

# High-Throughput and Memory-Efficient Zeroth-Order Fine-tuning LLMs with Distributed Parallel Computing

Liangyu Wang<sup>1,2</sup>, Huanyi Xie<sup>1,2</sup>, Di Wang<sup>1,2</sup>

<sup>1</sup>King Abdullah University of Science and Technology

<sup>2</sup>Provable Responsible AI and Data Analytics Lab

Correspondence: [di.wang@kaust.edu.sa](mailto:di.wang@kaust.edu.sa)

## Abstract

Fine-tuning large language models (LLMs) remains resource-intensive due to their sheer scale. While zeroth-order (ZO) optimization provides a memory-efficient alternative by eliminating backward passes, its application to multi-hundred-billion-parameter models is constrained by GPU memory and compute throughput. The ZO2 framework addresses the memory bottleneck by offloading model parameters to CPU memory and overlapping transformer block transfer with dual forward computation on a single GPU. However, ZO2 remains limited by its single-device execution and achieves modest throughput. In this work, we present DistZO2 (Distributed Zeroth-Order Offloading), a high-throughput, memory-efficient framework for distributed zeroth-order fine-tuning of LLMs. DistZO2 introduces three parallel strategies: (1) Perturbation Parallelism (PertP), which parallelizes the two perturbed forward passes across devices; (2) Distributed Data Parallelism (DDP), adapted to the scalar-gradient nature of ZO training; and (3) a unified 2D Parallelism design that combines PertP and DDP. To further mitigate communication bottlenecks introduced by parameter offloading, we propose a hardware-aware communication strategy that slices parameter blocks and redistributes them across GPUs via high-speed interconnects such as NVLink. DistZO2 scales zeroth-order fine-tuning to modern multi-GPU systems, preserving ZO2’s memory efficiency while substantially improving training throughput. In our experiments on OPT-175B, DistZO2 achieves a  $3\times$  speedup over ZO2 with distributed computing.

## 1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities across a wide range of tasks, but fine-tuning them remains prohibitively expensive. For instance, models such as GPT-175B (Brown et al., 2020), OPT-175B (Zhang et al.,

2022), LLaMA 405B (Grattafiori et al., 2024), and DeepSeek 671B (Liu et al., 2024) require substantial GPU memory and computational resources, making full-parameter fine-tuning challenging in practical settings.

Recent work has shown that zeroth-order (ZO) optimization offers a promising alternative for fine-tuning LLMs without requiring backward passes (Malladi et al., 2023). However, while prior efforts have addressed the single-device memory bottleneck, the broader challenge remains unresolved: how to scale ZO fine-tuning efficiently on CPU and multi-GPU heterogeneous systems.

For example, the ZO2 framework (Wang et al., 2025a) demonstrates that CPU offloading can enable fine-tuning of models as large as OPT-175B on a single GPU. While inspiring, ZO2 fundamentally remains a single-device solution, unable to address the throughput and scalability requirements of distributed training.

To improve throughput, distributed computing can be leveraged to parallelize forward computation across multiple devices. Although distributed training is well-studied for first-order (FO) methods, existing attempts to extend ZO into multi-GPU settings fail to exploit the unique dual-forward structure and CPU offloading patterns of ZO, leading to poor scalability and communication bottlenecks. In particular, ZO methods perform two independent and symmetric forward passes with perturbed parameters in the same direction, enabling natural parallel execution across devices. This contrasts with first-order (FO) methods (Ruder, 2016; Loshchilov and Hutter, 2017; Wang et al., 2026, 2025b), where the forward and backward passes are dependent and must be executed sequentially. We propose a new strategy called *Perturbation Parallelism (PertP)*, which distributes these two forward passes across two devices. While conceptually straightforward, PertP introduces nontrivial scheduling challenges, especially in managing

asynchronous execution, random number synchronization, and overlapping computation with communication.

Another avenue for improving throughput is *Distributed Data Parallelism (DDP)* (Li et al., 2020), which scales training by replicating models across multiple GPUs and synchronizing gradients via `all_reduce`. However, DDP is traditionally tailored to first-order optimizers (FO), where gradient synchronization happens after a backward pass. In ZO-based fine-tuning, there are no backward passes, and the model produces only scalar projected gradients. Since DDP was originally designed for first-order optimizers, applying it to zeroth-order training requires some adaptation, such as synchronizing scalar projected gradients instead of full tensors and ensuring consistent perturbation vectors across devices.

Moreover, plain distributed communication can degrade performance if not carefully optimized. In particular, when applying distributed computing to ZO2, the CPU must transmit the same  $M$ -sized parameter block to each GPU, putting significant pressure on limited PCIe bandwidth. ZO2’s original single-GPU design overlaps dual forward computation with CPU–GPU data movement, but PertP separates these computations across devices, potentially breaking this overlap. The problem is further exacerbated by advances in compute optimizations—such as FlashAttention (Dao, 2023) and `torch.compile` (Ansel et al., 2024)—which can accelerate forward passes to the point where communication becomes the bottleneck.

To address these challenges, we propose DistZO2 (Distributed Zeroth-Order Offloading), the first systematic framework for extending zeroth-order fine-tuning to CPU and multi-GPU heterogeneous systems. Our contributions are summarized as follows:

- We identify a fundamental gap in existing zeroth-order fine-tuning methods: while prior works (e.g., MeZO, ZO2) demonstrate strong memory efficiency on a single device, they lack systematic designs for CPU and multi-GPU heterogeneous training. Simply applying plain DDP to ZO training fails to exploit its dual-forward structure and CPU offloading patterns, leading to poor scalability.
- We introduce three distributed strategies tailored to ZO training with communication optimization: (1) Perturbation Parallelism (PertP)

– parallelizing the dual forward passes across GPUs, (2) ZO-adapted Distributed Data Parallelism (DDP) – synchronizing scalar projected gradients and consistent perturbations across GPUs, and (3) Unified 2D Parallelism – combining PertP and DDP to scale efficiently across many GPUs.

- We design a hardware-aware communication strategy that slices parameter blocks and redistributes them via NVLink, eliminating PCIe bottlenecks.
- Through extensive experiments, we show that PertP and DDP respectively address ZO’s compute and data bottlenecks, while our 2D parallelism and communication optimization further amplify throughput—enabling  $3\times$  speedup over ZO2 and scalable fine-tuning up to OPT-175B.

## 2 Related Work

Zeroth-order (ZO) optimization offers a memory-efficient alternative to first-order training by eliminating backward passes (Malladi et al., 2023; Wang et al., 2025a). While methods such as MeZO and ZO2 enable fine-tuning of multi-billion-parameter LLMs with reduced memory usage, they remain constrained to single-device execution, limiting throughput and scalability. Existing distributed training frameworks such as DDP (Li et al., 2020) and ZeRO (Rajbhandari et al., 2020) are tailored to first-order gradients and provide limited benefit when naively applied to ZO, since they fail to leverage its dual-forward structure and CPU offloading pattern.

DistZO2 addresses this gap as the first systematic framework for distributed ZO fine-tuning on CPU+multi-GPU heterogeneous systems. We introduce Perturbation Parallelism, ZO-adapted DDP, and a unified 2D parallelism strategy, together with a hardware-aware communication scheme, enabling scalable and high-throughput zeroth-order training. A more detailed discussion of related work is provided in Appendix B.

## 3 Preliminaries

Zeroth-order optimization (ZO) is a gradient-free optimization technique that estimates the directional derivative of the loss function using only forward computations. Given a loss function  $\mathcal{L}(\theta)$  and a random direction vector  $z \sim \mathcal{N}(0, I)$ , the

projected gradient, which is a scalar, is approximated as:

$$g = \frac{\mathcal{L}(\theta + \epsilon z) - \mathcal{L}(\theta - \epsilon z)}{2\epsilon}, \quad (1)$$

where  $\epsilon$  is a small positive scalar controlling the perturbation magnitude. This estimator is known as the central difference estimator. Then, the model parameters can be updated by:

$$\theta = \theta - \eta g z, \quad (2)$$

where  $\eta$  is the learning rate. Since ZO does not require backpropagation or storage of intermediate activations, it significantly reduces the memory footprint compared to first-order methods.

Recent systems such as MeZO (Malladi et al., 2023) and ZO2 (Wang et al., 2025a) demonstrate the feasibility of ZO fine-tuning for multi-billion-parameter models, primarily on single GPUs. MeZO achieves forward-only training with reduced memory footprint, while ZO2 introduces CPU offloading and stream-based scheduling to scale to models as large as OPT-175B. However, both remain constrained by single-device throughput, motivating our exploration of distributed zeroth-order training in this work.

