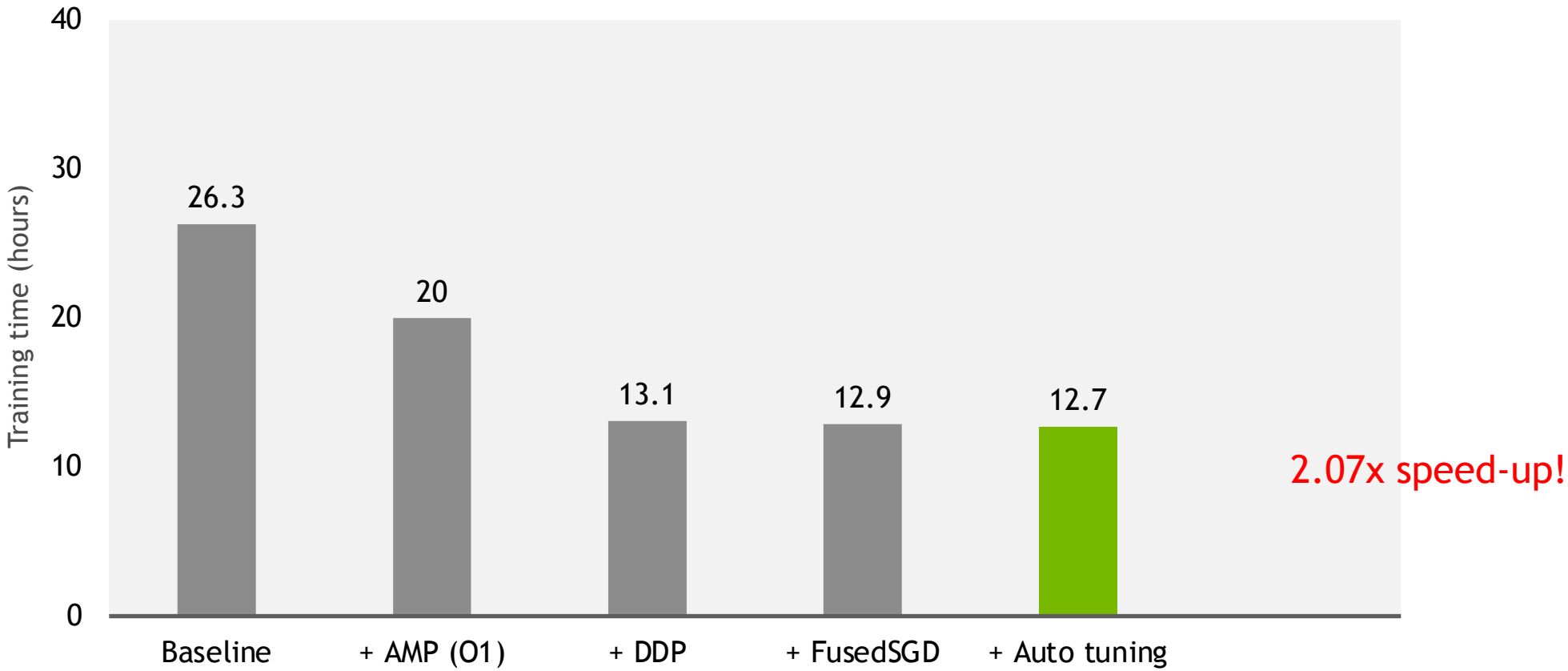
TRAINING OPTIMIZATIONS

cuDNN Convolutional Algorithm Autotuning

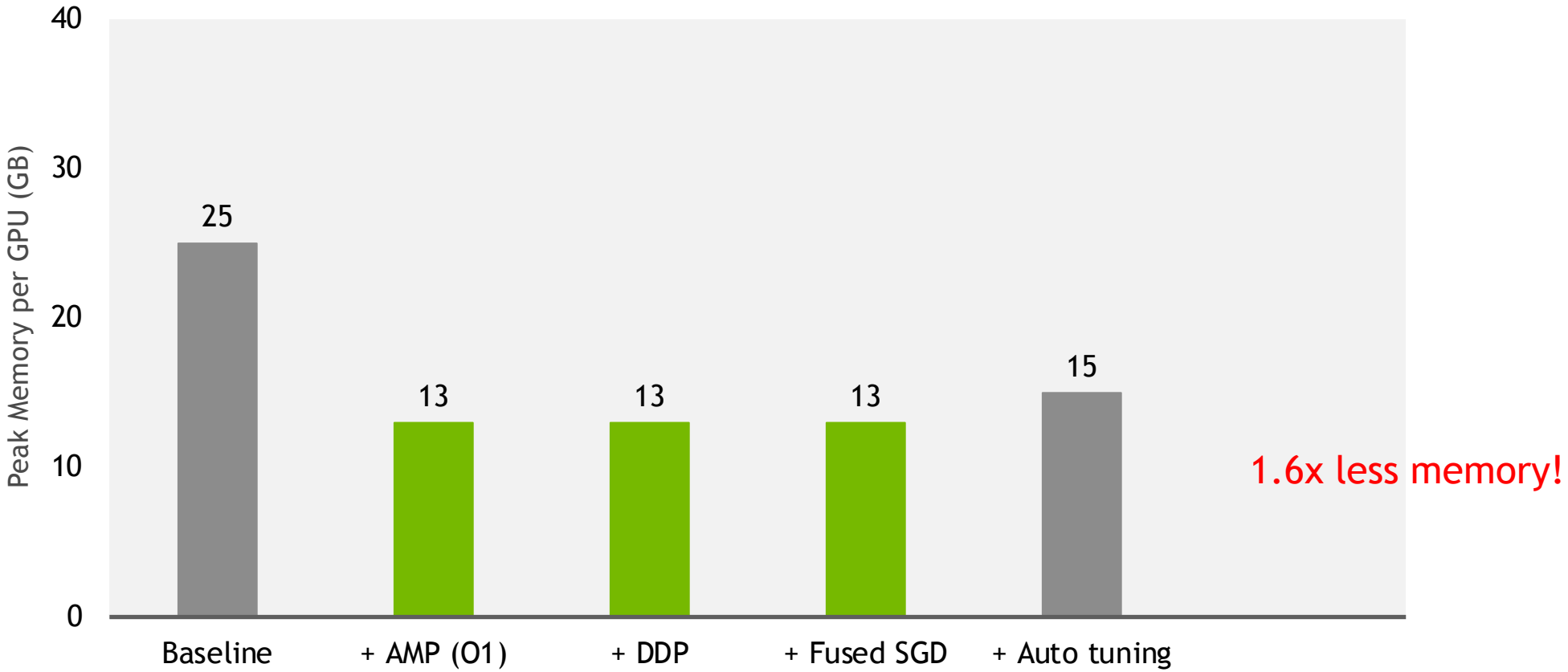
Enable cuDNN backend auto-tuner to select the best implementation of convolutional algorithm for your hardware.

```
torch.backends.cudnn.benchmark = True
```

