This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2026.3743842

Make the Parallel Truly Parallel: Interleaving Parallel ZeRO-DP for RLHF Training with *ZeroFuse*

Wenhao Huang, Yuhao Zhang, Laiping Zhao, Yiming Li, Yichi Chen, Yanjie Wang, Fupeng Li, Yihan Wang, Zhihang Tang, Deze Zeng, Keqiu Li *Fellow, IEEE*

***Abstract*—Reinforcement Learning from Human Feedback (RLHF) is highly time-consuming, with the training stage—updating the *Actor* and *Critic* models—dominating runtime. Existing systems often adopt the Zero Redundancy Data Parallelism (ZeRO-DP) paradigm to accelerate the training stage. However, they schedule the independent *Actor* and *Critic* models sequentially on GPUs. This yields large forward-compute and backward-communication bubbles due to the frequent small collective communications inherent to ZeRO-DP, resulting in low GPU and bandwidth utilization.**

To address this problem, we present *ZeroFuse*, which realizes truly interleaved *Actor–Critic* parallel execution under ZeRO-DP. *ZeroFuse* introduces the *LLMchunk* abstraction—a contiguous set of transformer blocks sized so that per-chunk compute time matches per-chunk communication time—and constructs a performance model to derive analytic bubble layouts. It then exploits bubble divergence by overlapping one model's forward pass with the other's backward pass and employs a dynamic-programming orchestrator to select an order-preserving, communication-exclusive, memory-feasible parallel schedule. Experiments show that, compared with state-of-the-art RLHF systems, *ZeroFuse* reduces *training* stage latency by $1.32\times$, substantially improving GPU computation and bandwidth utilization.

***Index Terms*—RLHF training, ZeRO-DP, Interleaved parallelism**

I. Introduction

REINFORCEMENT Learning from Human Feedback (RLHF) [1] aligns pre-trained LLMs (e.g., ChatGPT [2], Claude [3], Gemini [4], and Llama [5]) with human preferences by fine-tuning them using preference signals; it typically involves a complex workflow with multi-model, multi-stage inference and training. To simplify the RLHF pipeline and reduce training complexity, several RLHF frameworks have been proposed, such as Microsoft's DeepSpeed-Chat [6], NVIDIA's NeMo-RL [7], Alibaba's Roll [8], ByteDance's HybridFlow [9] and others [10]–[13].

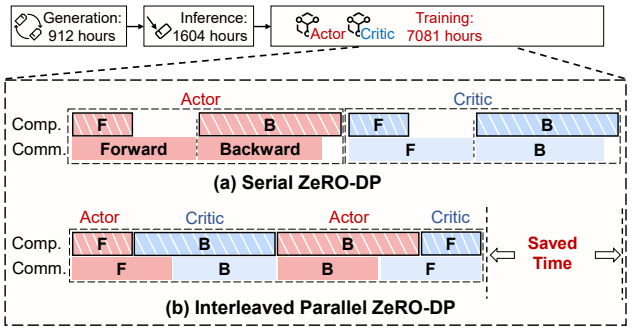


Fig. 1. Schematic overview of two ZeRO-DP methods. (a) Serial ZeRO-DP: existing work executes the *Actor* and *Critic* models sequentially. (b) Interleaved parallel ZeRO-DP: we achieve true parallelism through interleaved scheduling.

RLHF, although less costly than pre-training, remains highly time-consuming. For example, it requires at least 9,600 GPU-hours on NVIDIA A100 GPUs to train LLaMA2-70B [5], [14]. The RLHF process typically consists of three stages: (1) the *generation* stage, which samples prompts and generates responses; (2) the *inference* stage, which evaluates the generated responses to produce reward/value scores; and (3) the *training* stage, which updates model parameters based on these scores, as shown in Figure 1. As recent RLHF systems reduce the *generation* time through speculative inference [12], [15], [16] and redundant inference [17], [18], while leaving *training* latency unchanged, the *training* stage increasingly dominates the total RLHF runtime. Taking LLaMA2-70B as an example, the *training* stage alone accounts for over 73.77% of the end-to-end RLHF duration, highlighting the need to optimize this stage to reduce overall training time.

Existing RLHF frameworks [6], [8]–[10] train both the *Actor* and the *Critic* models during the *training* stage. They typically adopt ZeRO-DP (Zero Redundancy Data Parallelism, hereafter referring specifically to ZeRO-3) [19] for data parallelism within 5D parallel training, trading additional communication for reduced memory redundancy [20]–[24]. Unlike ZeRO-1 and ZeRO-2, which retain replicated parameters across data-parallel ranks, ZeRO-3 additionally shards parameters to reduce per-GPU memory consumption. This is particularly valuable in large-scale RLHF training, where the *Actor* and *Critic* require substantial memory for parameters, gradients, and optimizer states. However, parameter sharding requires repeated parameter all-gathers for

This work is supported by the National Key Research and Development Program of China under Grant No.2024YFB4505204, the National Natural Science Foundation of China under Grant 62372322, 62432015, Tianjin Science and Technology Plan Project 24ZXKJGX00060, and Beijing Natural Science Foundation L2603007.

Wenhao Huang, Yuhao Zhang, Laiping Zhao, Yiming Li, Yichi Chen, Yanjie Wang, Fupeng Li, Yihan Wang, Zhihang Tang and Keqiu Li are with College of Intelligence and Computing, Tianjin University, Tianjin 300350, China. (email: hwh@tju.edu.cn, yuhaozhang@tju.edu.cn, laiping@tju.edu.cn, l_ym@tju.edu.cn, chenyichi@tju.edu.cn, wangyanjie@tju.edu.cn, lfp_61@tju.edu.cn, ehein@tju.edu.cn, tangzhihang@tju.edu.cn, keqiu@tju.edu.cn).

Deze Zeng is with the School of Computer Science, China University of Geosciences, Wuhan, Hubei 430074, China. (email: deze@cug.edu.cn).

Laiping Zhao is the corresponding author.

Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on October 27,2026 at 12:11:43 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2026.3743842

forward and backward computation, which can leave exposed communication and computation bubbles despite compute–communication overlap. We therefore focus on ZeRO-3 to reduce these additional bubbles while preserving its memory savings.

Existing systems follow a "**parallel structure, serial execution**" approach, in which one model (e.g., the *Actor*) is updated with ZeRO-DP first, followed by the other (e.g., the *Critic*), as shown in Figure 1(a). However, our experiments on LLaMA2-70B show that the *training* stage still incurs at least 27.49% idle computation time, namely **compute bubbles** caused by slow communication, leading to substantial GPU underutilization. This inefficiency stems from ZeRO-DP's frequent parameter communication (up to 250 operations per second under the default 30 MB communication volume limit), as well as some gradient communication. Each communication operation involves kernel launches, multi-process synchronization, and other coordination costs, introducing substantial runtime overhead [25]–[27].

Given that the *Actor* and *Critic* are independent, a straightforward approach is to parallelize their execution and use one model's computation (or communication) to fill the other's bubbles, thereby improving overall execution efficiency. We refer to this design as the **"parallel structure, interleaved parallel execution"** paradigm (Figure 1(b)).

However, achieving interleaved parallel ZeRO-DP is highly challenging. *Challenge #1: Resource contention*. Since the *Actor* and *Critic* typically share the same transformer architecture, their demands for GPU compute and network bandwidth are highly similar, and executing them concurrently on limited GPU resources can result in severe contention. *Challenge #2: Parameter tuning*. Computation and communication times vary dynamically with factors such as batch size and model size. In parallel ZeRO-DP execution, we need to tune parameters—e.g., parallelism granularity and communication chunk size—so that the two models' bubble sizes match, allowing one model's bubbles to be precisely filled by the other.

To address *challenge #1: resource contention*, we analyze the computation and communication characteristics of ZeRO-DP in the forward and backward pass, and identify a bubble divergence phenomenon: *large compute bubbles in the forward pass and small communication bubbles in the backward pass*. Leveraging this insight, we design an interleaved-parallel execution strategy that staggers computation and communication across the two models, avoiding simultaneous contention for GPU compute units or network bandwidth and thereby mitigating performance degradation from resource contention.

To address *challenge #2: parameter tuning*, we analyze the computational and communication bubbles under different communication volumes, and identify a bubble balance phenomenon: as the communication volume increases, *compute bubbles in the forward pass approach the communication bubbles in the backward pass*. Based on this insight, we design a new abstraction, called *LLMchunk*, which consists of multiple transformer blocks of an LLM, serves as the basic unit of computation–communication, and conforms to the aforementioned bubble balance phenomenon. By solving for *LLMchunk*, we enable precise mutual filling of computational and communication bubbles under interleaved parallel execution.

Based on these techniques, we implement ***ZeroFuse***, the first RLHF system that employs interleaved parallel ZeRO-DP to mutually fill computational and communication bubbles between two concurrently trained ZeRO-DP models.

In summary, our contributions are as follows:

- **Analysis and discovery**: The large compute bubble stems from ZeRO-DP's excessively high communication frequency. Reducing the frequency enables intra-model bubble balancing.
- ***LLMchunk* abstraction**: A new abstraction for precise mutual filling of bubbles via interleaved parallel execution of *Actor* and *Critic* models.
- ***ZeroFuse* prototype**: An RLHF system featuring a parallel ZeRO-DP engine, bubble solver, orchestrator, and parameter reshuffle strategy.
- **Experimental evaluations**: Compared with state-of-the-art RLHF systems, ***ZeroFuse*** reduces *training* stage latency by a factor of 1.32×.

II. Background & Motivation

In this section, we present the background of RLHF and ZeRO-DP, and analyze the opportunities for parallel ZeRO-DP to reduce the training time.

A. RLHF Background

RLHF is a technique for aligning LLMs through reinforcement learning. It is performed after pre-training, when the model possesses only general language understanding and generation capabilities. RLHF first applies supervised fine-tuning (SFT) on a human-curated instruction–response dataset to teach the model to follow instructions, then trains a reward model (RM) to quantify human preferences, and finally employs reinforcement learning algorithms such as Proximal Policy Optimization (PPO) [2] to further optimize the policy model based on the reward signal, enabling the model to generate outputs that better align with human values and task requirements.

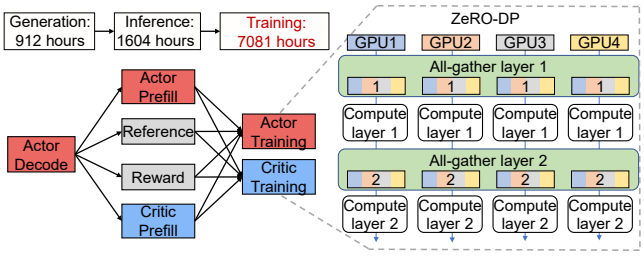
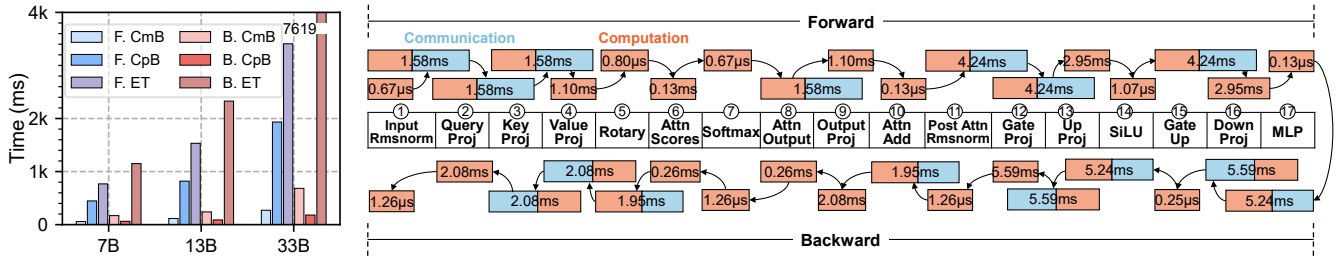


Fig. 2. Workflow of RLHF and ZeRO-DP.