## 4 Perturbation Parallelism (PertP)

### 4.1 Motivation

Zeroth-order optimization requires two forward passes with positively and negatively perturbed parameters. On a single GPU, these passes must be executed sequentially due to compute and memory constraints, which significantly limits throughput. In a heterogeneous CPU and multi-GPU setting, however, the two forward passes are naturally independent: each GPU can fetch parameters from CPU memory, execute its assigned forward computation, and asynchronously offload results. This observation motivates Perturbation Parallelism (PertP), the first distributed strategy that leverages both the dual-forward structure of ZO and CPU offloading in multi-GPU environments.

### 4.2 Workflow

PertP is a distributed training strategy designed for zeroth-order optimization in CPU and multi-GPU heterogeneous systems. It exploits the structural independence of the dual forward passes—one with positively perturbed parameters and the other with negatively perturbed ones—by assigning them to

different GPUs. Unlike single-device execution, where both passes must run sequentially due to compute and memory limits, PertP enables concurrent execution across devices while maintaining ZO’s block-wise CPU offloading mechanism.

Let  $W \in \mathbb{R}^{p \times d}$  denote the parameter matrix of a transformer block. In our design, the master copy of each transformer block is stored on the CPU, denoted as  $W_H$ , and is transferred to GPU memory only when needed. Let  $W_{D_k}$  be the device-side copy on GPU  $k \in 1, 2$ . Each GPU performs its assigned perturbed forward pass using  $W_{D_k}$ , while transfers are managed in a block-wise manner to preserve the low-memory footprint.

PertP integrates CPU offloading with distributed dual-forward execution. Each GPU independently uploads transformer blocks from CPU memory, performs its forward pass, and asynchronously offloads updated blocks. A key design principle is overlapping host–device transfers with computation: while one block is computing, the next block is uploading and the previous one offloading, ensuring high utilization in a heterogeneous system. Once both losses are computed, a lightweight communication step merges the results to form the final projected gradient.

Figure 1 illustrates the system architecture and execution flow. Each training iteration in PertP consists of the following steps:

**Step 1: Input Preparation and Shared RNG Initialization.** At the start of each iteration, a mini-batch is loaded into CPU memory and broadcast to both GPUs. A shared random seed  $s$  is generated and broadcast to ensure both devices sample the same Gaussian perturbation vector  $z \in \mathbb{R}^{p \times d}$ .

**Step 2: Distributed Dual Forward Execution.** Each GPU independently performs a forward pass on its local copy of the parameters. Specifically: (1) GPU 1 computes the forward pass with positively perturbed parameters:  $W_{D_1}^+ = W_H + \epsilon z$ . (2) GPU 2 computes the forward pass with negatively perturbed parameters:  $W_{D_2}^- = W_H - \epsilon z$ . Each GPU computes its own scalar loss, denoted as  $L_1^+$  and  $L_2^-$ , respectively.

**Step 3: Loss Synchronization and Projected Gradient Computation.** Once both losses are available, they are exchanged between devices using `all_gather` or `send/recv`. Then, each device computes the projected gradient using:  $g = \frac{L_1^+ - L_2^-}{2\epsilon}$ . This scalar gradient is identical across devices and can be reused for all updated parameters.

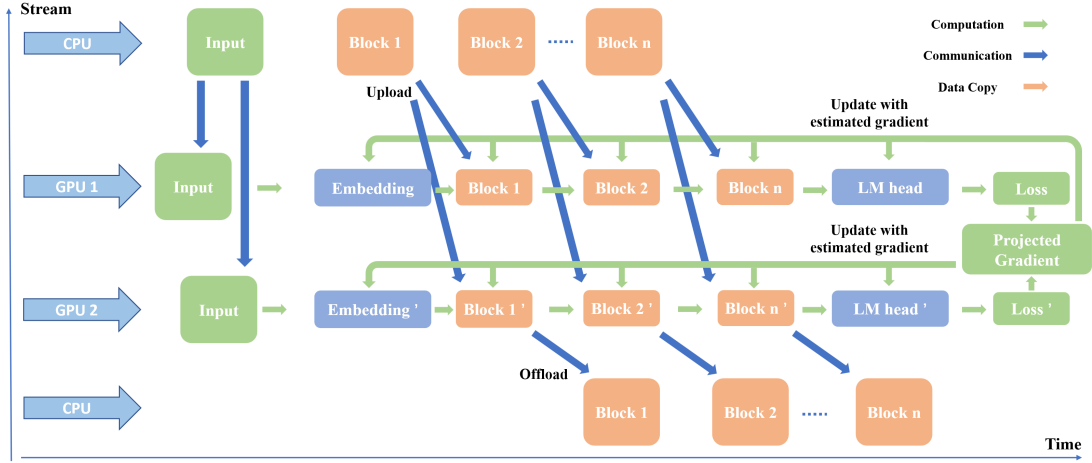


Figure 1: Perturbation Parallelism Workflow.

**Step 4: In-Place Parameter Update.** Each GPU applies the projected gradient  $g$  to its local parameters using the shared direction vector  $z$ :  $W_{D_k} \leftarrow W_{D_k} - \eta \cdot g \cdot z$ . Since both  $g$  and  $z$  are synchronized, the updates are consistent across devices without further communication.

**Step 5: Overlapping Computation and Communication.** To maximize throughput, each GPU maintains three asynchronous CUDA streams dedicated to uploading (H2D), forward computation, and offloading (D2H). These streams form a steady-state pipeline: while block  $i$  is computing, block  $i+1$  is uploading and block  $i-1$  is offloading. This overlap hides most PCIe transfer latency behind compute and makes PertP practical for CPU+multi-GPU heterogeneous training.

## 5 Applying Distributed Data Parallelism (DDP) to ZO2

### 5.1 Motivation and Workflow

Our design enables large-scale zeroth-order (ZO) fine-tuning in a distributed setting by offloading model parameters to CPU memory and avoiding backward passes, while scaling beyond the per-device compute limits of a single GPU. To further improve throughput, we adapt *Distributed Data Parallelism (DDP)* specifically to the ZO setting.

Applying DDP in ZO training differs from its use in first-order (FO) optimization. In FO methods, each GPU computes high-dimensional gradients via backpropagation, and `all_reduce` is used to synchronize full gradient tensors. In contrast, ZO-based training produces only a *scalar* projected gradient per iteration. This makes gradient synchronization highly efficient in the ZO setting and

allows DDP to scale with minimal communication overhead.

In each iteration, every GPU processes a different mini-batch and performs the dual forward passes required by ZO to estimate a local projected gradient  $g_k \in \mathbb{R}^1$ . To ensure model consistency across devices, we use an `all_reduce` to aggregate scalar gradients:  $g_{avg} = \frac{1}{K} \sum_{k=1}^K g_k$ .

In addition, all GPUs must use the same perturbation vector  $z$  for gradient estimation. We broadcast a shared random seed from the primary rank (e.g., GPU 0) to all other devices before sampling  $z$ , ensuring consistent perturbation across devices.

**Workflow Summary.** The complete ZO-adapted DDP workflow is as follows: (1) The input mini-batch is partitioned across  $K$  GPUs. (2) Each GPU uploads relevant transformer blocks from CPU to GPU memory, just as in PertP, then performs dual forward passes under  $\theta \pm \epsilon z$ . (3) Each GPU computes a local scalar projected gradient  $g_k$ . (4) Gradients  $\{g_k\}$  are averaged via `all_reduce` to obtain global  $g$ . (5) Each GPU applies the same gradient  $g$  and direction vector  $z$  to update parameters, and offloads updated blocks back to CPU memory, overlapping transfers with computation as in PertP.

