Indonesian Journal of Electrical Engineering and Computer Science
Vol. 43, No. 2, August 2026, pp. 662~671
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v43.i2.pp662-671

Evaluation of hybrid parallelism for scalable training of DenseNet-121 in diabetic retinopathy classification

Indar Sugiarto[1,2], Djoni Haryadi Setiabudi[3], Darrell Cornelius Rivaldo[3], Taweesak Kijkanjanarat[4], Resmana Lim[1]
[1]Department of Electrical Engineering, Petra Christian University, Surabaya, Indonesia
[2]Center for Artificial Intelligence and Computational Modeling Studies, Petra Christian University, Surabaya, Indonesia
[3]Department of Informatics, Petra Christian University, Surabaya, Indonesia
[4]Department of Electrical and Computer Engineering, Thammasat University, Bangkok, Thailand

Article Info

Article history:

Received Sep 12, 2025
Revised Jul 6, 2026
Accepted Jul 23, 2026

Keywords:

Data parallelism
DenseNet-121
Distributed training
Hybrid parallelism
Pipeline parallelism

ABSTRACT

Training large and complex deep learning models is often constrained by GPU memory limitations and prolonged training times. While several parallelism strategies have been proposed, this study specifically evaluates hybrid parallelism—a combination of data parallelism and pipeline parallelism—to address both challenges simultaneously. Using a case study on diabetic retinopathy (DR) classification with the DenseNet-121 architecture, we analyze the trade-off between computational efficiency and memory scalability. Results show that although hybrid parallelism does not yet provide speedup compared to a single-GPU setup—due to communication overhead and pipeline fragmentation—it enables training of large models that exceed the memory capacity of a single GPU. The trained model achieved a validation accuracy of 0.737, a quadratic weighted kappa (QWK) of 0.861, and a weighted F1-score of 0.749. In contrast, pure data parallelism showed a potential speedup of up to 1.9× in scenarios where the model still fits within a single GPU. These findings highlight the critical role of hybrid parallelism in overcoming the memory wall in large-scale model training, though optimization to reduce overhead remains a key challenge.

Corresponding Author:

Indar Sugiarto
Center for Artificial Intelligence and Computational Modeling Studies, Petra Christian University
Surabaya, Indonesia
Email: indi@petra.ac.id

1. INTRODUCTION

The current era of technological advancement is marked by the widespread adoption of artificial intelligence, particularly deep learning, across various domains [1]. This trend is fuelled by the explosive growth of data—projected to reach 181 zettabytes by 2025 [2]—as well as rapid advancements in hardware, notably GPUs, and their supporting software ecosystems [3]. To solve increasingly complex problems, deep learning models have grown in size and depth, with architectures comprising millions to billions of parameters designed to capture intricate features and achieve superior accuracy [4]. However, this progress has introduced significant computational challenges, particularly prolonged training times and limited GPU memory capacity [5]. Training massive models on large datasets can take days or even weeks [6]. When the model size exceeds the memory capacity of a single GPU, training becomes infeasible without specialized strategies.

To address these limitations, distributed training using multiple GPUs has become the standard solution. Several strategies have emerged, including data parallelism, model parallelism, and pipeline parallelism [7]. This study focuses on hybrid parallelism, a method that combines data parallelism (to handle

large volumes of data) with pipeline parallelism (to accommodate large models), aiming to improve both scalability and training efficiency.

As a case study, we apply this approach to diabetic retinopathy (DR) classification using fundus retinal images. DR is a complication of diabetes and a leading cause of preventable blindness when detected early [8]. With the number of DR cases projected to rise to 592 million by 2025 [9], the demand for fast and accurate automated detection systems is critical. Thus, DR classification serves as a highly relevant and impactful test case for evaluating distributed training strategies.

Previous studies have explored parallelism strategies for DR classification. However, many suffer from limitations such as overly simplistic models, lack of baseline comparisons (e.g., single-GPU training), and insufficient analysis of resource utilization and bottlenecks [7], [10]. This leaves a gap for a more comprehensive understanding of parallelism effectiveness in real-world scenarios.

To address these gaps, this paper offers a comprehensive evaluation of hybrid parallelism performance using the DenseNet-121 architecture. The main contributions of this study are:

- A quantitative analysis of the trade-off between training speedup and memory scalability in large-model scenarios.
- GPU utilization analysis to identify communication and computation bottlenecks across strategies.
- A single-GPU baseline to provide a solid comparison point for evaluating the effectiveness of distributed training.

