KAITIAN: Communication Framework for Efficient Collaboration Across Heterogeneous Accelerators in Embodied AI Systems

Jieke Lin*†, Wanyu Wang*†, Longxiang Yin†, and Yinhe Han†

*Hangzhou Institute for Advanced Study, UCAS

†Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
Emails: {linjinke23, wangwanyu23}@mails.ucas.ac.cn, {yinlongxiang, yinhes}@ict.ac.cn

Jieke Lin and Wanyu Wang contributed equally to this work.

Corresponding author: Longxiang Yin {yinlongxiang@ict.ac.cn}

Abstract—Embodied Artificial Intelligence (AI) systems, such as autonomous robots and intelligent vehicles, are increasingly reliant on diverse heterogeneous accelerators (e.g., GPG-PUs, NPUs, FPGAs) to meet stringent real-time processing and energy-efficiency demands. However, the proliferation of vendor-specific proprietary communication libraries creates significant interoperability barriers, hindering seamless collaboration between different accelerator types and leading to suboptimal resource utilization and performance bottlenecks in distributed AI workloads. This paper introduces KAITIAN, a novel distributed communication framework designed to bridge this gap. KAITIAN provides a unified abstraction layer that intelligently integrates vendor-optimized communication libraries for intra-group efficiency with general-purpose communication protocols for inter-group interoperability. Crucially, it incorporates a load-adaptive scheduling mechanism that dynamically balances computational tasks across heterogeneous devices based on their real-time performance characteristics. Implemented as an extension to PyTorch and rigorously evaluated on a testbed featuring NVIDIA GPUs and Cambricon MLUs, KAITIAN demonstrates significant improvements in resource utilization and scalability for distributed training tasks. Experimental results show that KAITIAN can accelerate training time by up to 42% compared to baseline homogeneous systems, while incurring minimal communication overhead (2.8-4.3%) and maintaining model accuracy. KAITIAN paves the way for more flexible and powerful heterogeneous computing in complex embodied AI applications.

Index Terms—Heterogeneous Computing, Distributed Communication, Embodied AI, Robotics, Deep Learning, Task Scheduling, Resource Management, PyTorch.

I. INTRODUCTION

MBODIED AI systems, ranging from autonomous drones navigating complex terrains to sophisticated robotic assistants interacting with the physical world, represent a paradigm shift in artificial intelligence [1]. These systems must perceive their environment, make intelligent decisions, and act upon them, often under strict real-time constraints and limited power budgets. To achieve the requisite computational power, modern embodied AI platforms increasingly integrate a diverse array of specialized hardware

accelerators, including General-Purpose Graphics Processing Units (GPGPUs) for massively parallel computations, Neural Processing Units (NPUs) optimized for deep learning inference and training, and Field-Programmable Gate Arrays (FPGAs) for custom logic. This trend towards *heterogeneous computing* promises significant performance and efficiency gains by matching computational tasks to the most suitable hardware [2].

However, realizing the full potential of heterogeneous architectures is fraught with challenges, particularly in the domain of inter-accelerator communication. Each hardware vendor typically provides its own proprietary communication library (e.g., NVIDIA's NCCL for GPUs, Cambricon's CNCL for MLUs). While these libraries are highly optimized for communication between homogeneous devices (e.g., GPU-to-GPU), they are often incompatible with libraries from other vendors. This "walled-garden" ecosystem creates significant interoperability barriers, making it exceedingly difficult to orchestrate collaborative computation across different types of accelerators within a single, unified application, such as a distributed deep learning training job. Consequently, developers are often forced to use only one type of accelerator or resort to cumbersome, inefficient manual data transfers via the host CPU, leading to underutilized hardware resources, increased communication latency, and stifled innovation in leveraging the combined strengths of diverse accelerators.

The problem is particularly acute in distributed training of large AI models, a common requirement for advanced embodied AI capabilities. Efficient scaling of training across multiple devices is paramount, but heterogeneity introduces complexities. Different accelerators may exhibit varying computational speeds and memory capacities, leading to load imbalances if tasks are distributed naively. Synchronizing model parameters and gradients across these disparate devices without incurring substantial communication overhead is a critical hurdle. Existing deep learning frameworks like

PyTorch [3] and TensorFlow [4] primarily support a single communication backend per training job, further limiting native support for true heterogeneous distributed training.

