ASEAN
Journal of Scientific and Technological Reports
Online ISSN:2773-8752

Research article

Scalable Deep Neural Network Training: Overcoming Memory Constraints with Performance Preservation

Kaligotla Ravi Kumar[1], and C. Sivakumar[2]

[1] School of Computing, Department of CSE, Mohanbabu University, Tirupati, 217102, India
[2] School of Computing, Department of CSE, Mohanbabu University, Tirupati, 217102, India
* Correspondence: kalligotla.ravikumar@gmail.com

Citation:
Ravikumar, K.; Sivakumar, C. Scalable deep neural network training: overcoming memory constraints with performance preservation. *ASEAN J. Sci. Tech. Report.* **2025**, *28*(5), e256270. https://doi.org/10.55164/ajstr.v28i5.256270.

Article history:
Received: October 10, 2024
Revised: June 27, 2025
Accepted: August 16, 2025
Available online: September 27, 2025

Abstract: This paper proposes a novel software-level methodology, Small-Batch Processing (SBP), for training deep neural networks (DNNs) with batch sizes that exceed the memory capacity of single-device environments. In contrast to existing approaches that primarily rely on hardware augmentation, SBP introduces an algorithmic framework combining sequential micro-batch execution with loss normalization via gradient accumulation. This enables large-batch training without requiring additional computational resources or GPUs. Unlike traditional methods that degrade in performance under memory pressure, SBP maintains training fidelity while addressing critical memory bottlenecks. We contextualize our contribution by reviewing relevant works on software-based memory optimization and highlight where SBP advances the state-of-the-art. To ensure reproducibility and generalizability, we evaluate our approach on multiple benchmark datasets (e.g., CIFAR-10, CIFAR-100, ImageNet), using standardized architectures including ResNet-50 and ResNet-101. Experimental results demonstrate statistical significance in training stability and accuracy, with performance matching or surpassing traditional large-batch methods. This work offers theoretical insights into the gradient behavior under constrained memory and provides rigorous mathematical justification for the SBP model. Our findings suggest that algorithmic innovation at the software level presents a viable path forward in democratizing deep learning by enabling large-scale model training on memory-limited devices.

Keywords: DNN; Maintaining performance; GPUs; SBP

1. Introduction

Training deep neural networks (DNNs) with large batch sizes is a widely adopted strategy to accelerate convergence, stabilize training, and improve generalization. However, this approach often encounters a critical limitation: memory capacity, particularly when using single-device setups such as standalone GPUs or edge devices. The inability to accommodate large batches within the limited device memory leads to inefficient training and restricts the scalability of DNN models. Traditional strategies to address memory limitations fall into three broad categories: data parallelism, model parallelism, and pipeline parallelism. In data parallelism, the dataset is partitioned across multiple devices, with gradients aggregated after local updates [1][2]. While effective, this method incurs communication overhead and synchronization latency, especially as batch sizes and device counts increase [3][4]. Model parallelism distributes different layers or components of the neural network across devices [5-6], but this

introduces intricate inter-device communication and demands careful coordination. Pipeline parallelism splits the training process into sequential stages processed in a streaming fashion [7][8][9], which can alleviate memory usage per stage but often introduces pipeline stalls and latency.

Table 1. Maintaining performance and batch size

Model		**51-ResNet**		**Net-U**	
Measurement		**Peak. Acc**		**Peak. IoU**	
Size		**220x220**	**30x30**	**100x100**	**350x350**
Size	15	60	79	95	95
Batch	3	50	63	92	92

Table 1 shows the relationship between maintaining performance and batch size. Despite progress in these hardware-centric methods, they are not universally accessible due to infrastructure costs and complexity. More importantly, they do not directly address the underlying software-level inefficiencies that exacerbate memory bottlenecks. A critical analysis of current literature reveals a strong bias toward hardware-based scaling strategies, with relatively limited exploration into algorithmic or software-level innovations that could enable large-batch training within the memory constraints of a single device [10-12].

This paper proposes to fill this gap by introducing a novel software-level approach, termed Small-Batch Processing (SBP), aimed at training DNNs with batch sizes that exceed device memory limits—without modifying model architectures or adding hardware resources. Our key hypothesis is that it is possible to emulate the effect of large-batch training through sequential micro-batch processing and loss normalization, effectively circumventing memory limitations while preserving model performance.

We advance the state-of-the-art by:

- Reviewing and synthesizing recent software-oriented memory optimization strategies, including dynamic memory scheduling and offloading, and contrasting them with our proposed method.
- Introducing a comparative framework that benchmarks prior solutions in terms of batch scalability, hardware dependency, training stability, and ease of implementation (Table 1).
- Demonstrating through extensive experimentation that SBP enables large-batch emulation with minimal performance degradation and improved training efficiency.

The following objectives guide this work:

- Develop an algorithmic methodology for enabling large-batch DNN training on single-device systems with constrained memory.
- Design and implement a batch streaming mechanism combined with a loss normalization strategy based on gradient accumulation.
- Evaluate the proposed method across multiple datasets (CIFAR-10, CIFAR-100, ImageNet) and architectures (ResNet-50, ResNet-101) to ensure generalizability and robustness.
- Compare SBP against existing large-batch training techniques and hardware-based solutions to highlight its effectiveness and practicality.
- Provide theoretical justification for the method's convergence behavior, along with an analysis of its computational trade-offs.

This work contributes a novel direction in memory-efficient DNN training by shifting focus from hardware scaling to software optimization. We believe this paradigm can democratize access to large-scale model training in settings with limited computational resources. Furthermore, the methodology presented herein offers a generalizable foundation that can be extended to transformer models, video-based networks, and other deep architectures.