As illustrated in Figure 2, RLHF follows a typical workflow comprising three stages and seven sub-tasks. Stage (1): *generation* uses the *Actor* model to produce responses via prompt-based autoregressive decoding. Stage (2): *inference* involves the *Actor*, *Reference*, *Reward*, and *Critic* models generating logits or values for each sample. Stage (3): *training* updates the *Actor* and *Critic* using reinforcement learning algorithms such as PPO. Among the three stages, Stage (3) constitutes

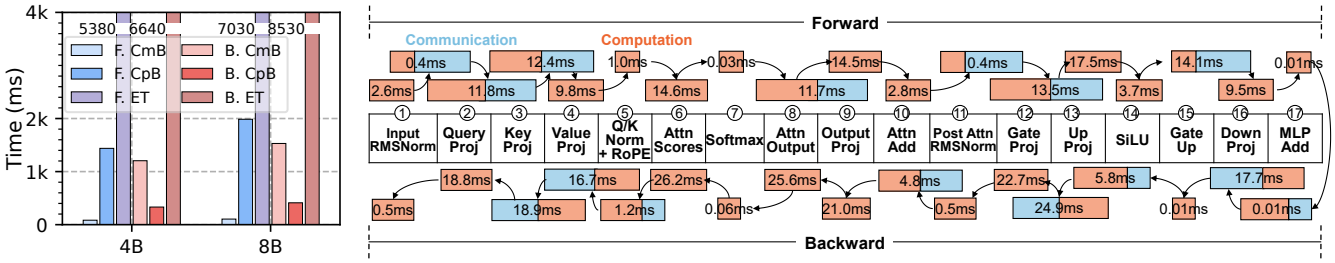
Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on October 27,2026 at 12:11:43 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2026.3743842



(a) Bubbles and execution time for LLaMA-7B/13B/33B.

(b) Runtime breakdown for a LLaMA-7B transformer block under ZeRO-3 parallelism on 32 A100 GPUs.



(c) Bubbles and execution time for Qwen-4B/8B.

(d) Runtime breakdown for a Qwen-8B transformer block under TP4 + ZeRO-3 parallelism on 8 H20 GPUs.

Fig. 3. In training execution time (ET), the forward pass (F) is dominated by compute bubbles (CpB), whereas the backward pass (B) is dominated by communication bubbles (CmB). These bubbles arise from the mismatch between computation and communication across layers within a transformer block.

the primary time bottleneck. For example, in LLaMA2-70B training, Stage (3) takes 7,081 out of 9,600 GPU-hours, accounting for 73.77%.

ZeRO-DP is a highly efficient data-parallel strategy widely adopted in state-of-the-art RLHF frameworks during Stage (3). By sharding model parameters, optimizer states, and gradients across different GPUs for storage and computation, it avoids the redundant storage of full parameters on each device as in conventional data parallelism, while achieving near-standard data-parallel compute efficiency in large-scale training. As illustrated in the Figure 2, the core mechanism of ZeRO-DP is that, for each layer to be computed, all GPUs perform an all-gather operation to collect the corresponding parameter shards from other devices and concatenate them into the full weights for forward or backward compute. In addition, during backpropagation, ZeRO-DP further transfers gradients via reduce-scatter, with the data volume being approximately one quarter of the parameter size [28]. Furthermore, ZeRO-DP overlaps the compute of the current layer with the parameter all-gather communication of the next layer to improve training efficiency.

B. Imbalanced Bubbles in ZeRO-DP

To analyze the efficiency of ZeRO-DP, we evaluate LLaMA-7B, 13B and 33B on 32 NVIDIA A100 GPUs under ZeRO-3 parallelism, and Qwen-4B/8B on 8 NVIDIA H20 GPUs under TP4 + ZeRO-3 parallelism, and examine their computation and communication characteristics. The detailed experimental configurations are provided in section V-A.

Observation #1: *Frequent collective communication in ZeRO-DP leads to high communication overhead, resulting in "**large compute bubbles in the forward pass and few communication bubbles in the backward pass.**"*

We use NVIDIA's `nsys` tool [29] to record the distrbution of bubbles during ZeRO-DP. These bubbles include **compute bubbles**——idle computation time caused by slow communication, where GPU compute resources remain unused——and **communication bubbles**——idle network bandwidth time caused by slow computation.

As shown in Figure 3(a) and (c), for LLaMA-7B/13B/33B and Qwen-4B/8B, respectively, forward compute bubbles are 446.98/819.89/1933.76 ms and 1438/1986 ms, while forward communication bubbles are 58.27/116.20/271.48 ms and 84/105.3 ms. In the backward pass, communication bubbles are 170.32/240.43/684.11 ms and 1205/1530 ms, compared with compute bubbles of 63.95/89.11/180.96 ms and 332/412 ms. Thus, compute bubbles dominate the forward pass (58.20%/53.48%/56.74% and 26.7%/28.3% of forward latency), whereas communication bubbles dominate the backward pass (14.81%/10.33%/8.98% and 18.1%/17.9% of backward latency). This yields a bubble pattern characterized by "*large compute bubbles in the forward pass and small communication bubbles in the backward pass*".

ZeRO-DP bubble breakdown: Using LLaMA-7B and Qwen-8B as the Actor (or Critic) models, and excluding the embedding and output layers, the models consist of 32 and 36 transformer blocks, respectively, accounting for more than 98.35% and 97.41% of their respective end-to-end latencies. These blocks share the same structure within each model and execute sequentially. Therefore, analyzing the computation and communication within a single transformer block of each model allows us to pinpoint where bubbles arise and why.

Figure 3(b) and (d) present detailed breakdowns of computation and communication time within a transformer block during LLaMA-7B and Qwen-8B training, respectively. To align the two breakdowns, Qwen's *Q Norm*, *K Norm*, and *RoPE*

Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on October 27,2026 at 12:11:43 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2026.3743842

are grouped into the slot corresponding to LLaMA's *Rotary* operation, adding lightweight normalization computation and associated parameter communication. The remaining operation slots correspond one-to-one and exhibit similar computation–communication characteristics. Each block is represented by 17 operation slots: seven GEMM-based projections (Query, Key, Value, Output, Gate, Up, and Down) with substantial computation and communication costs, and ten other operations (e.g., Softmax, SiLU).

In the forward pass, ZeRO-DP pipelines computation and communication by overlapping the computation of the current layer with the communication of the next layer. Both Figure 3(b) and (d) exhibit the resulting compute-bubble pattern; we use LLaMA-7B in Figure 3(b) for the following detailed overlap analysis. Among the 16 overlaps, in 7 cases the next layer is a matrix multiplication, where the communication time generally exceeds the computation time (layer ①, ②, ③, ⑧, ⑪, ⑫, ⑮ in the figure); in 4 cases the current layer is a compute-intensive matrix multiplication while the next layer is a pointwise operation, where the communication time is less than the computation time (layer ④, ⑨, ⑬, ⑯ in the figure); in the remaining overlaps, the difference is marginal. Overall, communication takes 19.07 ms while computation takes 13.56 ms in the forward pass, resulting in compute bubbles. Similarly, Qwen-8B has approximately 5.04 s of computation and 1.99 s of compute bubbles over the full forward pass, yielding a wall time of 7.03 s.

In the backward pass, ZeRO-DP overlaps the computation of the current layer with the communication of the previous layer. Since the backward pass requires an additional calculation of gradient, the backward computation volume is approximately doubled relative to the forward pass, while the parameter-communication volume remains unchanged; additionally, the gradient communication increases by about one quarter of the parameter size [28]. Consequently, in 7 of the 16 overlaps the previous layer is a matrix multiplication and the computation time exceeds the previous layer's communication time (layer ②, ③, ④, ⑨, ⑫, ⑬, ⑯ in the figure), whereas in the remaining overlaps communication dominates. Overall, communication takes 23.53 ms and computation takes 25.62 ms in the backward pass, resulting in communication bubbles. The same pattern appears for Qwen-8B in Figure 3(d), with approximately 1.53 s of communication bubbles over the full backward pass of 8.53 s. Its forward compute bubbles are approximately $1.30\times$ as long as its backward communication bubbles, consistent with the imbalance observed for LLaMA.

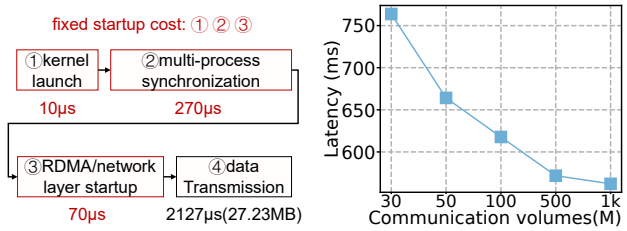
C. Bubble Balancing via Comm. Merging

Observation #2: *Merging multiple communications to reduce the number of transfers in ZeRO-DP can effectively lower communication latency, shifting the bubble distribution from "large compute bubbles in the forward pass and few communication bubbles in the backward pass" to "**reduced compute bubbles in the forward pass and increased communication bubbles in the backward pass**."*

The performance of collective communication is commonly modeled using the α–β communication model [25]–[27], where the communication time is given by

$$t = \alpha + \beta v,$$

with α denoting the communication startup overhead, β the per-byte transmission latency, and v the data volume per communication. In particular, the startup overhead (α) primarily consists of the kernel launch latency, multi-process synchronization latency, and RDMA/network layer startup latency. As shown in Figure 4(a), using `nsys` we measure a kernel launch latency of 10 μs, a multi-process synchronization latency of 270 μs, and an RDMA/network startup latency of 70 μs. For a communication volume of 27.23 MB per call in the ZeRO-DP of LLaMA-7B training, the measured transmission latency is $2,127$ μs, with the startup overhead accounting for 14.12%.



(a) Communication time breakdown. (b) Communication Volume vs. Total Communication Time.

Fig. 4. Since each communication incurs a fixed startup time of 14.12%, increasing the communication volume reduces the total communication time.

