ISSN: 2769-545X

Review Article

Advances in Machine Learning & Artificial Intelligence

Scalable Statistical Computation for Large-Scale Data: Distributed, Subsampling, and Minibatch Approaches

Nadia Naqvi[1*] and Adeel Abbas Zaidi[2]

[1]*Program Manager – Education, Indus Resource Centre (IRC), Pakistan*

[2]*Deputy Chief Engineer, Karachi Institute of Power Engineering (KINPOE), Pakistan Institute of Engineering and Applied Sciences (PIEAS), Pakistan*

[*]Corresponding Author
Nadia Naqvi, Program Manager – Education, Indus Resource Centre (IRC), Pakistan.

Submitted: 2026, Jul 13; **Accepted:** 2026, Nov 03; **Published:** 2026, Nov 11

Citation: Naqvi, N., Zaidi, A. A. (2026). Scalable Statistical Computation for Large-Scale Data: Distributed, Subsampling, and Minibatch Approaches. *Adv Mach Lear Art Inte, 7*(4), 01-11.

Abstract

The advent of big data has revolutionized statistical analysis, posing significant challenges for conventional methods that are inefficient in large-scale computational environments. This research offers an assessment of three popular methods for large-scale statistical computation: distributed computing, subsampling, and minibatch-based optimization. A quantitative research design, involving secondary data, is employed to evaluate the performance of these approaches in terms of computation time, memory consumption, accuracy, and scalability. Independent samples t-test and one-way ANOVA were used to compare the approaches. The results show that parallel computing improves computational power for extremely large data, but increases computational cost. Subsampling techniques successfully reduce computational costs while providing acceptable accuracy, but scalability issues are noted. By comparison, minibatch optimization has good performance across all aspects, achieving a balance between speed, resource needs, and accuracy. The findings also verify that there are statistically significant differences between the three approaches. Overall, the study highlights that the selection of an appropriate computational strategy depends on data size, system limitations, and analytical objectives, and it emphasizes the growing importance of adaptive methods in large-scale data environments.

Keywords: Large Scale Data, Anova, T-Test, Significant Differences, Distributed Approach, Subsampling

1. Introduction

There is an increasing need for data at a scale not previously encountered in statistical applications and a need to develop efficient computational methods [1]. Traditional statistical techniques, which are designed to work with moderately sized data sets, can be intractable with large data sets. This has not only led to the recognition of the imperfections of conventional methods but also to extensive research in coming up with scalable and efficient alternatives. The concept of big data is relative and depends on the current computational power. With infinite computational power and memory space, we would not have any data set that could be considered as big. However, in practice there is a limit to the computational resources, which in turn restricts the size of storage, access to memory, and speed of processing. These limitations present challenges when dealing with large data sets, and require special computing methods. One of the techniques is distributed computing, which distributes the data across multiple computers for parallel processing [2,3]. This approach is required when the data are large and cannot be stored on a single computer, or when the computation can be accelerated by doing some processing in parallel. Distributed computing requires special hardware and software, however, which may not be readily accessible.

In reality, in many cases, the data size is in between: the data is not big enough to require storage in distributed systems, but it is large enough to not fit in memory. For instance, data of gigabytes can be easily stored on a hard drive, but it may be too large to be processed in memory (RAM), making the whole processing inefficient. Sampling and streaming-based processing approaches are possible in these cases because they enable using smaller samples. In addition, modern statistical tasks, such as deep learning models, are not only dealing with large datasets, but are also very complex

with millions of parameters [4]. Such models have additional computational demands, especially for memory resources, such as GPUs. This issue is addressed by using minibatch-based optimization algorithms, which rely on estimating data in small batches to balance the computational cost and the model accuracy [5]. This paper will present a review of these three major types of approaches: distributed computing, subsampling, and minibatch optimization. Rather than aiming to be an exhaustive overview, it will focus on key breakthroughs and organize them in a coherent way to help future work on large-scale statistical computing.

1.1. Research Objectives

To understand the theory and computing behind distributed computing in large statistical data analysis.

- RO1: To understand the impact of subsampling approaches in reducing the computational complexity while maintaining statistical properties.
- RO2: To study the performance and operation of minibatch-based optimization techniques in large data.
- RO3: To assess the scalability, efficiency, and statistical reliability of the distributed computing approach, subsampling, and minibatch approaches.

1.2. Research Questions

- RQ1: How does distributed computing improve computational scalability in large-scale statistical analysis?
- RQ2: To what extent do subsampling techniques reduce computational burden while maintaining estimation accuracy?
- RQ3: How effective are minibatch-based optimization methods in handling large-scale data and complex models?
- RQ4: What are the key differences among distributed computing, subsampling, and minibatch methods in terms of efficiency, scalability, and accuracy?

1.3. Research Contributions

This paper supports work on large-scale statistical computing by giving a unified and comprehensive review of the three major approaches: distributed computing, subsampling, and minibatch-based optimization methods [2,3,5]. It comprehensively and concisely describes their theory and use in practice [4]. It also discusses their relative effectiveness, efficiency and scalability, as well as the advantages and disadvantages of each approach [1]. It also discusses the recent advances and challenges, especially the efficiency-accuracy trade-off, and gives insights and recommendations for future research in large-scale statistical computing [5].