To address these pressing challenges, we propose **KAITIAN**, a distributed communication framework designed to unlock the collaborative potential of heterogeneous accelerators in embodied AI systems. KAITIAN offers a software abstraction layer that intelligently manages communication across a diverse hardware landscape. The core contributions of this work are:

- A Hybrid Communication Architecture: KAITIAN seamlessly integrates vendor-specific, high-performance communication libraries (e.g., NCCL, CNCL) for efficient intra-group communication within clusters of homogeneous accelerators, while employing a generalpurpose communication backend (Gloo) for interoperable inter-group communication across different accelerator types, relayed via host CPU memory.
- A Load-Adaptive Scheduling Mechanism: To counteract performance disparities among heterogeneous devices, KAITIAN incorporates a dynamic load balancing strategy. It benchmarks accelerator performance and proportionally allocates data batches, ensuring that all devices contribute effectively and complete their computational workloads in a synchronized manner, thereby maximizing overall system throughput.
- Seamless Integration with PyTorch: KAITIAN is implemented as a pluggable communication backend for PyTorch, allowing researchers and developers to leverage heterogeneous resources with minimal changes to their existing distributed training scripts.
- Empirical Validation: We demonstrate KAITIAN's effectiveness through comprehensive experiments on a testbed comprising NVIDIA GPUs and Cambricon MLUs. Results show significant training speedups (up to 42%) in heterogeneous configurations compared to homogeneous baselines, with negligible communication overhead and preserved model accuracy.

The remainder of this paper is organized as follows: Section II discusses related work in heterogeneous communication and scheduling. Section III details the architecture and core mechanisms of KAITIAN. Section IV describes the implementation and presents a thorough experimental evaluation. Section V discusses the implications, limitations, and potential extensions of our work. Finally, Section VI concludes the paper and outlines future research directions.

II. RELATED WORK

The challenges of communication and coordination in heterogeneous computing environments have been addressed from various perspectives. This section reviews relevant literature in communication libraries, frameworks for heterogeneous systems, and load balancing techniques.

A. Communication Libraries and Frameworks

Standardized communication interfaces like the Message Passing Interface (MPI) [5] have long been a cornerstone of high-performance computing (HPC). While MPI is versatile, it may not always be optimized for the specific memory hierarchies and interconnects of modern accelerators. Vendor-specific libraries such as NVIDIA's NCCL [6] and AMD's RCCL [7] provide highly optimized collective communication primitives for their respective GPU architectures. Similarly, Cambricon offers CNCL for its MLUs. These libraries achieve excellent performance for homogeneous device clusters.

For inter-vendor communication, solutions are less mature. Gloo [8], developed by Facebook, is a collective communication library that supports various backends, including TCP/IP for CPU-based communication and direct GPU-to-GPU communication via NCCL or InfiniBand verbs. While Gloo can bridge different systems, its direct support for diverse accelerator types beyond GPUs is limited without custom extensions.

Several research projects have aimed to unify communication across heterogeneous devices. For instance, OpenCL [9] and SYCL [10] provide programming models for heterogeneous platforms, but they focus more on computation offloading and kernel execution rather than high-level distributed training communication patterns. Frameworks like StarPU [11] and Legion [12] offer task-based programming models that can manage dependencies and data movement in heterogeneous systems, but may require significant programming effort to integrate with existing deep learning workflows.

B. Load Balancing in Heterogeneous Systems

Effective load balancing is crucial for maximizing performance in heterogeneous environments where processing units can have vastly different computational capabilities. Static load balancing techniques assign tasks based on predetermined device characteristics, which can be suboptimal if performance varies dynamically. Dynamic load balancing strategies, on the other hand, adapt task allocation based on runtime conditions.

In the context of distributed deep learning, several approaches have been proposed. PipeDream [13] introduced pipeline parallelism that can mitigate straggler effects in heterogeneous clusters, but focuses on a different parallelism dimension. Some works have explored dynamic batch sizing or work stealing. For example, [14] proposed heterogeneity-aware dynamic load balancing for distributed deep learning training, which involves adapting workload sizes.

KAITIAN differentiates itself by providing a pragmatic and readily integrable solution within a popular deep learning framework (PyTorch). It focuses on a hybrid communication strategy that leverages the best of both worlds (vendoroptimized and general-purpose libraries) and combines it

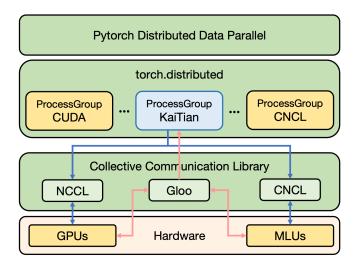


Fig. 1. KAITIAN Framework Integration with PyTorch.

with a simple yet effective load-adaptive mechanism specifically tailored for synchronous data-parallel training across diverse accelerator types. While other frameworks might offer more comprehensive but complex solutions, KAITIAN aims for ease of use and immediate applicability to existing embodied AI research workflows.