Through this analysis, we aim to offer clearer guidance on when and why hybrid parallelism is an appropriate strategy—particularly when the model exceeds the capacity of a single GPU. To our knowledge, no prior work has systematically analyzed the impact of hybrid parallelism under heterogeneous GPU clusters for DR classification.

2. METHOD

2.1. Dataset dan preprocessing

The dataset used in this study consists of human retinal images obtained from Aravind Eye Hospital in India, released by the Asia Pacific Tele-Ophthalmology Society (APTOS) as part of the APTOS 2019 Blindness Detection competition on the Kaggle platform [10]. Each image is labeled according to the severity level of DR ranging from 0 to 4: No DR (0), Mild (1), Moderate (2), Severe (3), and Proliferative DR (4). Figure 1 illustrates visual examples for each class [11].

The dataset comprises 3,662 images with an imbalanced class distribution as shown in Table 1. Prior to preprocessing, the dataset was split into a training set (80%) and a testing set (20%), a commonly used ratio to preserve evaluation integrity. The preprocessing pipeline was applied differently to each subset. To address class imbalance, data augmentation—including random oversampling to equalize the number of samples per class—was applied only to the training set. The test set was left untouched to reflect real-world data distribution [12]-[14].

Figure 1. Retinal images representing each DR severity class

Table 1. Class distribution of Aptos 2019 dataset

DR severity	Training data		Testing data
	Before augmentation	After augmentation	
0 – No DR	1444	1444	361
1 – Mild	296	1444	74
2 – Moderate	799	1444	200
3 – Severe	154	1444	39
4 – Proliferative	236	1444	59
Total	2999	7220	733

The preprocessing process consists of four main steps:

a) Cropping black borders to remove non-informative regions that may interfere with learning.
b) Applying the Benjamin Graham normalization technique to reduce illumination variance across images.
c) Resizing the longest side of each image to 512 pixels while preserving the aspect ratio.
d) Applying square padding to standardize the input dimensions.

Additionally, the training data was augmented using random transformations (horizontal/vertical flips, rotations, brightness/contrast adjustments) to improve model generalization and prevent overfitting.

2.2. Deep learning model

This study employs the DenseNet-121 architecture, a convolutional neural network (CNN) sourced from the PyTorch torchvision library. Densely connected convolutional networks (DenseNet) introduces dense connectivity, where each layer receives input from all previous layers, encouraging feature reuse and mitigating the vanishing gradient problem [15]. This dense connectivity is especially effective for medical images, ensuring that low-level features (e.g., fine vessel structures or texture) are accessible to deeper layers and aiding in the detection of subtle anomalies. The model was initialized with pretrained ImageNet weights and fine-tuned on the APTOS dataset for this specific classification task.

2.3. Kubernetes cluster

For distributed training orchestration, this study uses a Kubernetes cluster, which automates deployment, scaling, and container management. We adopted TorchX to simplify PyTorch job definitions and used Volcano as the batch scheduler to efficiently allocate GPU resources. The hardware configuration of the heterogeneous cluster is summarized in Table 2.

Table 2. Kubernetes cluster hardware configuration

Node	Role/node name	GPU configuration	VRAM per GPU
Node-1	Master node/PCR	-	-
Node-2	Compute node/CN1	2× NVIDIA RTX 3070	8 GB
Node-3	Compute node/CN2	1× NVIDIA RTX3090, 1× NVIDIA RTX3090ti	24 GB
Node-4	Compute node/CN3	2× NVIDIA Quadro RTX 6000	24 GB

2.4. Data parallelism

To accelerate training, data parallelism was implemented using PyTorch's torch.nn.parallel. Distributed data parallel (DDP) module with AllReduce synchronization architecture [16], [17]. The model is replicated on each GPU, and each replica processes a different subset of the data in parallel. After the backward pass, gradients are aggregated via AllReduce and averaged across replicas. Each replica then updates the model weights independently using the averaged gradients, ensuring model consistency at the end of every iteration [16], [18], [19].

2.5. Pipeline parallelism

We also implemented pipeline parallelism, enabling training of models that exceed single-GPU memory capacity. This was achieved using the torch.distributed.pipeline.sync module in PyTorch. The DenseNet-121 model was divided into two stages, executed sequentially, with the split performed after transition layer 2 to balance compute and memory loads across the two GPUs. During training, each input batch was split into micro-batches, which flowed through the pipeline to maximize GPU utilization and enable training of larger models [20]–[22]. Figure 2 shows the DenseNet-121 architecture divided into two parts. Stage 1 (on GPU 0) contains the initial layers up to transition 2 as shown in Figure 2(a). Stage 2 (on GPU 1) contains dense block 3 up to the classification layer as shown in Figure 2(b).