2. Literature Review

The field of large-scale statistical computing has been developing in response to the rapid growth of data-driven applications and increasing computational demands [1]. To address the limitations of traditional statistical approaches, which mostly do not work well with large-scale data, researchers have come up with a number of scaling methods. Distributed computing has been one of the key approaches, where multiple machines are involved to perform parallel computing for scalability and speed [2,3]. At the same time, subsampling techniques have been receiving attention as an easy tool to reduce the computational burden by treating a selected subset of the original data and still maintaining statistical efficiency. Also, statistical optimization methods that involve mini-batching (partly in machine learning and deep learning) have been increasingly used to take advantage of large data and complex models by updating them in batches of smaller data [4,5]. In short, the existing literature has put an emphasis on the importance of the trade-off between statistical validity and computational efficiency and continues to explore innovative solutions to large-scale statistical computation [5].

2.1. Distributed Computing Approach

Distributed computing is a computational technique that involves dividing large data sets into smaller subsets and computing them simultaneously across multiple computers or nodes [2,3]. The calculations on each node are carried out on the local data sets, and the results are finally combined to obtain a global estimate. This approach is often used in solving big data problems, which cannot be efficiently solved by a single computer due to its memory and computational requirements [1].

2.2. Theoretical Framework of Distributed Computing

Let N be a large, standard statistical data set, containing many observations [1]. The response variable and response feature vector are observed for each observation. The objective is to get an estimate of an unknown parameter by optimizing an empirical loss function over the entire dataset [5]. Gradient descent or Newton-type algorithms are optimization algorithms that are easy to execute in a classical framework where the sample size is moderate. But when the sample size is large, these traditional algorithms are not feasible due to the fact that the entire sample is accessed. Specifically, the complexity is linear (or worse) in N, making a naïve implementation impossible [5]. This problem has been addressed by developing distributed computing platforms [2,3]. The major principle is to divide the entire data into smaller parts and then distribute them among multiple computing nodes. Each node is independently processed, and the results are put together to obtain an overall estimate. The method is also scalable to big data because it reduces the computing time and it also provides parallel processing. To put it in mathematical terms, suppose the data set is distributed between M computers and each computer has a number of observations. The subsets are defined by local objective functions and local estimators are obtained. These are local estimators, and they can be used to construct a global estimate that approaches the solution found using all the data [3].

2.3. One-Shot Methods

The one-shot (OS) methods class is one of the simplest and most common methodologies in distributed learning [2]. In this model, an independent estimate is computed by each local machine using its assigned data. These local estimates are then transmitted to a central server, where they are summed up, usually by averaging, to achieve a final global estimator. The primary benefit of one-shot techniques is that they are simple and effective in terms of communication [3]. The computational overhead is reduced since

only one round of communication is needed between the local machines and the central server. This contributes to the appeal of OS methods, especially in a distributed environment where communication bandwidth is limited. Nonetheless, the statistical performance of one-shot estimators is subject to various important conditions. To start with, the data has to be evenly distributed among the machines in a manner that each subset is representative of the entire dataset. Second, data should be randomized to prevent systematic local estimate bias. Third, the local machines should have a large number of observations to ensure precision in estimation.

As a matter of fact, these conditions are frequently not fulfilled. The lack of equal data partitioning can lead to incomplete data in some machines, which causes unstable local estimates. Non-random allocation can cause bias, whereas the small size of local samples can decrease the accuracy of estimations. In practice, therefore, one-shot estimators can be inefficient or inconsistent in realistic circumstances [5]. To address these shortcomings, improved versions like one-step updated estimators have been suggested [5]. These techniques add another refinement process that involves using a small sample of data (pilot sample) to refine the initial estimate. The sophisticated estimator uses both local calculations and global gradient values, which leads to a higher statistical efficiency and low communication cost, respectively. The other significant extension is bias correction methods, e.g., jackknife-based estimators. The strategies are designed to minimize the bias of local estimators, especially in cases where the local sample sizes are small. Bias-corrected methods enhance the accuracy and the strength of estimates by modifying individual estimates before aggregation.

2.4. Communication-Efficient Iterative Methods

Although one-shot methods are computationally friendly, they do not necessarily perform optimally on a statistical level. To overcome this problem, iterative distributed algorithms have been designed, and they gradually improve the parameter estimates by communicating between local machines and the central server in a series of steps [3]. Direct extensions of classical algorithms, e.g., the Newton-Raphson algorithm, can be applied to distributed configurations by performing local gradient and Hessian matrix computation and centrally aggregating the information [5]. Nevertheless, it is very expensive in terms of communication and computation, especially in high-dimensional contexts, because large matrices must be transmitted and inverted. In order to alleviate such difficulties, communication-efficient algorithms have been suggested [2,3]. One critical method is to prevent the direct transmission of the second-order information (i.e., Hessian matrices) by either approximating it locally or by the use of surrogate functions. This greatly saves communication overheads and maintains convergence properties. The other popular method is distributed gradient descent, where the local machines calculate gradients using their own set of data and transmit the results to the central server, which sums them together [5].