III. THE KAITIAN FRAMEWORK DESIGN

KAITIAN is architected to provide a flexible and efficient communication backbone for distributed applications running on heterogeneous accelerator clusters. Its design philosophy centers on modularity, leveraging existing optimized libraries where possible, and providing a clear abstraction for managing cross-device data exchange.

The core principle of KAITIAN is its integration into PyTorch's distributed ecosystem, as illustrated in Figure 1. KAITIAN introduces a new custom ProcessGroup, 'ProcessGroupKaiTian', into 'torch.distributed' using PyTorch's C++ extension mechanism. When PyTorch's Distributed-DataParallel (DDP) module is initialized with this custom ProcessGroup, all subsequent collective communication calls (e.g., AllReduce for gradient synchronization, broadcast for model synchronization) are routed through KAITIAN. This allows KAITIAN to intelligently manage how these communication primitives are executed across a mix of hardware. Specifically, for operations involving only homogeneous accelerators (e.g., a group of NVIDIA GPUs), KAITIAN dispatches the calls to highly optimized vendor-specific libraries like NCCL. Similarly, for operations confined to Cambricon MLUs, CNCL is used. These intragroup homogeneous communications are depicted by blue data paths in Figure 1. For inter-group communication that spans different accelerator types (e.g., aggregating gradients from both GPUs and MLUs), KAITIAN utilizes the Gloo collective communication library, where data is relayed via host CPU memory. This heterogeneous communication path is shown with pink data paths.

The primary challenge KAITIAN addresses is the inherent limitation in mainstream deep learning frameworks like Py-Torch, which typically bind an entire distributed training job to a single communication backend (e.g., NCCL, Gloo, or MPI). This prevents the native, simultaneous use of multiple vendor-specific libraries optimized for different hardware. 'ProcessGroupKaiTian' acts as a meta-backend or a dispatch layer to overcome this.

A. Hierarchical Communication Management

KAITIAN's 'ProcessGroupKaiTian' manages communication across diverse accelerators by distinguishing between operations within groups of homogeneous accelerators and operations spanning heterogeneous accelerator groups.

- Intra-Group Homogeneous Communication: Within a group of homogeneous accelerators (e.g., all NVIDIA GPUs or all Cambricon MLUs involved in a collective operation), KAITIAN directs communication tasks (e.g., AllReduce, Broadcast) to the vendor's highly optimized communication library. For instance, operations solely among GPUs would use NCCL, while operations solely among MLUs would use CNCL. This ensures near-native performance for communication within such homogeneous groups.
- Inter-Group Heterogeneous Communication: For communication between different types of accelerator groups (e.g., exchanging gradients between a GPU-based group and an MLU-based group), KAITIAN employs a general-purpose communication layer, Gloo. Data transfer between different accelerator types is managed by:
 - Copying the tensor from the source accelerator's memory to host (CPU) RAM.
 - 2) Transmitting the tensor data from the source host's RAM to the target host's RAM using Gloo's TCP/IP backend (or other suitable Gloo backends if available for host-to-host communication).
 - 3) Copying the tensor from the target host's RAM to the target accelerator's memory.

This explicit staging through host memory is necessary because direct memory-to-memory communication between, for example, an NVIDIA GPU and a Cambricon MLU is generally not supported at the hardware or driver level. While this relay introduces additional memory copy overhead (accelerator-to-host, host-to-accelerator), it provides a universal mechanism for interoperability. KAITIAN's design aims to minimize the impact of such inter-group transfers, typically limiting them to essential synchronization points like gradient aggregation across all involved devices.

B. Hybrid Communication Backend Integration

To implement this hierarchical strategy, KAITIAN extends PyTorch's 'torch.distributed' C++ backend by introducing

'ProcessGroupKaiTian'. This meta-process group can internally manage or dispatch to multiple underlying communication backends. When a collective operation is initiated by PyTorch DDP using 'ProcessGroupKaiTian':

- 1) KAITIAN analyzes the participating processes and their device types to determine if the operation is within a single type of accelerator group (homogeneous) or spans multiple, different accelerator types (heterogeneous).
- 2) If the operation is homogeneous (e.g., an AllReduce among a set of GPUs), it dispatches the call to the appropriate vendor library (e.g., NCCL) for that group.
- 3) If the operation is heterogeneous (e.g., an AllReduce involving both GPUs and MLUs), it orchestrates the multi-step transfer via Gloo and host memory as described above.