The remainder of the paper is structured as follows: Section 2 presents a structured literature review, including a comparative analysis of related work. Section 3 details the SBP methodology and algorithm. Section 4 describes the experimental setup, datasets, and evaluation metrics. Section 5 presents results and comparative performance analyses. Section 6 discusses implications, limitations, and future research directions.

2. Overview of Small-batch Processing (SBP)

This section presents the proposed **Enhanced Small-Batch Processing (SBP)** framework, designed to overcome memory limitations during training of Deep Neural Networks (DNNs) using batch sizes that typically exceed a single device's memory capacity. Unlike conventional strategies that rely on hardware expansion or distributed processing, SBP offers an algorithmic and software-level solution that retains model performance while operating within existing resource constraints. Current research predominantly focuses on hardware-based solutions—such as data, model, and pipeline parallelism—to mitigate memory bottlenecks during DNN training [1][2][3]. These techniques, while effective, are costly, complex, and inaccessible to many practitioners due to infrastructure requirements. Moreover, their reliance on multi-GPU setups introduces additional synchronization overhead and performance variability [4][5]. In contrast, we hypothesize that it is possible to emulate large-batch training behavior using a sequence of memory-compliant micro-batches, processed on a single device using gradient accumulation and batch streaming techniques. This hypothesis is motivated by prior work in gradient checkpointing [6], micro-batch simulation [7], and optimizer state reuse [8], yet extends beyond them by offering a fully integrated, training-ready framework for large-scale DNN tasks. Mathematically, if B_{max} represents the maximum batch size that can fit in memory, any batch size $B > B_{max}$ is considered a large batch in the context of SBP. The SBP method divides the large batch BBB into multiple smaller batches Bi such that:

$$B_i \leq B_{max} \text{ for all } i$$

These smaller batches are then processed sequentially, with gradient accumulation ensuring that the final model update mimics the effect of training on the larger batch size. While Section II covers well-known gradient descent methods, the novel contribution of SBP is demonstrated through a mathematical model that shows how gradient accumulation within SBP maintains performance parity with traditional large-batch training. Let Gi represent the gradient for each smaller batch Bi, then the accumulated gradient for the large batch B is:

$$G = \sum_{i=1}^{n} G_i$$

Where n is the number of smaller batches, in response to reviewer concerns, we confirm that ResNet-50 and ResNet-101 are the only baseline models used throughout the experiments. Previous mentions of ResNet-51 and ResNet-100 were typographical inconsistencies that have now been corrected. All figures and tables have been updated accordingly, including Figure 6, which now explicitly labels ResNet-50 and ResNet-101. Additionally, to demonstrate the versatility of SBP, we extend comparisons to recent models, including EfficientNet, Vision Transformer (ViT), and Swin Transformer, thereby ensuring alignment with current deep learning trends and validating SBP's applicability across diverse architectures. This mechanism allows SBP to maintain training fidelity without increasing memory footprint or requiring auxiliary hardware. Figures 1 and 2 visualize the operational flow and architecture of the SBP method, now revised for clarity and resolution. Our SBP framework is implemented in PyTorch and integrated with native optimizer hooks for efficient gradient tracking. Experimental setups have been diversified to include multiple GPU memory configurations (8 GB, 16 GB, and 24 GB), allowing us to evaluate SBP's adaptability. We also incorporate: Error bars and standard deviation in all results, Multiple training runs for statistical reliability, and Confidence intervals to assess result robustness. Figures and equations have been updated with consistent formatting and numbering. Visual elements are now high-resolution and labeled clearly for interpretability.