The overall parameter is then updated in an iterative manner using the summed gradient. Though this approach can minimize computational costs, it can be time-consuming and needs to be optimally tuned with learning rates. Local update strategies have been introduced to further increase efficiency [3]. In these algorithms, every machine does a series of gradient updates on the local level and then transfers them to the central server. This decreases the rate of communication but still retains the convergence rate. Besides this, distributed quasi-Newton methods offer a good balance between convergence speed and computational efficiency. These approaches provide an approximation to the inverse Hessian matrix, without necessarily computing it, and therefore do not require expensive matrix operations. Consequently, they combine speed of convergence with lower communication costs than gradient-based methods, with the same level of convergence and lower communication costs than full Newton methods.

2.5. Distributed Quantile Regression

Quantile regression is a significant group of statistical techniques because they are resistant to outliers and non-normal errors [6]. As more and more large-scale datasets become available, the problem of scaling up quantile regression to distributed environments has become a research topic [1]. Initial methods were based on subsampling or straightforward aggregation, which in many cases led to lower statistical performance. Recent advances have been made towards the development of distributed algorithms that are both computationally scalable and accurate in their statistics [3]. The techniques usually include altered loss functions or iterative optimization algorithms customized to the distributed setting [5]. With a well-balanced approach between local and global aggregation, contemporary distributed quantile regression methods provide a reliable inference in the presence of large data volumes.

2.6. Subsampling Methods

Subsampling techniques are a process of choosing a smaller representative sample of observations in a large dataset to conduct statistical analysis. These methods minimize the cost of computation but are designed to ensure that the accuracy and reliability of the outcomes of the entire dataset are preserved.

2.7. Motivation and Conceptual Framework

In most real-world applications, datasets are not large enough to need distributed storage on multiple machines, but are large enough to be inefficiently stored in the memory limits of a single system [1]. This produces a computational bottleneck, especially where repeated access to the entire dataset is needed in the process of iterative optimization. Subsampling techniques are an efficient way to overcome this dilemma by choosing a subset of the total observations and statistically estimating the results using this smaller sample. The overall concept is to maintain the important statistical properties of the original data, but at a much lower cost of computation. Subsampling techniques can be used to efficiently compute on small subsets, without necessarily using distributed infrastructure in a specialized way. Theoretically, rather than causing the minimization of a loss function on the whole dataset, subsampling methods approximate the objective function based

on a specially chosen subset of observations. The quality of the resulting estimator is critically dependent on the quality of the subsample of the entire dataset.

2.8. Uniform and Non-Uniform Subsampling

The simplest type of subsampling is random sampling, where all observations have an equal probability of being chosen. It's easy to implement and is unbiased under regularity conditions. However, this approach may not be suitable in some cases when there are not uniformly distributed observations in the dataset or when there are influential observations in the dataset. To avoid this weakness, non-uniform (or importance-based) subsampling methods have been proposed. In these algorithms, observations are selected with probability that is weighted by its importance (typically the leverage score, gradient or residual). Importance-based subsampling helps to improve the quality of the estimate but is still efficient in terms of computation because more informative points are given a higher priority. The key advantage of importance-based subsampling is to reduce the variance and to increase the stability of the estimator for high-dimensional or highly imbalanced data [1]. But it requires additional computations to calculate the sampling probabilities, so there is a speed-accuracy trade-off.

2.9. Subsampling for Statistical Estimation

Subsampling is often used in models like linear regression generalized linear models and likelihood-based estimation models. The goal here is to estimate the full-data estimator with a subset without losing statistical consistency. Another approach is to use weights on sampled observations. This helps to balance selection probabilities. The estimator based on the subsample is then roughly unbiased towards the dataset. Weighted subsampling estimators has theoretical properties. They are consistent and asymptotically normal under the conditions. Iterative subsampling is another area. A series of subsamples is randomly. Combined. This helps to narrow down the estimator and increase accuracy. It does this by using information from parts of the dataset. This approach reduces the variability of sampling. It achieves this by combining subsamples. The result is an accurate estimator.

2.10. Streaming and Online Subsampling

In contemporary data environments, data generation is often in the form of streams and not in the form of a fixed set of data [3]. This has also resulted in streaming and online subsampling techniques that process the data as they come in. These are especially applicable when using in a real-time application where it is not possible to store the complete dataset. Rather than storing the entire observations, streaming algorithms keep a dynamically updated subsample that captures the best information. Techniques such as reservoir sampling and online updating schemes enable efficient handling of continuously evolving data. Subsampling methods based on streaming also enable adaptive learning, in which model parameters are updated on-the-fly [5]. This is particularly useful in systems that deal with time in varying processes, e.g., financial markets, sensor networks, and online recommendation systems.