This allows a single PyTorch distributed training script to transparently utilize a mix of accelerators without requiring the user to manually manage the underlying communication complexities for different hardware.

C. Load-Adaptive Scheduling Mechanism

Heterogeneous accelerators invariably possess different raw processing capabilities and memory bandwidths. In synchronous distributed training (like data-parallel AllReduce SGD), the overall pace is dictated by the slowest worker (straggler). Without load balancing, faster devices would frequently idle waiting for slower ones, leading to poor resource utilization. KAITIAN's load-adaptive mechanism aims to mitigate this by dynamically distributing the workload proportionally to each accelerator's effective processing speed.

The mechanism operates in two phases:

- Offline Benchmarking (Optional) / Online Profiling:
 - Initial Benchmarking: Before the main training loop, KAITIAN can optionally run a short profiling job (e.g., a few forward/backward passes of the target model with a small, fixed amount of data) on each participating accelerator. This provides an initial estimate of their relative speeds. The fastest device is assigned a score of 1.0, and other devices *i* are scored relative to it: score_i = time_{fastest}/time_i.
 - Online Adaptation (Future Work): While the current implementation primarily uses initial benchmarking, a more advanced version could continuously monitor computation times per batch during training and dynamically adjust scores. This would account for variations in performance due to factors like thermal throttling or contention for shared resources.
- Dynamic Data Allocation: During data loading for each training step, KAITIAN's custom 'Distributed-Sampler' (or a similar data partitioning mechanism)

allocates portions of the global mini-batch to each device i based on its score. If the total global batch size is B_{global} , and there are N devices, the batch size for device i, b_i , is calculated such that it is proportional to score $_i$, while ensuring $\sum_{i=1}^N b_i = B_{global}$. A common way to achieve this is:

$$\text{batch_size}_i = \frac{\text{score}_i}{\sum_{j=1}^{N} \text{score}_j} \times B_{\text{global}}$$

The allocated batch sizes are then rounded to the nearest integer, ensuring the sum matches $B_{\rm global}$. This aims to equalize the computation time across all devices for each step, as $T_i = {\rm Workload}_i/{\rm Speed}_i \approx (k \cdot {\rm batch_size}_i)/(c \cdot {\rm score}_i)$, where k and c are constants. By making ${\rm batch_size}_i \propto {\rm score}_i$, T_i should ideally be similar for all devices.

This load-adaptive approach ensures that faster accelerators process more data, while slower ones handle a proportionally smaller load, leading to more balanced computation times per iteration and improved overall training throughput.

D. System Coordination

KAITIAN utilizes a lightweight coordination service, such as Redis, for initial process discovery, group membership management, and synchronization of metadata (e.g., benchmark scores, rendezvous information). This is a common pattern in distributed PyTorch setups.

By combining these design elements—hierarchical communication management via a dedicated ProcessGroup, hybrid backend integration, and load-adaptive scheduling—KAITIAN provides a robust and efficient solution for harnessing the power of heterogeneous accelerator clusters in demanding embodied AI applications.

IV. IMPLEMENTATION AND EVALUATION

To validate the efficacy and performance of the KAITIAN framework, we implemented it as an extension to PyTorch's distributed communication library and conducted a series of experiments on a dedicated heterogeneous hardware testbed.

A. Implementation Details

KAITIAN's core logic is implemented in C++ as a custom process group backend for PyTorch (version 1.10 was used for development, but it's designed to be adaptable to newer versions). This 'ProcessGroupKaiTian' backend interfaces with vendor libraries (NCCL 2.11, CNCL 1.5) and Gloo (PyTorch's bundled version).

• Process Group Management: We extended PyTorch's 'ProcessGroup.hpp' and related classes to create 'ProcessGroupKaiTian'. This class is responsible for managing subgroups for homogeneous communication (e.g., by internally invoking 'ProcessGroupNCCL' or 'ProcessGroupCNCL' for operations confined to those device types) and orchestrates inter-group transfers using 'ProcessGroupGloo'.

- **DistributedSampler Override:** For load-adaptive scheduling, we implemented a custom 'KaitianDistributedSampler' that overrides PyTorch's default 'DistributedSampler'. This sampler takes per-device scores as input and distributes dataset indices accordingly.
- Control Plane: A set of Python utility scripts and command-line tools are provided to launch and manage KAITIAN training jobs. These scripts handle the setup for different accelerator environments and use a Redis server for rank discovery, initial handshake, and sharing benchmark scores.
- Environment Isolation in Experiments: For experimental purposes and to ensure reproducibility across different hardware stacks, Docker containers were employed. These containers encapsulated the distinct software environments required for NVIDIA GPUs (CUDA Toolkit, NVIDIA drivers) and Cambricon MLUs (CNToolkit, Cambricon drivers), preventing library conflicts. This use of Docker is an implementation detail of the experimental setup, not a conceptual part of the KAITIAN framework's core communication logic itself.