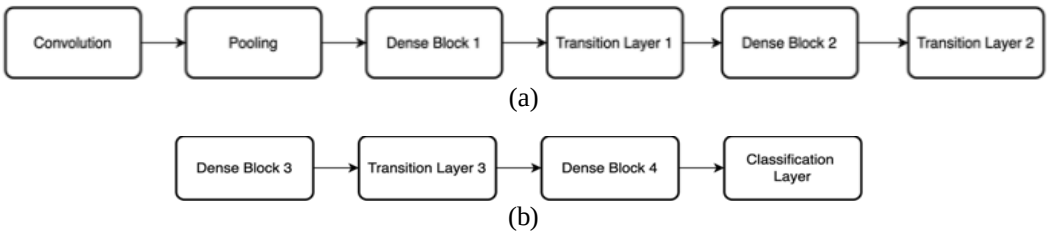


Figure 2. DenseNet-121 in, (a) Stage 1: from input to transition layer 2 and (b) Stage 2: from dense block 3 to classification

2.6. Hybrid parallelism

Hybrid parallelism combines both data and pipeline parallelism as shown in Figure 3. The pipelined model is replicated across all compute nodes (data parallelism), and within each node, the two pipeline stages are distributed across two GPUs (pipeline parallelism). Gradient synchronization via AllReduce (data parallelism) occurs after each full forward-backward pass cycle within the pipeline. This approach effectively distributes computational load and memory usage across the cluster [23]–[25].

This architecture as shown in Figure 3 effectively distributes both computation and memory usage across the cluster. Each node contains a two-stage pipeline replicated across nodes, enabling concurrent data processing while training a model that exceeds single-GPU memory limits.

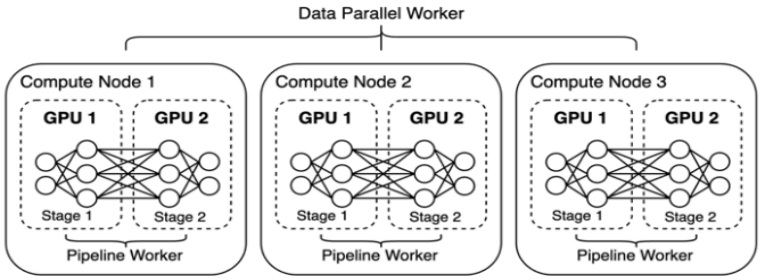


Figure 3. Hybrid parallel training across 3 compute nodes (pipeline + data parallelism)

2.7. Training

All training strategies used the same model architecture (DenseNet-121) initialized with pretrained ImageNet weights. The Adam optimizer with a learning rate of 0.001 and cross-entropy loss function was used throughout. To ensure fairness, weight updates were consistently performed every 30 samples across all training strategies to balance GPU memory usage and avoid out-of-memory (OOM) errors.

The global batch size was fixed at 30 for all strategies. For single GPU and pipeline parallelism, the full batch size of 30 was used. In data parallel and hybrid strategies, the batch size per replica was 30/N, where N is the number of data replicas. In pipeline parallelism, each mini-batch was further split into 5 micro-batches, resulting in 6 samples per forward/backward pass, compatible with fixed update intervals and pipeline execution. To clarify the configuration differences in each experimental scenario, Table 3 presents a summary of the training parameters used.

Table 3. Training configuration

Strategy	# GPUs	Batch size per replica	Global batch size	Update interval
Single GPU	1	30	30	30 samples
Data parallelism	2	15 (30/2)	30	30 samples
Pipeline parallelism	2	30	30	6 samples per microbatch
Hybrid parallelism	6	5 (30/6)	30	6 samples per microbatch

As shown in Table 3, the key point of this experimental design is to maintain a consistent global batch size of 30 for all strategies. This ensures that the total volume of data processed before each model weight update is the same, thus making the comparison of speed and performance between strategies fair and valid. Model performance was evaluated using three metrics: accuracy, weighted F1-score, and quadratic weighted kappa (QWK). Among these, QWK was considered the primary metric, as it is specifically designed for ordinal classification tasks like DR grading. QWK penalizes severe misclassifications more heavily (e.g., predicting "Severe" as "No DR") than milder ones (e.g., predicting "Severe" as "Proliferative").

3. RESULTS AND DISCUSSION

This section presents and analyzes the experimental results from two main aspects: the training time efficiency of different parallelism strategies and the classification performance of the resulting models.

