Puzzle: Efficiently Aligning Large Language Models through Light-Weight Context Switch

Kinman Lei, Yuyang Jin, Mingshu Zhai, Kezhao Huang, Haoxing Ye, and Jidong Zhai, *Tsinghua University*

https://www.usenix.org/conference/atc24/presentation/lei

978-1-939133-41-0

PUZZLE: Efficiently Aligning Large Language Models through Light-Weight Context Switch

Kinman Lei Yuyang Jin Mingshu Zhai Kezhao Huang Haoxing Ye Jidong Zhai

Tsinghua University

Abstract

Aligning Large Language Models (LLMs) is currently the primary method to ensure AI systems operate in an ethically responsible and socially beneficial manner. Its paradigm differs significantly from standard pre-training or fine-tuning processes, involving multiple *models and workloads* (context), and necessitates frequently switching execution, introducing significant overhead, such as parameter updates and data transfer, which poses a critical challenge: efficiently switching between different models and workloads.

To address these challenges, we introduce PUZZLE, an efficient system for LLM alignment. We explore model orchestration as well as light-weight and smooth workload switching in aligning LLMs by considering the similarity between different workloads. Specifically, PUZZLE adopts a two-dimensional approach for efficient switching, focusing on both intra- and inter-stage switching. Within each stage, switching costs are minimized by exploring model affinities and overlapping computation via time-sharing. Furthermore, a similarity-oriented strategy is employed to find the optimal inter-stage switch plan with the minimum communication cost. We evaluate PUZZLE on various clusters with up to 32 GPUs. Results show that PUZZLE achieves up to 2.12× speedup compared with the state-of-the-art RLHF training system DeepSpeed-Chat.

1 Introduction

Aligning Large Language Models (LLMs) has recently attracted significant interest, emphasizing the importance of these advanced AI systems operating in an ethically responsible and socially beneficial manner, as well as their enhanced accuracy. Notably, recent developments in InstructGPT [20] or ChatGPT like [18] models, including OpenAI's GPT-4 [19], Google's Bard [6], and Meta's LLaMA 2-Chat [25], have highlighted the importance of alignment in these models. The primary method for achieving this alignment is the Reinforcement Learning from Human Feedback (RLHF) approach.

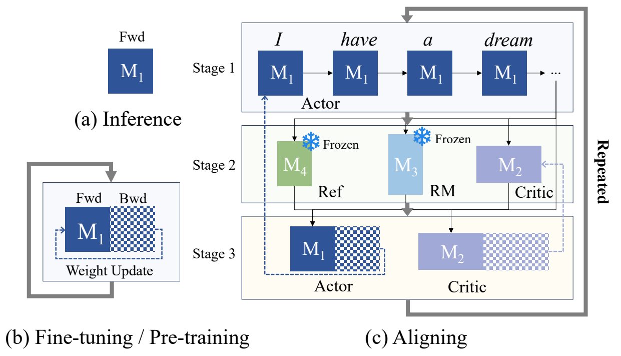

Figure 1: An illustration depicting (a) Inference, (b) Full Parameter Fine-tuning / Pre-training, and (c) Aligning Large Language Models using the PPO Algorithm.

Within the scope of this approach, Proximal Policy Optimization (PPO) emerges as a widely recognized standard algorithm, which significantly improves the reliability of LLMs in many deep learning tasks, including natural language processing [9, 18], text-to-image [11, 27].

The paradigms of aligning LLMs differ significantly from the standard full-parameter fine-tuning or pre-training processes that existing deep learning systems are designed for. As shown in Figure 1, we take the standard algorithm PPO in aligning LLMs as an example to illustrate the differences between the paradigms of inference, typical fine-tuning (or pre-training), and aligning. In fine-tuning, as illustrated in Figure 1(b), only a single model is required. However, in aligning, as depicted in Figure 1(c), multiple models are required, including *Actor*, *Critic*, *Reference*, and *Reward*. Based on the workload characteristics of models (inference or fine-tuning) and the data dependence between them, the paradigms can be divided into several *stages*. Data dependence between different models only exists between stages. Additionally, in the subsequent discussion, we refer to the workload of model and the model itself as *context*.

We observe that the current standard approach of LLM alignment has the following characteristics:

Heterogeneous models and workloads. Alignment methods such as PPO involve collaborative participation from various models, resulting in heterogeneity. The heterogeneity arises from two aspects, model structures and workloads. Firstly, there is significant variation in the number and structure of parameters across different models. Secondly, the alignment process also introduces diverse workloads, such as decoding, inference, and training. Consequently, these heterogeneous contexts make alignment process more complex than standard fine-tuning.

Frequent context switching. The entire process of aligning involves executing different workloads of different models (contexts) in a cyclic manner, resulting in frequent context switching. Context switching occurs not only between stages, but also within stages, for example when multiple models within a stage are executed in a sequential mode. Such frequent switching introduces significant overhead, including model reloading, parameter updates, data transfer, and more. For instance, DeepSpeed-Chat incurs an overhead of 12.43% for gathering updated parameters.

Currently, a large number of deep learning systems have been presented to tackle different complex scenarios such as inference and training. However, these systems have the following two limitations.

Exclusively concentrating on an individual model. Existing works [10, 16, 17, 34] concentrate on optimizing operators, compute graphs, and parallel execution plans for individual models. However, these works neglect scenarios that involve the orchestrated training of multiple models, ignoring the opportunities for fine-grained cross-model optimizations.

Overlooking diverse workloads in models. Existing inference [10, 13, 31] and training systems [28, 29] maintain fixed workloads for their models and do not necessitate configuration changes, such as parallel execution plan, during execution. In these systems, frequent context switching required in aligning processes is not sufficiently considered, resulting in significant and unacceptable costs.

Therefore, we conclude *efficient switching between heterogeneous contexts* as a critical challenge in the LLM alignment, which is still an open research problem.

To address these challenges, we propose PUZZLE, an efficient LLM alignment system that treats model context as a first-class citizen. We investigate the model orchestration in aligning LLMs, taking into account the heterogeneous contexts involved. The similarity between heterogeneous contexts is utilized to achieve light-weight and smooth context switching. Specifically, to efficiently handle the heterogeneous context properties in the aligning process, PUZZLE incorporates a two-dimensional approach, considering both intra-stage and inter-stage context switching. Within each stage, we minimize switching costs by exploring the affinities among different contexts and overlapping computation through time-sharing plans. Furthermore, to reduce switching overhead across stages, we develop a similarity-oriented plan to find the optimal inter-stage context switch plan with minimum communication cost. We evaluate PUZZLE on different clusters with up to 32 GPUs. Results show that PUZZLE achieves up to 2.12× speedup in end-to-end training compared with the state-of-the-art RLHF training system DeepSpeed-Chat [30].

In summary, we make the following contributions:

- We abstract the key concepts of context from the alignment problem and identify performance optimization opportunities by focusing on the context switches.
- We propose a time-sharing strategy to explore the affinities among different contexts within a stage, and integrate it into a hybrid plan.
- We develop a similarity-oriented plan to find the optimal inter-stage context switch plan with minimum communication cost.
- We build PUZZLE, an implementation of the above techniques into an end-to-end training system, and achieve up to 2.12× speedup over state-of-the-art systems.

2 Background and Motivation

2.1 Background

LLM Alignment. LLM alignment, pivotal in the advancement of conversational AI models like ChatGPT [18], LLaMA2 [25], and GPT-4 [19], focuses on harnessing their expansive knowledge and capabilities to produce specific, user-oriented responses and behaviors, and to enhance accuracy. This concept is critical for ensuring these models exemplify attributes of safety, efficacy, and manageability. It has been observed that LLMs, regardless of their size, can produce outputs that are misleading, harmful, or not particularly useful to the user. It's essential to recognize that simply expanding the size of these language models doesn't automatically translate to an improved alignment with user intentions. Existing human preference alignment methods can be broadly categorized into three major categories: reinforcement learning [20, 23], contrastive learning [21, 33], and hindsight instruction relabeling [14, 32].