The implementation effort focused on creating a clean interface that abstracts the underlying complexity from the enduser, allowing them to specify heterogeneous configurations with relative ease.

B. Experimental Setup

• Hardware Testbed:

- CPU: AMD EPYC 7763 (64-core)
- RAM: 64 GB DDR4
- GPUs: 2 x NVIDIA GeForce GTX 1080 (Pascal architecture, 8 GB VRAM each)
- NPUs: 2 x Cambricon MLU370-S4 (ShangNeng architecture, 16 GB VRAM each)
- Interconnect: PCIe Gen3 for accelerators. Gigabit Ethernet for host-level communication if Gloo uses TCP/IP across different physical nodes (though in this setup, all devices are in one server, so Gloo likely uses shared memory or local loopback for CPU-level transfers).

• Software Environment:

- OS: Ubuntu 20.04 LTS
- Containerization (for experimental isolation):
 Docker 20.10
- NVIDIA Stack: CUDA Toolkit 11.2, NVIDIA Driver 460.xx, NCCL 2.11
- Cambricon Stack: CNToolkit 3.2.2, Cambricon Driver 5.9.4, CNCL 1.5
- Deep Learning Framework: PyTorch 1.10 (compiled from source with custom backend)
- Coordination: Redis 6.0

• Benchmark Task:

 Model: MobileNetV2 [17] (a widely used efficient CNN architecture, suitable for embodied AI).

- Dataset: CIFAR-10 [16] (32x32 color images, 10 classes). While small, it allows for rapid iteration and focuses evaluation on communication and scheduling overheads rather than being I/O bound by a massive dataset.
- Training Configuration:
 - * Optimizer: Stochastic Gradient Descent (SGD) with momentum 0.9, weight decay 5e-4.
 - * Learning Rate: Initial LR 0.1, with a step decay schedule.
 - * Global Batch Size: 256 (distributed across devices).
 - * Epochs: 50 (sufficient to observe training trends and convergence).
 - * Loss Function: Cross-Entropy Loss.

• Baselines and Configurations Evaluated:

- **2G** (NCCL): Homogeneous training using 2 NVIDIA GPUs with NCCL.
- **2M** (**CNCL**): Homogeneous training using 2 Cambricon MLUs with CNCL.
- KAITIAN (1G+1M): Heterogeneous training with 1 GPU and 1 MLU.
- KAITIAN (2G+1M): Heterogeneous training with 2 GPUs and 1 MLU.
- KAITIAN (1G+2M): Heterogeneous training with 1 GPU and 2 MLUs.
- KAITIAN (2G+2M): Heterogeneous training with 2 GPUs and 2 MLUs.

For KAITIAN configurations, the load-adaptive mechanism was active.

C. Evaluation Metrics

The primary metrics for evaluation were:

- **Total Training Time:** Wall-clock time to complete 50 epochs. This is the key indicator of end-to-end performance.
- Model Accuracy: Top-1 accuracy on the CIFAR-10 test set after 50 epochs, to ensure that KAITIAN does not negatively impact convergence.
- b Communication Overhead (Homogeneous): Comparison of training time using KAITIAN with only homogeneous devices (e.g., 2 GPUs managed by KAITIAN but only using its NCCL path) versus native NCCL/CNCL, to quantify the overhead introduced by KAITIAN's framework layer.
- Scalability and Resource Utilization: Inferred from training time improvements as more heterogeneous devices are added and from the effectiveness of the loadadaptive mechanism.

D. Results and Analysis

1) Training Efficiency and Accuracy: Figure 2 shows the total training time and final model accuracy for various configurations.

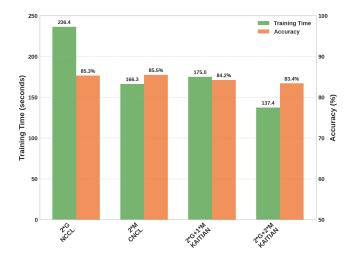


Fig. 2. KAITIAN Training Efficiency (Time to Complete 50 Epochs) and Model Accuracy Comparison on CIFAR-10 with MobileNetV2. Lower training time is better. Accuracy is Top-1 on the test set.