## 6 Further Optimization

### 6.1 Communication Optimization Strategy

One of the key strengths of our offloading-based ZO design lies in its ability to overlap compute-intensive dual forward passes with communication operations such as block uploads and offloads. However, in the PertP setting, the dual forward pass is decomposed and distributed across GPUs, making it more difficult to maintain this natural

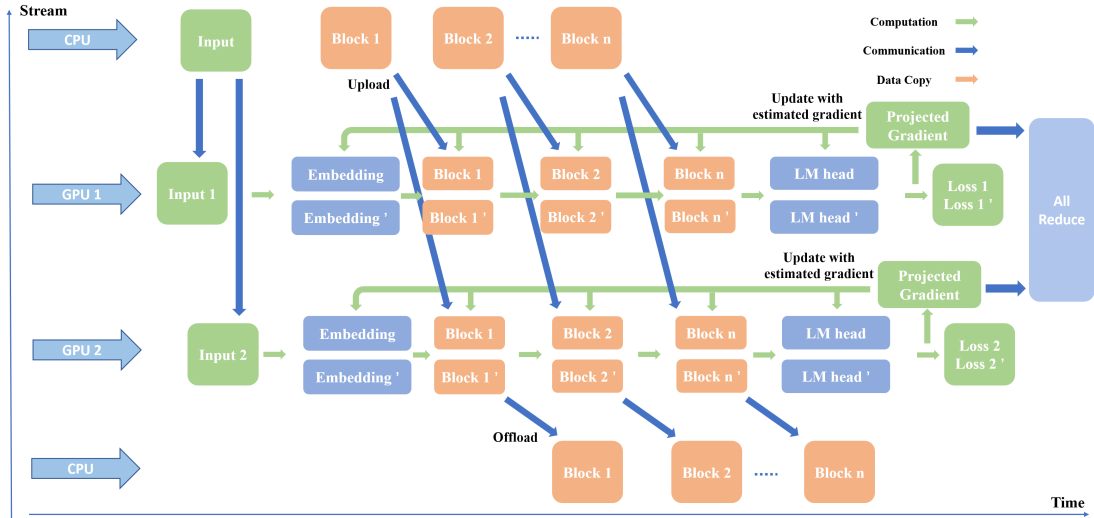


Figure 2: Applying Distributed Data Parallelism to ZO. Each GPU performs dual forward passes on its own batch, computes a scalar projected gradient, and synchronizes it via `all_reduce`.

overlap between computation and communication. Furthermore, in DDP configurations, a single CPU thread is often responsible for transferring parameter blocks to multiple GPUs simultaneously over PCIe. This can lead to transfer contention and bandwidth saturation, further increasing the overall communication latency. To address this challenge, we propose a communication optimization strategy that leverages the hardware topology of modern GPU clusters to redistribute and parallelize data transfer more efficiently.

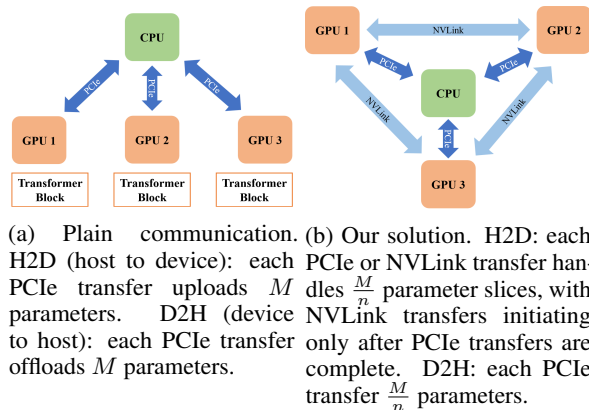


Figure 3: Communication Optimization.

**Hardware-Aware Communication Design.** As illustrated in Figure 3, the CPU is typically connected to GPUs via relatively slow PCIe links, while GPUs within the same node are connected via high-speed NVLink. In a plain offloading setup (Figure 3a), each GPU independently receives a full-size transformer block of  $M$  parameters via PCIe, resulting in redundant data transfers and con-

gested PCIe bandwidth.

**Optimized Upload Strategy.** Our strategy (Figure 3b) avoids this inefficiency by adopting the fine-grained tensor slicing and peer-to-peer communication strategies inspired by prior work on distributed tensor communication (Shoeybi et al., 2019; Rajbhandari et al., 2020; Jia et al., 2019; Zheng et al., 2022): the CPU slices each transformer block into  $n$  equal parts, each of size  $M/n$ , and sends one slice to each of the  $n$  GPUs via PCIe. After receiving its assigned slice, each GPU shares the remaining  $(n - 1)$  slices with its peers through NVLink-based peer-to-peer communication. This design reduces PCIe load per GPU by a factor of  $n$ , and utilizes the faster NVLink interconnect for intra-node communication. Since PCIe and NVLink communication streams can be executed concurrently, the total upload time per GPU becomes:  $T_{\text{comm}} = \frac{M}{n} (\text{PCIe}) + \frac{(n-1)M}{n} (\text{NVLink})$ .

**Optimized Offload Strategy.** After projected gradient computation, updated parameters must be transferred back to CPU memory. In naïve ZO2 offloading, each GPU would offload its entire copy of the updated parameter block, resulting again in redundant PCIe traffic. However, due to the deterministic and synchronized nature of zeroth-order updates, all GPUs hold identical updated parameters after the all-reduce operation. Thus, it is sufficient for each GPU to offload only its  $M/n$  slice. The CPU reassembles the full  $M$ -sized block by concatenating the  $n$  slices, with no loss of correctness. This strategy builds directly on the same block-wise offloading mechanism as PertP, ensur-

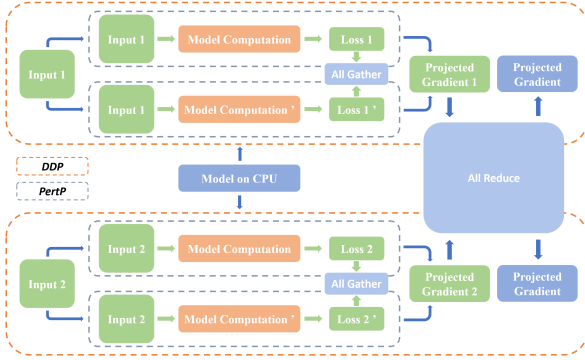


Figure 4: 2D Parallelism: inner PertP and outer DDP.

ing that computation–communication overlap is preserved while reducing redundant PCIe transfers.

**Thread-Aligned Memory Partitioning.** To avoid repeated slicing and merging of parameter blocks, we align model initialization with the communication pattern. During model setup, each CPU thread (assigned to control one GPU) is allocated a persistent memory region responsible for its  $M/n$  slice. This avoids the need to copy or re-partition parameters at runtime, and simplifies the interface between scheduling and communication.

## 6.2 2D Parallelism: Combining PertP and DDP

While Perturbation Parallelism (PertP) improves training throughput by distributing dual forward passes across GPUs, and Distributed Data Parallelism (DDP) enables model replication with efficient scalar synchronization, each strategy alone has limitations. PertP is constrained to two-way parallelism and does not scale beyond two GPUs without further extension, while DDP suffers from underutilization of per-GPU resources when each device must redundantly execute both perturbed computations.

To scale zeroth-order optimization efficiently across more number of GPUs, we propose to combine PertP and DDP into a unified **2D parallelism framework**, where each GPU performs only a single forward pass (with either  $+\epsilon z$  or  $-\epsilon z$ ) on a unique data shard. This fusion allows simultaneous parallelism along two orthogonal dimensions: *perturbation direction* and *data batches*.

We organize the total set of  $n$  GPUs as a 2D mesh of size  $n = n_b \times n_p$ , where  $n_b$  is the number of DDP batch-parallel groups and  $n_p = 2$  is the perturbation dimension. For each mini-batch, we perform the following steps:

(1) **Group Assignment.** Each GPU is assigned

a unique tuple  $(b_i, p_j)$ , where  $b_i$  denotes the data-parallel group and  $p_j \in \{+\epsilon, -\epsilon\}$  indicates the perturbation direction.

(2) **Forward Execution.** Each GPU computes a forward pass for its assigned perturbation on its local batch partition. No GPU computes both directions.

(3) **Intra-Group Communication.** Within each DDP group (fixed  $b_i$ ), the pair of GPUs corresponding to  $+\epsilon$  and  $-\epsilon$  exchange their scalar losses  $L^+$  and  $L^-$  to compute a shared projected gradient:  $g_i = \frac{L^+ - L^-}{2\epsilon}$ .

(4) **Global Gradient Synchronization.** The projected gradients  $\{g_i\}$  from all  $n_b$  groups are synchronized across all devices using `all_reduce`, yielding the final global gradient  $g$ .

(5) **Parameter Update.** Each GPU independently updates its parameters using  $g$  and the shared perturbation vector  $z$ , ensuring consistency across the entire model replica.

This design uses PertP as the *inner parallelism* within each DDP group and DDP as the *outer parallelism* across batches. This structure minimizes communication overhead, aligns with standard DDP usage, and scales efficiently. In contrast, reversing the order—using DDP as the inner parallelism—introduces unnecessary cross-group synchronization, complicates RNG coordination, and deviates from typical DDP frameworks, without offering performance gains. Thus, the PertP-inner structure is both simpler and more effective.