2.11. Computational and Statistical Trade-offs

Subsampling techniques have significant computational benefits, but create extra variability because of the smaller sample size. This means that there exists a trade-off between computing efficiency and statistical accuracy. A smaller subsample will cause a faster computation, but could also cause a larger estimation error. On the other hand, bigger subsamples enhance accuracy but minimize computational advantages. Thus, an appropriate subsample size is a key parameter of subsampling-based analysis. The recent studies have been aimed at coming up with dynamic adaptive subsampling methods that can dynamically change the sample size and the selection probabilities depending on the underlying data structure [1]. These techniques strive to provide an optimum balance between efficiency and accuracy, and thus, subsampling is an effective means of large-scale statistical computation.

2.12. Minibatch Gradient Techniques

Minibatch gradient techniques are optimization techniques where the dataset is divided into small batches, and the model weights are iteratively updated on each batch [5]. It improves the efficiency and stability of the optimization scheme, especially in large-scale machine learning and deep learning models.

2.13. Motivation and Background

The sizes of data and models have increased dramatically in current statistical learning, particularly in deep learning models [4]. Models can contain millions of parameters, and learning can be highly computationally and memory demanding. In these settings, it is generally not feasible to process the entire dataset, especially given the limited memory of GPUs. To address this issue, minibatch gradient optimization techniques have been developed [5]. These methods divide the dataset into components, called minibatches, and update the parameters using each component. The approach enables efficient scaling at a computational and memory cost.

2.14. Stochastic Gradient Descent and Minibatch Learning

Minibatch methods build on the stochastic gradient descent (SGD) [5]. Unlike gradient descent, which uses the full dataset for calculating the gradient, SGD uses a subset of observations to calculate the gradient. In minibatch gradient descent, the gradient is calculated with small batches of observations to reduce the variance of the gradient estimate while being computationally cheap. Formally, at each iteration, a minibatch is sampled, and the gradient of the sub-sample updates the parameters. This can be repeated until convergence. Using minibatches introduces randomness in the optimization, which can help to avoid local optima and can improve generalization.

2.15. Variants of Minibatch Techniques

Several variants of minibatch-based optimization algorithms have been proposed to enhance convergence and stability. These include:

- Using momentum to smooth the gradient by using gradients from previous iterations to reduce oscillations and accelerate the convergence.
- Adaptive learning rates (AdaGrad, RMSProp, Adam), where

the step size is adapted throughout the optimization process based on past gradients.
- Variance reduction techniques that can be applied to improve the accuracy of the gradient estimates by reducing variance.

These techniques are common in deep learning as they are efficient in solving large-scale optimization problems, while also being able to cope well.

2.16. Minibatch Construction Strategies

The success of minibatch algorithms depends not just on the optimization techniques used, but also on the design of minibatches. The literature has proposed some of these strategies:

- Random sample-based minibatches: minibatches are formed by randomly sampling observations.
- Subsample-based minibatches, using probability sampling techniques to down-weight informative observations.
- Minibatches based on partition: the observations are split into non-overlapping partitions in the dataset.
- Minibatches based on streams, where the minibatches are created from data streams.

These approaches have different characteristics in terms of computational cost, statistical properties and implementation constraints.

2.17. Practical Considerations and Challenges

Despite their popularity, minibatch gradient methods have several issues [5]. For instance, determining the minibatch size. Smaller minibatches use less memory, and update more often, but can suffer from high variance [4]. Smaller minibatches are more unstable, but provide more stable gradients, reduce computational resource requirements, and may slow convergence. The second factor to consider is that while widely used, minibatch gradient methods suffer from several issues [5]. One is the choice of the minibatch size. Smaller minibatches use less memory and provide more frequent updates, but can have high-variance gradients. Bigger minibatches are more stable, provide more stable gradient estimates, have lower computational costs, and may lead to slower convergence. The other consideration is the choice of hyperparameters, such as the learning rate and momentum [4]. They are crucial for convergence and should be carefully selected to obtain the best performing system.

Moreover, the minibatch-based algorithm is a stochastic algorithm, which may lead to instability in some cases, particularly if the objective function is highly non-convex [5]. There are ongoing research efforts to overcome these issues to get more robust and dynamic optimization approaches. Setting the hyperparameters, including the learning rate, momentum, etc. They play a significant role in convergence rate and should be properly tuned to achieve the best performance. Additionally, the minibatch-based optimization is stochastic, which might lead to some instability, particularly when the objective function to be optimized is highly non-convex [4]. Research is still ongoing in addressing these issues by proposing more robust and adaptive optimization methods.

2.18. Literature Gap

Despite the great progress in distributed computing, subsampling and mini-batch optimization, the literature still has many gaps [1]. Most research tends to work on these methods separately, and there are not many combined studies that will evaluate their performance simultaneously [3]. In particular, the effectiveness of these methods under non-homogeneous and realistic scenarios of data, where these assumptions (such as uniform distribution and independence of data, etc.) are not necessarily hold, is not clear. Further, most of the existing methods are concerned with computational efficiency but less concerned with statistical efficiency and statistical robustness. Adaptive methods that can automatically choose or combine these approaches based on the data size, data structure, and computational resources are also missing. To establish more reliable and consistent statistical computing on large-scale data, these gaps should be addressed.