Among these methods, the reinforcement learning approach stands out as the primary method for achieving this alignment. In this work, we focus on the PPO algorithm, recognized as the most efficient method used in RLHF [4], which demands high efficiency in alignment.

Reinforcement Learning from Human Feedback. In RLHF, the process encompasses three primary steps: collecting human feedback data, training a reward model using the collected human feedback data, and adopting RL optimization

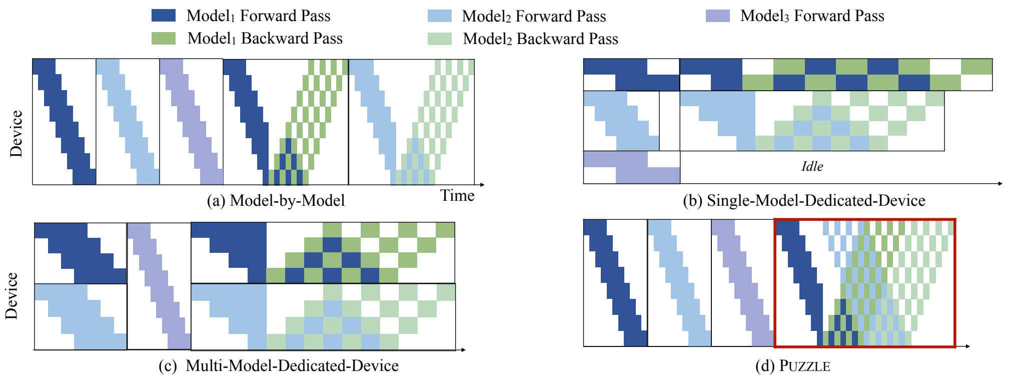

Figure 2: Diverse model placement and execution plans for three different models during part of the alignment process, including forward and backward pass, only pipeline parallelism are shown. (a) MBM: Each model is allocated the same resource setting and executed sequentially. (b) SMDD: Dedicated devices are allocated to each model, enabling their concurrent execution. (c) MMDD: Combines MBM and SMDD strategies but misses opportunities for optimization between models. (d) PUZZLE implements fine-grained time-sharing between models without dependency (The red box).

strategies on LLM [20]. This article primarily focuses on the third step.

Among all RL alignment methods, Proximal Policy Optimization (PPO) is the most popular choice at the third step. Many prominent models adopt the PPO algorithm either as the sole alignment method or in combination with others. An example of the PPO optimization policy step, as implemented by InstructGPT, is illustrated in Figure 1(c). This process involves multiple models, including `Actor`, `Ref`, `Critic`, and `RM`, as shown in the figure. Technically, a reward model (`RM`) is trained on human-labeled data to approximate human preferences. The `Actor` model serves as the policy model, while the `Critic` model functions as the value model in the PPO learning policy. The reference model (`Ref`) is employed for KL regularization. The `Actor` and `Ref` models are generally similar, with `Actor` being trainable while the parameters of `Ref` remain frozen. Similarly, `Critic` and `RM` share the same structure, with `Critic` being trainable.

The entire alignment process can be abstractly described as follows: it begins with the `Actor` generating responses based on given prompts. These responses are then passed to the `Ref`, `RM`, and `Critic` to create a batch of learnable experiences. These experiences are then used to train `Actor` and `Critic`, and the entire process is repeated until the end of the training. This process can be computationally intensive, as it typically involves loading four models into GPU memory and entails online generation and execution of the RL policy training.

Hybrid Parallelization. LLMs with tens of billions of parameters can no longer be trained or perform inference efficiently on a single device, thus necessitating parallel execution plans. The placement of weights dictates these plans, including parallel computation and communication strategies. Data, tensor, and pipeline are three commonly plan used in distributed training.

Data parallelism either duplicates model parameters across all devices (DP) or shards them across devices (Fully Sharded Data Parallel, FSDP). Each device is then given a different batch of training data, and the forward and backward passes are computed independently. The communication process involves aggregating the gradients or collecting the shared parameters.

Tensor parallelism, which also involves sharing parameters and data across devices, entails each worker conducting a portion of the operator computation. Communication is essential to aggregate the outputs and obtain the final result.

Pipeline parallelism divides the model into multiple parts sequentially. Each device maintains a specific part of the model. For a given part of the model, the corresponding device handles the data from the previous stage (or from training data), and the output needs to be communicated to another device that maintains the subsequent part of the model. While pipeline parallelism generally involves less communication, it faces the challenge of 'pipeline bubbles', which have been intensively studied in prior work [5, 16].

Hybrid Parallelism has emerged for enhanced distributed training performance, integrating various parallel strategies to accommodate specific models and unique training hardware configurations. Notably, Megatron-v2 [17] facilitates high-performance distributed training through expertly designed hybrid parallel execution plans, specifically for transformer-based models. Given the widespread adoption of hybrid parallelism [34], this work focuses on investigating the efficiency of hybrid parallelism for aligning LLMs.

Multi-Model Execution Plan. LLM alignment involves a variety of models with heterogeneous contexts, each demanding different computing resource. Consequently, the placement and execution of these models within the system are critical for optimal performance. As depicted in Figure 2, existing plans can be categorized into three types. In this figure, three different models are depicted during part of the alignment process, including both forward and backward passes. Notably, while only pipeline parallelism schedules are shown in the above example, PUZZLE is not limited to pipeline parallelism and can be applied to both data and tensor parallelism.

Model-by-model (MBM). MBM is an intuitive method for model placement and execution in LLM alignment, as shown in Figure 2(a), where models have identical device settings and sequential execution. Existing system [12, 30] adopts this plan. However, although this plan is straightforward and practical, it has limitations: The model occupies all resources during runtime and is limited to sequential execution. Furthermore, this plan can lead to inefficiency if workloads aren't parallelized effectively.

Single-Model-Dedicated-Device (SMDD). SMDD is another prevalent method for model placement and execution, illustrated in Figure 2(b). This approach involves pre-allocating devices to different models, ensuring each has a dedicated device for potentially concurrent execution. While adopted in works like [15], SMDD faces significant challenges. Its resource allocation strategy is complex and prone to inefficiency. For instance, if Model$_2$ must wait for Model$_1$ to complete its execution, this introduces idle time for certain devices, leading to inefficiencies in resource utilization.

Multi-Model-Dedicated-Device (MMDD). MMDD integrates the previously described MBM and SMDD methods. Figure 2(c) depicts this plan. This plan involves assigning some models to the same resource while allocating different resources to others. Consequently, it enables parallel execution for certain models, while those allocated the same resources are executed sequentially. Existing work, such as [7], adopts this plan. However, despite addressing the limitations of MBM and SMDD, we observe that MMDD lacks in its consideration for fine-grained scheduling, which is crucial for effectively managing heterogeneous context switches.

2.2 Challenges and Motivations

As highlighted earlier, efficient switching between heterogeneous contexts, including intra-stage context switching and inter-stage context switching, presents a critical challenge in aligning LLMs. In this section, we introduce the motivations for efficient intra- and inter-stage switching.