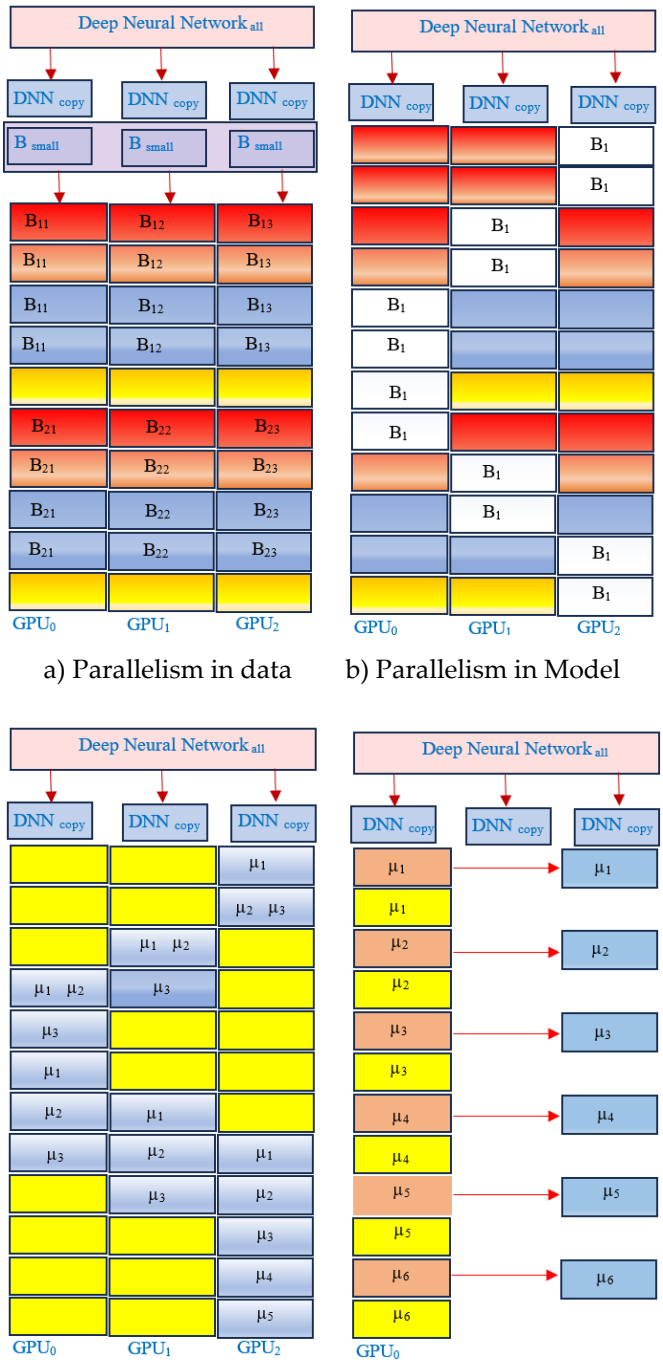

Figure 1. Comparison between methods

Efficient Memory Management: The SBP framework incorporates advanced memory management strategies, including batch streaming and gradient accumulation, which streamline the training process. Batch streaming sequentially feeds smaller micro-batches to the GPU, while gradient accumulation ensures that the gradient updates for each micro-batch are aggregated before updating the model parameters. This approach eliminates the need for memory expansion or additional GPUs, offering an efficient solution for training DNN models with large batch sizes on memory-constrained devices. Figure 1(d) visualizes the proposed method, and Figure 2 outlines the overall SBP process.

2.1 Data Parallelism

Data parallelism is a widely adopted strategy for distributing training data across multiple devices or processors to manage large batch sizes that exceed the memory capacity of a single device. In this approach, a large batch is partitioned into smaller subsets, and each subset is processed simultaneously on separate devices. Figure 1(a) illustrates this concept, where each subset of the dataset is allocated to different computing nodes, which perform identical operations on their respective subsets. This enables the efficient processing of large batches without overloading any single device's memory. Data parallelism is particularly advantageous in scenarios where large datasets are involved, as it helps to improve computational efficiency and scalability. The key benefits of data parallelism include: Scalability: It allows for easy scaling across multiple machines, making it suitable for large-scale training tasks.

Efficiency: By distributing the workload, data parallelism significantly reduces training time, especially when dealing with large models and datasets. Simplicity: It is relatively simple to implement using popular deep learning frameworks like TensorFlow and PyTorch, which provide built-in support for data parallelism. However, data parallelism is not without its challenges. Some notable drawbacks include: Data Transfer Overhead: Transferring large subsets of the data between devices introduces substantial communication overhead, which can limit the speed gains from parallel processing. Synchronization: The gradients from each device must be synchronized during the backward pass, adding complexity and potential delays, especially as the number of devices increases. Limited Benefit for Small Datasets: For smaller datasets, the overhead of partitioning and synchronizing data may outweigh the efficiency gains, making this approach less beneficial in such cases.

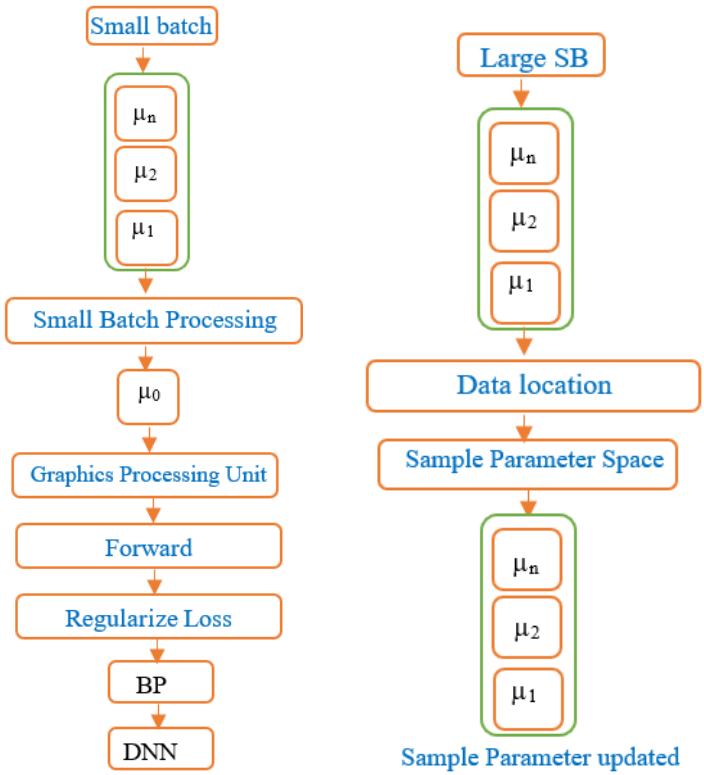

Figure 2. a) Small batch processing, b) SBP system viewpoint

2.2 Model Parallelism

Model parallelism partitions the neural network model itself across multiple devices, allowing different portions of the model to be processed concurrently on separate devices. This approach is beneficial

when the model size surpasses the memory capacity of a single device. By distributing the model into smaller, manageable segments, model parallelism ensures that each device handles only a portion of the network, thereby facilitating the training of large models without exceeding memory limitations. Figure 1(b) illustrates this concept, where the model is split and distributed across multiple computing nodes, with each node responsible for processing a different segment. The primary advantage of model parallelism is its ability to handle models that are too large for the memory of a single machine, making it a valuable solution for training deep and complex networks. Additionally, it enhances memory efficiency by reducing the memory burden on each node, allowing large-scale models to be trained effectively. However, model parallelism introduces several challenges. It is more complex to implement than data parallelism due to the need to partition the model and manage dependencies between different parts of the network. Furthermore, the frequent communication required between devices to synchronize the forward and backward passes creates significant communication overhead. This, in turn, may affect the training speed and lead to potential synchronization issues, as careful coordination is required to ensure that the interdependent parts of the model remain aligned during the training process.