Key observations:

- Heterogeneous Speedup: The KAITIAN (2G+2M) configuration achieved the fastest training time of 137.4 seconds. This represents a significant speedup of approximately 42% compared to the 2G (NCCL) baseline (236.4 seconds) and about 17% compared to the 2M (CNCL) baseline (166.3 seconds). This clearly demonstrates KAITIAN's ability to effectively harness the combined computational power of diverse accelerators.
- Scalability: As more accelerators were added in heterogeneous configurations (e.g., moving from 1G+1M to 2G+1M, then to 2G+2M), the training time generally decreased, indicating good scalability. For instance, 2G+1M (175.0s) was faster than both 2G (236.4s) and 1M (not shown, but would be slower than 2M's 166.3s / 2 = 332s if linear).
- Model Accuracy: Accuracy remained comparable across all configurations. The 2G+2M KAITIAN setup achieved 83.4% accuracy, while the 2G (NCCL) baseline reached 85.3% and the 2M (CNCL) baseline reached 85.5%. The slight variations (around 2%) are within typical experimental noise for CIFAR-10 training runs and suggest that KAITIAN's communication and scheduling mechanisms do not adversely affect model convergence or final performance. The primary goal of such a framework is to accelerate training without degrading accuracy, which KAITIAN achieves.
- 2) Effectiveness of Load-Adaptive Mechanism: Figure 3 illustrates the impact of the load-adaptive mechanism. To elaborate on the figure's implications:
 - Without load adaptation (e.g., if a naive 50/50 split of batches was used between a faster GPU and a slower MLU), the MLU would become a bottleneck, and the GPU would be underutilized. This would correspond to a suboptimal point (like "Strategy A" or "C" if they represent imbalanced fixed allocations).

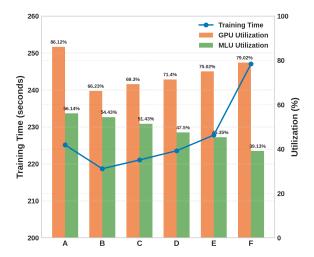


Fig. 3. Impact of Load Adaptive Mechanism on Heterogeneous Training (e.g., 1G+1M configuration). Strategy A might be naive equal batch splitting, Strategy B KAITIAN's adaptive splitting, Strategy C a suboptimal fixed ratio. The y-axis likely represents training time per epoch or overall, and x-axis different strategies or device utilization ratios. KAITIAN (Strategy B) finds a balance that minimizes training time by ensuring both GPU and MLU are effectively utilized according to their capabilities.

- KAITIAN's mechanism, by benchmarking and assigning scores (e.g., GPU score 1.0, MLU score 0.7 if MLU is 70
- This result underscores the critical importance of load balancing in heterogeneous environments. Even with an efficient communication layer, performance gains can be nullified if workloads are not intelligently distributed.
- *3) Communication Overhead Analysis:* Figure 4 quantifies the overhead introduced by KAITIAN's framework layer when operating in a purely homogeneous setting.

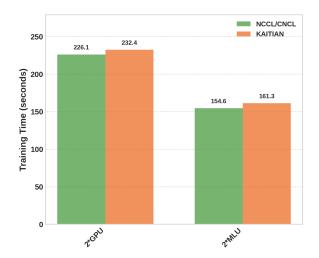


Fig. 4. Communication Overhead of KAITIAN in Homogeneous Settings. Comparison of training time using native NCCL/CNCL versus KAITIAN managing the same homogeneous devices.

The results indicate:

• **GPU Homogeneous:** For 2 GPUs, native NCCL training took 226.1 seconds. When KAITIAN managed

these 2 GPUs (internally still using NCCL for the actual communication), the time was 232.4 seconds. This represents an overhead of (232.4 - 226.1) / 226.1 \approx 2.8%.

• MLU Homogeneous: For 2 MLUs, native CNCL training took 154.6 seconds. With KAITIAN managing these 2 MLUs (internally using CNCL), the time was 161.3 seconds. This is an overhead of (161.3 - 154.6) / 154.6 ≈ 4.3%.

These overheads are minimal and demonstrate that KAITIAN's abstraction layer for managing multiple backends and its data routing logic impose only a slight performance penalty when operating on homogeneous devices. This is crucial, as it means users do not pay a significant "KAITIAN tax" for the added flexibility when heterogeneity is not actively being exploited for a particular sub-task. The overhead is likely due to the extra dispatch logic within KAITIAN's custom process group.

Overall, the experimental results validate that KAITIAN successfully enables efficient distributed training across heterogeneous accelerators, achieving substantial speedups by effectively utilizing combined resources and intelligently balancing loads, all while introducing minimal framework overhead and maintaining model accuracy.