TRAINING TIME

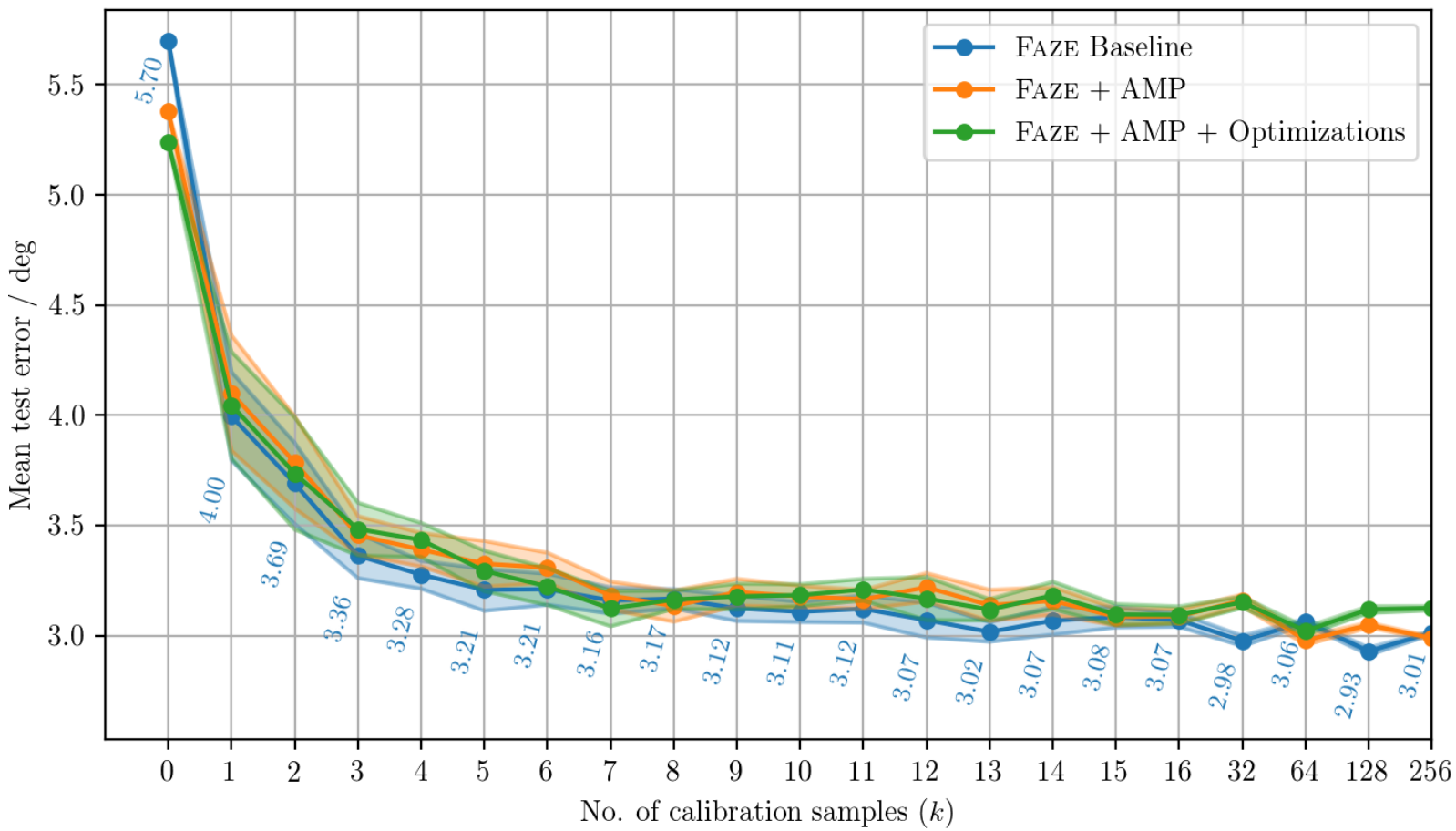


MEMORY REQUIREMENT



EFFECT ON ACCURACY

MPIIGaze Dataset



Accuracy is unchanged!

TRAINING OPTIMIZATIONS

Summary

Baseline: V100 x 8 (Data Parallel), batch size 2048, image size 64 x 256 (input and output)

- + AMP (O1)
- + Distributed Data Parallel
- + FusedSGD
- + cuDNN Algorithm Autotuning

2.07x speed-up, 1.6x reduction in memory, no change in accuracy


NVIDIA