



# EURO<sup>4SEE</sup>

Optimizing Deep Learning Systems for Hardware  
Assoc. Prof. Erdem AKAGÜNDÜZ, METU

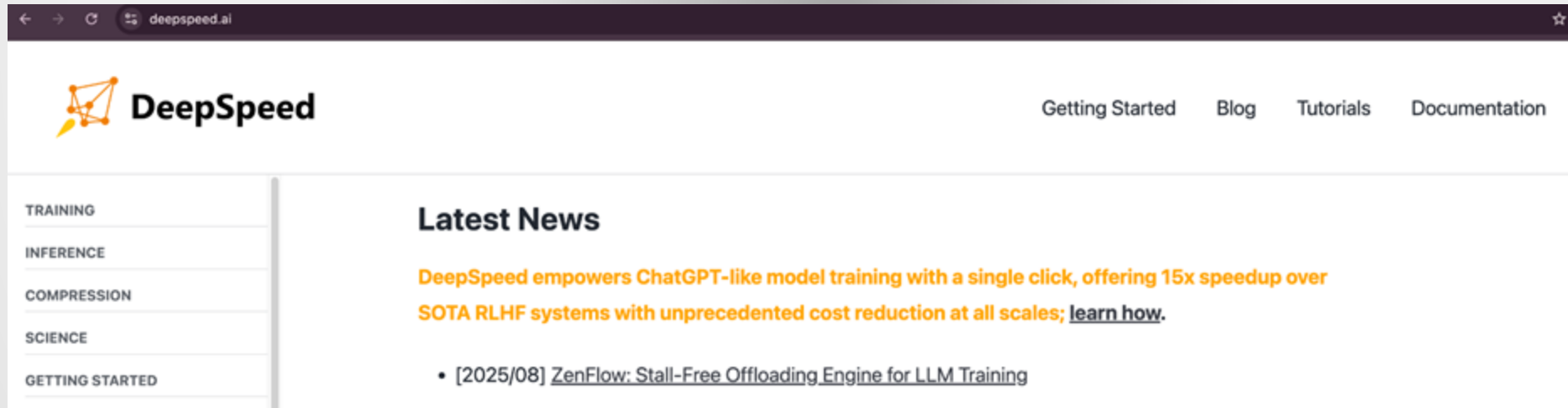
# Part V : Scaling DL



- PV.a : Introduction to DeepSpeed
- PV.b : Conclusions and Directions

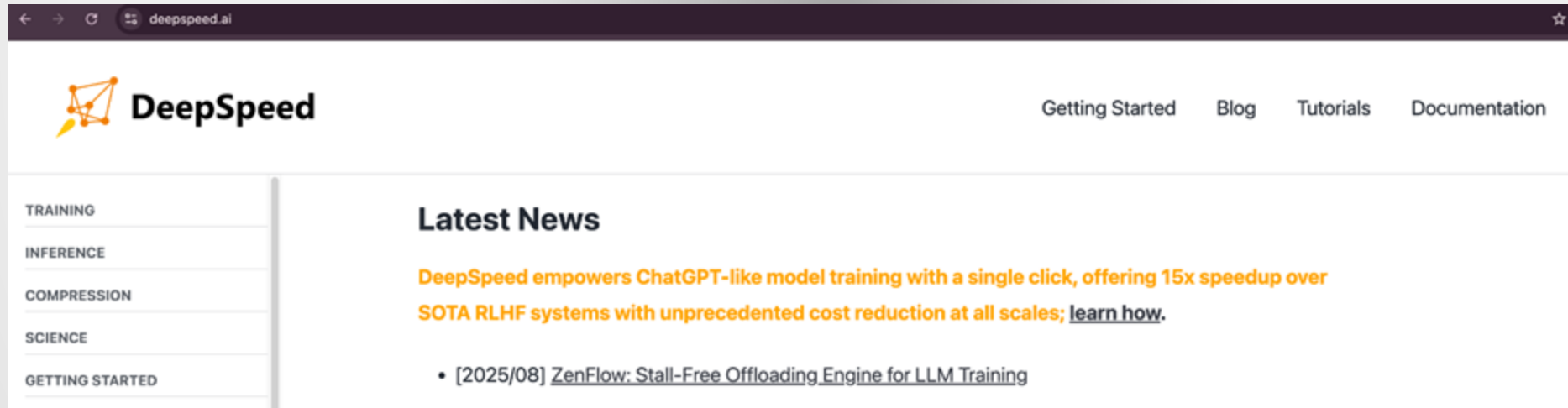
# Part V.a: DeepSpeed?