3.1. Training time efficiency analysis

We first examine ideal speedup performance, followed by an investigation of performance degradation in heterogeneous and multi-node scenarios.

3.1.1. Performance in homogeneous environments: ideal scenario

The initial experiments were conducted on node CN3, which has two identical GPUs (NVIDIA Quadro RTX 6000), to establish an ideal performance baseline. As shown in Table 4, data parallelism achieved an almost linear speedup of 1.9×, demonstrating high efficiency as expected. Pipeline parallelism yielded a moderate speedup of 1.19×, though less efficient due to pipeline bubbles and synchronization delays. Interestingly, the hybrid parallelism strategy that involved three nodes resulted in a slight slowdown (0.99×) compared to the single-GPU baseline, an anomaly explored further in subsequent sections.

Table 4. Speedup on a homegeneous node (CN3)

Strategy	Node and GPU configuration	Speedup
Single GPU	CN3 (1× NVIDIA Quadro RTX 6000)	1×
Data parallelism	CN3 (2× NVIDIA Quadro RTX 6000)	1.9×
Pipeline parallelism	CN3 (2× NVIDIA Quadro RTX 6000)	0.19×
Hybrid parallelism	CN1 (2× NVIDIA RTX 3070)	0.99×
	CN2 (NVIDIA RTX 3090 and NVIDIA RTX 3090 Ti)	
	CN3 (2× NVIDIA Quadro RTX 6000)	

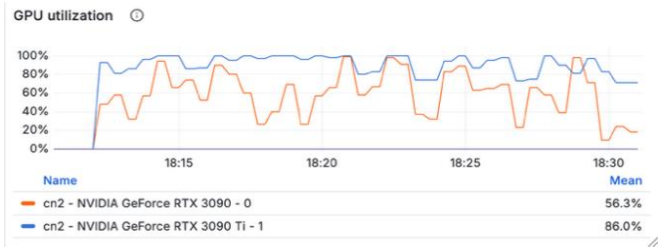
3.1.2. Impact of hardware heterogeneity

The next scenario was run on node CN2, which contains two heterogeneous GPUs (RTX 3090 and RTX 3090 Ti). As presented in Table 5, pipeline parallelism experienced a significant degradation in speed with a slowdown of 0.86×, contrary to expectations. This performance degradation is visually evident from the GPU utilization profile in Figure 4, where large fluctuations are observed. The faster GPU frequently waits for the slower GPU to finish its pipeline stage, creating pipeline bubbles and reducing efficiency as shown in Figures 4(a) and 4(b).

Table 5. Speedup on a heterogeneous node (CN2)

Strategy	Node and GPU configuration	Speedup
Single GPU	CN2 (NVIDIA RTX 3090 only)	1×
Data parallelism	CN2 (NVIDIA RTX 3090 and NVIDIA RTX 3090 Ti)	1.56×
Pipeline parallelism	CN2 (NVIDIA RTX 3090 and NVIDIA RTX 3090 Ti)	0.86×
Hybrid parallelism	CN1 (2× NVIDIA RTX 3070)	0.67×
	CN2 (NVIDIA RTX 3090 and NVIDIA RTX 3090 Ti)	
	CN3 (2× NVIDIA Quadro RTX 6000)	

(a)



(b)

Figure 4. GPU utilization in pipeline parallelism on, (a) CN3 and (b) CN2

In the hybrid configuration, this problem becomes even worse as heterogeneity exists across multiple nodes, resulting in highly unstable GPU utilization patterns and further degradation. Figure 5 shows this worst GPU utilization situation.