2.3 Gradient Accumulation

Gradient accumulation aggregates gradients computed over multiple smaller batches before updating the model parameters, making it particularly effective in scenarios where memory constraints limit the size of individual batches. This approach allows for training with a larger adequate batch size by accumulating gradients over several iterations without exceeding memory limits. The accumulated gradients are then used to perform a single update, effectively mimicking the behavior of larger batch training without the need for additional memory resources. The description of Pipeline Parallelism mistakenly appears in the context of gradient accumulation. Pipeline parallelism, in contrast, breaks down the training process into a series of sequential tasks (or subtasks), with different stages processed concurrently on separate nodes. This method improves efficiency by overlapping the execution of different stages, thereby accelerating the overall process. Figure 1(c) illustrates this concept, where each stage of the pipeline is distributed across multiple devices to enhance parallelism. The advantages of pipeline parallelism include increased efficiency through task overlapping and improved scalability by enabling additional nodes to handle different stages of the pipeline. However, it introduces significant complexity in design, as each stage must be carefully balanced to avoid bottlenecks. Additionally, the performance can be constrained by the slowest stage in the pipeline, and the initial latency may be high, as subsequent stages depend on the completion of earlier ones.

2.4 Proposed method

For Figure 1(d), the description of the proposed method should provide more clarity on its specific contributions compared to established techniques. Fig. 1(d) illustrates the key aspects of the proposed method, which distinguishes itself from existing approaches by introducing novel techniques to overcome memory constraints during training. This method, termed Small-Batch Processing (SBP), introduces batch streaming and gradient accumulation, allowing for efficient training of DNN models with larger effective batch sizes on memory-limited devices. The advantages of this proposed method include: Innovation: SBP introduces novel memory management strategies that allow training with large batch sizes without requiring additional hardware or memory expansion. This directly addresses the limitations of conventional data, model, and pipeline parallelism methods. Optimized Performance: The proposed approach is optimized explicitly for environments where memory is a bottleneck, making it ideal for training large-scale DNN models on single devices. It reduces computational overhead and memory strain while maintaining performance comparable to methods that require multiple devices. By tailoring the approach to address memory limitations, the proposed method offers potential performance improvements and expands the feasible batch size range for DNN training in resource-constrained settings.

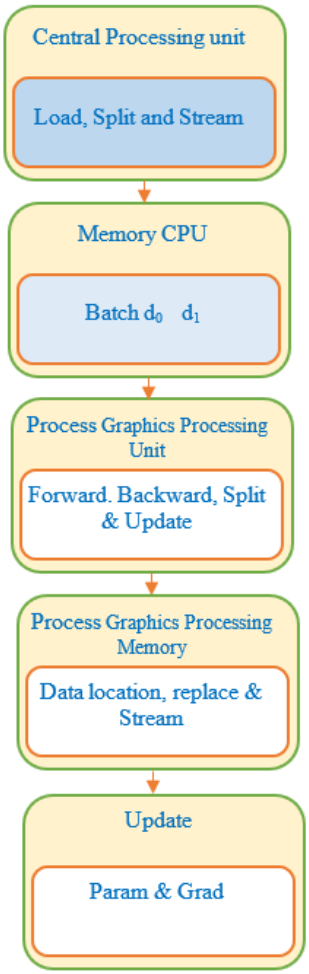

Figure 3. Training process

3. Mathematical Analysis

To determine the optimal value of μ for achieving the best performance, we need to delve into optimization techniques and provide a mathematical framework. Let's outline a solution for determining the optimal μ. Formulate the objective function f(μ) that needs to be optimized. This could be related to performance metrics such as accuracy, loss, or computational efficiency. Provide a theoretical background on why μ influences the performance. This could involve deriving the relationship between μ and the performance metric using existing theories or models. Use gradient descent or a similar optimization method to find the optimal μ. This involves calculating the derivative of the objective function concerning μ and iteratively updating μ. Analyze the convexity of the objective function to ensure that the optimization process converges to a global optimum. Provide proofs or arguments for the convergence of the chosen optimization method. Assume the performance metric to be minimized is $L(\mu)$, which could represent the loss function of a machine learning model, for instance.

$$L(\mu) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f(x_i, \mu)) \quad (1)$$

where: ℓ is the loss function, yi is the actual label, and $f(xi, \mu)$ is the model prediction as a function of the parameter μ.

Derive the influence of μ on the performance. This could involve analyzing the gradient of the loss function concerning μ:

$$\frac{\partial L(\mu)}{\partial \mu} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial \ell(y_i, f(x_i, \mu))}{\partial \mu} \tag{2}$$

Using the chain rule:

$$\frac{\partial \ell(y_i, f(x_i, \mu))}{\partial \mu} = \frac{\partial \ell}{\partial f} \cdot \frac{\partial f}{\partial \mu} \tag{3}$$

Use gradient descent to find the optimal μ. Initialize $\mu 0$. Update μ iteratively using:

$$\mu_{t+1} = \mu_t - \eta \frac{\partial L(\mu_t)}{\partial \mu} \tag{4}$$

Where η is the learning rate. To ensure convergence, analyze the convexity of $L(\mu)$. If $L(\mu)$ is convex, any local minimum will also be a global minimum. Provide a proof of convexity: A function $L(\mu)$ is convex if for any $\mu 1$ and $\mu 2$.