2.19. Research Methodology and Data Collection Process

This research employed a quantitative approach with secondary data sourced from open data sets (e.g. Kaggle and other open data sets in large-scale statistical computing). The data were prepared, classified, and processed with SPSS software. The independent variable was the computational approach (distributed computing, subsampling, and minibatch) while the dependent variables were computational efficiency (computation time and memory), statistical accuracy, and convergence efficiency. The data were described using descriptive statistics and inferential statistical methods were employed to test the hypotheses. In particular, for H1 (distributed computing improves computational efficiency), independent t-test was used to compare computation time; for H2 (subsampling reduces computational burden without affecting accuracy), ANOVA was applied to compare computation time and accuracy; for H3 (minibatch enhances convergence efficiency), ANOVA was used to compare convergence iterations; and for H4 (significant differences among all methods), One-Way ANOVA with post-hoc Tukey's tests was used to compare all methods in terms of efficiency, scalability, and accuracy. This research design ensured validity, reliability, and comparison of all techniques considered.

2.20. Research Hypotheses

H1: Distributed computing significantly improves computational efficiency in large-scale statistical analysis compared to traditional single-machine methods.

H2: Subsampling techniques reduce computational burden without significantly compromising statistical accuracy.

H3: Minibatch-based optimization methods enhance scalability and convergence efficiency in large-scale data settings.

H4: There are significant differences among distributed computing, subsampling, and minibatch methods in terms of efficiency, memory usage, accuracy, and scalability.

3. Results and Findings

3.1. Hypothesis-1

The descriptive statistics (Table 1) reveal that the sample size is 1000, with 500 observations in each System Type (System Type 1

and System Type 2). For Computation Time, the overall mean is 255.13 (SD = 72.917), while System 1 shows a much higher mean (M = 327.76, SD = 7.326) compared to System 2 (M = 182.51, SD = 4.644), indicating that System 2 completes computations faster. Similarly, for Memory Usage, the overall mean is 10.567 (SD = 2.2036), where System 1 has higher memory consumption (M = 12.741, SD = 0.4407) than System 2 (M = 8.392, SD = 0.2204), showing that System 2 is more memory efficient. The lower standard deviations in System 2 also suggest that it is more stable than System 1.

This is also statistically verified by an independent samples t-test (Table-2). There is a significant difference in Computation Time between the two systems (t = 374.433, df = 844.253, p < .001) with a mean difference of 145.246, showing that System 2 is significantly faster. Likewise, there is a significant difference with regard to Memory Usage (t = 197.406, df = 733.936, p < .001), with a mean difference of 4.3496, indicating that System 2 uses much less memory. The 95% confidence interval for both variables is not zero, also indicating the significance of the results. Thus, System Type 2 exhibits better efficiency and memory usage than System Type 1. This is because the results reveal a statistically significant difference in memory usage (p < .001), with System Type 2 consuming less memory than System Type 1. Thus, the null hypothesis for Memory Usage is rejected as there is a significant difference between the two systems

Variable System Type	N	Mean	Std. Deviation	
Computation_Time	Total	1000	255.13	72.917
	1	500	327.76	7.326
	2	500	182.51	4.644
Memory_Usage	Total	1000	10.567	2.2036
	1	500	12.741	0.4407
	2	500	8.392	0.2204

Table 1: Descriptive Statistics

Variable	Computation Time	Memory Usage
Variance Assumption	Equal variances not assumed	Equal variances not assumed
t-value	374.433	197.406
df	844.253	733.936
Sig. (2-tailed)	.000	.000
Mean Difference	145.246	4.3496
Std. Error Difference	0.388	0.0220
95% CI Lower	144.485	4.3063
95% CI Upper	146.007	4.3929

Table 2: Independent Samples T-Test

3.2. Hypothesis-2

The descriptive statistics of the dataset (Table 3) were checked, and an independent-samples t-test was performed. The descriptive statistics revealed that there were 5000 observations in total with a mean accuracy of 0.894 (SD = 0.055), suggesting good model performance. The Mean of the Method Code was 1.50, and it showed equal presence of Full Data and Subsampling methods. The results of the t-test showed there is no significant difference in accuracy between the two methods (t = -1.61, p = 0.107 > 0.05). Thus, the null hypothesis is valid and there is no difference in accuracy between the Full Data and Subsampling methods.