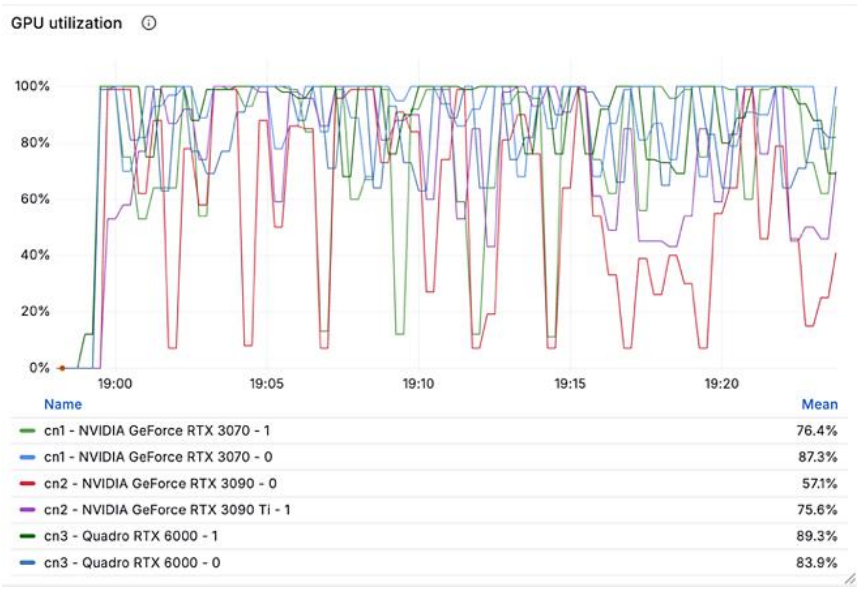


Figure 5. GPU utilization in hybrid parallelism across CN1–CN3

3.1.3. Communication bottlenecks in multi-node setup

The noticeable slowdown in multi-node strategies (hybrid and inter-node DP) prompted a deeper look into communication overhead. Table 6 compares intra-node data parallelism (on CN3) and inter-node DP across CN1–CN3. Results indicate that inter-node data parallelism is up to 27% slower than intra-node data parallelism due to synchronization delays. To confirm the cause, we measured GPU communication latency across 1,000 trials, as summarized in Table 7. Intra-node latency via PCIe is sub-millisecond, while inter-node latency via Ethernet exceeds 18 ms, more than 20× slower.

Table 6. Speedup comparison: intra-node Vs. inter-node

Node and GPU configuration	Speedup
CN3 (2× NVIDIA Quadro RTX 6000)	1×
CN2 (NVIDIA RTX 3090 Ti) and CN3 (NVIDIA Quadro RTX 6000)	0.73×
CN1 (NVIDIA RTX 3070), CN2 (NVIDIA RTX 3090 Ti), and CN3 (NVIDIA Quadro RTX 6000)	0.78×

Table 7. GPU communication latency (average of 1,000 trials)

Comm. pattern	From	To	Avgerage time (ms)
Within a node	CN1	CN1	0.774
	CN2	CN2	0.774
	CN3	CN3	0.397
Across nodes	CN1	CN2	18.917
	CN2	CN3	18.908
	CN3	CN1	18.150

3.1.4. Verifying the hypothesis by reducing nodes

To verify that inter-node communication latency was the main bottleneck, we reduced the hybrid setup from three to two nodes (removing CN1). As shown in Table 8, performance improved by 5%, confirming that network latency was a dominant limiting factor.

Table 8. Speedup after node reduction in the hybrid setup

Node and GPU configuration	Speedup
CN1 (2× NVIDIA RTX 3070), CN2 (NVIDIA RTX 3090 and NVIDIA RTX 3090 Ti), and CN3 (2× NVIDIA Quadro RTX 6000)	1×
CN2 (NVIDIA RTX 3090 and NVIDIA RTX 3090 Ti) and CN3 (2× NVIDIA Quadro RTX 6000)	1.05×

3.2. Classification performance analysis

Following the efficiency analysis, we evaluated classification performance. Each strategy was trained for 30 epochs, repeated 10 times, with identical initialization and shuffled data in each epoch. Metrics are averaged over the final epoch from each run as shown in Table 9.

Table 9. Classification performance (10-run average)

Strategy	Metrics		
	Validation accuracy	QWK *)	Weighted F1-score
Single	0.774	0.868	0.781
Data parallelism	0.737	0.861	0.749
Pipeline parallelism	0.776	0.869	0.781
Hybrid parallelism	0.777	0.874	0.784

*) *Quadratic Weighted Kappa*

The results in Table 9 reveal an intriguing finding: the hybrid parallel strategy (Data parallel + Pipeline parallel) achieved the highest classification performance across all key metrics, despite being the least efficient in terms of training time. Models trained using single-GPU and data parallel strategies demonstrated very competitive and closely similar performance. In contrast, the pipeline parallel strategy consistently yielded the lowest performance among all configurations.

The superior performance of the hybrid-trained model can be attributed to several hypothetical factors. First, training in a highly heterogeneous environment may serve as a form of implicit regularization, preventing the model from overfitting to the computational characteristics of a single GPU architecture. Second, the aggregation of gradients from replicas running at slightly different speeds and hardware configurations may introduce beneficial noise into the weight update process—similar to noise-based regularization techniques. These observations suggest a complex trade-off between training efficiency and the potential regularization benefits introduced by a diverse, distributed system architecture. Such findings emphasize that system-level heterogeneity, which is often viewed as a limitation, can act as an advantage for enhancing model generalization under certain conditions.