## 7 Experiments

### 7.1 Experiment Settings

We evaluate our framework using the Open Pre-trained Transformer (OPT) (Zhang et al., 2022) model family, including sizes ranging from 1.3B to 175B, to assess scalability across model scales. All experiments are conducted on the SST-2 (Socher et al., 2013) dataset using the MeZO (Malladi et al., 2023) and ZO2 (Wang et al., 2025a) methods as baselines. Our implementation builds upon ZO2 and extends it to multi-GPU settings.

Experiments are performed on a server with 4xNVIDIA H20 GPUs (96GB each) and an AMD Milan CPU, using PyTorch 2.3.0, CUDA 12.7, and Python 3.12. We use consistent hyperparameters across all runs: learning rate  $1e-7$ , batch size (BS) 4, sequence length 2048, and 100 training steps. GPU memory usage per GPU and training throughput (tokens/second) are reported for each configura-

Table 1: **Main results of DistZO2 performance for various model configurations and both FP32 and FP16 modes.** Instances of ‘-’ in the table indicate scenarios where the corresponding method failed to execute due to memory constraints.

Model	Memory Usage (MB) / GPU ↓					Throughput (tokens/sec) ↑				
	MeZO	ZO2	DistZO2(PertP)	DistZO2(DDP)	DistZO2(2D)	MeZO	ZO2	DistZO2(PertP)	DistZO2(DDP)	DistZO2(2D)
OPT-1.3B	5877	4979	5627	4451	6275	18720	18625	34225	31585	60091
OPT-2.7B	8617	5419	6021	4845	6669	9582	9582	17720	16106	30918
OPT-6.7B	16791	6941	7393	5823	8041	4133	4133	7653	7062	14417
OPT-13B	28999	8103	8455	7081	9201	2197	2190	4091	3730	7057
OPT-30B	62435	10723	10883	9901	11825	976	976	1820	1661	3221
OPT-66B	-	13263	13735	12163	14775	-	446	885	764	1370
OPT-175B	-	18713	18829	19881	18713	-	166	331	290	508

Table 2: **Throughput Analysis (on 4 GPUs): Communication Optimization (Billion Parameters per Second).**

Model	Upload	Upload+ CommOpt	Offload	Offload+ CommOpt
OPT-1.3B	4.73	19.24	1.79	8.32
OPT-2.7B	4.67	18.35	1.42	6.76
OPT-6.7B	4.33	12.06	2.00	7.92
OPT-13B	4.45	17.75	1.81	8.29
OPT-30B	3.69	17.71	1.98	8.80
OPT-66B	4.08	14.20	1.99	7.06
OPT-175B	3.82	17.66	2.12	8.77

tion. Our implementation leverages CUDA streams for overlapping upload, compute, and offload. Due to space constraints, we only present our main experimental results in the body of the paper. Further experimental results have been moved to the Appendix D.

## 7.2 Main Experiment

The goal of our main experiment is to evaluate the effectiveness of our distributed strategies under a fixed batch size setting. Specifically, we compare DistZO2 variants—including Perturbation Parallelism (PertP) and Distributed Data Parallelism (DDP) using 2 GPUs, and the unified 2D with PertP(inner)+DDP(outer) configuration using 4 GPUs—against single-GPU baselines (MeZO and ZO2).

Table 1 summarizes results across OPT model scales from 1.3B to 175B parameters. We make the following observations: **(1) Memory Efficiency.** All DistZO2 variants preserve the memory efficiency of ZO2. Notably, PertP and DDP maintain similar or lower per-GPU memory usage compared to ZO2, despite involving additional computation or communication. **(2) Throughput Gains from PertP and DDP.** Introducing Perturbation Parallelism (ZO2+PertP) achieves up

to  $2\times$  speedup across all models by parallelizing the dual forward passes. Distributed Data Parallelism (ZO2+DDP) provides consistent throughput improvements by scaling across mini-batches. ZO2+PertP consistently achieves slightly higher throughput than ZO2+DDP. **(3) Best Performance with Unified 2D Parallelism.** The combination of PertP and DDP (ZO2+PertP+DDP) delivers the highest throughput across all configurations. On OPT-175B, our unified parallelism achieves a  $3\times$  speedup over ZO2. **(4) Scaling to Extremely Large Models.** MeZO fails to run models beyond OPT-30B due to memory limitations, while DistZO2 successfully fine-tunes up to OPT-175B using only 18–20GB GPU memory per device.

## 7.3 Ablation Study

We conduct a series of ablation studies to isolate and quantify the contributions of each design component in DistZO2. Specifically, we analyze the effect of Perturbation Parallelism (PertP) (Section 4), Distributed Data Parallelism (DDP) (Section 5), our hardware-aware communication optimization (Section 6.1), and their combined form in 2D parallelism (Section 6.2). All experiments use a fixed BS (2, 4, 6, 8) and are conducted across multiple model scales to ensure generality. Although our distributed strategies are primarily designed to accelerate ZO2, they are orthogonal to the memory management style of the optimizer. Since MeZO and ZO2 share the same forward-pass structure and scalar-gradient computation, our methods can be directly applied to MeZO. This also allows for more controlled comparisons when isolating the effects of parallelism alone.

**Impact of Perturbation Parallelism (PertP) or Distributed Data Parallelism (DDP).** Table 3 Columns 3-6 show that applying PertP or DDP allows ZO2 to maintain throughput comparable to MeZO across all model scales. This indicates that

Table 3: **Throughput (Tokens per Second) Analysis Across Different Parallelism Methods.**

Model	BS	MeZO(PertP)	DistZO2(PertP)	MeZO(DDP)	DistZO2(DDP)	DistZO2(DDP+PertP)	DistZO2(PertP+DDP)
OPT-1.3B	2	28196	28196	24160	24160	32364	33815
OPT-2.7B		14512	14512	12426	12394	20837	20818
OPT-6.7B		6342	6342	5622	5391	6624	9227
OPT-13B		3369	3369	2982	2925	3747	4883
OPT-30B		1500	1500	1329	1329	2193	2238
OPT-1.3B	4	34416	34225	31772	31585	54343	60091
OPT-2.7B		17746	17720	16148	16106	28998	30918
OPT-6.7B		7653	7653	7062	7062	12669	14417
OPT-13B		4091	4091	3730	3730	6676	7057
OPT-30B		1820	1820	1661	1661	3079	3221
OPT-1.3B	6	37643	36965	35375	35375	64031	67212
OPT-2.7B		19035	18904	17888	17823	32874	33943
OPT-6.7B		8305	8184	7835	7653	14406	15346
OPT-13B		4407	4398	4104	4104	7592	7962
OPT-30B		1936	1936	1812	1812	3343	3350
OPT-1.3B	8	38869	37777	36892	36744	68623	72164
OPT-2.7B		19907	19573	18930	18830	35544	36414
OPT-6.7B		8582	8522	8122	8122	15251	15944
OPT-13B		4584	4557	4325	4325	8142	8487
OPT-30B		2015	2001	1924	1924	3618	3802

our distributed execution strategies effectively offset the inherent performance cost of ZO2’s CPU offloading and block-wise scheduling across all BSs. In other words, even though ZO2 trades computation speed for memory efficiency, PertP and DDP restore its throughput to match that of MeZO without compromising scalability or memory savings.

**Combined 2D Parallelism Efficiency.** We combine PertP and DDP into a unified 2D parallelism framework, where each GPU performs a single forward pass on a disjoint data shard. Table 3, Columns 7–8, compare two design variants: (1) *DDP(inner) + PertP(outer)* and (2) *PertP(inner) + DDP(outer)*. Across all model sizes on 4 GPUs, the *PertP(inner) + DDP(outer)* strategy consistently achieves higher throughput across all BSs. This result empirically confirms our design choice and supports the theoretical justification presented in Section 6.2, where we argue that minimizing cross-branch communication and maintaining standard DDP compatibility makes PertP(inner) the preferable configuration.

**Effect of Communication Optimization.** To reduce PCIe bottlenecks, we introduce a communication strategy that slices parameter blocks and redistributes them via NVLink. Table 2 reports upload and offload throughput (in billion parameters/sec) before and after applying our optimization. We isolate communication bandwidth rather than overall training throughput, since computation may

still dominate end-to-end runtime. Nevertheless, our optimization achieves up to  $4.6\times$  improvement in both upload and offload speeds, which is essential for enabling effective overlap with forward computation in multi-GPU execution.