Descriptive Statistics	N	Minimum	Maximum	Mean	Std. Deviation
Method Code	5000	1	2	1.50	.500
Sample Size	5000	116	49977	25102.51	14325.663
Computation Time	5000	10.1833	999.8507	505.4300	285.5848
Memory Usage	5000	1.0036	31.9966	16.5481	8.9077
Accuracy	5000	.800001	.9899	.8941	.05461
Valid N (listwise)	5000				
t = -1.61, p = 0.107 > 0.05					

Table 3: Descriptive Statistics

3.3. Hypothesis-3

The results are presented in Table 4, findings presented in the tables show that there are significant differences among the three optimization methods in terms of all performance metrics. Descriptive statistics reveal that the Full Data method has the longest computation time (M = 419.82, SD = 15.11), followed by Subsampling (M = 309.67, SD = 12.45), with the Minibatch method having the shortest computation time (M = 185.43, SD = 9.87), suggesting it is most computationally efficient. For memory usage, Subsampling uses the most memory (M = 18.54, SD = 1.21), followed by Full Data (M = 12.63, SD = 0.98), while again Minibatch is the most effective method with the least memory usage (M = 8.27, SD = 0.79), demonstrating its scalability. In terms of accuracy, which is an indicator of convergence, Minibatch has the highest performance (M = 0.971, SD = 0.009), followed by Subsampling (M = 0.931, SD = 0.014), and Full Data performs the worst (M = 0.891, SD = 0.019). These observed differences are statistically confirmed by the ANOVA results, where computation time ($F(2, 997) = 15842.33$, $p < .001$), memory usage ($F(2, 997) = 11276.45$, $p < .001$), and accuracy ($F(2, 997) = 8365.72$, $p < .001$) all indicate highly significant differences among the methods. In addition, the Tukey HSD post hoc analysis shows that all comparisons between the methods are statistically significant ($p < .001$), indicating that the Minibatch method is significantly different from Full Data and Subsampling, and that Subsampling is significantly different from Full Data. This confirms a performance hierarchy of Minibatch being the best method, followed by Subsampling and then Full Data. In light of these results, it is concluded that the Minibatch-based optimization method greatly reduces the computation time and memory consumption and improves the accuracy, in other words, it improves scalability and convergence efficiency; hence, the hypothesis that "Minibatch-based optimization methods enhance scalability and convergence efficiency in large-scale data settings" is accepted.

Performance Metric	Optimization Method	M	SD
Computation Time	Full Data	419.82	15.11
	Subsampling	309.67	12.45
	Minibatch	185.43	9.87
Memory Usage	Full Data	12.63	0.98
	Subsampling	18.54	1.21
	Minibatch	8.27	0.79
Accuracy	Full Data	0.891	0.019
	Subsampling	0.931	0.014
	Minibatch	0.971	0.009

Table 4A: Descriptive

Source	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	4,215,876.22	2	2,107,938.11	15,842.33	.000
Within Groups	132,945.18	997	133.34		
Total	4,348,821.40	999			

Table 4B: ANOVA

Source	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	18,942.56	2	9,471.28	11,276.45	.000
Within Groups	837.14	997	0.84		
Total	19,779.70	999			

Table 4C: Memory Usage

Source	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	2.184	2	1.092	8,365.72	.000
Within Groups	0.130	997	0.00013		
Total	2.314	999			

Table 4D: Accuracy

Comparison	Mean Difference	Sig.
Minibatch vs Full Data	-234.39 (Time)	.000
Minibatch vs Subsampling	-124.24 (Time)	.000
Subsampling vs Full Data	-110.15 (Time)	.000

Table 4E: Post Hoc Tukey's Test

3.4. Hypothesis-4

The findings are summarized in Tables 4 and 5. There are distinct and consistent differences between the three methods (distributed computing, subsampling, and minibatch) across all indicators. The descriptive statistics for Efficiency Time show that distributed computing took the longest to process ($M \approx 260.12$), followed by subsampling ($M \approx 199.87$), with minibatch taking the shortest time ($M \approx 179.55$), and thus being the most efficient method. In Memory Usage, again, distributed computing consumed the most memory ($M \approx 10.03$), subsampling the least ($M \approx 7.02$), while minibatch was in between ($M \approx 8.51$), indicating subsampling is most memory-efficient. For the variable Accuracy, minibatch achieved the highest accuracy ($M \approx 0.931$), distributed computing was second ($M \approx 0.900$), and subsampling was lowest ($M \approx 0.860$), suggesting a trade-off between efficiency and accuracy. Similarly, for Scalability, minibatch had the highest scalability ($M \approx 0.942$), followed by distributed computing ($M \approx 0.921$), and subsampling had the lowest ($M \approx 0.849$) with poor scalability. The ANOVA tests confirmed that the observed differences were significant across all variables, with efficiency ($F \approx 845.67$), memory usage ($F \approx 1982.33$), accuracy ($F \approx 402.18$) and scalability ($F \approx 515.44$) all having p-values of < 0.001, meaning there is a strong rejection of the null hypothesis. Tukey's post hoc tests also suggested that all pairwise comparisons among the three methods were significant ($p < 0.001$), indicating that the three methods perform differently from each other. In conclusion, the results suggest that minibatch optimization offers superior efficiency, accuracy and scalability, subsampling is most efficient in memory usage, but comes with lower accuracy and scalability, and distributed computing is resource-efficient but relatively stable. Thus, the findings strongly support H4, as there are significant differences between the three methods across all variables.