Intra-Stage Context Switching. Multiple models and workloads in the LLM alignment process involve heterogeneous contexts. Furthermore, because these heterogeneous contexts within a stage have no data dependence between them, they can be executed with the multi-model parallel execution plans introduced in Section 2.1. We observe that the selected plans frequently exhibit significant idle time during parallel execution, as illustrated in Figure 2. This can be primarily attributed to the following reason: The presence of significant differences in heterogeneous contexts results in idle time, even when the most optimal combination of plans is employed.

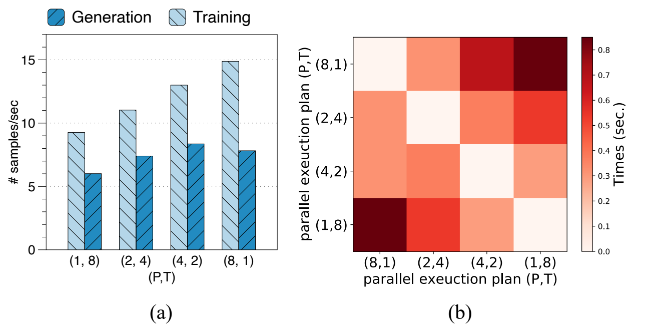

Figure 3: The performance comparison of LLaMA-7B on an 8×A100 PCIe cluster is presented, where P and T denote pipeline and tensor parallelism, respectively, with data parallelism fixed at 1. (a) This includes throughput analysis for various *(P, T)* configurations during generation and training with batch sizes of 64. (b) It also covers the transformation overhead associated with different parallel execution plans.

As illustrated in Figure 2(d), our motivation is centered on exploring the affinities between contexts via fine-grained scheduling, with the goal of minimizing idle time through the implementation of time-sharing in relatively optimal plans.

Inter-Stage Context Switching. In the alignment process, contexts with the same model may have different workloads at various stages, such as generation (decoding) and training. The differences in workload characteristics lead to totally different optimal parallel execution plans. For instance, Figure 3(a) illustrates the performance of both generation and training for LLaMA-7B model under different parallel execution plans with batch size of 64. For clarity, we have omitted the data parallelism dimension in this figure and the data dimension shown in the figure is 1. *(P, T)* represents the dimensions of pipeline and tensor parallelism respectively. The optimal plans *(P, T)* for generation and training are (8, 1) and (4, 2), respectively. Furthermore, in order to achieve the best performance of the same model across different stages, the parameter shuffle overhead between different plans, which is the main cause of inter-stage context switching cost, also needs to be carefully considered. As shown in Figure 3(b), the figure illustrates the switching costs under different plans. We observe that for the plans with higher similarity, the switching cost between them is much lower.

3 Overview

We propose PUZZLE, an efficient system for aligning LLMs. The name comes from the fact that each heterogeneous context resembles a piece of a puzzle. Our goals center on strategically placing these pieces and executing them efficiently within the system.

Figure 4 provides an overview of PUZZLE. The core concept of this system is based on the unique heterogeneous model and workload attributes (referred to as contexts in this paper) encountered during the alignment process. The heterogeneous context properties necessitate frequent switching between intra- and inter-stages, leading to increased overhead. PUZZLE aims to reduce this overhead and enhance system efficiency by exploring potential opportunities within intra- and inter-stage processes.

In Section 4, we minimize intra-stage switching costs by exploring the *affinities* among different contexts and overlapping computations through time-sharing plans.

We then introduce the inter-stage context switch in Section 5. We minimize inter-stage switching costs by finding optimal parallel execution plans with minimal communication costs with highly *similarity*.

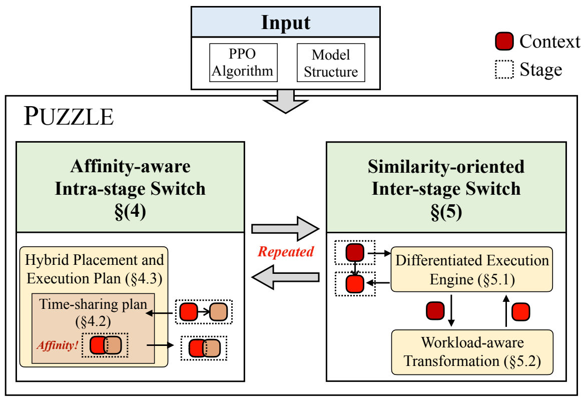
Input
PPO Algorithm
Model Structure
Context
Stage
PUZZLE
Affinity-aware Intra-stage Switch §(4)
Similarity-oriented Inter-stage Switch §(5)
Repeated
Hybrid Placement and Execution Plan (§4.3)
Time-sharing plan (§4.2)
Affinity!
Differentiated Execution Engine (§5.1)
Workload-aware Transformation (§5.2)

Figure 4: Overview of PUZZLE

4 Affinity-aware Intra-stage Switch

Although existing parallel execution techniques are highly efficient, opportunities for further improvement in alignment remain, particularly due to idle time during intra-context switches in parallel execution plans. The heterogeneous context in the alignment process can be divided into different stages, and we have found that orchestrating the context within each stage can significantly enhance intra-stage context switching efficiency.

This section describes the *affinity-aware intra-stage switch* techniques, which are applied in PUZZLE to explore the placement and execution plan among different contexts. Initially, we analyze the overhead and opportunities of intra-stage context switch(§4.1) and introduce the new time-sharing switching plan(§4.2) to explore context collaboration. Finally, integrating all existing approaches results in a diversified space of placement and execution strategies, termed the hybrid plan (§4.3).

4.1 Opportunities of Placement and Execution Plan

Although previous research has explored support for model placement and execution, a common limitation is the lack of attention to collaboration between contexts. Significant room for improvement remains in computational efficiency within the same stage of the alignment process. We first analyze the overhead and opportunities associated with intra-stage context switching.

The Definition of Overhead. First, we analyze the types of overhead that intra-stage context switching can introduce. For placement and execution plans where resources are not shared, there is no need for context switching between them, and thus no overhead is introduced. However, in scenarios where resources are shared, contexts are executed sequentially, thereby introducing overhead.

Opportunities. As illustrated in Figure 2(d), we observe that two contexts can overlap. This occurs because PUZZLE employs hybrid parallelism, which includes tensor parallelism, data parallelism, and pipeline parallelism. Our primary focus is on the overhead from pipeline parallelism, commonly known as pipeline bubbles. This overhead causes certain devices to remain idle either during the pipeline's warm-up phase or as it nears completion, thereby impacting the overall throughput. This idle time in the pipeline can be utilized for other computations, especially in scenarios involving dependencies.

The Affinity-aware Method. During the alignment process, which involves the participation of multiple models, each model's parallel strategy may different. However, depending on the chosen parallel strategy for each model, we can explore their affinities. Here, *affinity* refers to the efficient overlapping of computational processes when two parallel strategies are similar.

4.2 Time-sharing Switching Plan

This section introduces a new main execution plan. This plan is designed to reduce the overhead of context switching within a stage, enabling fine-grained **time-sharing** among different contexts that share the same resources. PUZZLE employ such time-sharing technique within stages to explore context collaboration during the alignment process.