3.3. Discussion

3.3.1. Hybrid strategy's failure to improve speedup

The first and most prominent finding of this study is that the hybrid strategy, which combines data and pipeline parallelism, failed to deliver the expected speedup. Specifically, the results presented in Table 4 and Table 5 indicate that the hybrid strategy executed on six GPUs was no faster (0.99× speedup vs. the Quadro RTX 6000) or was even significantly slower (0.67× speedup vs. the RTX 3090) when compared to training on a single GPU. Conversely, the highest speedup was achieved by a simple data parallel strategy on two GPUs, which yielded a speedup between 1.56× and 1.9×. This finding directly addresses the finding regarding computational efficiency.

This failure can be attributed to several interrelated factors. First, hardware heterogeneity proved to be a primary obstacle for pipeline parallelism. As shown in Table 5, pipeline parallelism on a heterogeneous node (CN2) resulted in a slowdown (0.86× speedup). This is supported by the visualization in Figure 4, which reveals unstable GPU utilization. Faster GPUs were forced to wait for slower ones, creating significant pipeline bubbles and negating the potential benefits of parallelization.

Second, the inter-node communication bottleneck emerged as the dominant limiting factor. The analysis in Tables 6 and 7 quantitatively substantiates this: inter-node communication latency over the external network was more than 20 times slower than intra-node communication via PCIe. This communication overhead occurs during each gradient synchronization cycle, drastically reducing the efficiency of multi-node strategies such as the hybrid approach. This hypothesis was verified when the removal of one node from the hybrid cluster resulted in a 5% increase in speed (see Table 8), thereby isolating network latency as the principal cause.

3.3.2. Trade-off between efficiency and classification performance

Although computationally inefficient, the second key finding of this study is that the hybrid strategy consistently yielded the best classification performance. The results in Table 9 show that the hybrid model achieved the highest QWK score (0.8739), outperforming both the single-GPU (0.8675) and the data parallel (0.8689) strategies.

This phenomenon highlights a complex trade-off between computational efficiency and model accuracy. The superior performance of the hybrid model can be explained by the concept of implicit

regularization. The training process in a highly heterogeneous environment with varying computational speeds, GPU architectures, and communication latencies, likely introduces a form of beneficial "noise" into the gradient update process. The aggregation of gradients from these slightly asynchronous replicas may act as a regularizer, helping the model to escape sharp local minima and converge to a more generalizable solution. This aligns with other research demonstrating that variations in distributed training processes can enhance a model's generalization capabilities [17]. Conversely, the pipeline parallel strategy, which suffered the most from inefficiency (i.e., idle time), produced the lowest classification performance, indicating that not all types of inefficiency confer a regularization benefit. This finding, indicating that the "fastest" parallelization strategy does not necessarily yield the "best" model, which is a crucial consideration for deep learning practitioners.

4. CONCLUSION

The primary conclusion is that in a heterogeneous hardware environment, the hybrid strategy is inefficient for accelerating training time. Bottlenecks caused by inter-node communication latency and workload imbalances due to GPU heterogeneity significantly hinder potential speedup. A simple data parallel strategy on a single homogeneous node proved to be a far more efficient approach. However, paradoxically, the model trained with this inefficient hybrid strategy achieved the highest classification performance. This phenomenon suggests that system heterogeneity may act as a form of implicit regularization, indirectly enhancing the model's generalization capability.

The implication of these findings is that deep learning practitioners must consider more than just throughput or speedup when designing distributed training workflows. The choice of parallelization strategy can directly impact final model accuracy, and the most computationally efficient architecture may not be the one that yields the most accurate model. Limitations of this study include the use of a single model architecture (DenseNet-121) and a specific dataset (APTOS 2019). The behavior of this trade-off may differ with other architectures, such as Transformers, or with different data modalities.

Future research directions could focus on developing more intelligent and heterogeneity-aware schedulers or synchronization mechanisms. Further research could also systematically investigate the regularizing effects of varying levels of network latency and GPU heterogeneity to leverage them intentionally, rather than as a side effect, for improving model performance.

ACKNOWLEDGMENTS

The authors would like to express their sincere gratitude to Mr. Kardi Teknomo for his valuable input and insightful discussions during the preparation of this manuscript. His constructive suggestions significantly contributed to the refinement of our ideas and the overall quality of this research.