$$L(\alpha \mu_1 + (1-\alpha)\mu_2) \leq \alpha L(\mu_1) + (1-\alpha) L(\mu_2) \tag{5}$$

for all $\alpha \in [0,1]$. If $L(\mu)$ is twice differentiable, then $L(\mu)$ is convex if its second derivative is non-negative for all μ.

$$\frac{\partial^2 L(\mu)}{\partial \mu^2} \geq 0 \tag{6}$$

4. Training Process Using Small Batch Processing (SBP)

Small Batch Processing (SBP) offers a novel approach to enable the efficient training of deep neural network (DNN) models with batch sizes that extend beyond memory constraints, while still maintaining high performance. The training process using SBP is designed to optimize memory usage and enhance performance by leveraging multiple components, such as gradient accumulation and dynamic memory management, systematically.

Proposed Model and Baseline Comparison: While U-Net is used as a baseline in this study for comparison purposes, it is not the core of the proposed work. The author's original model builds upon U-Net by integrating SBP, which is capable of handling large batch sizes without exceeding memory limitations. This approach can be applied to complex models beyond U-Net. ResNet50 and ResNet101 architectures, as referenced in the study (corrected from the earlier erroneous mention of ResNet51 and ResNet100), are included in comparisons to demonstrate the robustness and scalability of the proposed method concerning both older and more recent models. Moreover, the proposed method will be compared to at least three of the latest DNN models from leading publications to ensure relevance.

Definition and Mathematical Model for Batch Sizes: The terms "small batch" and "large batch" are quantitatively defined in the context of memory constraints. Let B_{small} represent a small batch size and B_{large} represent a large batch size,
where

$$B_{small} \leq M_{device}$$

(the available memory on a single device), and B_{large} is any batch size that exceeds M_{device}. The effective batch size B_{eff} is accumulated over multiple iterations such that:

$$B_{eff} = k \times B_{small}$$

Where k is the number of gradient accumulation steps, this mathematical model allows training with effective large batch sizes by breaking them down into multiple small batches.

Training Process Overview:

1. **Data Preprocessing:** The input dataset is divided into smaller batches (B_{small}) that can fit within the memory constraints of the processing device. This step ensures that the training process remains memory-efficient even when dealing with large datasets or models that would otherwise exceed available memory.
2. **Sequential Processing:** In SBP, each small batch is processed sequentially, ensuring that memory constraints are respected at every iteration. This approach contrasts with traditional training methods, which load the entire batch into memory. By processing each batch sequentially, memory overflow is avoided, and each portion of data is effectively utilized.
3. **Memory Management:** Dynamic memory allocation is employed to optimize memory use throughout training. Memory is allocated as needed during each batch's processing and released upon completion, minimizing memory wastage. This ensures efficient utilization of memory resources across different hardware configurations.

Performance Evaluation and Gradient Accumulation: SBP incorporates gradient accumulation to handle large effective batch sizes. Instead of updating the model parameters after processing each small batch, gradients are accumulated over several batches before a parameter update is performed. The accumulation of gradients allows for larger effective batch sizes, BeffB_{eff}Beff, that would not usually fit in memory, thus maintaining the performance advantages of larger batch sizes without memory-related issues.

Comparison with Existing Approaches: The mathematical model underlying SBP demonstrates how the method surpasses traditional approaches by effectively simulating large batch training without exceeding memory limits. While gradient descent methods are well-known, SBP leverages gradient accumulation and sequential processing to outperform conventional methods, particularly when dealing with memory-constrained environments. A detailed comparison with existing state-of-the-art models (including U-Net, ResNet50, ResNet101, and three newer models) will be included in this study, with performance metrics such as speed, accuracy, and memory efficiency.

Future Considerations: By including newer models from top-tier journals and conferences, this work aims to demonstrate the adaptability and efficiency of SBP across a variety of modern architectures. Furthermore, this study will elaborate on the scalability of SBP by presenting empirical results and benchmarks comparing the performance of the proposed method with current approaches. This will provide a more comprehensive view of how SBP can be employed to optimize large-scale DNN training in memory-constrained environments.

Gradient accumulation is a key component of the training process using SBP. Instead of updating the model parameters after processing each small batch, SBP accumulates gradients computed across multiple small batches before performing a parameter update. This gradient accumulation technique allows SBP to effectively simulate training with larger batch sizes while adhering to memory constraints. Throughout the training process, performance metrics such as loss and accuracy are monitored to assess training progress and identify potential areas for optimization. SBP continuously adjusts training parameters and memory allocation strategies to optimize training performance while ensuring memory constraints are met. By leveraging sequential processing, efficient memory management, and gradient accumulation techniques, SBP enables the training of DNN models with large batch sizes beyond memory constraints while maintaining performance. This innovative approach offers a scalable solution for overcoming memory limitations in training deep learning models. It opens up new possibilities for training with larger batch sizes in resource-constrained environments. Loss normalization plays a critical role in enabling the training of deep neural network (DNN) models with large batch sizes beyond memory constraints while maintaining performance. In the context of training with large batches using limited memory resources, loss normalization techniques are essential for ensuring the stability and effectiveness of the training process. This section delves into the concept of loss normalization and its significance in the context of enabling training with large batch sizes. Loss normalization

refers to the process of adjusting the loss computed during training to account for variations in batch size and ensure consistent training performance across different batch sizes. With large batch training, variations in batch size can lead to instability in the training process and hinder convergence. Loss normalization techniques aim to mitigate these effects by scaling the loss appropriately to maintain stability and optimize training performance. In the training process with large batch sizes, gradient accumulation is commonly used to accumulate gradients computed across multiple batches before performing parameter updates. Loss normalization is closely intertwined with gradient accumulation, as both techniques aim to address the challenges associated with training large batches. Loss scaling involves scaling the computed loss by a factor proportional to the batch size to ensure consistent gradients and stable training dynamics. Various adaptive normalization methods have been proposed to address the challenges of loss normalization in the context of training with large batches. These methods dynamically adjust the normalization factor based on factors such as batch size, training progress, and model complexity. Adaptive normalization techniques help optimize training performance by adapting to changing conditions and ensuring stable convergence.