- It is an easy-to-use deep learning optimization software suite that powers scale and speed for both training and inference.
- Scale?



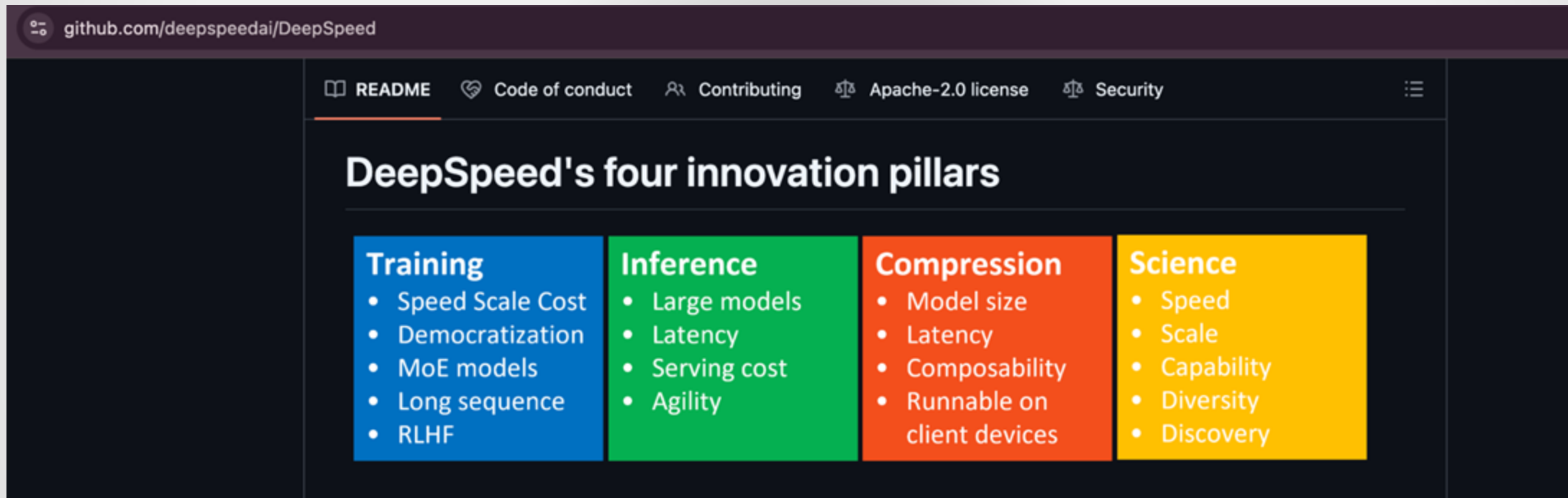
# Part V.a: DeepSpeed?

- It is an easy-to-use deep learning optimization software suite that powers scale and speed for both training and inference.
- Scale: the ability to efficiently train and deploy very large deep learning models—often with billions or even trillions of parameters—across large numbers of GPUs (or other accelerators) and compute nodes.



# Part V.a: DeepSpeed?

- It is open-source and available as a GitHub repo.



The screenshot shows the GitHub repository for DeepSpeed. The page title is "DeepSpeed's four innovation pillars". Below the title, there are four colored boxes representing the pillars: Training (blue), Inference (green), Compression (orange), and Science (yellow). Each box contains a list of bullet points.

Training	Inference	Compression	Science
<ul style="list-style-type: none"><li>• Speed</li><li>• Scale</li><li>• Cost</li><li>• Democratization</li><li>• MoE models</li><li>• Long sequence</li><li>• RLHF</li></ul>	<ul style="list-style-type: none"><li>• Large models</li><li>• Latency</li><li>• Serving cost</li><li>• Agility</li></ul>	<ul style="list-style-type: none"><li>• Model size</li><li>• Latency</li><li>• Composability</li><li>• Runnable on client devices</li></ul>	<ul style="list-style-type: none"><li>• Speed</li><li>• Scale</li><li>• Capability</li><li>• Diversity</li><li>• Discovery</li></ul>

# DeepSpeed and HPC?

- DeepSpeed is designed to exploit the infrastructure and software stack of HPC systems to make large-scale model training feasible:
  - HPC Hardware Parallelism: data parallelism, tensor/model parallelism, and pipeline parallelism ...
  - Resource Utilization & Throughput: DeepSpeed ensures that the HPC resources are fully utilized by optimizing communication, memory footprint, and computation.
  - HPC Software Ecosystem: DeepSpeed integrates with HPC job schedulers (Slurm, PBS) and distributed training frameworks (e.g., MPI, NCCL).