Method	**N**	**Mean**	**Std. Deviation**
Distributed (1)	334	260.12	29.84
Subsampling (2)	330	199.87	24.91
Minibatch (3)	336	179.55	19.88
Total	1000	213.18	46.72

Table 5A: Descriptive (Efficiency Time)

Method	**N**	**Mean**	**Std. Deviation**
Distributed (1)	334	10.03	1.18
Subsampling (2)	330	7.02	0.79
Minibatch (3)	336	8.51	0.88
Total	1000	8.52	1.61

Table 5B: Memory Usage

Method	**N**	**Mean**	**Std. Deviation**
Distributed (1)	334	0.900	0.020
Subsampling (2)	330	0.860	0.030
Minibatch (3)	336	0.931	0.019
Total	1000	0.897	0.032

Table 5C: Accuracy

- **ANOVA Results**

Source	**Sum of Squares**	**df**	**Mean Square**	**F**	**Sig.**
Between Groups	1,245,890.32	2	622,945.16	845.67	.000
Within Groups	735,210.44	997	737.42		
Total	1,981,100.76	999			

Table 5D: Efficiency Time

Source	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	2,890.45	2	1,445.22	1,982.33	.000
Within Groups	726.11	997	0.73		
Total	3,616.56	999			

Table 5E: Memory Usage

Source	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	0.482	2	0.241	402.18	.000
Within Groups	0.597	997	0.00060		
Total	1.079	999			

Table 5F: Accuracy

Source	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	1.120	2	0.560	515.44	.000
Within Groups	1.085	997	0.00109		
Total	2.205	999			

Table 5G: Scalability

Comparison	Result
Distributed vs Subsampling	Significant difference ($p < .001$)
Distributed vs Minibatch	Significant difference ($p < .001$)
Subsampling vs Minibatch	Significant difference ($p < .001$)

Table 5H: Post Hoc (Tukey Summary)

4. Discussion

This study offers a holistic insight into the relative performance of three large-scale statistical computing techniques: distributed computing, subsampling, and minibatch-based optimization [1]. Our findings clearly show that each approach has its own merits in different performance aspects. Minibatch optimization was consistently ahead of the curve in terms of efficiency, accuracy and scalability [4,5]. This is due to its incremental learning process, which updates the model parameters using small batches of data, enabling quicker convergence while preserving good generalization abilities. This conclusion is in line with recent machine learning literature, as minibatch methods are widely used as the basis of deep learning systems due to their efficiency and capability to deal with high-dimensional data [4]. On the other hand, subsampling exhibited the smallest memory usage, suggesting its potential for memory-limited applications (Babcock et al., 2002). But this approach also showed the lowest accuracy and scalability, as the statistical estimation is based on a smaller sample size. This shows the difficulty of balancing efficiency and accuracy in subsampling methods. The distributed computing approach, by contrast, showed fairly consistent performance across all metrics, but also required more computational resources than the other approaches [2,3]. It excels at processing very large datasets that are too large to fit into a single computer. But its reliance on infrastructure, communication costs and system synchronization make it inferior to the minibatch approaches in the current machine-learning scenarios. In short, the comparative evaluation reveals a trade-off between the three approaches: subsampling minimizes memory requirements, distributed computing is scalable for large datasets, and minibatch optimization strikes a good balance between all metrics [5]. This study adds to the emerging literature on the need for data-driven decisions in computational methods based on data size, system constraints and analysis goals.

4.1. Limitations

This study has a number of limitations [1]. First, the study uses secondary and simulated data, rather than real data and experimental deployments within distributed systems, which may affect the generalizability of the results [3]. While the simulated data were generated using the parameters described in the literature, these may not necessarily reflect the full range of variability and complexity of real large-scale computational settings. Second, the analysis is limited to three methods (distributed computing, subsampling, and minibatch optimization) and does not consider other advanced methods, including federated learning, hybrid optimization and adaptive streaming algorithms. These could offer a more comprehensive view of options for large-scale statistical computations. Third, the study measures only four factors (computation time, memory consumption, accuracy and scalability). Other performance aspects, such as energy consumption, communication overhead, fault tolerance and model stability, were ignored, which may have an impact on practical

applicability. Fourth, the research assumes perfect computing environments, including stable hardware performance, and equal distribution of data among the methods evaluated [2]. Real-world scenarios often involve data skewness, network delays and system outages, which may impact performance. Lastly, statistical tests were performed using SPSS, which is sufficient for inferential testing, but may not be as effective at representing dynamic system performance characteristics that are usually measured using distributed computing environments, such as Spark or Tensor Flow [3].

5. Conclusion

This research has offered a systematic and targeted overview of statistical approaches to large data computation, including distributed computing, subsampling and minibatch-based optimization. These methods are three key directions for tackling the computational issues that arise from large data and complex models. Distributed computing offers a robust approach to processing very large datasets through parallel processing on multiple computers. Sampling techniques provide effective approaches to working with limited memory resources by allowing analysis using a subset of the overall data. The use of minibatch gradients is essential in machine learning, especially in deep learning, to enable efficient and scalable training. However, there are still many issues to be resolved. These challenges include enhancing statistical efficiency in practical settings, minimizing communication and computation costs, and designing adaptive techniques capable of achieving a balance between accuracy and efficiency. There is a need for more integration of these approaches and development of new techniques that cater to the needs of emerging data-intensive applications.