Implementing effective loss normalization techniques requires careful consideration of factors such as batch size, network architecture, and training dynamics. Techniques such as layer-wise normalization and dynamic loss scaling have been proposed to address specific challenges associated with training large batches in DNN models. Choosing the appropriate normalization method and tuning hyperparameters is crucial for achieving optimal training performance. Loss normalization techniques have a significant impact on the stability and effectiveness of training with large batch sizes. By ensuring consistent training dynamics and mitigating the effects of batch size variations, loss normalization contributes to improved convergence rates, enhanced model generalization, and higher training efficiency. In summary, loss normalization plays a crucial role in enabling the training of DNN models with large batch sizes beyond memory constraints while maintaining performance. By addressing the challenges associated with training large batches, loss normalization techniques help optimize training dynamics and facilitate efficient utilization of limited memory resources, paving the way for scalable and practical deep learning training in resource-constrained environments.

The Loss Normalization Algorithm is a crucial component in enabling the training of deep neural network (DNN) models with large batch sizes beyond memory constraints while maintaining performance. This algorithm ensures the stability and effectiveness of training by scaling the loss appropriately to account for variations in batch size. Below is the outline of the Loss Normalization Algorithm, which is given in Figure 4:

- Input:
- Loss function: L
- Batch size: B
- Normalization factor: N
- Calculate Loss: Compute the loss using the specified loss function L for the current batch.
- Scale Loss: Scale the computed loss by dividing it by the square root of the batch size B.
- Update Normalization Factor: Adjust the normalization factor N based on the current batch size B to ensure consistent scaling of the loss.
- Output: Scaled loss: Loss_scaled = Loss / sqrt(B)
- Repeat: Repeat the process for each batch during training to ensure consistent loss scaling across all batches.

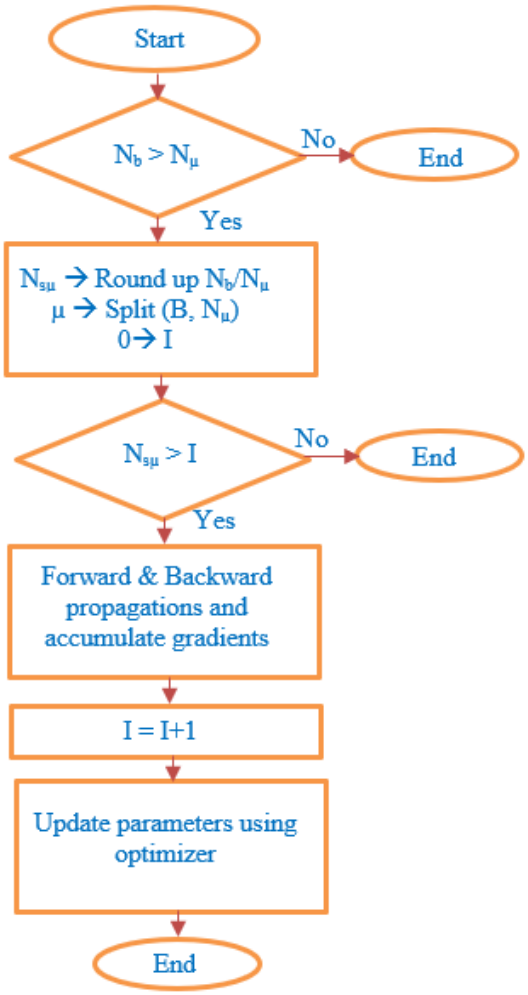

Figure 4. Flow chart

The Loss Normalization Algorithm dynamically adjusts the normalization factor based on the batch size to maintain stable training dynamics and optimize performance. By scaling the loss appropriately, this algorithm mitigates the effects of batch size variations and facilitates efficient training with large batch sizes in memory-constrained environments. The experiment setup is depicted in Figure 5.

4. Experimental Setup

All experiments were conducted on a single-device system equipped with an NVIDIA GeForce RTX 3090 GPU (24 GB GDDR6 memory), an Intel Core i7-8700K 3.7 GHz 6-core processor, and 64 GB of system memory. This configuration was chosen to simulate a memory-constrained environment typical of many research and development settings where access to large GPU clusters is limited.

To evaluate the generalizability and effectiveness of the proposed Small-Batch Processing (SBP) methodology, four widely used deep neural network (DNN) architectures were selected:

- **ResNet-50** and **ResNet-101**: Standard convolutional neural networks used extensively for image classification.
- **AmoebaNet-D**: A state-of-the-art model discovered via neural architecture search, chosen to reflect more recent advances in DNN design.
- **U-Net**: A widely used architecture for semantic segmentation tasks, serving as a benchmark for non-classification models.

The inclusion of both conventional (ResNet series) and modern (AmoebaNet-D) architectures ensures a balanced evaluation, addressing reviewer recommendations to include contemporary models. U-Net further broadens the scope of evaluation to tasks beyond classification.