FUNDING INFORMATION

This research project was supported by National Research and Innovation Agency (or BRIN – Badan Riset and Inovasi Nasional) of the Indonesian government through the RIIM Grant No. 194/IV/11/2023 and No. 2679/UKP/2023. The authors express sincere gratitude for the support.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Indar Sugiarto	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓
Djoni Haryadi Setiabudi	✓	✓	✓	✓	✓	✓		✓	✓	✓		✓		
Darrell Cornelius Rivaldo		✓	✓			✓		✓	✓	✓	✓	✓		
Taweesak Kijkanjanarat				✓	✓		✓			✓				
Resmana Lim							✓			✓			✓	✓

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

The data that support the findings of this study are available on request from the corresponding author, [IS]. The data, which contain information that could compromise the privacy of research participants, are not publicly available due to certain restrictions.

REFERENCES

[1] H. I. Sarker, "Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions," *SN Computer Science*, vol. 2, no. 420, 2021, doi: 10.1007/s42979-021-00815-1.

[2] P. Taylor, "Data growth worldwide 2010-2025," Statista, Nov. 16, 2023. [Online]. Available: https://www.statista.com/statistics/871513/worldwide-data-created/. [Accessed: Oct. 29, 2024].

[3] M. Pandey, M. Fernandez, F. Gentile, O. Isayev, A. Tropsha, A. C. Stern, and A. Cherkasov, "The transformational role of GPU computing and deep learning in drug discovery," *Nature Machine Intelligence*, vol. 4, pp. 211-221, 2022, doi: 10.1038/s42256-022-00463-x.

[4] J. H. Park, G. Yun, C. M. Yi, N. T. Nguyen, S. Lee, J. Choi, S. H. Noh, and Y.-R. Choi, "Heterogeneous GPU clusters through integration of pipelined model parallelism and data parallelism," in *2020 USENIX Annual Technical Conference*, 2020.

[5] M. A. Maruf, A. Azim, N. Auluck, and M. Sahi, "Optimizing DNN training with pipeline model parallelism for enhanced performance in embedded system," *Journal of Parallel and Distributed Computing*, vol. 190, p. 104890, 2024, doi: 10.1016/j.jpdc.2024.104890.

[6] C. N. Thompson, K. Greenewald, K. Lee, and F. G. Manso, "The computational limits of deep learning," *arXiv preprint arXiv:2007.05558*, 2022, doi: 10.48550/arXiv.2007.05558.

[7] M. S. Patil and S. Chickerur, "Study of data and model parallelism in distributed deep learning for diabetic retinopathy," *Procedia Computer Science*, vol. 218, pp. 2253-2263, 2023, doi: 10.1016/j.procs.2023.01.201.

[8] S. Jan, I. Ahmad, S. Karim, Z. Hussain, M. Rehman, and M. A. Shah, "Status of diabetic retinopathy and its presentation patterns in diabetics at ophthalmology clinics," *Journal Postgraduate Medical Institute*, vol. 32, no. 1, pp. 24-27, 2018.

[9] S. Qummar, F. G. Khan, S. Shah, A. Khan, S. Shamshirband, and Z. U. Rehman, "A deep learning ensemble approach for diabetic retinopathy detection," *IEEE Access*, vol. 7, pp. 150530-150539, 2019, doi: 10.1109/ACCESS.2019.2947484.

[10] S. A. Patil, M. S. Patil, S. Giraddi, S. Chickerur, V. M. Boormane, and G. Gamanagatti, "Pipeline parallelism in distributed deep learning for diabetic retinopathy classification," *Procedia Computer Science*, vol. 215, pp. 393-402, 2022, doi: 10.1016/j.procs.2022.12.041.

[11] N. C. Frey *et al.*, "Benchmarking resource usage for efficient distributed deep learning," *In 2022 IEEE High Performance Extreme Computing Conference (HPEC)*, 2022, doi: 10.1109/HPEC55821.2022.9926375.

[12] M. A. K. Raiaan, K. Fatema, I. U. Khan, S. Azam, M. R. U. Rashid, and M. S. H. Mukta, "A lightweight robust deep learning model gained high accuracy in classifying a wide range of diabetic retinopathy images," *IEEE Access*, vol. 11, pp. 42361-42388, 2023, doi: 10.1109/ACCESS.2023.3272228.

[13] M. Karthik, and S. Dane, "APTOS 2019 blindness detection," Kaggle, 2019. [Online]. Available: https://www.kaggle.com/competitions/aptos2019-blindness-detection. [Accessed: Nov. 3, 2024].