# Others?

- Yes. There are alternatives in the HPC landscape
  - Megatron-LM (NVIDIA): Strong for tensor-model parallelism on GPU clusters; often used together with DeepSpeed. NVIDIA pushes it for their DGX/SuperPod systems.
  - Horovod (Uber, now open-source): Early leader for distributed deep learning (data parallelism via MPI/NCCL). Still used, but less efficient for trillion-parameter scale compared to DeepSpeed.
  - PyTorch Distributed: Continues to evolve, and now includes many optimizations once only in external frameworks.
  - Colossal-AI: One of the newer frameworks targeting HPC/LLM training, offering pipeline + tensor parallelism with automation.

# DeepSpeed: Getting started

- What we do:
  - moving from local training → HPC-scale training with DeepSpeed
- Where to start:
  - Understand the DeepSpeed config file: `ds_config.json`
- This file defines:
  - Training precision (FP16, BF16, ZeRO stages, gradient accumulation, etc.)
  - Optimizer & scheduler settings
  - Parallelism / memory options (ZeRO stage, activation checkpointing, offloading)
  - Batch size settings



# DeepSpeed: ds\_config.json

- a minimal ds\_config.json file would look like:

```
{  
  "train_batch_size": 64,  
  "gradient_accumulation_steps": 4,  
  "fp16": {"enabled": true},  
  "zero_optimization": {"stage": 1}  
}
```

- A non-minimal would have many features (of course!)

# DeepSpeed: ds\_config.json

- `train_batch_size": 64,`
- This is the global batch size seen by the model in each step.
- In DeepSpeed this is after accounting for gradient accumulation and data parallel replicas.
  - Example: if you run on 4 GPUs, each GPU might only hold 4 samples, but with accumulation and replication, the effective batch size will be 64. (4 x 4 x 4)
- Accumulation? Replication?

# DeepSpeed: ds\_config.json

- `"gradient_accumulation_steps": 4,`
- Instead of updating weights every mini-batch, DeepSpeed can accumulate gradients over multiple forward/backward passes before an optimizer step.
- In this case, it will do 4 passes (each with a micro-batch) → then update once.
- This is how you can simulate large batch sizes without blowing up GPU memory.

# DeepSpeed: ds\_config.json

- `"fp16": {"enabled": true},`
- Enables mixed precision training in FP16 (half precision).
- Remember the benefits:
  - Faster computation (tensor cores optimized for FP16/BF16).
  - Lower memory usage.
- DeepSpeed automatically handles loss scaling to avoid underflow issues.
- On newer GPUs, you might also consider bf16 mode
  - (which have the HW for it)

# DeepSpeed: ds\_config.json

- `"zero_optimization": {"stage": 1}`
- Activates ZeRO (Zero Redundancy Optimizer): DeepSpeed's memory optimization.
- Stage: 1 means:
  - Optimizer states (e.g., Adam's momentum, variance) are partitioned across GPUs instead of fully replicated.
  - Saves memory, allowing larger models.
- Higher stages:
  - Stage 2: partitions gradients as well.
  - Stage 3: partitions model parameters too (biggest memory savings, but more communication overhead).

# DeepSpeed: ZeRO

## ZeRO (Zero Redundancy Optimizer)

- We aim to train models that are larger than a single GPU's memory by partitioning memory states across devices. OK!
- Memory consumers in training are:
  - Model parameters (weights)
  - Gradients
  - Optimizer states (e.g., Adam: momentum + variance)

# DeepSpeed: ZeRO

## ZeRO (Zero Redundancy Optimizer)

- ZeRO stages are about how “deep” we do this:
  - Stage 1: Partition optimizer states across GPUs
  - Stage 2: Partition optimizer states + gradients
  - Stage 3: Partition optimizer states + gradients + parameters
- Higher stages →
  - more communication overhead,
  - but much bigger memory efficiency.