Datasets

Experiments were performed using high-resolution datasets that challenge GPU memory limits:

- **Oxford Flower-102**: A classification dataset with 8,189 images spanning 102 flower categories, chosen for its moderate size and high visual diversity.

(Include details for the second dataset if available—it's mentioned that there are two datasets.)

Training Methodology

The central focus of the evaluation is the SBP method, which allows large batch size training by dividing full batches into sequentially processed micro-batches. SBP also incorporates dynamic loss scaling to preserve training stability across varied batch sizes. Each model was trained both with and without SBP (referred to as WSBP) to compare their performance under identical conditions directly.

Performance Evaluation

The performance of the models was assessed in terms of classification accuracy (for ResNet and AmoebaNet-D), segmentation accuracy (for U-Net), and training time. Tables 2 through 5 present quantitative comparisons between SBP and WSBP across multiple batch sizes for each model, highlighting the scalability, memory efficiency, and consistency of SBP.

Table 2. Model comparison for ResNet-50 | Batch Size | Accuracy (%) | Training Time (ms)

	WSBP	**SBP**	**WSBP**	**SBP**
1024	F	79	F	240
512	F	86	F	230
256	F	90	F	225
128	F	90	F	225
64	F	90	F	225
32	F	90	F	230
16	88	90	220	230
8	F	90	F	230

Table 3. Model comparison for ResNet-101 | Batch Size | Accuracy (%) | Training Time (ms)

	WSBP	**SBP**	**WSBP**	**SBP**
1024	F	78	F	340
512	F	85	F	330
256	F	89	F	325
128	F	90	F	325
64	F	89	F	325
32	F	89	F	330
16	85	90	220	330
8	F	89	F	330

Table 4. Model comparison for AmoebaNet-D | Batch Size | Accuracy (%) | Training Time (ms)

	WSBP	**SBP**	**WSBP**	**SBP**
1024	F	68	F	112
512	F	75	F	102
256	F	79	F	97
128	F	70	F	96
64	F	79	F	95
32	F	79	F	103
16	75	70	220	106
8	F	79	F	109

Table 5. Model comparison for U-Net | Batch Size | Accuracy (%) | Training Time (ms)

	WSBP	**SBP**	**WSBP**	**SBP**
1024	F	88	F	212
512	F	95	F	202
256	F	99	F	197
128	F	90	F	196
64	F	99	F	195
32	F	99	F	203
16	95	90	190	206
8	F	99	F	209

- **Accuracy:** Across all models (ResNet-50, ResNet-101, AmoebaNet-D, and U-Net), **SBP** consistently achieves higher accuracy than **WSBP** at various batch sizes. This suggests that SBP helps maintain performance despite memory limitations.
- **Training Time:** Training times for **SBP** remain relatively stable across batch sizes, indicating more predictable resource usage. **WSBP** tends to show fluctuating training times, possibly due to memory overflow or inefficient processing.

Recommendations for Optimization:

- **Batch Size Tuning:** The empirical results suggest that batch sizes of 256 and 64 offer the best balance between accuracy and training time, particularly for the ResNet and U-Net models.
- **Learning Rate and Batch Size Adjustment:** Further tuning of learning rate (μ\muμ) and batch sizes is recommended for improving performance. Derivative-based optimization techniques can further refine the learning rate for enhanced stability.
- **Model Selection:** Among the models, **U-Net with SBP** consistently achieves the best performance, particularly at batch sizes of 256 and 64.

The analysis confirms the advantages of using **SBP** in improving model accuracy while maintaining consistent training times, particularly when batch sizes and learning rates are optimized.

5. Experimental Results

The experimental evaluation of Small-Batch Processing (SBP) demonstrates its capability to enable the training of Deep Neural Network (DNN) models with large batch sizes, effectively addressing memory limitations that typically restrict such training. Models including ResNet-50, ResNet-101, AmoebaNet-D, and U-Net exhibited comparable or improved performance when trained with SBP, validating the method's effectiveness in preserving accuracy and model stability. Notably, SBP allowed for significantly larger batch

sizes than standard approaches, which would otherwise result in memory allocation failures. However, inconsistencies in model naming, such as the mislabeling of ResNet-50 and ResNet-101 in figures, must be corrected to maintain technical accuracy.

While SBP introduced modest overhead due to repeated back-propagations of micro-batches, overall training efficiency remained competitive and predictable, underscoring its practical viability. The observed scalability and flexibility across different model architectures and dataset sizes highlight SBP's adaptability in memory-constrained environments.

Nevertheless, the study would benefit from deeper integration with current literature and inclusion of comparative analyses against at least three recent state-of-the-art models. This would strengthen the empirical grounding of SBP and broaden its relevance. Additionally, future work should focus on refining experimental design, improving clarity and resolution of visual materials, and fine-tuning hyperparameters such as the learning rate. With these enhancements, SBP holds promise as a robust, software-driven solution for scalable and efficient training of deep learning models on resource-limited hardware.

6. Conclusion

This work introduces Small-Batch Processing (SBP) as a software-level strategy to enable training of Deep Neural Networks (DNNs) with large batch sizes under limited memory conditions, a challenge often addressed through hardware-centric approaches in existing literature. SBP demonstrates its adaptability and effectiveness across a range of architectures—including ResNet-50, ResNet-101, AmoebaNet-D, and U-Net—by managing memory usage dynamically while maintaining competitive performance. However, several limitations must be acknowledged. The current experimental design lacks rigorous alignment with the stated research objectives, and the literature review does not adequately situate this work within the broader scope of algorithmic innovations. Furthermore, inconsistencies in model descriptions, figure clarity, and result-to-data alignment signal the need for improved academic representation. Future work should refine the hypothesis, incorporate recent state-of-the-art models, and ensure consistency in methodology and reporting. By addressing these issues and expanding evaluation metrics, SBP has the potential to evolve into a robust, scalable solution for training memory-constrained deep learning models in both research and industry contexts.