Figure 4(b) shows the change of all-gather collective communication time with the amount of aggregated communication volume under ZeRO-DP. Fitting with the α–β model yields $\alpha = 0.3503$ ms and $\beta = 4.07 \times 10^{-8}$ ms/Byte. The goodness of fit, measured by the coefficient of determination R^2 [30], [31] (i.e., the fraction of variance explained by the model, ranging from 0 to 1), is 0.9972. If converted to bandwidth (i.e., β is the transmission time per byte, and its reciprocal is the equivalent bandwidth), it is 24.6 GB/s, which is very close to the RDMA bandwidth in our environment (25 GB/s).

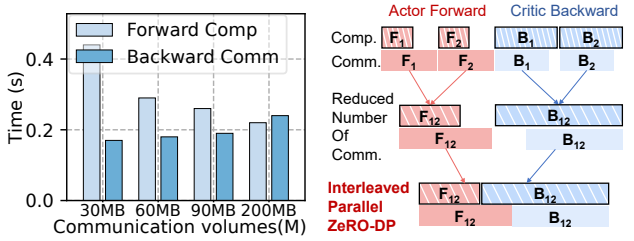
Given the α–β model, with the total communication volume fixed, reducing the number of communication calls reduces the overall communication cost. For example, with LLaMA-7B the default configuration performs about 478 all-gather calls per training iteration; aggregating messages to reduce the calls to 28 decreases the total communication time from 763.79 s to 571.76 s, achieving a $1.33\times$ speedup.

As the number of collective communication operations decreases, the total communication time in both the forward and backward passes is reduced. Consequently, the forward compute bubbles gradually shrink but still remain, while the backward communication bubbles steadily increase. As shown in Figure 5(a), when the per-communication volume in LLaMA-7B increases from the default 30MB to 60MB, 90MB, and 200MB, the number of communications decreases from the default 478 to 253, 181, and 84, respectively. Correspondingly, the forward compute bubbles decrease from 0.44s

Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on October 27,2026 at 12:11:43 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2026.3743842

to 0.29s, 0.26s, and 0.22s, while the backward communication bubbles increase from 0.17s to 0.18s, 0.19s, and 0.24s, leading to a gradual balance between forward compute bubbles and backward communication bubbles. At this point, for the *Actor* and *Critic* with balanced bubbles, the forward pass of one model can be executed in interleaved parallel with the backward pass of the other, thereby filling both compute and communication bubbles (Figure 5(b)).



(a) The relationship between communication volume and bubbles.

(b) Diagram of interleaved parallel ZeRO-DP.

Fig. 5. As communication volume increases, the compute and communication bubbles within a single model approach balance. At this point, interleaved parallel execution of the two models has the opportunity to eliminate bubbles.

D. Implications

In summary, we identify two root causes of ZeRO-DP inefficiency: (i) frequent collective communications, which create large compute bubbles in the forward pass, and (ii) inter-layer variability, which induces small communication bubbles in the backward pass (**Observation #1**). Under fixed hardware resources and a fixed total communication volume, increasing the per-call communication size (i.e., merging communications) reduces overall communication time; consequently, forward compute bubbles and backward communication bubbles tend to balance (**Observation #2**). Therefore, we design a method that combines dynamic communication volume adjustment with interleaved parallel ZeRO-DP to improve training efficiency.

III. Design

In this section, we present the design of *ZeroFuse*.

A. LLMchunk *Abstraction*

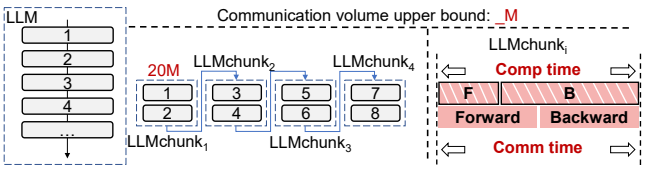


Fig. 6. The LLM is partitioned into multiple *LLMchunks*, where the total parameter size of each *LLMchunk* does not exceed the communication volume.

To derive the per-transfer communication volume that balances bubbles (**Observation #2**), we introduce *LLMchunk* as the basic unit for joint computation–communication design. Let $L = \langle l_1, l_2, \ldots, l_m \rangle$ denote the ordered sequence of transformer blocks in an LLM with m blocks. As shown in Figure 6, we partition L into n contiguous *LLMchunks* $\mathcal{C} = (C_1, \ldots, C_n)$ such that for each i:

$$C_i = \langle l_u, \ldots, l_v \rangle \qquad 1 \le u \le v \le m,$$

and the chunks are pairwise disjoint and cover the model:

$$C_i \cap C_j = \varnothing \ (i \neq j), \qquad \bigcup_{i=1}^{n} C_i = L.$$

Each *LLMchunk* is therefore a consecutive subsequence whose compute time ($\tau_{\text{comp}}(C_i)$) matches its communication time ($\tau_{\text{comm}}(C_i)$), i.e., it is bubble-balanced:

$$\tau_{\text{comp}}(C_i) \approx \tau_{\text{comm}}(C_i).$$

Note that different LLM models may have variations in the computation and communication volumes of each block. Therefore, the number of blocks in each *LLMchunk* may differ. Additionally, computation and communication performance can vary across different hardware environments. Thus, the partitioning of *LLMchunks* depends on both the LLM model and the hardware environment, allowing for different partitioning strategies. When TP or PP is used, *ZeroFuse* operates only on the outer sharded data-parallel dimension, preserving the TP/PP execution model and targeting the additional bubbles introduced by ZeRO-DP collective communication. Specifically, with TP, an *LLMchunk* retains its logical block boundaries, while each TP rank executes its local partition and applies ZeRO-3/FSDP sharding within the corresponding data-parallel group. For an extension to PP, each pipeline stage is represented as a chain of *LLMchunks*, with no chunk crossing a pipeline-stage boundary.

B. *ZeroFuse* Overview

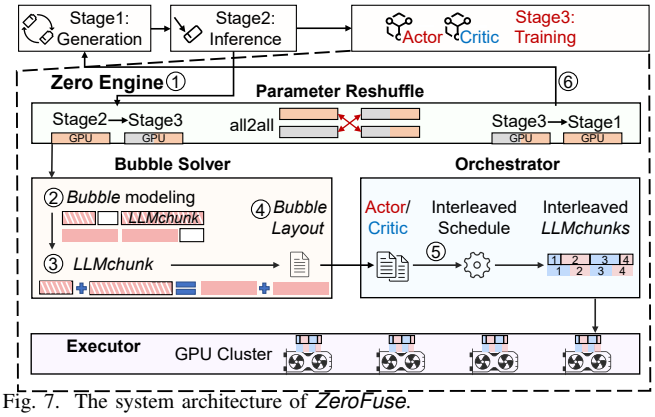


Fig. 7. The system architecture of *ZeroFuse*.

The basic idea of *ZeroFuse* is to orchestrate *LLMchunks* across the *Actor* and *Critic* to achieve truly interleaved-parallel execution, thus accelerating RLHF training. Figure 7 illustrates the system architecture of *ZeroFuse*. The *zero engine* specifically consists of two primary modules: the *bubble solver* and the *orchestrator*. The *bubble solver* performs a formal analysis to determine the *LLMchunk* size (i.e., the sum of total parameter of the *LLMchunk*) for

Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on October 27,2026 at 12:11:43 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2026.3743842

the *Actor* and *Critic* models separately, which balances the forward and backward bubbles of each individual model. Based on this *LLMchunk*, the *bubble solver* generates the bubble layout (i.e., the start and end times of all bubbles) for each model. The *orchestrator* then uses the bubble layouts of the *Actor* and *Critic* to traverse all possible interleaved-parallel execution schedules using a dynamic programming algorithm, retaining only the schedule with the shortest end-to-end time.

The RLHF workflow proceeds as follows: when RLHF begins, samples first pass through Stage 1 (*generation*) and Stage 2 (*inference*), after which the generated trajectories (i.e., including state, action, reward, and value/logits) are forwarded to Stage 3 (*training*). At this point, the *ZeroFuse*'s *zero engine* is activated. It first models the compute time and communication time of each *LLMchunk* ②, then solves for both the *LLMchunk* size ③ and the corresponding bubble layouts of the *Actor* and *Critic* ④. And then the *zero engine* employs the *orchestrator* to generate an interleaved-parallel execution schedule ⑤. According to this schedule, the *Actor* and *Critic* sequentially launch their corresponding computation and communication kernels. In addition, at the Stage 2→Stage 3 and Stage 3→Stage 1 boundaries, *ZeroFuse* applies a parameter reshuffle strategy to efficiently convert model parameters between ZeRO-DP's parameter format and the inference parameter format ①⑥.

C. Bubble Solver

After the *inference* stage completes, we obtain four-tuples: $\langle state, action, reward, value/logits \rangle$. Here, *state* refers to the input prompt sequence, *action* is the generated response sequence, *reward* is an immediate scalar preference score evaluating the complete (state, action) pair, and *value/logits* represents, for each token, both the value estimates provided by the *Critic* and the policy probability distributions generated by the *Actor*. In the *training* stage, we concatenate the *state* and *action* sequences to form the training tokens and use the remaining signals to compute the loss. These sequences constitute trajectories and are then sent to the *bubble solver* for recomputing the *LLMchunk* size and interleaved schedule based on the current trajectories at each training step.

Given the trajectories and the *Actor*/*Critic* models, the *bubble solver* constructs a performance model to estimate computation and communication times, and then partitions each model into n contiguous *LLMchunks*: $\mathcal{C} = (C_1, C_2, \ldots, C_n)$. Since, apart from the input (Embedding) and output (LayerNorm) layers, the main body of an LLM consists of identical transformer blocks, we define each *LLMchunk* as a group of consecutive blocks.

The following derivation considers ZeRO-3 and expresses per-*LLMchunk* computation and communication times as functions of a common parameter size s. Under this setting, by solving for s to balance the compute and communication of a single *LLMchunk*, the computation and communication of the entire LLM are likewise balanced. When TP or PP is used, the model can be extended by adjusting the local computation and parameter-transfer volumes and incorporating the additional TP/PP communication into the execution timeline while preserving its dependencies.

Under ZeRO-DP's compute–communication pipeline, the computation of one *LLMchunk* overlaps with the communication of the next. Consequently, the n *LLMchunks* induce $(n-1)$ computation–communication overlaps in the forward pass: $B^F = \{b_1^F, \ldots, b_{n-1}^F\}$, and $(n-1)$ overlaps in the backward pass: $B^B = \{b_1^B, \ldots, b_{n-1}^B\}$, yielding a total of $2(n-1)$ bubbles. Each bubble b_i is defined by the interval:

$$b_i = [\, \tau_i^{\text{start}},\ \tau_i^{\text{end}} \,],$$

where τ_i^{start} and τ_i^{end} denote its start and end times, respectively. Because the forward and backward passes differ in both computational workload and execution order, we derive the bubble layout for each pass separately.

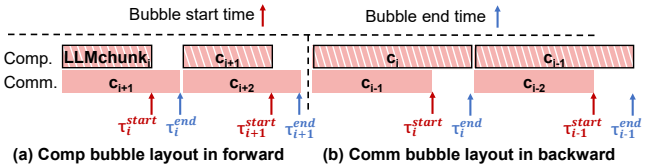


Fig. 8. Bubble layouts.