We first revisit the native strategy of training two models in pipeline parallelism manner, Model_1 and Model_2, where the execution order is sequential. As depicted in Figure 5(a), this example illustrates the training process for two models. Each model is divided into four parts, with each part allocated to a device, forming a four-layer pipeline. The workflow also includes four micro-batches, with the two models executed sequentially. However, as shown by the gray box in Figure 5(a), a large number of bubbles are generated between two context

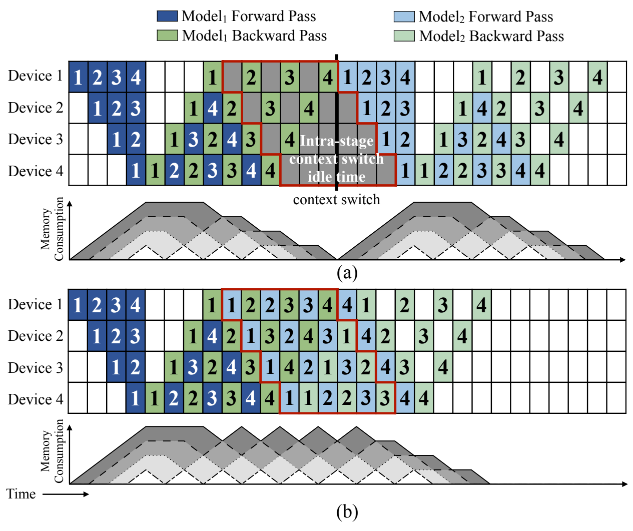

Figure 5: An illustration of the training processes of Model$_1$ and Model$_2$ within stage in pipeline parallelism manner. The figure demonstrates that through efficient time-sharing, idle time during context switching can be effectively reduced without an increase in memory consumption. (a) In native pipeline, models within the same stage are executed sequentially. (b) The training processes of two models can naturally overlap, reducing the empty periods in the pipeline.

switches. During these idle periods, the device remains idle. Therefore, this scenario presents significant opportunities for optimization, particularly in executing tasks that do not have dependencies.

Figure 5(b) illustrates an example of the time-sharing technique. By integrating the subsequent training task with the preceding one, the idle time in the previous task can be effectively utilized. This time-sharing technique is analogous to process concurrency in operating systems. Since these tasks lack dependencies, they can be executed concurrently. For instance, in the case where the forward pass of the first micro-batch of Model$_2$ is executed immediately after the backward pass of the first micro-batch of model Model$_1$, this execution occurs at time 9 instead of the delayed time 15. Furthermore, the schedule depicted in the figure is similar to completing the training of eight micro-batches in a single model.

Theoretical Speedup. In Figure 5(b), it is shown that the first micro-batch forward of Model$_2$ starts after Model$_1$'s first micro-batch backward, placing Model$_1$ in its cool-down phase and Model$_2$ in the warm-up phase. Let T^f represent the duration of each forward pass, T^b the backward pass, and P the number of pipeline stages. Assuming equal model sizes and micro-batch numbers, with $T_1^f = T_2^f = T^f$ and $T_1^b = T_2^b = T^b$, the pipeline idle time reduces from $T_{idle}^{ser} = 2(P-1)(T^f + T^b)$ to $T_{idle}^{ts} = (P-1)(T^f + T^b)$ in time-sharing scenarios. This implies a halving of the idle time with time-sharing.

However, due to the heterogeneous contexts in LLM alignment, their durations vary. To gain a more comprehensive understanding of the speedup brought about by the use of fine-grain time-sharing techniques, we conduct a theoretical analysis here. Revisiting the essence of time-sharing technology, which interleaves two contexts, this approach results in the reduction of the warmup and cool-down phases for contexts with shorter durations. Therefore, given that the number of microbatches is denoted as N_B, the theoretical speedup can be expressed as:

$$Theoretical\ Speedup = \frac{(N_B + P - 1)\sum_{k\in\{1,2\}}(T_k^f + T_k^b)}{N_B \sum_{k\in\{1,2\}}(T_k^f + T_k^b) + (P-1)\max\limits_{k\in\{1,2\}}(T_k^f + T_k^b)} \quad (1)$$

Memory Consumption. Compared to the original sequential version, our new plan exhibits the same peak memory consumption. Since our system follows the 1F1B synchronous pipeline pattern, its memory consumption first peaks with the retention of forward pass activations and then gradually decreases with the completion of the backward pass.

In Figure 5(a), memory consumption initially rises from valley to peak, then falls back to the valley, a process that is repeated twice. Conversely, considering Device1 as an example, when the activation tensor of the first micro-batch forward propagation of Model$_1$ is released, our system begins the first micro-batch forward propagation of Model$_2$, as shown in Figure 5(b). Therefore, this time-sharing approach does not increase memory pressure and, furthermore, it optimizes the computational resources of the device.

4.3 Hybrid Placement and Execution Plan

Finally, by combining all the existing methods, the composite space of placement and execution plans is enriched, referred to as the hybrid plan. In the hybrid plan, the alignment process can be divided into multiple stage groups based on the workload characteristics of each stage, and these stage groups can be further divided into sub-stage groups. Therefore, each sub-stage can adopt an appropriate placement and execution plan based on its characteristics. The execution plans between sub-stage groups are sequential. This means that each sub-stage group can occupy all computing resources to execute the functions of that stage group. For stage groups requiring fine-grained scheduling, parallel execution plans should remain consistent. Figure 6 provides some simple examples to demonstrate the working principle of the hybrid layout and execution plan.

As shown in Figure 6(a), the alignment process can be divided into three stages, each with dependencies on the previous stage. The different boxes represent various models, with

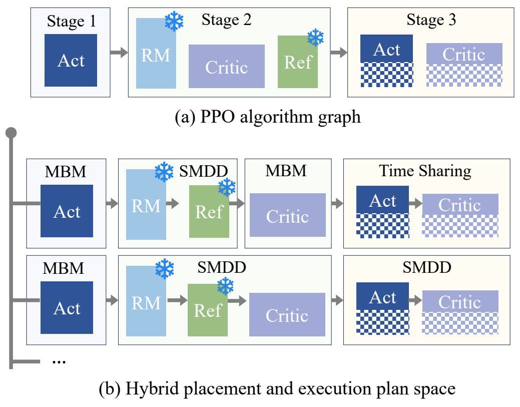

Figure 6: Example of different hybrid placement and execution plans. (a) The aligning algorithm graph is divided into stages, with each stage having data dependencies on the previous one. Within a stage, there are no dependencies. (b) Illustration of two possible examples of placement and execution plans according to the algorithm graph.

models potentially existing in multiple stages simultaneously. As previously mentioned, stage groups can be divided into smaller sub-stage groups, with each sub-stage group able to choose different placement and execution plans. Two potential plans are shown in Figure 6(b). In the first plan, single model stage group is equivalent to applying the MBM plan, while the `Act` and `Critic` stage groups leverage a fine-grained time-sharing plan. In the second plan, similar to the scheduler approach, each stage group basically uses the SMDD plan, where each model within a stage group exclusively utilizes computing resources.

5 Similarity-oriented Inter-stage Switch

An adaptive parallel execution plan is necessary for the alignment process to maintain optimal performance between stages, taking into account its current computing characteristics. However, adjusting a parallel execution plan at runtime is non-trivial, especially in hybrid parallelism where different plans distribute parameters across devices based on the parallelism degree. This results in variations in the number of layers and parameters per layer. Therefore, the main challenge in runtime transformation is the efficient switching of plans with minimal overhead, while considering the distribution of parameters.

This section introduces the *similarity-oriented inter-stage switch* technique, which is applied in PUZZLE to maximize performance while considering switching overhead. An example of inter-stage switching in PUZZLE is shown in Figure 7. PUZZLE uses a similarity criterion to find optimal parallel execution plans with minimal communication costs. Initially,

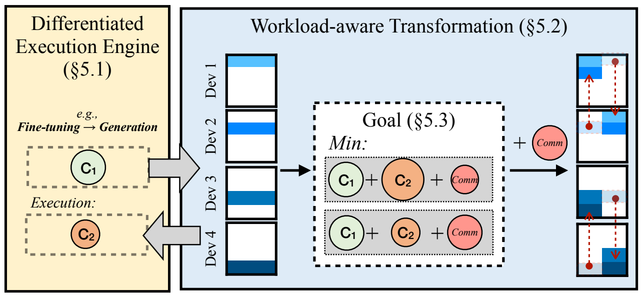
Differentiated Execution Engine (§5.1)
e.g., Fine-tuning → Generation
C_1
Execution:
C_2
Workload-aware Transformation (§5.2)
Dev 1
Dev 2
Dev 3
Dev 4
Goal (§5.3)
Min:
C_1 + C_2 + Comm
C_1 + C_2 + Comm
+ Comm

Figure 7: An illustration of inter-stage switching in PUZZLE

we introduce the differentiated execution engine(§5.1) which is used to execute different workloads for the same model. Second, the workload-aware transformation (§5.1) is introduced to transform the execution plan with minimal communication costs. Finally, we outline the goal of inter-stage switching(§5.3) and propose our similarity-oriented plan.