[14] C. Mohanty, S. Mahapatra, B. Acharya, F. Kokkoras, V. C. Gerogiannis, I. Karamitsos, and A. Kanavos, "Using deep learning architectures for detection and classification of diabetic retinopathy," *Sensors*, vol. 23, no. 12, p. 5567, 2023, doi: 10.3390/s23125726.

[15] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, doi: 10.1109/CVPR.2017.243.

[16] S. Li *et al.*, "PyTorch distributed: experiences on accelerating data parallel training," *arXiv preprint arXiv:2006.15704*, vol. 13, no. 12, pp. 3005-3018, 2020, doi: 10.14778/3415478.3415530.

[17] S. J. Park, J. Fried, S. Kim, M. Alizadeh, and A. Belay, "Efficient strong scaling through burst parallel training," *Proceedings of Machine Learning and Systems*, 2021.

[18] D. Xiong, L. Chen, Y. Jiang, D. Li, S. Wang, and S. Wang, "Revisiting the time cost model of allreduce," *arXiv preprint arXiv:2409.04202*, 2024.

[19] S. Kim *et al.*, "Parallax: sparsity-aware data parallel training of deep neural networks," *In Proceedings of the Fourteenth EuroSys Conference*, 2018, doi: 10.1145/3302424.3303957.

[20] F. Brakel, U. Odyurt, and A. L. Varbanescu, "Model parallelism on distributed infrastructure: a literature review from theory to LLM case-studies," *arXiv preprint arXiv:2403.03699*, 2024.

[21] D. Narayanan *et al.*, "Efficient large-scale language model training on GPU clusters using megatron-LM," *In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, doi: 10.1145/3458817.3476209.

[22] Y. Huang *et al.*, "GPipe: efficient training of giant neural networks using pipeline parallelism," *In Proceedings of the 33rd International Conference on Neural Information Processing Systems (NeurIPS 2019)*, Vancouver, BC, Canada, 2019, pp. 12826-12837.

[23] D. Narayanan *et al.*, "PipeDream: generalized pipeline parallelism for DNN training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, (pp. 1-15), 2019, doi: 10.1145/3341301.3359646.

[24] S. Li, and T. Hoefler, "Chimera: efficiently training large-scale neural networks with bidirectional pipelines", in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1-14, November, 2021, doi: 10.1145/3458817.3476145.

[25] S. Zhao, "V pipe: a virtualized acceleration system for achieving efficient and scalable pipeline parallel dnn training," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 3, pp. 489-506, 2021, doi: 10.1109/TPDS.2021.3094364.

BIOGRAPHIES OF AUTHORS

Dr.-Ing, Indar Sugiarto, S.T., M.Sc., received M.Sc. degree in automation engineering from the University of Bremen, Germany, and the Ph.D. degree in electrical and information technology from The Technical University of Munich, Germany. Currenty, he is an associate professor at the Department of Electrical Engineering of Petra Christian University. He is also the head of the Center for Artificial Intelligence and Computational Modeling Studies in Petra Christian University. His current research area focuses on AI application development as well as computation modeling of dynamic systems. He can be contacted at email: indi@petra.ac.id.

Djoni Haryadi Setiabudi received M.Eng. degree in computer science from the Asian Institute of Technology, Thailand. Currenty, he is an associate professor in Informatics Department at Petra Christian University. His current research area focuses on AI application development. He can be contacted at email: djonihs@petra.ac.id.

Darrell Cornelius Rivaldo received the B.Sc. degree in informatics from Petra Christian University, Indonesia. He is currently a cohort at the Apple Developer Academy, Universitas Ciputra Surabaya, where he focuses on software engineering, mobile application development, and the integration of AI into innovative applications. His interests include scalable application development, user-centered design, and leveraging technology to create impactful digital solutions. He can be contacted at email: c14210025@john.petra.ac.id.

Asst. Prof. Dr. Taweesak Kijkanjanarat received the M.S. degree in Electrical Engineering from Columbia University, U.S.A. And the Ph.D. degree in Electrical Engineering from NYU Tandon School of Engineering, U.S.A. He is currently an assistant professor in the Department of Electrical and Computer Engineering, Thammasat University, Thailand. He can be contacted at email: taweesak@engr.tu.ac.th.

Ir. Resmana Lim, M.Eng., received M.Eng. degree in information technology from the Asian Institute of Technology, Thailan. Currenty, he is an associate professor in Electrical Engineering at Petra Christian University. He also teaches in the Professional Engineer Education Program (PPI). Currently, he serves as the Head of the Center for Information and Communication Technology, focusing on IT infrastructure to support the university's computing needs. He can be contacted at email: resmana@petra.ac.id.