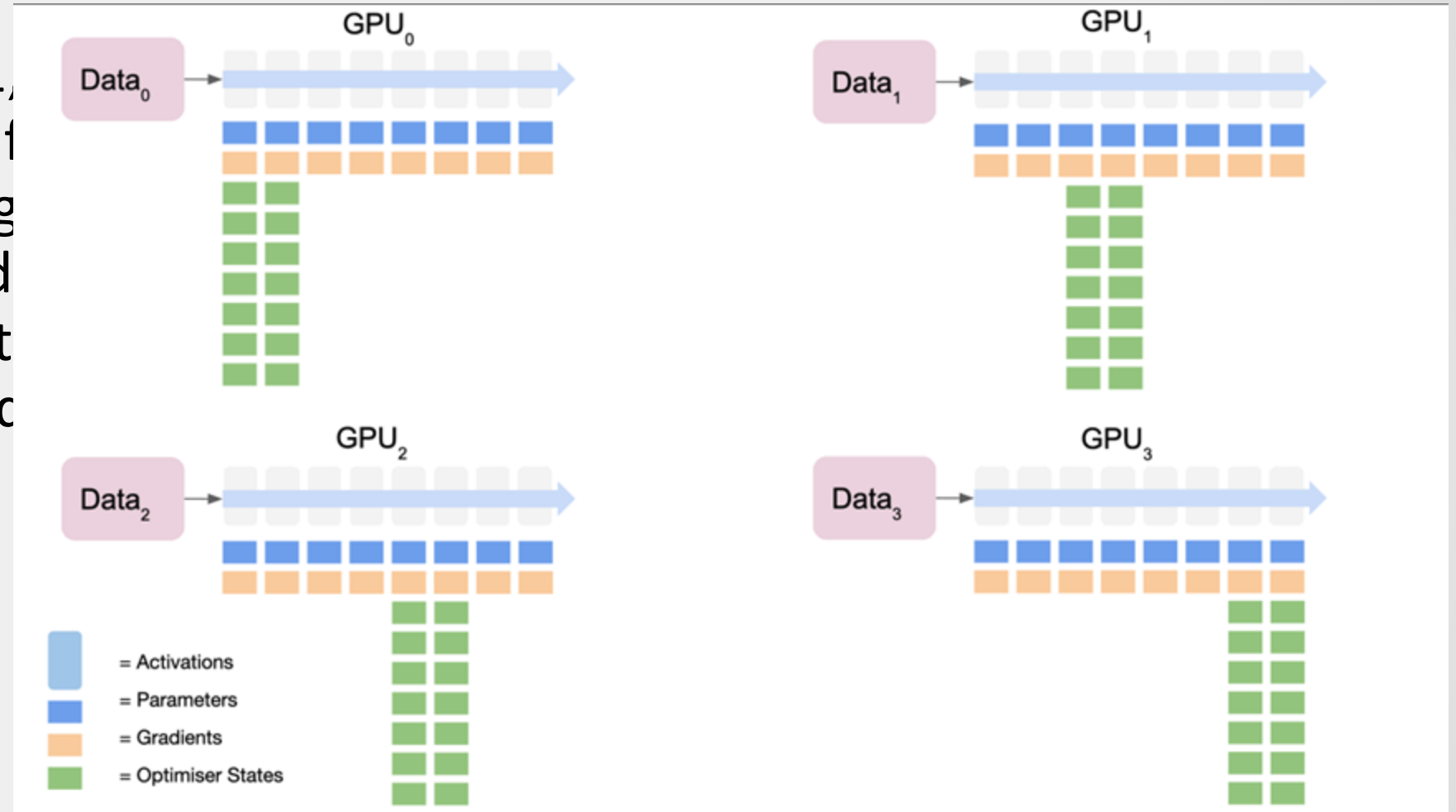
# DeepSpeed: ZeRO

- For ZeRO stage 1, in the backward pass, each device calculates the (first order) gradients for the final section of the model.
- The final device gathers all these gradients, averages them and then computes the Adam optimised gradient with the optimisation states.
- It then broadcasts back the new parameter states for the final section of the model to all devices.



# DeepSpeed: ZeRO

- For ZeRO stage 1, (order) gradients
- The final device g computes the Ad
- It then broadcast the model to all c



# DeepSpeed: how to launch?

- `deepspeed train.py --deepspeed --deepspeed_config ds_config.json`
  - Training launch is very compact.
  - All complexity (precision, parallelism, memory optimizations, etc.) is handled by the configuration file
  - Properly tuned to your hardware setup (GPUs, memory, interconnects), this single command can scale from local machines → HPC clusters

## Next: Part V.b

- PV.a : Introduction to DeepSpeed
- PV.b : Conclusions and Directions

# Thanks!



Co-funded by  
the European Union



**EuroHPC**  
Joint Undertaking

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101191697. The JU receives support from the Digital Europe Programme and Germany, Türkiye, Republic of North Macedonia, Montenegro, Serbia, Bosnia and Herzegovina.