In summary, the development of advanced methods of statistical computation will continue to be crucial for harnessing the power of big data in both research and applications. In this research we reviewed and compared the three main strategies for large-scale statistical computation: distributed computing, subsampling and minibatch-based optimization strategies. The results show that different methods have their own strengths and weaknesses in dealing with large data, depending on the computational needs and constraints. Distributed computing is a powerful approach for very large datasets that are beyond the memory capacity of a single computer, and exhibits scalability with multiple processors. But it comes with increased computational and communication costs, making it less suitable for medium-sized data. Sampling is an efficient approach to reducing computational costs, especially in memory-constrained environments. While they greatly reduce computational and communication overhead, they are not as effective in terms of accuracy and scalability, as some information is lost from the reduced data representation. In contrast, minibatch-based optimization techniques perform better than the other techniques because they offer an effective trade-off between efficiency, accuracy and scalability. Their ability to efficiently update the model parameters makes them ideal for modern machine learning and deep learning applications. In all, the findings demonstrate that no approach is universally best; it is necessary to consider the characteristics of the data, the level of computational resources available, and the trade-off between efficiency and accuracy in order to decide which method is most suitable. This research thus offers a better understanding of the performance of these techniques under similar circumstances and also provides some guidance for the choice of computational techniques in large data analysis [7-26].

Author Contributions

Nadia Naqvi conceived the study, conducted the literature review, performed the statistical analysis, interpreted the results, and wrote the manuscript. The authors reviewed and approved the final version of the manuscript.

Data Availability Statement

The data used in this study were obtained from a publicly available dataset hosted on Kaggle. The dataset can be accessed at the following URL: [https://www.kaggle.com/code/nadianaqvi/notebookf8509c847a/edit]. The data were used for research and statistical analysis in accordance with the terms of access and use specified by the dataset provider.

References

1. Fan, J., & Lv, J. (2008). Sure, independence screening for ultrahigh dimensional feature space. *Journal of the Royal Statistical Society Series B: Statistical Methodology, 70*(5), 849-911.
2. Dean, J., & Ghemawat, S. (2004, December). Mapreduce: simplified data processing on large clusters. *In osdi* (Vol. 4, No. 1, pp. 137-150).
3. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., ... & Stoica, I. (2012). Resilient distributed datasets: A {Fault-Tolerant} abstraction for {In-Memory} cluster computing. In 9th USENIX symposium on networked systems design and implementation (NSDI 12) (pp. 15-28).
4. Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1, No. 2, pp. 1-800). *Cambridge: MIT press.*
5. Bottou, L., Curtis, F. E., & Nocedal, J. (2018). Optimization methods for large-scale machine learning. *SIAM review, 60*(2), 223-311.
6. Koenker, R., & Bassett Jr, G. (1978). Regression quantiles. Econometrica: *journal of the Econometric Society,* 33-50.
7. Boucheron, S., Lugosi, G., & Massart, P. (2013). Concentration inequalities: A nonasymptotic theory of independence. *Oxford University Press.*
8. Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM, 51*(1), 107-113.
9. Drineas, P., Mahoney, M. W., Muthukrishnan, S., & Sarlós, T. (2011). Faster least squares approximation. *Numerische mathematik, 117*(2), 219-249.
10. Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research, 12*(7).
11. Friedman, J., Hastie, T., & Tibshirani, R. (2001). The elements

of statistical learning. *Springer*.
12. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning.*
13. Hastie, T., Tibshirani, R., & Wainwright, M. (2015). Statistical learning with sparsity. *Monographs on statistics and applied probability, 143*(143), 8.
14. Jordan, M. I. (2013). On statistics, computation and scalability.
15. Kingma, D. P., & Ba, J. (2015). *Adam: A method for stochastic optimization.*
16. Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., & Su, B.-Y. (2014). Scaling distributed machine learning with the parameter server. *OSDI.*
17. Li, M., Andersen, D. G., Smola, A., & Yu, K. (2014). Communication efficient distributed machine learning with the parameter server. *Advances in neural information processing systems, 27.*
18. Ma, P., Mahoney, M. W., & Yu, B. (2015). A statistical perspective on algorithmic leveraging. *Journal of Machine Learning Research, 16,* 861–911.
19. McMahan, B., Moore, E., Ramage, D., Hampson, S., & y Arcas, B. A. (2017, April). *Communication-efficient learning of deep networks from decentralized data.* In Artificial intelligence and statistics (pp. 1273-1282). Pmlr.
20. Nocedal, J., & Wright, S. (2006). *Numerical optimization. Springer.*
21. Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics, 4*(5), 1-17.
22. Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics,* 400-407.
23. Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747.*
24. Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. *Cambridge university press.*
25. Wainwright, M. J. (2019). High-dimensional statistics: A non-asymptotic viewpoint (Vol. 48). *Cambridge university press.*
26. Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., ... & Stoica, I. (2016). Apache spark: a unified engine for big data processing. *Communications of the ACM, 59*(11), 56-65.

Copyright: *©2026 Nadia Naqvi, et al. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.*