5.1 Differentiated Execution Engine

In LLM alignment, the same model may be responsible for different types of workloads, termed differentiated workloads. For instance, the `Actor` model needs to handle both generation and fine-tuning tasks, representing the context in the two stages respectively. However, these two workloads exhibit different computational characteristics and demands. Therefore, a differentiated execution engine has been designed to accommodate the model's varying workloads at different stages. As shown in Figure 7, before the model executes the next stage, such as changing from fine-tuning to generation, it needs to go through workload-aware transformation (§5.2) to adopt a more efficient parallel execution plan with minimal communication costs. After the transformation, the differentiated execution engine executes the workload based on the transformed stage.

5.2 Workload-aware Transformation

We now introduce the workload-aware transformation used in PUZZLE, designed for efficient inter-stage context switching with minimal communication costs, as illustrated in Figure 7. From our previous analysis, we understand that the transformation of parallelism from training to generation involves shifting from a higher to a lower degree of pipeline parallelism. This process involves sharing the parameters of the original layer with other peers to form a tensor parallel group. Therefore, we initially analyze the communication traffic associated with such switching and introduce the concept of similarity to identify optimal parallel execution plans with minimal communication costs.

Communication Volume Analysis. In PUZZLE, hybrid parallelism, which encompasses data, tensor, and pipeline parallelism, is employed to significantly reduce communication. Below, we conduct an analysis of this communication reduction, using mixed precision training as a case study.

Suppose a model contains L layers, and the training is conducted on a cluster of N workers. P, D, T represent the ways of pipeline, data, and tensor parallelism respectively. The number of broadcasts required by each device can be determined by the formula $f_P = \frac{P_{src}}{P_{dst}}$. We use a concrete example in Figure 7 to illustrate the process, where $(L, N) = (4, 4)$. The plan transformation involves shifting from $(P, D, T) = (4, 1, 1)$ to $(2, 1, 2)$. It is important to note that the above example only illustrates the case where $D = 1$. However, PUZZLE is capable of handling cases where $D > 1$. Therefore, during a single training step in PUZZLE, the transfer involves only $D - \frac{1}{f_P}$ copies of parameters between GPUs.

Routing Table Generation. PUZZLE utilizes a *routing table* to determine which worker needs to broadcast data and which group of workers needs to receive it. Based on the parallel execution plans before and after the transformation, PUZZLE is able to generate a corresponding routing table.

Drawing inspiration from number system conversion, we designed Algorithm 1 to generate routing tables for plan transformations. In this algorithm, *base* is defined as the triplet (P, D, T), which indicates that the rank follows to the dimension order of tensor, data, and pipeline parallelism. For example, if the rank is 3, in $base_1 = (4, 1, 1)$ it corresponds to $(3, 0, 0)_{base_1}$, whereas in $base_2 = (2, 1, 2)$, it corresponds to $(1, 0, 1)_{base_2}$. We utilize the function GETINDEX to convert the rank to an index with *base*, and the function GETRANK to revert to the rank. The variables f_P, f_D, f_T represent the ratios of different dimensions between *src* and *dst*.

The function GETROUTINGTBL generates a routing table that requires broadcasting parameters from *rank* to the destinations. In this function, the *rank* is first converted to the index id_{src} under $base_{src}$ (Line 10), and then the corresponding index id_{dst} under $base_{dst}$ is calculated (Line 12). Finally, the rank number is obtained using GETRANK. Each *rank* corresponds to f_P destinations to which data needs to be broadcast.

Asynchronous Weight Transformation. In the alignment process, training data generation typically utilizes the latest parameters, which are updated during the optimizer step. Furthermore, parameter updates are performed using optimizer states and gradients, with no dependencies among these updates. Therefore, the optimizer step and parameter exchange can be executed asynchronously and interleaved.

We minimize the weight transformation overhead with a two-stream schedule. In PUZZLE, trainable parameters are partitioned into n segments, such as $P_1, P_2, \ldots, P_k$. Following each partial optimizer step and parameter update, the transformation engine broadcasts these parameters to other devices, enabling parallel operations and reducing end-to-end latency.

Algorithm 1 Routing Table Generation

```
1:  function GETINDEX($r, base$)                ▷ $base$ is $(P, D, T)$
2:      $P, D, T \leftarrow base$
3:      return $(\frac{r}{D \times T} \bmod P, \frac{r}{T} \bmod D, r \bmod T)$
4:  function GETRANK($idx, base$)
5:      $P, D, T \leftarrow base$
6:      return $idx[0] \times D \times T + idx[1] \times T + idx[2]$
7:  function GETROUTINGTBL($rank, base_{src}, base_{dst}$)
8:      $TBL \leftarrow \{rank : [\,]\}$
9:      $f_P, f_D, f_T \leftarrow$ BITWISEDIV($base_{src}, base_{dst}$)
10:     $id_{src} \leftarrow$ GETINDEX($rank, base_{src}$)
11:     for $i \leftarrow 0$ to $f_P - 1$ do
12:         $id_{dst} \leftarrow (\frac{id_{src}[0]}{f_P}, id_{src}[1], \frac{id_{src}[2]}{f_T} + i)$
13:         Append GETRANK($id_{dst}, base_{dst}$) to $TBL[rank]$
14:     return $TBL$
```

5.3 Similarity-oriented Plan

The transformation of parallel execution plans enables PUZZLE to identify more efficient strategies for executing differentiated workloads, although this process incurs additional communication costs. In this section, our goal is to delineate the objectives of inter-stage context switching, with a particular focus on achieving this transition in a lightweight manner.

Assuming the objective is to transition from context A to context B by moving from plan i to plan j. We define the time taken for context A under plan i as T_i^A, and the time for context B under plan j as T_j^B. Additionally, the time required to switch from plan i to plan j is denoted as T_{ij}. With these definitions, the goal of the inter-stage switch can be expressed as follows:

$$\min_{i,j} T_i^A + T_j^B + T_{ij} \tag{2}$$

However, identifying the optimal solution to achieve the minimum objective in parallel execution plans is non-trivial. As observed in Figure 3, the overhead tends to be lower when the parallel execution plans are more similar. This 'similarity' is defined in terms of the distance between the tuples (P_i, D_i, T_i) and (P_j, D_j, T_j).

Consequently, we propose a *similarity-oriented* approach for the inter-stage switch. Drawing from these observations, we begin with the (P, D, T) configuration of plan i. The process involves gradually reducing the size of P and increasing the size of T, while continually assessing whether there is a decrease according to Equation (2). This process is repeated until no further decrease is observed.