Bubble modeling in the forward pass: During the forward pass, ZeRO-DP overlaps the computation of the i-th *LLMchunk* with the communication of the $(i+1)$-th *LLMchunk*. Because forward-pass communication typically takes longer than computation (Figure 8(a)), a compute bubble arises whose start time equals the computation end time of i-th *LLMchunk* and whose end time equals the communication end time of $(i+1)$-th *LLMchunk*. Let τ_i^{now} be the start time of chunk i. For $i \in [1, .., n-1]$, the bubble interval is:

$$b_i^F = \left[\ \tau_{i,F}^{\text{now}} + \tau_{i,F}^{\text{comp}},\ \tau_{i,F}^{\text{now}} + \tau_{i+1,F}^{\text{comm}}\ \right],$$

with duration $\Delta_i^F = \max(0,\ \tau_{i+1,F}^{\text{comm}} - \tau_{i,F}^{\text{comp}})$. Specifically, the communication time of the $(i+1)$-th *LLMchunk* can be obtained using the α–β model in Observation #2:

$$\tau_{i+1,F}^{\text{comm}} = \alpha + \beta s,$$

where s is the parameter size of the $(i+1)$-th *LLMchunk*. For computation time of the (i)-th *LLMchunk*, the time can be obtained by dividing the (i)-th *LLMchunk* computation g_i by the GPU throughput f:

$$\tau_{i,F}^{comp} = g_i^F / f,$$

where g_i^F is the operation count (FLOPs) and f is the sustained GPU throughput. Under standard FLOP accounting for transformer forward pass [32], [33], we have $g_i^F = 2ks$, where s is the parameter size of the *LLMchunk*, and k is the number of tokens processed per GPU. We estimate k as $k = \sum_{j=1}^{l} \left(e_j^s + e_j^a\right)/o$, where e_j^s is the prompt length (*state*) and e_j^a is the generated length (*action*) produced in Stage 2 for sample j, l is the number of samples and o is the number of GPUs.

Note that the start time of the first *LLMchunk* is $\tau_{1,F}^{\text{now}} = 0$, and for the remaining *LLMchunks*, τ_F^{now} is the completion time of the previous *LLMchunk*.

Bubble modeling in the backward pass: During the backward pass, ZeRO-DP overlaps the computation of the i-th

Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on October 27,2026 at 12:11:43 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2026.3743842

LLMchunk with the communication of the $(i-1)$-th *LLMchunk*. Because backward-pass computation typically outlasts communication (Figure 8(b)), a communication bubble arises whose start time equals the end of communication for $(i-1)$-th *LLMchunk* and whose end time equals the end of computation for i-th *LLMchunk*.

The backward bubble of the i-th *LLMchunk* b_i^B can be represented as:

$$b_i^B = [\tau_{i,B}^{\text{now}} + \tau_{i-1,B}^{\text{comm}},\ \tau_{i,B}^{\text{now}} + \tau_{i,B}^{\text{comp}}].$$

Since the parameter transfer volume is identical for the forward and backward passes for the same *LLMchunk*, the communication bandwidth is reduced by one-fifth due to the additional gradient transfer [28], and the computation is twice as large for the backward pass (as an additional gradient computation is required), we have $\tau_{i-1,B}^{\text{comm}} = \alpha + 5\beta s/4$ and $\tau_{i,B}^{comp} = g_i^B/f = 4ks/f$.

After obtaining the forward and backward bubble layouts (i.e., $\{b_i^F\}$ and $\{b_i^B\}$), we solve for the *LLMchunk* size s_i by equating the forward and backward computation times with their respective communication times for each *LLMchunk*. We have,

$$\tau_{i,B}^{comp} - \tau_{i-1,B}^{comm} = \tau_{i+1,F}^{comm} - \tau_{i,F}^{comp}.$$

Then, under the condition of balanced computation and communication within a single *LLMchunk*, we solve for s gives:

$$s = \frac{8f\alpha o}{24\sum_{j=1}^{l}\left(e_j^s + e_j^a\right) - 9f\beta o}$$

Finally, the *bubble solver* passes s along with the corresponding bubble layout of *Actor* and *Critic* to the *orchestrator* for scheduling.

Case study: We consider a cluster of 32 NVIDIA A100 GPUs training LLaMA-7B/13B/33B. The prompt and generated lengths are set to $e^s = e^a = 256$, with $l = 608$ samples. The communication model uses a startup overhead $\alpha = 0.3503$ ms and a per-byte latency $\beta = 4.07 \times 10^{-8}$ ms/Byte, and the sustained GPU throughput is $f = 312$ TFLOP/s. Under these settings, the balanced *LLMchunk* size is $s \approx 178.4$M parameters, corresponding to about 9, 5, 1, and 5 transformer blocks for LLaMA-7B/13B/33B/51B, respectively (Figure 9). We select the number of blocks per *LLMchunk* to achieve a relatively small alignment error with integer multiples of s while keeping the weight payload small. This trade-off motivates selecting five blocks per *LLMchunk* for the 51B example. As shown in Figure 10, the modeled and measured bubble times are highly consistent. The model predicts that compute bubbles dominate the forward pass (46.16%, 42.95%, 49.02%), while communication bubbles dominate the backward pass (9.13%, 8.38%, 6.49%). These estimates are slightly lower than the measured values because non-dominant bubbles are merged. Overall, the model accurately captures both bubble types and their layouts.

D. Orchestrator

The *bubble layouts* computed by the *bubble solver* for either the *Actor* and *Critic* achieve a balance between compute and communication bubbles within a single model. However, to enable interleaved parallel execution of both *Actor* and *Critic* models, it is necessary to fill each other's bubbles. Since the two models may differ in size and their *LLMchunk* designs, directly executing *Actor* and *Critic* cannot fully utilize the bubbles. Therefore, we introduce an *orchestrator* to minimize the training time.

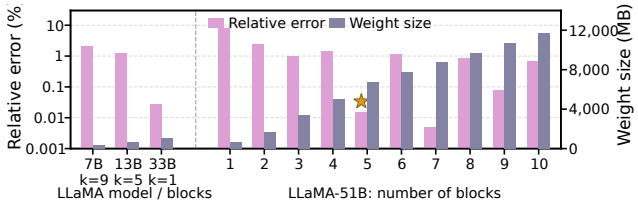


Fig. 9. Relative error/weight size for different numbers of transformer blocks.

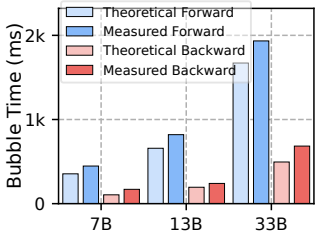


Fig. 10. Theoretical and measured bubble time.

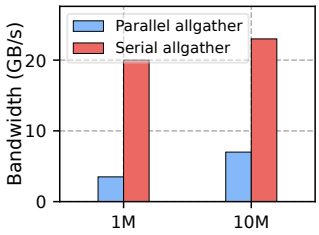


Fig. 11. Parallel and Serial allgather bandwidth.

Under order-preserving, communication-exclusion, and memory-capacity constraints, the *orchestrator* uses the *LLMchunk* partitions and *bubble layouts* to compute—via dynamic programming—an interleaved *Actor*-*Critic* schedule that maximally overlaps complementary bubbles. This staggered schedule raises GPU and network utilization and shortens end-to-end training time.

Notation and inputs: Let the *LLMchunks* in *Actor* and *Critic* be indexed as $\mathcal{C}^A = \{C_1^A, \ldots, C_{a_{\max}}^A\}$, $\mathcal{C}^C = \{C_1^C, \ldots, C_{c_{\max}}^C\}$. For any *LLMchunk* $C \in \mathcal{C}^A \cup \mathcal{C}^C$, let $\lambda(C) \in \{comp, comm\}$ be its bubble type.

Constraints: The *orchestrator* must enforce three hard constraints. First, *communication mutual exclusion*: the *Actor* and *Critic* may not issue collective communications concurrently, as overlapping collectives degrade effective bandwidth (Figure 11) and risk multi-process NCCL deadlocks. We therefore serialize NCCL collectives using a communication lock, ensuring that the two models execute communication operations in an interleaved manner. Second, *memory capacity*: the instantaneous footprint—including static parameters, activations, gradients, optimizer states, and any prefetched parameters—must remain within the GPU's available memory to avoid Out-Of-Memory (OOM) error. Third, *order preservation*: each model's original compute/communication order must be maintained; the *orchestrator* may only place one model's operations into the other model's bubble intervals and must not reorder operations within a model.

Dynamic programming formulation: We define a three-dimensional dynamic programming matrix:

$$t[a][c][u],$$

where $a \in [1, .., a_{max}],\ c \in [1, .., c_{max}],\ u \in \{0, 1, 2, 3\}$.

Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on October 27,2026 at 12:11:43 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2026.3743842

Algorithm 1 Interleaving Scheduling

Require: *Actor*'s Bubble layout L_a, *Critic*'s Bubble layout L_c, *Actor*'s *LLMchunks* $\mathcal{C}^{\mathcal{A}}$, *Critic*'s *LLMchunks* $\mathcal{C}^{\mathcal{C}}$, the number of *Actor* and *Critic LLMchunks* a_{max} and c_{max}, GPU memory capacity m_{gpu};

Ensure: Execution time and interleaving schedule;

1: **function** DP-SCHEDULER(L_a, L_c, a_{max}, c_{max})
2: $t[0\dots a_{max}][0\dots c_{max}][0\dots 3] \leftarrow \infty$
3: $t[0][0][0] \leftarrow 0$
4: **for** $a \in [1, a_{max}]$ **do**
5: **for** $c \in [1, c_{max}]$ **do**
6: **for** $u \in \{0,1,2,3\}$ **do**
7: TRANSITION($a, c, u, m_{gpu}, L_a, L_c$)
8: **return** $\min t[a_{max}][c_{max}][u]$, schedule
9: **function** TRANSITION($a, c, u, m_{gpu}, L_a, L_c$)
10: $\tau_{a+1}, b_c, b_{a+1} \leftarrow L_a, L_c$
11: $\tau_{\text{overlap}}, \tau_{\text{disjoint}} \leftarrow \tau_{a+1}, b_c, b_{a+1}$
12: **if** $(u \in \{2,3\}) \wedge (\lambda(C_c^C) \neq \lambda(C_{a+1}^A))$ **then**
13: $t[a+1][c][u] \leftarrow \min\langle t[a+1][c][u], \tau_{\text{overlap}}\rangle$, $u = 0$
▷ The *Actor* can also choose not to overlap
14: $t[a+1][c][u] \leftarrow \min\langle t[a+1][c][u], \tau_{\text{disjoint}}\rangle$, $u = 3$
15: **else**
16: $t[a+1][c][u] \leftarrow \min\langle t[a+1][c][u], \tau_{\text{disjoint}}\rangle$, $u = 1$ *or* 3
17: Check whether the memory exceeds m_{gpu}

It represents the minimum completion time after finishing the first a *Actor LLMchunks* and the first c *Critic LLMchunks*. The state label u encodes residual bubble availability after scheduling up to (C_a^A, C_c^C), $u = 0$ if the bubble of the C_a^A and C_c^C have already been filled; $u = 1$ if the bubble of C_a^A is unfilled, while that of C_c^C has been filled; $u = 2$ if the bubble of C_a^A has been filled, while that of C_c^C is unfilled; and $u = 3$ if both the bubble of C_a^A and C_c^C are unfilled. We initialize $t[0][0][0] = 0$ and set all other states $t[a][c][u] = +\infty$.