## 8 Conclusion

We present DistZO2, the first systematic framework for high-throughput and memory-efficient zeroth-order fine-tuning of large language models on CPU+multi-GPU heterogeneous systems. DistZO2 introduces three distributed strategies—Perturbation Parallelism, Distributed Data Parallelism, and their unified 2D parallelism—that significantly improve throughput while preserving the low-memory benefits of zeroth-order optimization. Our hardware-aware communication optimization further enhances efficiency by mitigating PCIe bottlenecks through block slicing and NVLink redistribution.

Through extensive experiments on models up to OPT-175B, we demonstrate that DistZO2 achieves up to  $3\times$  speedup over prior ZO methods, maintains sub-20GB GPU memory usage, and scales effectively across diverse model sizes. These results establish DistZO2 as a practical and scalable solution for fine-tuning multi-hundred-billion-parameter LLMs, opening a new direction for distributed zeroth-order optimization.

## Limitations

While DistZO2 demonstrates the first systematic exploration of distributed zeroth-order optimization on CPU and multi-GPU heterogeneous systems, several limitations remain. First, our study focuses exclusively on zeroth-order methods; extending the design to other optimization paradigms (e.g., hybrid zeroth- and first-order schemes) is left for future work. Second, although we verify the scalability of DistZO2 on multi-node clusters (up to 16 GPUs), our evaluation does not extend to massive-scale distributed environments. Empirical validation on industrial-scale clusters (e.g., hundreds of GPUs) would provide further evidence of practical scalability but is currently constrained by available computational resources. Finally, our evaluation is limited to fine-tuning of language models. Applying DistZO2 to pre-training scenarios or to domains beyond NLP (e.g., multimodal models) remains an open direction.

## Acknowledgments

This work is supported in part by the funding BAS/1/1689-01-01, RGC/3/7125-01-01, FCC/1/5940-20-05, FCC/1/5940-06-02, and King Abdullah University of Science and Technology (KAUST) – Center of Excellence for Generative AI, under award number 5940 and a gift from Google.

## References

- Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, and 1 others. 2024. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 929–947.
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.
- HanQin Cai, Yuchen Lou, Daniel McKenzie, and Wotao Yin. 2021. A zeroth-order block coordinate descent algorithm for huge-scale black-box optimization. In *International Conference on Machine Learning*, pages 1193–1203. PMLR.
- HanQin Cai, Daniel McKenzie, Wotao Yin, and Zhenliang Zhang. 2022. Zeroth-order regularized optimization (zoro): Approximately sparse gradients and adaptive sampling. *SIAM Journal on Optimization*, 32(2):687–714.
- Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. 2017. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 15–26.
- Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.
- Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *Machine learning challenges workshop*, pages 177–190. Springer.
- Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*.
- Marie-Catherine De Marneffe, Mandy Simons, and Judith Tonhauser. 2019. The commitmentbank: Investigating projection in naturally occurring discourse. In *proceedings of Sinn und Bedeutung*, volume 23, pages 107–124.
- Abraham D Flaxman, Adam Tauman Kalai, and H Brendan McMahan. 2004. Online convex optimization in the bandit setting: gradient descent without a gradient. *arXiv preprint cs/0408007*.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- Zhihao Jia, Matei Zaharia, and Alex Aiken. 2019. Beyond data and model parallelism for deep neural networks. *Proceedings of Machine Learning and Systems*, 1:1–13.
- Daniel Khoshdel, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. 2018. Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 252–262.
- Hector Levesque, Ernest Davis, and Leora Morgenstern. 2012. The winograd schema challenge. In *Thirteenth international conference on the principles of knowledge representation and reasoning*.

- Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and 1 others. 2020. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*.
- Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.
- Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.
- Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D Lee, Danqi Chen, and Sanjeev Arora. 2023. Fine-tuning language models with just forward passes. *Advances in Neural Information Processing Systems*, 36:53038–53075.
- Yurii Nesterov and Vladimir Spokoiny. 2017. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566.
- Mohammad Taher Pilehvar and Jose Camacho-Collados. 2018. Wic: the word-in-context dataset for evaluating context-sensitive meaning representations. *arXiv preprint arXiv:1808.09121*.
- Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE.
- Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.
- Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.
- Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*.
- Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.
- Liangyu Wang, Jie Ren, Hang Xu, Junxiao Wang, Huanyi Xie, David E Keyes, and Di Wang. 2025a. Zo2: Scalable zeroth-order fine-tuning for extremely large language models with limited gpu memory. *arXiv preprint arXiv:2503.12668*.
- Liangyu Wang, Junxiao Wang, Jie Ren, Zihang Xiang, David E Keyes, and Di Wang. 2025b. Private training large-scale models with efficient dp-sgd. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- Liangyu Wang, Siqi Zhang, Junjie Wang, Yiming Dong, Bo Zheng, Zihan Qiu, Shengkun Tang, Di Wang, Rui Men, and Dayiheng Liu. 2026. Canzona: A unified, asynchronous, and load-balanced framework for distributed matrix-based optimizers. *arXiv preprint arXiv:2602.06079*.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.
- Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, and 1 others. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.
- Yihua Zhang, Pingzhi Li, Junyuan Hong, Jiayang Li, Yimeng Zhang, Wenqing Zheng, Pin-Yu Chen, Jason D Lee, Wotao Yin, Mingyi Hong, and 1 others. 2024. Revisiting zeroth-order optimization for memory-efficient llm fine-tuning: A benchmark. *arXiv preprint arXiv:2402.11592*.
- Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P Xing, and 1 others. 2022. Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 559–578.

## Appendix

### A More Figures and Algorithms

#### B Full Related Work

Zeroth-order (ZO) optimization has emerged as a memory-efficient alternative for fine-tuning large language models (LLMs), as it avoids backward passes by estimating gradients using forward-only computations. MeZO (Malladi et al., 2023) and ZO2 (Wang et al., 2025a) demonstrate that ZO methods can significantly reduce memory usage and enable tuning of multi-hundred-billion-parameter models on a single GPU. However, these methods suffer from low throughput and poor scalability.

##### B.1 Zeroth-Order Optimization

Zeroth-order (ZO) optimization provides a gradient-free alternative to first-order methods by estimating gradients solely from function evaluations. Foundational approaches such as random perturbation-based gradient estimation (Flaxman et al., 2004) and two-point finite differences (Nesterov and Spokoiny, 2017) have established theoretical guarantees under smoothness assumptions. ZO methods have found applications in black-box adversarial attacks (Chen et al., 2017), reinforcement learning (Salimans et al., 2017), and model-agnostic optimization.

Despite scalability concerns due to dimension-dependent query complexity, techniques such as block-coordinate estimation (Cai et al., 2021) and sparsity-aware gradient estimation (Cai et al., 2022) have been proposed to improve efficiency in high-dimensional settings.

##### B.2 Zeroth-Order Optimization for LLM Fine-Tuning

As large language models (LLMs) grow in scale, memory-efficient fine-tuning becomes increasingly critical. MeZO (Malladi et al., 2023) introduced a backpropagation-free fine-tuning framework using ZO-SGD, enabling forward-only updates with drastically reduced memory footprint. This approach eliminates the need to store activation states and gradient tensors, allowing large-scale models to be fine-tuned on commodity hardware.