V. DISCUSSION

The development and evaluation of KAITIAN offer valuable insights into the challenges and opportunities of heterogeneous computing for embodied AI. Our results demonstrate that by thoughtfully integrating existing communication technologies and implementing adaptive scheduling, significant performance gains can be unlocked.

A. Implications for Embodied AI

Embodied AI systems often operate under tight latency and energy constraints. The ability to efficiently utilize a mix of accelerators (e.g., power-efficient NPUs for routine inference, high-performance GPUs for bursts of complex computation or on-device training/adaptation) is critical. KAITIAN provides a foundational software layer that can facilitate such flexible hardware utilization. For instance:

- Faster Model Development Cycles: Researchers can iterate more quickly by distributing training across all available accelerators, regardless of vendor.
- **Deployment of Larger Models:** By pooling resources, more complex and capable AI models could potentially be trained and even deployed on edge platforms that feature heterogeneous SoCs (System-on-Chips).
- **Energy Efficiency:** Future extensions could incorporate power consumption into the load-adaptive mechanism, optimizing for energy efficiency in addition to speed, which is paramount for battery-powered robotic systems.

B. Analysis of Inter-Group Communication Overhead

The primary overhead in KAITIAN's inter-group communication stems from the data relay through host CPU memory (Device \rightarrow CPU \rightarrow Gloo \rightarrow CPU \rightarrow Device). While our results show substantial net speedups, this indirect path is inherently slower than direct device-to-device communication (like NVLink for GPU-GPU or specific interchip links). The performance gain from adding heterogeneous accelerators must outweigh this overhead. KAITIAN's success in the 2G+2M configuration suggests that for the MobileNetV2/CIFAR-10 task, the computational power added by the MLUs (and vice-versa for GPUs) was significant enough to overcome this. For tasks with very frequent or very large inter-group synchronizations, this overhead might become more dominant.

C. Limitations

Despite its promising results, KAITIAN has limitations:

- **CPU Bottleneck:** The reliance on CPU-mediated transfers for inter-group communication can become a bottleneck if the CPU's processing power or memory bandwidth is insufficient, especially with many heterogeneous devices or very frequent synchronization.
- Scope of Heterogeneity: The current implementation supports GPUs and MLUs. Extending it to other accelerator types like FPGAs or specialized ASICs would require developing or integrating corresponding lowlevel communication primitives and device management code.
- Dynamic Task Graphs: KAITIAN is primarily designed for data-parallel training with regular synchronization. More complex distributed patterns, such as pipeline parallelism or models with highly dynamic computational graphs, might require more sophisticated scheduling and communication strategies.
- Fault Tolerance: The current framework does not explicitly address fault tolerance, which can be important in larger distributed systems.
- Online Profiling Granularity: The load-adaptive mechanism currently relies on initial benchmarking. A more fine-grained online profiler that adapts to performance fluctuations during training could yield further benefits but adds complexity.

D. Future Work

KAITIAN opens several avenues for future research and development:

- Advanced Load Balancing: Incorporate online monitoring of computation times and potentially power consumption to create a more dynamic and multi-objective load balancing scheduler. Explore predictive models for task completion times on different accelerators.
- Direct Inter-Device Communication Exploration: Investigate emerging technologies or research efforts that

might enable more direct (or lower-overhead) communication paths between different vendors' accelerators, potentially bypassing the CPU for certain transfers (e.g., using technologies like CXL if supported by future devices).

- **Broader Accelerator Support:** Extend KAITIAN to support a wider range of accelerators, including FPGAs and other NPUs, by developing new device-specific interface modules for 'ProcessGroupKaiTian'.
- Support for Diverse Parallelism Strategies: Enhance KAITIAN to support other forms of parallelism beyond data parallelism, such as model and pipeline parallelism, across heterogeneous devices.
- Robotic Platform Integration and Benchmarking: The ultimate goal is to deploy and evaluate KAITIAN on real-world robotic platforms. This involves porting to embedded heterogeneous SoCs (e.g., NVIDIA Jetson series, Qualcomm Robotics platforms) and benchmarking on representative embodied AI tasks like SLAM (Simultaneous Localization and Mapping), visual servoing, and grasp planning. This will expose new challenges related to resource constraints and real-time performance.
- Integration with Higher-Level Workflow Managers: Explore integration with workflow managers like Kubeflow or Ray to simplify the deployment and management of KAITIAN-accelerated applications in larger clusters.

By addressing these areas, KAITIAN can evolve into an even more powerful and versatile framework for the next generation of embodied AI systems.