Objective: Minimize the total execution time:

$$\min_{u\in\{0,1,2,3\}} \quad t[a_{max}][c_{max}][u].$$

State transition and matrix update: Suppose that the next *LLMchunk* to be scheduled is C_{a+1}^A. Let the execution times of the C_c^C and C_{a+1}^A be τ_c and τ_{a+1}, respectively, then we have $\tau_c = \tau_c^{comm} + \tau_c^{comp}$ and $\tau_{a+1} = \tau_{a+1}^{comm} + \tau_{a+1}^{comp}$. Let b_c and b_{a+1} denote their corresponding bubble sizes, which can be obtained from the bubble layout.

In the case where the bubble properties of C_c^C and C_{a+1}^A are opposite and the bubble of C_c^C has not been filled, C_c^C and C_{a+1}^A overlap, and we have:

$$t[a+1][c][u] = t[a][c][u] + \tau_{a+1} - \min\{b_c, b_{a+1}\},$$
$$\lambda(C_c^C) \neq \lambda(C_{a+1}^A) \wedge u \in \{2,3\}.$$

After the overlap, the bubbles of C_c^C and C_{a+1}^A are both consumed, and the state is updated to $u = 0$.

In contrast, when the bubble properties of C_c^C and C_{a+1}^A are the same, or the bubble of C_c^C is filled, C_c^C and C_{a+1}^A do not overlap, we have:

$$t[a+1][c][u] = t[a][c][u] + \tau_{a+1},$$
$$\lambda(C_c^C) = \lambda(C_{a+1}^A) \vee u \in \{0,1\}.$$

After scheduling C_{a+1}^A, if C_c^C does not have a bubble beforehand, then only C_{a+1}^A has a bubble after scheduling, and the state is updated to $u = 1$. Otherwise, both C_c^C and C_{a+1}^A have bubbles, and the state is updated to $u = 3$.

Memory check: At each state transition, the *orchestrator* derives the GPU memory usage, including static parameters m_s, activations m_a, gradients m_g, optimizer states m_o, and prefetched parameters m_p (i.e., *LLMchunk* size): $m_{total} = m_s + m_a + m_g + m_o + m_p$. If $m_{total} > m_{gpu}$, the transition is infeasible and will be skipped.

Algorithm 1 presents the details of interleaving scheduling algorithm, which mainly consists of two parts: iterative scheduling and state transition. For the first part, we first initialize the three-dimensional DP matrix (Lines 2–3). Then, by enumerating all possible states (a, c, u) through triple loops, the algorithm invokes TRANSITION to update the state and matrix (Lines 4–7). Finally, it returns the scheduling result with the shortest execution time. For the second part, the algorithm first computes bubble and execution times under the current state (Lines 10–11). It then performs state transitions for both overlapping and non-overlapping cases and updates the matrix (Lines 12–17). Finally, it checks whether memory usage after scheduling fits within GPU capacity; otherwise, the state and matrix are rolled back (Line 18). The function shows transitions for the *Actor*; those for the *Critic* are symmetric.

E. Parameter Reshuffle

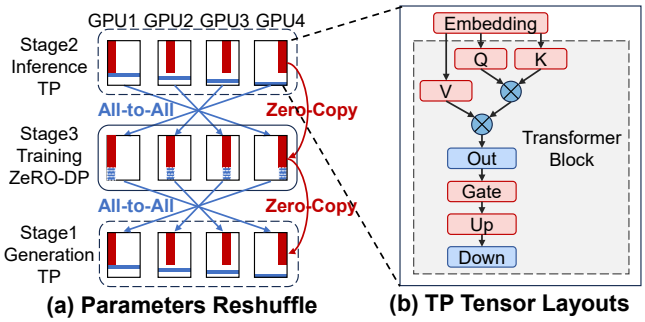


Fig. 12. Reshuffle of column-partitioned ZeRO-DP and partial row-partitioned TP.

The RLHF framework employs tensor parallelism (TP) in both Stage 1 (*generation*) and Stage 2 (*inference*), whereas Stage 3 (*training*) is based on ZeRO-DP, leading to differences in parameter placement layouts across the stages. As illustrated in Figure 12, the TP layout uses a hybrid row-column partitioning scheme, where embedding, QKV matrices, and gate/up-projection layers are column-partitioned, and output and down-projection layers are row-partitioned. In contrast, ZeRO-DP partitions all parameters by columns. This discrepancy necessitates parameter reshuffling when transitioning between Stage 2 and Stage 3, as well as between Stage 3 and Stage 1.

To reduce data movement between Stages 2 and 3 and Stages 3 and 1, we introduce a *layout-aware reshuffle strategy*.

Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on October 27,2026 at 12:11:43 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2026.3743842

During Stage 2 → 3, TP column-partitioned parameters remain unchanged, while row-partitioned parameters are reshuffled into a ZeRO-DP-compatible column layout. Receivers obtain parameter handles (dtype/shape/offset) via P2P communication, enabling zero-copy access to unchanged parameters through GPU memory mapping; the remaining parameters are reordered via all-to-all communication. Stage 3 → 1 reverses this conversion, restoring the original row partitions.

IV. Implementation

ZeroFuse is developed as an extension of the open-source RLHF training framework, *DeepSpeed-Chat* [6], released by Microsoft. It not only implements the core interleaved parallel Zero-DP module but also replaces the inference engine used in Stage 1 and Stage 2 of the existing *DeepSpeed-Chat* with the more efficient *SGLang* engine [34], significantly improving the overall efficiency of the RLHF training process.

We develop the *bubble solver* and *orchestrator* using Python to solve for the *LLMchunk* size and interleaved schedule. Additionally, we implement two execution controllers: the *compute controller* and the *communication controller*. In the *compute controller*, we design an execution queue and provide read/write interfaces to intercept kernel launches from PyTorch, ensuring that the pre-scheduled compute operators of the two models are also executed in a serialized and interleaved manner. During runtime, *DeepSpeed* first executes communication kernels such as `all-gather` through the `fetch_sub_module()` interface, followed by the corresponding computation kernels. The *communication controller* synchronizes with the *bubble solver* using a JSON configuration file and updates the variable `max_params_to_prefetch`, ensuring that each communication step transfers exactly one complete *LLMchunk* of parameters.

To enable cross-process state synchronization between the *Actor* and *Critic*, we integrate Redis as a shared key–value store into the `PPOTrainer` class of *DeepSpeed-Chat*. Redis maintains semaphores to indicate the beginning and end of each RLHF phase. Before advancing to the next phase, each process of the *Actor* and *Critic* checks the corresponding semaphore to ensure that the intermediate data is ready.

For data transfers across GPU process groups, we implement `"tensor_reference_passing"` interfaces to avoid CPU copies of tensors. The sender generates a GPU-side tensor handle and transmits this handle via TCP. The receiver reconstructs the tensor with `handle.rebuild_cuda_tensor()` and copies the data into its local buffer using `tensor.copy()`.

V. Evaluation

A. Experimental Setup

Testbeds: We evaluate *ZeroFuse* on two testbeds. *(1) A100 testbed:* Four servers provide 32 NVIDIA A100-80GB GPUs, with 8 GPUs per server interconnected via NVLink at a total intra-node bandwidth of 600GB/s. RDMA-enabled NICs provide 25GB/s inter-node bandwidth. This testbed is used for the LLaMA experiments. The software stack employs CUDA 12.1, PyTorch 2.2.0, SGLang v0.4.10, and DeepSpeed 0.16.3. *(2) H20 testbed:* Eight NVIDIA H20 GPUs are interconnected via NVLink at a total intra-node bandwidth of 900GB/s. The software stack employs CUDA 12.9, PyTorch 2.7.0, SGLang v0.5.7, and DeepSpeed 0.19.5.

Models and setup: On testbed (1), we train the LLaMA-7B, LLaMA-13B, and LLaMA-33B models [35] in half precision (FP16). We evaluate five *Actor*/*Critic* pairings: 7B/7B, 13B/13B, 33B/33B, 33B/7B, and 33B/13B. For data, we use the standard RLHF dataset *Dahoas/rm-static* [36], with both prompt and generation lengths capped at 256 tokens, a global batch size of 1024, and a mini-batch size of 256. On testbed (2), we evaluate Qwen-4B/4B, Qwen-8B/8B, and LLaMA-7B/7B under the same H20 workload settings. We use dataset DeepMath-103K [37] with a maximum rollout output length of 4096 tokens. The actual output lengths vary across samples.

Baseline systems: We compare *ZeroFuse* with six state-of-the-art RLHF systems:

(1) *HybridFlow* [9] decouples computation and data in the RLHF workflow. It supports *vLLM* as the inference engine and *FSDP* as the training engine. (2) *Puzzle* [38] employs a 3D parallel engine based on *Megatron* [39] for both training and inference. It proposes an interleaved pipeline to fill the idle bubbles. (3) *DeepSpeed-Chat* [6] adopts ZeRO-DP for both training and inference, and leverages techniques such as compute–communication overlap and checkpointing to achieve high throughput and memory efficiency. (4) *RollPacker* [17] employs tail batching to consolidate long-tail responses into a small number of rollout rounds, reducing GPU idle time during generation. (5) *SLIME* [11] supports multiple training methods, with *Sglang* for rollout generation and *Megatron-LM* for training. (6) *OpenRLHF* [10] uses *Ray* to coordinate GPU allocation and data exchange, with *vLLM* for rollout generation and *DeepSpeed* for training.

B. Training Latency

Low latency: *ZeroFuse* achieves average speedups of $1.32\times$ in Stage 3 training.

Figure 13 isolates Stage 3, where communication aggregation and interleaved Actor–Critic execution operate. Reshuffle overhead and auxiliary end-to-end and inference-stage results are reported separately in Figures 14 and 15.

For LLaMA models on 32 A100 GPUs, *ZeroFuse* achieves an average Stage 3 speedup of $1.98\times$ across the evaluated baselines (Figure 13(b)), with average speedups of $1.17\times$, $4.76\times$, and $1.11\times$ over *HybridFlow* , *Puzzle*, and *DeepSpeed-Chat*, respectively. *HybridFlow* and *DeepSpeed-Chat* execute serial ZeRO-DP, leaving residual computation bubbles despite intra-model compute–communication overlap. *ZeroFuse* reduces these bubbles through communication aggregation and interleaved execution of the Actor and Critic. The larger gain over *Puzzle* reflects pipeline overhead and imbalance; *Puzzle* also encounters OOM for the 33B/13B and 33B/33B pairings.