Subsequent work (Zhang et al., 2024) expanded the study of ZO methods by benchmarking a wide range of optimizers, including ZO-Adam,

---

#### Algorithm 1 MeZO (Malladi et al., 2023)

---

**Require:** Model parameters  $\theta \in \mathbb{R}^d$ , loss function  $L : \mathbb{R}^d \rightarrow \mathbb{R}$ , training iterations  $T$ , perturbation step size  $\epsilon$ , data batch  $B$ , learning rate  $\eta_t$

- 1: **for**  $j = 1, \dots, T$  **do**
- 2:   Set random seed  $s$  and sample batch  $B \subset D$
- 3:    $\theta \leftarrow \text{PerturbParameters}(\theta, \epsilon)$
- 4:    $\ell_+ \leftarrow L(\theta; B)$
- 5:   Reset RNG with seed  $s$
- 6:    $\theta \leftarrow \text{PerturbParameters}(\theta, -2\epsilon)$
- 7:    $\ell_- \leftarrow L(\theta; B)$
- 8:   Reset RNG with seed  $s$
- 9:    $\theta \leftarrow \text{PerturbParameters}(\theta, \epsilon)$
- 10:    $g \leftarrow (\ell_+ - \ell_-)/(2\epsilon)$
- 11:   Reset RNG with seed  $s$
- 12:    $\theta \leftarrow \text{UpdateParameters}(\theta, g)$
- 13: **end for**
  
- 14: **function** UPDATEPARAMETERS( $\theta, g$ )
- 15:   **for each**  $\theta_i \in \theta$ , where  $\theta_i \in \mathbb{R}^{d_i}$  **do**
- 16:      $z_i \sim \mathcal{N}(0, 1) \in \mathbb{R}^{d_i}$
- 17:      $\theta_i \leftarrow \theta_i - \eta_t \cdot g \cdot z_i$
- 18:   **end for**
- 19:   **return**  $\theta$
- 20: **end function**
  
- 21: **function** PERTURBPARAMETERS( $\theta, \epsilon$ )
- 22:   **for each**  $\theta_i \in \theta$ , where  $\theta_i \in \mathbb{R}^{d_i}$  **do**
- 23:      $z_i \sim \mathcal{N}(0, 1) \in \mathbb{R}^{d_i}$
- 24:      $\theta_i \leftarrow \theta_i + \epsilon z_i$
- 25:   **end for**
- 26:   **return**  $\theta$
- 27: **end function**

---

momentum-based ZO-SGD, and block-wise strategies. These improvements yield better stability and convergence, although most assume the full model fits in GPU memory.

More recently, ZO2 (Wang et al., 2025a) demonstrated that ZO optimization is particularly well-suited for CPU offloading due to its forward-only execution pattern. ZO2 introduces system-level innovations such as an RNG state manager, a lightweight CUDA stream-based scheduler, and AMP-aware low-bit communication. These enable efficient fine-tuning of models up to 175B parameters using just 18GB of GPU memory, without sacrificing accuracy or throughput.

Together, these works show that ZO-based fine-

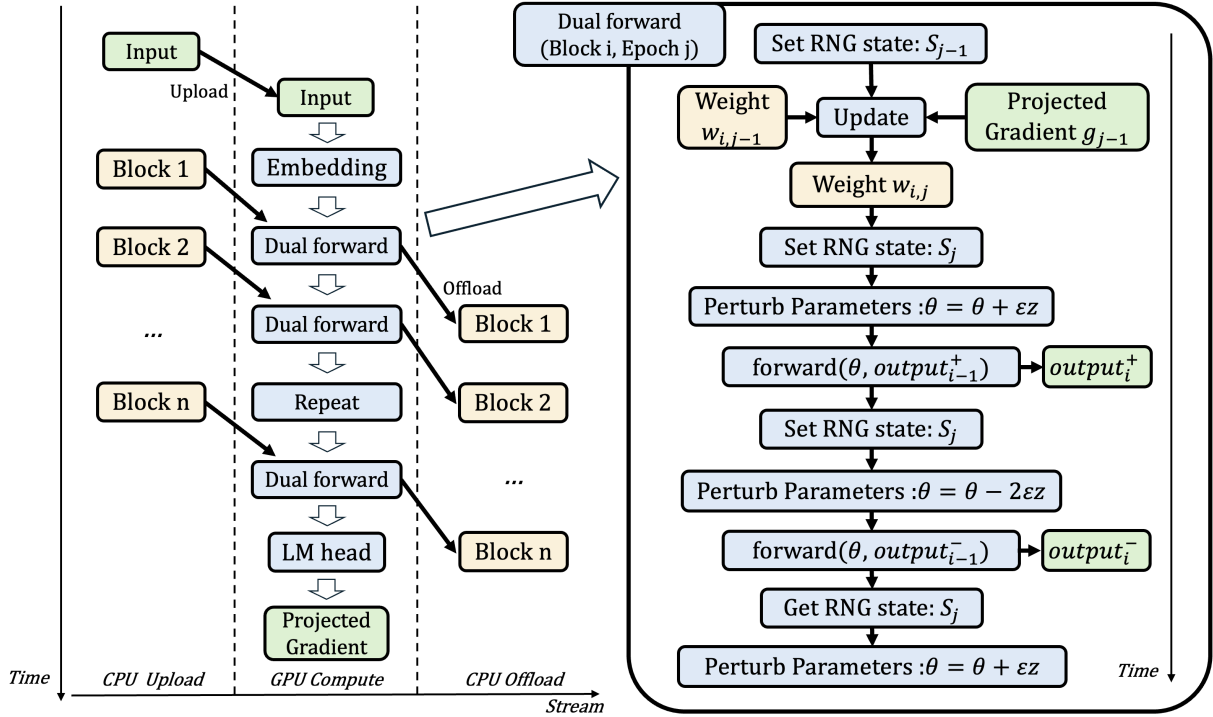


Figure 5: ZO2 (Wang et al., 2025a) workflow.

tuning, once considered too inefficient for large-scale models, is becoming a practical solution for memory-constrained LLM training.

### B.3 Distributed Computing

Meanwhile, distributed training strategies such as Distributed Data Parallelism (DDP) (Li et al., 2020) is widely used in first-order training (Ruder, 2016; Loshchilov and Hutter, 2017), but require adaptation for ZO due to its scalar-gradient nature and dual forward passes. In particular, DDP must synchronize scalar gradients and coordinate consistent perturbation vectors across devices. These adaptations are non-trivial and introduce scheduling and communication challenges.

To address these, DistZO2 first introduces Perturbation Parallelism (PertP), and then adapts DDP to support ZO training. Furthermore, it proposes a unified 2D parallelism framework that combines PertP with DDP, enabling parallelism along both perturbation and data dimensions. It further proposes a hardware-aware communication strategy that slices parameter blocks and redistributes them via fast interconnects like NVLink to alleviate PCIe bottlenecks. These innovations allow DistZO2 to significantly improve throughput while preserving ZO2’s memory efficiency. Our approach is also compatible with recent forward acceleration techniques such as FlashAttention (Dao, 2023) and

torch.compile (Ansel et al., 2024), which further emphasize the need for efficient communication in forward-only training.

### C Theoretical Communication Analysis

To rigorously quantify the benefits of our hardware-aware strategy, we formulate the communication cost model. Let  $M$  be the parameter size of a transformer block and  $n$  be the number of GPUs. In a standard ring-based or tree-based communication topology, the optimized transmission time per block,  $T_{\text{block}}$ , using our slicing strategy is modeled as:

$$T_{\text{block}} = \frac{M}{n \cdot B_{\text{PCIe}}} + \frac{(n-1)M}{n \cdot B_{\text{NVLink}}} \quad (3)$$

where  $B_{\text{PCIe}}$  and  $B_{\text{NVLink}}$  represent the effective bandwidths of PCIe and NVLink, respectively.

The total training time per step is determined by the maximum of the computation time and the communication time, due to the overlapping mechanism provided by CUDA streams:

$$T_{\text{step}} = \max(N \cdot T_{\text{block}}, T_{\text{comp}}) \quad (4)$$

where  $N$  is the number of transformer blocks and  $T_{\text{comp}}$  is the pure computation time for the dual forward passes. This model demonstrates that as  $B_{\text{NVLink}} \gg B_{\text{PCIe}}$ , the inter-GPU communication

---

**Algorithm 2** ZO2 (Wang et al., 2025a) Dynamic Scheduler

---

**Require:** Transformer blocks  $\{W_i\}_{i=1}^N$  with number of transformer blocks  $N$ , embedding parameters  $Embedding$ , and LM head  $LMhead$ .

- 1: Initialize a dynamic scheduler  $S\{\cdot\}$  to control dual forward computation  $C(\cdot)$ , uploading  $U(\cdot)$ , and offloading  $O(\cdot)$  operations.
  - 2: Asynchronously launch  $S\{C(Embedding), U(W_1)\}$ .
  - 3: **for**  $i = 1$  to  $N - 1$  **do**
  - 4:     Synchronously wait until  $U(W_i)$  finished.
  - 5:     **if**  $i = 1$  **then**
  - 6:         Asynchronously launch  $S\{C(W_i), U(W_{i+1})\}$ .
  - 7:     **else**
  - 8:         Synchronously wait until  $C(W_{i-1})$  finished.
  - 9:         Asynchronously launch  $S\{O(W_{i-1}), C(W_i), U(W_{i+1})\}$ .
  - 10:     **end if**
  - 11: **end for**
  - 12: Synchronously wait until  $C(W_{N-1})$  and  $U(W_N)$  finished.
  - 13: Asynchronously launch  $S\{O(W_{N-1}), C(W_N)\}$ .
  - 14: Synchronously wait until  $C(W_N)$  finished.
  - 15: Asynchronously launch  $S\{O(W_N), C(LMhead)\}$ .
- 

overhead becomes negligible, and the bottleneck is effectively distributed across parallel PCIe lanes.