VI. CONCLUSION

In this paper, we introduced KAITIAN, a novel distributed communication framework designed to overcome interoperability barriers and unlock the potential of heterogeneous accelerator clusters for embodied AI applications. By strategically integrating vendor-specific communication libraries for high-performance intra-group communication with a general-purpose layer for inter-group data exchange, and by incorporating a load-adaptive scheduling mechanism, KAITIAN enables efficient collaboration between diverse hardware like NVIDIA GPUs and Cambricon MLUs.

Our implementation within PyTorch and comprehensive evaluations on an image classification task demonstrated that KAITIAN can significantly accelerate distributed training—achieving up to a 42% reduction in training time in a 2-GPU + 2-MLU configuration compared to homogeneous baselines. This performance gain is achieved with minimal communication overhead (2.8–4.3% in homogeneous settings) and without compromising model accuracy. The load-adaptive mechanism proved crucial in balancing workloads and maximizing resource utilization across devices with varying computational capabilities.

KAITIAN represents a significant step towards more flexible, scalable, and powerful computing paradigms for embodied AI. It empowers researchers and developers to leverage the full spectrum of available hardware resources, accelerating the development and deployment of complex AI models that are vital for intelligent autonomous systems. Future work will focus on extending KAITIAN's capabilities, supporting a broader range of accelerators and parallelism strategies, and deploying it on physical robotic platforms to tackle real-world embodied intelligence tasks.

ACKNOWLEDGMENT

This research was partially supported by the National Key Research and Development Program of China (2022YFB4501600).

REFERENCES

- [1] W. Sun, S. Hou, Z. Wang, B. Yu, S. Liu, X. Yang, S. Liang, Y. Gan, and Y. Han, "DaDu-E: Rethinking the Role of Large Language Model in Robotic Computing Pipeline," *arXiv* preprint arXiv:2412.01663, 2024.
- [2] S. Mittal, "A Survey of Techniques for Architecting and Managing Deep Learning Accelerators for Hyperscale Datacenters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 8, pp. 1762-1781, Aug. 2020.
- [3] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in Proc. 33rd Int. Conf. Neural Inf. Process. Syst. (NeurIPS), Vancouver, BC, Canada, Dec. 2019, pp. 8026–8037.
- [4] M. Abadi et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," arXiv preprint arXiv:1603.04467, 2016.
- [5] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard, Version 4.0," 2021. [Online]. Available: https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf
- [6] NVIDIA, "NVIDIA Collective Communications Library (NCCL)," NVIDIA Developer, 2023. [Online]. Available: https://developer. nvidia.com/nccl.
- [7] AMD, "ROCm Communication Collectives Library (RCCL)," AMD ROCm Documentation, 2023. [Online]. Available: https://rocm.docs. amd.com/projects/rccl/en/latest/
- [8] Facebook, "Gloo: A Collective Communications Library," GitHub Repository, 2023. [Online]. Available: https://github.com/facebookincubator/gloo.
- [9] Khronos OpenCL Working Group, "The OpenCL Specification," Version 3.0. [Online]. Available: https://www.khronos.org/opencl/
- [10] Khronos SYCL Working Group, "SYCL Specification," Version 2020. [Online]. Available: https://www.khronos.org/sycl/
- [11] C. Augonnet et al., "StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures," Concurrency and Computation: Practice and Experience, vol. 23, no. 2, pp. 187-198, 2011.
- [12] M. Bauer et al., "Legion: Expressing Locality and Independence with Logical Regions," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal. (SC)*, 2012.
- [13] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "PipeDream: Generalized Pipeline Parallelism for DNN Training," in *Proc. 27th ACM Symp. Oper. Syst. Principles (SOSP '19)*, Oct. 2019, pp. 25–38.
- [14] M. Li, J. Li, C. Wu, T. Zhang, M. Chen, and B. Li, "Heterogeneity-aware dynamic load balancing for distributed deep learning training," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 11, pp. 2725–2739, Nov. 2021.
- [15] J. H. Park, G. Yun, C. M. Yi, N. T. Nguyen, S. Lee, J. Choi, S. H. Noh, and Y. Choi, "HetPipe: Enabling Large DNN Training on (Whimpy) Heterogeneous GPU Clusters through Integration of Pipelined Model Parallelism and Data Parallelism," in *Proc. 2020 USENIX Annu. Tech. Conf. (USENIX ATC '20)*, Jul. 2020, pp. 307–321.

- [16] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," Univ. of Toronto, Toronto, ON, Canada, Tech. Rep., 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.
- [17] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, Jun. 2018, pp. 4510–4520.