For Qwen on 8 H20 GPUs, *ZeroFuse* achieves an average Stage 3 speedup of $1.36\times$ across all six baselines (Figure 13(a)): $1.35\times$ over *HybridFlow*, $1.39\times$ over *Puzzle*, $1.29\times$ over *DeepSpeed-Chat*, $1.23\times$ over *RollPacker*, $1.35\times$

Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on October 27,2026 at 12:11:43 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2026.3743842

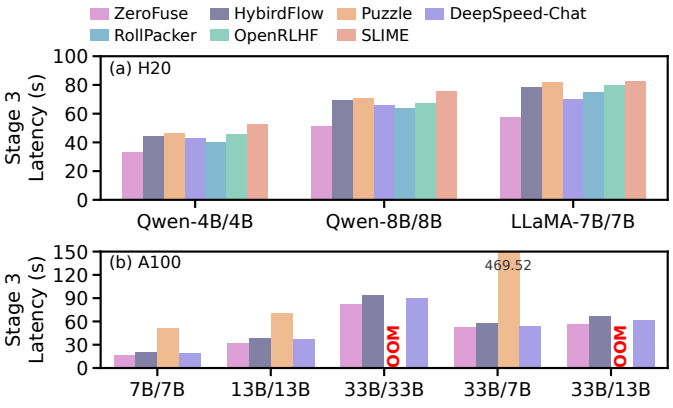


Fig. 13. Stage 3 training latency: (a) Qwen and LLaMA on 8 H20 GPUs; (b) LLaMA on 32 A100 GPUs. *ZeroFuse* uses TP4 + ZeRO-3 on H20 and ZeRO-3 on A100.

over *OpenRLHF*, and 1.54× over *SLIME*. Against the common ZeRO-based baselines, the average speedup increases from 1.14× on LLaMA/A100 to 1.32× on Qwen/H20.

We additionally evaluate LLaMA-7B/7B alongside Qwen on the same 8 H20 GPUs using the H20 workload settings. All six baselines, including *OpenRLHF* and *SLIME*, are included in this LLaMA comparison. For LLaMA-7B/7B, *ZeroFuse* achieves Stage 3 speedups of 1.36×, 1.43×, 1.22×, 1.31×, 1.39×, and 1.44× over *HybridFlow*, *Puzzle*, *DeepSpeed-Chat*, *RollPacker*, *OpenRLHF*, and *SLIME*, respectively.

With *ZeroFuse*, LLaMA-7B/7B takes 57.35 s in Stage 3, compared with 51.00 s for Qwen-8B/8B, or 1.12× the latency despite its smaller parameter count. The main contributor is the attention architecture: LLaMA-7B uses multi-head attention (MHA), whereas Qwen3-8B uses grouped-query attention (GQA), which reduces key/value projection and storage requirements, thereby affecting training latency.

Reshuffle overhead: Reshuffling occurs at the Stage 2 → 3 and Stage 3 → 1 boundaries of each training step, averaging 13.83% of *ZeroFuse*'s end-to-end latency across the original two equally weighted testbeds (Figure 14). For Qwen/H20, reshuffling accounts for 7.04%, 9.07%, 9.07%, and 7.79% of end-to-end latency for *HybridFlow*, *RollPacker*, *OpenRLHF*, and *SLIME*, respectively, comparable to 9.83% for *ZeroFuse*.

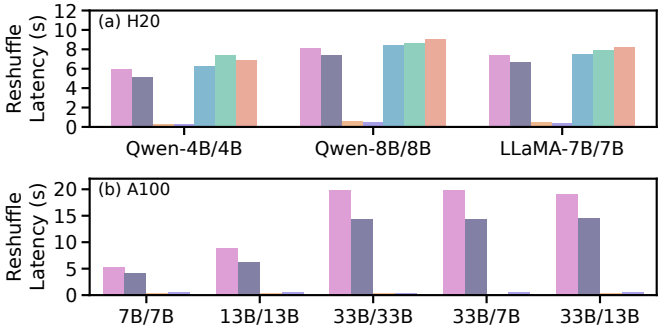


Fig. 14. Per-step reshuffle latency: (a) Qwen and LLaMA on 8 H20 GPUs; (b) LLaMA on 32 A100 GPUs. System colors follow Figure 13.

Auxiliary results: *ZeroFuse* achieves average end-to-end speedups of 1.12× for LLaMA against *HybridFlow*, and 1.26× for Qwen against *HybridFlow*, *RollPacker*, *OpenRLHF*, and *SLIME* (Figure 15). These averages exclude *Puzzle* and *DeepSpeed-Chat*, whose low Stage 1 efficiency disproportionately increases the overall speedup. The end-to-end gains combine Stage 3 acceleration with inference-stage improvements and account for reshuffle overhead. Stage 1 and Stage 2 differences reflect the selected execution engines and parallel configurations.

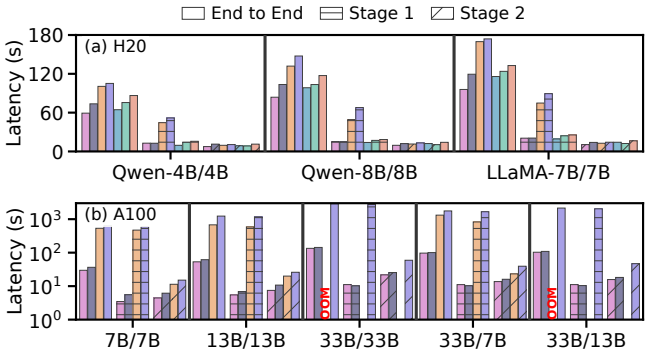


Fig. 15. Auxiliary end-to-end, Stage 1 generation, and Stage 2 inference latency: (a) Qwen and LLaMA on 8 H20 GPUs; (b) LLaMA on 32 A100 GPUs. System colors follow Figure 13.

C. Serial vs Interleaved Parallel ZeRO-DP

Resource efficiency: Interleaved Parallel Zero-DP improves RDMA NIC utilization by 1.32× and reduces computation bubbles by 2.05×. We compare the SM utilization, RDMA NIC bandwidth, and memory usage of *Interleaved Parallel ZeRO-DP* in *ZeroFuse* with the *Serial ZeRO-DP* used by *HybridFlow* and *DeepSpeed-Chat*. We employ `nsys` to trace kernel execution time and SM usage, `Linux RoCE counters` to measure RDMA throughput, and `nvidia-smi` to monitor GPU memory utilization. These metrics are collected during training of the LLaMA-13B/13B models, and we conduct a fine-grained analysis of forward and backward computation timelines within a 1-second window.

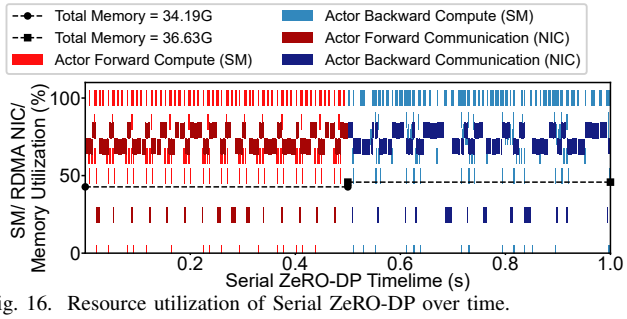


Fig. 16. Resource utilization of Serial ZeRO-DP over time.

Figure 16 illustrates the execution of *Serial ZeRO-DP* (we only present *HybridFlow* since its performance is similar to *DeepSpeed-Chat*). Segments marked with asterisks denote communication kernels. Within the 1-second window, nearly 100 communication kernels are launched, each transferring only small amounts of data, leading to an RDMA NIC throughput of 17.34 GB/s (69.36% utilization). Regular line segments represent compute kernels, and because *Serial ZeRO-DP* cannot eliminate intra-model compute bubbles, the bubble

Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on October 27,2026 at 12:11:43 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2026.3743842

ratio reaches 27.49%. The black dashed line indicates total GPU memory usage, including resident model states and temporary allocations: it reaches 42.73% in the forward phase and 45.78% in the backward phase within this trace.

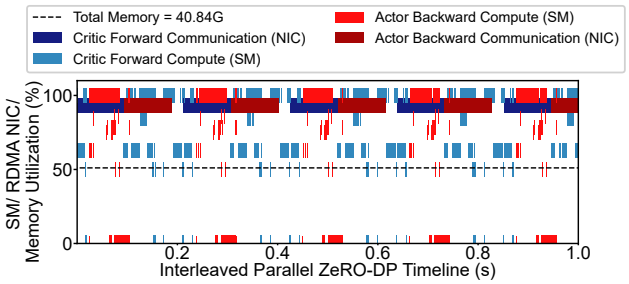


Fig. 17. Resource Utilization of Interleaved parallel Zero-DP over time.

Figure 17 shows the execution of *Interleaved Parallel ZeRO-DP*, which interleaves and overlaps the computation and communication of the *Actor* and *Critic*, reducing the compute bubble ratio to 13.35% (a 2.05× reduction). By performing large-volume communications, it requires only 12 operations per second while raising peak RDMA NIC throughput to 23.05 GB/s (a 1.32× improvement). The early-step trace shows 1.11× the peak memory usage of serial execution, but this is a configuration-specific observation rather than a general memory-overhead bound. The steady-state evaluation below measures how this overhead changes with batch size, sequence length, and parallel configuration.

D. GPU Memory Overhead

We measure steady-state peak GPU memory after the 50th training step, comparing serial forward, serial backward, and interleaved execution for Qwen-4B and Qwen-8B. Figure 18 varies global batch size, maximum sequence length, and parallel configuration. Under ZeRO-3, Qwen-4B uses batch/length pairs of 16/2048, 16/3072, 16/4096, and 24/4096, while Qwen-8B uses 8/2048, 8/3072, 8/4096, and 12/4096. Under TP4 + ZeRO-3, the corresponding pairs are 16/4096 and 24/4096 for Qwen-4B, and 8/4096 and 12/4096 for Qwen-8B.

Interleaved execution uses 1.18×–1.24× the serial backward peak memory under ZeRO-3 and 1.29×–1.33× under TP4 + ZeRO-3. The higher overhead under TP4 + ZeRO-3 coincides with a smaller gap between serial forward and backward peak memory. In addition, on H20 the compute throughput is relatively weaker while per-GPU memory is larger, so interleaved execution can utilize memory that remains idle when kernels become compute-bound.

E. Performance Analysis Breakdown

We analyze three variants: (1) *ZeroFuse* w/o Schedule, which removes interleaved scheduling; (2) *ZeroFuse* w/o LLMchunk, which retains interleaving but communicates at individual-layer granularity; and (3) *ZeroFuse* w/o Reshuffle, which disables switching between training and inference engines.