### Hardware Generality and Performance Bounds.

This theoretical model highlights two critical properties of the DistZO2 communication design: **1. Optimization for Modern Infrastructure:** As  $B_{NVLink} \gg B_{PCIe}$  in modern high-performance clusters (e.g., NVIDIA A100/H100 systems), the second term in Eq. (3) becomes negligible. This allows the system to effectively bypass the PCIe bottleneck by offloading the majority of data transfer to the high-speed intra-node interconnect, maximizing throughput. **2. Performance Lower Bound on Commodity Hardware:** In commodity environments lacking high-speed interconnects (i.e., where  $B_{NVLink} \approx B_{PCIe}$  or purely PCIe-based), the slicing strategy naturally degrades to the baseline transmission performance. Importantly, the system ensures a performance lower bound: it performs no worse than standard unoptimized transmission. This guarantees that DistZO2 remains fully functional and robust across diverse hardware configurations, treating the high-speed interconnect as an accelerator rather than a hard dependency.

## D More Experiments

### D.1 Evaluation on Qwen3 Models

We chose the OPT family because it is the de facto model suite in the ZO optimization literature—used

by MeZO (Malladi et al., 2023) and ZO2 (Wang et al., 2025a), which makes it ideal for fair comparison. Nevertheless, our framework is fully compatible with modern architectures such as LLaMA and Qwen, since our offloading and scheduling mechanisms operate at the transformer-block level. Each transformer block is treated as an independent scheduling unit, and both LLaMA and Qwen share the same stacked-block structure as OPT.

To further validate our proposed PerP, DDP, and 2D parallelism strategies, we conducted additional experiments on the Qwen3 (Yang et al., 2025). Following the same experimental settings as with the OPT-13B model, our results confirm that these strategies are equally effective. As detailed in Table 4, our methods provide significant improvements in training efficiency and resource utilization for the Qwen3 architecture as well, demonstrating their broad applicability to modern large language models.

### D.2 Additional Scaling Results with GPU Count

To further validate the scalability of DistZO2, we compare MeZO+DDP and ZO2+DDP across 1, 2 and 4 GPUs while fixing the batch size to 4 and using fp16 precision. As shown in Table 5, both methods scale consistently with GPU count, achieving nearly linear throughput improvements from 1 to 4 GPUs. Importantly, the performance gap between MeZO and ZO2 remains negligible through-

out, confirming that ZO2’s offloading mechanism does not introduce any significant parallelization overhead when using DDP.

Table 4: **Throughput (Tokens per Second) Analysis On Qwen3.**

Model	BS	ZO2 +PertP	ZO2 +DDP	ZO2 +PertP+DDP
Qwen3-1.7B		27415	8482	40590
Qwen3-4B		14028	4881	23540
Qwen3-8B	4	6599	2158	10628
Qwen3-14B		3845	1273	6250
Qwen3-32B		2546	867	3973

Furthermore, we evaluated the scalability of DistZO2 on a multi-node cluster to assess its performance in constrained bandwidth environments. Experiments were conducted on Qwen models using the Unified 2D Parallelism strategy across 1, 2, and 3 nodes (with 4 GPUs per node). As shown in Figure 6, DistZO2 achieves near-linear scalability even across node boundaries. (It should be noted that while our single-node main experiments utilized NVIDIA H20 GPUs, this multi-node evaluation was conducted on NVIDIA A100-80GB nodes due to the unavailability of a multi-node H20 cluster.)

It is worth noting that inter-node communication typically presents a significant bottleneck due to lower bandwidth compared to intra-node NVLink. DistZO2 effectively circumvents this limitation through its ZO-adapted DDP mechanism: unlike first-order methods that typically require synchronizing massive gradient tensors across the cluster, DistZO2 only needs to communicate a **single scalar projected gradient** and a random seed per iteration. Consequently, the inter-node communication overhead remains negligible, allowing the system to scale efficiently even on clusters with limited interconnect bandwidth.

### D.3 Sequence Length Sensitivity Across Parallelism Strategies

We also evaluate how different parallelism strategies perform under varying sequence lengths, ranging from 1024 to 8192. As shown in Table 7, all methods experience reduced throughput with longer sequences, as expected. However, DistZO2 maintains strong relative performance across all sequence lengths, with the unified 2D strategy (ZO2+PertP+DDP) consistently achieving the highest throughput.

Table 5: **Throughput (Tokens/Second) Comparison Across Models and GPU Counts.** Batch size is fixed at 4, using fp16 precision.

Model	GPUs	MeZO+DDP	ZO2+DDP
OPT-1.3B	1	18509	18446
OPT-2.7B		9485	9566
OPT-6.7B		4062	4060
OPT-13B		2177	2179
OPT-30B		968	961
OPT-1.3B	2	31772	31585
OPT-2.7B		16148	16106
OPT-6.7B		7062	7062
OPT-13B		3730	3730
OPT-30B		1661	1661
OPT-1.3B	4	48041	45684
OPT-2.7B		24802	24640
OPT-6.7B		11234	11284
OPT-13B		5966	5955
OPT-30B		2658	2637

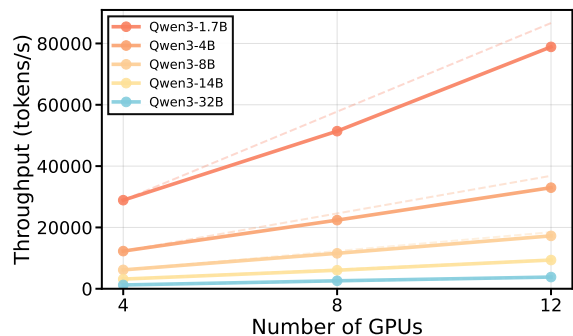


Figure 6: Throughput (Tokens/Second) Comparison Across Models and Nodes.

## D.4 Results on Accuracy Alignment

The primary contribution of DistZO2 is engineering scalability and throughput optimization. As mentioned before, criticisms regarding the performance of ZO methods on challenging tasks target the optimization algorithm itself rather than the distributed system design. However, to rigorously verify that our proposed distributed strategies (Perturbation Parallelism and ZO-adapted DDP) do not introduce any implementation-level errors or numerical instability compared to the original algorithm, we conducted an additional verification on accuracy.

Following ZO2’s experiment settings, to evaluate the accuracy of DistZO2, we tested it against the OPT-13B model on seven standard NLP benchmarks: SST-2, RTE, CB, BoolQ, WSC, WIC, and MultiRC (Socher et al., 2013; Dagan et al., 2005; De Marneffe et al., 2019; Clark et al., 2019; Levesque et al., 2012; Pilehvar and Camacho-Collados, 2018; Khashabi et al., 2018). These tasks cover a range of language understanding abilities, including sentiment analysis, entailment, coreference, QA, and word sense disambiguation. As shown in Table 6, DistZO2 achieves identical precision rates to the baseline MeZO and ZO2 on all benchmarks, demonstrating its effectiveness in maintaining model accuracy. This is particularly important since MeZO has already been compared to first-order (FO) methods, allowing us to focus our analysis on the performance parity between our method and its direct predecessor.

## E More Information

### E.1 Potential Risks

As with many efficiency-focused training frameworks, DistZO2 lowers the computational barrier for fine-tuning large language models. This improved accessibility may also increase the risk of misuse, such as training models for harmful content generation. In addition, although our design improves efficiency, large-scale training still consumes considerable compute and energy, potentially contributing to resource concentration among well-funded institutions. Finally, zeroth-order optimization inherently depends on stochastic perturbations, which may introduce stability risks if synchronization across devices is not carefully maintained.

### E.2 Citations of Artifacts

In this work, we make use of several publicly released artifacts and cite their original creators accordingly. For pretrained models, we evaluate on the OPT series of large language models (Zhang et al., 2022) on dataset SST-2 (Socher et al., 2013). For zeroth-order baselines, we include MeZO (Maladi et al., 2023) and ZO2 (Wang et al., 2025a). For software frameworks, our implementation relies on PyTorch (Li et al., 2020) and standard GPU communication libraries (e.g., NCCL). All datasets and code dependencies used in our experiments are cited in their original publications where appropriate. All these artifacts are distributed under permissive research or open-source licenses: OPT is released by Meta under a research-use license, MeZO and ZO2 implementations are publicly available under open-source licenses, PyTorch is licensed under a BSD-style license, and datasets are released under Creative Commons or similar research-use terms. We ensure compliance with all corresponding licenses and terms of use.