6 Implemenation

We implemented PUZZLE based on Megatron-LM [17] which applies hybrid parallelism. PUZZLE utilizes the light-weight context switch techniques described in §4 and §5 to optimize the heterogeneous context switches introduced by LLM alignment. Moreover, the number of GPUs allocated can vary for each model depending on the placement and execution plan. Therefore, PUZZLE configures a communicator for each model to adapt different parallel execution plans. Here are some implementation details about PUZZLE:

Hybrid Inference Scheduling. Since the alignment process involves both generation and training, we have implemented the hybrid inference scheduling proposed in DeepSpeed Inference [1] to enhance generation performance when using pipeline parallelism. Additionally, a few configurations also need to be considered, particularly the micro-batch size for different phases in generation.

Shadow Generation Model. To adapt the training-to-generation transformation within the alignment process, PUZZLE utilizes a shadow model that has the same architecture as the original model. This shadow model only occupies memory during the parallel execution plan transformation and generation process, and releases it upon completion of the process.

Performance Profiling. To determine the placement and execution plan of each model within a stage and the transformation of the parallel execution plan between stages, PUZZLE employs an offline profiler to gather runtime data under various settings prior to the alignment process. For each model, three key metrics are considered: *(P, D, T)*, which represent pipeline parallelism, data parallelism, and tensor parallelism, respectively. Moreover, the micro-batch size in each phase of generation needs to be considered.

7 Evaluation

7.1 Experimental Setup

Cluster Testbed. We evaluate PUZZLE on two different representative clusters, which differ in network topology and network bandwidth.

orion is a cluster with 32 GPUs on four worker nodes. Each worker node has 8 NVIDIA A100-PCIe-80GB GPUs connected to 2 CPU sockets via PCIe switches and is equipped with 100 Gb/s Infiniband EDR.

phoenix is another cluster with 32 GPUs with four worker nodes. Each node is equipped with 8 NVIDIA A100-SXM-80GB GPUs, interconnected via NVLink to form a heterogeneous ring, where half of the connections have double bandwidth due to a bond of two links. Infiniband EDR at 100Gb/s is used for communication. Compared to PCIe, NVLink provides significantly higher communication throughput for GPU connections.

LLMs Configuration. Our experimental focus centers on the Proximal Policy Optimization (PPO) algorithm, a prominent technique in the RLHF alignment of modern LLMs. We employ LLaMA models with a range of parameter sizes, specifically 350M, 7B, 13B, and 33B, with specifications detailed in Table 1.

Table 1: Specification of models setup

# Parameters	350M	7B	13B	33B
num_layer	16	32	40	64
hidden_size	512	4096	5120	6656
intermediate_size	11008	11008	13824	17920

PPO Hyperparameters and Datasets. Throughout our experiments, we use the following hyperparameter configuration for the PPO algorithm. The number of PPO epochs and the batch size for rollouts are both set to 1. Additionally, the prompt sequence length and the generation length are each set to 256. We configured the lower tensor parallelism setting in the generation phase of DeepSpeed-Chat to fit the limited memory capacity of the GPU. We utilize the default dataset `Dahoas/rm-static` in DeepSpeed-Chat. This is an open-source dataset hosted on HuggingFace, designed for aligning LLMs. This dataset is typically used for training a helpful and harmless assistant using RLHF.

Baselines and Software Configuration. We compare PUZZLE with the state-of-the-art RLHF training framework DeepSpeed-Chat [30]. In comparative tests with DeepSpeed-Chat, a hybrid engine [30] implemented by DeepSpeed is used, capable of switching parallel strategies during the training and generation phases. This strategy employs ZeRO stage 3 [22] during training, while it utilizes tensor parallelism during the generation phase.

Evaluation Metrics. For the end-to-end evaluation, we measure the sample throughput during alignment. Sample throughput is defined as the rate of processing samples from start to finish, typically quantified in samples per second (samples/sec).

Table 2: PPO algorithm setup used for Evaluation

Configuration	Batch Size	# GPUs
7B/350M	128	8
7B/7B	256	16
13B/350M	256	16
13B/7B	256	16
33B/7B	128	32

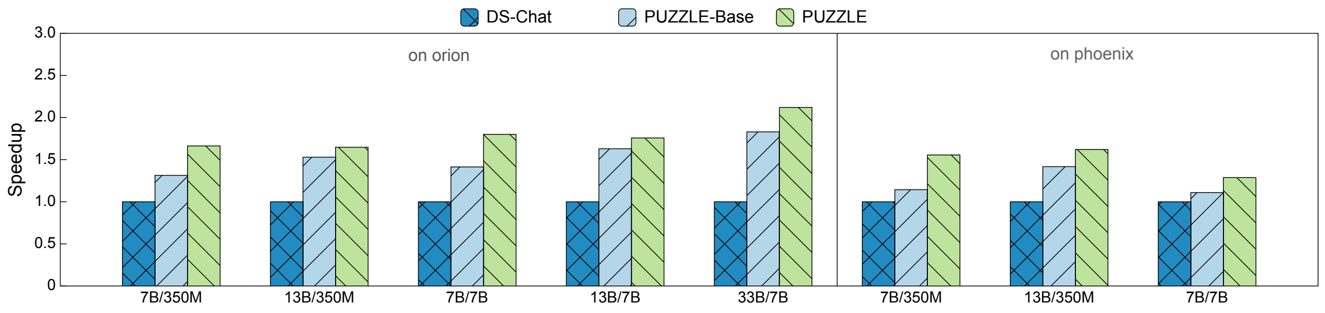

Figure 8: End-to-End Speedup of different configurations.

7.2 End-to-End Performance

We evaluate the end-to-end performance of various model combinations across two distinct clusters. The evaluation setup is detailed in Table 2. Configuration X/Y reflects parameter counts in PPO's actor and critic models, mirrored in the reference and reward models, respectively. These configurations are done in the form of weak scaling, where batch sizes are increased along with the number of GPUs. Except for the 33B/350M configuration, which is adjusted to accommodate the GPU memory capacity.

Figure 8 illustrates the end-to-end speedup comparison between PUZZLE and DeepSpeed-Chat with a hybrid engine under two different cluster configurations, while also incorporating PUZZLE-Base, which exclusively applies pipeline parallelism, into the experiment. PUZZLE achieves an average speedup of 1.83× and 1.50×, respectively, on the two clusters. In the comparison between *phoenix* and *orion*, PUZZLE exhibits a higher speedup on *orion*. This reason is due to the bandwidth limitations among the cluster's GPUs and frequent intra-node communication, there is a reduction in efficiency. Specifically, during the generation stage, DeepSpeed-Chat opts for increased tensor parallelism to accommodate GPU memory constraints. PUZZLE addresses this issue by selecting the optimal hybrid parallel execution plan for various stages, utilizing light-weight inter-stage switching.

PUZZLE-Base, which solely utilizing pipeline parallelism and is devoid of additional optimizations, and its batch size in decode phase is evenly divided by the pipeline size, was further evaluated. This evaluation highlights our system's baseline performance, which notably surpasses that of DeepSpeed-Chat. Furthermore, this also demonstrates that relying solely on static parallelism in the alignment process is suboptimal due to heterogeneous models and workloads involved. Therefore, it necessitates considering optimizations both intra-stage and inter-stage to achieve enhanced performance.

7.3 Intra-stage Ablation Study

This section analyzes the effectiveness of time-sharing techniques used for intra-stage context switching. As shown in Figure 9, we compared the performance of intra-stage across different configurations on two cluster, as outlined in Table 2. We used configurations without time-sharing as a baseline for comparison. It is evident that the use of time-sharing techniques significantly enhances intra-stage context switch performance, reducing idle time, with up to 1.12× speedup within stage. The figure also reveals that the speedup ratios in both clusters are essentially consistent, indicating that time-sharing is effective across different clusters. Furthermore, under configurations with similar models, a more pronounced speedup is observed, which will be analyzed subsequently.