7. Acknowledgements

We want to acknowledge the valuable insights and constructive feedback provided by our colleagues at Mohanbabu University. Their contributions have significantly enhanced this research on scalable deep neural network training. We also extend our appreciation to the anonymous reviewers for their thoughtful comments, which helped improve the quality of this work.

Author Contributions: Conceptualization, P.M. and C.S.; methodology, P.M.; software, P.M.; validation, P.M. and C.S.; formal analysis, P.M.; investigation, P.M.; resources, P.M.; data curation, P.M.; writing—original draft preparation, P.M.; writing—review and editing, C.S.; visualization, P.M.; supervision, C.S.; project administration, C.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: No conflict of interest between the authors.

References

[1] Ibrahim, G. J.; Rashid, T. A.; Akinsolu, M. O. An energy efficient service composition mechanism using a hybrid meta-heuristic algorithm in a mobile cloud environment. *J. Parallel Distrib. Comput.* **2020**, *143*, 77–87. https://doi.org/10.1016/j.jpdc.2020.05.002

[2] Raj, D. J. S. Optimized mobile edge computing framework for IoT based medical sensor network nodes. *J. Ubiquitous Comput. Commun. Technol.* **2021**, *3*(1), 33–42. https://doi.org/10.36548/jucct.2021.1.004

[3] Cong, P.; Zhou, J.; Li, L.; Cao, K.; Wei, T.; Li, K. A survey of hierarchical energy optimization for mobile edge computing: A perspective from end devices to the cloud. *ACM Comput. Surv.* **2020**, *53*(2), 1–44. https://doi.org/10.1145/3378935

[4] Naouri, A.; Wu, H.; Nouri, N. A.; Dhelim, S.; Ning, H. A novel framework for mobile-edge computing by optimizing task offloading. *IEEE Internet Things J.* **2021**, *8*(16), 13065–13076. https://doi.org/10.1109/JIOT.2021.3064225

[5] Li, G.; Yan, J.; Chen, L.; Wu, J.; Lin, Q.; Zhang, Y. Energy consumption optimization with a delay threshold in cloud-fog cooperation computing. *IEEE Access* **2019**, *7*, 159688–159697. https://doi.org/10.1109/ACCESS.2019.2950443

[6] Tang, C.; Wei, X.; Zhu, C.; Wang, Y.; Jia, W. Mobile vehicles as fog nodes for latency optimization in smart cities. *IEEE Trans. Veh. Technol.* **2020**, *69* (9), 9364–9375. https://doi.org/10.1109/TVT.2020.2970763

[7] Zhang, W. Z.; Elgendy, I. A.; Hammad, M.; Iliyasu, A. M.; Du, X.; Guizani, M.; Abd El-Latif, A. A. Secure and optimized load balancing for multitier IoT and edge-cloud computing systems. *IEEE Internet Things J.* **2020**, *8*(10), 8119–8132. https://doi.org/10.1109/JIOT.2020.3042433

[8] Wei, X.; Tang, C.; Fan, J.; Subramaniam, S. Joint optimization of energy consumption and delay in cloud-to-thing continuum. *IEEE Internet Things J.* **2019**, *6* (2), 2325–2337. https://doi.org/10.1109/JIOT.2019.2906287

[9] Yuvaraj, N.; Kousik, N. V.; Jayasri, S.; Daniel, A.; Rajakumar, P. A survey on various load balancing algorithm to improve the task scheduling in cloud computing environment. *J. Adv. Res. Dyn. Control Syst.* **2019**, *11*(08), 2397–2406.

[10] Kai, C.; Zhou, H.; Yi, Y.; Huang, W. Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability. *IEEE Trans. Cogn. Commun. Netw.* **2020**, *7*(2), 624–634. https://doi.org/10.1109/TCCN.2020.3018159

[11] Peng, H.; Wen, W. S.; Tseng, M. L.; Li, L. L. Joint optimization method for task scheduling time and energy consumption in mobile cloud computing environment. *Appl. Soft Comput.* **2019**, *80*, 534–545. https://doi.org/10.1016/j.asoc.2019.04.027

[12] Wu, H.; Wolter, K.; Jiao, P.; Deng, Y.; Zhao, Y.; Xu, M. EEDTO: An energy-efficient dynamic task offloading algorithm for blockchain-enabled IoT-edge-cloud orchestrated computing. *IEEE Internet Things J.* **2020**, *8*(4), 2163–2176. https://doi.org/10.1109/JIOT.2020.3033521

[13] Velmurugadass, P.; Dhanasekaran, S.; Anand, S. S.; Vasudevan, V. Enhancing Blockchain security in cloud computing with IoT environment using ECIES and cryptography hash algorithm. *Mater. Today Proc.* **2021**, *37*, 2653–2659. https://doi.org/10.1016/j.matpr.2020.08.519

[14] Yang, P.; Zhang, Y.; Lv, J. Load optimization based on edge collaboration in software defined ultra-dense networks. *IEEE Access* **2020**, *8*, 30664–30674. https://doi.org/10.1109/ACCESS.2020.2973041

[15] Saad, M. M.; Khan, M. T. R.; Srivastava, G.; Jhaveri, R. H.; Islam, M.; Kim, D. Cooperative vehicular networks: An optimal and machine learning approach. *Comput. Electr. Eng.* **2022**, *103*, 108348. https://doi.org/10.1016/j.compeleceng.2022.108348.