Furthermore, our use of existing artifacts is consistent with their intended purpose. The OPT models are released for research use only, and our experiments comply with this restriction. MeZO and ZO2 are employed as baselines in the manner described by their original authors, and our reliance on open-source frameworks such as PyTorch and NCCL follows their intended usage. For the new DistZO2 framework, we release our code for research purposes only, fully compatible with the access conditions of the original artifacts on which it builds.

### E.3 Data Statement

Our experiments rely on publicly available models and datasets, including the OPT series (Zhang et al., 2022), which are released under research-use licenses. We did not collect any new data. To the best of our knowledge, these datasets have undergone filtering by their original creators, and we did not encounter personally identifiable information (PII) during our experiments. As our work focuses on system-level optimization, we do not alter the data distribution nor introduce new privacy or content risks.

### E.4 Artifact Documentation

We rely on publicly released artifacts such as the OPT models (Zhang et al., 2022), which are doc-

Table 6: **Main results of ZO2 precision on OPT-13B**

Method	SST-2 (%)	RTE (%)	CB (%)	BoolQ (%)	WSC (%)	WIC (%)	MultiRC (%)
MeZO	91.4	66.1	67.9	67.6	63.5	61.1	60.1
ZO2	91.4	66.1	67.9	67.6	63.5	61.1	60.1
DistZO2	91.4	66.1	67.9	67.6	63.5	61.1	60.1

umented by their creators with respect to training data domains, languages, and filtering processes. We do not introduce new datasets in this work. Our contribution, the DistZO2 framework, will be released with accompanying documentation describing installation instructions, supported model types, and usage examples to facilitate reproducibility.

### E.5 Statistical Reporting

All reported results are from a single run. We did not repeat experiments due to the prohibitive cost of large-scale GPU runs. As our evaluation focuses on system-level metrics such as throughput and memory usage, which are highly stable across runs, we expect variance to be negligible.

### E.6 Implementation Details

Our implementation is based on PyTorch (Li et al., 2020) with NCCL for GPU communication. We evaluate using publicly released OPT models (Zhang et al., 2022), with configurations following their official implementation. No additional preprocessing or external evaluation packages (e.g., NLTK, ROUGE) are used. All hyperparameters and training settings (batch size, learning rate, sequence length) are reported in Section 7.

### E.7 Use of AI Assistants

We used AI assistants (e.g., ChatGPT) to support grammar checking, text polishing, and editing suggestions during the preparation of this manuscript. All technical contributions, experiments, and analysis were designed and carried out by the authors.

### E.8 Comparison with First-Order Baselines

Our evaluation focuses on comparing DistZO2 against state-of-the-art zeroth-order methods (MeZO, ZO2). We prioritize this comparison because the advantages of zeroth-order methods over first-order (FO) optimizers in strictly memory-constrained settings have been well-established in prior literature (Malladi et al., 2023; Wang et al., 2025a).

Furthermore, from a theoretical standpoint, DistZO2 inherently minimizes per-step latency

compared to FO baselines. Standard first-order training requires both a forward propagation and a computationally expensive backward propagation per iteration. In contrast, DistZO2’s Perturbation Parallelism distributes the workload such that each device executes only a single forward pass per step. By structurally eliminating the backward pass, DistZO2 achieves a theoretical throughput advantage over FO methods under identical offloading conditions, making the intra-class comparison with other ZO methods the most relevant metric for assessing system scalability.

Table 7: **Throughput (Tokens per Second) Analysis Across Different Parallelism Methods.**

Model	Sequence Length	ZO2 +PerP	MeZO +DDP	ZO2 +DDP	ZO2 +DDP+PertP	ZO2 +PertP+DDP
OPT-1.3B	1024	29701	24845	24194	33796	32608
OPT-2.7B		15110	12850	12399	11070	10231
OPT-6.7B		7250	5738	5711	5151	5351
OPT-13B		2118	3034	2996	2807	2658
OPT-30B		1703	1344	1344	1119	1239
OPT-1.3B	2048	34225	31772	31585	54343	60091
OPT-2.7B		17720	16148	16106	28998	30918
OPT-6.7B		7653	7062	7062	12669	14417
OPT-13B		4091	3730	3770	6676	7057
OPT-30B		1820	1661	1661	3079	3221
OPT-1.3B	4096	35008	33762	33465	63377	44876
OPT-2.7B		17983	17326	17234	22558	24713
OPT-6.7B		8077	7715	7763	11033	10941
OPT-13B		4327	4147	4135	5917	5943
OPT-30B		1920	1863	1860	2657	2625
OPT-1.3B	8192	31948	31269	31236	61766	51007
OPT-2.7B		16231	15955	15942	26245	26741
OPT-6.7B		7782	7609	7625	12581	12668
OPT-13B		4218	4145	4139	6867	6886
OPT-30B		1780	1883	1874	3116	3100

---

**Algorithm 3** ZO2 (Wang et al., 2025a) Computation with RNG State Manager

---

**Require:** Transformer blocks  $\{W_i\}_{i=1}^N$  with number of transformer blocks  $N$ , embedding parameters  $Embedding$ , and LM head  $LMhead$ , module parameter  $\theta$ , module forward function  $forward$ , loss function  $L$ , training iterations  $T$ , perturbation step size  $\epsilon$ , data batch  $B$ , learning rate  $\eta_t$ , random seed  $s$ , random state buffer  $rsb$ , last iteration's random state  $lrs$ .

```
1: Initialize  $g = 0$ .
2: for  $j = 1, \dots, T$  do
3:   Set random seed  $s$  and sample batch  $B \subset D$ 
4:   Get random state  $rs = \text{GetRngState}(s)$ , and push  $rs$  into  $rsb$ .
5:   if  $j > 1$  then
6:     Update last iteration's random state  $lrs = \text{PopLeft}(rsb)$ 
7:   else
8:      $lrs = \text{None}$ 
9:   end if
10:   $out_+, out_-, rs, lrs = \text{DualForward}(Embedding, \epsilon, s, rs, lrs, g, B, B)$ 
11:  for  $i = 1$  to  $N$  do
12:     $out_+, out_-, rs, lrs = \text{DualForward}(W_i, \epsilon, s, rs, lrs, g, out_+, out_-)$ 
13:  end for
14:   $out_+, out_-, rs, lrs = \text{DualForward}(LMhead, \epsilon, s, rs, lrs, g, out_+, out_-)$ 
15:   $\ell_+ = L(out_+), \ell_- = L(out_-)$ 
16:   $g \leftarrow (\ell_+ - \ell_-)/(2\epsilon)$ 
17: end for

18: function DUALFORWARD( $\theta, \epsilon, s, rs, lrs, g, input_+, input_-$ )
19:   if  $g \neq 0$  then
20:     SetRngState( $s, lrs$ );  $\theta, lrs \leftarrow \text{UpdateParameters}(\theta, g)$ 
21:   end if
22:   SetRngState( $s, rs$ ),  $\theta \leftarrow \text{PerturbParameters}(\theta, \epsilon)$ 
23:    $out_+ \leftarrow forward(\theta; input_+)$ 
24:   SetRngState( $s, rs$ ),  $\theta \leftarrow \text{PerturbParameters}(\theta, -2\epsilon)$ 
25:    $out_- \leftarrow forward(\theta; input_-)$ 
26:   SetRngState( $s, rs$ ),  $\theta \leftarrow \text{PerturbParameters}(\theta, \epsilon)$ 
27:    $rs = \text{GetRngState}(s)$ 
28:   return  $out_+, out_-, rs, lrs$ 
29: end function

30: function SETRNGSTATE( $s, rs$ )
31:   Store  $rs$  associated with seed  $s$ .
32: end function

33: function GETRNGSTATE( $s$ )
34:   return  $rs$  associated with seed  $s$  from the storage.
35: end function

36: function UPDATEPARAMETERS( $\theta, g$ )
37:   Same as Algorithm 1's UpdateParameters
38: end function

39: function PERTURBPARAMETERS( $\theta, \epsilon$ )
40:   Same as Algorithm 1's PerturbParameters
41: end function
```

---