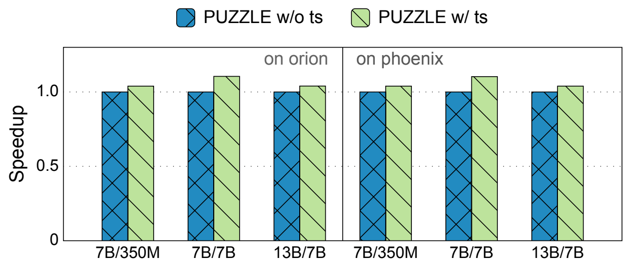

Figure 9: Performance of using time-sharing

Figure 10 illustrates the speedup achieved by PUZZLE under various configurations and batch sizes relative to theoretical speedup, calculated using Equation (1). (P,D,T) represents data, tensor, and pipeline parallelism dimensions. It has been observed that the actual speedup closely aligns with the theoretical predictions. We observed that with an increased number of pipelines, the performance improves, as demonstrated by the (16,1,1) configuration under 7B/7B in the figure. This is reasonable because time-sharing primarily eliminates idle time, and more pipelines mean more potential idle time. Furthermore, it was noted that using more similar models or smaller batch sizes also lead to better performance, achieving up to a 1.34× speedup within the stage.

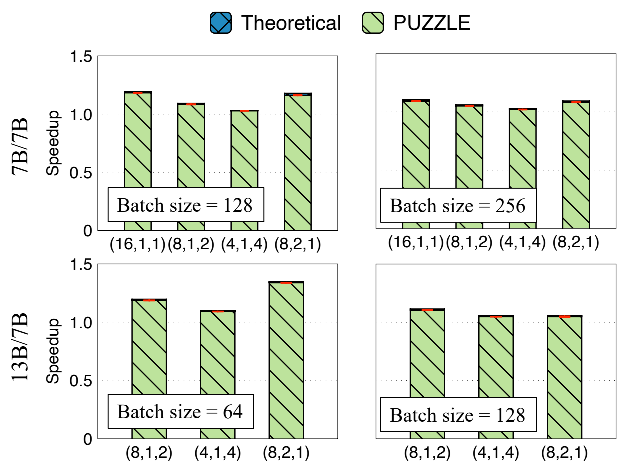

Figure 10: Theoretical speedup and PUZZLE speedup of intra-stage switching

7.4 Inter-stage Ablation Study

We now evaluate the performance improvements resulting from the parallel execution plan transformation involving inter-stage context switching in PUZZLE. Our evaluation follows to the configurations listed in Table 2. Fixed generation-optimal and training-optimal parallel execution plans (same with PUZZLE-Base) were used as the basis for comparison in the experiment. Figure 11 illustrates the end-to-end speedup across various settings.

PUZZLE achieves an average speedup of 1.22× compared to the generation-optimal plan and 1.34× compared to the training-optimal plan. The overhead is studied in the next section. Additionally, we found that generation-optimal strategies consistently outperform training-optimal ones, primarily because generation stages are typically more time-consuming, making the choice of an appropriate generation plan more critical. Moreover, a greater speedup was observed in the phoenix setting with higher communication throughput, attributed to enhanced opportunities for tensor parallelism during generation.

This further demonstrates that selecting a parallel strategy based solely on the characteristics of a single workload is insufficient for optimal performance. Conversely, an optimal approach requires considering the characteristics of various workloads collectively and accounting for the overhead during transitions to enhance performance in the alignment process.

7.5 Fine-Grained Performance Breakdown

To thoroughly understand the PUZZLE alignment process, we conducted a fine-grained performance breakdown, comparing it with DeepSpeed-Chat and incorporating PUZZLE-Base into the analysis. Our study, focusing on the 7B/7B alignment in two different cluster configurations, is presented in Figure 12, where the performance of each stage is measured separately. This analysis clearly outlines performance metrics for each context within an iteration, highlighting our method's advantages.

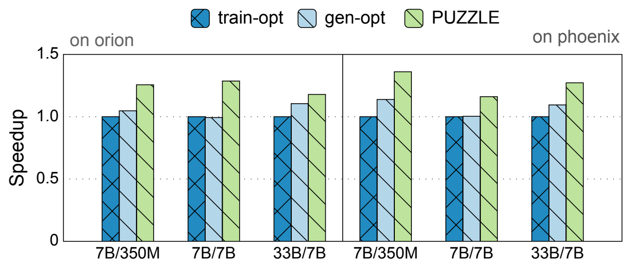

Figure 11: Performance of PUZZLE is compared to that of optimal parallelism plan for training and generation.

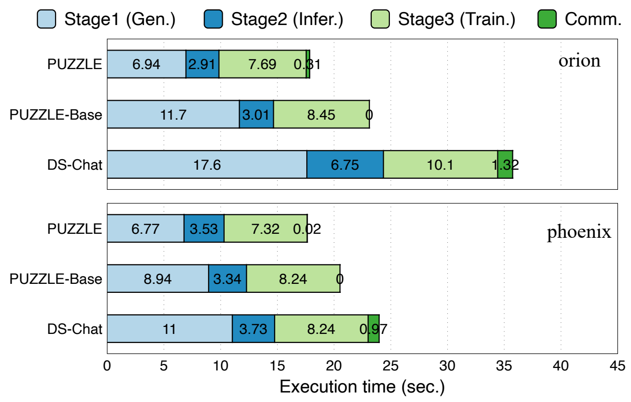

Figure 12: Fine-grained performance breakdown in one iteration in aligning 7B/7B under two different cluster configurations.

The data shows PUZZLE outperforms DeepSpeed-Chat in various stages. We discuss the reasons behind the performance optimization of these three stages respectively. In Stage 1, PUZZLE uses light-weight parallel execution plan transformation for optimal strategy selection, favoring tensor parallelism over pipeline parallelism to enhanced performance. In contrast, DeepSpeed-Chat employs only tensor parallelism leads to lower performance. For Stages 2 and 3, PUZZLE achieves significant improvements by selecting appropriate placement and execution plans, with notable gains in Stage 2. The effectiveness of Phase 3 is further enhanced by the use of time-sharing technology.

The communication overhead is detailed in Table 3. This table compares the overheads of PUZZLE and DeepSpeed-Chat in two different clusters. The inter-stage switch in the experiment focused on `Actor`, we altered only the parameters

Table 3: Inter-stage Switching Overhead Breakdown

Config.	*orion*		*phoenix*	
	PUZZLE	DS-Chat	PUZZLE	DS-Chat
7B/350M	0.31 s	1.32 s	0.02 s	1.03 s
13B/350M	0.37 s	3.02 s	0.20 s	2.02 s
33B/7B	0.49 s	7.54 s	0.48 s	3.37 s

of `Actor`, and followed the configurations outlined in Table 2.

As the table shows, the overhead of PUZZLE in both clusters is lower than that of DeepSpeed-Chat, with each being less than one second. Particularly in the *orion* cluster, and notably in the 33B/7B configuration, where the 33B model's parameters are relatively large and are distributed across 32 GPUs, the switch incurs significant overhead, with up to 7.54 seconds in DeepSpeed-Chat. PUZZLE, however, achieves the switch with minimal overhead, transitioning from $(32,1,1)$ to $(16,1,2)$, incurring only 0.49 seconds in the *orion*. Meanwhile, in the *phoenix* cluster, transitioning to $(8,1,4)$ increases overhead but also significantly enhances performance. This demonstrates the effective balance PUZZLE achieves between overhead and performance optimization across various cluster configurations. Moreover, in the *phoenix* cluster, where most communication is completed intra-node, PUZZLE effectively leverages the communication advantages of NVLink, resulting in lower overhead.