We first isolate the two Stage 3 components in Figure 19. *ZeroFuse* w/o Schedule executes serial ZeRO-DP, increasing

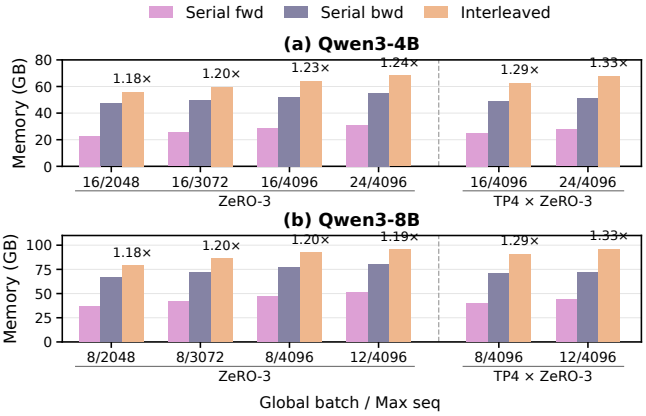


Fig. 18. Steady-state GPU memory usage for Qwen-4B and Qwen-8B under ZeRO-3 and TP4 + ZeRO-3. Labels above the bars report interleaved peak memory relative to serial backward peak memory; the horizontal axis gives global batch size and maximum sequence length.

Stage 3 latency by 1.10×–1.21× (Figure 19(a)). *ZeroFuse* w/o LLMchunk retains interleaving but communicates at individual-layer granularity, increasing Stage 3 latency by 1.14×–1.22× (Figure 19(b)). These comparisons distinguish the contributions of interleaving and communication aggregation from inference-engine effects.

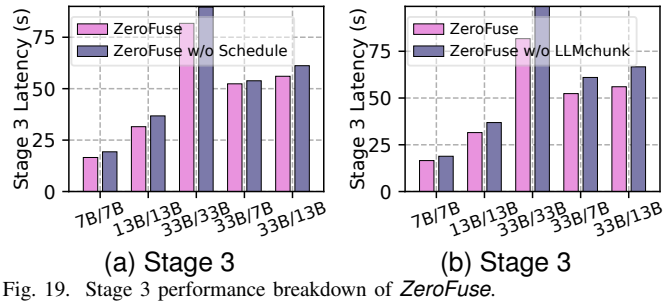


Fig. 19. Stage 3 performance breakdown of *ZeroFuse*.

This variant runs all three stages on ZeRO-DP instead of using *SGLang* for generation and inference, increasing Stage 1 latency by 147.76×–244.89× (Figure 20(a)) and Stage 2 latency by 1.33×–1.67× (Figure 20(b)) due to a less efficient parallelism configuration.

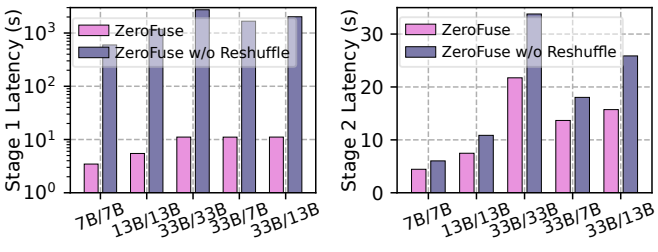


Fig. 20. Auxiliary inference-stage comparison of *ZeroFuse* and *ZeroFuse* w/o Reshuffle.

F. Large Scale Simulation

Training efficiency: *ZeroFuse* reduces the modeled Stage 3 training latency by approximately 14.8% compared with serial

Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on October 27,2026 at 12:11:43 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2026.3743842

ZeRO-DP.

To evaluate the training efficiency of *ZeroFuse* compared with *HybridFlow* and *DeepSpeed-Chat* under large-scale LLM training settings, we simulate the training of LLaMA-3.1-405B using theoretical TFLOPs-based estimation. The hardware configuration assumes 1,024 NVIDIA A100-80GB GPUs with an inter-node bandwidth of 200 Gbps. For large-scale communication, we assume that NCCL AllGather achieves 25% of peak bandwidth at this scale [40], [41]. The dataset consists of 25 million synthetic examples [42], each with an input and output length of 4096 tokens.

Stage 3 trains both the *Actor* and *Critic* over the dataset. Let C_f and C_b denote the per-layer forward and backward computation times, and A_f and A_b the corresponding communication times. We estimate these quantities as $C_f = F_{\text{layer}}/P_{\text{GPU}}$, $C_b \approx 2C_f$, $A_f = V_f/B_{\text{eff}}$, and $A_b = V_b/B_{\text{eff}}$, where F_{layer} is the forward FLOP count per layer, P_{GPU} is the assumed GPU compute throughput, V_f and V_b are the communication volumes, and B_{eff} is the effective communication bandwidth. With compute–communication overlap, serial ZeRO-DP has per-layer times $T_f = \max(C_f, A_f)$ and $T_b = \max(C_b, A_b)$. Thus, the total Stage 3 training time for *HybridFlow* and *DeepSpeed-Chat* is $T_{\text{serial}} = \frac{N}{KB}(T_f + T_b)L\,2$, where N is the number of examples, K is the GPU count, B is the per-GPU training batch size, and L is the number of transformer layers; the factor of two accounts for the *Actor* and *Critic*.

Under the modeled configuration, forward execution is communication-dominated, whereas backward execution is computation-dominated. By interleaving one model's forward pass with the other model's backward pass, *ZeroFuse* overlaps their complementary bubbles. Assuming that the modeled communication can be hidden by the interleaved computation, the effective per-layer time becomes $T_{\text{fused}} = C_f + C_b$, yielding a total Stage 3 training time of $T_{\text{ZeroFuse}} = \frac{N}{KB} T_{\text{fused}} L\,2$. The relative latency reduction is therefore $\Delta_{\text{Stage 3}} = (1 - T_{\text{ZeroFuse}}/T_{\text{serial}}) \times 100\% \approx 14.8\%$.

Figure 21 presents the three-stage comparison for context. Stages 1 and 2 primarily reflect the effects of the selected inference engines and optimized parallelism configurations and are not the optimization focus of this simulation; interleaved parallel execution reduces Stage 3 latency by 14.8%.

G. Overhead

Low overhead: *ZeroFuse* incurs less than 1 second of overhead per training batch, which is fully hidden by the reshuffle time. The primary overhead of *ZeroFuse* comes from the dynamic programming performed by the *orchestrator* at the end of stage 2. This overhead increases with model size. As shown in Figure22, the orchestration costs are from 0.18s to 0.8820s. However, since the reshuffle times between stage 2 and stage 3 range from 2.64s to 9.92s (Section V-B), the scheduling overhead is completely hidden by reshuffle and has no impact on end-to-end latency.

VI. Related Work

RLHF acceleration targets either generation or training. Training-stage approaches include (1) pipeline merging and (2) ZeRO-DP-based intra-model computation–communication overlap.

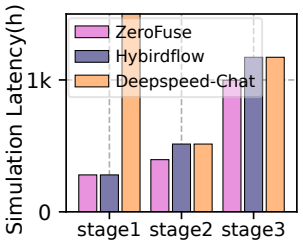


Fig. 21. Latency of simulation.

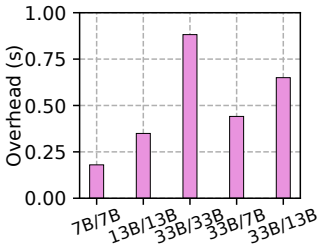


Fig. 22. The DP execution time.

(1) Pipeline merging: Puzzle [38] and Megatron-LM [39] use interleaved pipelines, while Chimera [43] and RLHFuse [44] use bidirectional pipelines to reduce warm-up and cool-down bubbles. However, imbalanced partitions leave substantial bubbles, and ZeRO-DP's different communication pattern may limit these techniques' effectiveness.

(2) Intra-model computation–communication overlap: DeepSpeed-Chat [6] and HybridFlow [9] execute Actor and Critic serially with ZeRO-DP across all GPUs, while OpenRLHF [10] runs them independently on separate GPU groups. These approaches rely on intra-model overlap, which cannot fully hide compute bubbles under communication bottlenecks.

For *RLHF generation*, existing approaches mitigate GPU underutilization through: (3) **speculative inference**, using idle GPUs for speculative-model training (TLT [12]) or multiple static speculative models (SpecActor [16]), or compressed suffix trees for synchronization (Seer [15]); (4) **redundant inference**, separating long- and short-tail samples (RollPacker [17]) or reordering samples by response length (SortedRL [18]); and (5) **long-tail scheduling**, assigning dedicated GPUs to predicted long-tail samples (StreamRL [45]). ReaL [14] also selects parallelization strategies through simulation. These generation-stage optimizations are orthogonal to our work.

VII. Conclusions

In this paper, we present *ZeroFuse*, an RLHF system that enables truly interleaved *Actor–Critic* parallel execution under ZeRO-DP. We introduce a new abstraction, *LLMchunk*, as the basic unit of computation–communication in LLM training. The *LLMchunk* size is determined by solving a performance model that balances computation and communication, and we employ a dynamic programming approach to interleave the *Actor* and *Critic LLMchunks*. Furthermore, we adopt a parameter reshuffle strategy to reduce communication overhead during the switch between training and inference engines. *ZeroFuse* achieves up to $1.32\times$ speedup in *training* stage compared to three state-of-the-art systems, substantially improving GPU computation and bandwidth utilization.

References

[1] P. F. Christiano, J. Leike *et al.*, "Deep reinforcement learning from human preferences," in *Advances in Neural Information Processing Systems*, vol. 30, 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf

Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on October 27,2026 at 12:11:43 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2026.3743842

[2] L. Ouyang, J. Wu *et al.*, "Training language models to follow instructions with human feedback," pp. 27 730–27 744, 2022. [Online]. Available: https://doi.org/10.52202/068431-2011

[3] Y. Bai, S. Kadavath *et al.*, "Constitutional ai: Harmlessness from ai feedback," 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2212.08073

[4] "Introducing Gemini: our largest and most capable AI model," https://blog.google/technology/ai/google-gemini-ai/, 2025.

[5] H. Touvron, L. Martin *et al.*, "Llama 2: Open foundation and fine-tuned chat models," 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2307.09288

[6] Z. Yao, R. Y. Aminabadi *et al.*, "Deepspeed-chat: Easy, fast and affordable rlhf training of chatgpt-like models at all scales," 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2308.01320

[7] "NVIDIA-NeMo/RL," https://github.com/NVIDIA-NeMo/RL, 2026.

[8] W. Wang, S. Xiong, and a. et, "Reinforcement learning optimization for large-scale learning: An efficient and user-friendly scaling library," 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2506.06122

[9] G. Sheng, C. Zhang *et al.*, "Hybridflow: A flexible and efficient rlhf framework," in *Proceedings of the Twentieth European Conference on Computer Systems*, ser. EuroSys '25, 2025, p. 1279–1297. [Online]. Available: https://doi.org/10.1145/3689031.3696075

[10] J. Hu, X. Wu, *et al.*, "Openrlhf: An easy-to-use, scalable and high-performance rlhf framework," 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2405.11143

[11] Z. Zhu, C. Xie *et al.*, "slime: An llm post-training framework for rl scaling," https://github.com/THUDM/slime, 2025.

