- The problem the paper is trying to tackle.
- What's the impact of the work, e.g., why is it an important problem to solve?
- The main proposed idea(s).
- A summary of your understanding of different components of the proposed technique, e.g., the purpose of critical design choices.
- Your perceived strengths and weaknesses of the work, e.g., novelty, significance of improvements, quality of the evaluation, easy-to-use.
- Is there room for improvement? If so, which directions you may want to explore or ideas you have for improving the techniques?

The main problem the paper is trying to tackle is the scalability of large model training in more constrained scenarios (i.e. to people without a large GPU farm). This not only increases accessibility to training large models, but also increases the possible max size of trainable models on existing clusters of HW. ZeRO-Offload also aims to be easy to use and requires no model refactoring, which are points that further impact its usability and accessibility.

The impact of these works is a novel deep learning library called Deepspeed. Deepspeed provides implementations of model optimization frameworks and strategy (including ZeRO), and Offload has been implemented in Deepspeed. This easy-to-use Python library allows for anybody to readily use ZeRO-Offload to train larger models on more restrictive hardware while also improving throughput over existing strategies.

The main optimization Offload targets is offloading certain computation to the CPU to take advantage of the compute capabilities while also taking advantage of memory offloading. The reason why ZeRO is chosen as the backbone of Offload is that, unlike other solutions, ZeRO allows computation to scale with multiple GPUs using MP and DP. This means that ZeRO can improve on existing CPU-offloading solutions by allowing multi-GPU scalability.

This involves the creation of a simplified DFG denoting granular computations in order to partition computation to GPU vs. CPU. The authors find that it is optimal to partition batch-size-related computation (i.e. forward and backward passes) to the GPUs and optimization updates to the CPU. Combined with modifications to the optimizer (Adam), such as CPU optimizations for Adam, as well as smart data-transfer scheduling (i.e. delayed parameter update), this allows Offload to utilize ZeRO to take advantage of multi-GPU compute and scalability while also utilizing the CPU in non-bottleneck computation.

Offload is, in my opinion, an extremely novel and important paper with a very good implementation. It is very usable, requiring little change to a regular PyTorch training codebase, and also very scalable, taking advantage of both CPU memory, CPU compute, and aggregate GPU memory. The results show that in many cases, ZeRO-Offload allows for the training of extremely large models in cases where other methods aren’t as efficient or feasible (due to lack of sufficient aggregate GPU memory).

However, for larger models, the paper implementation doesn't scale as well due to the CPU bottleneck and the lack of parameter partitioning. This has been fixed in subsequent releases with ZeRO-3 Offload (the parameter partitioning issue), but another downside is that for very small batch sizes and for larger GPU clusters, L2L and stock ZeRO outperform it, respectively. This is because L2L bypasses the CPU bottleneck, and ZeRO is able to operate on similar batch sizes once memory becomes less of a restriction.

Overall, ZeRO-Offload is a great step forward in utilizing the CPU for large model training, allowing for larger models to be efficiently trained on less hardware.