8 Related Work

LLM Alignment. Existing alignment methods, especially human preference alignment can be broadly categorized into three major categories: reinforcement learning [20, 23], contrastive learning [21, 33], and hindsight instruction relabeling [14, 32]. Among these methods, the reinforcement learning approach stands out as the primary method for achieving this alignment. Representative work is InstructGPT [20].

Following the development of ChatGPT [18] or InstructGPT, numerous distributed frameworks have been proposed. DeepSpeed-Chat [30] compose the full system capability of DeepSpeed training and inference into a unified engine called hybrid engine. ColossalAI [12] uses various techniques like ZeRO [22] to accelerate the RLHF alignment paradigms. trlX [7] propose both online and offline training methods, along with algorithmic system enhancements to minimize compute and memory requirements. Beaver [2], leveraging DeepSpeed, is introduced to parallelize InstructGPT-like alignment paradigms using Safe RLHF methods. These frameworks primarily utilize the PPO algorithm for alignment. It represents the mainstream approach for aligning LLMs.

Parallelization Training Systems. Previous works target different parallel strategies. GPipe [8] employs pipeline parallelism by distributing layer-level weights across multiple devices. It facilitates parallel computations among these devices in a sequential, pipelined fashion. PipeDream [16] and DAPPLE [5] proposes pipeline parallelism planners for efficient pipeline partitioning and scheduling. Tofu [26] generates tensor model parallel execution plans by a novel DP algorithm. Alpa [34] generates more sophisticated execution plans, considering both inter-operator and intra-operator parallelism. These works primarily focus on optimizing a single target model. In contrast, PUZZLE addresses the efficient context switching of multiple models under an optimal parallel strategy, making it orthogonal to these studies.

LLM Inference Optimization. DeepSpeed-Inference [1] employs a range of techniques, including hybrid inference scheduling, to utilize the power of multi-GPU systems and enhance inference performance. vLLM [10] implements PagedAttention, which partitions each sequence's KV cache into fixed-size blocks, facilitating memory sharing and reducing memory consumption. FlashAttention [3] provides an optimized implementation of SelfAttention operation by reducing data movement via blockwise computation. FlexGen [24] designed an efficient swapping schedule to maximize throughput on a single GPU while sacrificing latency. These works have implemented a series of optimizations for inference and can be effectively applied to PUZZLE.

9 Conclusion

We propose PUZZLE, an efficient system for aligning LLMs that treats model context as a first-class citizen. We abstract the key concepts of context from the alignment problem and identify the key challenge of LLM as efficient switching between heterogeneous contexts. To address the challenge, we propose a two-dimensional approach for efficient handling of heterogeneous contexts, focusing on both intra- and inter-stage context switching. In each stage, we reduce switching costs by analyzing affinities between various contexts and leveraging overlapping computation through time-sharing strategies. Additionally, to minimize switching overhead between stages, we formulate a similarity-based approach to identify the most efficient inter-stage context-switching plan, aiming to lower communication expenses. PUZZLE achieves up to 2.12× speedup in end-to-end LLM alignment.

10 Acknowledgments

We sincerely thank the anonymous reviewers for their valuable feedback on this paper. This work is supported by National Key R&D Program of China under Grant 2023YFB3001704, NSFC for Distinguished Young Scholar (62225206), and National Natural Science Foundation of China under Grants 62302251, U23B2027. Jidong Zhai is the corresponding author of this paper (zhaijidong@tsinghua.edu.cn).

References

[1] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2022.

[2] Josef Dai, Xuehai Pan, Ruiyang Sun, Jiaming Ji, Xinbo Xu, Mickel Liu, Yizhou Wang, and Yaodong Yang. Safe rlhf: Safe reinforcement learning from human feedback. *arXiv preprint arXiv:2310.12773*, 2023.

[3] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[4] Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Alpacafarm: A simulation framework for methods that learn from human feedback. *arXiv preprint arXiv:2305.14387*, 2023.

[5] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, et al. Dapple: A pipelined data parallel approach for training large models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 431–445, 2021.

[6] Google. Bard. https://bard.google.com/.

[7] Alexander Havrilla, Maksym Zhuravinskyi, Duy Phung, Aman Tiwari, Jonathan Tow, Stella Biderman, Quentin Anthony, and Louis Castricato. trlx: A framework for large scale reinforcement learning from human feedback. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8578–8595, 2023.

[8] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Xu Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 103–112, 2019.

[9] Julia Kreutzer, Joshua Uyheng, and Stefan Riezler. Reliability and learnability of human bandit feedback for sequence-to-sequence reinforcement learning. *arXiv preprint arXiv:1805.10627*, 2018.

[10] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[11] Kimin Lee, Hao Liu, Moonkyung Ryu, Olivia Watkins, Yuqing Du, Craig Boutilier, Pieter Abbeel, Mohammad Ghavamzadeh, and Shixiang Shane Gu. Aligning text-to-image models using human feedback. *arXiv preprint arXiv:2302.12192*, 2023.

[12] Shenggui Li, Hongxin Liu, Zhengda Bian, Jiarui Fang, Haichen Huang, Yuliang Liu, Boxiang Wang, and Yang You. Colossal-ai: A unified deep learning system for large-scale parallel training. In *Proceedings of the 52nd International Conference on Parallel Processing*, pages 766–775, 2023.

[13] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 663–679, 2023.

[14] Hao Liu, Carmelo Sferrazza, and Pieter Abbeel. Languages are rewards: Hindsight finetuning using human feedback. *arXiv preprint arXiv:2302.02676*, 2023.

[15] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th USENIX symposium on operating systems design and implementation (OSDI 18)*, pages 561–577, 2018.

[16] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 1–15, 2019.

[17] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 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*. Association for Computing Machinery, 2021.

[18] OpenAI. Chatgpt. https://chat.openai.com/chat.

[19] OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2308.01320*, 2023.

[20] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[21] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.

[22] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[23] Rajkumar Ramamurthy, Prithviraj Ammanabrolu, Kianté Brantley, Jack Hessel, Rafet Sifa, Christian Bauckhage, Hannaneh Hajishirzi, and Yejin Choi. Is reinforcement learning (not) for natural language processing?: Benchmarks, baselines, and building blocks for natural language policy optimization. *arXiv preprint arXiv:2210.01241*, 2022.

[24] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pages 31094–31116. PMLR, 2023.

[25] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[26] Minjie Wang, Chien-chin Huang, and Jinyang Li. Supporting very large models using automatic dataflow graph partitioning. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 1–17, 2019.

[27] Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*, 2023.

[28] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 595–610, 2018.

[29] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. {AntMan}: Dynamic scaling on {GPU} clusters for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 533–548, 2020.

[30] Zhewei Yao, Reza Yazdani Aminabadi, Olatunji Ruwase, Samyam Rajbhandari, Xiaoxia Wu, Ammar Ahmad Awan, Jeff Rasley, Minjia Zhang, Conglong Li, Connor Holmes, et al. Deepspeed-chat: Easy, fast and affordable rlhf training of chatgpt-like models at all scales. *arXiv preprint arXiv:2308.01320*, 2023.

[31] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, 2022.

[32] Tianjun Zhang, Fangchen Liu, Justin Wong, Pieter Abbeel, and Joseph E Gonzalez. The wisdom of hindsight makes language models better instruction followers. *arXiv preprint arXiv:2302.05206*, 2023.

[33] Yao Zhao, Rishabh Joshi, Tianqi Liu, Misha Khalman, Mohammad Saleh, and Peter J Liu. Slic-hf: Sequence likelihood calibration with human feedback. *arXiv preprint arXiv:2305.10425*, 2023.

[34] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. 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, 2022.