[12] Q. Hu, S. Yang, and a. et, "Taming the long-tail: Efficient reasoning rl training with adaptive drafter," p. 1933–1948, 2026. [Online]. Available: https://doi.org/10.1145/3779212.3790231

[13] W. Fu, J. Gao *et al.*, "Areal: A large-scale asynchronous reinforcement learning system for language reasoning," pp. 36 256–36 282, 2025. [Online]. Available: https://doi.org/10.52202/085713-1218

[14] Z. Mei, W. Fu *et al.*, "Real: Efficient rlhf training of large language models with parameter reallocation," 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2406.14088

[15] R. Qin, W. He, and a. et, "Seer: Online context learning for fast synchronous llm reinforcement learning," 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2511.14617

[16] R. Cheng, K. Zhou *et al.*, "Fast llm post-training via decoupled and fastest-of-n speculation," 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2511.16193

[17] W. Gao, Y. Zhao *et al.*, "RollPacker: Taming Long-Tail rollouts for RL Post-Training with tail batching," in *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*. Renton, WA: USENIX Association, May 2026, pp. 849–866. [Online]. Available: https://www.usenix.org/conference/nsdi26/presentation/gao-wei

[18] Y. Zhang, H. Jiang *et al.*, "SortedRL: Accelerating RL training for LLMs through online length-aware scheduling," in *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models*, 2025. [Online]. Available: https://openreview.net/forum?id=YoV9lIZ827

[19] S. Rajbhandari, J. Rasley *et al.*, "Zero: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–16. [Online]. Available: https://doi.org/10.1109/SC41405.2020.00024

[20] S. Zhang, S. Roller *et al.*, "Opt: Open pre-trained transformer language models," 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2205.01068

[21] A. Grattafiori, A. Dubey, and A. J. et al., "The llama 3 herd of models," 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2407.21783

[22] A. Chowdhery, S. Narang *et al.*, "Palm: Scaling language modeling with pathways," 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2204.02311

[23] S. Smith, M. Patwary *et al.*, "Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model," 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2201.11990

[24] A. Q. Jiang, A. Sablayrolles *et al.*, "Mistral 7b," 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2310.06825

[25] H. Kim, J. Ryu, and J. Lee, "Tccl: Discovering better communication paths for pcie gpu clusters," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, p. 999–1015. [Online]. Available: https://doi.org/10.1145/3620666.3651362

[26] A. Gangidi, R. Miao *et al.*, "Rdma over ethernet for distributed training at meta scale," in *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024, p. 57–70. [Online]. Available: https://doi.org/10.1145/3651890.3672233

[27] S. K. Monga, S. Kashyap, and C. Min, "Birds of a feather flock together: Scaling rdma rpcs with flock," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, p. 212–227. [Online]. Available: https://doi.org/10.1145/3477132.3483576

[28] G. Wang, H. Qin *et al.*, "Zero++: Extremely efficient collective communication for giant model training," 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2306.10209

[29] "NVIDIA Nsight Systems," https://developer.nvidia.com/nsight-systems/, 2024.

[30] D. R. Rao, "Machine learning techniques on multidimensional curve fitting data based on r-square and chi-square methods," *International Journal of Electrical and Computer Engineering*, vol. 6, no. 3, p. 974, 2016. [Online]. Available: https://doi.org/10.11591/ijece.v6i3.9155

[31] C. Onyutha, "From r-squared to coefficient of model accuracy for assessing" goodness-of-fits"," *Geoscientific Model Development Discussions*, vol. 2020, pp. 1–25, 2020. [Online]. Available: https://doi.org/10.5194/gmd-2020-51

[32] R. Pope, S. Douglas *et al.*, "Efficiently scaling transformer inference," in *Proceedings of Machine Learning and Systems*, vol. 5. Curan, 2023, pp. 606–624. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2023/file/c4be71ab8d24cdfb45e3d06dbfca2780-Paper-mlsys2023.pdf

[33] J. Hoffmann, S. Borgeaud *et al.*, "Training compute-optimal large language models," 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2203.15556

[34] L. Zheng, L. Yin *et al.*, "Sglang: efficient execution of structured language model programs," in *Proceedings of the 38th International Conference on Neural Information Processing Systems*, 2025. [Online]. Available: https://doi.org/10.52202/079017-2000

[35] H. Touvron, T. Lavril *et al.*, "Llama: Open and efficient foundation language models," 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2302.13971

[36] "Datasets: Dahoas/rm-static," https://huggingface.co/datasets/Dahoas/rm-static, 2025.

[37] Z. He, T. Liang *et al.*, "Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning," 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2504.11456

[38] K. Lei, Y. Jin *et al.*, "PUZZLE: Efficiently aligning large language models through Light-Weight context switch," in *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. USENIX Association, Jul. 2024, pp. 127–140. [Online]. Available: https://www.usenix.org/conference/atc24/presentation/lei

[39] D. Narayanan, M. Shoeybi *et al.*, "Efficient large-scale language model training on gpu clusters using megatron-lm," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476209

[40] "Massively Scale Your Deep Learning Training with NCCL 2.4," https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4, 2025.

[41] J. Fernandez, L. Wehrstedt, L. Shamis, M. Elhoushi, K. Saladi, Y. Bisk, E. Strubell, and J. Kahn, "Hardware scaling trends and diminishing returns in large-scale distributed training," 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2411.13055

[42] "meta-llama/Llama-3.1-405B," https://huggingface.co/meta-llama/Llama-3.1-405B, 2025.

[43] S. Li and T. Hoefler, "Chimera: efficiently training large-scale neural networks with bidirectional pipelines," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476145

[44] Y. Zhong, Z. Zhang, B. Wu, S. Liu, Y. Chen, C. Wan, H. Hu, L. Xia, R. Ming, Y. Zhu, and X. Jin, "Optimizing RLHF training for large language models with stage fusion," in *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX Association, Apr. 2025, pp. 489–503. [Online]. Available: https://www.usenix.org/conference/nsdi25/presentation/zhong

[45] Y. Zhong, Z. Zhang, X. Song, H. Hu, C. Jin, B. Wu, N. Chen, Y. Chen, Y. Zhou, C. Wan, H. Zhou, Y. Jiang, Y. Zhu, and D. Jiang, "Streamrl: Scalable, heterogeneous, and elastic rl for llms with disaggregated stream generation," 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2504.15930

Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on October 27,2026 at 12:11:43 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2026.3743842

VIII. Summary of Symbols

Notation	Meaning
C_i, n	Chunk i and total chunk count.
s	Parameters per chunk.
e_j^s, e_j^a	Prompt/response token counts.
l, o, k	Samples, GPUs, and tokens per GPU.
α, β	Startup time and per-byte latency.
f	Per-GPU compute throughput.
g_i^F, g_i^B	Forward/backward FLOPs of chunk i.
τ^{comp}, τ^{comm}	Computation/communication durations.
b_i^F, b_i^B	Forward/backward bubble intervals.
Δ_i^F	Forward compute-bubble duration.
$\mathcal{C}^A$, $\mathcal{C}^C$	Actor/Critic chunk collections.
L_a, L_c	Actor/Critic bubble layouts.
a, c	Completed Actor/Critic chunk counts.
$a_{\max}$, $c_{\max}$	Total Actor/Critic chunk counts.
$\lambda(C)$	Bubble type of chunk C.
u	Residual bubble-availability state.
$t[a][c][u]$	Minimum time for scheduling state (a, c, u).
m_{total}, m_{gpu}	Per-GPU memory usage/capacity.



Wenhao Huang Wenhao Huang received the bachelor's degree from Zhengzhou University, China, in 2020, and the master's degree from Tianjin University, China, in 2023. He is currently pursuing the Ph.D. degree with the College of Intelligence and Computing, Tianjin University, China. His research interests include serverless computing and AI infrastructure.



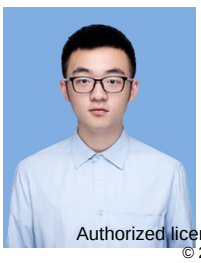
Yuhao Zhang Yuhao Zhang received the Ph.D. degree from the School of Computer Science and Technology, Shandong University, Qingdao, China, in 2022. He is currently an Associate Researcher with the College of Intelligence and Computing, Tianjin University, China. His research interests include storage systems and AI systems.



Laiping Zhao Laiping Zhao received the B.S. and M.S. degrees from the Dalian University of Technology, China, in 2007 and 2009, respectively, and the Ph.D. degree from the Department of Informatics, Kyushu University, Japan, in 2012. He is a Professor with the College of Intelligence and Computing, Tianjin University. His research interests include intelligent computing systems and operating systems.



Yiming Li Yiming Li received the bachelor's and master's degrees from Tianjin University, China, in 2017 and 2019, respectively. He is currently pursuing the Ph.D. degree with the College of Intelligence and Computing, Tianjin University, China. His research interests include cloud computing.



Yichi Chen Yichi Chen received the bachelor's degree from Tianjin University, China, in 2024. He is currently pursuing the master's degree with the College of Intelligence and Computing, Tianjin University, China. His research interests include serverless computing and AI agent runtime.



Yanjie Wang Yanjie Wang received the bachelor's degree from Tianjin University, China, in 2024. He is currently pursuing the master's degree with the College of Intelligence and Computing, Tianjin University, China. His research interests include AI infrastructure.



Fupeng Li Fupeng Li received the bachelor's degree from Tianjin University, China, in 2025. He is currently pursuing the master's degree with the College of Intelligence and Computing, Tianjin University, China. His research interests include AI infrastructure.



Yihan Wang Yihan Wang received the bachelor's degree from Anhui University, China, in 2024. He is currently pursuing the master's degree with the College of Intelligence and Computing, Tianjin University, China. His research interests include serverless computing and LLM inference.



Zhihang Tang Zhihang Tang received the bachelor's degree from Shanghai University of Electric Power, China, in 2018, and the master's degree from Zhejiang University, China, in 2021. He is currently pursuing the Ph.D. degree with the College of Intelligence and Computing, Tianjin University, China. His research interests include distributed computing, edge computing, and LLM systems.



Deze Zeng Deze Zeng received the bachelor's degree from the School of Computer Science and Technology, Huazhong University of Science and Technology, China, in 2007, and the M.S. and Ph.D. degrees in computer science from the University of Aizu, Aizu-Wakamatsu, Japan, in 2009 and 2013, respectively. He is currently a Full Professor with the School of Computer Science, China University of Geosciences, Wuhan, China. His research interests include edge computing, cloud computing, and future networking technologies.



Keqiu Li (Fellow, IEEE) received the bachelor's and master's degrees from the Department of Applied Mathematics, Dalian University of Technology, Dalian, China, in 1994 and 1997, respectively, and the Ph.D. degree from the Graduate School of Information Science, Japan Advanced Institute of Science and Technology, Nomi, Japan, in 2005. He is currently a Full Professor, the Dean of the College of Intelligence and Computing, Tianjin University, China. He is the recipient of National Science Foundation for Distinguished Young Scholars of China. He keeps working on the topics of mobile computing, datacenter, and cloud computing. He has more than 150 papers published on prestigious journals or conferences such as the IEEE/ACM Transactions on Networking, IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, IEEE Transactions on Mobile Computing, INFOCOM, and ICNP.

Authorized licensed use limited to: TIANJIN UNIVERSITY. Downloaded on October 27,2026 at 12:11:43 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.